MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
"IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Oleshchenko Liubov

Big Data Technologies

Lecture Notes

Approved by the Academic Council of Igor Sikorsky Kyiv Polytechnic Institute
as lecture notes for students studying in the specialty F2 “Software Engineering”

Electronic online educational publication

KYIV
IGOR SIKORSKY KYIV POLYTECHNIC INSTITUTE
2025

Reviewer: Poltorak Vadym, PhD, Associate Professor

Editor -in-chief: Legeza Viktor, Doctor of Technical Sciences, Professor

The approval was granted by the Methodological Council of Igor Sikorsky Kyiv Polytechnic Institute (minutes of meeting № 1 dated October 2, 2025)
upon the submission of the Academic Council of Faculty of Applied Mathematics
(minutes of meeting № 14 dated June 30, 2025)
Electronic online educational publication

Oleshchenko Liubov Mykhailivna, PhD, Associate Professor

BIG DATA TECHNOLOGIES

LECTURE NOTES

Big Data Technologies: Lecture Notes [Electronic resource]: tutorial is aimed at students of the speciality F2 “Software Engineering” (educational program «Software Engineering of Multimedia and Information Retrieval Systems») / Igor Sikorsky Kyiv Polytechnic Institute; Oleshchenko Liubov M. – Electronic text data. – Kyiv: Igor Sikorsky Kyiv Polytechnic Institute, 2025. – 225 p.

The tutorial is developed for acquaint students with theoretical information of programming technologies for big data processing, big data architectural models, virtualization technologies, container technologies for executing program code on the server. The purpose of the lecture notes is to provide knowledge of theoretical and practical basics of big data processing using Python and R programming languages, and also distributed data processing technologies. The tutorial includes the introduction and 18 sections devoted to a certain lecture. There are a plan, theoretical information, questions for self-assessment and a list of recommended literature are given as well. The textbook is intended for students majoring in F2 "Software Engineering" within the educational program "Software Engineering of Multimedia and Information Retrieval Systems" at the Faculty of Applied Mathematics, Igor Sikorsky Kyiv Polytechnic Institute.

© L.M. Oleshchenko, 2025

© Igor Sikorsky Kyiv Polytechnic Institute, 2025

CONTENTS

Introduction 8
Lecture 1. Sources of Big Data. Internet of Things. Definition of Big Data 9
1.1. Internet of Things and the growth of data 9
1.2 Kaggle 9
1.3. DrivenData 12
1.4. Definition of Big Data 13
1.5. Real-world examples of big data 14
1.6. Open data 15
1.7. Data privacy 16
1.8. Structured and unstructured data 16
1.9. Cloud and fog computing 17
1.10. Data at rest and data in motion 19
1.11. Big Data infrastructure 19
1.12. Distributed data and its processing 22
Conclusion to lecture 1 26
Questions for reinforcement 27
List of recommended literature 27
Lecture 2. Developing software for analyzing websites that provide open data using Python Pandas. Open data, its formats and processing tools 28
2.1. Capabilities of data analysis tools 28
2.2. The role of Python in data analysis 29
2.3. Traditional big data analytics and next-generation analytics 29
2.4. Data analysis lifecycle 31
2.5. Open data, its formats and processing tools 33
2.6. Web scraping 34
2.7. Extracting, transforming, and loading data 35
Conclusion to lecture 2 38
Questions for reinforcement 39
List of recommended literature 39
Lecture 3. Formatting time and date data, reading and writing files in Python. Interaction with external applications 40
3.1. Formatting time and date data in Python 40
3.2. Reading and writing files in Python 42

3.3. Interaction with external applications 43
Conclusion to lecture 3 46
Questions for reinforcement..46
List of recommended literature...46
Lecture 4. Python and SQLite Programming. Purpose of the csvsql utility 47
4.1. Basic SQL operations 47
4.2. Working with SQLite in Python 49
4.3. Purpose of the csvsql utility 50
Conclusion to lecture 4 52
Questions for reinforcement..53
List of recommended literature...53
Lecture 5. Procedure for importing data from files into Pandas. Importing data from the Internet. Tools for correlation analysis in Pandas 54
5.1. Statistical approaches to big data analytics 54
5.2. Using Pandas 58
5.3. Importing data from files 59
5.4. Importing data from the Internet.. 60
5.5. Descriptive statistics in Pandas 61
5.6. Tools for correlation analysis in Pandas 61
Conclusion to lecture 5 63
Questions for reinforcement..64
List of recommended literature...64
Lecture 6. Handling missing data. Converting data types and manipulating date frames in Python 65
6.1. Handling missing data 65
6.2. Data type conversion 66
6.3. Manipulating date frames 68
Conclusion to lecture 6 71
Questions for reinforcement..71
List of recommended literature...71
Lecture 7. Data regression analysis in Python 72
7.1. Machine learning analysis methods and types 72
7.2. Regression analysis 74
7.3. Types of regression analysis 75
7.4. Application of regression analysis 81

Conclusion to lecture 7 82
Questions for reinforcement.......... 82
List of recommended literature.......... 82
Lecture 8. Errors in data analysis and predictive analytics. Estimating regression errors using Python. Purpose of the scikit-learn library 83
8.1 Errors in data analysis and predictive analytics 83
8.2. Estimating regression errors using Python 87
8.3. Purpose of the scikit-learn library 92
Conclusion to lecture 8 93
Questions for reinforcement.......... 93
List of recommended literature.......... 93
Lecture 9. Data classification algorithms. Applications and problems of classifications. Decision tree classifier model 94
9.1. Classification problems 94
9.2. Classification algorithms 95
9.3 . Visualization of classifications 97
9.4. Application and validation of classifications 99
Conclusion to lecture 9 102
Questions for reinforcement.......... 102
List of recommended literature.......... 102
Lecture 10. Pyplot Module. Plotly Tool. Types of data visualization. Anomaly visualization. Using Folium and Leaflet.js libraries to build maps 103
10.1. Pyplot module 103
10.2. Plotly tool 107
10.3. Types of data visualization 112
10.4. Visualization of anomalies 116
10.5. Using Folium and Leaflet.js libraries to build maps 118
Conclusion to lecture 10 120
Questions for reinforcement.......... 120
List of recommended literature.......... 120
Lecture 11. Data analysis in R. Factors, lists, frames.......... 121
11.1. History of R language development 121
11.2. R language features.......... 122
11.3. Objects, packages, functions 122
11.4. Vectors, matrices, and operations in R 124

11.5. Factors, lists, frames 131
Conclusion to lecture 11 142
Questions for reinforcement....143
List of recommended literature....143
Lecture 12. Export, import and data processing in R 144
12.1. Exporting and importing data into R 144
12.2. Using R for time series analysis 148
12.3. Data processing in R 150
Conclusion to lecture 12 157
Questions for reinforcement....158
List of recommended literature....158
Lecture 13. Basic tools for data analysis and visualization in R 159
13.1. The plot() function and its parameters 159
13.2. Managing common parameters - arguments of graphical functions 161
13.3. Types of graphs in R 164
Conclusion to the lecture 13 168
Questions for reinforcement....169
List of recommended literature....169
Lecture 14. Architectural models of Big Data. Virtualization technologies. Hypervisors. Container technology for executing program code on the server. 170
14.1. Architectural models of Big Data engineering 170
14.2. Data centers and cloud computing 172
14.3. Virtualization technologies 173
14.4. Layers of abstraction 175
14. 5. Hypervisors 176
14.6. Container technology for executing program code on the server 177
14.7. Data engineering 179
Conclusion to lecture 14 182
Questions for reinforcement....182
List of recommended literature....182
Lecture 15. Hadoop Big Data technologies. Distributed MapReduce processing....183
15.1. Scalability through Big Data 183
15.2. Data storage and processing in distributed file systems 184
15.3. Distributed databases 184
15.4. Hadoop distributed file system (HDFS) 185

15.5. MapReduce 187
Conclusion to lecture 15 188
Questions for reinforcement...189
List of recommended literature...189
Lecture 16. Kafka distributed streaming platform. Cassandra advantages 190
16.1. Data reception problem 190
16.2. Kafka distributed streaming platform 191
16.3. Cassandra advantages 193
Conclusion to lecture 16 200
Questions for reinforcement...200
List of recommended literature...200
Lecture 17. Apache Spark platform 201
17.1. The problem of the computable function..201
17.2. Spark technology 202
17.3. Comparison of Spark and MapReduce 204
17.4. Spark and sparklyr for working with big data in R 205
Conclusion to the lecture 17 217
Questions for reinforcement...217
List of recommended literature...217
Lecture 18. Lambda and Kappa architectures for big data processing 218
18.1. Lambda - architecture 218
18.2. Advantages and disadvantages of Lambda architecture 221
18.3. Kappa – architecture 222
18.4. Advantages and disadvantages of Kappa architecture 223
Conclusion to the lecture 18...224
Questions for reinforcement...225
List of recommended literature...225

INTRODUCTION

Big data analytics is one of the most in-demand skills in modern business today. Knowledge of new programming technologies and the ability to develop software for managing and analyzing large amounts of information are used to ensure the digitalization of all areas of life.

The discipline "Big Data Technologies" is part of the cycle of professionally oriented disciplines of the bachelor's training plan for students studying in specialty F2 "Software Engineering".

The subject of the discipline is the theoretical and practical foundations of big data processing. General methods of big data processing, the capabilities of the Python and R programming languages for data analysis and visualization are considered.

The goal of the discipline is to provide knowledge of the theoretical and practical foundations of big data processing using the Python and R programming languages and distributed data processing technologies.

The purpose of the lecture notes is to obtain the necessary level of knowledge of programming technologies for processing big data, architectural models of Big Data, virtualization technologies, container technologies for executing program code on the server. In this lecture notes, students will get acquainted with the main data sources and technologies for processing big data using Python and R tools.

The course of lectures on the discipline "Big Data Technologies" is designed for 36 academic hours of classroom lessons. The lecture notes consist of 18 sections, each of which is devoted to one lecture on the discipline "Big Data Technologies". Each section provides theoretical information on each topic, self-test questions, and a list of recommended literature.

Lecture 1.
Sources of Big Data. Internet of Things.
Definition of Big Data

Lecture plan

1.1. *Internet of Things and the growth of data.*
1.2. *Kaggle platform.*
1.3. *DrivenData.*
1.4. *Definition of Big Data.*
1.5. *Real-world examples of big data.*
1.6. *Open data.*
1.7. *Data privacy.*
1.8. *Structured and unstructured data.*
1.9. *Cloud and fog computing.*
1.10. *Data at rest and data in motion.*
1.11. *Big data infrastructure.*
1.12. *Distributed data and its processing.*

1.1. Internet of Things and the growth of data

Our world is very complex. This complexity is generating an ever – increasing amount of data that needs to be stored and analyzed. The rate of data generation shows no signs of slowing down. The diversity of data is expanding into new areas that were never before available for analysis. Interaction between people using media platforms, automation of processes, and aggregation of data from different sources are creating the Internet of Things.

Internet of Things (IoT) not only attaches sensors to existing things, it creates a market for new connected things. All of these connected things generate data. This adds up to an unimaginable amount of data, called *Big Data*.

It is not always possible to collect all the available data for a project or solution. The amount of data that can be collected is determined by the capabilities of the sensors, network, computers, and other equipment. It is also determined by the need, for example, to check the correct alignment of the label of each bottle moving along a high-speed beverage bottling line. In this case, the data of each bottle is important. For another sensor, such as a moisture sensor in a cornfield, it is not necessary to report the measured moisture value every tenth second. Every five to ten minutes may

be sufficient. To make this data useful, it must be cleaned. Cleaning is the process of removing unwanted data, changing incorrect data, and filling in missing data. It is common to use program code to clean data. This is achieved by searching for criteria or their absence and manipulating the data until there are no more anomalies. Once the data is cleaned, it can be more easily searched, analyzed, and visualized. Data analysis can uncover interesting insights and trends. This often leads to new queries that have not yet been implemented. If we can't find additional value from a certain set of data, we can experiment with how it's organized and presented. For example, a security camera that monitors parking for crimes could also be used to inform drivers about the number and location of available spaces. If we assume that each grain is equivalent to one byte of data, then with successive doubling of grains in each subsequent cell the number of grains in the last square of a chessboard would be equivalent to nine exabytes. One exabyte is approximately 1.07 billion gigabytes. Nine exabytes is roughly equivalent to the amount of Internet traffic in 2014 [1].

1.2. Kaggle platform

Innovation allows companies to never stop developing. Today, more and more organizations are placing sensors in their products. Their goal is to collect and analyze data to obtain valuable outcomes. To harness the capabilities of IoT, organizations need skilled and creative people. Online platforms such as Kaggle allow companies to find talented people from around the world.

Kaggle is a platform that connects businesses and organizations with questions about their data and people who know how to find answers to those questions (Fig. 1.1). Various organizations hold online competitions to build the world's best data models. Competitors generate many models using a variety of techniques. Players from all over the world have different backgrounds and specializations. They can join teams or simply help each other. The winner or winning team of each competition wins a prize. Usually this can be money, but sometimes it can be an invitation to work in a relevant company. In each competition, participants continually improve as each

winner beats the previous score. New predictive data models consistently outperform existing best models. Mayo Clinic, NASA, GE, and Deloitte are just a few of the businesses and organizations that have hosted Kaggle competitions [3].

kaggle Compete Datasets Notebooks Communities Courses ••• Search Sign In

Start with more than a blinking cursor

Kaggle offers a no-setup, customizable, Jupyter Notebooks environment. Access free GPUs and a huge repository of community published data & code.

REGISTER WITH GOOGLE

Register with Email

Predict Malicious Websites: XGBoost Draft saved

File Edit Insert Run View Help

+ Add dataset Commit

```
data = pd.read_csv("../input/dataset.csv")

# clean up column names
data.columns = data.columns.\
    str.strip().\
    str.lower()

# remove non-numeric columns
data = data.select_dtypes(['number'])

# split data into training & testing
train, test = train_test_split(data, shuffle=True)

# peek @ dataframe
train.head()
```

```
# split training data into inputs & outputs
X = train.drop(['type'], axis=1)
Y = train['type']

# specify model (xgboost defaults are generally fine)
model = xgb.XGBRegressor(free_met)

# fit our model
model.fit(y=Y, X=X)
```

Inside Kaggle you'll find all the code & data you need to do your data science work. Use over 50,000 public datasets and 400,000 public notebooks to conquer any analysis in no time.

Machine Learning

Machine Learning is the hottest field in data science, and this track will get you started quickly

65k

Pandas

Short hands-on challenges to perfect your data manipulation skills

87k

Python

Learn the most important language for Data Science

65k

Deep Learning

Use TensorFlow to take Machine Learning to the next level. Your new skills will amaze you

12k

kaggle Competitions Datasets Models Code Discussions Courses ••• Search Sign In Register

Level up with the largest AI & ML community

Join over 23M+ machine learners to share, stress test, and stay up-to-date on all the latest ML techniques and technologies. Discover a huge repository of community-published models, data & code for your next project.

Register with Google Register with Email

Fig. 1.1. Kaggle platform [3]

1.3. DrivenData

Technologies used in IoT and data analytics can be used to solve various social problems. The collected data can be used to predict various trends. For example, using available data, entrepreneurs in a country can predict which water pumps are functioning and which need repair or are not working. Thanks to prediction, the operation of the water service devices is more efficient. Competitions that are performed for various social projects, for example, can be found on the website and DrivenData [4]. Mission is to use best practices in data analysis and crowdsourcing in to solve global social problems. Like Kaggle, they host online challenges, where a global community of scientists compete to build the best statistical model for complex prediction problems. These models can contribute to positive change in the world.

DrivenData starts by asking a predictive question that can be answered with existing data and has a measurable, real-world impact. They work with nonprofits to understand their needs and identify productive partnerships. Next, DrivenData hosts an online open innovation competition where software developers and data scientists submit statistical models. Using their competitive platform and scoring engine, the models are ranked based on how well they predict their competitors' data. Finally, they work with the organization to implement the best model, new statistical approach, or data analysis tool. This enables the nonprofit to more effectively and sustainably fulfill its mission (Fig. 1.2).

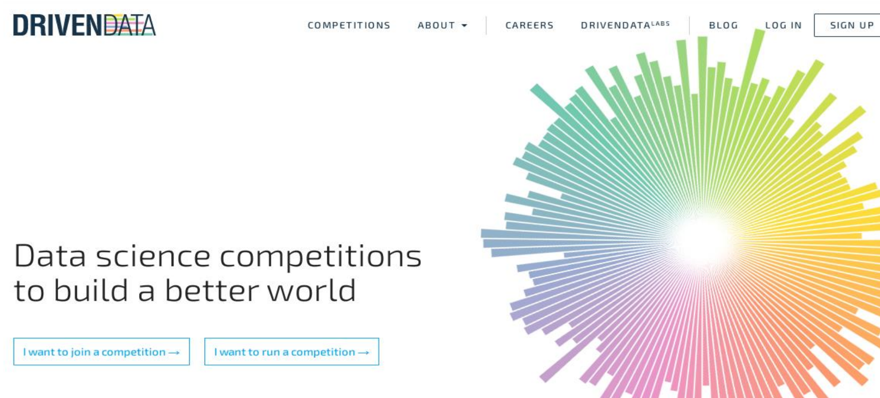


Fig. 1.2. DrivenData platform [4]

1.4. Definition of Big Data

The exponential growth of data has created a new area of interest in technology and business called "Big Data." In general, a data set or business problem falls into the Big Data classification when its data is so large or complex that it becomes impossible to store, process, and analyze using traditional approaches to data storage and analysis.

How much data does it take to become Big Data? Is 100 terabytes or 1,000 petabytes enough? Volume is only one criterion, as the need to process data in real time (also called data in motion) or the need to integrate structured and unstructured data can qualify a problem as a big data problem.

For example, the International Data Corporation (IDC) uses 100 terabytes as the size of a dataset that is defined as Big Data. If the data is streaming, the size of the dataset can be smaller than 100 terabytes but is still considered Big Data as long as the data being created is growing at more than 60% per year.

According to the National Institute of Standards and Technology (NIST): "The big data paradigm consists of distributing data systems across horizontally linked independent resources to achieve the scalability required to efficiently process large data sets." To distinguish data from big data, four Vs are used next (Fig. 1.3).

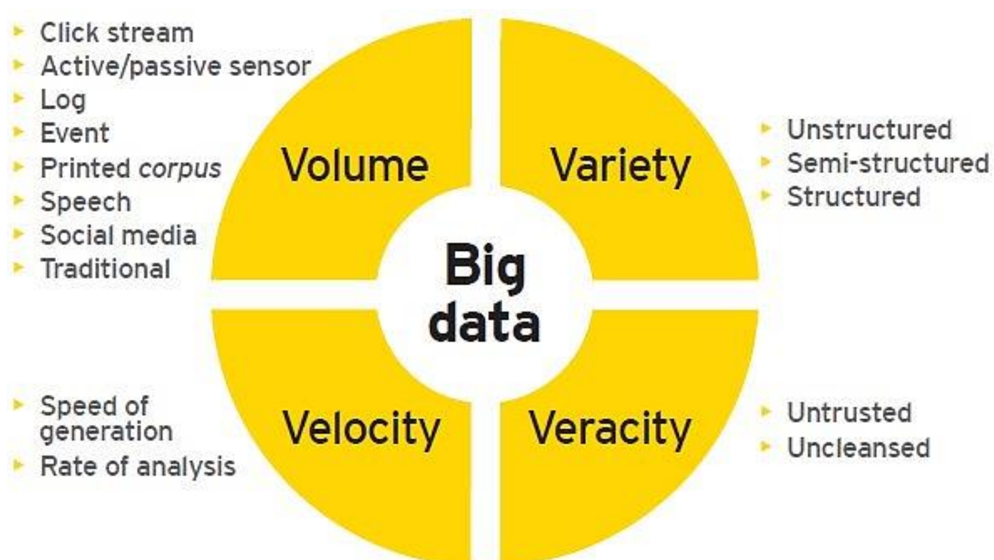


Fig. 1.3. Characteristics of Big Data [5]

Volume - the amount of data being transmitted and stored. The current challenge is to find ways to most efficiently process the growing volume of data.

Velocity - the speed at which data is generated. For example, the data generated by a billion shares traded on the New York Stock Exchange cannot simply be stored for further analysis.

Variety is a type of data that is rarely in a state that is perfectly ready for processing and analysis. Big Data is unstructured data, which is estimated to make up 70 to 90% of the world's data.

Veracity is the process of preventing inaccurate descriptions of data sets. For example, people may create an online account and use false contact information. Increased veracity in data collection reduces the amount of data cleaning required.

Although the four Vs are listed here, most discussions, tools, and documents only address the first three (volume, velocity, variety).

1.5. Real-world examples of big data

Let’s look at some real-world examples of big data generators. An Airbus A380 Engine generates 1 petabyte of data during a flight from London to Singapore. Large Hadron Collider (LHC) generates 1 gigabyte of data every second (Fig. 1.4). The Square Kilometre Array (SKA) is the largest radio telescope in the world. It generates 20 exabytes of data per day. This is equivalent to 20 billion gigabytes per day [6].

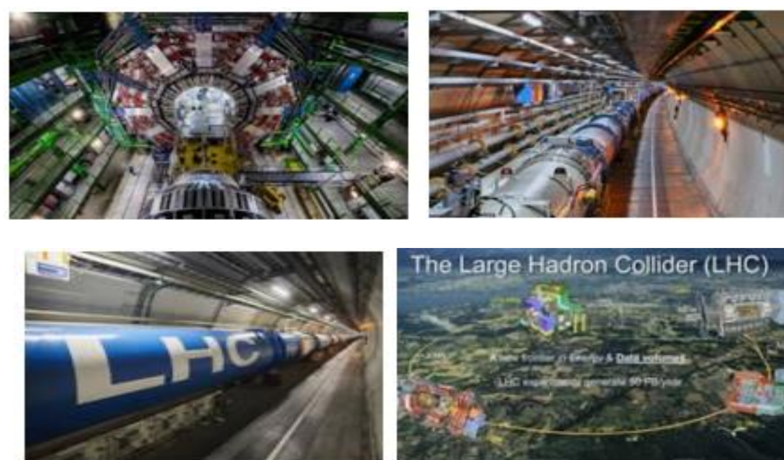


Fig. 1.4. Large Hadron Collider

Internet of Things uses sensors to generate data. Data can come from temperature and humidity sensors found in agriculture. Sensors are now in everything from smartphones to cars and jet engines to home appliances. The list of things with sensors is growing every year, which also contributes to the exponential growth of Big Data.

1.6. Open data

With the increasing importance of data to businesses and individuals, many questions arise about privacy and the availability of large repositories of public and private data. It is important for analysts to understand the continuum between open and private data. Making decisions about how different types of data will be used in an organization is as important as knowing how to implement distributed storage and processing of Big Data.

"Open Knowledge Foundation " [7] defines open knowledge as “any content, information, or data that people can use and redistribute without any legal, technological, or social constraints.” Open data is a component of open knowledge. Open knowledge is when open data is made useful and used.

The value of open data can be seen immediately by browsing sites like the New York City Open Data Portal, Open Data NYC, where a visitor can quickly find restaurant ratings based on annual inspections by the Department of Health and Mental Hygiene. The portal is a repository of over 1,300 datasets from city agencies to promote government transparency and civic engagement.

Dataset is a collection of related discrete records that can be accessed for management individually or as a whole.

Gapminder – non-profit enterprise that promotes sustainable global development [8] The site provides analytics on open datasets with statistical refinements on topics such as:

- from the health and wealth of nations;
- CO2 emissions since 1820;
- infant mortality;
- HIV infection.

The Open Data Portal of Ukraine contains datasets by groups Construction, State, Ecology, Economy and Business, Land, Youth and Sports, Education and Culture, Healthcare, Taxes, Agriculture, Social Protection, Standards, Transport, Finance, Justice in formats such as csv, xls, JSON.

1.7. Data privacy

As new software applications are developed, more and more data is required from the end user to give companies and advertisers more information to make business decisions. Using SafeAnswers, openPDS only provides answers to specific queries, and raw data is not sent. The calculation for the answer is done in the user's personal data store (PDS): "Only the answers, the aggregated data needed by the application, leave the user's PDS (for example, exporting GPS data for the application to find out if the user is active or to find out about the general geographical area, the calculation can be done in the user's PDS of the corresponding Q & A module). The privacy of user data was first discussed by specialists in the 1990s, and today the idea is being promoted that the future of privacy cannot be ensured solely by compliance with legislation and regulations; rather, ensuring privacy should become the default mode of operation of the organization.

1.8. Structured and unstructured data

Previously, we classified data as open or private in terms of its availability. Data can also be classified by the way it is organized, as structured or unstructured.

Structured data is entered and maintained in fixed fields in a file or record. We can easily enter, classify, query, and analyze structured data with a computer. This includes data found in relational databases and spreadsheets.

If the data set is small enough, structured data is handled using Structured Query Language (SQL), a programming language designed to query data in relational databases. SQL only works on structured data sets. For Big Data, structured data can be part of the data set, but Big Data tools are independent of that structure. Big Data typically has data sets that are composed of unstructured data.

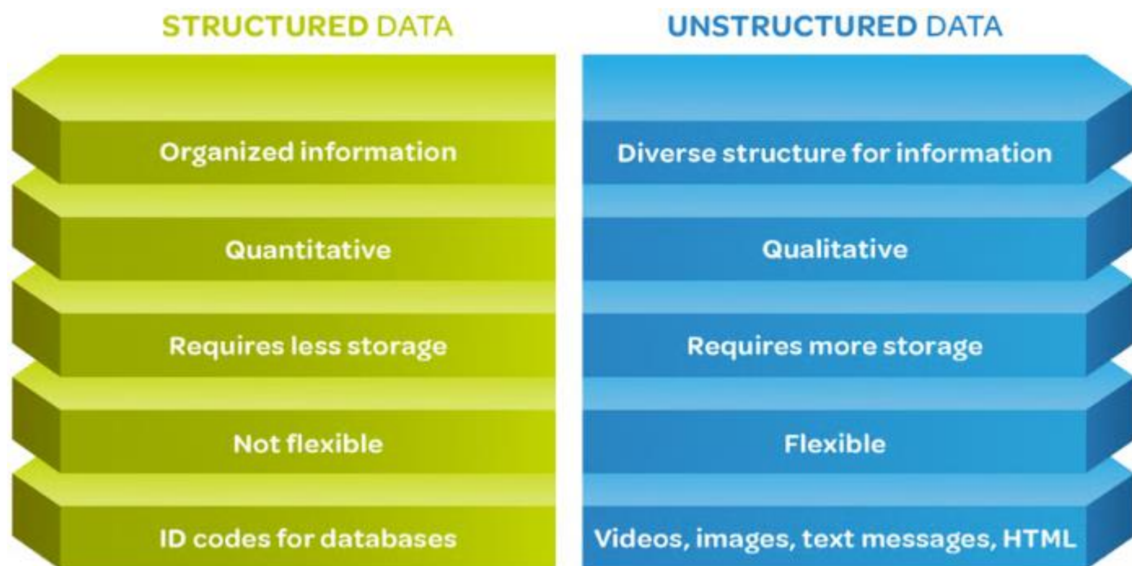


Fig. 1.5. Structured and unstructured data comparison [2]

Unstructured data is raw data that is not pre-organized and does not have a fixed schema that identifies the type of data. Unstructured data lacks a set way of entering or grouping data and analyzing it. Examples of unstructured data include the content of photos, audio, video, web pages, blogs, books, magazines, posts, PowerPoint presentations, articles, emails, wiki files, plain text documents, and text in general. An example of unstructured data is a PDF version of a book. The text is searchable, but it is not organized in a predefined form, such as using fields and records.

Both structured and unstructured data are valuable to people, organizations, industries, and governments. It is important for organizations to embrace all forms of data and determine how to format it so that it can be managed and analyzed.

1.9. Cloud and fog computing

In the past, data sets were largely static, residing on a single server or a collection of servers within an organization, and processed using a database programming language such as SQL. While this model still exists, the storage of large data sets has moved to data centers. Today, with the rise of cloud computing, the role of Big Data, and the need for real-time data analysis, data continues to reside in DPCs. Data also needs to be available for analysis closer to where it was created,

and the insights gained from that data can have the greatest impact. This way of processing data is called *fog computing*. A fog is a cloud that is located close to the source of data generation. Fog computing is not a replacement for cloud computing; rather, fog computing enables the development of new tools.

In the fog computing model, there is a relationship between the cloud and the fog, especially when it comes to data management and analytics. Fog computing provides compute, storage, and network services between end devices and traditional data centers. Fog computing generates a huge amount of data from various sensors and controllers. When working with data on the Internet, next three important factors must be considered. The amount of energy used by an IoT sensor and depends on the sensor's sampling rate. The range between devices can also affect the amount of energy that must be used to transmit sensor data to controllers. The further away the sensor is, the more energy must be used to transmit data.

When many sensors are transmitting data, communication latency can occur if there is not enough bandwidth to support all the devices. Additional analysis in the fog can help reduce some of the bandwidth requirements.

Real-time data analysis is affected by too much latency on the network. It is very important that only necessary communications with the cloud are performed, and that the computation takes place as close to the data source as possible.

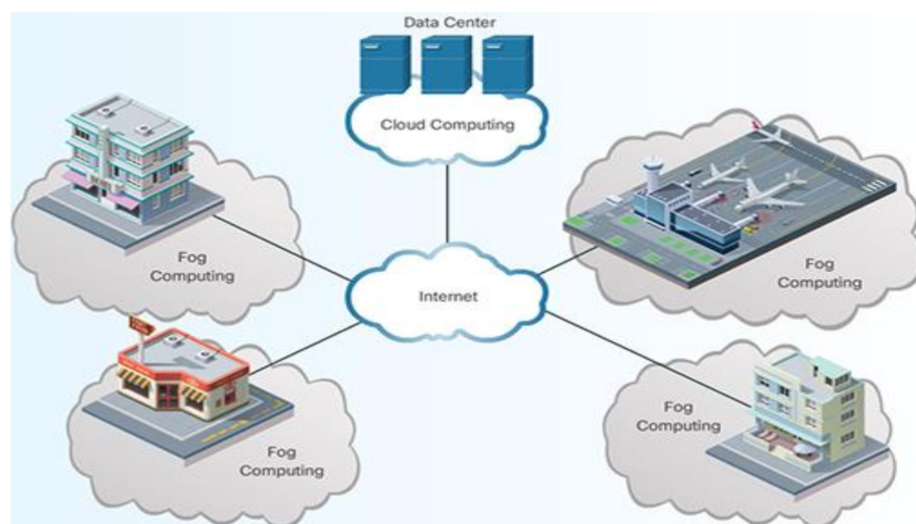


Fig. 1.6. The model of fog computing [2]

1.10. Data at rest and data in motion

Data at rest is static data that is stored in a physical location, such as a hard drive on a server or in a data center. The data is stored in a database and then analyzed and interpreted. Decision makers are notified whether action is needed.

Data in motion is dynamic data that requires real-time processing before it becomes irrelevant or outdated. It is a continuous interaction between people, processes, and things. Devices at the edge of the network work together to immediately act on the insights gained from dynamic data analysis. The order of analysis, action, and notification can vary. An important distinction between data at rest and data in motion is that with data in motion, the data is acted upon before the data is stored. Data in motion is used by various industries that rely on extracting values from the data before storing it. Sensors in a farmer's field continuously send data about temperature, soil moisture, and sunlight to a local controller, which analyzes the data. If conditions are not right, the controller takes immediate action by sending signals to actuators in the field to begin watering. The controller then sends a message to the field owner to start watering and sends the data for storage.

Due to the nature of Big Data, it is impossible to duplicate and store all this data in a centralized data warehouse. New device implementations include a large number of sensors that capture and process data. Decisions and actions need to be made at the edge, where and when the data is created. As sensors gain more processing power and become more context-aware, it is now possible to bring intelligence and analytics closer to the data source. In this case, data in motion stays where it is created and provides real-time insights, driving better, faster decisions.

1.11. Big Data infrastructure

Many companies understand that it makes sense to invest in Big Data technologies to remain competitive in their market. Currently, their data infrastructure may look something like Fig. 1.7, with database servers and traditional data processing tools. Typically, access to the data is limited to a few responsible individuals within the organization. Companies are rapidly moving to use Big Data

technologies to drive business intelligence. According to the National Institute of Standards and Technology (NIST), the Big Data paradigm consists of distributing data systems across horizontally connected, independent resources to achieve the scalability needed to efficiently process large data sets. This is horizontal scalability. It differs from vertical scalability in that it does not attempt to add more processing power and memory to existing machines. These infrastructures allow many users to access data simultaneously, seamlessly and securely. One such example is thousands of online shoppers or mobile gamers.

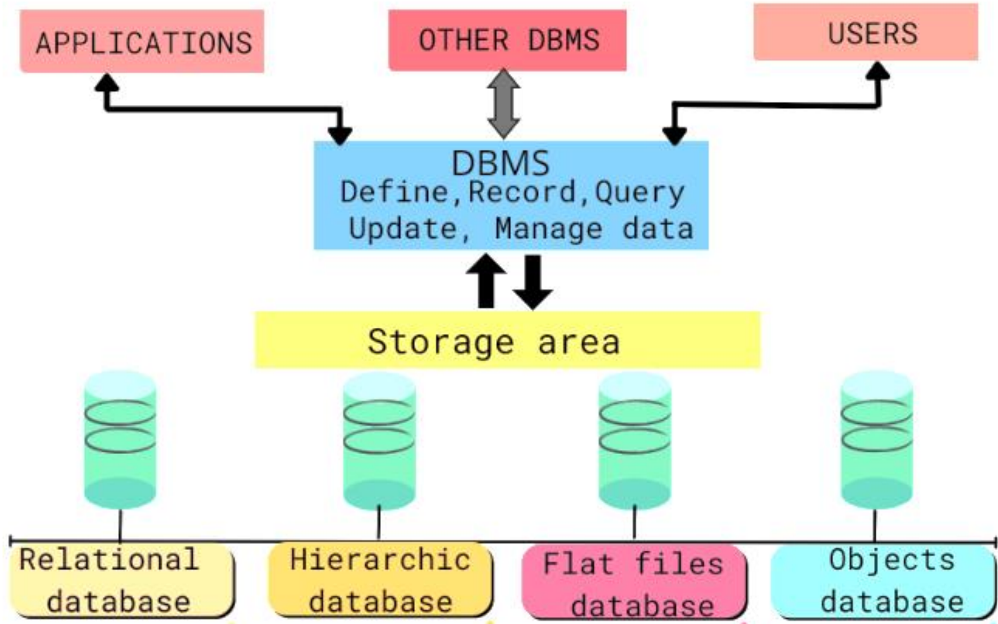


Fig. 1.7. Traditional database management system [2]

This diagram shows the structure and interaction of a Database Management System (DBMS). Applications, users, and other DBMSs communicate with the DBMS, which is responsible for defining, recording, querying, updating, and managing data. The DBMS interfaces with a storage area where different types of databases – relational, hierarchical, flat file, and object-oriented – are maintained. This setup highlights how a DBMS centralizes data management across multiple storage formats and user interactions.

Fig. 1.8 shows the devices in an organization's big data infrastructure where the business infrastructure needs on-site decision- making using cloud computing.

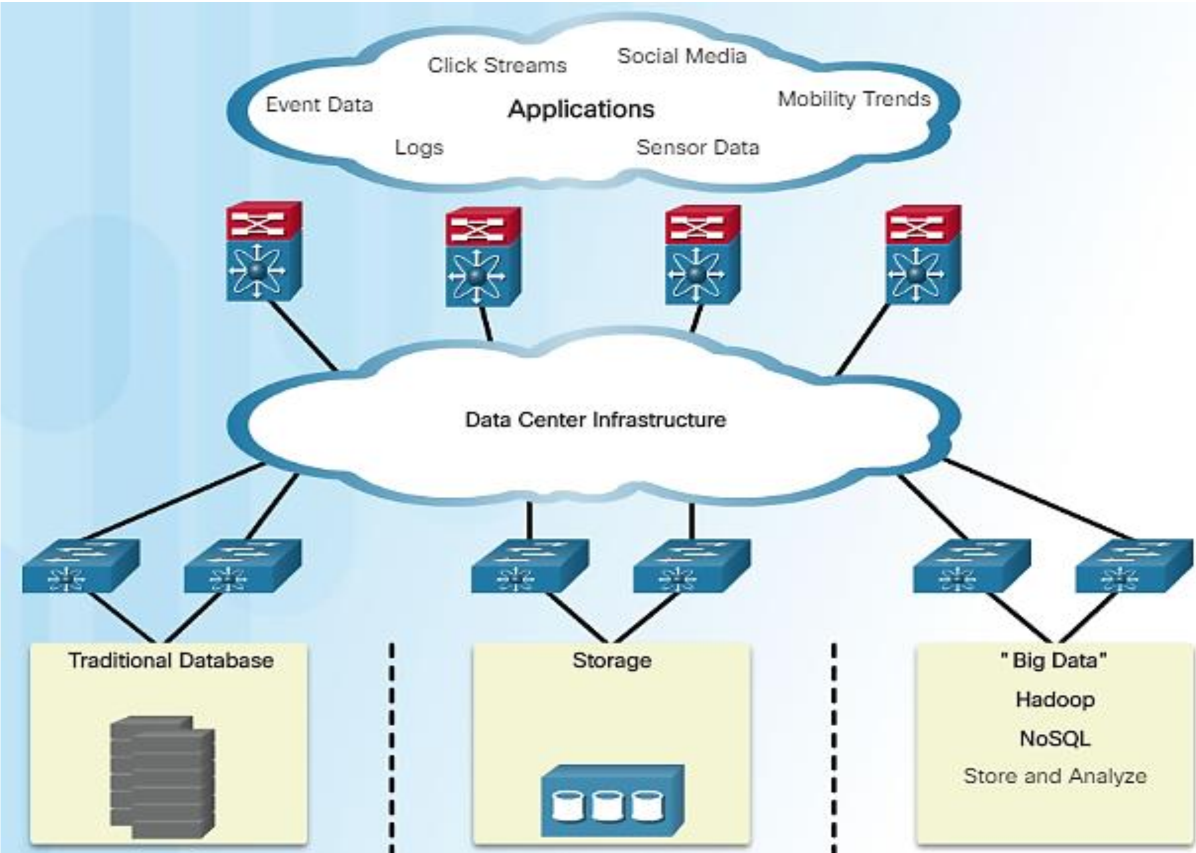


Fig. 1.8. Big data infrastructure [2]

This diagram illustrates a high-level architecture for data management in a modern data center environment. Applications generate various types of data such as click streams, social media updates, event logs, sensor outputs, and mobility trends which flow into a centralized Data Center Infrastructure.

From there, the data is distributed to different backend systems based on its nature and usage: traditional databases for structured data, general storage solutions for miscellaneous data, and specialized big data platforms like Hadoop and NoSQL for large-scale storage and analysis. The setup emphasizes scalable processing, categorization, and efficient access to diverse data types.

1.12. Distributed data and its processing

The next generation of data management emerged with the help of the relational database management system (RDBMS). For 30 years, this has been the standard approach to data management. Relational databases capture relationships between different sets of data, creating more useful information.

Most commercial RDBMS solutions use SQL as their query language to this day. An example of an SQL query is: SELECT id, name, pFig inventory i where pFig, for example, is <20. Examples of products that use a structured query language to access data include MySQL, SQLite, MS SQL, Oracle, and IBM DB2.

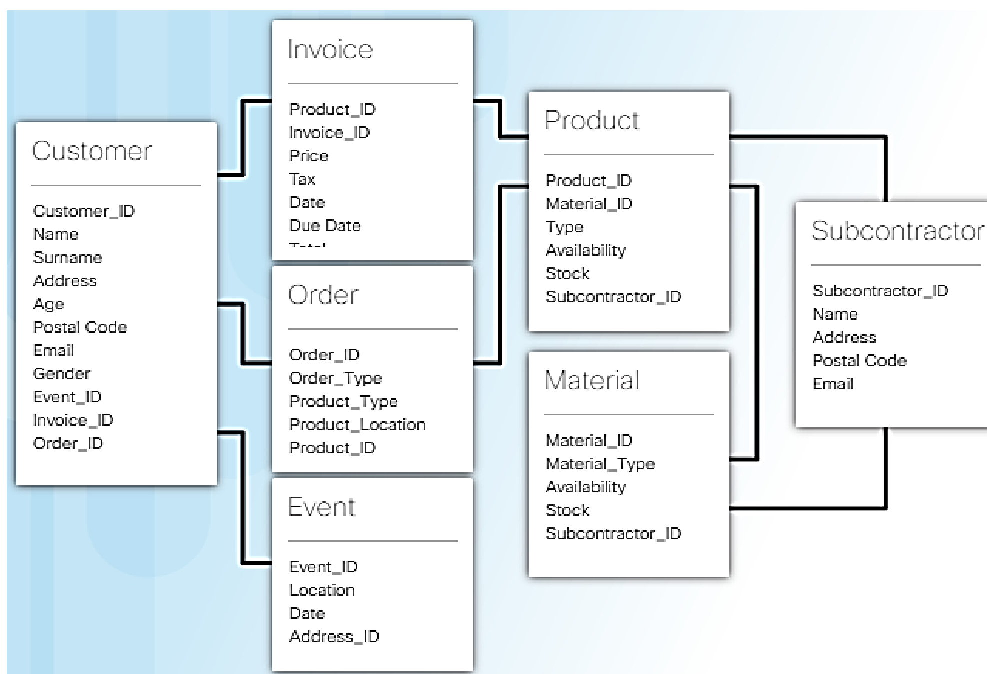


Fig. 1.9. Example of a relational database [2]

Another characteristic of relational databases is the distinction between the database and the management system used to query the database. Typically, with an RDMS and a base database, multiple users can query the relational database at the same time. The user is usually not aware of all the relationships that exist within the database, as the user is more likely to summarize the representation of the database that meets their needs.

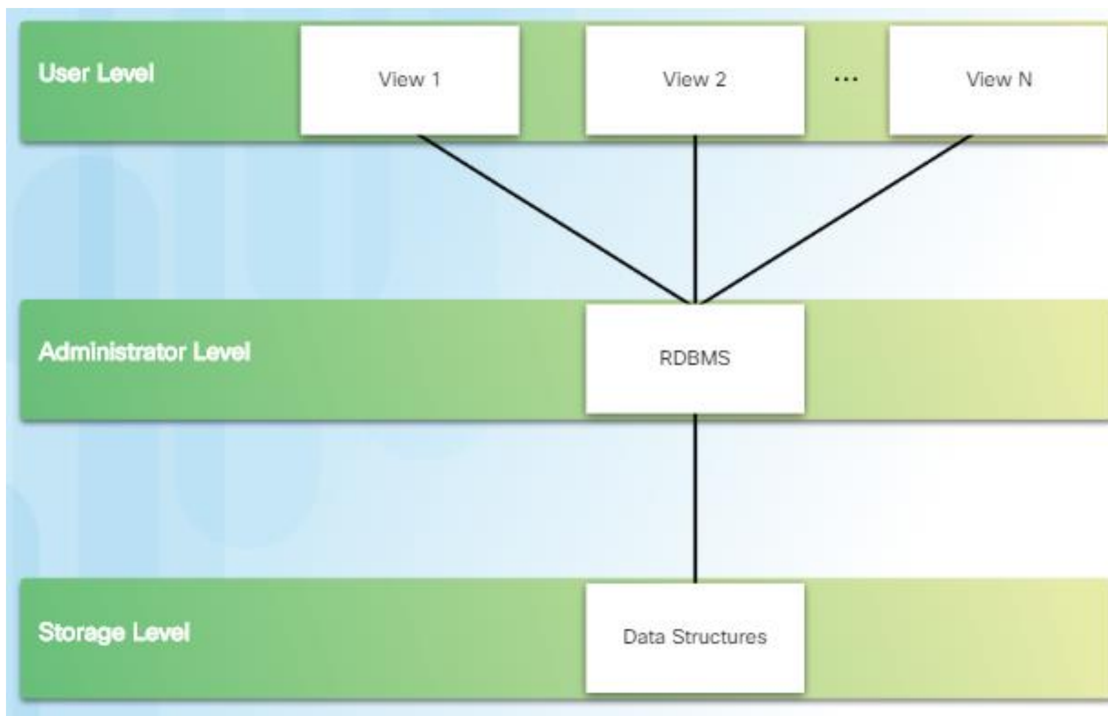


Fig. 1.10. Data abstraction in a relational database [2]

The lowest level of abstraction describes how the data is physically stored. The next level describes what data is stored and the relationships between the data. This is the level at which the database administrator works. The user level is the highest level, which describes which part of the database a particular user or group of users can access. Many different simultaneous connections to the database can be defined at any given time. Non-relational SQL (NoSQL) databases scale very well with distributed databases because NoSQL can handle big data and real-time web applications better than RDBMSs, queries to NoSQL databases are focused on collecting documents, such as information gathered from websites. NoSQL also allows clusters of machines to process data and provide better control over its availability. NoSQL databases are widely used to solve business problems.

From a data management perspective, analytics was simple when only humans created data. The volume of data was manageable. Relational databases serve the needs of data analysts. With the prevalence of business automation systems and the explosive growth of web applications and data generated, analytics is becoming more difficult to manage with just an RDBMS solution. Approximately 90% of the data

that exists today was collected in just the last two years. This increased volume in a short period of time is a property of exponential growth. This large volume of data is difficult to process and analyze. Instead of large databases being processed by large, powerful mainframe computers and stored in giant disk arrays (vertical scaling), distributed data processing takes a large amount of data and breaks it into smaller pieces. These smaller pieces of data are spread across many locations, which are processed by many computers with smaller processors. Each computer in a distributed architecture analyzes its own part of the Big Data picture (horizontal scaling).

Most distributed file systems are designed to be invisible to client applications. A distributed file system finds files and moves data, but users have no way of knowing that the files are distributed across many different servers or nodes. Users access these files as if they were local to their own computers. All users see the same view of the file system and access the data at the same time as other users.

Hadoop was created to address these large data volumes. The Hadoop project started with two strands: the Hadoop Distributed File System (HDFS), which is a distributed file system, and MapReduce, which is a distributed way of processing data. Hadoop has evolved into a comprehensive ecosystem of software for managing big data. There are many other distributed file system (DFS) applications. To name a few: Ceph, GlusterFS, and Google File System.

A NoSQL database stores and accesses data differently than relational databases. NoSQL is sometimes referred to as "not just SQL", "not SQL", or "non-relational". NoSQL systems can support SQL-like query languages. NoSQL databases use data structures such as key-value, wide column, graph, or document. Many NoSQL databases provide "successful consistency".

With succesful consistency, changes to the database are reflected across all nodes over time. This means that queries for data may not provide the latest available information. The reason for creating NoSQL was to simplify database design. It is easier to scale clusters of nodes with NoSQL than it is to do so in standard relational databases. The most popular NoSQL databases in 2015 were MongoDB, Apache Cassandra, and Redis.

Structured Query Language (SQL) is designed for managing, searching, and processing data, including Big Data. SQLite is a library that uses a self-contained, transactional SQL database engine. The code for SQLite is open source, meaning it can be freely used for commercial and private purposes. SQLite is the most widely deployed database in the world. SQLite is also an embedded SQL database system. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to regular disk files.

SQLite is a popular choice for the database engine in mobile phones, MP3 players, set-top boxes, and other electronic gadgets. SQLite has a small code footprint, allows for efficient use of memory, disk space, and bandwidth, is highly reliable, and does not require the maintenance of a database administrator. SQLite is often used instead of an RDBMS for testing. SQLite requires no configuration, which greatly simplifies testing. Let's look at some useful SQLite features.

- No setup or administration required. It has an easy-to-use API.
- The complete database is stored in a single cross-platform disk file. Can be used as a program file format.
- Has a small code footprint.
- It is a cross-platform SQL. It supports Android, iOS, Linux, Mac, Windows, and several other operating systems.
- The sources for SQLite are in the public domain.
- Has a separate command line interface (CLI).
- All changes within a single transaction are either complete or not at all. This is true even in the event of a program or operating system crash or power failure.

Using SQL and database technologies is effective for retrieving a subset of data from an existing dataset stored in a database. The SQL expression that performs this action is called an SQL query. Many important business problems cannot be solved with a simple SQL query and require a more complex analytical process. This is where a more powerful data analysis programming language such as R or Python comes into play. R and Python have large developer communities. Their users are known for developing data analysis modules and providing them to the community

for free. Because of this, any user can download and use pre-programmed modules and tools. The ability to build data analysis tools from scratch allows for custom applications. The process of building a data analysis tool from scratch can be divided into two main parts: the model and the code.

Modeling is about determining what to do with the data to achieve the desired results and conclusions. Let's say we want to create a personal fitness tracker and there is no pre-programmed module that does exactly what we want to do . The tracker module contains an accelerometer, which is a sensor that can measure the acceleration of the device. The accelerometer can be used to determine the speed and direction of movement. The speed and direction of movement of the device always correspond to the speed and direction of its user when it is attached to the user.

But what if the device is attached to a dumbbell weight or a tennis racket? The device will still receive the same data, speed and direction of movement, but through different applications the interpretation of this data must be adapted to the new use. In this context, modeling can be seen as a way to interpret and process the data.

For example, if a fitness tracker is attached to a user, two consecutive points of no movement (velocity is zero) are likely to represent the start and end of a sprint.

If they are attached to a dumbbell weight, the data is likely to represent the moment the dumbbell was lifted from the floor and the highest point the user was able to lift it before returning it to the floor. Code is the second part of building data analysis tools from scratch. Code is the program that processes data and must be written according to the model we created. Although the model and code are two separate entities, they are related because the code is built on the model.

Conclusion to lecture 1

The data can be words in a book, the contents of a spreadsheet, photos, files, or measurement streams sent by a device. The four Vs of big data are volume, velocity, variety, and validity. Structured data is data entered into fixed fields of a database file or record. Unstructured data does not have a fixed schema that defines the type of

data. Data at rest is static data stored in a physical location. Data in motion analyzes and extracts value from data before it is stored. Hadoop was built to handle large amounts of data. A NoSQL database stores and accesses data differently than a relational database.

Questions for reinforcement

1. What is the impact of the Internet of Things and the growth of data?
2. What is the Kaggle platform used for?
3. What is the definition of big data?
4. Give real-world examples of big data.
5. What data is open?
6. What data is structured and unstructured?
7. What is cloud and fog computing ?
8. Describe data at rest and data in motion.
9. What is big data infrastructure?
10. What is the role of Hadoop in distributed data processing?

List of recommended literature

1. Byte Size Infographic: Visualising data. URL: https://www.redcentricplc.com/resources/infographics/byte-size/

2. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics

3. Kaggle. URL: https://www.kaggle.com/

4. DrivenData. URL: https://www.drivendata.org/

5. Big Data: the 3 VS explained. URL: https://bigdataldn.com/intelligence/big-data-the-3-vs-explained/

6. Computing. URL: https://home.cern/science/computing

7. Open Knowledge Foundation. URL: https://okfn.org

8. Gapminder. URL: https://www.gapminder.org

Lecture 2.
Developing software for analyzing websites that provide open data using Python Pandas.
Open data, its formats and processing tools

Lecture plan

2. 1. Capabilities of data analysis tools.
2.2. The role of Python in data analysis.
2.3. Traditional big data analytics and next-generation analytics.
2.4. Data analysis life cycle.
2.5. Open data, their formats and processing tools.
2.6. Web scraping.
2.7. Data extraction, transformation and loading.

2.1. Capabilities of data analysis tools

The tool we use depends on the type of analysis we want to perform. Some tools are designed to handle manipulation and visualization of large data sets. Other tools are designed with mathematical modeling capabilities for forecasting. Regardless of which tools we use, they should be appropriate for the the following requirements.

- **Ease of use** – a tool that is easy to learn and use is often more effective than a tool that is difficult to use. Additionally, an easy-to-use tool requires less training and support.
- **Data Manipulation** – software should allow users to clean and modify data to make it more useful. This results in data being more reliable as anomalies can be detected, corrected or removed.
- **Sharing** – researchers must use the same datasets to be able to collaborate effectively and interpret the data in the same way.
- **Interactive visualization** – to fully understand how data changes over time, it’s important to visualize trends. Basic charts and graphs can’t fully represent the evolution of information the way a heat map or time-lapse view can.

2.2. The role of Python in data analysis

There are a variety of programs used to format, clean, analyze, and visualize data. Many companies and organizations are turning to open source tools to process and summarize data.

Python programming language has become a common tool for data processing.

Python was created in 1991 as an easy-to-learn programming language with many libraries used for data manipulation, machine learning, and data visualization.

By using these libraries, programmers do not have to learn multiple programming languages or spend time learning how to use different programs to perform the functions of these libraries.

Python is a flexible language that is growing and becoming increasingly integral to scientific research due to its flexibility and ease of learning.

Let's take a look at the main Python libraries for data analysis.

- **NumPy** – this library adds support for working with arrays and matFigs. It has many built-in mathematical functions for use on datasets.
- **Pandas** – this library adds support for tables and time series. It is used for data manipulation and cleaning.
- **Matplotlib** – this library adds support for data visualization. This is a library for simple and complex 3D and contour plots.

2.3. Traditional big data analytics and next-generation analytics

Before the era of big data, the role of time in data analytics was limited to how long it would take to compile a data set from different sources or how long it would take to run a data set through a certain calculation. With big data, time becomes important in other ways as well, as much of the data is gained through creating opportunities to take immediate action. Data is constantly being generated by sensors, consumers of goods and services, social media users, jet engines, the stock market, and almost anything else connected to the internet. This data is not only growing in

volume, it is also changing in real time, which also requires real-time data analysis during data collection.

When discussing Big Data and decision making Analytics-driven business solutions can improve business ROI as a function of time. Data-driven solutions have the following benefits:

- increased time for research and development of goods and services;
- increased efficiency and faster manufacturing and time to market;
- more effective marketing and advertising.

In the past, when most data sets were relatively small and manageable, analysts could use traditional tools such as Excel or a statistical program such as SPSS to extract meaningful information from that data. Typically, the data set contained historical data, and the processing of that data was not always time-dependent. The data, if not too large, could be cleaned, filtered, processed, summarized, and visualized using charts, graphs, and dashboards. As data sets increase in volume, velocity, and diversity, the complexity of storing, processing, and aggregating data becomes a challenge for traditional analytics tools. Large data sets can be distributed and processed across multiple geographically dispersed physical devices as well as in the cloud. These large data sets require big data tools such as Hadoop and Apache Spark to enable real-time analysis and predictive modeling.

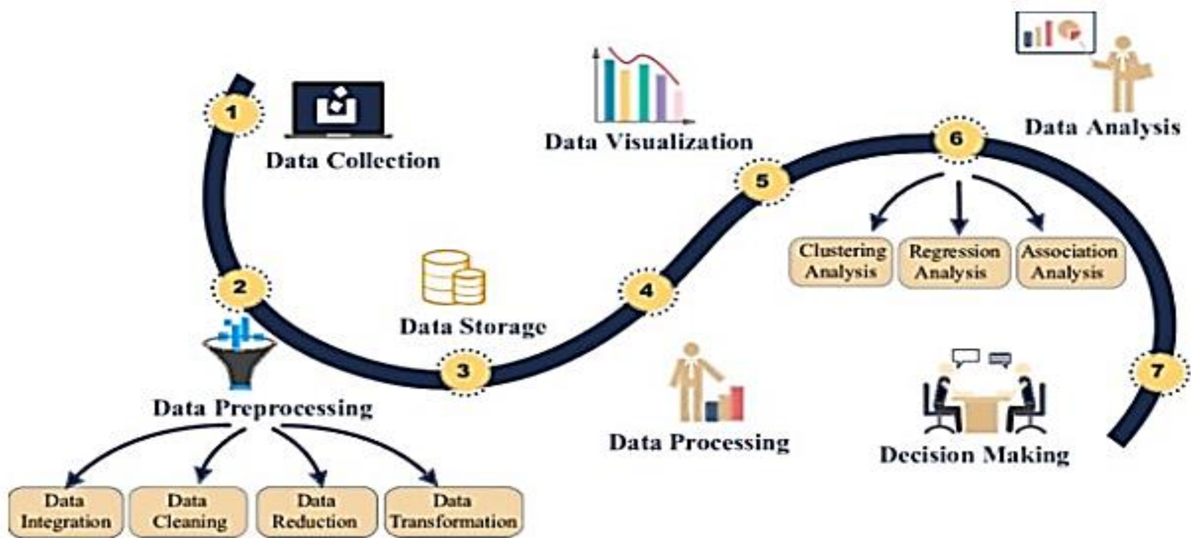


Fig. 2.1. Big data analysis [1]

For businesses to make optimal decisions, it is no longer enough to collect data from the previous fiscal year and perform descriptive analysis of query types. It is increasingly necessary to use predictive analytics tools to remain competitive in a world where the pace of change is accelerating.

Next-generation analytics should not rely solely on performing statistical analysis on the entire data set, as was done with traditional tools. Due to the vast number of data points and attributes, new behaviors and insights can be obtained through advanced analytics, which improves forecasting accuracy.

For example, we can answer the following questions to make adjustments in real time:

«Which stocks are likely to have the highest daily gain based on trading in the last hour?»

«What is the best way to move trucks for today's afternoon, based on the morning's sales, available inventory, and current traffic reports?»

«What maintenance is required for the aircraft based on performance data obtained during the last flight?»

Machine-generated data processing, combined with the geographic reach of very large systems, the number of devices generating data, the diversity of device manufacturers, the frequency of data generation, and the total volume of data, requires new infrastructure software. This infrastructure software must be able to distribute compute and data storage between the edge, fog, and cloud where it best meets business needs. Next-generation data analytics enables businesses to better understand the impact of their products and services, adjust their methods and goals, and deliver better products to their customers faster. The ability to derive new insights from their data delivers business value.

2.4. Data analysis life cycle

There are many methodologies for conducting data analysis, including the popular Cross-Industry Standard Data Processing Process (CRISP-DM), which is used by more than 40% of data analysts.

About 27% of data analysts use their own methodology. The rest use other methodologies. As similar as possible to the scientific method, the data analysis life cycle is designed for use in a business environment (Fig. 2.2). Arrows point in both directions between some steps. This emphasizes the fact that the life cycle may require many iterations before decision makers are confident enough to move forward.

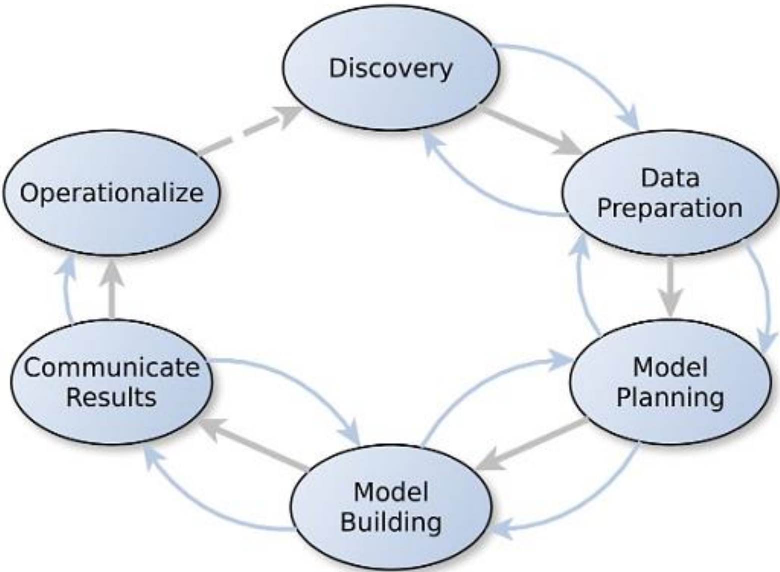


Fig. 2.2. Data analysis life cycle [1]

As in the scientific method, the data analysis lifecycle begins with a question. For example, we might ask, " What was the most common crime in Kyiv on August 20, 2025?" Each step of the data analysis lifecycle includes many tasks that must be completed before moving on to the next step.

Data collection is the process of finding data, determining whether there is enough data to complete the analysis. For example, we are looking for an open data set on crime for Kyiv during August 2025.

Data preparation – this step can involve a lot of work to convert the data into a format suitable for the tool that will be used. The crime dataset may already be prepared for analysis. There are usually some adjustments that can be made to help answer the question.

Model selection – this step involves choosing the analysis technique that will best answer the question with the available data. After selecting the model, the tool (or tools) for analyzing the data is selected.

Data analysis is the process of testing a model on data and determining the reliability of the model and the analyzed data.

Reporting results is usually the last step for data analysts. It is the process of communicating the results to decision makers. Sometimes, data analysts are asked to recommend actions. Given crime data on August 20, a bar chart, pie chart, or some other representation might be used to communicate which crimes are most prevalent. The analyst might suggest increasing police presence in certain areas to deter crime on a specific day, such as August 20.

Decision-making is the final step in the data analysis lifecycle. Organizational leaders incorporate new oucomes as part of an overall strategy. The process begins anew with data collection.

2.5. Open data, its formats and processing tools

There are many different sources of data. A large amount of data can be found in files such as MS Word documents, emails, spreadsheets, MS PowerPoints, PDFs, HTML, and plain text files. These are just a few types of files that contain data. Big data can also be found in public and private archives. Scanned paper archives containing historical data from various sources are definitely Big Data. For example, there is a huge amount of data in health insurance forms and invoices, business statements and customer interactions, and tax documents. This list is only a small part of the archived data.

Internal output for organizations is created through customer relationship management systems, learning management systems, human resource systems and records, intranets, and other processes.

Different programs create files in different formats, which are not necessarily compatible with each other. For this reason, a universal file format is needed. Comma-separated values (CSV) files are a type of plain text file defined in RFC 4180.

CSV files use commas to separate columns in a data table and a newline character to separate rows. Each row is a record. Although they are commonly used for import and export in traditional databases and spreadsheets, there is no specific standard. JSON and XML are also plain text file types that use a standard way to represent data records. These file formats are compatible with a wide range of applications. Converting data to a common format is a valuable way to combine data from different sources.

Fig. 2.3. Data formats [1]

The Internet is a good place to find big data. We can find images, videos, audio. Public web forums also create data. Social media such as YouTube, Facebook, instant messaging, RSS, and Twitter add to the data found on the Internet. Most of this data is unstructured, which means that it is not easy to classify it into a database without some type of processing.

2.6. Web scraping

Web pages are designed to provide information to people, not machines.

Web scraping tools automatically "extract" data from HTML pages. This is similar to a web crawler or search engine spider that explores the Internet to extract data and create a database to answer search queries. Web scraping software can use the Hypertext Transfer Protocol or a web browser to access the network.

Web crawling is an automated process that uses a bot or web crawler. Specific data is collected and copied from the Internet into a database or spreadsheet. The data can then be analyzed.

To perform web scraping, we first need to download a web page and then extract the data we need from it. Web scrapers usually extract something from a page to use for another purpose elsewhere (to find and copy names, phone numbers, and addresses). This is a process known as contact scraping.

In addition to working with contacts, web scraping is used for other types of data retrieval, such as real estate listings, weather data, research, and pFig comparison. Many large web service providers, such as Facebook, provide standardized interfaces for automatic data collection using APIs.

The most common approach is to use RESTful application program interfaces (APIs). RESTful APIs use HTTP as the communication protocol and JSON as the data encoding structure. Websites such as Google and Twitter collect large amounts of static data and time series. Knowing the APIs for these sites allows data analysts and engineers to access the vast amounts of data that are constantly being generated on the Internet.

2.7. Extracting, transforming, and loading data

Databases contain data that has been extracted, transformed, and loaded (ETL – extract, transform, load). ETL is the process of "cleaning" raw data so that it can be placed in a database. Data is often stored in multiple databases and must be combined into a single data set for analysis.

Most databases contain data that is owned by an organization and is private.

After accessing data from various sources, the data needs to be prepared for analysis. Data Science professionals estimate that data preparation can take up 50 to 90 % of the time required to perform analysis.

While the data that will comprise the data set for analysis may come from a variety of sources, they are not necessarily compatible when combined. Another problem is that data that may be represented as text must be converted to a numeric

type if it is to be used for statistical analysis. Data types are important when languages such as Python or R are used to work with data. In addition to different data types, the same data type can be formatted differently depending on its source. For example, different languages may use different characters to represent the same word. For example, British English may use different spellings than American English.

Time and date formats present complexities. Although times and dates are very specific, they are represented in a wide variety of formats. Time and date are essential for the analysis of time series observations. Therefore, they must be converted to a standard format for the analysis to be meaningful. For example, dates may be formatted with the year first followed by the day and month in some countries, while other countries may represent data with the month followed by the day and year.

Similarly, time can be represented in a 12-hour format with AM and PM notation, or it can be represented in a 24-hour format.

When discussing data, we can think of a hierarchy of structures. For example, a data warehouse is a place that stores many different databases in such a way that the databases can be accessed using the same system. A database is a collection of data tables that are related to each other in one or more ways.

Data tables are made up of fields, rows, and values, similar to columns, rows, and cells in a spreadsheet. Each data table can be thought of as a file, and a database can be thought of as a collection of files. Other data structures or objects are used by Python. For example, Python uses strings, lists, dictionaries, tuples, and sets as its basic data structures.

Each data structure has its own set of functions or methods that can be used to manipulate an object. In addition, a popular Python data analysis library called "pandas" uses other data structures, such as date data frames. Much of the data that will be placed in a database so that it can later be queried comes from a variety of sources and in a wide range of formats.

Extraction, transformation and loading (ETL) is the process of collecting data from a variety of sources, transforming the data, and then loading the data into a database.

A company's data can be found in Word documents, spreadsheets, plain text, PowerPoints, emails, and PDFs. This data can be stored on different servers using different formats.

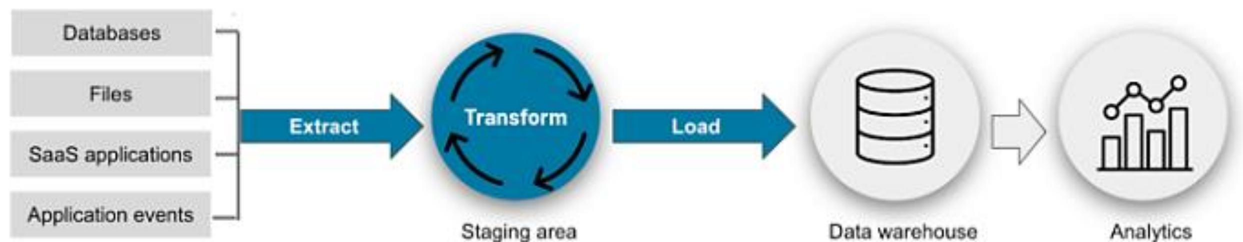


Fig. 2.4. Data extraction, transformation and loading [2]

The ETL process contains the following three main steps.

Step 1. Extraction – data is collected from multiple sources.

Step 2. Transformation – once the data is collected, it must be transformed. Data transformation can include aggregating, sorting, cleaning, and joining the data.

Step 3. Loading – the transformed data is loaded into the database for queries.

The above descriptions of the three stages of the ETL process are simplified. In reality, there is a lot of work that needs to be done before data can be loaded into a database and then queried. The extraction stage collects the desired data from the source and makes it available for processing. Mining transforms the data into a single format that is ready for transformation.

For example, combining data from a NOSQL server and an Oracle DB will provide data in different formats. This data must be converted to a single format. The data must also be validated to ensure that it has the desired type of information (values). This is done using validation rules. If the data does not meet the validation rules, it can be rejected. Sometimes this rejected data is corrected and validated.

During extraction, all necessary data from the source(s) is obtained using minimal computing resources so as not to impact the network or computer performance.

Fig. 2.5. The process of "extracting" data [1]

The transformation step uses rules to convert the source data into the data type required by the target database. This includes converting any measured data to the same dimension. The transformation step also requires several other tasks.

Some of these tasks include combining data from multiple sources, aggregating it, sorting it, determining new values that are calculated from the aggregated data, and applying validation rules. The data (including some rejected data) may go through another part of the transformation step, known as data “cleaning.”

The cleaning part of the transformation step further ensures that the source data is consistent. The loading step is when the transformed data is loaded into the target database. Some organizations may overwrite existing data with cumulative data.

Loading new transformed data can occur hourly, daily, weekly, or monthly. This can only occur when a certain number of changes have occurred to the transformed data.

During the load step, rules defined in the database schema are applied. Some of these rules check for uniqueness and consistency of data, fields that have mandatory ownership, have required values, etc. These rules help ensure that the load and subsequent data queries are successful.

Conclusion to lecture 2

Data today is no longer stored on multiple machines and processed by just one tool . Decision makers are increasingly relying on data analytics to extract the right information at the right time, in the right place, and make the right decision.

Predictive analytics predicts outcomes and suggests courses of action that will have the greatest benefit to the organization. Files, data from the Internet, sensors, and databases are examples of data sources. The capabilities of modern data analysis tools allow efficient handling of large and complex datasets, making open data more accessible and valuable for research and industry applications. Python's flexibility and extensive ecosystem make it a central technology for contemporary data analysis. A comparison between traditional and next-generation Big Data analytics illustrated the shift toward more agile, real-time, and AI-driven methods. Understanding the data analysis lifecycle from data collection to interpretation is crucial for building effective data-driven solutions. Special attention was given to open data formats and processing tools, with techniques such as web scraping and ETL (Extract, Transform, Load) pipelines being vital for acquiring and preparing data from diverse sources.

Questions for reinforcement

1. What is the role of Python in data analysis?
2. Describe traditional big data analytics and next-generation analytics.
3. Describe the data analysis life cycle.
4. Describe open data, its formats and processing tools.
5. What is web scraping?
6. Explain the processes of data extraction, transformation, and loading.

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. Extract, transform, and load (ETL). URL: https://docs.microsoft.com/en-us/azure/architecture/data-guide/relational-data/etl

Lecture 3.
Formatting time and date data, reading and writing files in Python. Interacting with external applications

Lecture plan

3.1. Formatting time and date data in Python.
3.2. Reading and writing files in Python.
3.3. Interaction with external applications.

3.1. Formatting time and date data in Python

The Python module used to process time and date data is called **datetime** [1]. The **datetime** module is included in most Python distributions as a standard library; it must be imported to be used in code. The features of the **datetime** module are presented in an object-oriented programming paradigm.

The module consists of time and date classes. Each class has its own methods that can be called to operate on instances of the classes, which are called objects [2]. Fig. 3.1 illustrates the use of some basic objects and methods, and includes in module **datetime**.

Term	Example	Use
module	`datetime`	`import datetime as dt`

class	`time`, `date`, `datetime`	`dt.time`
		`dt.date`
		`dt.datetime`

object	Variables `t`, `d`, and `dateAndTime`	`t = dt.time(12, 31, 0)`
		`d = dt.date(2025, 12, 31)`
		`dateAndTime = dt.datetime.now()`

method	`strftime()`, `weekday()`, `isoformat()`	`t.strftime("%H:%M:%S")`
		`d.weekday()`
		`dateAndTime.isoformat()`

Fig. 3.1. Basic **datetime** objects and methods [3]

The stripping method uses a series of formatting codes or directives as its parameters (Fig. 3.2).

Directive	Meaning
%a	Locale's abbreviated weekday name
%A	Locale's full weekday name
%b	Locale's abbreviated month name
%B	Locale's full month name
%c	Locale's appropriate date and time representation
%d	Day of the month as a number [01,31]
%H	Hour (24-hour clock) as a number [00,23]
%I	Hour (12-hour clock) as a number [01,12]
%j	Day of the year as a number [001,366]
%m	Month as a number [01,12]
%M	Minute as a number [00,59]
%p	Locale's equivalent of either AM or PM
%S	Second as a number [00,61]
%w	Weekday as a number [0,6]
%W	Week number of the year (Monday as the first day of the week) as a number [00,53]
%y	Year without century as a number [00,99]
%Y	Year with century as a number
%Z	Time zone name (no characters if no time exists)

Fig. 3.2. List of **datetime** module formatting codes [3]

Fig. 3.3 shows Python code that uses the **datetime** module to represent dates and times in a format commonly used in the United States.

```
#load the datetime module as dt
import datetime as dt

#create a datetime object that contains the current time
currentDT = dt.datetime.now()

#view the value of currentDT
print(currentDT)

#create a new string object that contains the reformatted date and time
UDdt = currentDT.strftime('%b %d, %Y %I:%M %p')

#display the result
UDdt
```

Fig. 3.3. Using the **datetime** module [3]

3.2. Reading and writing files in Python

Csv module is part of the Python standard library. The **csv** module allows read and write to *.csv* files. Python also has basic methods for creating, opening, and closing external files. Data tables will only exist in RAM until they are saved to files.

Open() method is used to create a new file or to open an existing file that will contain data that we want to save. The **close()** function removes data from the buffers and ends the file write operation for the specified file. It is important to close all files that we do not want to write data to. This saves system resources and protects the file from corruption. In Fig. 3.4 shows the syntax of the **open()** function and explains some important values that can be provided to the method. It also illustrates the use of the **close()** method. This code creates a file, closes it, and then reopens it in append mode so that data can be appended to the file.

```
myFile = open("newText.txt", "w")
myFile.close()
myFile = open("newText.txt", "a")

print(myFile)
```

```
<open file 'newText.txt', mode 'a' at 0x729e2338>
```

Fig. 3.4. Using **open** () methods and **close() datetime** module [3]

Opening a non-existent file in "a" mode will create the file in the same way as in " w " mode. The only difference is where the pointer is. It will point to the beginning of the file or the end of the file. Fig. 3.5 explains some of the important values that can be passed to the **open()** method. These parameters can be combined (the "+" symbol) to indicate that they should be used in read, write, and append modes.

Modes	Description
w	Opens an existing file that has the file name specified. If the file does not exist, it opens a new file with that name.
r	Opens an existing file in read only mode.
a	Opens an existing file and will append new data to the end of the file.

Fig. 3.5. Parameters of the **open** () method [3]

Data can be written to a file using the **write()** method. If the file was opened in "a" mode, the data will be appended to the end of the file. We may need to add " \ " characters to the format for formatting. For example, the \n or \r\n characters will add line breaks to the end of the written line of data.

The file method **read()** reads the contents of an open file object (Fig. 3.6).

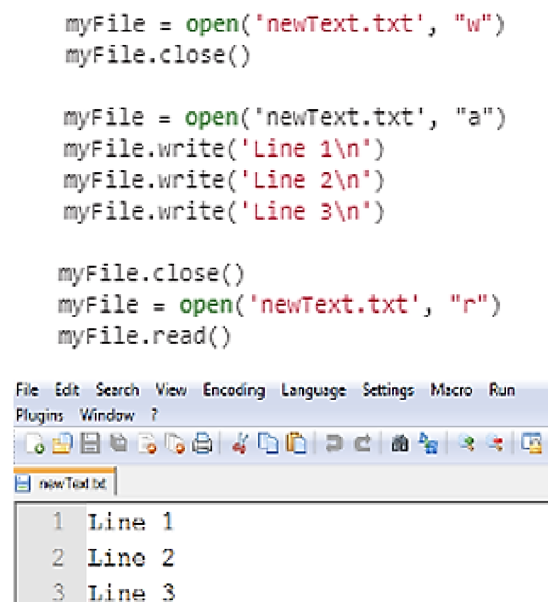


Fig. 3.6. Reading and writing data to a file [3]

In Fig. 3.6. a new file is created in input cell one. The file is closed. In cell two, the file is reopened in append mode. Three lines of text are written to the file. In cell three, the file is closed to verify that the text has been written to the file.

The file is reopened in read mode, and the method **read()** is used to view the file. The file is also displayed as a text editor, which formats the text using the " \ " characters to create three separate lines.

3.3. Interaction with external applications

Python allows interact with external applications and the operating system. Jupyter Notebook is an open source web application that allows create and share documents containing running code, mathematical equations, visualizations, and text.

Uses include: data cleaning and transformation, numerical modeling, statistical modeling, data visualization, machine learning, and more [4].

In Jupyter Notebooks The "!" character allows interact directly with the operating system. For example, Fig. 3.7 shows two Linux commands executed in Jupyter Notebooks. The commands begin with the "!" character.

```
! ls -al
```

```
total 80
drwxr-xr-x 4 root root 16384 Feb 22 19:47 .
drwxr-xr-x 5 root root 16384 Feb  6 22:34 ..
drwxr-xr-x 4 root root 16384 Feb 19 21:01 chapter 3
drwxr-xr-x 2 root root 16384 Feb 22 19:47 .ipynb_checkpoints
-rw-r--r-- 1 root root    72 Feb 22 19:47 Untitled.ipynb
```

```
!head logins.csv
```

```
Student,Login_Day,Login_Time
Rose,12/4/2016,20:29
Joe,9/4/2016,22:37
Jerry,2/20/2017,4:18
Lawrence,4/7/2016,13:17
Roy,6/24/2016,15:30
Robin,9/6/2016,20:55
Harry,8/12/2016,14:26
Mary,1/26/2017,1:46
Stephanie,12/31/2016,10:23
Tammy,5/25/2016,7:08
```

Fig. 3.7. Linux commands executed in Jupyter Notebooks to interact with the operating system [3]

Fig. 3.8 shows the use of the **subprocess** module to communicate with an external program and store the output of a command issued to that program in a Python object. First, an object is created to hold the command that the program sends. In this case, we intend to send the command to the ping utility, which is available from the Linux shell. We then send this command, broken down into individual words, to the program using the **subprocess** method.

Finally, we store the output of the command in a variable and break it into a string. We can then iterate through the contents of the object with printing and address its individual elements using string indexing.

```python
'''import the subprocess library which is necessary for communication with external apps'''
import subprocess
#We now execute the ping process as of from the shell:

pingCmd = 'ping -c 127.0.0.1'
process = subprocess.Popen(pingCmd.split(), stdout=subprocess.PIPE)
'''create an object to hold the output of the process and split the output elements into a list''

process_output = process.communicate()[0]
process_output = process_output.split()
#view the contents of the output object
print(process_output)
```

```
['PING'. '127.0.0.1', '(127.0.0.1)', '56(84)', 'bytes', 'of', 'data.', '64', 'bytes',
 'from', '127..0.0.1', 'icmp_seq=1', 'ttl=64', 'time=0.094', 'ms', '64', 'bytes',
 'from', '127.0.0.1', icmp_seq=2', 'ttl=64', 'time=0.052', 'ms', '---', '127.0.0.1',
 'ping', 'statistics', '---', '2', 'packets', 'transmitted', '2', 'received,', '0%',
 'packet', 'loss,', 'time', '999ms', 'rtt', 'min/avg/max/mdev', '=',
 '0.052/0.073/0.094/0.021', 'ms']
```

```python
#view the first five elements of the list
process_output[0:5]
```

```
['PING', '127.0.0.1', '(127.0.0.1)', '56(84)', 'bytes']
```

Fig. 3.8. Using the **subprocess** module to communicate with an external program [3]

This code snippet demonstrates how to use Python’s **subprocess** module to execute a system command, in this case, a ping to the local host address 127.0.0.1. The **subprocess.Popen** function is used to run the ping command as if it were entered directly into the system shell, and the standard output is captured via a pipeline.

After the process completes, **communicate()** retrieves the command’s output, which is then split into a list of words for easier manipulation and analysis.

The printed output shows the full result of the ping operation, broken into individual components like IP addresses, response times, packet statistics, and other metrics. Additionally, the code demonstrates how to access specific parts of this output by slicing the list, specifically viewing the first five elements. This approach highlights a fundamental method for interacting with external system processes through Python and processing their outputs for further analysis or automation.

Conclusion to lecture 3

To process time and date data, the Python **datetime** module is used. The **csv** module allows read and write to *.csv* files.

Python also has basic methods for creating, opening, and closing external files. Data tables will only exist in RAM until they are saved to files. The **open()** method is used to create a new file or to open an existing file that will contain the data to be saved. The **close()** function removes data from the buffers and ends the file write function for the specified file.

Python allows interact with external applications and the operating system. The **subprocess** module is used to communicate with an external program and save the output of a command issued to that program into a Python object.

Questions for reinforcement

1. How is time and date data formatted in Python?
2. Reading and writing files in Python?
3. How does Python interact with external applications?
4. What is Jupyter Notebook used for?
5. What is the subprocess module used for in Python?

List of recommended literature

1. datetime – Basic date and time. URL:
https://docs.python.org/3/library/datetime.html
2. Python – Object Oriented. URL:
https://www.tutorialspoint.com/python/python_classes_objects.htm
3. Data Analytics Essentials. URL:
https://www.netacad.com/courses/iot/big-data-analytics
4. Jupyter Notebook. URL: https://jupyter.org/

Lecture 4.
Python and SQLite programming.
Purpose of the csvsql utility

Lecture plan

4.1. Basic SQL operations.
4.2. Working with Python and SQLite.
4.3. Purpose of the csvsql utility and the execute() method.

4.1. Basic SQL operations

There are many ways to work with external files in Python.

SQLite is an implementation of SQL that works well with Python. Instead of using a client-server approach, it uses a connection established between Python and a SQL database, creating a SQL connection object. This object will have methods associated with it. After the connection object is created, the cursor object is created using the create method. The cursor object has SQLite methods available to perform SQL operations on the database. SQL is a language for interacting with databases and tables. There are a number of dialects of SQL, but some basic operations are standard, they work similarly to SQLite, MySQL, or other SQL implementations. SQLite can be run interactively from the command line. The Python programming language can interact with SQLite by importing modules into.

SQLite offers several advantages, including its lightweight nature, ease of setup, and minimal configuration requirements, making it ideal for embedded applications, prototyping, and small to medium-sized projects. It is serverless, meaning it does not require a separate server process, which simplifies deployment and reduces overhead. SQLite stores the entire database in a single file, ensuring portability and ease of backup. Additionally, it is highly reliable, supports ACID (Atomicity, Consistency, Isolation, Durability) transactions, and is fully open-source, allowing developers to integrate it freely into commercial and open applications. SQL is a language that consists of three special-purpose languages.

The first is the data definition language. It is used to create and manipulate the structure of databases and SQL tables (Table 4.1).

Table 4.1. Common data definition language SQL commands

Command	Explanation
ALTER TABLE	Changes the structure of an existing table
CREATE DATABASE	Creates a new empty database
CREATE TABLE	Creates a table within an existing database
DESCRIBE	Displays the structure of a table
DROP ATABASE	Completely deletes an entire database
DROP TABLE	Deletes a table from a database
USE	Opens the database to work with

The second is a data manipulation language. It is used to add, delete, or transform data that is in data tables (Table 4.2).

Table 4.2. Common commands of data manipulation language SQL

Command	Explanation
DELETE	Deletes existing data
INSERT	Adds new data
REPLACE	Replaces records that have duplicate data with the records to be inserted
UPDATE	Replaces values in columns of data with new values based on specified criteria

The third is a data query language, which is used to access data in data tables to generate information (for example, the SELECT command accesses data based on a given set of criteria).

4.2. Working with SQLite in Python

Let's look at a sequence of commands that illustrate the basics of SQLite operations. First, we need to install an external tool called csvkit on operating system so that we can import a csv file into an SQLite database. The following commands show the steps of creating an SQLite database, importing CSV data into the database, executing a query on the data table, and viewing the query results.

```
import sqlite3 as sq1
```

Create a new database and connect to it:

```
conn = sq1.connect('logins.db')
```

```
!csvsql --db sqlite://///logins.db --insert logins.csv
```

We create object cursor for SQL queries :

```
cur = conn.cursor()
```

Creating an SQL query as a string object:

```
query = 'SELECT * FROM logins LIMIT 5'
```

SQL query on the object cursor now contains the query result:

```
cur.execute(query)
```

C we create a loop that iterates over the number of rows in the object and cursor and prints their contents:

```
for row in cur:
    print(row)
```

```
(u'John', u'2016-12-29', u'1970-01-01 14:24:13.000000')
(u'Allan', u'2016-12-29', u'1970-01-01 03:16:54.000000')
(u'Robert', u'2016-12-30', u'1970-01-01 04:54:25.000000')
(u'Eve', u'2016-12-30', u'1970-01-01 08:32:14.000000')
(u'Leslie', u'2016-12-30', u'1970-01-01 20:34:54.000000')
```

The command "!csvsql --db ..." can be executed as the first command. This is an external tool that needs to be installed on the OS. We can use the command line

(Linux CLI) to execute this command line, but to simplify things, we can execute this external command directly from laptop by prefixing the command with "!" [1].

4.3. Purpose of the csvsql utility

Sqlcsv is a simple command line tool that can be used to retrieve data from a database and export the result as CSV and insert data into a database from CSV. Works with Python 3 only [2] The examples below use the following MySQL table schema:

CREATE TABLE testtable(

id INT AUTO_INCREMENT PRIMARY KEY,

int_col INT,

float_col FLOAT,

varchar_col VARCHAR(255))

Let's look at how to access a database from the Python programming language. Other languages use a similar model: the names of libraries and functions may differ, but the concepts are the same. Here's a short Python program that retrieves latitudes and longitudes from an SQLite database stored in a file called survey.db:

```
1. import sqlite3
2. connection = sqlite3.connect("survey.db")
3. cursor = connection.cursor()
4. cursor.execute("SELECT Site.lat, Site.long FROM Site;")
5. results = cursor.fetchall()
6. for r in results:
7. print(r)
8. cursor.close()
9. connection.close()
```

Result of program execution:

```
(-49.85, -128.57)
(-47.15, -126.72)
(-48.87, -123.4)
```

The program starts by importing the sqlite3 library. If we were connecting to MySQL or another database, we would import a different library, but they all provide the same functionality, so the rest of our program shouldn't change (at least not much) if we switch from one database to another. Line 2 establishes a connection to the database. Since we're using SQLite, all we need to specify is the name of the database file. Other systems may require us to provide a username and password. Line 3 then uses this connection to create a cursor object. Much like a cursor in an editor, its role is to keep track of where we are in the database. In line 4, we use this cursor to ask the database to execute a query for us.

The query is written in SQL and passed to cursor.execute as a string. We need to make sure that the SQL is formatted correctly; if it isn't, or if something goes wrong when it's executed, the database reports an error.

The database returns the results of the query in response to cursor.fetchall on line 5. This result is a list with one entry for each record in the result set; if we iterate over this list (line 6) and print those list entries (line 7), we see that each of them is a tuple with one element for each field we requested. Finally, lines 8 and 9 close our cursor and connection, since the database can only keep a limited number of them open at a time.

Since establishing a connection takes time, we should not open a connection, perform one operation, then close the connection, only to reopen it a few microseconds later to perform another operation. Instead, it is better to create a single connection that will remain open for the lifetime of the application [3].

Consider Python code example using SQLite:

```
import sqlite3
# Connect to a database (or create one if it doesn't exist)
connection = sqlite3.connect('example.db')
# Create a cursor object to interact with the database
cursor = connection.cursor()
# Create a table
cursor.execute('''
```

```
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER
)
''')
# Insert some data
cursor.execute('INSERT INTO users (name, age) VALUES (?, ?)', ('Alice', 30))
cursor.execute('INSERT INTO users (name, age) VALUES (?, ?)', ('Bob', 25))
# Save (commit) the changes
connection.commit()
# Query the data
cursor.execute('SELECT * FROM users')
rows = cursor.fetchall()
# Print the results
for row in rows:
    print(row)
# Close the connection
connection.close()
```

This Python script demonstrates how to use SQLite for basic database operations. First, it connects to a database file named example.db, creating it if it doesn't exist. Then, it creates a table called users with three fields: id, name, and age. The code inserts two sample users into the table and commits the changes to save them. After that, it retrieves all records from the users table and prints them out. Finally, it closes the database connection to free up resources.

Conclusion to lecture 4

SQLite is an implementation of SQL that works well with Python. Instead of using a client-server approach, it uses connections established between Python and a SQL database by creating a SQL connection object.

After creating the connection object, it uses the create method to create a cursor object. The cursor object has SQLite methods available to perform SQL operations on the database. SQLite can be run interactively from the command line, and Python can also interact with SQLite through module imports.

SQLite is an implementation of SQL that works well with Python, providing a lightweight, fast, and self-contained database engine that requires no server setup. It allows developers to manage structured data easily within their Python applications, using simple commands to create, query, and manipulate databases. This makes SQLite especially useful for small to medium-scale big data projects where quick data storage and retrieval are needed without the complexity of full-scale database management systems. Its seamless integration with Python's standard library through the sqlite3 module makes it a popular choice for prototyping, data analysis, and even mobile and IoT applications dealing with large volumes of structured data.

Questions for reinforcement

1. What three special-purpose languages does SQL consist of?
2. List the basic commands of the SQL language.
3. What is the purpose of the csvsql utility?
4. How to access databases from programs written in Python?
5. Purpose of the csvsql utility?

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. sqlcsv. URL: https://pypi.org/project/sqlcsv/
3. Programming with Databases – Python. URL: https://swcarpentry.github.io/sql-novice-survey/10-prog/index.html

Lecture 5.
Procedure for importing data from files into Pandas.
Importing data from the Internet.
Tools for correlation analysis in Pandas

Lecture plan

5.1. Statistical approaches to big data analytics.
5.2. Using Pandas.
5.3. Importing data from files.
5.4. Importing data from the Internet.
5.5. Descriptive statistics in Pandas.
5.6. Tools for correlation analysis in Pandas.

5.1. Statistical approaches to big data analytics

Big data analytics uses various statistical approaches. Descriptive statistics describe a sample. It is useful for understanding sample data and for determining its quality. When working with large amounts of data coming from many sources, many problems can arise. Sometimes data points can be corrupted, incomplete, or missing altogether. Descriptive statistics can help determine how much of the data in a sample is useful for analysis and determine criteria for removing data that is irrelevant or problematic. Descriptive statistics plots are a useful way to quickly make judgments about a sample. For example, a sample of tweets may be selected for analysis. Some tweets in the sample contain only characters, while other tweets contain characters and images. We can analyze tweets that contain images or tweets without images. This will allow us to identify invalid tweets based on a very simple criterion. Data points that do not meet the basic criteria will be removed from the sample before the analysis begins.

Machine learning methods are often used in big data analytics.

- **Cluster analysis** – used to find groups of observations that are similar to each other.
- **Association** – used to find common occurrences of values for different variables.

• **Regression** – used to quantify the relationship between variations in one or more variables, if any.

In machine learning, computer software is either provided with or given its own set of rules that are used to perform analysis. Machine learning methods can require a lot of processing power and have only become viable with the availability of parallel processing.

Fig. 5.1 shows a table with two fields. One field contains a variable, and the other contains statistics that describe the value of that variable. In this example, ten students took a quiz for ten points. When the teacher analyzes the scores, a score distribution is created, as shown in the second table. This expresses the number of times the score occurred in the class. The probability of a score is expressed as the ratio of the score frequency to the total number of scores.

Student	Quiz (10 points)	Score	Score Frequency	Score Probability
Student 1	6	1	0	0
Student 2	7	2	0	0
Student 3	7	3	0	0
Student 4	8	4	0	0
Student 5	7	5	1	0.1
Student 6	9	6	1	0.1
Student 7	10	7	4	0.4
Student 8	8	8	2	0.2
Student 9	7	9	1	0.1
Student 10	5	10	1	0.1

Fig. 5.1. Example of tables with student grades for a completed quiz [1]

A frequency distribution consists of all the unique values of a variable and the number of times the values occur in a data set. Probability distributions use the proportion of time a value occurs in the data instead of frequencies. A histogram can represent the distribution of a data set. In the case of a discrete variable, each bin of the histogram is assigned a specific value. In the case of a continuous variable, each

bin is associated with a range of values. In both cases, the height of the bin represents the number of times the value of the variable takes on a given value or falls within the range, respectively.

The histogram representation of the distribution of data can take any shape. In the case of a continuous variable, the shape will also depend on the width of the bins, i.e. their range. Some shapes can be modeled using well-defined functions called probability distribution functions.

Probability distribution functions allow us to represent the shape of the entire distribution of a data set using only a small set of parameters, such as the mean and variance. A probability distribution function that is particularly well suited to representing many events that occur in nature is the Gaussian, or normal, distribution, which is symmetrical and bell-shaped. Other distributions are not symmetrical. The peak of the graph can be to the left or right of the center. This property of the distribution is called skewness. Some distributions will have two peaks and are known as bimodal. The right and left ends of the distribution graph are known as the tails. One commonly used characteristic of distributions is the measure of central tendency. These measures express the value that the variable has that is closest to the central position in the data distribution. Common measures of centrality are the mean, median, and mode. The mode of a data sample is the value that occurs most often (Fig. 5.2). Values that are closer to the center of the distribution occur with greater frequency.

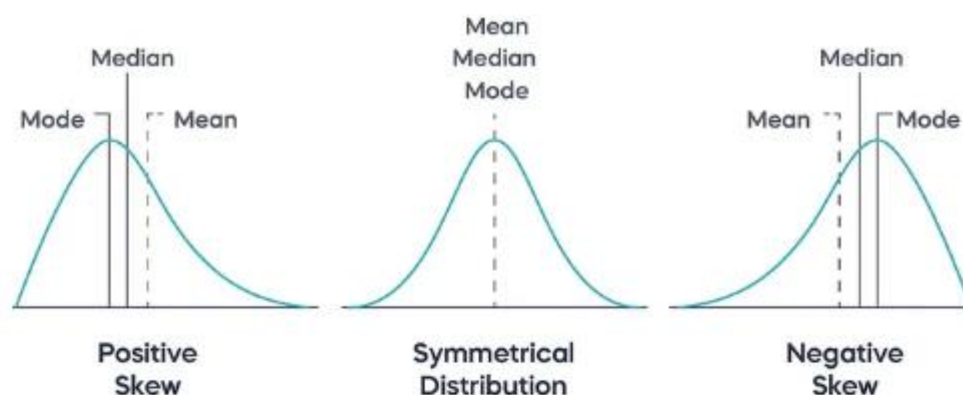


Fig. 5.2. Common measures of centrality are the mean, median, and mode [1]

The mean, also known as the average, is the most well-known measure of central tendency. It is equal to the sum of all the data values divided by the number of values in the data set. Although the mean is commonly used in everyday life, it is generally not the best indicator of the most representative value for a distribution. For example, if there are unusually high or low values in a distribution, the mean can be greatly influenced by those extreme values, called outliers. Depending on the number of outliers in the data set, the mean or median becomes "skewed" or shifts in one direction or the other.

The median is the middle value in a data set after the list of values has been ordered (sorted). The median is not sensitive to these extreme values. Since the total number of values and the actual values in the data set are the same, the middle of the list, or median, remains the same. Depending on the number of outliers in the data set, the mean or average is "skewed" or shifted to one side or the other.

While the mean is used to describe many distributions, it leaves out an important part of the overall picture, which is the variability of the distribution. For example, we know that outliers can skew the mean. The median gets us closer to what is most important in the distribution, but we still don't know how spread out the values in the sample are. The basic way to describe variability in a sample is to calculate the difference between the highest and lowest values for the variable. This statistic is known as the range.

The variance of a distribution is a measure of how far each value in a data set is from the mean. Associated with variance is the standard deviation, which is used to standardize distributions as part of a normal curve, as shown in Fig. 5.3.

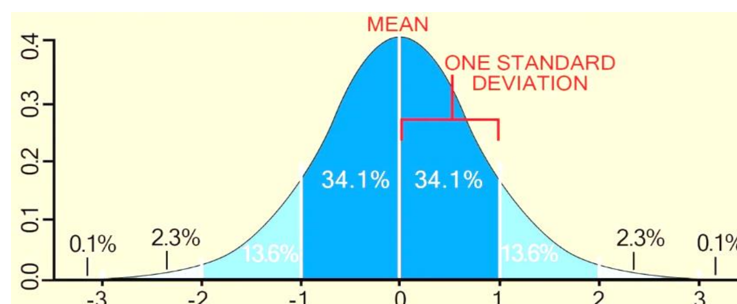


Fig. 5.3. Standard deviations example [1]

5.2. Using Pandas

Pandas is an open source library for Python that adds high-performance data structures and tools for analyzing large data sets. Pandas data structures include series structures and dataframe structures. Dataframes are the primary structure of Pandas and are the most commonly used. A dataframe is similar to a spreadsheet with rows and columns. In addition, dataframes can have additional indexes and columns, which are labels for rows and columns [2].

Data frames are easily constructed from a number of other data structures and external files, such as *csv*. A wide range of methods are available to data frame objects. Rows and columns can be manipulated in various ways, and operators are available to perform mathematical, string, and logical transformations on the contents of the data frame (Fig. 5.4).

	First Name	Last Name	Phone Number
1	Mary	Pratt	410-555-9697
2	Oscar	Milde	555-887-9547
3	Timmy	Thomas	471-555-9687

- two-dimensional
- labeled
- different types accomodated

Fig. 5.4. Pandas date frame components example [1]

Pandas is imported into a Python program using **import**, just like other modules. It is usually easier to type using import **pandas as pd** to reference pandas components. Below is the code needed to create the date frame shown in Fig. 5.5.

```
import pandas as pd

data = {'First Name': ['Mary', 'Oscar', 'Timmy'],
        'Last Name': ['Pratt', 'Milde','Thomas'],
        'Phone Number': ['410-555-9697', '555-887-9547', '471-555-9687']}
directory = pd.DataFrame(data, columns=['First Name', 'Last Name', 'Phone Number'])
```

	First Name	Last Name	Phone Number
0	Mary	Pratt	410-555-9697
1	Oscar	Milde	555-887-9547
2	Timmy	Thomas	471-555-9687

Fig. 5.5. Creating a date frame manually [1]

5.3. Importing data from files

Large data sets are collected from various sources and may exist in various file types. Creating a Pandas data frame by encoding data values individually is not very useful for big data analysis. Pandas includes some easy-to-use functions for importing data from external files, such as *csv*, into date frames. We will recreate the date frames of a telephone directory, this time from a larger csv file. Pandas includes a date frame function called **read_csv().**

The procedure is as follows.

Step 1. Import the Pandas module.

```
import pandas as pd
```

Step 2. Check if the file is available in the current working directory. In this case, the **head command** Linux is used to inspect a file and preview its contents.

```
!head -n 5 directory.csv
```

```
First Name,Last Name,Phone Number
Mary,Pratt,410-555-9697
Oscar,Milde,555-887-9547
Timmy,Thomas,471-555-9687
John,Smith,252-959-8421
```

Step 3. To import the file into a date frame object, use the Pandas **read_csv() method.**

```
df_directory = pd.read_csv('directory.csv')
```

Step 4. Use the pandas **info()** dateframe method to view a summary of the file contents.

```
df_directory.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15 entries, 0 to 14
Data columns (total 3 columns):
First Name      15 non-null object
Last Name       15 non-null object
Phone Number    15 non-null object
dtypes: object(3)
```

Step 5. Display the date frame. The head **()** method is used to display the headers, index, and values for the first five rows.

```
df_directory.head()
```

5.4. Importing data from the Internet

Importing data from the web with Pandas is easy. While there are many application programming interfaces (APIs) available for accessing web data, including streaming data, static datasets can also be retrieved from the web based on a file URL.

In the example shown in Fig. 5.6, the dataset is imported into a dataset from the large collection of the Humanitarian Data Exchange [3]. This website is a good resource for people interested in exploring data related to international humanitarian issues.

```
import pandas as pd
url =
'http://manage.humdata.org/hdx/api/exporter/indicator/csv/TT014/source/mdgs/fromYear
/1950/toYear/0/language/en/TT014_Baseline.csv'

from_url = pd.read_table(url, sep=',')

from_url.head()

from_url.info()
```

Fig. 5.6. Importing data from the Internet into Pandas [1]

We import a dataset containing information on the percentage of women serving in national parliaments for a number of countries over a number of years. The process involves the following steps:

Step 1. Import Pandas.

Step 2. Create a string object that contains the file URL.

Step 3. Import file into a date frame object using the pandas **read_table()** method.

Step 4. Verify the import using **head()** and **info()**. The output of **info()** indicates the number of missing values (null records), which is the difference between the total number of records and the number of non-null records for each year.

For example, sites like Google and Twitter have APIs that allow Python programs to connect to streaming data.

5.5. Descriptive statistics in Pandas

Pandas provides simple way to view basic descriptive statistics for a data frame. The **describe()** method for date frame objects displays the following for numeric data types:

- **count** – the number of values included in the statistics;
- **mean** – average value;
- **std** – standard deviation of the distribution;
- **min** – the smallest value in the distribution;
- **25%** – value for the first quartile (25% of values are at or below this value);
- **50%** – the value for the second quartile or median (50% of the values are at or below this value);
- **75%** – value for the second quartile (75% of values are at or below this value);
- **max** – the highest value in the distribution.

5.6. Tools for correlation analysis in Pandas

Cause and effect and correlation are types of relationships between conditions or events. A cause and effect relationship is a relationship in which one thing changes or is created directly because of something else.

For example, rising global temperatures cause the Arctic ice cap to shrink. This is an intuitive relationship to the phenomena. Rising global temperatures can also lead to a decrease in the consumption of wool for use in making warm clothing. The warmer the climate, the less demand there will be for warm clothing.

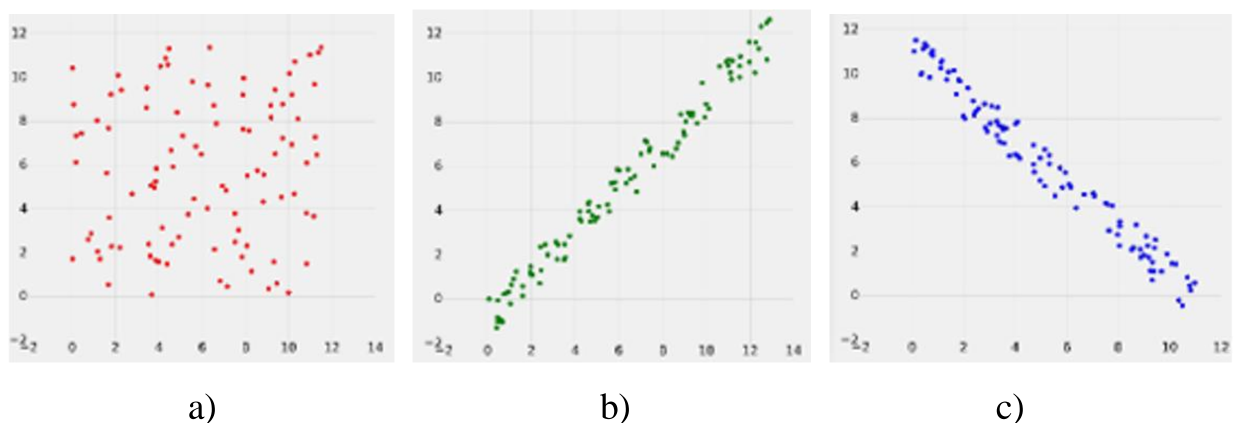
Correlation is a relationship between quantities in which two or more quantities change at the same rate. For example, if global temperature and wool consumption decrease, these quantities change at the same rate and in a similar direction (both decrease).

Correlations can be positive or negative. Positively correlated quantities change in the same direction. If one quantity increases, the other also increases. Negative correlation occurs when the quantities change in some similar proportion

but in opposite directions. In other words, if one increases, the other decreases similarly. Correlations between quantities can be quantified using statistical approaches.

The most common statistic for calculating correlation is the Pearson correlation coefficient, a quantity expressed as a value between -1 and 1. Positive values indicate a positive relationship between changes in the two quantities. Negative values indicate an inverse relationship.

The magnitude of either positive or negative values indicates the degree of correlation. The closer the value is to 1 or -1, the stronger the relationship, 0 indicates no correlation (Fig. 5.7).



a) b) c)

Fig. 5.7 Scatterplots of small data sets that have low, positive and negative correlation, a) low correlation, r =0.0114, b) strong positive correlation, r =0.99, c) strong negative correlation, r =-0.985 [1]

Correlations can be calculated for multiple variables at once. This will result in the calculation of correlation coefficients between all fields supplied in the data frame. The result can be a large table of correlation coefficients.

A visualization called a **heat map** is useful for understanding how the values of the correlation coefficients relate to each other. Scatter plots are useful for quickly visualizing possible correlations in a data set. In a heat map, the fields in the data form horizontal and vertical labels for a grid of values (Fig. 5.8).

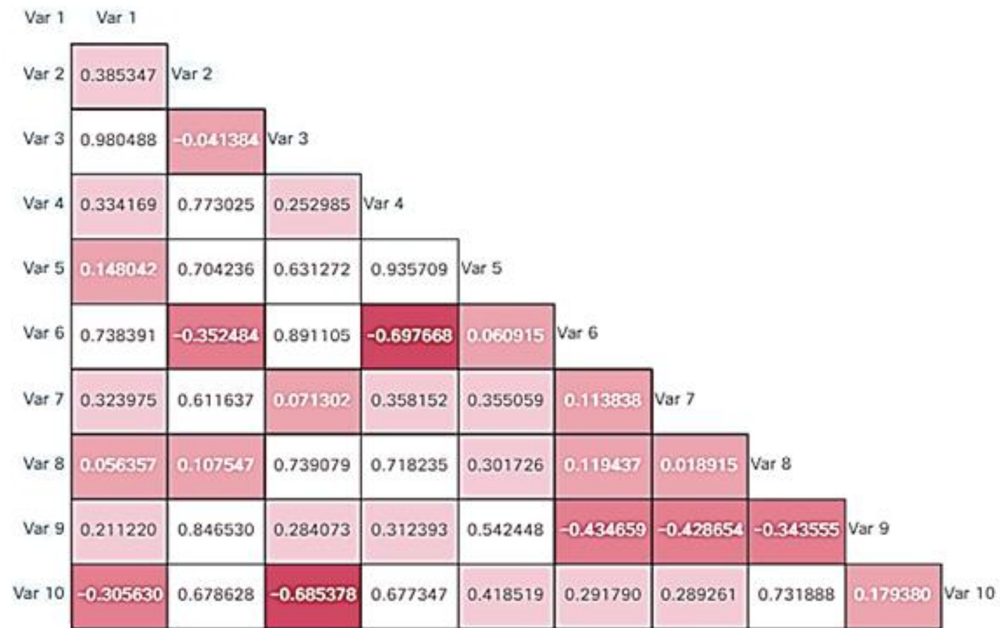


Fig. 5.8. Heat map of correlation coefficients between 10 variables [1]

Each cell value is the correlation coefficient of the field on the horizontal dimension of the grid with the field on the vertical axis.

The value at the intersection of the selected dimensions is the coefficient for that pair of values. To help interpret the data, the correlation values are color-coded. The intensity or shade of color for each value is proportional to that value. For example, all negative correlation coefficients might be represented in a shade of red, and all positive ones in a shade of blue. The deeper the color, the closer the value is to 1 or -1. This helps to understand the meaning of the correlation data.

Conclusion to lecture 5

Exploratory data analysis provides descriptive and graphical summaries of data in order to results revealed interesting patterns. Observations, variables, and values are crucial to analysis.

Pandas is an open source Python library with tools for analyzing large data sets, importing data from files and the web, examining descriptive statistics, and finding statistical relationships between data sets. Data usually requires cleaning, transformation, and processing before data analysis.

Pandas is a crucial tool for big data processing in Python because it provides powerful, flexible data structures like DataFrames that simplify the handling, cleaning, and analysis of large datasets. Its intuitive syntax and broad functionality make it easy to perform complex operations such as filtering, aggregation, merging, and time-series analysis with just a few lines of code.

Pandas efficiently manages memory and integrates seamlessly with other big data tools and formats like SQL databases, CSV files, and JSON, making it essential for both rapid prototyping and scalable data processing tasks.

Questions for reinforcement

1. What statistical approaches to big data analytics do you know?
2. What is the purpose of the Pandas library?
3. How to import data from files into Pandas?
4. How is data imported from the Internet?
5. How to check descriptive statistics of data in Pandas?
6. What tools for correlation analysis in Pandas do you know?
7. What is a heat map used for?

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. Pandas. URL: https://github.com/pandas-dev/pandas
3. The Humanitarian Data Exchange. URL: https://data.humdata.org/
4. Pandas. URL: https://www.w3schools.com /python/pandas/pandas_intro.asp

Lecture 6.
Handling missing data. Converting data types and manipulating date frames in Python

Lecture plan

6.1. Handling missing data.
6.2. Data type conversion.
6.3. Manipulating date frames.

6.1. Handling missing data

An example of a dataset that needs pre-cleaning is a dataset with *NaN (Not-A-Number)* values. NaNs are used to represent data that is undefined or cannot be represented. Pandas refers to missing data as NaN values, which are also often referred to as NA values. NaNs can cause data analysis functions to stop abruptly during calculations, throw errors, or give incorrect results.

NaNs can also be intentionally intended to uniformly represent all pieces of information that are missing from the dataset, or incorrect or null values, or data that simply isn't there. Many datasets have missing data because they weren't collected properly or were missing to begin with. Another common cause of NaNs is re-sampling data in the dataset.

Let's look at an example.

```
import pandas as pd
import numpy as np
# Create a dataframe with random numbers
dataframe = pd.DataFrame(np.random.randn(3,2), index=['a','c','e'],columns=['one','two'])
dataframe
```

	one	two
a	-0.610609	0.761838
c	-0.460771	-0.646487

```
# Reindexing the dataframe creates NaNs
dataframe = dataframe.reindex(['a','b','c','d','e'])
dataframe
```

	one	two
a	-0.610609	0.761838
b	NaN	NaN
c	-0.460771	-0.646487
d	NaN	NaN

Missing values can take different forms based on the data type. Pandas data types are objects/strings, int64/integers, float64/floats, and datatime64/timestamps. NaN is used for undefined strings, integers, and real numbers, and NaT is used for timestamps. There may also be situations where the Python value None will also represent missing data. To make it easier to detect missing values in a dataset, Pandas provides the **isnull()** and **notnull()** functions.

6.2. Data type conversion

Pandas has many built-in functions for converting data types. In the following example, dataset 2 consists of integers, strings, and floats. Let's change column 2 from an object/string data type to a numeric data type.

```
import pandas as pd

data2 = [[22,'2017',0.20],[100,'0.33',1.112],[6,'12',0.33]]
df2 = pd.DataFrame(data2, columns = ['col1','col2','col3'])
df2
```

	col1	col2	col3
0	22	2017	0.200
1	100	0.33	1.112
2	6	12	0.330

```
df2.dtypes
```

```
col1      int64
col2     object
col3    float64
dtype: object
```

The convert_objects function converts column 2 from a string/object to a numeric data type.

```
df2['col2'] = df2['col2'].convert_objects(convert_numeric=True)
df2
```

	col1	col2	col3
0	22	2017.00	0.200
1	100	0.33	1.112
2	6	12.00	0.330

```
df2.dtypes
```

```
col1      int64
col2    float64
col3    float64
dtype: object
```

Column 2 has been converted to type float 64 . This was due to the presence of the string "0.33" in column 2. If the string were "33", the column would be converted to integers. If the string were "x" instead of "0.33", the conversion would result in an error because "x" cannot be converted to a numeric value, integer or real.

In the following example, the data in column 3 is converted from the float64 data type to an object (string) data type. The **dtypes** property is used to verify the data type change.

```
df2['col3'] = df2['col3'].astype(str)
df2
```

	col1	col2	col3
0	22	2017.00	0.2
1	100	0.33	1.112
2	6	12.00	0.33

```
df2.dtypes
```

```
col1      int64
col2    float64
col3     object
dtype: object
```

In the following example, the data in column 1 is converted from the int64 data type to the float64 data type. The **dtypes** property is used to verify the data type change.

```
df2['col1'] = df2['col1'].astype(float)
df2
```

	col1	col2	col3
0	22.0	2017.00	0.2
1	100.0	0.33	1.112
2	6.0	12.00	0.33

```
df2.dtypes
```

```
col1    float64
col2    float64
col3     object
dtype: object
```

6.3. Manipulating date frames

Cleaning a dataset is a preliminary task before performing data analysis. Manipulating a two-dimensional data frame with Pandas in Python can involve removing, adding, or renaming columns or rows of data. This requires calling the **drop()** function, the **loc()** function, and **rename()** function.

To drop a column of data, call the **drop()** function. Let's create a simple dataset. We'll remove and add columns and rows using **the drop()** and **add()** functions.

```
import pandas as pd
data3 = [[.351,.446,.512],[.112,.980,.122],[.216,.612,.575]]
df3 = pd.DataFrame(data3, columns=['one','two','three'])
df3
```

	one	two	three
0	0.351	0.446	0.512
1	0.112	0.980	0.122

We will delete the first column using the **drop()** function.

```
df3.drop(['one'], axis=1, inplace=True)
df3
```

	two	three
0	0.446	0.512
1	0.980	0.122

The axis refers to columns numbered from left to right, starting with the index column at axis=0 and one column at axis=1.

The same result can be achieved using the **del** command.

del df3 ['one']

Now we will delete the first two rows (0,1) with the following function **drop()**, where [0,1] are the rows to be dropped and axis=0 refers to the leftmost index column.

```
df3.drop([0,1], axis=0, inplace=True) df3
```

Let's add a column, assigning it a label and value.

```
df3['four'] =.323
df3
```

To add a row, we can use **location** or **loc()** method. The location method also finds the maximum index or number of the last row and then adds 1 to it, creating row 3.

```
df3.loc[df3.index.max() +1] = [.232,.444,.587]
df3
```

If we call the **loc()** function and pass it an index number, it will be added as a new bottom row. Notice how the added row is numbered 1, even though it is the last row.

```
df3.loc[1] =.763
df3
```

We can change the index by assigning new index values using **indexframe** property of the dataframe.

```
i = [1,2,3]
df3.index = i
df3
```

	two	three	four
1	0.612	0.575	0.323
2	0.232	0.444	0.587

Along with **drop()** and **loc()** functions, Pandas also provides a **rename()** function to rename column labels to "one", "two", and "three" respectively. To do this, we need to use the **rename()** function and assign the columns in the key:value pair with the old name and the new name as a key-value pair.

```
df3.rename(columns = {'two':'one','three':'two','four':'three'}, inplace = True)
df3
```

	one	two	three
1	0.612	0.575	0.323

Pandas has built-in functions for statistical analysis of data sets, including functions for calculating means, standard deviations, and correlations. Let's create a data set and a data frame using the data array data 4. The data frame is displayed on the screen.

```
import pandas as pd
import numpy as np

data4 = [[2,3,5,2,11,3,7,8,10,2,12,7,9,7,4,7,8,12,9,10,6,7]
df4 = pd.DataFrame(data4, columns = ['nums']
df4
```

	Nums
0	2
1	3
2	5
3	4
4	11
5	3
6	7
7	8
8	10
9	2

The average of all the numbers is calculated using the dataframe **mean()** method. The average will be 6.863636. Using the dotted syntax, the result can also be rounded to the nearest integer by appending **round()** method after **mean()**.

```
means = df4.mean()
print(means)
```

```
nums 6.863636
dtype: float64
```

```
means = df4.mean().round()
print(means)
```

```
nums 7.0
dtype: float64
```

```
thesum = df.sum()
print(thesum)

thecount = df4.count()
print(thecount)

themeans = (df4.sum()/df4.count()).round()
print(themeans)
```

```
nums   151
dtype: int64
nums   22
dtype: int64
nums   7.0
dtype: float64
```

If there is an odd number of items in a data set, the median is the middle number, sorted in numerical order. If there is an even number of items in a data set, the median is calculated using the middle two numbers.

```
median = df4.median()
print(means)
```

```
nums 7.0
dtype: float64
```

```
thestands = df4.std()
print(thestands)
```

```
nums 3.196657
dtype: float64
```

The second box shows the **std()** method for calculating the standard deviation. The standard deviation shows the amount of variation in a set of data values.

A low standard deviation indicates that the numbers in the data set tend to be close to the mean. The standard deviation is found by taking the square root of the mean square deviation of the values from the mean, or average, in the data set.

Conclusion to lecture 6

Often, the datasets we work with will have inconsistencies. Data cleaning may involve removing missing or unwanted values or changing the format of values to make them consistent. NaN (not a number) values are used to represent data that is undefined or cannot be represented. Pandas refers to missing data as NaN values. NaT are used for timestamps. Pandas has many built-in functions for converting data types, manipulating date frames, and performing statistical analysis on data sets. Handling missing data, converting data types, and manipulating DataFrames are essential tasks in Python for preparing datasets for analysis, especially when working with real-world big data. Using libraries like Pandas, missing values can be detected and addressed through methods such as filling, interpolation, or removal. Data type conversion ensures that operations are performed correctly and efficiently, for example converting strings to datetime or numerical formats.

Questions for reinforcement

1. How is missing data handled in Pandas?
2. How is data type conversion done?
3. How is data frame manipulation done in Pandas?
4. What are the convert_objects, drop (), loc() and rename() functions used for?

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. Pandas. URL: https://www.w3schools.com/python/pandas/pandas_intro.asp

Lecture 7.
Data regression analysis in Python

Lecture plan

7.1. Machine learning analysis methods and types.
7.2. Regression analysis.
7.3. Types of regression analysis.
7.4. Application of regression analysis.

7.1. Machine learning analysis methods and types

Machine learning addresses the challenges and opportunities presented by Big Data analytics to model existing data to predict future outcomes.

In his book, Kevin Patrick Murphy defines machine learning as "a set of techniques that allow automatically detect patterns in data and then use those patterns to predict future data or to perform other types of decision-making under uncertainty". For example, a computer program is developed by a video service to recommend movies to individual users. The algorithm analyzes movies that viewers have already watched and movies that people with similar viewing preferences have rated highly. The goal is to increase customer satisfaction with the video service.

Machine learning methods are used for a wide range of applications, including speech recognition, medical diagnostics, driving schools, advertising applications with sales recommendations, and many others. Regardless of the application, machine learning algorithms improve their performance on specific tasks based on repeated execution of those tasks, if the algorithm and model are able to cope with the increased variability introduced by additional data. This is the main trigger for the search for better models and algorithms.

Machine learning encompasses many different algorithms, some with a broad range of applicability, while others may be suitable for specific applications. These algorithms can be divided into two main categories: supervised and unsupervised. Supervised machine learning algorithms are the most commonly used machine learning algorithms for predictive analytics. These algorithms rely on datasets that

have been processed by human experts (hence the word "supervised"). The algorithms then learn how to independently perform the same processing tasks on new datasets. In particular, supervised methods are used to solve regression and classification problems. Regression problems are about evaluating mathematical relationships between continuous and discrete variables. This mathematical relationship can be used to calculate the values of one unknown variable given the known values of others. Examples of regression include estimating the position and speed of a car using GPS, predicting the path of a tornado using weather data, or predicting the future value of a stock using historical data and other sources of information. To visualize the simplest example of regression, imagine two variables whose values are visualized as points on a two-dimensional graph, similar to Fig.7.1.

Performing regression means finding a line that best fits the values. The line can take many forms and is expressed as a regression function. A regression function allows estimate the value of one variable given the value of the other for values that have not been observed before.

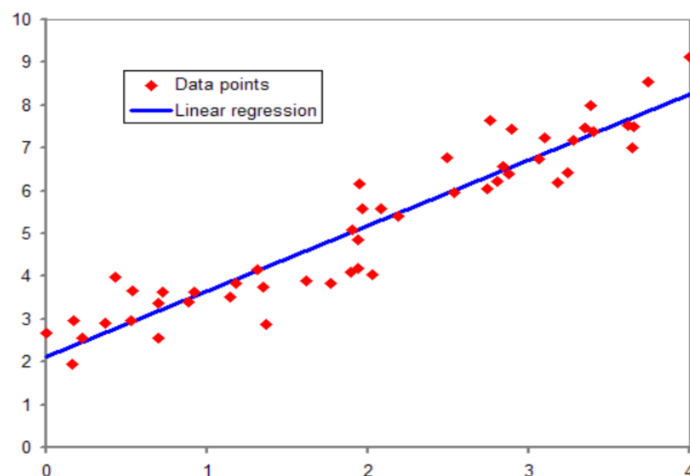


Fig. 7.1. Graphical regression model [1]

Classification problems are used when the unknown variable is discrete. Typically, the problem is to estimate which of a set of predefined classes a particular sample belongs to. Typical examples of classification are image recognition or diagnosing pathologies using medical tests or detecting faces in an image.

A visual interpretation of the classification problem can be seen in two dimensions, where points belonging to different classes are marked with a different symbol, similar to the image in Fig. 7.2. The algorithm "learns" examples of locations and shapes at the boundary line between classes.

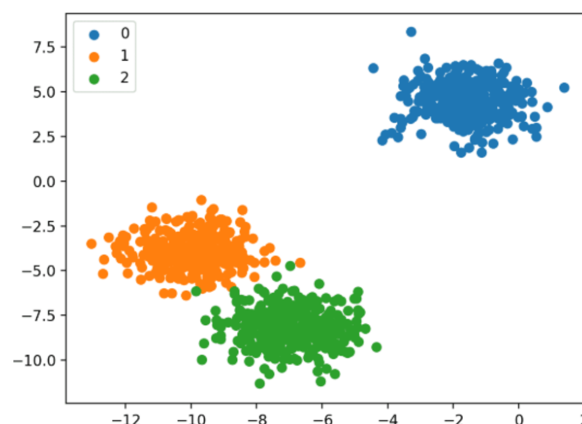


Fig. 7.2. Scatter plot of multi-class classification dataset [1]

7.2. Regression analysis

Regression analysis is one of the oldest and most commonly used statistical methods of data analysis. The basic idea of regression is to quantify the mathematical relationship between one or more independent (also called predictor) variable(s) and a dependent variable (also called target). Regression analysis relies on a data set of observed predictors and target values. The relationship, or regression function, can be used to estimate the values of the dependent variable outside the range of observed values. In other words, a regression model allows the analyst to extrapolate beyond the available data set.

For example, when working with time series data, regression allows analysts to predict future values from historical data. Regression looks for relationships between any type of continuous variable. Specifically, it attempts to answer the general question: "by how much will variable V_1 change if variable(s) V_2 (V_3 , V_4 , V_5) change by an amount X?" A simple way to visualize the regression function is imagine a set of points in two dimensions (Fig. 7.3).

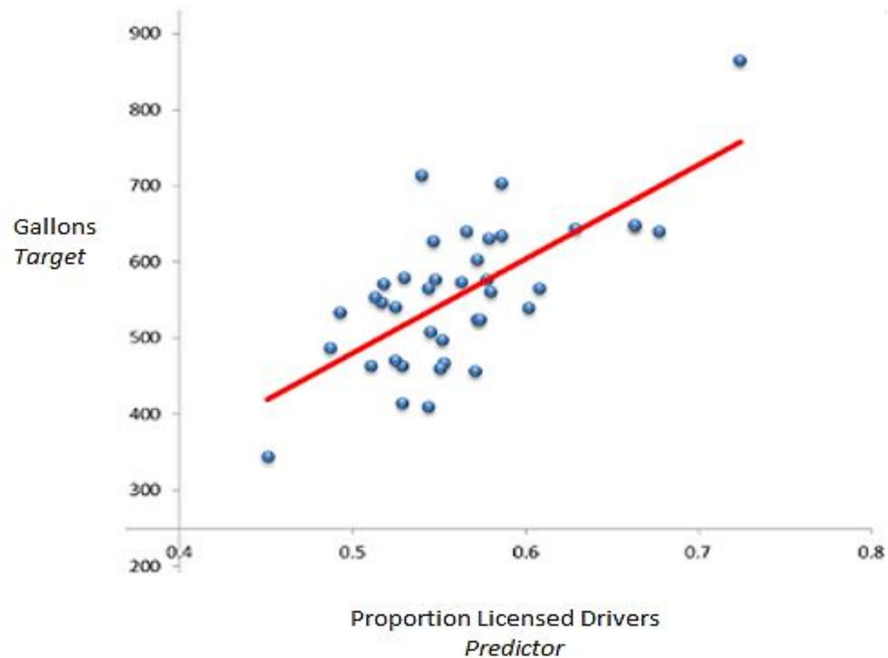


Fig. 7.3. Gasoline consumption and percentage of licensed drivers [1]

The conditional predictor variable plotted on the X-axis is the proportion of licensed drivers in different geographic areas. The Y-axis uses the values for the target variable, the corresponding gasoline consumption. In this case, the possible regression function is represented by the red line. The fact that it is a straight line in this example suggests a intuitive result: an increase in licensed drivers in this area will cause a proportional increase in gasoline consumption. Although a simple visual inspection of the distribution of data points suggests that the line is the best fit, regression does not constrain the shape of the regression function.

7.3. Types of regression analysis

The most common type of regression is linear regression. They are the simplest from both a computational and mathematical perspective; and therefore represent the first option for the data analyst presented with a regression problem. Despite the name, linear regression does not involve fitting a line through the data points. The term linear means that the regression function will always attempt to fit the data using a weighted average of the other functions, whether linear or not. The property of linearity simplifies the calculation of the parameters of a regression model, while

allowing the use of almost any shape to fit the observations. The simplest case of linear regression consists of fitting a straight line. This is also called a simple linear model, as shown in Fig. 7.4.

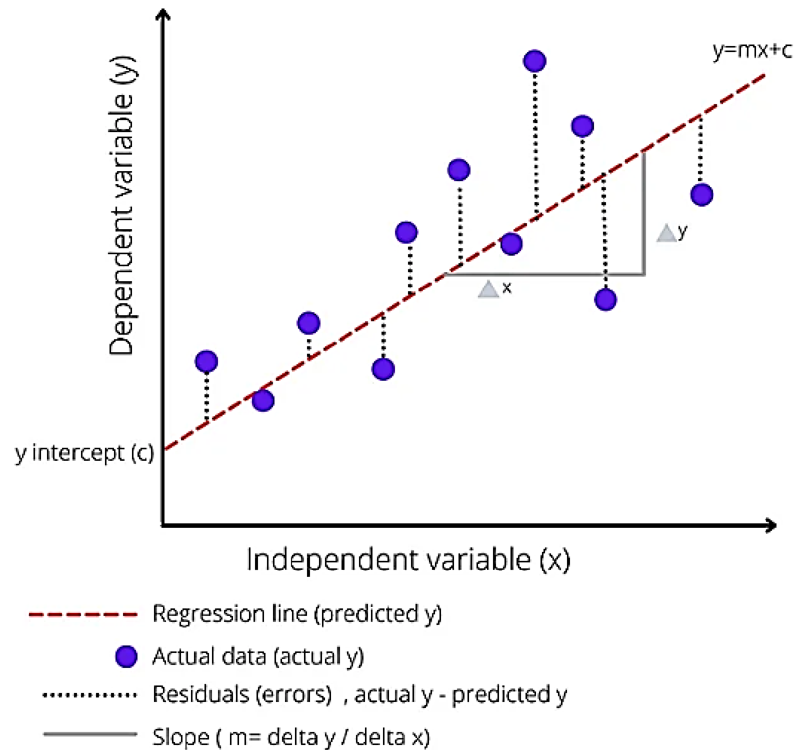
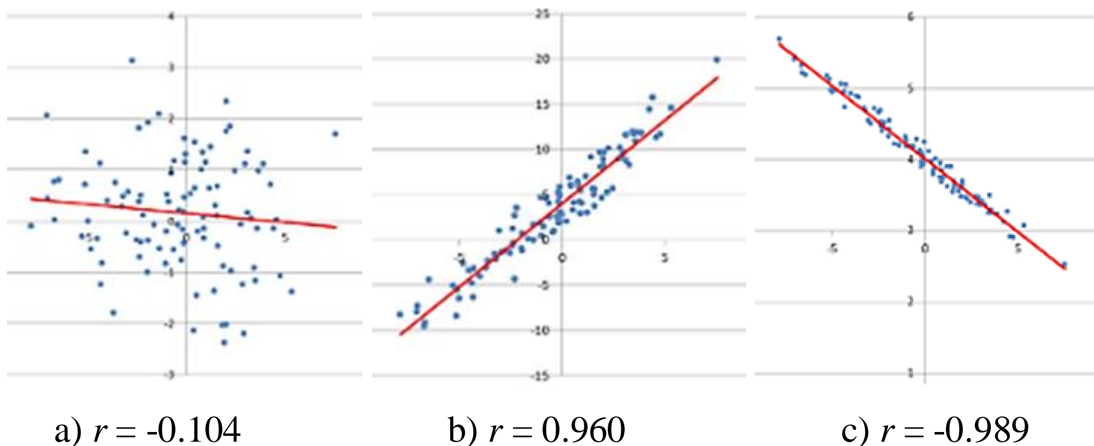


Fig. 7.4. Example of linear regression [1]

A high Pearson correlation indicates that a simple linear model is a good candidate for fitting the data (Fig. 7.5).



a) r = -0.104 b) r = 0.960 c) r = -0.989

Fig. 7.5. Examples of no correlation (a), strong positive (b) and negative correlated observations (c) [1]

The regression process in this case consists of finding the slope and intercept of the line that minimizes the sum of the distances between the line and all the data points, as shown in Fig. 7.6.

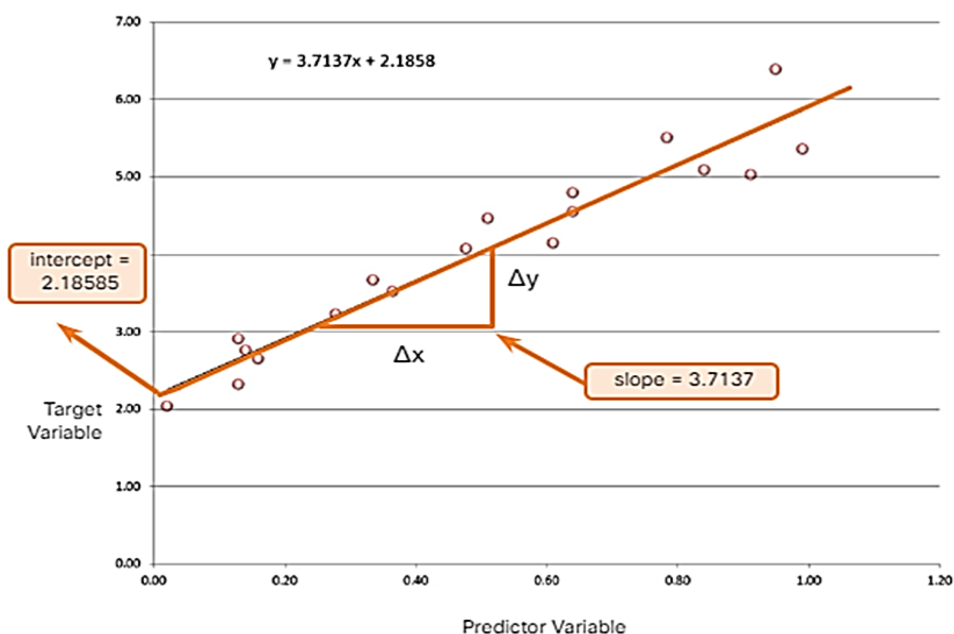


Fig. 7.6. Finding the slope and Y -intercept of a line [1]

When using linear models, the most common algorithm used to estimate these optimal model parameters is the method least squares (LS).

In Fig. 7.7 we see three data sets, each with one target and one predictor variable. In all three cases, we can see how, despite the noise affecting the observations, there is a clear line that captures the underlying relationships between the variables. The red line represents a linear regression model that minimizes the distance from all observations. The models were obtained using linear regression.

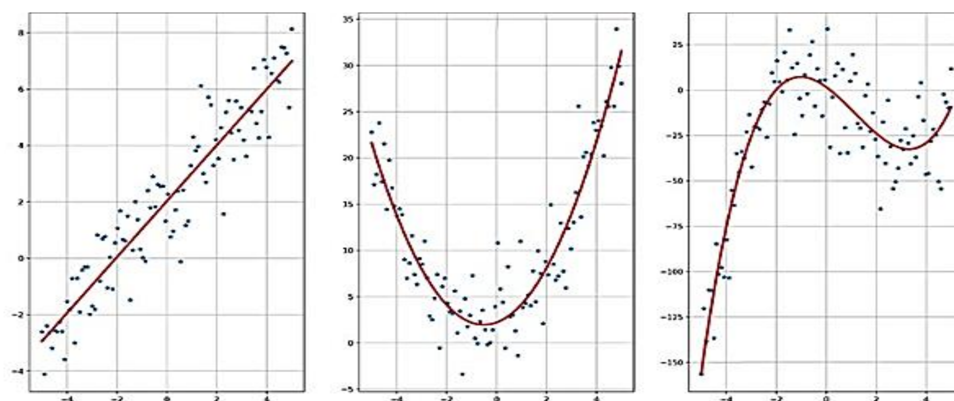


Fig. 7.7. Linear and polynomial regression models [1]

Consider an example of multiple regression where we have more than one independent variable. Suppose the data set contains some information about cars (Fig. 7.8).

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99

Fig. 7.8. Fragment of a dataset about different cars
(model information, engine capacity, car weight, CO2 emissions) [5]

We can predict the CO2 emissions of a car based on the engine size and mass of the car to make the prediction more accurate. To do this, we import the Pandas module, which allows us to read CSV files (in our case, the file cars.csv) and return a DataFrame object. We will create a list of independent values and call this variable *X*. We put the dependent values in a variable called *y*. We will use some methods from the sklearn module, so we will also need to import this module:

```
from sklearn import linear_model
```

From the sklearn module, we will use the **LinearRegression()** method to create a linear regression object. This object has a method called **fit()** that takes independent and dependent values as parameters and populates the regression object with data describing the relationship:

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Now we have a regression object ready to predict CO_2 values based on the weight and volume of the car:

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm^3:

predictedCO2 = regr.predict([[2300, 1300]])

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("cars.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm³:
predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

The result of the program is the number 107.2087328. That is, we predicted that a car with a 1.3-liter engine and a mass of 2300 kg would emit approximately 107 grams of CO2 for each kilometer traveled.

The next step is to find the coefficients of our regression. In this case, we can find the coefficient of the car's mass vs. CO2, as well as the engine size vs. CO2. The answer we get tells us what would happen if we increased or decreased one of the independent values. To do this, we use the following code:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("cars.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

print(regr.coef_)
```

Result of program execution:

```
[0,00755095 0,00780526]
```

The result array represents the coefficient values for weight and volume.

These values tell us that if the mass increases by 1 kg, CO2 emissions increase by 0.00755095 g. And if the engine size (volume) increases by 1 cm^3, CO2 emissions increase by 0.00780526 g.

We have already predicted that if a car with a 1300 cm3 engine weighs 2300 kg, CO2 emissions will be approximately 107 g.

Consider a situation where we increase the mass of the car by 1000 kg (change the weight from 2300 to 3300 kg).

To do this, we will use the following code:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("cars.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[3300, 1300]])

print(predictedCO2)
```

Result of program execution:

```
[114,75968007]
```

We predicted that a car with a 1.3-liter engine and a mass of 3300 kg would emit approximately 115 grams of CO2 for every kilometer traveled.

Which shows that the coefficient 0.00755095 is correct:

107.2087328 + (1000 * 0.00755095) = 114.75968 [5].

7.4. Application of regression analysis

Regression analysis has many applications. It is often used in business and financial analysis with historical data to inform future strategies. It can be used to predict trends in the economy and can inform policy actions to guide economic growth. It can also be used to predict customer behavior to distinguish normal from fraudulent behavior in the insurance and consumer credit industries.

In healthcare, multiple regression can be used to assess which of a number of variables may influence a target variable. The relationship between a group of lifestyle choices, such as smoking, exercise, and dietary habits, can be analyzed to determine how they affect a health variable, such as blood pressure, diabetes, or even life expectancy. Regardless of the application, any machine learning model requires validation. Some models are sensitive to external influences or data anomalies. Other models may generate results that may not be suitable for answering the research question. In real estate, regression analysis is commonly used to estimate property values. By examining variables such as location, number of bedrooms, square footage, proximity to schools or public transport, and recent sale prices of nearby properties, regression models can help appraisers and buyers determine fair market values. These models are especially useful for detecting over- or underpriced listings, supporting better decision-making in dynamic housing markets.

Scientists use regression analysis to understand how variables like carbon dioxide levels, temperature, and sea ice coverage interact over time. Linear regression can estimate the rate of global temperature rise based on historical emission data, helping predict future climate patterns and assess the potential impact of mitigation strategies. In manufacturing, regression models are used to optimize production processes and improve quality control. For example, a factory might use regression to understand how machine settings, raw material quality, and ambient conditions affect product defect rates. By identifying the most influential factors, companies can adjust their operations to minimize waste and ensure consistent output, boosting efficiency and reducing costs.

Conclusion to lecture 7

Regression analysis is one of the most common statistical methods of data analysis. The main purpose of regression is to determine the mathematical relationship between one or more independent variables and a dependent, or target, variable. Linear regressions are the simplest, both computationally and mathematically. The term " linear " means that the regression function will always try to fit the data using a weighted average of other functions, regardless of whether those functions are linear or not. Regression analysis is often used in financial and business analysis to formulate strategies for further action. It can be used to predict economic trends and to manage economic growth.

Questions for reinforcement

1. What methods and types of machine learning analysis do you know?
2. What is regression analysis used for?
3. What types of regression analysis do you know?
4. What real-life applications of regression analysis do you know? Give examples.

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. Machine Learning. URL: https://www.w3schools.com/python/python_ml_getting_started.asp
3. Linear Regression. URL: https://www.w3schools.com/python/python_ml_linear_regression.asp
4. Polynomial Regression. URL: https://www.w3schools.com/python/python_ml_polynomial_regression.asp
5. Multiple Regression. URL: https://www.w3schools.com/python/python_ml_multiple_regression.asp

Lecture 8.
Errors in data analysis and predictive analytics. Estimating regression errors using Python.
Purpose of the scikit-learn library

Lecture plan

8.1 Errors in data analysis and predictive analytics.
8.2. Estimating regression errors using Python.
8.3. Purpose of the scikit-learn library.

8.1 Errors in data analysis and predictive analytics

Errors and uncertainties affect the data analysis process at different levels. The first type of error is *measurement error*. Data needs to be cleaned because the values of variables can be corrupted by noise. But where does this noise come from? Often, the error is caused by the sensor, or by a person reading or using the sensor. Any measuring device is limited in its accuracy. Therefore, all measurements have a built-in error component. A device will always have a built-in error.

For example, a measuring tape used to mark the line for cutting plywood into pieces has a built-in measurement error. An error of a few millimeters will not affect the effectiveness of storm shutters.

For example, more accuracy is required when cutting materials for kitchen cabinets. Due to measurement error, the true value cannot be known, but this error can be studied statistically and accounted for and is defined as the difference between the true value (which is unknown) and the measured one.

Another type of error is *prediction error.* In supervised learning, prediction error is quantified as the difference between the value predicted by the model and the observed value. The observed value is affected by measurement error, and although it cannot be eliminated, there are techniques such as cross-validation to limit its effects.

Gross errors – caused by an error in the instrument used to make the measurement or in the recording of the measurement result. For example, an observer records 1.10 instead of the actual measurement of 1.01.

Random errors – caused by factors that randomly affect the measurement data. For example, a calibrated scale at the grocery store may have an error of plus or minus 1 gram each time weigh the same item.

Systematic errors – caused by instrumental or environmental factors that affect all measurements taken over a period of time. For example, an uncalibrated scale will generate a systematic error every time a measurement is taken. Random errors tend to follow a normal distribution around the mean observation (Fig. 8.1). A statistical model of the error can be constructed, in which case regression algorithms can easily account for it. For some methods, the fact that the error follows a normal distribution is a requirement.

Systematic errors shift the distribution of observations (Fig. 8.1) in one direction or another. Therefore, systematic error is more difficult to eliminate, since the true value is unknown, the only way to detect systematic error is to use another measurement system that we consider more reliable.

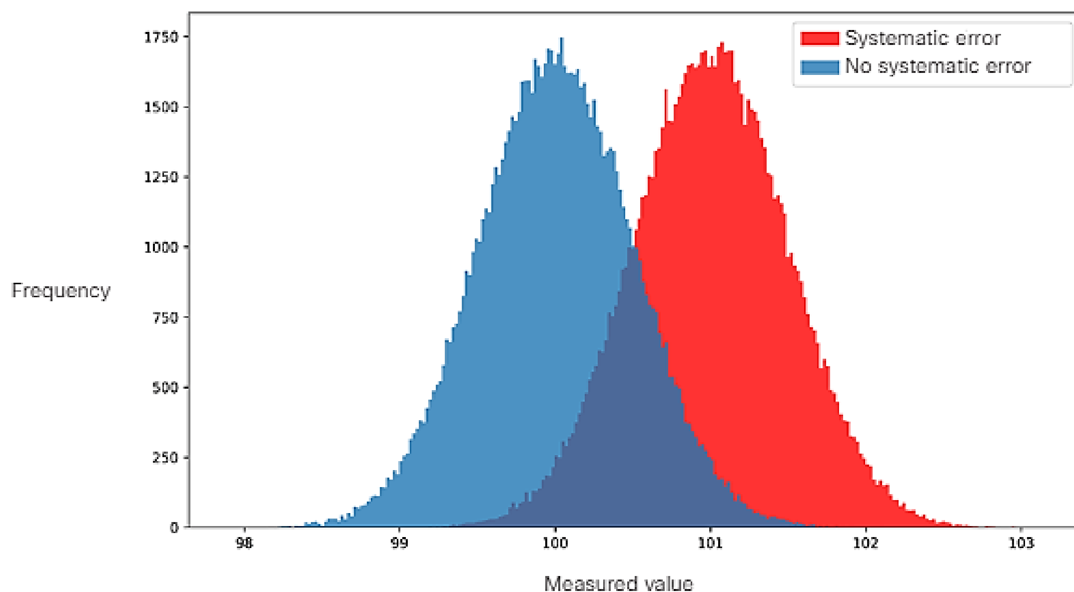


Fig. 8.1. Distribution of observations for systematic and random errors [1]

Prediction error is the difference between the value predicted by a regression or classification model and the measured value.

The prediction error is the distance between the regression function and the data points. Specifically, it is common to estimate the prediction error using the average of the sum of the squared distances for all points.

In regression, the error on the training set is smaller than the error on the validation set. The prediction error has two components. The first component is due to the choice of model. Regardless of the algorithm, whenever we approach a regression or classification model, we make assumptions about how the data is distributed, which are inevitably approximations. For example, we might fit a model that is a good approximation only for a given range of samples, but fails to capture the relationship outside of it. The motto of data analysts is: "Every model is wrong, but some of them are useful." Fig. 8.2 shows the difference between a second-order polynomial model and a third - order model. Fig. 8.3 shows the difference between a third-order model and a second-order polynomial model.

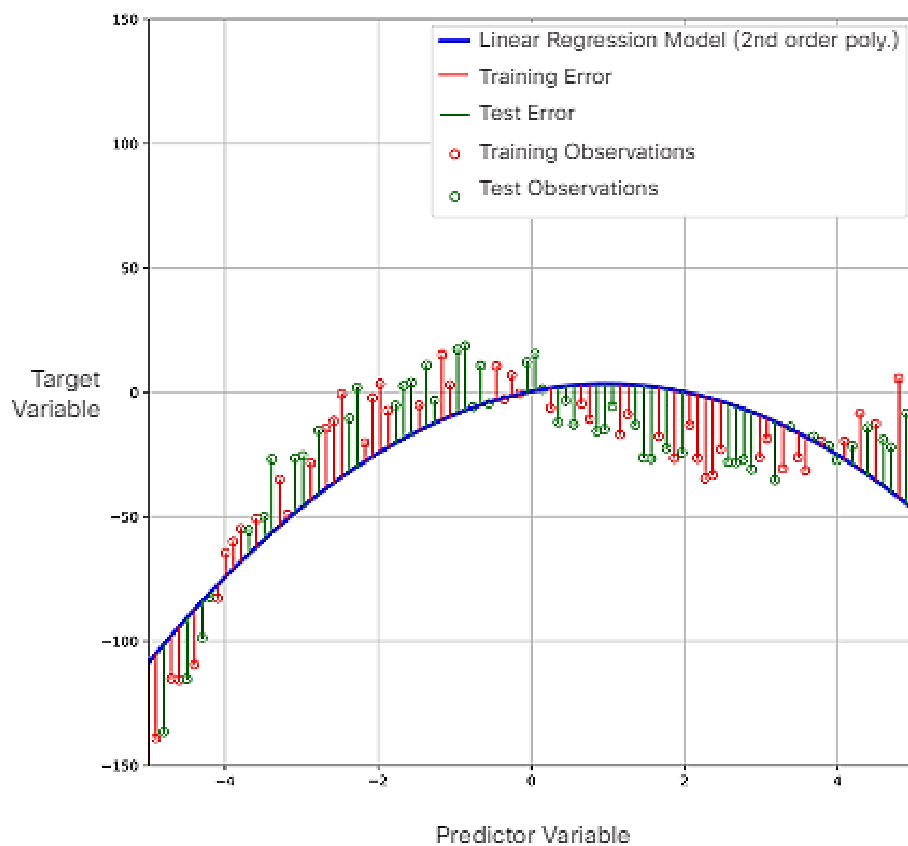


Fig. 8.2. Second-order linear regression model [1]

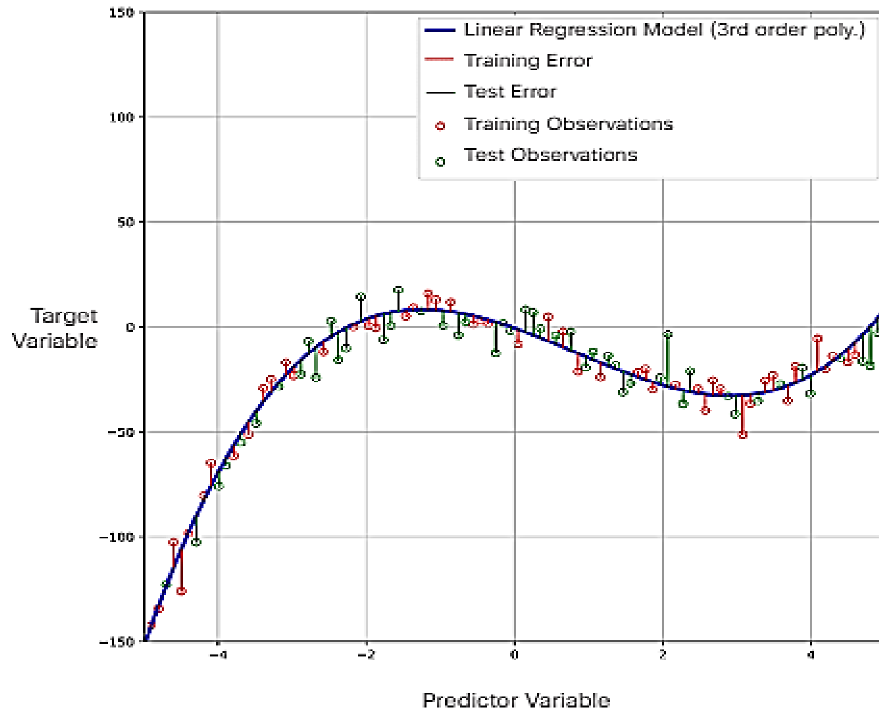


Fig. 8.3. Third- order linear regression model [1]

The third-order model definitely performs better in terms of errors (Fig. 8.4).

Fig. 8.4. Forecast errors for linear regression models of the second and thirdly, in order [1]

Even when the chosen model perfectly reflects the true distribution, there will still be a difference between the predicted and actual values due to measurement

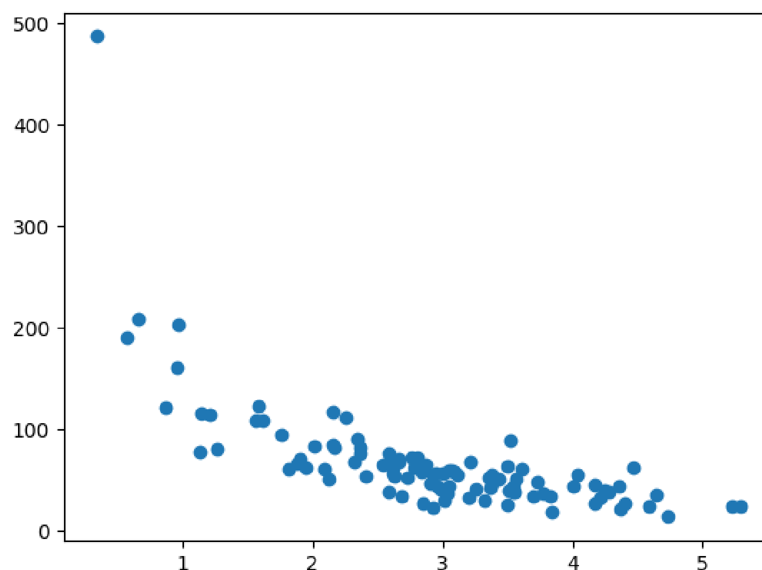
error. This cannot be eliminated; therefore, measurement error affects the regression model.

8.2. Estimating regression errors using Python

Measuring the accuracy of the model is called Train/Test because we divide the data into two sets: a training set and a testing set (80 % of the data for training and 20% for testing). We train the model using the training set and validate the model using the testing set. To train a model means to create a model. Model validation means checking the accuracy of the model. Our dataset illustrates 100 shoppers in a store and their shopping habits based on their time spent in the store.

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

plt.scatter(x, y)
plt.show()
```

Result:



The x-axis represents the number of minutes until the purchase is made.

The y-axis represents the amount of money spent on the purchase.

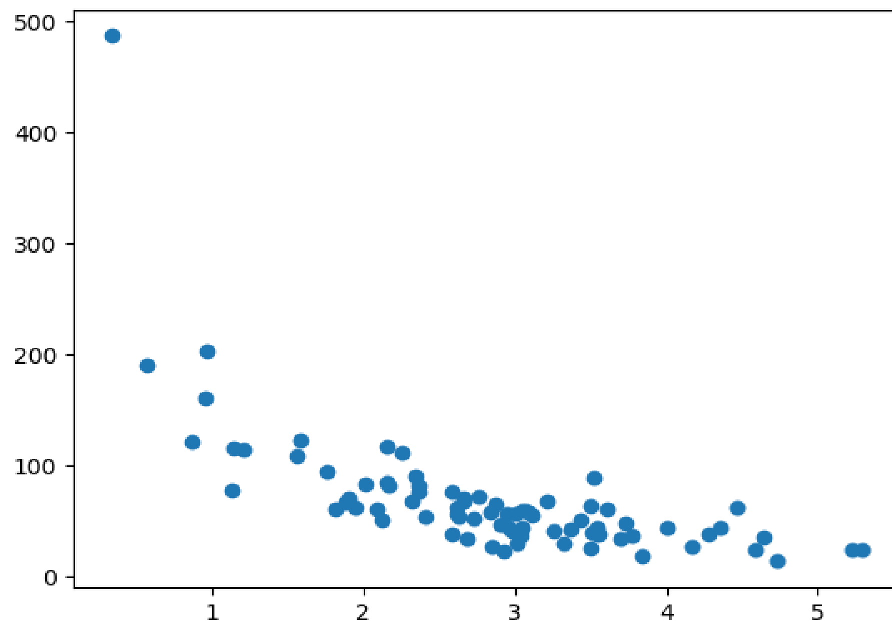
The training set should be a random selection of 80% of the original data. The testing set should be the remaining 20% .

```
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]
```

Let's display the same scatter plot with the training set:

```
plt.scatter(train_x, train_y)
plt.show()
```

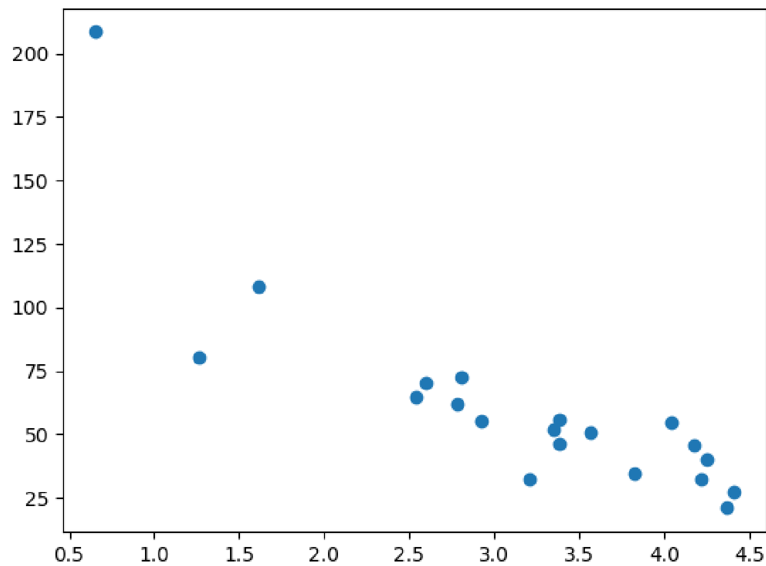
Result:



To make sure the testing kit isn't completely different, we'll also take a look at the testing kit.

```
plt.scatter(test_x, test_y)
plt.show()
```

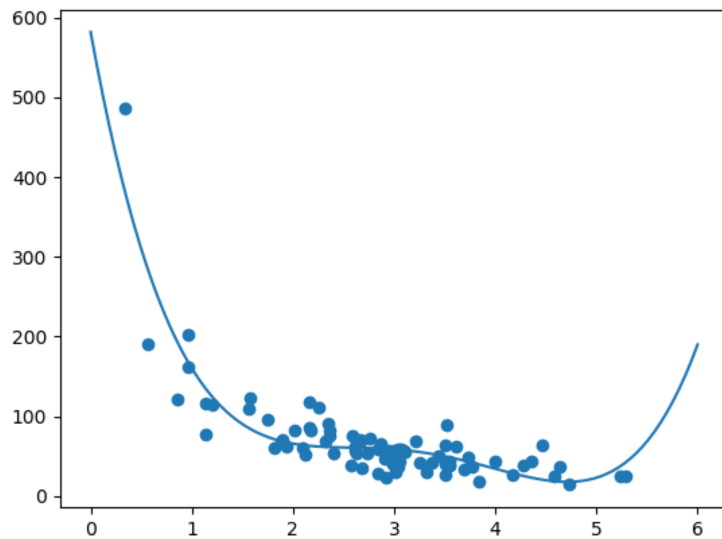
Result: The test set also looks like the original data set:



Polynomial regression is likely the best option, so let's draw a polynomial regression line through the data points. To draw a line through the data points, we use **plot()** method **matplotlib** module:

```
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

myline = numpy.linspace(0, 6, 100)

plt.scatter(train_x, train_y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:



The result may help we can predict values outside the data set. Example: a line indicates that a customer who spends 6 minutes in a store is likely to make a purchase worth 200 $. The R-squared score is a good indicator of how well a data set fits the model. R-squared measures the relationship between the x-axis and the y-axis, and the value ranges from 0 to 1, where 0 means no relationship and 1 means completely interconnected. The **sklearn** module contains **the r2_score()** method, which will help us find this relationship. We measure the relationship between the minutes a customer spends in the store and the amount of money spent. Let's check how well the training data fits the polynomial regression.

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(train_y, mymodel(train_x))

print(r2)
```

The result of 0.799 shows that there is a fairly good relationship [2]. Now we have created a model that is normal, at least when it comes to training data. Now we want to test the model with the test data to see if it gives us the same result.

Let's find the R2 estimate using the testing data:

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

r2 = r2_score(test_y, mymodel(test_x))

print(r2)
```

The result of 0.809 shows that the model fits the test set, and we are confident that we can use the model to predict future values. Now that we have established that everything is fine with our model, we can start predicting new values. We will determine how much money a customer will spend if he or she stays in the store for 5 minutes, for this we use the command **print(mymodel(5))**:

```
import numpy
from sklearn.metrics import r2_score
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel = numpy.poly1d(numpy.polyfit(train_x, train_y, 4))

print(mymodel(5))
```

Result:

```
22.87962591812061
```

We predicted that the customer might spend $ 22.88, which corresponds to the corresponding point on the chart:

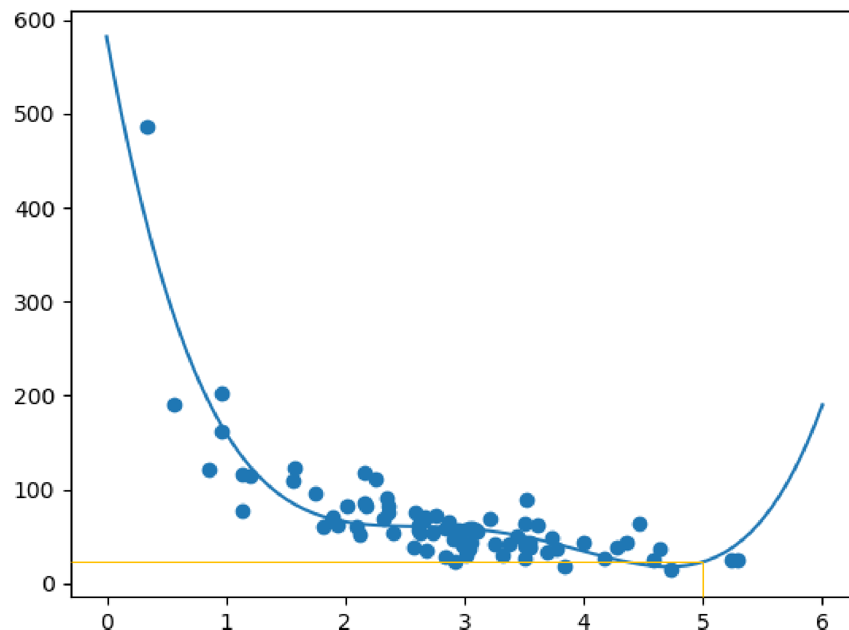


Fig. 8.4. Predicting customer spending relative to store time [2]

8.3. Purpose of the scikit-learn library

Scikit-learn is a free machine learning library for the Python programming language. It has various classification, regression, and clustering algorithms, including SVM (support vector machines), trees solutions, k-means, and DBSCAN (Density-Based Spatial Clustering of Applications with Noise), and is designed to interface with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn is about learning information from one or more datasets that are represented as 2D arrays. They can be thought of as a list of multidimensional observations.

The main API implemented by scikit-learn is the estimator. An estimator is any object that studies data; it can be a classification, regression, or clustering algorithm, or a transformer that extracts/filters useful features from raw data. All estimator objects have a fitting method that uses a data set (usually a 2D array) [3].

Conclusion to lecture 8

Errors and uncertainties affect the data analysis process at different levels. The first type of error is measurement error. Another type of error is prediction error . In supervised learning, prediction error is quantified as the difference between the value predicted by the model and the observed value. Gross errors – caused by an error in the instrument used for measurement or in the recording of the measurement result. Random errors are caused by factors that randomly affect measurements in a data sample. Systematic errors – caused by instrumental or environmental factors that affect all measurements taken over a period of time. Random errors tend to produce a normal distribution around the mean of the observation.

The prediction error is the distance between the regression function and the data points. The prediction error has two components. The first component is due to the choice of model, we make assumptions about how the data is distributed, which is inevitably an approximation. Even when the chosen model perfectly reflects the true distribution, there will still be differences between predicted and actual values due to measurement error.

Questions for reinforcement

1. What types of errors in data analysis and predictive analytics do you know?
2. How is regression error estimation done using Python?
3. What is the purpose and capabilities of the scikit-learn library?

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. Machine Learning - Train/Test. URL: https://www.w3schools.com/python/python_ml_train_test.asp
3. An introduction to machine learning with scikit-learn. URL: https://scikit-learn.org/stable/tutorial/basic/tutorial.html

Lecture 9.
Data classification algorithms. Applications and problems of classifications. Decision tree classifier model

Lecture plan

9.1. Classification problems.
9.2. Classification algorithms.
9.3. Visualization of classifications.
9.4. Application and validation of classifications.

9.1. Classification problems

Classification is a common machine learning problem that falls under the category of supervised learning. Significant improvements have been made over the past decade, especially in the field of image recognition. Classification can be viewed as a regression problem where the target variable is discrete and represents the class into which a human expert classifies a data sample.

In classification problems, it is common to provide not only a set of example data points of each class, but also to establish which features of each data point are more useful for estimating the corresponding class. These features may be available from sensors, but more often they need to be computed (or extracted) from the original data before being fed to the training algorithm.

Determining the appropriate features is a crucial step that, except for advanced algorithms such as deep learning, relies on the knowledge of human experts. For example, a travel company is interested in providing a reliability rating for the flights it finds for customers. Through trial and error of different models, it has been determined which variables among all the data sets are most relevant for the classifications. These are also known as the variables with the highest discriminant power. Only these relevant features are extracted from the data and used to train the classifier. The company decided to use the classifier to predict which flights are most likely to belong to the groups of flights that are on time, delayed, or canceled. Through trial and error of different models, it was determined which variables from

all the variables in the data set are most relevant for the classifications (also said to have the highest discriminant power). Only these relevant features are extracted from the data and used to train the classifier. The reliability rating is intended to communicate the degree of probability that a flight will be on time, delayed, or canceled. The travel company has access to a large amount of data about different airlines, flights, origins and destinations, flight status, and other information.

9.2. Classification algorithms

Consider some classifier algorithms that are popular for various purposes.

The k-nearest neighbors algorithm (k -NN) – perhaps the simplest classifier, which uses the distance between training samples as a measure of similarity.

To visualize how a k-NN classifier works, imagine that each sample has two features, for which the values can be represented on a two-dimensional graph.

In Fig. 9.1, the data points of each class are labeled with different symbols.

The distance between the points represents the difference between the feature values. Given a new data point, the k-NN classifier will consider the nearest training points. The predicted class for the new point will be the most common class among the k neighbors.

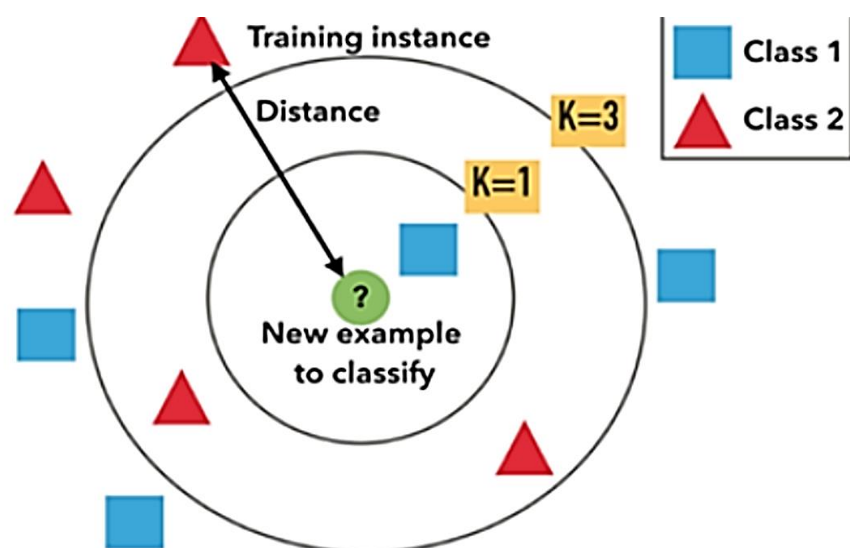


Fig. 9.1. The k-nearest neighbors (k-NN) method [1]

Support vector machines (SVMs) are an example of supervised machine learning classifiers. Instead of basing category membership on the distance from other points, support vector machines calculate a boundary or hyperplane that best separates groups. In Fig. 9.2, H3 is the hyperplane that maximizes the distance between the training points of the two classes (marked with color or black and white). When a new data point is presented, it will be classified based on whether it lies on one side or the other of H3.

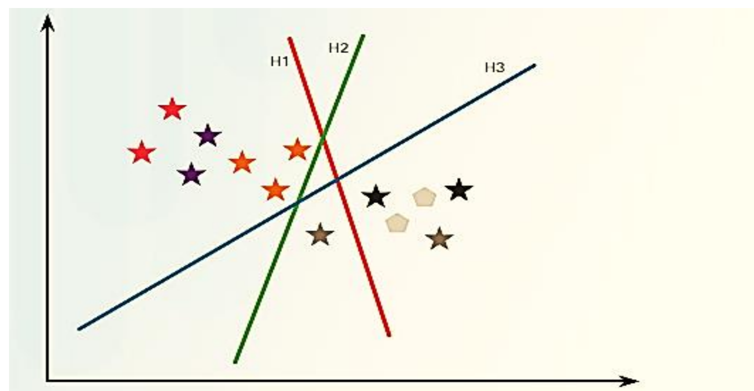


Fig. 9.2. Support vector machines method visualization [1]

Decision trees represent a classification problem as a set of decisions based on feature values. Each node in the tree represents a threshold over the feature value and splits the training samples into two smaller sets (Fig. 9.3).

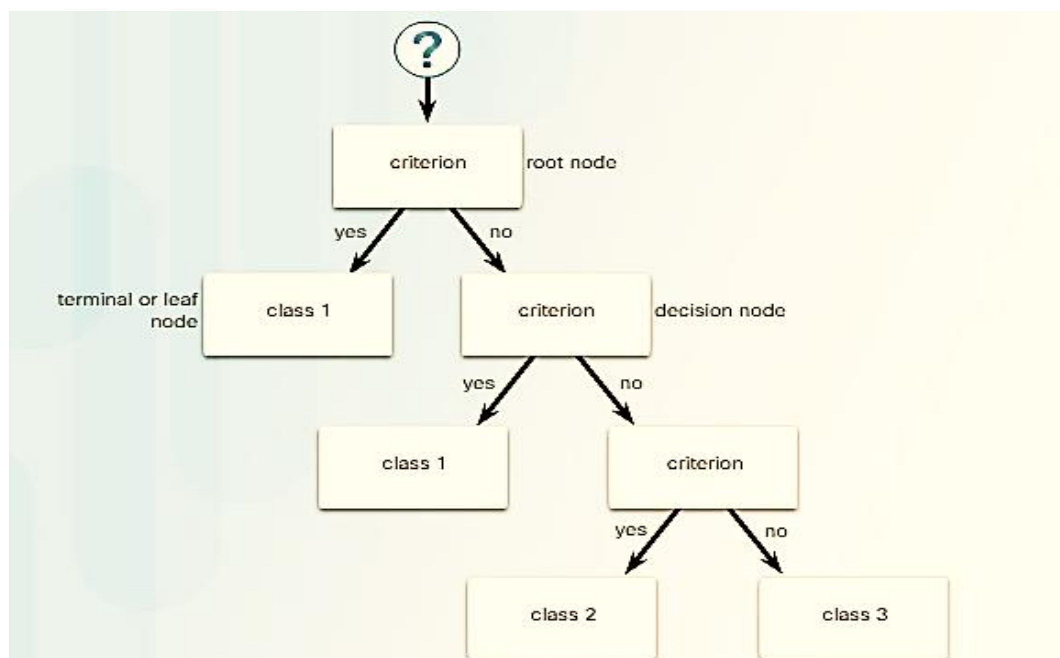


Fig. 9.3. Simplified view of binary decision tree and node types [1]

The decision process is repeated over all features, growing the tree until the optimal way to split the samples is calculated. The classification of a new sample can be obtained by following the branches of the tree based on the values of its features.

9.3. Visualization of classifications

Different types of visualizations enhance the exploratory use of classification algorithms.

Visualizing the K-Nearest Neighbors (KNN) algorithm in Python is a great way to understand how this supervised learning method works and how it makes predictions. In essence, visualizing KNN involves plotting the decision boundaries that the algorithm creates based on the number of nearest neighbors (K) it considers. Here's a straightforward guide on how to do this. Visualizing helps in understanding how changing the value of K affects the decision boundaries and the accuracy of the model as well as help in identifying outliers or data skew.

To visualize KNN in Python, we follow next steps:

```
# Import necessary libraries
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets, neighbors
from sklearn.model_selection import train_test_split

# Step 1: Generate a synthetic dataset with 2 features and 4 centers (clusters)
X, y = datasets.make_blobs(n_samples=500, n_features=2, centers=4,
cluster_std=1.5, random_state=4)

# Step 2: Split the dataset into training and testing sets (80% training, 20%
testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 3: Initialize the KNN classifier with 5 neighbors
knn = neighbors.KNeighborsClassifier(n_neighbors=5)

# Step 4: Train the KNN model using the training data
knn.fit(X_train, y_train)

# Step 5: Visualize the decision regions of the trained KNN model
from mlxtend.plotting import plot_decision_regions
```

```
plot_decision_regions(X, y, clf=knn, legend=2)

# Step 6: Label the axes and add a title to the plot
plt.xlabel('X')
plt.ylabel('Y')
plt.title('KNN with K=5')

# Step 7: Save the plot as an image file with tight bounding box and high
resolution (150 dpi)
plt.savefig('KNN with K=5.jpeg', bbox_inches="tight", dpi=150)
# Step 8: Display the plot
plt.show()
```

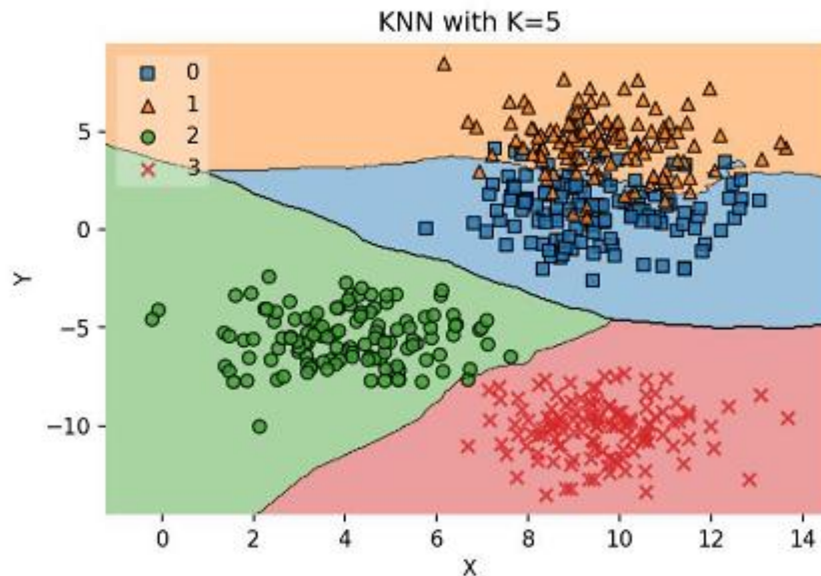


Fig. 9.4. Visualization for k-NN analysis [1]

Fig. 9.5 shows a simple visualization of a decision tree for a classifier built to predict which passengers on the sunken cruise ship Titanic would survive or perish.

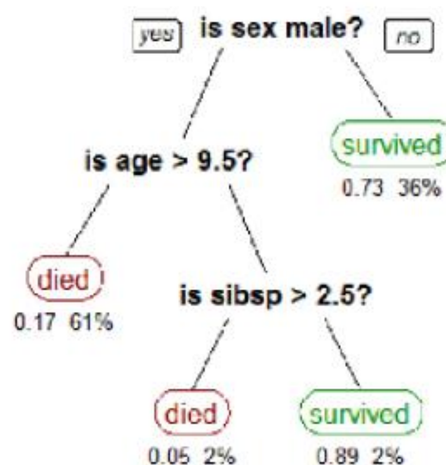


Fig. 9.5. Visualization of a simple binary decision tree [1]

The nodes of the decision tree include a probability measure and the percentage of passengers that each node represents. This decision tree is useful in identifying factors, such as gender, that had the greatest impact on survival. This system could be fed with fictional passengers, and it would classify new passengers accurately based on their survival outcomes.

Fig. 9.6 shows a three-dimensional plot of the SVM method. In this case, the hyperplane separating the two groups is obtained from the three variables for each observation. There is a small amount of error, as shown by the data points appearing on the wrong side of the hyperplane.

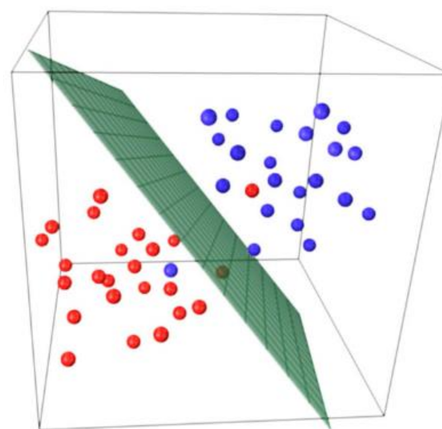


Fig. 9.6. Visualization of the support vector method in 3 dimensions data space [1]

9.4. Application and validation of classifications

Let us consider examples of the application of classification algorithms.

Risk assessment – classification systems can be used to determine which of many factors contributes to the likelihood of different risks.

For example, a number of factors can be used to classify car insurance users into low, medium, and high risk categories and adjust the premiums paid by drivers according to the level of risk.

In risk assessment, a classification model can be trained using historical car insurance data to categorize new applicants into low, medium, or high risk groups. Variables such as age, driving history, vehicle type, location, and previous claims can be used as input features. Based on these, the system predicts the most probable

risk category for each applicant, allowing insurers to tailor policy premiums accordingly and minimize exposure to potential losses.

Medical diagnostics – classification systems can use guided questions to build decision trees that can help diagnose various diseases and disease risks.

Machine learning classification systems can also perform preliminary analysis of large numbers of diagnostic images and identify suspicious conditions for review by doctors.

In medical diagnostics, classification algorithms can be used to construct decision trees that help guide medical professionals through a sequence of diagnostic questions. For instance, a system may start with general symptoms such as fever or fatigue and gradually narrow down potential diagnoses like influenza, COVID-19, or pneumonia. Deep learning models trained on thousands of labeled X-ray or MRI scans can automatically flag potentially abnormal images for further evaluation by radiologists, reducing diagnostic workload and enhancing early detection.

Image recognition – for example, for handwriting recognition, the system might work on the task of identifying handwritten digits.

In image recognition, especially in handwriting analysis, classification models are trained to distinguish between different digits from 0 to 9. Using datasets like MNIST, the model learns from thousands of labeled handwritten samples to recognize key features of each digit, such as curves or line intersections. Once trained, the model can accurately predict which number a new handwritten input represents, enabling applications such as digitizing handwritten forms or enabling handwriting input on mobile devices.

Scientific discoveries often come from using the scientific method, which includes the following steps.

Step 1. Ask questions about the observation, such as what, when, how, or why?

Step 2. Research.

Step 3. Form a research hypothesis.

Step 4. Test the hypothesis through experiments.

Step 5. Analyze the experimental data to draw a conclusion.

Step 6. Communicate the results of the process.

Issues of validity and reliability are fundamental to supporting the results claimed by any experiment or study.

Researchers distinguish four types of reliability.

Inter-rater reliability – how similar are different people's ratings on the same test?

Test-retest reliability – how much variation is there between scores for the same person taking the test multiple times?

Parallel form reliability – how similar are the results of two different tests constructed from the same content?

Internal consistency reliability – what is the difference in scores for different subjects on the same test?

In data analytics, repeating an experiment can be too expensive or even impossible. An example is the image classification system implemented by Facebook, which allows users to search for images using text descriptions. In developing such a solution, Facebook data analysts have access to millions of images with text descriptions provided by human experts. They can fine-tune their classification algorithm to improve its performance, but they cannot know how it will behave with new images that users post in the future, and they cannot assess whether it will give the correct answer or not. So how can they be sure that the classification system will work on images that it hasn't processed before? They resort to a method called cross-validation, where we train algorithm using only a randomly selected sample of data, called the training set.

The model is then evaluated on the rest of the data, called the validation set. The classification performance obtained by the classification system on the training set is usually higher than on the validation set. It is a better reflection of how the algorithm behaves on samples that it has not previously processed.

The success of the classification algorithm using images that a user posts in the future will depend on how representative the training set was of the entire dataset.

For example, if users start posting images of a solar eclipse, the system will only know how to classify it if there were examples of solar eclipses in the training set.

Data analytics solutions are sometimes subject to the same human biases because this bias is expressed in the datasets used to train them.

Conclusion to lecture 9

Classification can be viewed as a regression problem where the target variable is discrete and represents the class in which a human expert classified the data sample. The main classification algorithms are k-NN analysis, SVM, and decision trees. Classification algorithms have many applications. For example, they are used in risk assessment, to determine which of many factors contributes to the likelihood of different risks. Medical diagnostics – classification systems can use guided questions to build a decision tree that can help diagnose various diseases and disease risks. Image recognition – when recognizing handwriting, the system can work on the task of identifying handwritten digits.

Questions for reinforcement

1. What is data classification?
2. What classification algorithms do you know?
3. What is the use of classifications?
4. What is the KNN algorithm used for?
5. What is the decision tree algorithm used for?

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. Visualizing KNN, SVM, and XGBoost on Iris Dataset. URL: https://www.kaggle.com/code/mgabrielkerr/visualizing-knn-svm-and-xgboost-on-iris-dataset

Lecture 10.
Pyplot module. Plotly tool.
Types of data visualization. Anomaly visualization. Using Folium and Leaflet.js libraries to build maps

Lecture plan

10.1. Pyplot module.
10.2. Plotly tool.
10.3. Types of data visualization.
10.4. Visualization of anomalies.
10.5. Using Folium and Leaflet.js libraries to build maps.

10.1. Pyplot module

Pyplot is a **matplotlib** module that includes a set of style functions that can be used to create and customize a plot.

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot([1,2,3,4], [1, 4, 9, 16])
plt.plot([1,2,3,4], [1, 5.7, 15.6, 32])
plt.plot([1,2,3,4], [1, 8, 27, 48])
plt.plot([1,2,3,4], [1, 11.3, 46.8, 128])
plt.xlabel('X Label Here')
plt.ylabel('Y Label Here')
plt.title('Title Here')
plt.show()
```

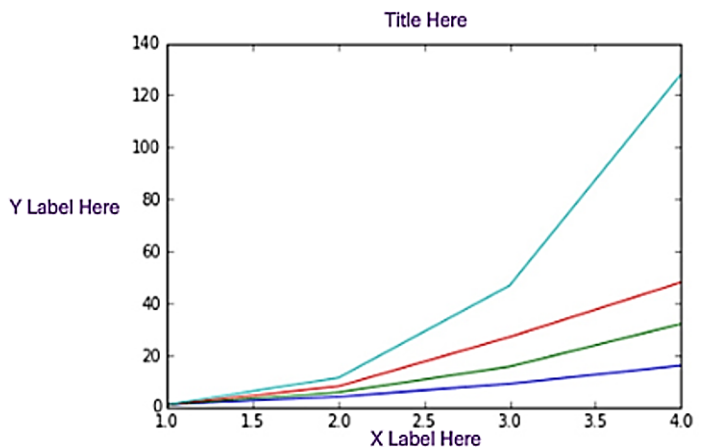


Fig. 10.1. Creating a quick story with typical styles [1]

We can customize the default style by adding inline code. In Fig. 10.2, the following modifications are implemented.

The standard size of a figure is 6.4 by 4.8 inches. To change the size, use the code **plt.figure (figsize = (width, height))**.

The default line width is 1.5 points. To change the size, add the linewidth attribute.

The default font size is 10.0 points. To increase the font size, add the fontsize attribute.

```
plt.figure(figsize = (9.6,7.2))
plt.plot([1,2,3,4], [1, 4, 9, 16])
plt.plot([1,2,3,4], [1, 5.7, 15.6, 32])
plt.plot([1,2,3,4], [1, 8, 27, 48])
plt.plot([1,2,3,4], [1, 11.3, 46.8, 128], linewidth = 3.0)
plt.xlabel('X Label Here', fontsize = 14)
plt.ylabel('Y Label Here', fontsize = 14)
plt.title('Title Here', fontsize = 20)
plt.xticks(fontsize = 10)
plt.yticks(fontsize = 10)
plt.show()
```

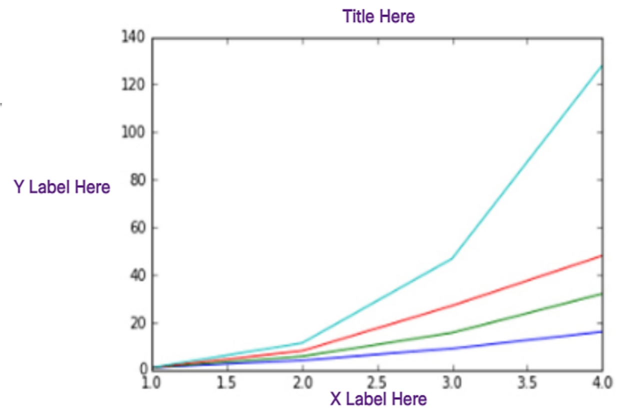


Fig. 10.2. Changing default Pyplot styles [1]

If we reuse the same inline code for multiple stories, we can create a custom style sheet. By referencing a custom style sheet, we can ensure that all stories have the same style features and avoid errors in the inline code.

To create a custom style sheet, we need to open a text editor and add a line for each element in the story that we want to customize. We can use the Linux terminal to create and edit a text file [2].

The code in Fig. 10.3 changes some of the many plot attribute settings.

```
root @ ~ / myfiles # cat mystyle.mplstyle
 axes.titlesize: 24
 axes.labelsize: 16
 lines.linewidth: 5
 xtick.labelsize: 10
 ytick.labelsize: 10
```

Fig. 10.3. Example of a custom style [1]

Save the stylesheet in an appropriate location with the extension .mplstyle. We can save it anywhere, but we will need to provide path information when we reference the stylesheet. For example, if we save the stylesheet in "myfiles", code for referencing the stylesheet would be:

plt.style.use('/home/pi/notebooks/myfiles/mystyle.mplstyle')

If we store the configuration file in the matplotlib configuration directory, the path information is not needed.

Use **get_configdir()** command to find the default **matplotlib** configuration files. Then copy the stylesheet to that location, as shown in Fig. 10.4.

```
!pwd; ls
import matplotlib as mpl
mpl.get_configdir()
```

```
/home/pi/notebooks/myfiles
Custom Style Sheet Test.ipynb mystyle.mplstyle
My First Notebook.ipynb       temp-plot.html

u'/root/.config/matplotlib'
```

```
!cp mystyle.mplstyle /root/.config/matplotlib/mystyle.mplstyle
!ls /root/.config/matplotlib
```

```
mystyle.mplstyle
```

Fig. 10.4. Copying a stylesheet to a directory [3]

Now all that is needed is the name of the style sheet to apply it to the plot, as shown in Fig. 10.5.

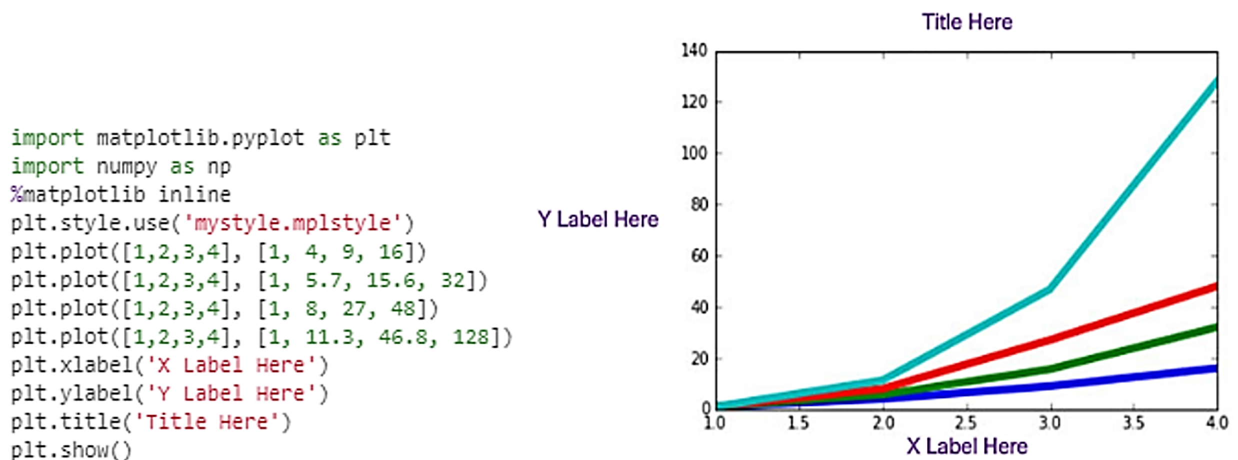


Fig. 10.5. Link to a custom style sheet [3]

Mat plotlib has several styles that can be called in code. Use **print (plt.style.available)** to view **matplotlib styles**, as shown in Fig. 10.6.

```
print(plt.style.available)
```

```
[u'seaborn-darkgrid', u'seaborn-notebook', u'classic', u'seaborn-ticks',
 u'grayscale', u'bmh', u'seaborn-talk', u'dark_background', u'ggplot',
 u'fivethirtyeight', u'seaborn- colorblind', u'seaborn-deep',
 u'seaborn-whitegrid', u'seaborn-bright', u'seaborn- poster',
 u'seaborn-muted', u'seaborn-paper', u'seaborn-white', u'seaborn-pastel',
 u'seaborn-dark', u'seaborn-dark-palette']
```

Fig. 10.6. Available styles in Matplotlib [3]

Matplotlib uses the classic style by default. Also use plt.style.use to change the style. For example, **plt.style.use('grayscale')** will change the style to grayscale, as shown in Fig. 10.7.

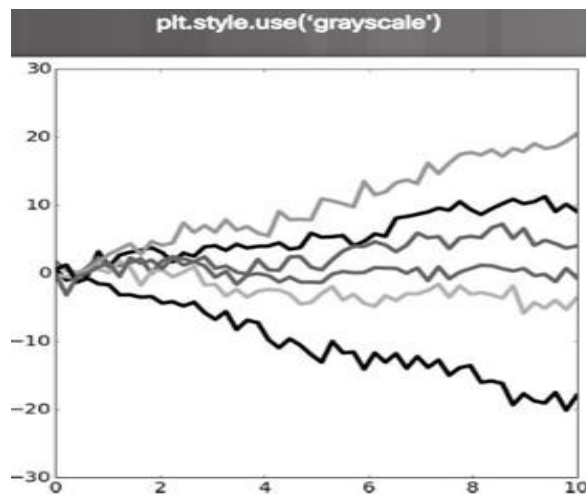


Fig. 10.7. Grayscale style in Matplotlib [3]

Fig. 10.8 shows other examples of style sheets available in matplotlib.

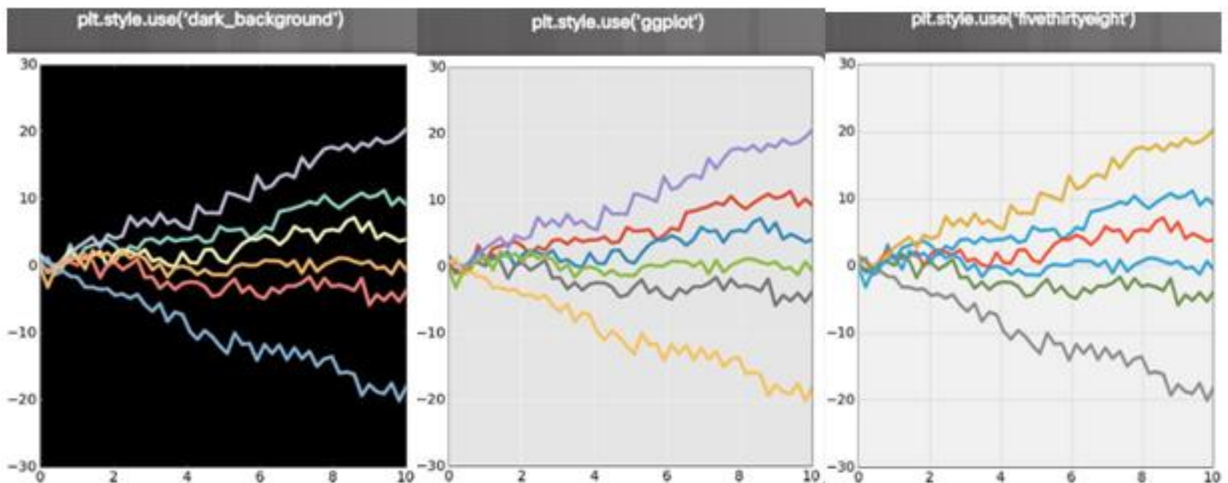


Fig. 10.8. Some other styles in Matplotlib [3]

10.2. Plotly tool

Plotly is a powerful online tool that allows quickly generate beautiful data visualizations. Plotly has a variety of resources for data analysts and web developers, including API libraries, shape converters, Google Chrome extensions, and an open source JavaScript library.

The Plotly website has a large amount of material available for free, including a graphical interface for creating and visualizing data. To save visualizations, we need to sign up for a free user account, as shown in Fig. 10.9.

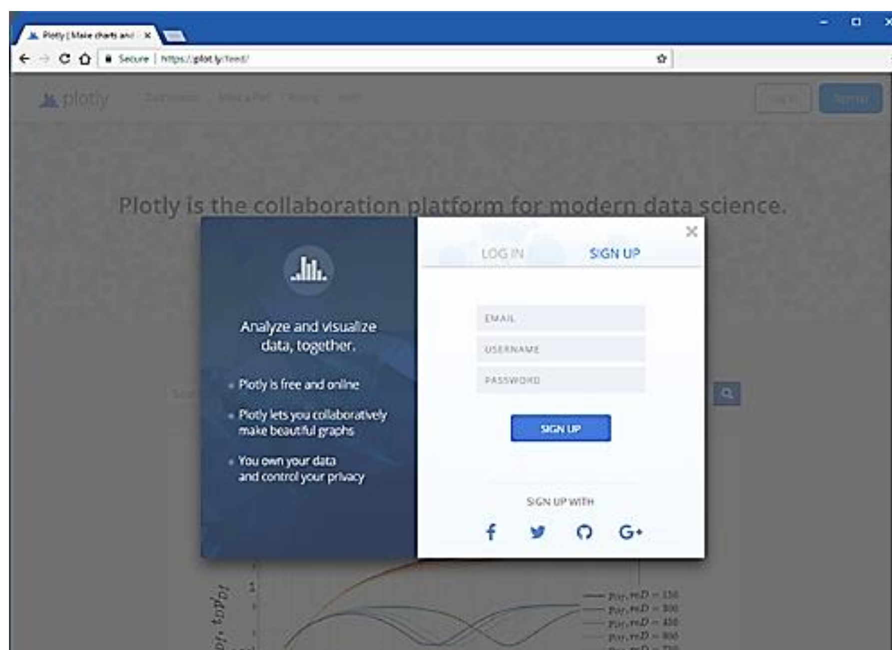


Fig. 10.9. Plotly account

After signing up, we can browse a website with examples of public data visualizations made by other users. The Python plotly library (plotly.py) is an open-source interactive graphing library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3D use cases.

Built on top of the Plotly JavaScript library (plotly.js), plotly.py allows Python users to create beautiful interactive web visualizations that can be displayed in a Jupyter Notebook, saved as standalone HTML files, or served as part of Python-built web applications using Dash.

After successfully logging in, we can use the Plotly graphical GUI to create several different software products. As shown in Fig. 10.10, click Create, then Chart to start creating a new chart from scratch.

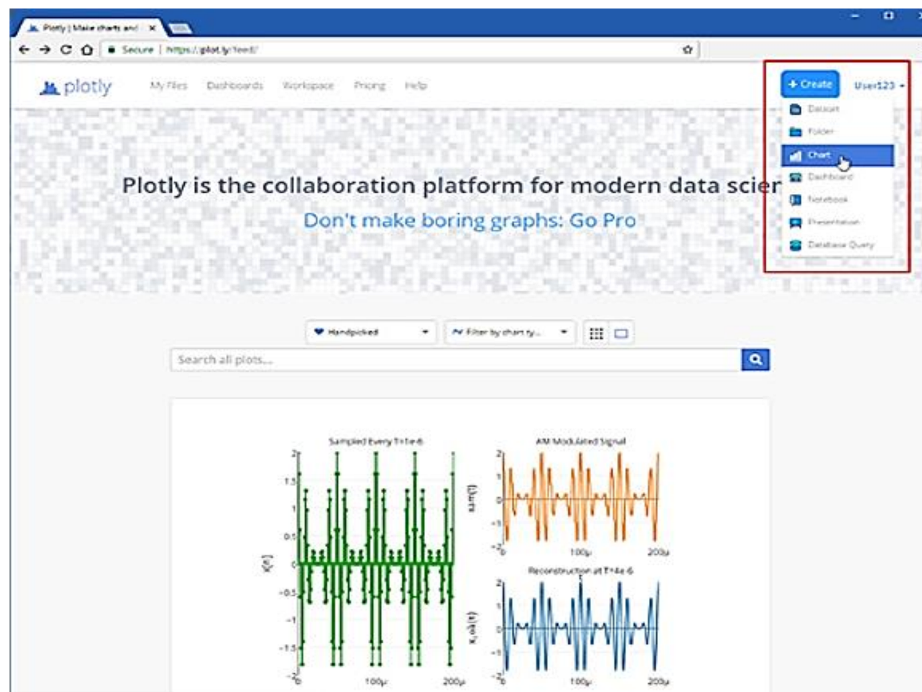


Fig. 10.10. Sign up for a free Plotly account

Click on "Chart Type" as shown in Fig. 10.11. Notice all the available chart types. Many of them are freely available. Others need updating. Select "Line Chart" and enter some test data into the spreadsheet and tools to create plot. Export is disabled, we need to click on the "Save" button.

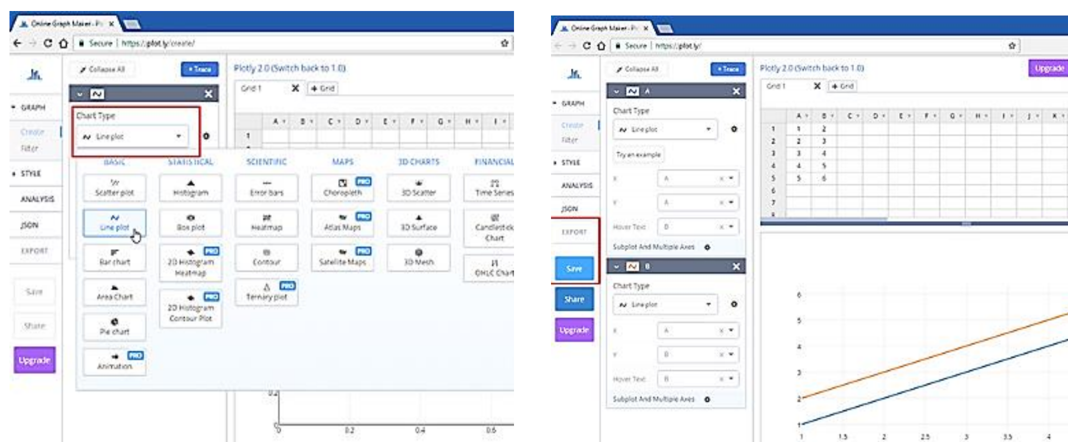


Fig. 10.11. Selecting a chart type and saving it in Plotly

Click Export to see the available options from the Create page. We are limited to a few image file formats and HTML export (Fig. 10.12). HTML export defaults to JavaScript.

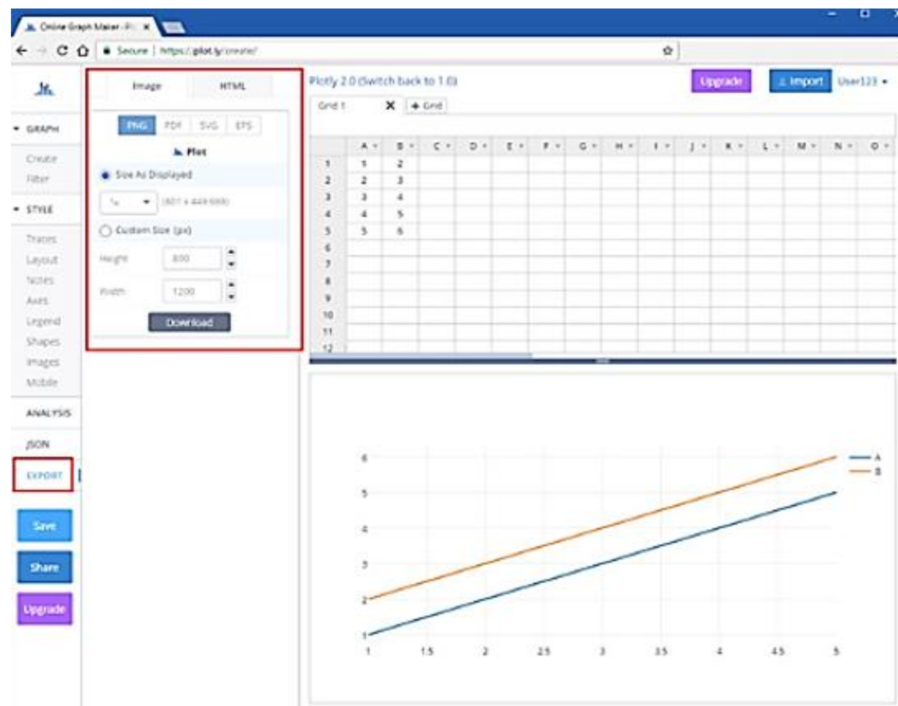


Fig. 10.12. Exporting a chart to Plotly

To access all available export formats, including Python, click username in the upper right corner, then “My Files,” as shown in Fig. 10.13.

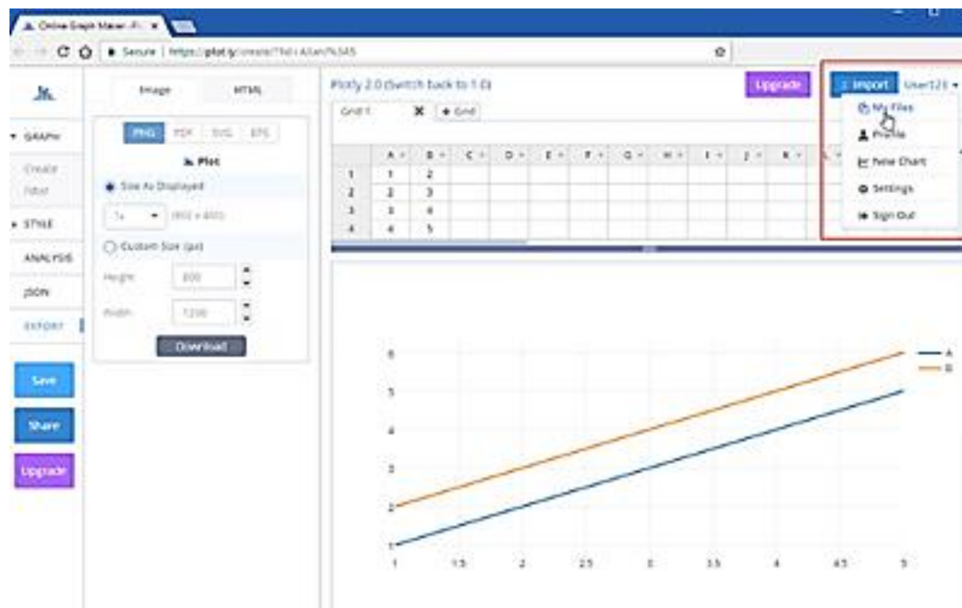


Fig. 10.13. Access to all available export formats in Plotly

In the list of your files, hover your mouse over the file you want to export and click the View button, as shown in Fig. 10.14. This will take you to a page where you can view your graph, data, code, and sources. The Code tab allows view the code in different languages. You can copy the code and paste it into a text file.

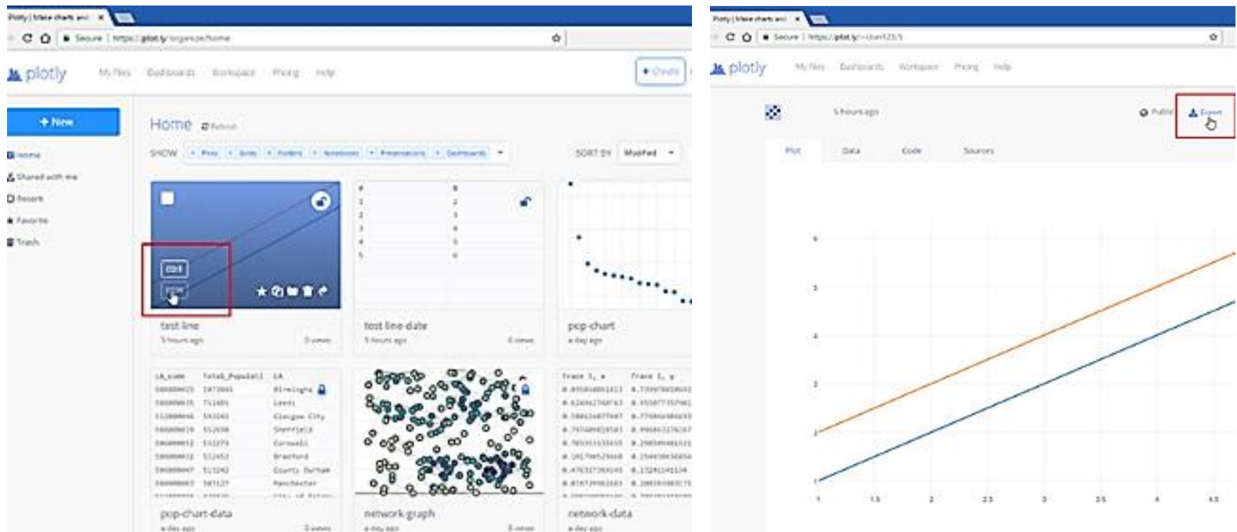


Fig. 10.14. Viewing and exporting a file in Plotly

You can also click Export, which will display all the export options available on the Plotly website. The code export formats are shown in Fig. 10.15. Click Python to download .py file with all the code needed to render the chart.

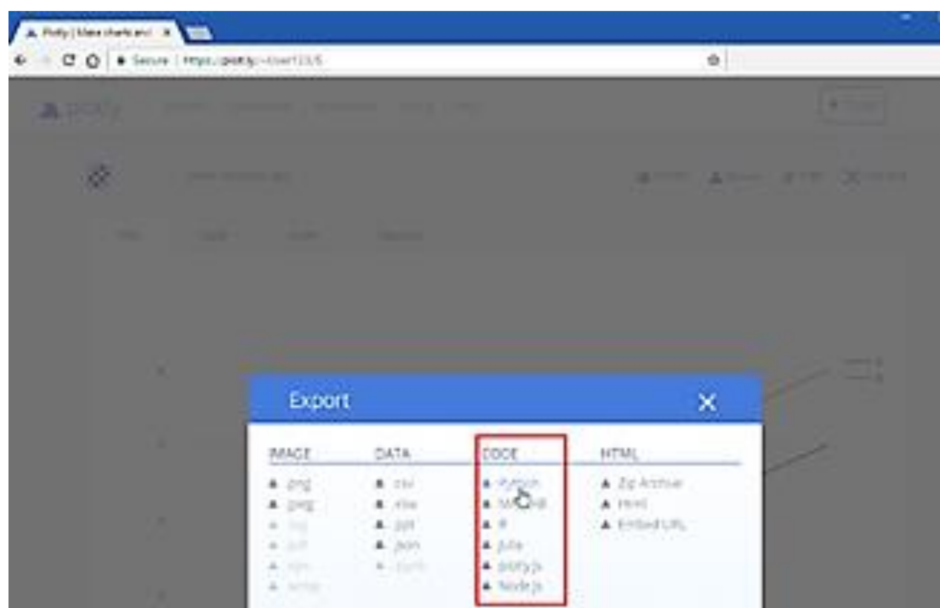


Fig. 10.15. Uploading a file to Plotly

You can also use Plotly to create web pages offline. To do this, you need to install Plotly in a Jupyter Notebook with the following code:

!pip install plotly

The code in Fig. 10.16 will create a line graph and save it in the current directory. You can then publish the web page to your web server. Plotly has many tutorials and help pages for this data visualization platform (Fig. 10.17).

```
import plotly
              from plotly.graph_objs import Scatter, Layout
              plotly.offline.plot({
                   "data": [Scatter(x=[1, 2, 3, 4], y=[4, 3, 2, 1])],
                   "layout": Layout(title="Hello World!")
              })
```

```
'file:///home/pi/notebooks/myfiles/temp-plot.html'
```

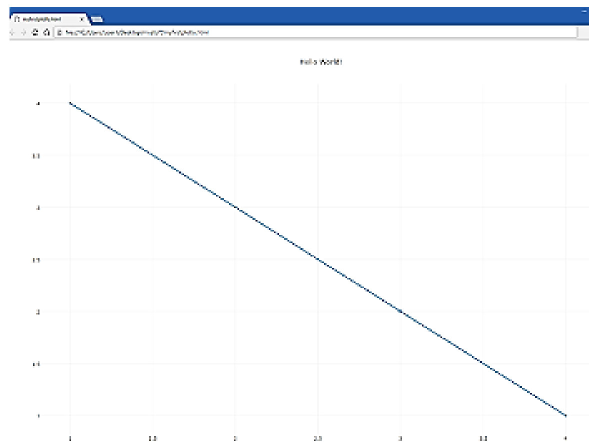


Fig. 10.16. Creating a graph and saving it in the current directory

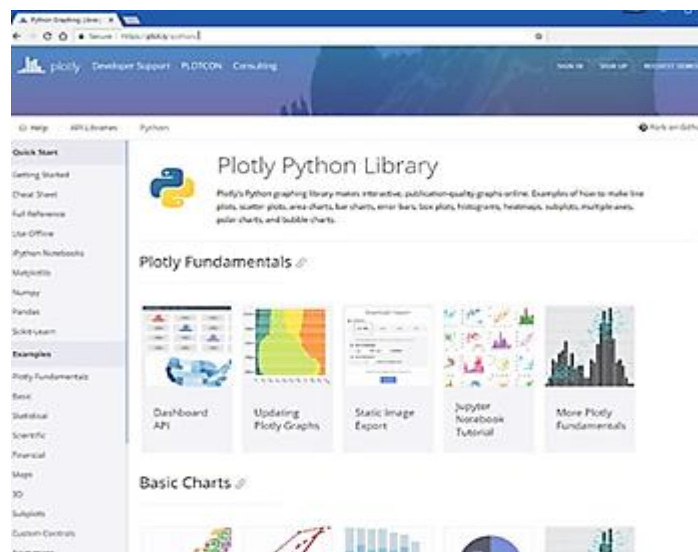


Fig. 10.17. Reference materials in Plotly

10.3. Types of data visualization

Determining the best type of chart usually depends on the answers to the following questions:

How many variables are you going to show?

How many data points are there in each variable?

Is your data over time or are you comparing items?

Line charts are one of the most commonly used types of comparison charts, they are used when you have a continuous data set, the number of data points is high, and you want to show the trend of the data over time.

Fig. 10.18. Line diagram [3]

Examples:

- quarterly sales for the last five years;
- number of customers per week in the first year of a new retail store.

Let's look at some best practices for building line charts.

- Label your axes.
- Plot time on the x-axis (horizontal) and data values on the y-axis (vertical). Label both axes.
- Use a solid line for data to emphasize the continuity of the data.
- Minimize the number of data sets. We should have a good reason for plotting more than four rows. It is a good idea to add a legend to the chart to help your audience understand what they are looking at.

Column charts are placed vertically, they are probably the most common type of chart used when you want to display the value of a specific data point and compare that value across similar categories.

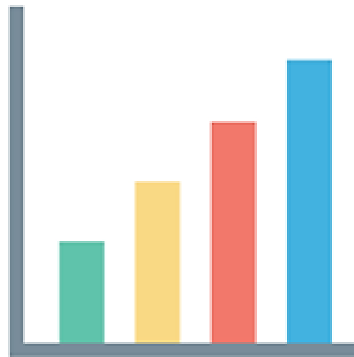


Fig. 10.19. Bar chart [3]

Examples:

- population of countries;
- sales for the top four automobile companies;
- average test scores for sixth grade students in mathematics.

Let's look at some best practices for building bar charts.

- Label your axes.
- If time is one of the dimensions, it should be plotted on the x-axis.
- If time is not part of the data, consider ordering the column height to rise or fall.
- Fill columns with a solid color. If we want to make one column stand out, consider using an accent color and make all other columns the same.
- Columns work best when there are no more than seven categories on the horizontal axis. This helps the viewer clearly see the values for each column.
- Start the y-axis values at zero to accurately reflect the full value of the column.
- The distance between columns should be approximately half the width of the column.

Histograms are similar to bar charts, except they are arranged horizontally. Longer bars indicate larger numbers. They are best used when the names for each data point are long.

Fig. 10.20. Histogram [3]

Examples:

- gross domestic product (GDP) of the top 25 countries;
- number of cars sold by each sales representative;
- exam scores for each student in mathematics.

Let's look at some best practices for building a histogram.

- Label your axes.
- Consider arranging the bars so that the lengths go from longest to shortest. The data type will likely determine whether the longest bar should be at the bottom or top.
- Fill the bars with a solid color. If you want to make one bar stand out, consider using an accent color and make all the other columns the same.
- Start the x-axis value at zero to accurately reflect the full bar value.
- The distance between the bars should be approximately half the width of the bar.

Pie charts are used to display the composition of a static number. The segments represent a percentage of that number. The total sum of the segments should equal 100% (Fig. 10.21).

Fig. 10.21. Pie chart [3]

Examples:

- annual expenses for the corporation (e.g., rent, administrative services, utilities, production);
- energy sources in the country (oil, coal, gas, solar energy, wind, etc.);
- survey results for favorite movie type (e.g., action, comedy, drama, science fiction).

Consider some best practices for pie charts.

- Keep the number of categories to a minimum so that the viewer can differentiate between segments. After ten or more segments, the slices start to lose meaning and impact.
- If necessary, consolidate smaller segments into a single segment with a label, such as "Other" or "Miscellaneous."
- Use a different color or gray scale for each segment.
- Arrange the segments according to size.
- Make sure that the value of all segments is 100%.

Scatter plots used for correlation visualizations, to demonstrate the distribution of a large number of data points, to demonstrate clustering or identify residuals in data, etc. (Fig. 10.22).

Fig. 10.22. Scatter diagram [3]

Examples:

- comparing the life expectancy of each country with its GDP;
- comparing daily ice cream sales with average outdoor temperature;
- comparing weight with height of each person.

Let's look at some best practices for constructing scatter plots.

- Label your axes.
- Make sure the dataset is large enough to provide visualization for clustering.
- Start the y-axis value at zero to accurately represent the full value of the bar. The x-axis value will depend on the data. For example, age ranges can be labeled on the x-axis.
- If the scatter plot shows a correlation between the x and y axes, consider adding a trend line.
- Do not use more than two trend lines.

When you need to add a third variable to a scatterplot, consider using a bubble plot. For example, a scatterplot showing the relationship between life expectancy and GDP could be improved by increasing the size of each data point to represent the population.

10.4. Visualization of anomalies

As an aid to visualizing 3D graphics, it can be useful to view data from different angles. To do this, you need to import the interactive class from the **ipywidgets** library to add tools for adjusting the azimuth and elevation of the 3D graph.

Azimuth will allow rotate the plot horizontally. Elevation will allow rotate the plot vertically (Fig. 10.23).

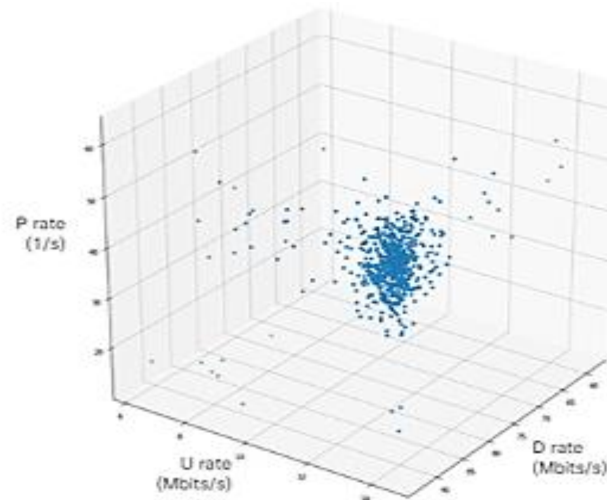


Fig. 10.23. Changing the azimuth and elevation of a 3D plot [3]

Anomalies can represent abnormal data or values. Data can be corrupted or distorted by many factors during measurement, transmission, or storage. These values deviate so far from expected values that they can distort the results of the analysis. These observations are often removed from the data set after careful consideration. There are other types of anomalies that can indicate serious problems with the object being measured. For example, unusually high temperatures or vibration measurements taken by sensors attached to a machine could indicate that a part of it is about to fail. In this case, an IoT streaming data analytics application could send an alarm to alert maintenance personnel that the machine needs attention.

In Fig.10.23 several data points lie outside the cluster region. These are anomalies. Anomalies can be identified by finding points that lie above the mean. By measuring the difference between the x, y, and z coordinates of each data point and the mean x, y, and z coordinates, we obtain a list of distances for each data point.

To detect anomalies, you need to define a decision boundary that determines whether a data point is normal or an anomaly. To do this, you first need to normalize the distance data by setting the largest distance to 1. Then, define a threshold between 0 and 1 for the decision boundary.

10.5. Using Folium and Leaflet.js libraries to build maps

The Folium mapping library combines the power of Python with the display capabilities of the Leaflet.js library. Folium allows take Python date frames and display them on an interactive Leaflet map (Fig. 10.24).

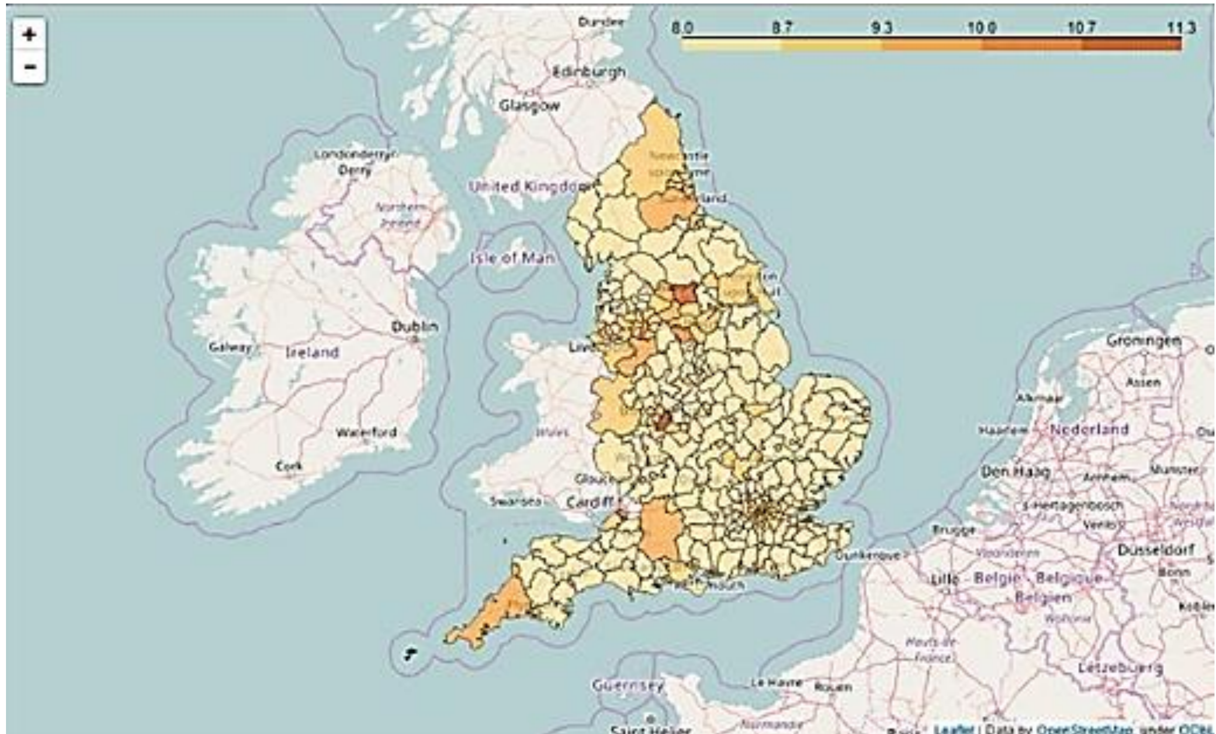


Fig. 10.24. Example of using Folium to display data
on the Leaflet map [3]

Folium Tilesets are a collection of raster or vector data that can display a map on mobile devices or in a browser.

The Folium library supports a number of different tilesets, including OpenStreetMap, Mapbox, and Stamen. By default, Folium uses the OpenStreetMap tileset. Mapbox and Stamen maps can be specified using the tiles attribute (Mapbox requires a user account to obtain API access tokens).

By importing the Folium library and using the OpenStreetMap tileset, you can create a map by specifying a set of coordinates (Fig. 10.25).

```
import folium
svmap = folium.Map(location=[37.3861, -122.0839])
svmap
```

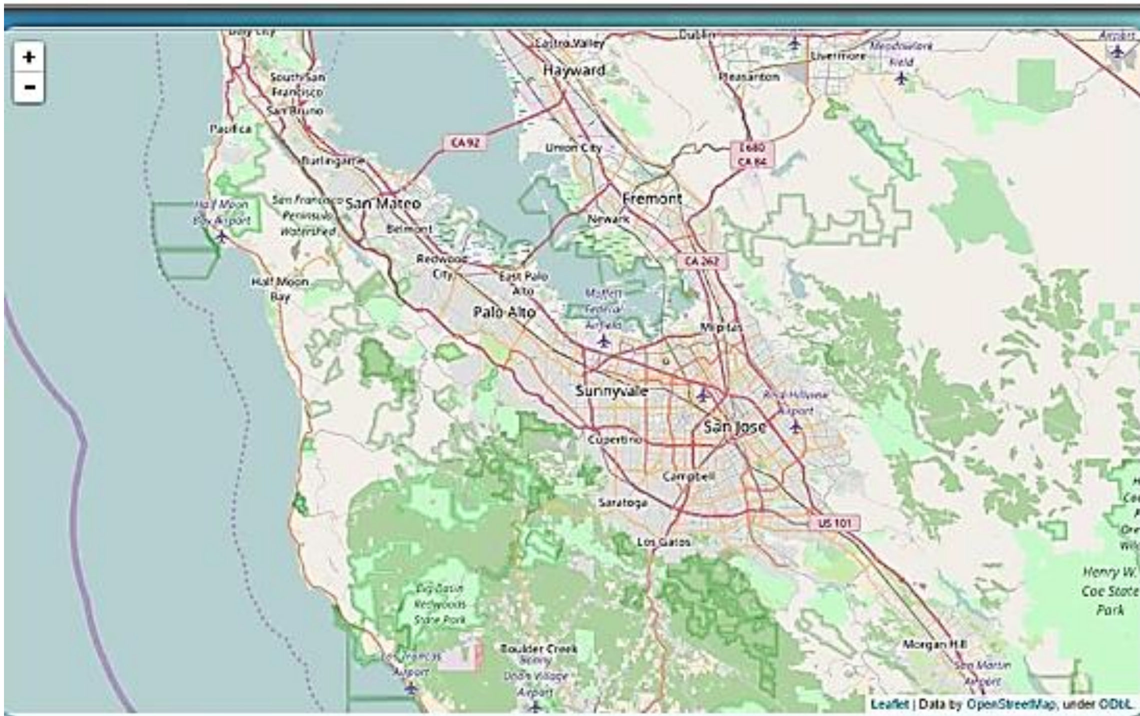


Fig. 10.25. Example of using the Folium library [3]

Consider example of how to use the **Folium** library in Python to create an interactive map with a marker:

```python
import folium

# Create a map centered at a specific location (latitude, longitude)
map_center = [40.7128, -74.0060]  # New York City coordinates
mymap = folium.Map(location=map_center, zoom_start=12)

# Add a marker to the map
folium.Marker(
    location=map_center,
    popup='New York City',
    tooltip='Click for more info'
).add_to(mymap)

# Save the map to an HTML file
mymap.save("nyc_map.html")
```

Explanation:

- **folium.Map()** initializes a map at the given coordinates.
- **folium.Marker()** adds a marker with a tooltip and popup.
- **mymap.save()** saves the map as an HTML file you can open in a browser.

Conclusion to lecture 10

Pyplot is a matplotlib extension that includes a set of styling functions for creating and customizing plots, allows reuse the same built-in code for multiple plots, and creates custom stylesheets. Plotly is a powerful online tool for quickly creating data visualizations. In data analysis, line charts are used for continuous data sets where the number of data points is large and we want to show a trend over time. Column charts are arranged vertically. This type of chart is used when you want to display the value of a particular data point and compare that value across similar categories. Histograms are similar to column charts, except they are arranged horizontally. Pie charts are used to display the composition of a static number. The segments represent a percentage of that number. The sum of the segments totals 100%. Scatter charts are used to visualize the correlation or distribution of a large number of data points and to cluster or identify outliers in the data. The Folium library is used to visualize data on an interactive map in Python.

Questions for reinforcement

1. What is the Pyplot module used for?
2. Describe the Plotly tool.
3. What types of data visualization do you know?
4. What are data anomalies?
5. What is the Folium library used for in Python?

List of recommended literature

1. Customizing Matplotlib with style sheets and rcParams. URL: https://matplotlib.org/tutorials/introductory/customizing.html
2. How to Quickly Create a Text File Using the Command Line in Linux. URL: https://www.howtogeek.com/199687/how-to-quickly-create-a-text-file-using-the-command-line-in-linux/
3. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics

Lecture 11.
Data analysis in R.
Factors, lists, frames

Lecture plan

11.1. History of R language development.
11.2. R language features.
11.3. Objects, packages, functions.
11.4. Vectors, matrices, and operations in R.
11.5. Factors, lists, frames.

11.1. History of R language development

R programming language was first introduced in the early 1990s by Robert Gentleman and Ross Ihaka, both lecturers at the University of Auckland (Fig.11.1). The first letter of their names gave the name to the new system.

Fig. 11.1. Developers of the R language (Robert Gentleman, Ross Ihaka)

R language was created as an analogue of the statistical computing language S, which was implemented by John Chambers, Rick Becker, Trevor Hastie, Allan Wilkes, and others at Bell Labs in the mid-1970s and published in the early 1980s. Robert and Ross created R as an open source project in 1995. Since 1997, the R project has been managed by the R Core Group.

The first release of R was released in February 2000. Today, new tools are being created for R, and there are over 10,000 user-created libraries to enhance R's functionality. These packages are crowdsourced for quality assurance and supported by recognized leaders in each field [1].

11.2. R language features

R is a statistical programming language that data scientists use all over the world – from mapping broad social and marketing trends on the internet.

A big data scientist can use two great tools: R and Python, but learning statistical modeling is more important than learning a programming language. A programming language is just a tool for analyzing data. The most important task in data science is how the scientist works with the data, such as importing, cleaning, preparing, developing functions, and selecting functions. This should be the main focus. Trying to learn R and Python at the same time for data analysis without having sufficient knowledge of statistics is unwise. The job of an analyst is to understand the data, manipulate it, and discover the best approach.

The main audience for data science is business professionals. A few years ago, R was not as structured as other programming tools. To overcome this serious problem, Hadley Wickham created a collection of packages called tidyverse. The rules of the game have changed for the better. Data manipulation becomes a trivial and intuitive process. Creating graphs is also not difficult. The best algorithms for machine learning can be implemented using R. Packages such as Keras and TensorFlow allow solve machine learning problems. Data analysis, such as clustering, correlation, and reduction, can also be performed using R.

The R language also has a package for running Xgboost, one of the best algorithms for Kaggle competitions. R can "talk" to other programming languages. You can call Python, Java, C ++ in R. The analyst can connect R to various databases such as Spark or Hadoop. The R language allows parallelize operations to speed up calculations. The package for parallel computing allows run tasks on different cores of the machine.

11.3. Objects, packages, functions

R language is a high-level object-oriented programming language. For a layperson, the strict definition of the concept of "object" is quite abstract. For

simplicity, you can call objects everything that was created in the process of working with R. There are two main types of objects in R.

1. Objects intended for data storage ("data objects") are individual variables, vectors, matFigs and arrays, lists, factors, data tables.

2. Functions ("function objects") are specified programs designed to create new objects or perform certain actions on them.

Objects of the R environment, intended for collective and free use, are assembled into packages that are united by similar topics or data processing methods. There is a certain difference between the terms package ("package") and library ("library"). The term "library" defines a directory that can contain one or more packages. The term "package" denotes a set of functions, HTML pages, manuals and examples of data objects intended for testing or training. Packages are installed in a specific directory of the operating system or, in an uninstalled form, can be stored and distributed in archived * .zip files of Windows (the version of the package must correspond to a specific version of R). All R packages belong to one of three categories: base ("base"), recommended ("recommended") and others installed by the user. You can get a list of them on a specific computer by issuing the library () command or:

```
installed.packages (priority = "base")
installed.packages (priority = "recommended")
# Get the full list of packages
packlist <- rownames (installed.packages ())
# Output information to the clipboard in Excel format
write.table (packlist, "clipboard", sep = "\t", col.names = NA)
```

The basic and recommended packages are usually included in the R installation file. Of course, there is no need to install many different packages at once "just in case". To install a package, simply select the "Packages > Install Package(s)" menu item in the R Console command window or enter, for example, the command:

```
install.packages (c ( "vegan", "xlsReadWrite", "car"))
```

Table 11.1. Some basic R packages

R packages	Purpose
base	Basic R constructs
compiler	Compiler of R packages
datasets	A set of tables with data for testing and demonstrating functions
graphics	Basic graphic functions
grDevices	Graphic device drivers, color palettes, fonts
grid	Functions for creating graphic layers
methods	Object-oriented programming components (classes, methods)
splines	Functions for working with regression splines of various types
stats	Basic statistical analysis functions
stats4	Methods of S4 class statistical functions
tcltk	User interface components (menus, selection boxes, etc.)
tools	Information support, administration and documentation
utils	Various debugging, input-output, archiving, etc. utilities

All data objects (and therefore variables) in R can be divided into the following classes (i.e. object types):

° numeric – objects that include integer and real numbers (double);

° logical – logical objects that accept only two values: FALSE (F) and TRUE (T);

° character – character objects (variable values are specified in doubles, or single quotes).

In R, we can create names for different objects (functions or variables) in both Latin and Cyrillic, but it should be noted that a (Cyrillic) and a (Latin) are two different objects. R is case-sensitive, that is, lowercase and uppercase letters are different in it. Variable names (identifiers) in R must start with a letter (or a period .) and consist of letters, numbers, periods, and underscores.

Using the ?<Name> command, you can check whether a variable or function with the specified name exists. The is.numeric (<object_name>), is.integer

(<name>), is.logical (<name>), is.character (<name>) functions check whether a variable belongs to a certain class, and to convert an object to another type, you can use the as.numeric (<name>), as.integer (<name>), as.logical (<name>), as.character (<name>) functions.

There are a number of special objects in R:

° Inf – positive or negative infinity (usually the result of dividing a real number by 0);

° NA – "missing value" (Not Available);

° NaN – "not a number" (Not a Number).

You can check whether a variable belongs to any of these special types using the is.finite (<name>), is.na (<name>), and is.nan (<name>) functions, respectively.

An expression in R is a combination of elements such as an assignment operator, arithmetic or logical operators, object names, and function names. The result of an expression is usually immediately displayed in a command or graphics window. When an assignment operation is performed, the result is stored in the corresponding object and is not displayed on the screen. As an assignment operator in R, we can use either the "=" symbol, or a pair of symbols "<-" (assigning a certain value to an object on the left) or "->" (assigning a value to an object on the right). It is considered good programming style to use "<-".

R language expressions are organized in a script by lines. We can enter multiple commands on a single line, separating them with a ";" character. A single command can also be placed on two (or more) lines.

11.4. Vectors, matrices, and operations in R

A vector in R is a named one-dimensional object containing a set of elements of the same type (numeric, logical, or text values – no combinations of them are allowed). To create vectors of small length in R, the concatenation function c() (from "concatenate" – to unite, to connect) is used. The arguments of this function are listed as comma-separated values to be combined into a vector, for example:

```
my.vector <- c (1, 2, 3, 4, 5)
```

```
my.vector
[1] 1 2 3 4 5
```

To create vectors containing a sequential set of numbers, the seq() function (from "sequence") is convenient. For example, a vector named S containing a set of integers from 1 to 7 can be created as follows:

```
S <- seq(1,7)
S
[1] 1 2 3 4 5 6 7
```

The identical result will be obtained using the command

```
S <- 1: 7
S
[1] 1 2 3 4 5 6 7
```

As an additional argument to the seq() function, you can specify the step for increasing numbers:

```
S <- seq (from = 1, to = 5, by = 0.5)
S
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Vectors containing the same values are created using the rep() function (from "repeat"). For example, to create a text vector Text that will contain five "test" values, you would run the following command:

```
Text <- rep("test", 5)
Text
[1] "test" "test" "test" "test" "test"
```

The R system is capable of performing a wide variety of operations on vectors. For example, several vectors can be combined into one using the concatenation function discussed above:

```
v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)
V <- c(v1, v2)
V
```

[1] 1 2 3 4 5 6

If try to combine, for example, a text vector with a numeric vector, an error message will not appear – the program simply converts all values to text. To work with a specific element of a vector, we need to have a way to distinguish it from other elements. To do this, when creating a vector, all its components are automatically assigned index numbers, starting from 1. To refer to a specific element, we need to specify the name of the vector and the index of this element in square brackets:

Create a numeric vector y containing 5 numeric values:

y <- c(5, 3, 2, 6, 1)

Let's check what the third element of the vector y is:

[3]

[1] 2

Using index numbers, you can perform various operations on selected elements of different vectors. Indexing is a powerful tool that allows create sets of values according to specified criteria. For example, to display the 3rd, 4th, and 5th values of the vector y, we need to execute the command:

y [3: 5]

[1] 2 6 1

From the same vector, we can select, for example, only the first and fourth values, using the well-known concatenation function c():

y [with (1, 4)]

[1] 5 6

Similarly, we can remove the first and fourth values from the vector y by applying a minus sign before the concatenation function:

y [-c (1, 4)]

[1] 3 2 1

A logical expression can serve as a criterion for selecting values. For example, let's select all values greater than 2 from the vector y:

y [y> 2]

[1] 5 3 6

Indexing is also a handy tool for making corrections to existing vectors. For example, you can correct the second value of the vector z from 0.1 to 0.3:

z[2] <- 0.3

To sort the values of a vector in ascending or descending order, use the sort() function in combination with the argument decreasing = FALSE or decreasing = TRUE, respectively:

sort(z) # default decreasing = FALSE

[1] 0.3 0.5 0.6

sort(z, decreasing = TRUE)

[1] 0.6 0.5 0.3

A matrix is a two-dimensional vector. In R, the function of the same name is used to create matFigs:

my.mat <- matrix (seq (1, 16), nrow = 4, ncol = 4)

my.mat

[, 1] [2] [3] [4]

[1,] 1 5 9 13

[2,] 2 6 10 14

[3,] 3 7 11 15

[4,] 4 8 12 16

By default, the matrix is filled in columns, that is, the first four values are included in the first column, the next four values are

in the second column, etc. This filling order can be changed by setting the special argument byrow (from "by row" - by rows) to TRUE:

my.mat <- matrix (seq (1, 16), nrow = 4, ncol = 4, byrow = TRUE)

my.mat

[, 1] [2] [3] [4]

[1,] 1 2 3 4

[2,] 5 6 7 8

[3,] 9 10 11 12

[4,] 13 14 15 16

The corresponding index numbers are automatically output as row and column headings of the created matrix (rows: [1,], [2,], etc.; columns: [, 1], [2], etc.). To add user-defined headings to the rows and columns of matFigs, use the rownames() and colnames() functions, respectively. For example, to label the rows of the matrix my.mat with the letters A, B, C, and D, do the following:

rownames (my.mat) <- c ("A", "B", "C", "D")

my.mat

[, 1] [2] [3] [4]

A 1 2 3 4
B 5 6 7 8
C 9 10 11 12
D 13 14 15 16

The matrix my.mat has 16 values that fit into the available 4 rows and 4 columns. If we try to fit a vector of 12 numbers into a matrix of the same size, R will fill in the missing values by "looping" the short vector:

my.mat2 <- matrix (seq (1, 12), nrow = 4, ncol = 4, byrow = TRUE)

my.mat2

[, 1] [2] [3] [4]

[1,] 1 2 3 4

[2,] 5 6 7 8

[3,] 9 10 11 12

[4,] 1 2 3 4

As we can see, the program again used the numbers 1, 2, 3, and 4 to fill the cells of the last row of the matrix my.mat2.

An alternative way to create matFigs is to use the dim() function (from "dimension"). So, we could create the matrix my.mat from a one-dimensional vector as follows:

my.mat <- 1:16

of the vector my.mat to 4 x 4:

dim (my.mat) <- c (4, 4)

my.mat

[, 1] [2] [3] [4]

[1,] 1 5 9 13

[2,] 2 6 10 14

[3,] 3 7 11 15

[4,] 4 8 12 16

The dim() function allows check the dimension of an existing matrix (or data table):

dim (my.mat)

[1] 4 4

A matrix can also be assembled from several vectors using the cbind() (from colum and bind – column and bind) or rbind() (from row and bind – row and bind) functions:

create four vectors of the same length:

a <- c (1, 2, 3, 4)

b <- c (5, 6, 7, 8)

d <- c (9, 10, 11, 12)

e <- c (13, 14, 15, 16)

combine vectors using the cbind() function:

cbind (a, b, d, e)

abde

[1,] 1 5 9 13

[2,] 2 6 10 14

[3,] 3 7 11 15

[4,] 4 8 12 16

Combine the same vectors using the rbind() function:

rbind (a, b, d, e)

[, 1] [2] [3] [4]

a 1 2 3 4

b 5 6 7 8

9 10 11 12

e 13 14 15 16

Almost all vector operations apply equally to matFigs and arrays. For example, using an index, we can get the necessary elements from matFigs and then subject them to the necessary transformations.

Consider a few examples:

Extract the element of the matrix my.mat located at the intersection of the 2nd row and the 3rd column:

my.mat [2, 3]

[1] 7

Extract from the matrix all elements located in the 4th column (for this, the row number before the comma can be omitted):

my.mat [, 4]

[1] 4 8 12 16

Extract all elements from the matrix that are in the 1st row (in this case, there is no need to specify column numbers):

my.mat [1,]

[1] 1 2 3 4

4. Multiply the 1st and 4th columns of the matrix (element by element):

my.mat [, 1] * my.mat [, 4]

[1] 4 40 108 208

If necessary, the matrix can be transposed (swap rows and columns) using the t() function (from transpose):

t (my.mat)

ABCD

[1,] 1 5 9 13

[2,] 2 6 10 14

[3,] 3 7 11 15

[4,] 4 8 12 16

11.5. Factors, lists, frames

In statistics, data is often grouped according to certain characteristics, for example, gender, social status, stage of the disease, place of sampling, etc.

In R, there is a special class of vectors – factors, which are designed to store codes for the corresponding levels of nominal characteristics. Often, factor levels are encoded as numbers. In such cases, it is important to "instruct" the program so that it "recognizes" the levels of the nominal variable from numbers as such. Suppose that an experiment to test the effectiveness of a new medical drug involved 10 volunteer patients, of whom six patients took the new drug, and the other four – placebo (for example, activated charcoal tablets). To designate the participants in these two groups, we can use the codes 1 (drug) and 0 (placebo). We could store information about all ten participants in the experiment as the following vector:

```
treatment <- c (1, 1, 1, 1, 1, 1, 0, 0, 0, 0)
treatment
[1] 1 1 1 1 1 1 0 0 0 0
```

With this approach, the program will "consider" the vector treatment as a numeric one. This will be an error on our part, since zero and one denote only two levels of a nominal variable. We could just as easily use, for example, 10 to denote the control group of patients (i.e., patients taking placebo) and 110 to denote patients taking the study drug. To convert a numeric (or text) vector into a factor in R, there is a function of the same name factor():

```
treatment <- factor (treatment, levels = c (0, 1))
treatment
[1] 1 1 1 1 1 1 0 0 0 0
Levels: 0 1
```

Note that now when displaying the contents of the treatment object, the program tells us that this object is a factor with two levels (Levels: 0 1). We can additionally verify this using the class (treatment) command:

```
class (treatment)
```

[1] "factor"

A more robust approach that avoids confusion when performing the analysis is to encode factor levels using text values rather than numbers. For example, in our example, we could assign the value yes to patients who took the drug and the value no to patients in the control group. We can recode the levels of an existing factor treatment using the levels() function:

```
levels (treatment) <- c ( "no", "yes")
treatment
[1] yes yes yes yes yes yes no no no no
Levels: no yes
```

Note that when the contents of the treatment vector are printed, the patient codes are not enclosed in double quotes, as is usually the case with text values. This is one of the external signs that we are dealing with a factor, and not with a text vector containing six "yes" values and four "no" values. The same factors can easily be converted back to a numeric vector consisting of the ordinal numbers of the factor levels:

```
as.numeric (treatment)
[1] 2 2 2 2 2 2 1 1 1 1
```

There is also a special command for creating factors:

```
gl (n, k, length = n * k, labels = 1: n),
```

where n is the number of factor levels; k is the number of repetitions for each level; length is the size of the summary object; labels is an optional argument that can be used to specify the names of each factor level.

For example, executing the following command will create the vector my.fac, which is a factor with two levels – Control and Treatment, with each of the labels "Control" and "Treatment" repeated 8 times:

```
my.fac = gl (2, 8, labels = c ("Control", "Treatment"))
my.fac
[1] Control Control Control Control Control Control Control
[8] Control Treatment Treatment Treatment Treatment Treatment Treatment
```

[15] Treatment Treatment

Levels: Control Treatment

Another useful command creates factors by dividing the range of a numerical vector x into intervals:

cut (x, breaks, labels),

where the argument breaks can be either the required number of intervals or a vector containing a list of "break points", and labels defines the names of the levels:

```
x <- c (1,2,3,4,5,2,3,4,5,6,7)
cut (x, breaks = 3)
[1] (0.994,3] (0.994,3] (3,5] (3,5] (3,5] (0.994,3] (3,5]
[8] (3,5] (3,5] (5,7.01] (5,7.01]
Levels: (0.994,3] (3,5] (5,7.01]
cut (x, breaks = 3, labels = letters [1: 3])
[1] a a b b b a b b b c c
Levels: a b c
cut (x, breaks = quantile (x, c (0, .25, .50, .75,1)),
labels = c ( "Q1", "Q2", "Q3", "Q4"), include.lowest = TRUE)
[1] Q1 Q1 Q2 Q2 Q3 Q1 Q2 Q2 Q3 Q4 Q4
```

Levels: Q1 Q2 Q3 Q4In the third code fragment, the numeric vector is "cut" into quartile values, and the include.lowest parameter is specified to avoid the appearance of the uncertainty "NA" for the value x = 1.

Unlike a vector or matrix, which can only contain data of one type, a list or data frame can contain a combination of any data type. This allows store disparate information in a single object. Each component of a list can be a variable, a vector, a matrix, a factor, or another list. These elements can be of different types: numbers, strings, or Boolean variables.

Lists are the most common means of storing intrasystem information: in particular, the results of most statistical analyses in the R program are stored in list objects. The list() function of the same name is used to create lists in R.

Let's create three vectors of different types – with text, numeric, and logical values:

```
vector1 <- c ( "A", "B", "C")
vector2 <- seq (1, 3, 0.5)
vector3 <- c (FALSE, TRUE)
```

combine these three vectors into a single list object, whose components we will name Text, Number, and Logic:

```
my.list <- list (Text = vector1, Number = vector2, Logic = vector3)
# review the contents of the created list:
my.list
$ Text
[1] "A" "B" "C"
$ Number
[1] 1.0 1.5 2.0 2.5 3.0
$ Logic
[1] FALSE TRUE
```

List elements can be accessed using three different indexing operations. The $ sign is used to refer to named components. So, to remove the Text, Number and Logic components from the list my.list we created, you need to enter the following commands in sequence:

```
my.list $ Text
[1] "A" "B" "C"
my.list $ Number
[1] 1.0 1.5 2.0 2.5 3.0
my.list $ Logic
[1] FALSE TRUE
```

It is possible to insert from the list not only its specified vector components, but also individual elements included in these vectors. To do this, we need to use the method already considered earlier – indexing using square brackets.

The peculiarity of working with lists is that first we need to specify the name of the list component using the $ sign, and only then the number(s) of individual elements of this component:

```
my.list $ Text [2]
[1] "B"
my.list $ Number [3: 5]
[1] 2.0 2.5 3.0
my.list $ Logic [1]
[1] FALSE
```

List components can also be extracted using double square brackets for the list component number:

```
my.list [[1]]
[1] "A" "B" "C"
my.list [[2]]
[1] 1.0 1.5 2.0 2.5 3.0
my.list [[3]]
[1] FALSE TRUE
```

After double square brackets with the index number of the list component, we can also specify the number(s) of individual elements of this component:

```
my.list [[1]] [2]
[1] "B"
my.list [[2]] [3: 5]
[1] 2.0 2.5 3.0
my.list [[3]] [1]
[1] FALSE
```

The created list my.list contained only three small vectors, and we knew what vectors they were and where they were in the list. In practice, we may encounter lists whose indexing can be complicated due to the lack of understanding of their structure. To find out the structure of objects in the R language, there is a special function str () (from structure):

str (my.list)

List of 3

$ Text: chr [1: 3] "A" "B" "C"

$ Number: num [1: 5] 1 1.5 2 2.5 3

$ Logic: logi [1: 2] FALSE TRUE

From the above example, it follows that the list my.list includes 3 components (List of 3) with the names Text, Number and Logic (listed on separate lines after the $ sign). These components belong to the character (chr), numeric (num) and logical (Logic) vector types, respectively.

The str () command prints the first few elements of each vector.

A data frame is an R object, similar in structure to a Microsoft Excel spreadsheet. Each column of the table is a vector containing data of a certain type. The rule is that all columns must be the same length (from the "point of view" of R, a data frame is a special case of a list in which all vector components have the same size).

Data tables (frames) are the main class of R objects for storing data.

Typically, such tables are prepared using external programs (for example, Microsoft Excel) and then loaded into the R environment. A small table can be assembled from several vectors using R tools. For this, the data.frame() function is used. Suppose we have observations of the total number of male (Male) and female (Female) populations in three cities City1, City2, and City3. Let's represent this data in the form of a single table named CITY. First, we create text vectors with the names of cities (City) and sex (sex), as well as a vector with the values of the number of representatives of each sex (number):

city <- c ("City1", "City1", "City2", "City2", "City3", "City3")

sex <- c ("Male", "Female", "Male", "Female", "Male", "Female")

number <- c (12450, 10345, 5670, 5800, 25129, 26000)

Combine these three vectors into a single data table:

CITY <- data.frame (City = city, Sex = sex, Number = number)

CITY

City Sex Number

1 City1 Male 12450

2 City1 Female 10345

3 City2 Male 5670

4 City2 Female 5800

5 City3 Male 25129

6 City3 Female 26000

The column headers can be any user-defined names that meet the requirements of R. Extract individual components tables to perform the necessary calculations, as in the examples with lists, can be done using the $ sign, square brackets indicating two indices [<row_number>, column_number>], double square brackets [[]], or directly on the column name:

CITY $ Sex

[1] Male Female Male Female Male Female

Levels: Female Male

CITY $ Number

[1] 12450 10345 5670 5800 25129 26000

Identical results can be obtained using the commands:

CITY [, 2]

[1] Male Female Male Female Male Female

Levels: Female Male

CITY [[3]]

[1] 12450 10345 5670 5800 25129 26000

CITY ["Sex"]

[1] Male Female Male Female Male Female

Levels: Female Male

CITY ["Number"]

[1] 12450 10345 5670 5800 25129 26000

After the column name or index number, we can specify the index numbers of individual table cells, which allow extract the contents of these cells:

```
# Extract the 4th element from the Number column:
CITY $ Number [4]
[1] 5800
# Extract elements 1-3 from the Number column:
CITY $ Number [1: 3]
[1] 12450 10345 5670
# Extract all values of the number exceeding 10000
CITY $ Number [CITY $ Number> 10000]
[1] 12450 10345 25129 26000
# Extract all values of the male population:
CITY $ Number [CITY $ Sex == "Male"]
[1] 12450 5670 25129
# Repeat the same commands, but using []:
CITY [4, 3]
[1] 5800
CITY [1: 3, 3]
[1] 12450 10345 5670
CITY [CITY $ Number> 10000, 3]
[1] 12450 10345 25129 26000
CITY [CITY $ Sex == "Male", 3]
[1] 12450 5670 25129
```

When working with large data tables, it can be difficult to visually examine their contents before starting the analysis. Full summary information about tables (as well as other R objects) can be obtained using the str() function:

```
str(CITY)
'Data.frame': 6 obs. of 3 variables:
$ City: Factor w/3 levels "City1", "City2", .. 1 1 2 2 3 3
$ Sex: Factor w/2 levels "Female", "Male": 2 1 2 1 2 1
```

$ Number: num 12450 10345 5670 5800 25129 ...

In the presented report, the CITY object is a data table that includes three variables with six observations each. Two of these variables, City and Sex, were automatically recognized by the program as factors with three and two levels, respectively. The Number variable is quantitative. For convenience, the first few values of each variable are also displayed. Often, it is necessary to find out only the names of the variables included in the data table. This can be done using the names () command:

names (CITY)

[1] "City" "Sex" "Number"

It is also possible to quickly view the first few or last few values of each variable in a data table. This is done using the head() and tail() functions, respectively:

head (CITY, n = 3)

City Sex Number

1 City1 Male 12450

2 City1 Female 10345

3 City2 Male 5670

tail (CITY, n = 2)

City Sex Number

5 City3 Male 25129

6 City3 Female 26000

If we need to make corrections to the table, you can use the data editor built into R. Outwardly, this editor resembles a regular Excel sheet, but it has limited functionality. All it allows do is add new or correct already entered variable values, change column headings, and add new rows and columns.

When working in the standard version of R, the data editor can be launched from the "Files>Data Editor" menu, or by executing the fix () command from the R console command line (for example, fix (CITY)). After making corrections, simply close the editor – all changes will be saved automatically.

Often in practice, some values in the table are missing, which can be due to a large number of reasons: at the time of measurement, the device failed, due to the inattention of the staff, the measurement was not entered into the research protocol, the subject refused to answer a certain question in the questionnaire, the sample was lost, etc. Cells with such missing values in R data tables cannot simply be empty – otherwise the columns of the table will be of different lengths. To denote missing observations in the R language, as indicated earlier, there is a special value – NA (not available). If the value NA has the meaning of zero (for example, no specimens of a certain species were found), then this replacement in the table is easy to make with the DF command:

```
DF [is.na (DF)] <- 0
```

Sorting tables

To sort table rows by different keys, the order() function is used:

```
DF <- data.frame (X1 = c (1,15,1,3), X2 = c (1,0,7,0), X3 = c (1,0,1,2),
X4 = c (7,4,41,0), X5 = c (1,0,5,3))
row.names (DF) <- c ( "A", "B", "C", "D")
# DF1 is a table with columns sorted in descending order by sum of values
DF1 <- DF [, rev (order (colSums (DF)))]
# DF2 is a table with rows sorted in ascending order by 1 column, then in descending order by the second
DF2 <- DF [order (DF $ X1, -DF $ X2),]
```

Joining tables

Let's say we have two tables:

DF1				
Y	N	A	B	C
12	22	0	1	0
12	23	1	3	0
12	24	0	0	1

DF2				
Y	N	A	B	D
13	22	0	1	2
13	23	0	3	0
13	24	1	0	5

You can combine their columns using the cbind() function:

```
cbind (DF1, DF2)
```

Y N A B C Y N A B D

1 12 22 0 1 0 13 22 0 1 2

2 12 23 1 3 0 13 23 0 3 0

3 12 24 0 0 1 13 24 1 0 5

To merge rows, we must first convert the tables to a single list of columns:

DF1 [, names (DF2) [! (Names (DF2)% in% names (DF1))]] <- NA

DF2 [, names (DF1) [! (Names (DF1)% in% names (DF2))]] <- NA

rbind (DF1, DF2)

Y N A B C D

1 12 22 0 1 0 NA

2 12 23 1 3 0 NA

3 12 24 0 0 1 NA

4 13 22 0 1 NA 2

5 13 23 0 3 NA 0

6 13 24 1 0 NA 5

We can perform a similar operation using the merge(DF1, DF2, all = TRUE) command. The merge() function allows merge tables using all common SQL join operations.

Conclusion to lecture 11

R is a popular programming language used for statistical computing and data visualization. The most common uses are data analysis and visualization, and machine learning. R provides many statistical methods (statistical tests, classification, data clustering, etc.). The R language runs on various platforms (Windows, Mac, Linux), is open and free, has great community support, and contains many packages (function libraries) that can be used to solve various data analysis tasks. One of R's greatest strengths lies in its extensive ecosystem of packages, which allow users to easily access advanced statistical techniques and cutting-edge machine learning tools.

Whether performing regression analysis, creating complex visualizations, or building predictive models, R provides robust capabilities that suit both beginners and advanced data scientists. Its integration with tools like RStudio and Shiny also makes it ideal for interactive reporting and dashboard creation. Due to its academic roots and strong documentation, R remains a preferred choice in research, healthcare, finance, and other industries that rely heavily on data interpretation and statistical modeling. Its concise syntax, extensive libraries, and supportive community make it an efficient environment for experimenting with new methods and exploring large datasets in a reproducible manner.

Questions for reinforcement

1. What is the history of the R language?
2. What are the capabilities of the R language?
3. What are objects, packages, functions in R?
4. How are vectors and matFigs created in R? How do you perform operations on them in R?
5. What are factors? How are lists, frames, and actions on them defined in R?
6. How does R compare with Python in terms of data analysis?
7. What are some of the most useful R packages for data science?
8. How is data imported and exported in R?

List of recommended literature

1. Microsoft R Application Network. URL: https://mran.microsoft.com/documents/what-is-r
2. R Introduction. URL: https://www.w3schools.com/r/r_intro.asp
3. R Syntax. URL: https://www.w3schools.com/r/r_syntax.asp
4. R Data Structures. URL: https://www.w3schools.com/r/r_variables.asp
5. R Vectors. URL: https://www.w3schools.com/r/r_vectors.asp

Lecture 12.
Exporting, importing and data processing in R

Lecture plan

12.1. Exporting and importing data into R.
12.2. Using R for time series analysis.
12.3. Data processing in R.

12.1. Exporting and importing data into R

Let's look at the most common ways to import data tables into the R workspace, but first, it's worth familiarizing yourself with the rules for preparing the load.

There should be no empty cells in the imported data table; if some values are missing for one reason or another, NA should be entered instead.

It is recommended to convert the imported data table into a simple text file with one of the allowed extensions. In practice, files with the extension *.txt* are usually used, in which the values of the variables are separated by tabs (tab-delimited files), as well as files with the extension *.csv* (comma separated values), in which the values of the variables are separated by commas or another symbol.

It is recommended to enter the variable column headers as the first line of the imported table. If such a line is missing, this must be reported in the description of the command that will control the file loading (for example, **read.table ()**). All subsequent lines of the file can contain row headers (if provided) as the first element, followed by the values of each of the variables in the table.

It is better to assign table column names in accordance with the rules for identifying R variables, i.e., to exclude spaces and other special characters, except for periods and underscores. To avoid problems related to encoding, all text values in imported files should be created using Latin letters.

It is recommended to place the file to be imported in the program's working directory, that is, the folder in which R will "try to find" this file by default. To find

out the path to the R working directory on your computer, use the **getwd()** command (get working directory):

```
getwd()
[1] "C:/Temp/"
```

We can change the working directory using the **setwd()** command (set working directory):

```
setwd("C:/My Documents")
```

When we execute this command, nothing will happen externally, but subsequent use of the **getwd()** command will show that the path to the working directory has changed:

```
getwd()
[1] "C:/My Documents/"
```

Below is a fragment of a typical data table that can be successfully loaded into the R environment for analysis:

```
Group Variable1 Variable2 Variable3
Ivan A 102 1.3 14
Vitaliy A 98 1.4 11
Oleg B 45 NA 8
Mikhail B 50 3.2 6
```

As we can see, the given fragment has dimensions of 5×5, that is, it consists of five rows and five columns. The first row presents the headings of all columns in the table, except for the first. The first column, although it does not have its own heading, is not empty – it contains the names of volunteers who participated in a certain experiment (Ivan, Vitaliy, etc.). The second column has the heading Group and contains labels by which it is possible to find out whether the subjects belong to a particular experimental group (A, B, etc.). In In R terms, the Group variable is called a **factor**. The following columns (with headings Variable1, Variable2, etc.) contain the values of the variables measured during the study. In the table fragment shown, there is one missing value, which is replaced by NA.

One of the most accessible tools for preparing data for further analysis using R is Microsoft Excel. To save Excel tables as txt or csv files, it is usually suggested to use the Save as option in the File section of the main menu of this program. Another simple and reliable way to export data from Excel is to create a new file in Notepad and transfer the entire table or a selected part of it to it via the clipboard.

The main function for importing data into the R workspace is **read.table()**. This powerful function allows customize the process of loading external files, and therefore has a large number of control arguments (Table 12.1.).

Table. 12.1. Control arguments of the read.table () function

Argument	Purpose
file	Used to specify the path to the imported file. The path is given either in absolute form (for example, file = "C: /Temp/MyData.dat"), or only the name of the file being imported is specified (for example, file ="MyData.txt"), but provided that the latter is stored in the working folder of the program. As a name, you can specify a URL link to the file that is supposed to be downloaded from the network (for example: file = "http://somesite.net/YourData.csv"). Starting from version R 2.10, it has become possible to import archived files in zip format.
header	Used to notify the program about the presence of rows with column headers in the download. By default, it takes the value FALSE. If there is a row with column headers, this argument should be assigned the value TRUE.
row.names	Used to specify the number of the column that contains the row names (for example, in the example considered above it was the first column, so row.names = 1). It is important to remember that all row names must be unique, i.e. the same names for two or more rows are not allowed.

sep	allows to specify the separator of variable values used in the file (separator). By default, variable values are assumed to be separated by "empty space", for example, in the form of a space or tab character (sep = ""). In csv format files, variable values are separated by commas, and therefore for them sep = ",".
dec	Used to specify the character used in the file to separate the integer part of a number from the fraction. The default is dec = ".". In many countries, a comma is used, which is important to remember before loading the file and, if necessary, use dec = ",". It is necessary to ensure that dec and sep are not the same.
nrows	Expressed as an integer indicating the number of rows to be read from the loaded table. Negative and other values are ignored. Example: nrows = 100.
skip	Expressed as an integer indicating the number of rows in the file that should be skipped before starting the import. Example: skip = 5

To load the prepared files, it is enough to use the minimum set of arguments of **the read.table ()** function. We need to load the *hydro_chem.txt file*, which is stored in the R working folder and contains data on the chemical composition of the water in the reservoir. We load the data table that we want to save in the object window with the name chem:

```
chem <- read.table (file= "hydro_chem.txt", header= TRUE)
```

Files imported into R are usually in csv format. To load them, we can use the read.table() function, specifying the character used as a separator between variable values in the file (for example, a comma):

```
chem <- read.table (file= "hydro_chem.csv", header= TRUE, sep = ",")
```

The analogue of read.table() for reading csv files is the read.csv() function:

```
chem <- read.csv (file= "hydro_chem.csv", header= TRUE)
```

If the file to be loaded is stored in a folder other than the R working folder, we must specify the full path to it. Windows users should remember that R does not use a single backslash (\) to specify full paths to files, but a single forward slash (/) or double backslash (\\). For example, the following two commands will be successfully accepted by R and will produce the same result – loading the file hydro_chem.txt and saving it as a chem object:

```
chem <- read.csv (file = "D: \\ Documents \\ hydrochem.txt", header = TRUE)
chem <- read.csv (file = "D: /Documents/hydrochem.txt", header = TRUE)
```

To interactively select a download that is stored outside the R working folder, we can use the file.choose() helper function. This command opens a Windows dialog box in which the user selects the folder containing the required file. We can manually combine file.choose() with the read.table() or read.csv() commands, for example [1]:

```
chem <- read.table (file = file.choose (), header = TRUE, sep = ",") .
```

12.2. Using R for time series analysis

In R, there is a special class of objects for working with time series – **ts** (from time series). To create objects of this class, the function of the same name **ts ()** is used. Consider monthly data on birth rates in New York City, collected from January 1946 to December 1959 [2] :

```
birth <-scan ("http://robjhyndman.com/tsdldata/data/nybirths.dat")
```

The birth object is a vector with 168 monthly birth rates (in thousands of people), which can be verified using the function:

```
is.vector (birth)
[1] TRUE
```

The head() function allows view the first few values of the birth vector (by default, the first 6 values):

```
head (birth)
[1] 26,663 23,598 26,931 24,740 25,806 24,364
```

Converting the birth object into a time series is simple:

```
birth.ts <- ts (birth, start = c (1946 1), frequency = 12)
```

The start argument was used to specify the date from which the birth.ts time series begins (1946, 1st month). The additional frequency argument allows specify the increment of subsequent dates – in the example considered, the year is divided into 12 intervals, the increment is 1 month. The birth.ts object created in this way looks like a matrix when viewed. In this case, the rows and columns of this matrix were automatically assigned the appropriate names, based on the values of the start and frequency arguments (data for the Oct, Nov, Dec columns are omitted):

birth.ts

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
1946	26.663	23.598	26.931	24.740	25.806	24.364	24.477	23.901	23.175
1947	21.439	21.089	23.709	21.669	21.752	20.761	23.479	23.824	23.105
1948	21.937	20.035	23.590	21.672	22.222	22.123	23.950	23.504	22.238
1949	21.548	20.000	22.424	20.615	21.761	22.874	24.104	23.748	23.262
1950	22.604	20.894	24.677	23.673	25.320	23.583	24.671	24.454	24.122
1951	23.287	23.049	25.076	24.037	24.430	24.667	26.451	25.618	25.014
1952	23.798	22.270	24.775	22.646	23.988	24.737	26.276	25.816	25.210
1953	24.364	22.644	25.565	24.062	25.431	24.635	27.009	26.606	26.268
1954	24.657	23.304	26.982	26.199	27.210	26.122	26.706	26.878	26.152
1955	24.990	24.239	26.721	23.475	24.767	26.219	28.361	28.599	27.914
1956	26.217	24.218	27.914	26.975	28.527	27.139	28.982	28.169	28.056
1957	26.589	24.848	27.543	26.896	28.878	27.390	28.065	28.141	29.048
1958	27.132	24.924	28.963	26.589	27.931	28.009	29.229	28.759	28.405
1959	26.076	25.286	27.660	25.951	26.398	25.565	28.865	30.000	29.261

The **is.ts()** function allows us to check whether the birth.ts object we created is actually a time series:

is.ts (birth.ts)

[1] TRUE

R has a fairly large set of methods for working with objects of the ts class. In particular, the plot() function can be used to quickly plot a time series graphically:

plot (birth.ts, xlab = "", ylab = "")

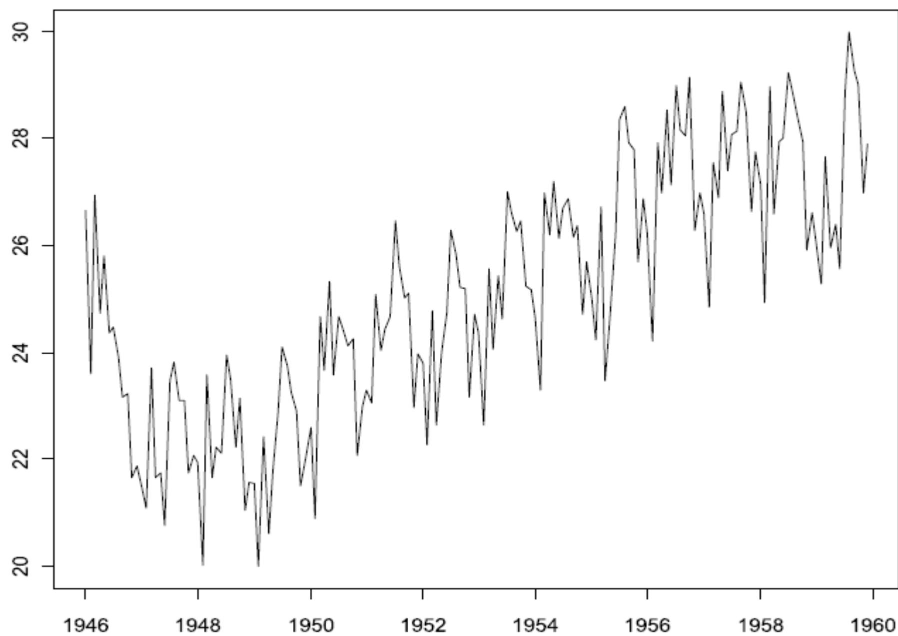


Fig. 12.1. Graphical representation of a time series [2]

12.3. Data processing in R

Most data processing procedures in R are implemented using functions. Functions are named program code consisting of a certain set of variables, constants, operators, and other functions, designed to perform specific operations and tasks. As a rule (but not always), functions return the result of their execution in the form of an R language object – a variable of a certain class: vector, list, table, etc. By their purpose, functions can be divided into characteristic groups: arithmetic, symbolic, statistical, and others. Functions can be built-in (presented in basic or loaded packages) and custom (written directly by users themselves).

Three of the characteristics of R as a high-level language are its modularity, object orientation, and vectorization of computation. Modularity refers to the extensive use of groups of expressions and functions. Expressions expr, consisting of data objects, function calls, and other language operators, can be grouped in curly braces: {expr_1; ...; expr_m}, and the value returned by this group is the result of the last expression. Since such a group is also an expression, it can, for example, be enclosed in parentheses and used as part of a more general expression.

The group of commands below calculates the mean and standard deviation of a natural number sequence from 1 to 10 and returns a vector of these values:

{Aver <- mean (1:10); stdev <- sd (1:10); c (MEAN = aver, SD = stdev)}

MEAN SD

5.50000 3.02765

If this calculation needs to be performed many times for different sets of input data, then it should be formatted as a function. The general syntax for formatting own user function is as follows:

function_name <- function (argl, arg2, ...) {

expression_group

return (object)}

where function_name is the name of the function being created, argl, arg2, ... are the formal arguments of the function. The return () operator is needed in cases where a group of expressions does not return the target result.

Before its first execution, the function must be defined in the current script or loaded using the source() command from a script file where it was previously prepared.

Then the function call can be made as function_name(argl, arg2, ...), where argl, arg2, ... are the actual arguments associated with the formal parameters of the function in the order they follow, or by name.

For the example above, you can write a function:

stat_param <- function (x) {

aver <- mean (x); stdev <- sd (x); c (MEAN = aver, SD = stdev)}

and include it in the collection of custom functions located in the file my_func.R.

Then the result we need, given above, can be obtained by executing

source ("my_func.R")

stat_param (1:10)

The argument list components in a function header can be required or take optional values. For example, the following function raises a numeric object *x* to the power of *n*, but if the power is not specified, it automatically cubes it:

power <- function (x, n = 3) {x ^ n}

Function arguments can be objects of various types, for example, the names of other functions. For example, the following function performs arbitrary transformations of uniformly distributed random variables:

```
my_example <- function (n, func_trans)
{X <- runif (n); abs (func_trans (x))}
```

Branching and loops are widely used in R. The conditional operator has the following structure:

```
if (logical_expression)
{expression_group_1 if logical_expression is TRUE}
else {expression_group_2 otherwise}
```

For example, the following function compares the sizes of two vectors:

```
compare <- function (x, y) {nl <- length (x); n2 <- length (y)
if (nl! = n2) {
if (nl > n2) {z = (nl - n2)
cat ("The first vector has at", z, "elements more than\n")}
else {z = (n2 - nl)
cat ("The second vector has ", z, " elements more than \ n")}}
else {cat("Number of elements is the same", nl, "\n")}
}
x <- c (1:4)
in <- with (1: 9)
compare (x, y)
```

There is also a shortened form of implementing branches:

```
ifelse (logical_expression, expression_group_1, expression_group_2).
```

Repetition of the same computational operations in a loop is carried out using the for (), while () or repeat () constructs, which have the following syntax:

```
for (index in for_object) { group_of_expressions}
while (logical_expression) {expression_group}
repeat {expression_group; break}
```

The for_object object can be a vector, array, table, or list, and the expression_group is executed each time for each element index of that object.

In R, instead of sequentially performing scalar operations on each element of an array, it is much more efficient to perform parallel calculations, in which the program processes the entire array (vector) simultaneously, or several elements of the vector at a time. Obviously, this approach can potentially lead to significant acceleration of the same type of calculations on large data sets.

Let's look at the simplest example of vectorized computation in R. Suppose we have a vector of 10 positive numbers and we want to find the square root of each of them. Instead of writing a loop to perform this operation on each element in turn, this vector is fed to the sqrt() function, which returns a vector with the results of the computation:

```
x <- 1:10
sqrt(x)
[1] 1,000 1,414 1,732 2,000 2,236 2,449 2,645 2,828 3,000 3,162
```

The principle of vectorized calculations can be applied not only to vectors, but also to more complex R objects – matFigs, lists and data tables (for R, there is no difference between the last two types of objects: in fact, a data table is a list of several components – vectors of the same size).

Family of functions designed to organize vectorized calculations on such objects. The name of these functions contains the word apply, preceded by a letter indicating the principle of operation of a particular function, while.

Apply(), unlike for(), can be easily parallelized (by renaming the function to its parallel version from the snow package);

Algorithms written without loops are easier to modify, contain fewer errors, and are easier to type on the command line.

The result of apply() can be a function argument, any function can be inserted as an argument into apply().

Apply() function is used in cases where it is necessary to apply a function to all rows or columns of a matrix (or larger-dimensional arrays):

apply (x, MARGIN, FUN, ...)

where x is the transformed object, MARGIN is an index indicating the direction of the calculation process (column or row), FUN is the function used for the calculation, ... are any other parameters of the applied function. For a matrix or data table, MARGIN = 1 denotes rows, and MARGIN = 2 denotes columns. Since FUN denotes any R function, the apply() function is a powerful tool for modular data processing. Let's create a two-dimensional matrix:

M <- matrix (seq (1,16), 4, 4)

Let's find the minimum values in each row of the matrix:

apply(M, 1, min)

[1] 1 2 3 4

Let's find the minimum values in each column of the matrix:

apply(M, 2, max)

[1] 4 8 12 16

Example with a three-dimensional array:

M <- array (seq (32), dim = c (4,4,2))

Let's apply the sum() function to each element of M[*,,], i.e., we will perform the summation over dimensions 2 and 3:

apply(M, 1, sum)

The result is a one-dimensional vector:

[1] 120 128 136 144

Let's apply the sum() function to each element of M [*, *,], that is, we will perform the summation along the third dimension:

apply(M, c(1,2), sum)

The result is a matrix:

[, 1] [2] [3] [4]

[1,] 18 26 34 42

[2,] 20 28 36 44

[3,] 22 30 38 46

[4,] 24 32 40 48

If it is necessary to calculate sums and averages by rows or columns of matFigs, it is recommended to use the fast and specially optimized functions colSums(), rowSums(), colMeans() and rowMeans().

Lapply() function is used in cases where it is necessary to apply a function to each component of a list and obtain the result in the form of a list (the letter "l" in the name lapply() stands for list).

Sapply() function is used in cases where it is necessary to apply a certain function to each component of a list, but the result should be output as a vector (the letter "s" in the name sapply() means simplify).

A list of three components:

x <- list (a = 1, b = 1: 3, c = 10: 100)

Let's find out the size of each component of the list x:

sapply(x, FUN = length)

abc

1 3 91 # result returned as a vector

Summing all elements in each component of list x:

sapply(x, FUN = sum)

abc

1 6 5005 # result returned as a vector.

Replicate() function is a kind of "wrapper" for the sapply() function and allows perform calculations to generate a set of numbers according to a given algorithm. The function syntax is as follows:

replicate (n, expr, simplify = TRUE)

where n is the number of repetitions, expr is a function or group of expressions that must be repeated n times, simplify = TRUE is an optional parameter that simplifies the result and represents it as a vector or matrix of values.

Vapply() function is similar to sapply(), but is faster because the user explicitly specifies the type of values to be returned (the "v" in the name vapply() stands for velocity). This approach avoids the error messages (and aborts) that occur with sapply() in some situations.

Mapply() function is used when it is necessary to apply any function element by element to several objects simultaneously. The result is returned as a vector or array of a different dimension. The letter "m" in the name mapply() stands for multivariate (simultaneous calculation of elements of several objects).

mapply (sum, 1: 5, 1: 5, 1: 5)

[1] 3 6 9 12 15

Rapply() function is used in cases where it is necessary to apply a function to the components of a nested list (the letter "r" in the name rapply() stands for recursively).

Tapply() function is used in cases where it is necessary to apply any function fun to separate groups of elements of the vector x, specified according to the levels of any factor group:

tapply (x, group, fun, ...)

In the following example, the **sample()** function is used to create two random samples: one of 50 integer values from 1 to 4 and their associated labels of the four AD groups.

The tapply() function calculates the sum of x for each of the factor values:

x <- sample (1: 4, size = 50, replace = T)

gr <- as.factor (sample (c ("A", "B", "C", "D"), size = 50, replace = T))

tapply(x, gr, sum)

A B C D

29 25 30 37

By() function is analogous to the tapply() function for tables. The data table is divided into subsets of subtables based on a given group column, and a fun function is defined to process each part:

by (data, group, fun, ...)

The function allows perform the combinatorial operation fun on the elements of two arrays or vectors x and y without resorting to the explicit use of a "double" loop:

outer (x, y, fun = "*", ...)

By default, the pairwise multiplication operation is performed:

```
x <- 1: 5; y <- 1: 5
outer (x, y)
  [, 1] [2] [3] [4] [5]
[1,] 1   2   3  4   5
[2,] 2   4   6  8   10
[3,] 3   6   9  12  15
[4,] 4   8   12  16  20
[5,] 5  10   15  20 25
```

Using the outer() function together with the paste() function, we can generate all possible pairwise combinations of "connections" of the elements of a character and integer vector:

```
x <- з ( "A", "B", "C", "D")
y <- 1:10
outer (x, y, paste, sep = "")
      [, 1]   [2]   [3]   [4]   [5]    [6]    [7]    [8]    [9]   [, 10]
[1,] "A1" "A2" "A3" "A4" "A5" "A6" "A7" "A8" "A9" "A10"
[2,] "B1" "B2" "B3" "B4" "B5" "B6" "B7" "B8" "B9" "B10"
[3,] "C1" "C2" "C3" "C4" "C5" "C6" "C7" "C8" "C9" "C10"
[4,] "Dl" "D2" "D3" "D4" "D5" "D6" "D7" "D8" "D9" "D10".
```

Conclusion to lecture 12

The main function for importing data into the R workspace is the read.table() function, which allows customize the process of loading external files using control arguments. In R, there is a special ts object class for working with time series, and the ts() function is used to create objects of this class.

Most data processing procedures in R are implemented using functions. The characteristic features of the R language are modularity, object orientation, and vectorization of computations.

The R language allows parallelize operations to speed up calculations. The **parallel computing** package allows execute tasks on different cores of the machine.

Vectors and matrices are fundamental data structures in R that enable efficient data storage and manipulation. Vectors in R are one-dimensional arrays that hold elements of the same data type (numeric, character, logical, etc.). They are essential for performing element-wise operations, filtering data, and applying functions across datasets. Due to R's vectorized operations, working with vectors is not only intuitive but also highly efficient, making them a core component of statistical computations.

Matrices, on the other hand, are two-dimensional extensions of vectors and are particularly useful in linear algebra and advanced statistical modeling. In R, matrices are created using the matrix() function and support a wide range of mathematical operations including matrix multiplication, transposition, and inversion.

Questions for reinforcement

1. How to export and import data in R?
2. What R language tools are used for time series analysis?
3. How is data processing done in R?
4. What are the apply (), lapply (), sapply (), replicate (), mapply (), rapply (), tapply (), sample (), by (), outer () functions used for ?
5. How does indexing work with vectors in R?

List of recommended literature

1. R Lists. URL: https://www.w3schools.com/r/r_lists.asp
2. R Matrices. URL: https://www.w3schools.com/r/r_matrices.asp
3. R Arrays. URL: https://www.w3schools.com/r/r_arrays.asp
4. R Data Frames. URL: https://www.w3schools.com/r/r_data_frames.asp
5. R Factors. URL: https://www.w3schools.com/r/r_factors.asp

Lecture 13.
Basic tools for data analysis and visualization in R

Lecture plan

13.1. The plot() function and its parameters.
13.2. Managing common parameters - arguments of graphical functions.
13.3. Types of graphs in R.

13.1. The plot() function and its parameters

Graphs are an integral part of data mining, as they allow identify patterns and trends in complex data sets, and can also be the result of statistical analysis (for example, when building classification trees). Typically, creating a graph begins with a high-level function that defines its overall structure: dimensionality (1D, 2D, 3D), axis scales, titles, etc.

The most commonly used high-level graphing functions are plot(), hist(), boxplot(), scatterplot(), and pairs(). Using a rich set of low-level functions, additional elements can be added to the constructed graph: text, lines, legend, etc. (Examples of such functions are lines(), points(), text(), and axis()).

The features of the details of a graphic image are controlled by a set of parameters, usually common to most high-level and low-level functions, which determine the color, types and sizes of symbols or markers, the thickness and nature of lines, hatching, the graphic frame, etc.

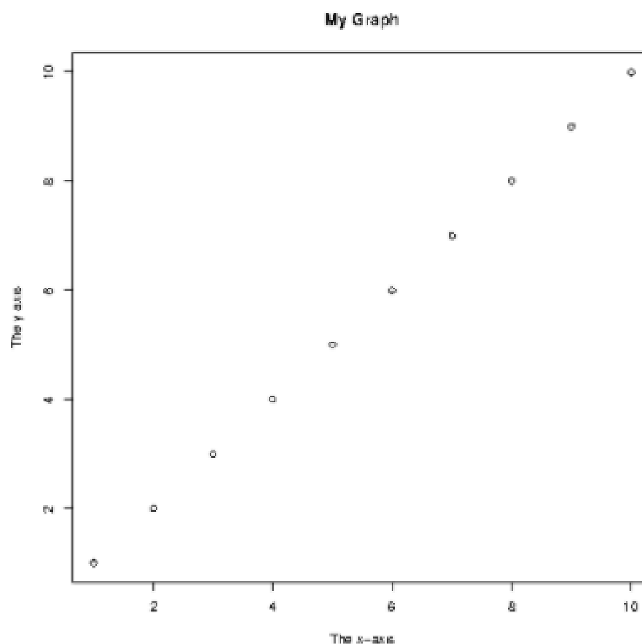
Plot() function is the main workhorse used for plotting in R. The behavior of this high-level function is determined by the class of objects that are specified as its arguments. Accordingly, plot() can be used to create a wide range of different types of plots.

The plot() function has a large number of control parameters that allow fine-tune the appearance of the graph.

1. Parameters xlab and ylab

The **xlab** and **ylab** parameters are used to change the names of the X and Y axes, respectively.

```
# We need this line of code to show graphs in our compiler
bitmap(file="out.png")

plot(1:10, main="My Graph", xlab="The x-axis", ylab="The y axis")
```



2. Type parameter

Type parameter allows change the appearance of the points on the graph. It takes one of the following values:

◦ "p" – points (used by default);

◦ "l" – lines (lines);

◦ "b" – both points and lines are displayed;

◦ "o" – points are displayed over lines (points over lines);

◦ "h" – histogram (histogram);

◦ "s" – steps (steps);

◦ "n" – no points are displayed (no points).

3. Parameters xlim and ylim

These two parameters control the range of values on each of the axes of the graph. By default, they both take the value NULL – in this case, the range is selected

automatically by the program. To cancel the automatic settings, the corresponding parameter must be assigned a value in the form of a numeric vector containing the minimum and maximum values that should be displayed on the axis. For example:

plot(indo.times, means, xlab = "Time", ylab = "Concentration", xlim = c(0, 15))

plot(indo.times, means, xlab = "Time", ylab = "Concentration", ylim = c(0, 5))

4. Parameters axes and ann

These two parameters control the display of axes and their names, respectively. Each of them can take one of two possible values – TRUE or FALSE:

plot (indo.times, means, xlab = "Time", ylab = "Concentration", axes = TRUE, ann = TRUE)

plot (indo.times, means, xlab = "Time", ylab = "Concentration", axes = FALSE, ann = TRUE)

plot (indo.times, means, xlab = "Time", ylab = "Concentration", axes = TRUE, ann = FALSE)

5. Log parameter

Using the log argument, we can convert one or both axes of the graph to a logarithmic scale, for example:

plot (indo.times, means, xlab = "Time", ylab = "Concentration", log = "x")

plot (indo.times, means, xlab = "Time", ylab = "Concentration", log = "y")

plot (indo.times, means, xlab = "Time", ylab = "Concentration", log = "xy")

6. Main parameter

The main argument is used to create the title of the graph. By default, the title is placed at the top of the figure:

plot (indo.times, means, xlab = "Time", ylab = "Concentration", main = "Indomethacin elimination rate", type = "o")

13.2. Managing common parameters - arguments of graphical functions

Consider the graphical parameters that control the appearance of graphs, for example, the type, size, and color of symbols and lines, the type and size of the font in the titles of the graph and its axes, the use of mathematical symbols in the titles,

the placement of the legend, etc. They are used as arguments not only when calling plot (), but also many other functions.

1. Character type

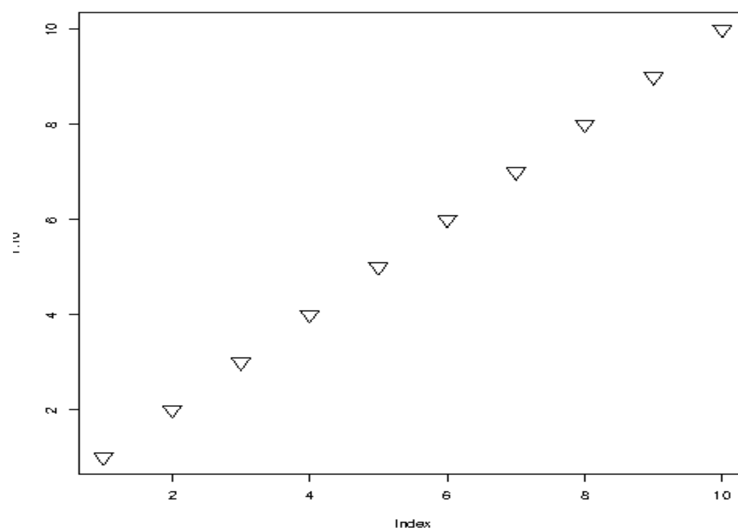
Pch (plotting character) argument allows change the type of characters used to display observations. In standard cases, this argument takes numeric values from 1 to 25 (Fig. 13.1).

0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24 25

Fig. 13.1. Table of 25 standard markers and their corresponding numerical codes

For example, at pch = 25 the characters will turn into filled triangles:

```
plot(1:10, pch=25, cex=2)
```



2. Marker size

The size of the markers is set using the **cex** (character extension) argument, which defaults to 1. Increasing or decreasing this parameter causes the corresponding proportional changes in the size of the characters. If necessary, we

can also change the width of the character's stroke line. This is done using the **lwd** (line width) parameter.

3. Marker color

The color of any graphic object can be specified in several ways:

◦ by color name: for example, col = "red", col = "green", or col = "black".

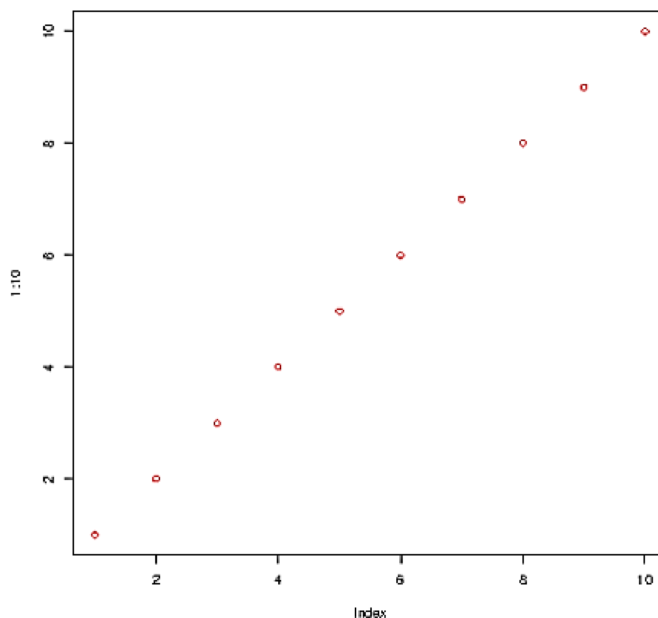
There are a total of 675 standard colors in R. Their names are available with the colors() command;

◦ by directly specifying the red, green and blue components of the RGB spectrum, for example: "#RRGGBB";

◦ by numerical code , for example: col = 2 (red), col = 3 (green), or col = 1 (black).

The color of the markers is set using the **col** (color) argument.

```
plot(1:10, col="red")
```



To plot a line, use the plot() function and the **type parameter** with the value "l".

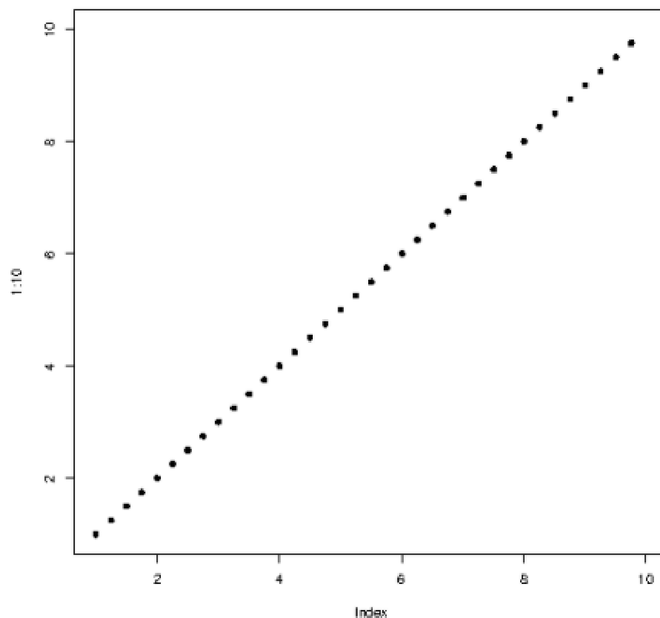
4. Line width

The line width is specified using the **lwd** (line width) argument of the plot() function. To change the line width, use the **lwd** parameter (1 is the default, while 0.5 means 50% less and 2 means 100% more).

The line is solid by default. To specify the line format, use the **lty** parameter with a value from 0 to 6.

For example, if lty=3, a dotted line will be displayed instead of a solid line:

```
plot(1:10, type="l", lwd=5, lty=3)
```



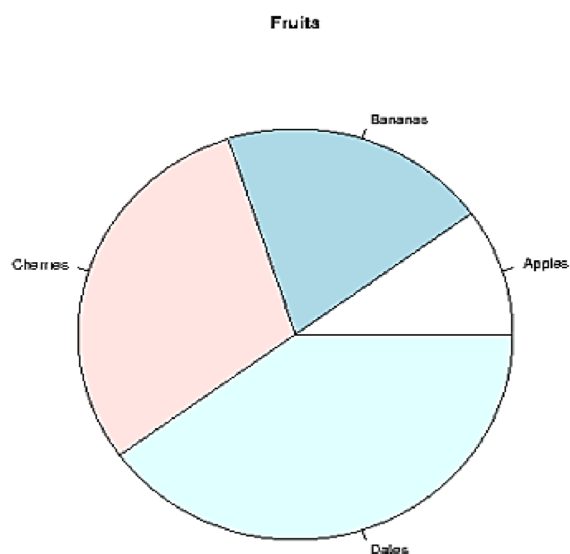
13.3. Types of graphs in R

Pie charts are created using the **pie()** function. The **label** parameter is used to add a label to the chart, and the **main** parameter is used to add a title.

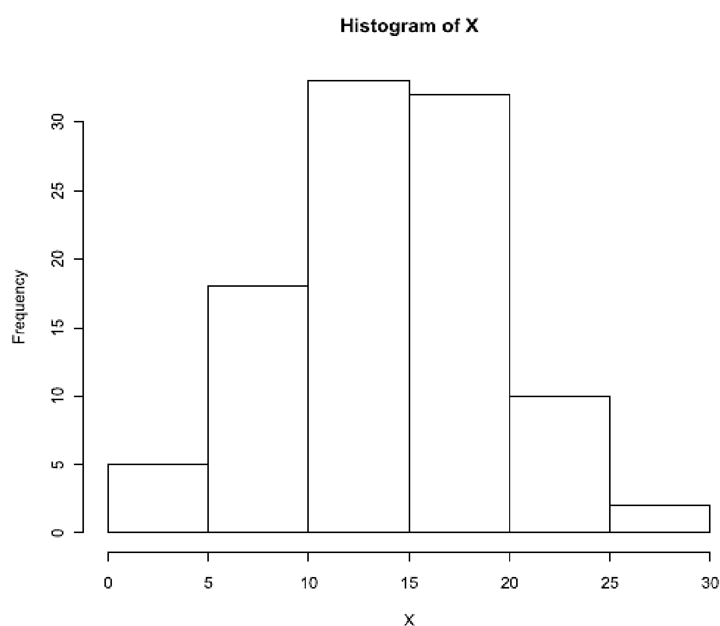
```
# Create a vector of pies
x <- c(10,20,30,40)

# Create a vector of labels
mylabel <- c("Apples", "Bananas", "Cherries", "Dates")

# Display the pie chart with labels
pie(x, label = mylabel, main = "Fruits")
```

To add a list of explanations for each sector, the **legend()** function is used.

```
# Create a vector of labels
mylabel <- c("Apples", "Bananas", "Cherries", "Dates")

# Create a vector of colors
colors <- c("blue", "yellow", "green", "black")

# Display the pie chart with colors
pie(x, label = mylabel, main = "Pie Chart", col = colors)

# Display the explanation box
legend("bottomright", mylabel, fill = colors)
```



A histogram allows visually present the distribution of values of the analyzed variable. In the R, **hist()** function is used to construct histograms. Its main argument is the name of the analyzed variable.



As an example, let's create a normally distributed population X of 100 observations with a mean of 15 and a standard deviation of 5:

X <- rnorm (n = 100, mean = 15, sd = 5)

Rnorm() (from random and norm) is used to create the variable X. Using a random number generator, this function generates normally distributed populations with a given size (n), mean (mean), and standard deviation (sd).

It is simple to display the value of the variable X in the form of a histogram:

history (X)

The hist() function automatically selects the number of columns to display on the graph, creates axis names and a graph title. Such a drawing obtained using automatic settings may be quite sufficient (for example, when conducting a quick exploratory data analysis). It often requires additional refinement. First of all, it is important to pay attention to the step size for dividing the data into classes when constructing a histogram. In the example above, the program automatically divided the values of the variable X into 6 classes. Such a rough division may not accurately reflect the properties of the analyzed population. To study these properties in more detail, we can increase the division of the data into classes (use a smaller class interval). This is done using the breaks argument of the hist() function.

If necessary, the histogram columns can be filled with the desired color. To do this, use the col argument of the plot() function. In the example above, the light blue color of the columns ("lightblue") is selected:

hist (X, breaks = 20, col = "lightblue")

As we can see, the result of executing the previous command was a histogram with twenty columns, which allows analyze the distribution of the values of the variable X in more detail.

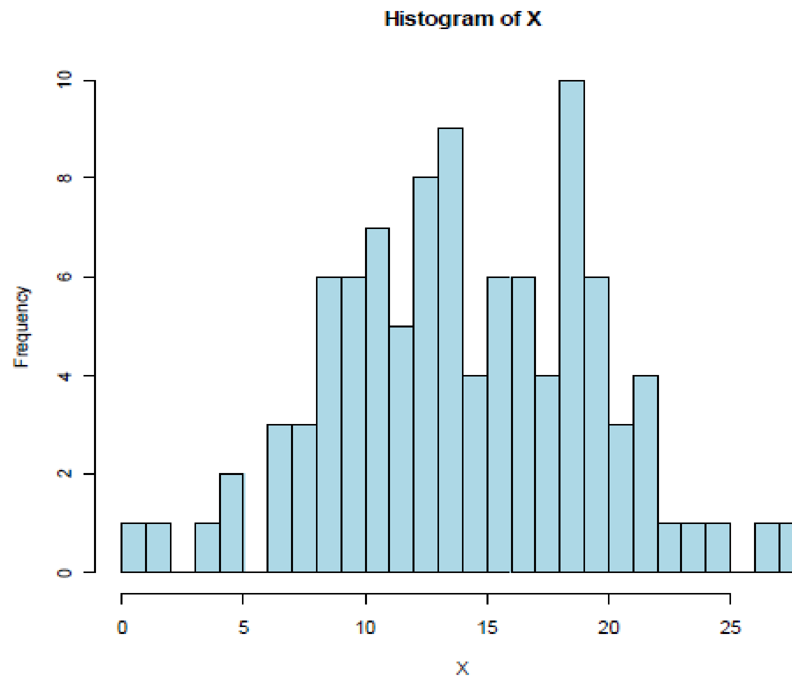
Histogram of X

X

Frequency

0 5 10 15 20 25

0 2 4 6 8 10



By default, the hist() function displays the frequency for each class of X values on the ordinate axis. This behavior can be changed by setting the freq argument (from frequency) to FALSE. In this case, the ordinate axis will display the probability density of each class so that the total area under the histogram is 1:

```
hist (X, breaks = 20, freq = FALSE)
```

In some cases, particularly with a small number of observations, histograms can give an incorrect idea of the properties of the population, for example, due to a small number of sparsely spaced columns:

```
X <- rnorm (n = 50, mean = 15, sd = 5)
hist (X, breaks = 20, freq = FALSE, col = "lightblue").
```

Instead of a histogram (or in parallel with it), it is recommended to use a probability density curve in such cases. The probability density is estimated using the density() function, which can be used as an argument to the plot() function to graphically display the result:

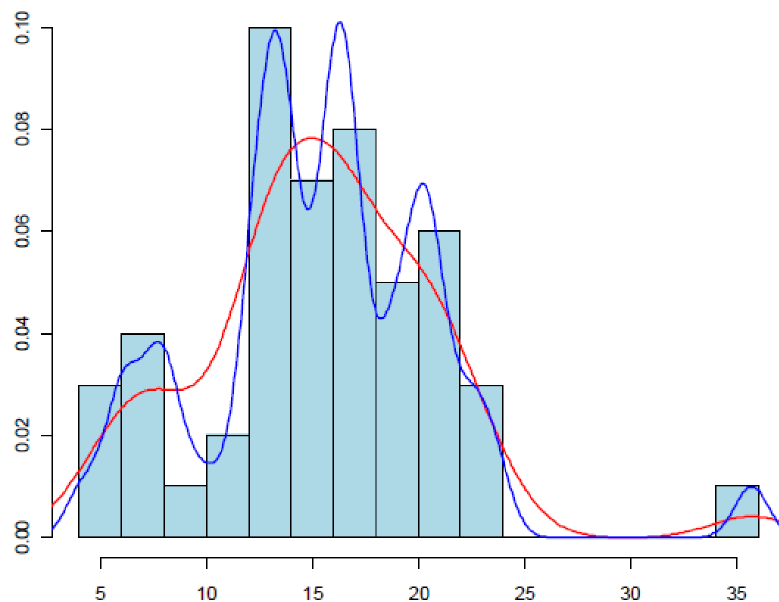
```
plot(density(X))
```

The smoothness of the resulting curve is adjusted using the bw argument (from bandwidth - window width), for example:

```
plot(density(X, bw = 0.8)).
```

For completeness, the histogram can be combined with a probability density curve. In this case, first you need to construct the histogram itself, and then add a density curve to it using the lines() function:

```
hist (X, breaks = 20, freq = FALSE, col = "lightblue", xlab = "", ylab = "")
lines (density (X), col = "red", lwd = 2)
lines (density (X, bw = 0.8), col = "blue", lwd = 2)
```



The col (to set the line color) and lwd (to set the line thickness) arguments were used as control arguments for the lines() function.

The examples above show the versatility of a number of other arguments that control the behavior of plot(), hist(), and other high-level R graphics functions.

Conclusion to lecture 13

The most commonly used high-level graphical functions in R are plot(), hist(), boxplot(), scatterplot(), and pairs(), using low-level functions, additional elements (text, lines, legend, etc.) can be added to the constructed graph. The features of the details of the graphical image are controlled by a set of parameters common to most functions, which determine the color, type and size of symbols or markers, the thickness and character of lines, hatching, the frame of the graph, etc.

The plot() function is used to plot graphs in R. The behavior of this function is determined by the class of objects that are specified as its arguments. Using plot(),

you can create a large set of different types of graphs. The plot() function has a large number of control parameters that allow customize the appearance of the graph.

Data visualization in R is a powerful aspect of the language, enabling users to explore, understand, and communicate data effectively. R offers a variety of built-in functions for creating basic charts such as histograms, bar plots, scatterplots, and boxplots. R provides high-level libraries like ggplot2, which allows for the construction of complex and aesthetically appealing visualizations using a layered grammar of graphics. Visualization in R is highly customizable, allowing control over every aspect of the plot — from colors and labels to scales and themes.

One of the strengths of visualization in R is its integration with statistical analysis, making it possible to easily overlay models, trends, and summaries onto plots. This enables users not only to observe patterns but also to validate hypotheses and present findings in a meaningful way. With additional packages like plotly, R also supports interactive graphics, enhancing user engagement and enabling dynamic data exploration.

Questions for reinforcement

1. What is the plot() function used for? What are its parameters?
2. Name the common parameters – arguments of graphical functions.
3. What types of graphs in R do you know?
4. What functions are used to label axes and titles in R plots?
5. How to create a histogram in R and customize its appearance?

List of recommended literature

1. R Plotting. URL: https://www.w3schools.com/r/r_graph_plot.asp
2. R Line. URL: https://www.w3schools.com/r/r_graph_line.asp
3. R Scatter Plot. URL: https://www.w3schools.com/r/r_graph_scatterplot.asp
4. R Pie Charts. URL: https://www.w3schools.com/r/r_graph_pie.asp
5. R Bar Charts. URL: https://www.w3schools.com/r/r_graph_bars.asp

Lecture 14.
Big Data architectural models. Virtualization technologies. Hypervisors. Container technology for executing program code on the server. SaaS, PaaS and IaaS

Lecture plan

14.1. Architectural models of Big Data engineering.
14.2. Data centers and cloud computing.
14.3. Virtualization technologies.
14.4. Layers of abstraction.
14.5. Hypervisors.
14.6. Container technology for executing program code on the server.
14.7. Data engineering.

14.1. Architectural models of Big Data engineering

Transforming data into valuable information requires computing power and memory. Different IoT architectures have different approaches to where and when data is processed and stored (Fig. 14.1).

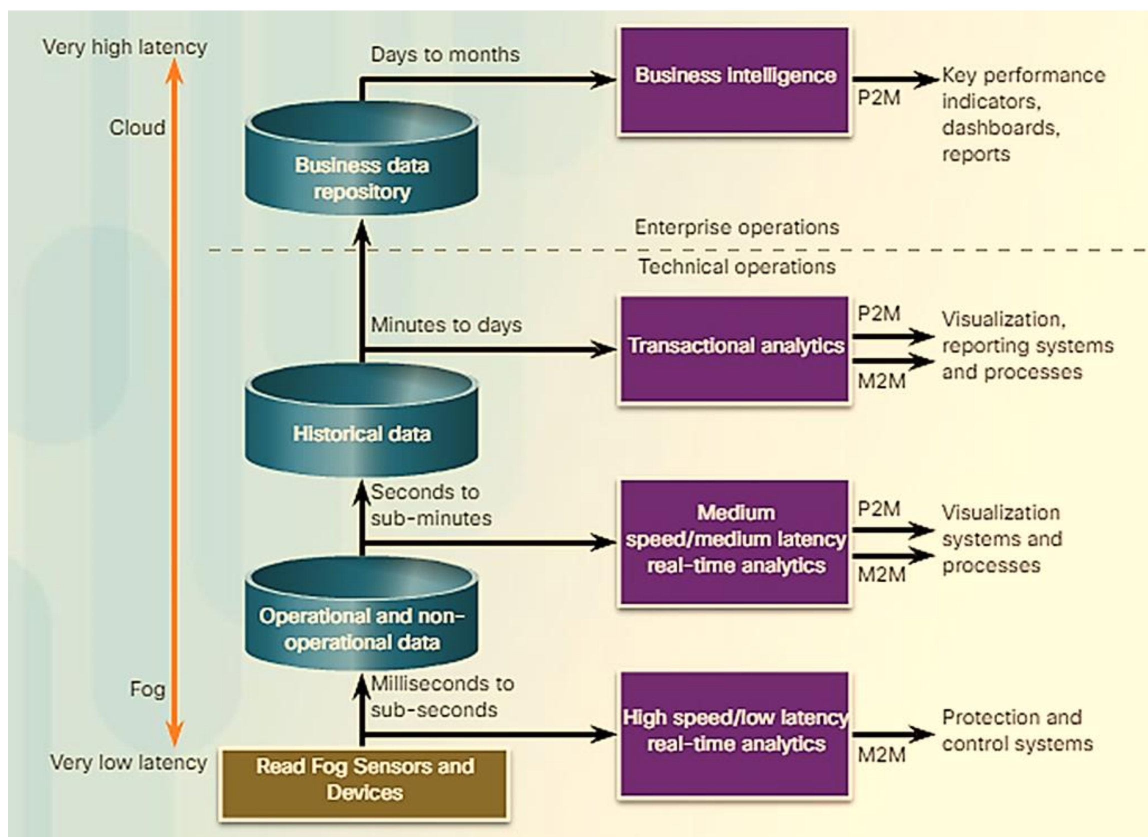


Fig. 14.1. Device-Network-Cloud architectural model [1]

For example, in the Device-Network-Cloud architectural model, all data points collected by sensors included in a connected device are sent directly to the cloud for storage and processing.

For example, data collected by a fitness tracker is sent to the cloud. There, it is transformed using descriptive analytics and presented to the user in a web profile. This architectural model is simple but does not scale. When the number of sensors increases along with the number of data points generated, or when data processing requires significantly shorter response times, these are situations where data needs to be processed closer to where it was generated. This is where the Device-Gateway-Network-Cloud architecture is used. Depending on the application, data can be processed almost immediately after it is generated, close to its source at the gateway or other intermediate locations in the network (fog computing). This source area is also known as the edge, and for this reason this approach is also called edge analytics. Examples of applications that require fog computing are sensor networks that are geographically distributed, such as soil moisture sensors in a vineyard and traffic sensors at every intersection in a city. Fog computing helps to reduce response time (low latency) and reduce the amount of data that needs to be sent to the cloud. For example, fog computing removes redundant data when a sensor variable does not change, because it is not rational to keep sending the same values. Instead, data is sent only when its values change.

Regardless of the IoT architecture used, most or all of the data will eventually be collected and stored in the cloud, where computing power and storage capacity are available. Data center networks use virtualization technology.

Data centers have thousands (or hundreds of thousands) of servers available, as well as storage capacity for data over high-speed network connections. Virtualization technology allows one or more virtual machines to be created inside each physical server, where the data analysis process can run. Cloud platform providers have a network of fault data centers.

Data moves across network infrastructure. Organizations depend on their IT operations. The ability of the infrastructure to quickly make resources available

directly impacts the speed of data transfer. Leveraging big data to gain insights and business insights requires powerful solutions such as data centers (DPCs). For example, as organizations grow, they require increasing amounts of computing power and hard drive space. If left unchecked, this will negatively impact an organization’s ability to deliver critical services. Loss of critical services means lower customer satisfaction, lower revenue, and in some cases, loss of assets.

Large enterprises typically own data centers to manage their storage and data access needs. In a data center, a single enterprise tenant is the sole customer using the data center services. As data volumes continue to expand, even large enterprises are expanding their data storage capabilities by using data center services that can be used to meet internal IT needs (private cloud). Data centers can also offer these same goods and services to other companies and organizations (public cloud).

14.2. Data centers and cloud computing

To help address the four Vs of Big Data (volume, variety, velocity, and veracity), many organizations are turning to cloud computing. Cloud computing supports a variety of data management challenges:

- access to organization data anywhere and at any time;
- streamlining an organization's IT operations, subscribing only to the services you need;
- reducing the need for IT equipment, maintenance and on-site management;
- reduction of equipment costs, energy, physical requirements for installations and personnel training needs;
- rapid response to increasing data volume needs;
- computing and storage costs, rather than investing in infrastructure, capital expenditures are converted into operating expenses.

The three main cloud computing services defined by the National Institute of Standards and Technology (NIST) in Special Publication 800-145 are:

- **SaaS** – Software as a Service;
- **PaaS** – Platform as a Service;

- **IaaS** – Infrastructure as a Service.

Cloud service providers have expanded this model to provide IT support for each of the cloud computing services (ITaaS).

There are currently over 3,000 data centers worldwide that offer general data hosting and processing services to organizations. There are many more data centers that are owned and operated by the private industry for their own use.

Data centers are centralized locations that house a large amount of computing and networking equipment. This equipment is used to collect, store, process, distribute, and provide access to vast amounts of data. Its primary function is to ensure business continuity by keeping computing services available when and where they are needed.

Virtually every organization needs its own data center or access to a data center. Some organizations build and maintain their own data centers. Other organizations rent servers in co-location locations. Still others use public, cloud-based services. Amazon Web Services, Microsoft Azure, Rackspace, and Google are examples of companies that provide public cloud services.

Due to the operational complexity of data centers, a few organizations manage their own data centers. Therefore, many organizations lease space in specialized data centers owned by service providers to house their systems.

14.3. Virtualization technologies

Operating systems (OS) separate programs from hardware. OSes create an "abstraction" of the details of a program's hardware resources.

Virtualization separates the OS from the hardware.

Cloud providers offer services that can dynamically provision servers as needed. Server virtualization takes advantage of idle resources on a physical machine and consolidates multiple virtual servers onto a single machine. It also allows multiple operating systems to be run on a single hardware platform. For example, in Fig. 14.2, the original eight dedicated servers were combined into two servers using hypervisors to support multiple virtual OS instances.

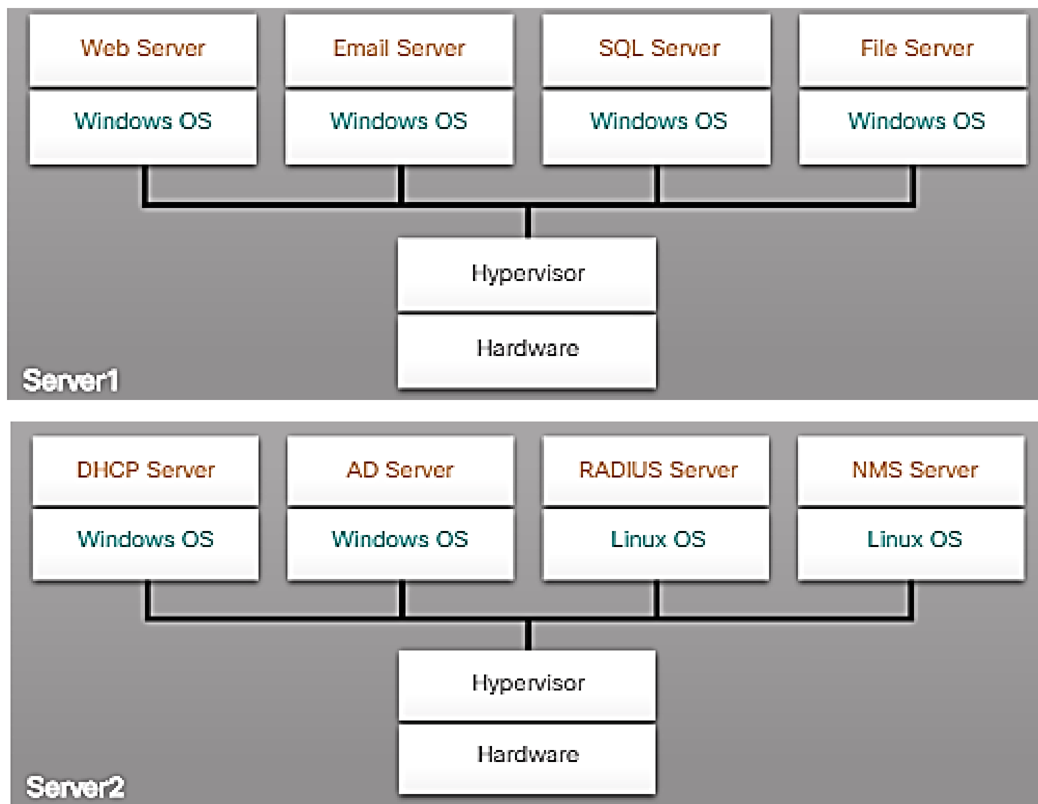


Fig. 14.2. Installing the OS hypervisor [1]

Hypervisor is a program, firmware, or hardware that adds a layer of abstraction on top of the actual physical hardware.

Abstraction layer is used to create virtual machines that have access to all the hardware of a physical machine, such as processors, memory, disk controllers, and NICs. Each of these virtual machines runs a separate operating system. Through virtualization, enterprises can consolidate the number of servers they own and operate. For example, it is not uncommon for 100 physical servers to be consolidated as virtual machines on top of 10 physical servers using hypervisors.

Virtualization uses typically include redundancy to protect against a single point of failure. Redundancy can be implemented in a variety of ways. If a hypervisor fails, the VM can be restarted on another hypervisor.

Additionally, the same VM can run on two hypervisors simultaneously, copying RAM and CPU instructions between them. If one hypervisor fails, the VM continues to run on the other hypervisor.

14.4. Layers of abstraction

To explain how virtualization works, it is useful to use the abstraction layers in computer architectures. The abstraction layers (programs, operating systems, hardware) are also used by the OSI reference model to help describe network protocols (Fig. 14.3). The diagram illustrates the layered architecture of a computer system, divided into three main levels: User Mode, Kernel Mode, and Hardware.

User Mode includes user programs, application programs, and system programs that interact with the system through libraries.

Kernel Mode contains the operating system, which manages resources and provides core system services.

Hardware layer includes the CPU, main memory, and peripheral devices, all interconnected via address, data, and control buses.

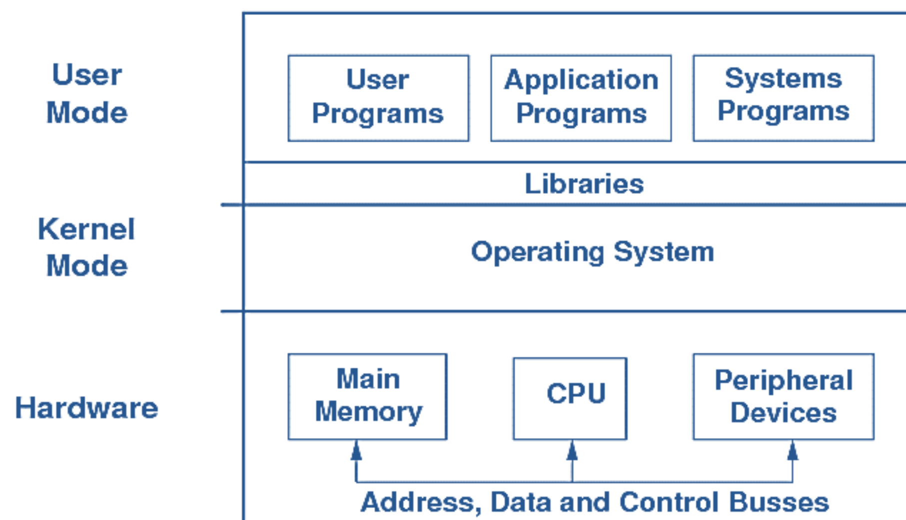


Fig. 14.3. Layers of abstraction of a computer system (programs, OS, hardware) [1]

At each of these abstraction layers, some type of programming code is used as an interface between the layer below and the layer above. For example, the C programming language is often used to program firmware that accesses hardware.

The hypervisor is installed between the firmware and the OS. The hypervisor can support multiple instances of the OS.

14.5. Hypervisors

Hypervisor is software that creates and runs VM instances. The computer on which the hypervisor supports one or more VMs is the host machine.

There are two types of hypervisors.

Type 1 hypervisor – is a "bare metal" approach, as the hypervisor is installed directly on the hardware (Fig. 14.4). Type 1 hypervisors are typically used on enterprise servers.

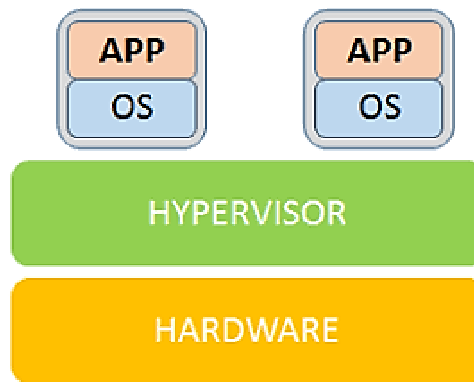


Fig. 14.4. Type 1 hypervisor [1]

Type 2 hypervisor – is a "hosting" approach. Type 2 hypervisor adds an additional layer of abstraction. This is because the hypervisor is a program that runs on the physical host OS, and additional instances of the OS are installed in the hypervisor (Fig. 14.5).

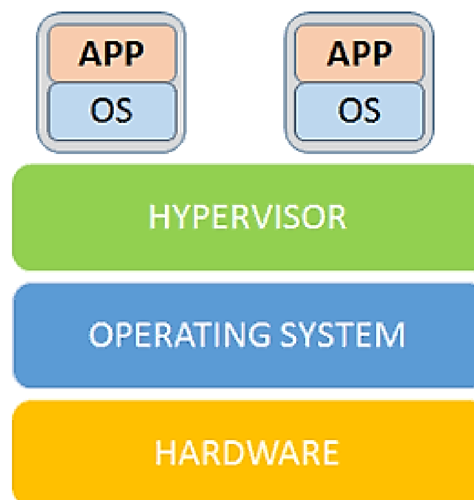


Fig. 14.5. Type 2 hypervisor [1]

14.6. Container technology for executing program code on the server

Hypervisors allow each virtual machine to have its own operating system while sharing the same hardware. This configuration is useless if the operating systems used in the virtual machines are the same as the operating system running on the host computer. Containers solve this problem.

A container is a specialized "virtual space" where applications can run independently of each other while sharing the same OS and hardware. From the application's perspective, it is the only application running on the computer.

By sharing with the host operating system, most of the program's resources are reused, which optimizes performance.

A container only needs the necessary part of the operating system, system resources, and any programs and libraries needed to run the application. This allows the server to support many more containers than a virtual machine could at any one time (Fig. 14.6).

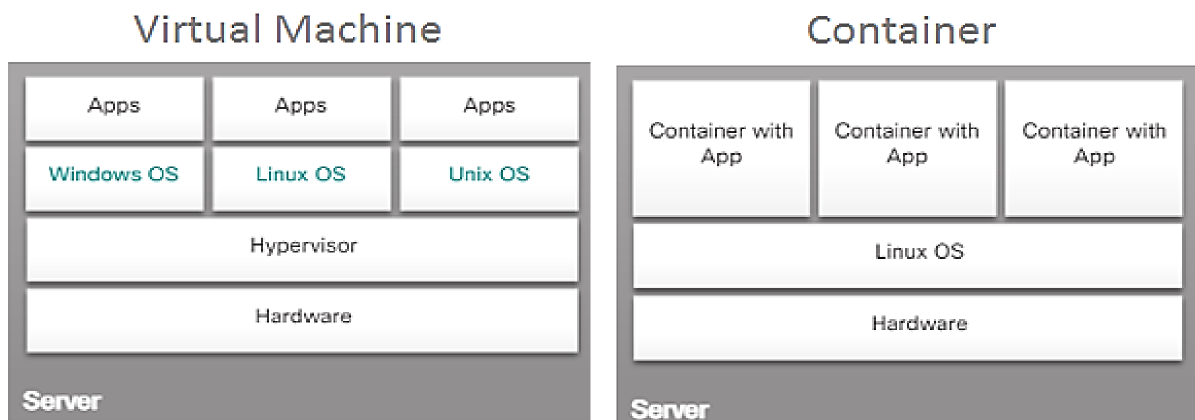


Fig. 14.6. Container structure compared to a traditional hypervisor [1]

More applications can be run on a server using container technology, containers require the operating system of the virtual machine to be the same as the host computer. If there is a need for multiple operating systems, hypervisors should be used.

Data centers can also use virtualization to reduce costs and expand cloud provider offerings.

Amazon Web Services (AWS) is a cloud service provider that offers computing resources and services on demand in the cloud. This means that we can run on demand on one or more virtual servers on AWS when we need them for a specific period of time.

AWS can be used to store data, host websites and web applications, host a Learning Management System (LMS), and process big data generated by IoT. This is an example of IaaS (Infrastructure as a Service), where building a computing infrastructure is converted into purchasing a service. Amazon machine learning allows developers to build applications for fraud detection, demand forecasting, targeted marketing, and click prediction. Amazon's machine learning algorithms create machine learning (ML) models by finding patterns in existing data. These models then process new data and generate predictions. One possible application of an ML model is to predict how likely a customer is to purchase a particular product based on their past behavior. There are many cloud providers. In North America, these include Microsoft Azure and Google Cloud. In Europe, some cloud providers include Aruba Cloud, UpCloud, and CenturyLink.

Storage virtualization combines physical storage from multiple network storage devices into a single storage device. The storage device is managed from a central console. Storage virtualization makes backup, archiving, and recovery easier and faster. Storage virtualization is implemented using software, or with hardware and software hybrid devices.

The benefits of storage virtualization include increased storage capacity, automated management, reduced downtime, and simplified upgrades.

Virtual Networking (NV) is the creation of virtual networks within a virtualized infrastructure. The process combines hardware and software network resources and network functionality into a single, software-based administrative structure. This entity is a virtual network. Network virtualization combines network resources by dividing bandwidth into channels. Each channel is independent of the others and is

assigned to a specific server or device. Each channel is independently secured. Each network subscriber has shared access to all resources of this network. For network virtualization, the control plane function is removed from each device and performed by a centralized controller.

The centralized controller communicates the control plane functions to each device. Each device can now focus on data transmission, while the centralized controller manages the data flow, improves security, and provides a service chain. Data centers are often environments where many users live. NV can provide separate virtual networks to different clients in a virtual environment. This virtual network is completely isolated from other network resources, and traffic is divided into zones or containers to avoid mixing with other resources.

14.7. Data Engineering

Data engineering involves an information system where information (data) is collected or generated, processed, stored, distributed, and analyzed.

The ability to analyze data is typically accomplished through a database and a database management system (DBMS). Data engineering and data analysis are useful for any business or organization that wants to direct its resources based on meaningful information and statistics.

A relational database and the programming language Structured Query Language (SQL) are the basis of a relational database management system (RDMS). SQL is a programming language designed to retrieve information from relational databases using relational database management systems (RDBMS).

Relational database systems, such as MySQL, Microsoft SQL Server, Oracle, and IBM DB2, are the most popular database management systems. An RDBMS can have multiple users with multiple database transactions. The Atomic, Consistent, Isolated, and Durable (ACID) transaction model defines how database transactions maintain data integrity and survive failures.

In the early 2000s, the emergence of Web 2.0, e-commerce, and companies like Google made it clear that relational databases could not meet the volume and speed

of web search queries. To meet this demand, Google developed the Google Distributed File System (GFS), the distributed parallel processing algorithm MapReduce, and the distributed NoSQL database BigTable. In 2004, Jeffrey Dean and Sanjay Hemamat of Google published a paper, "Simplified Data Processing on Large Clusters," which inspired two programmers, Doug Keating and Mike Cafarella, to create Apache Hadoop. The MapReduce approach subsequently became the basis for the development of the Hadoop ecosystem and the HBase database, as well as the Amazon Dynamo key-value NoSQL database.

NoSQL databases can use a key-value store approach instead of a relational table-based approach. Other NoSQL databases store data as structured documents in XML or JSON formats. NoSQL databases are significantly faster than relational databases and can import unstructured data. NoSQL databases are designed to scale out, meaning that storage and management capacity can be increased simply by adding more machines to the cluster. The most popular NoSQL systems include MongoDB, Couchbase, Riak, Memcached, Redis, CouchDB, Hazelcast, Apache Cassandra, HBase, and Dynamo, which are all open source software products.

All of these technologies have become solutions to the problem of Big Data. Data is so large, fast, or diverse that it cannot be managed by a single computer. For this reason, these software solutions are commonly referred to as “big data technologies.” In reality, big data problems cannot be reduced to any one technology, but must include new and old technologies.

With the advent of IoT and Big Data, new job categories are emerging and adjustments to existing jobs are being made.

The digitization of business creates a wealth of accessible business data. To get the data we need, it is important to be able to ask the right question in the right way. A business analyst is someone who can study a business or industry and then formulate a specific question. Business analysts are data experts who work with a company’s stakeholders to identify a problem question. This question is then rephrased into a specific data problem. Business analysts then create business intelligence reports of various types for the company’s stakeholders.

Data analysts query and process data, provide reports, summarize and visualize data. They use existing tools and techniques to solve problems, and help understand specific queries through custom reports and graphs. Data analysts need to understand basic statistical principles, the process of cleaning different types of data, data visualization, and exploratory data analysis. Some of the tools and programs that help data analysts do their work include SAS, Rapid Miner, and programming languages such as R or Python.

A data analyst takes raw data and transforms it into meaningful information. Such researchers apply statistics, machine learning, and analytical approaches to answer critical business questions. Data science is an existing field that has expanded due to IoT and Big Data.

Data analysts must interpret and present their findings using visualization techniques, building applications for scientific data. They work with datasets of various sizes and shapes and run algorithms on large datasets.

Some of the tools that help data scientists do their work are Python, R, Scala, Apache Spark, Hadoop, data mining tools and algorithms, machine learning, statistics.

None of the three parts described above can exist without a data engineer. Data engineers create the infrastructure that supports Big Data. They design and build the platform on which all this data is stored and processed. Data engineers also manage all this data. They ensure that the data is accessible to scientists and analysts.

Data engineers can integrate data from different sources and perform data cleansing. Because data engineers primarily design the Big Data infrastructure for their company, they typically do not need to know machine learning or analytics.

Some of the tools and applications that data engineers regularly use include Hadoop, MapReduce, Hive, Pig, MySQL, MongoDB, Cassandra, streaming data, NoSQL, SQL, and programming. In some environments, there may be overlap between data analysts, data scientists, and data engineers.

As IoT grows and big data becomes even more prevalent, job descriptions may also change.

Conclusion to lecture 14

The virtualized data center supports Big Data and analytics. With fog computing, data can be processed almost immediately after it is created. Data centers are centralized locations that house large amounts of computing and networking equipment. Virtualization separates the OS from the hardware. Storage virtualization combines physical storage from multiple network storage devices into a storage that appears as a single storage device. Network virtualization (NV) is the creation of virtual networks within a virtualized infrastructure. Data engineering includes collection, processing, storage, distribution and information analysis.

Questions for reinforcement

1. What architectural models of Big Data engineering do you know?
2. What are data centers and cloud computing used for?
3. What are virtualization technologies used for?
4. What are hypervisors?
5. What are containers?
6. What is the container technology for executing software code on the server?
7. What is data engineering?

List of recommended literature

1. Virtualization Technology. URL: https://www.sciencedirect.com/topics/computer-science/virtualization-technology
2. What is virtualization technology. URL: https://pandorafms.com/blog/virtualization-technology/
3. Containerization. URL: https://www.ibm.com/cloud/learn/containerization
4. What are containers. URL: https://searchitoperations.techtarget.com/definition/container-containerization-or-container-based-virtualization

Lecture 15.
Hadoop Big Data technologies.
Distributed MapReduce processing. HDFS

Lecture plan

15.1. Scalability through big data.
15.2. Data storage and processing in distributed file systems.
15.3. Distributed databases.
15.4. Hadoop Distributed File System (HDFS).

15.1. Scalability through big data

In the context of Big Data, scalability means developing a solution that can meet the exponential growth needs of companies like Google and Facebook. It also applies to any company, organization, or government agency that needs to store and analyze unstructured and structured data from vast and diverse data sources. Scalability in this context means the ability to scale both data storage and data processing. The Hadoop Big Data ecosystem follows a scaling approach similar to Google, shown in Fig. 15.1.

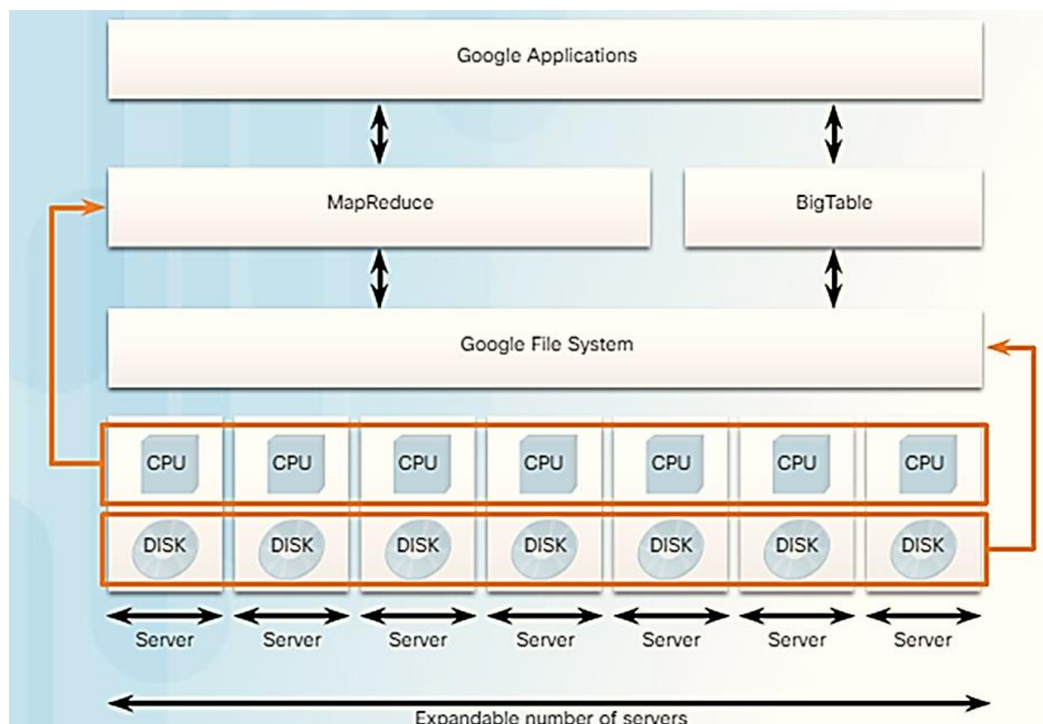


Fig. 15.1. Scaling Google data storage and processing [1]

15.2. Data storage and processing in distributed file systems

Hadoop uses a distributed file system that scales by adding more computers and hard disk drives. Data storage is increased by adding servers that run on hardware. The ability to use commodity hardware instead of expensive SAN or NAS storage systems reduces costs.

When Google needs additional processing power, it uses Google's Modular Data Center, a custom-built solution that adds an additional 1,000 processors that work together in a cluster using Google's MapReduce technology. The Hadoop Big Data ecosystem is also based on distributed MapReduce processing. Adding new servers or nodes to a Hadoop cluster not only adds additional disk storage, but also provides additional processor processing.

Companies that deal with big data or a large number of web transactions also have to deal with database scaling issues. Traditional relational databases were designed for a client-server architecture. They were not designed to be distributed across multiple database servers and to handle unstructured data or extremely large objects and rows of data. Google's BigTable distributed database, Amazon's Dynamo, and Hadoop's HBase are value stores key-value, non-relational databases designed to be distributed across multiple servers and scale as new servers are added.

Maintaining accessibility is a major concern for many companies working with Big Data. A website that cannot respond within 3 seconds will lose visitors. Companies can improve website availability by deploying load-balancing web servers and DNS servers. Replicate web servers can be deployed in data centers in different locations around the world to improve response times across the Internet.

15.3. Distributed databases

In Big Data, availability refers to the speed at which large volumes of data can be searched and processed. Distributed computing, including database storage and management, improves the processing speed and availability of data. A relational database was designed to run on a single server, not many.

To adapt to the need for distributed computing, a relational database can be distributed across multiple servers. Sharding requires a lot of complexity and reduces the relational database for row-level access one shard at a time. The businesses of large companies such as Google and Yahoo, Facebook, Twitter, Amazon need to be constantly online and available 24/7. Big data ecosystems such as Hadoop help ensure fault tolerance in data delivery and processing.

In the early 2000s, it became clear that a new database management system would be needed to catalog all the data on the World Wide Web. No single database or server could handle the vast amounts of data involved in such a large task. To solve this problem, Google needed a distributed computing system consisting of a cluster of servers, using a single file system spread across multiple servers, with each server sharing a portion of the processing load.

15.4. Hadoop distributed file system (HDFS)

Hadoop is a set of open source software utilities that facilitates the use of a network of many computers to solve problems involving large amounts of data and computation. Hadoop provides a software framework for distributed storage and processing of big data using the MapReduce programming model.

All modules in Hadoop are designed with the fundamental assumption that hardware failures are frequent occurrences and should be automatically handled by the framework.

Fig. 15.2. Hadoop Logo [2]

Hadoop Distributed File System (HDFS) is the file system where Hadoop stores data. HDFS does not replace the Linux file system on individual servers, but instead sits on top of a cluster of servers as a single file system spanning the entire cluster.

HDFS stores data in 64MB chunks using at least three DataNode servers (Fig. 15.2). HDFS manages information across the cluster from a centralized NameNode coordination server.

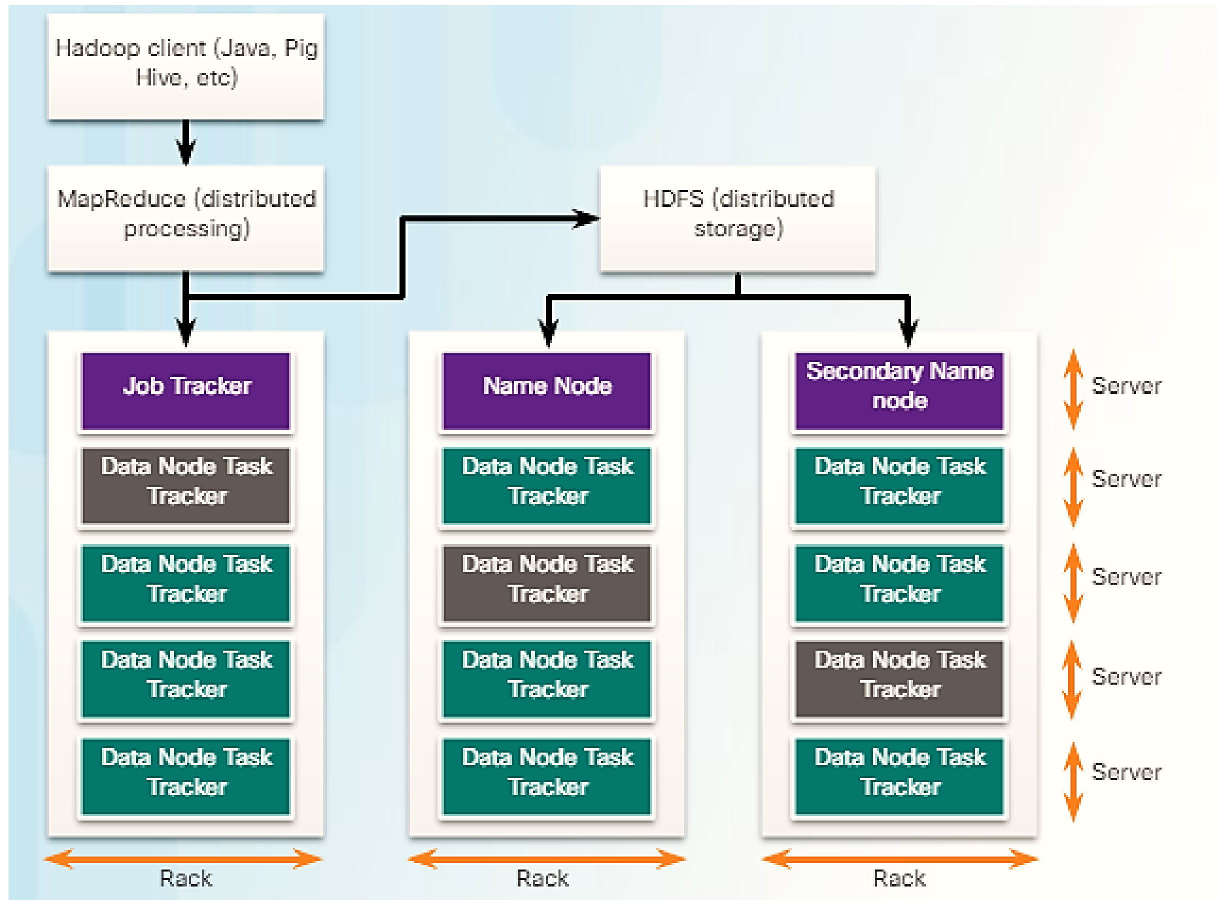


Fig. 15.3. Example of using the Hadoop file system version 1.0 [1]

The NameNode server keeps track of what data is on the various DataNodes. When data is brought into the system, it is imported into the NameNode, then the NameNode divides the data into chunks (64Mb) which are then replicated across three or more DataNodes depending on the configuration.

This provides fault tolerance similar to a mirrored RAID (Redundant Array of Independent Disks) array, in that if one DataNode fails, the DataNode can be replaced and the file system and data will be restored from the duplicate DataNodes.

15.5. MapReduce

MapReduce is a distributed framework for parallelizing the processing of large data sets across a large number of servers.

MapReduce divides data processing into two phases:

1. the mapping stage, when data is broken into parts that can be processed by separate threads, even running on separate machines;

2. the reduction phase, which combines the output from multiple maps into the final result (Fig. 15.4).

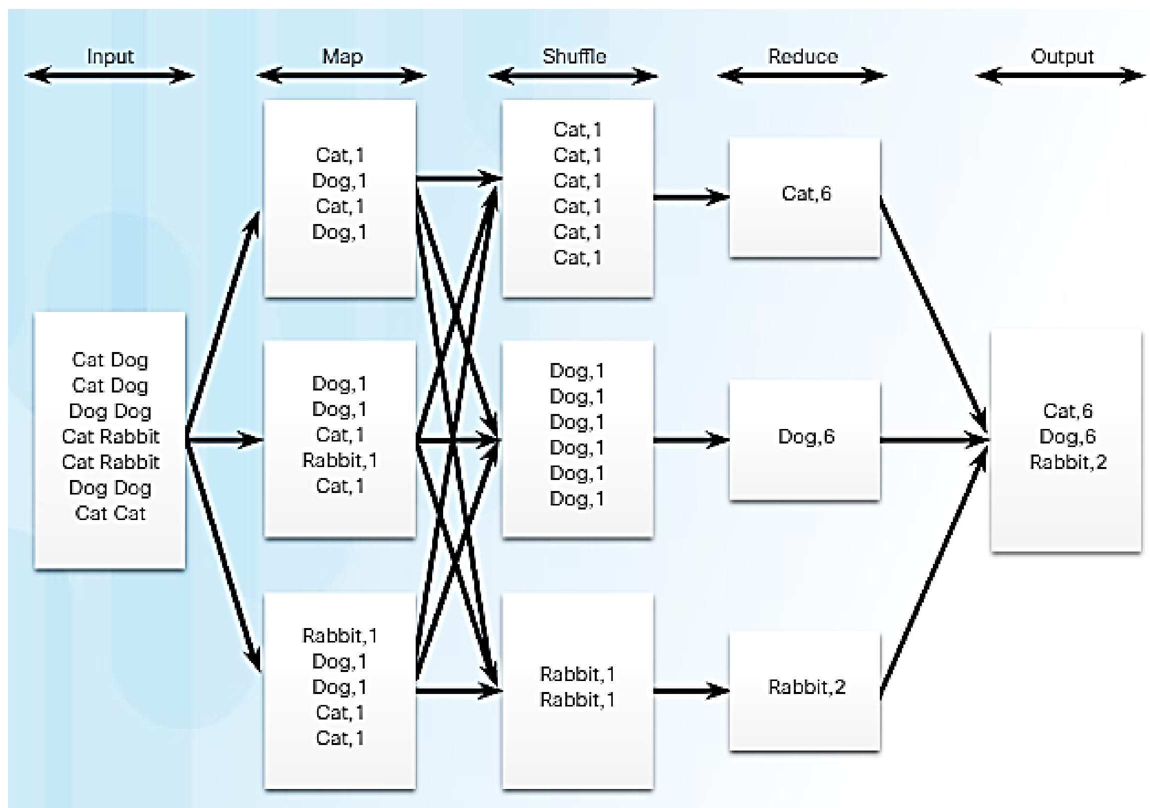


Fig. 15.4. Example of parallel data processing in MapReduce [1]

Hadoop v2.0 is an ecosystem of applications that work together and includes the following technologies:

- HDFS distributed file system;
- distributed database HBase;
- MapReduce distributed processing;
- Hive provides an interface similar to SQL.

YARN – a platform for managing computing resources in clusters and using them to plan user programs, which coordinates resources across multiple processor engines:

• **Spark** for running processes in memory;

• **Tez** for running batch processes.

Additional client applications that can access Hadoop include **Apache Pig**, a high-level framework for building applications that run on Hadoop, and Mahout as a machine learning interface.

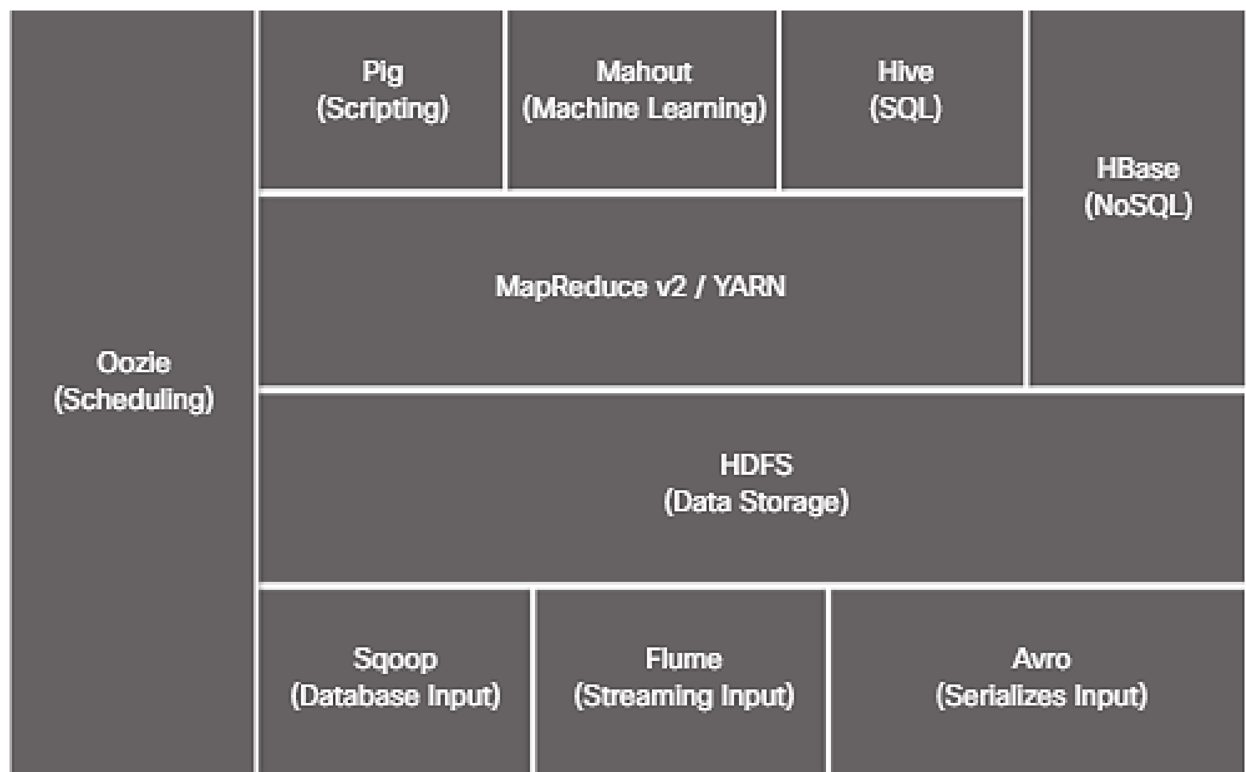


Fig. 15.5. Data processing in MapReduce [1]

Conclusion to lecture 15

Hadoop is a set of open source software utilities that facilitates the use of a network of many computers to solve problems involving large amounts of data and computation and provides a software framework for distributed storage and processing of big data. The Hadoop Distributed File System (HDFS) is the file system in which Hadoop stores data. MapReduce is a distributed processing and parallelization system for computing. large data sets between a large number of

servers. YARN is a platform for managing computing resources in clusters and using them to schedule user applications and allocate resources across multiple processor engines. Apache Pig is a high-level platform for building applications that run on Hadoop and Mahout as a machine learning interface.

Hadoop offers significant advantages for big data processing due to its ability to store and process massive volumes of data across distributed clusters of low-cost hardware. Its scalability, fault tolerance, and flexibility make it suitable for both structured and unstructured data. Hadoop's open-source nature allows for continuous improvements and customization, while tools in its ecosystem (like HDFS, MapReduce, Hive, and Spark) enable parallel data processing and efficient analytics. These features make Hadoop a powerful framework for organizations needing to handle large-scale data workloads.

Questions for reinforcement

1. How is data stored and processed in distributed file systems?
2. What is the purpose of distributed databases?
3. What is the purpose of MapReduce?
4. What is Hadoop?
5. What is the structure of Hadoop?
6. What is YARN?
7. What is HDFS?

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. Apache Hadoop. URL: http://hadoop.apache.org/
3. HDFS. URL: https://www.ibm.com/analytics/hadoop/hdfs
4. MapReduce Tutorial. URL: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Lecture 16.
Kafka distributed streaming platform.
Cassandra advantages

Lecture plan

16.1. Data reception problem.
16.2. Kafka distributed streaming platform.
16.3. Cassandra advantages.

16.1. Data reception problem

Web services that help reliably and at specified intervals process data and move it between different computing services are called pipelines and perform data ingestion, storage, and processing (Fig. 16.1). For each of these components, there are many software platforms that are used to perform each task.

Depending on the type of data, the type of ingestion, and the computing requirements, the components of each data pipeline are unique, often built together, and adjusted to work with each other.

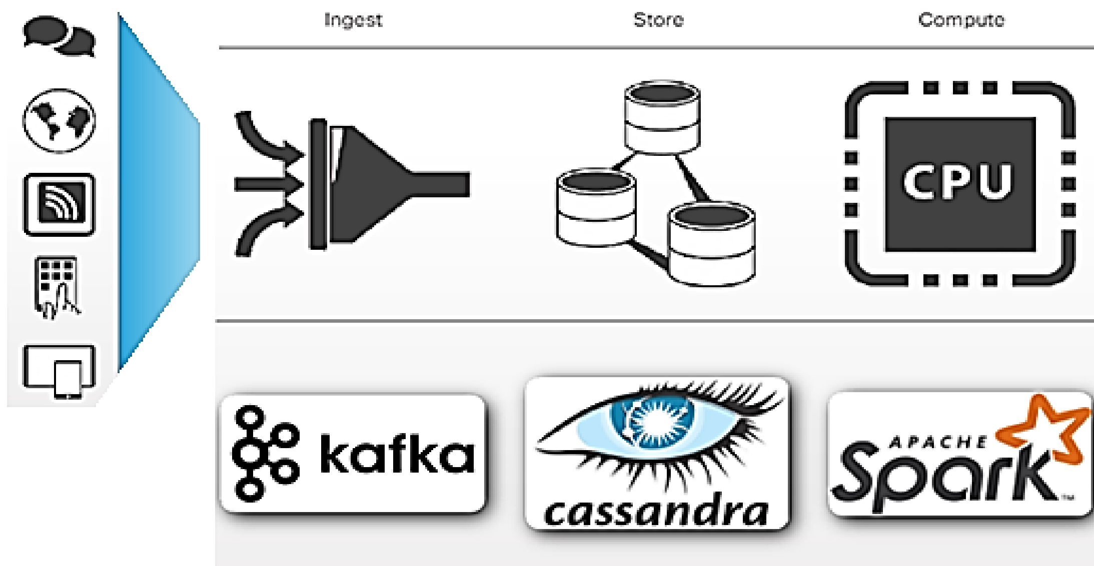


Fig. 16.1. Big data pipeline [1]

Big data often comes from many different sources. Much of this data is streaming and must be ingested in real time. Scalability is also an issue. As more

devices connect, more data needs to be ingested. The quality of this data can also be an issue. The data can be structured, unstructured, or in complex formats. There are many different tools for collecting and moving large amounts of data from different sources into a data warehouse.

16.2. Kafka distributed streaming platform

For real-time data transmission, we need to use a distributed streaming platform such as Kafka. Kafka features:

- record streams are published and subscribed to (pub-sub) similarly to an enterprise messaging system;
- record streams are fault-tolerant due to distributed storage4
- record streams are processed as they occur, in real time.

Kafka is used to transmit real-time streaming data between different systems and applications. Kafka is also used to transform and respond to data streams across applications in real-time.

Kafka was developed by LinkedIn but became open source software in 2011. Cisco Systems, Netflix, and eBay are just a few of the enterprises that use Kafka.

Consider the main provisions of Kafka.

- Kafka runs on one or more servers as a cluster. The servers in a cluster are known as brokers.
- The cluster stores streams of records in groups called topics.
- Each record contains a key, a value, and a timestamp.

Kafka has four main APIs (Fig. 16.2).

- Producers. An API where a stream of records is published by an application to Kafka topics.
- Stream processors. APIs where input streams from topics are consumed and output streams to topics can be produced.
- Connectors. APIs where Kafka topics connect to existing systems and software applications.
- Consumers. The API where a stream of posts in topics is created.

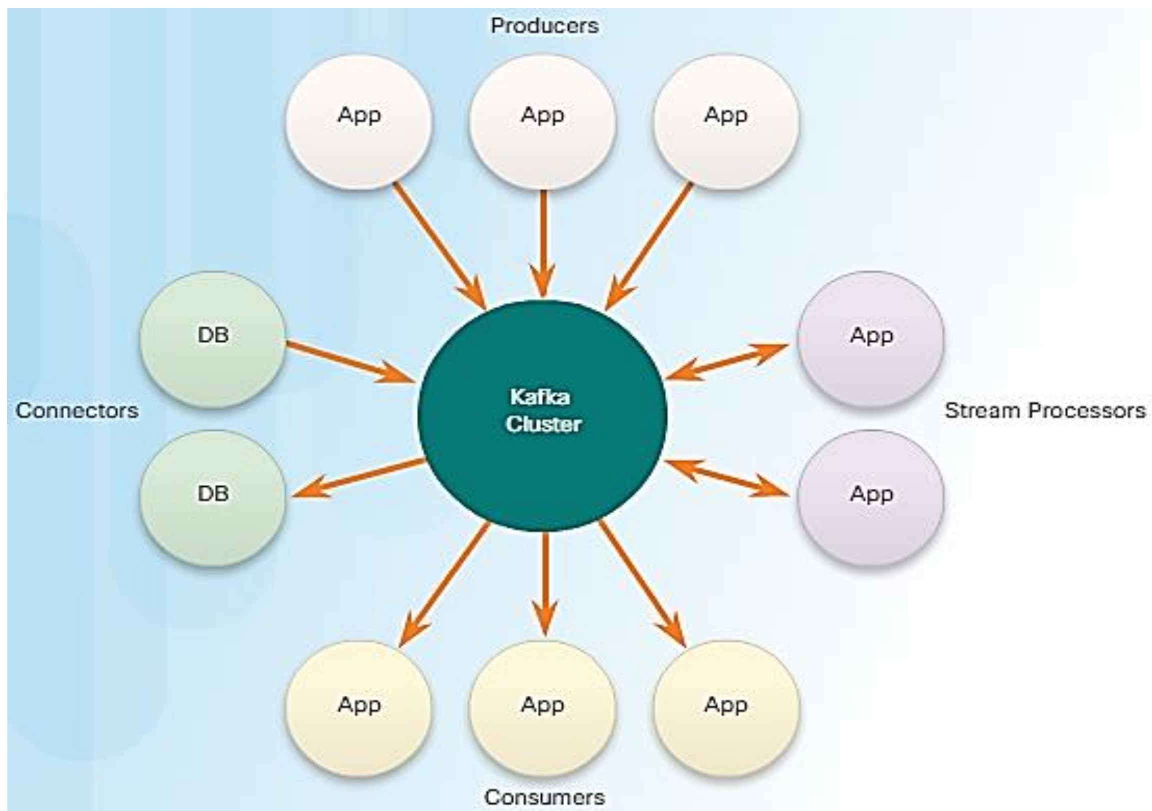


Fig. 16.2. Kafka API components [1]

Kafka acts as a messaging system to centralize communication between all data producers and consumers in a system. Kafka uses distributed processing of huge amounts of data and scalability, ensuring fault tolerance of hardware and software. Message transmission in Kafka occurs in two ways.

- **Publish and subscribe** – requested messages are transmitted to all consumers.
- **Point-to-point** – multiple consumers read messages from the server. Each of these messages arrives at one of the consumers.

Kafka is much more than just a messaging server. Kafka works similarly to a distributed database.

Messages written to Kafka are replicated across multiple servers and written to disk. Due to its distributed design, Kafka is highly available, supports automatic data recovery, and is highly resilient to network failures.

Kafka differs from traditional message brokers in its use of transaction logs. Each topic consists of a set of logs called **partitions**. Producers add these logs, and

consumers can read the logs when needed. **Topics** are data that is replicated across multiple brokers to achieve fault tolerance.

Kafka can handle high throughput by offloading many tasks to producers and consumers. This keeps brokers lightweight and makes Kafka a desirable tool for many IoT platforms. With the vast amount of data coming in from sensors and other devices in real time, Kafka can scale data ingestion and serve data to multiple consumers simultaneously.

IBM's Big Data estimates, "every day we create 2.5 quintillion bytes of data" every day , because every minute:

- over 300 hours of YouTube videos are uploaded;
- over 3.5 million text messages are sent;
- over 86,000 hours of Netflix video are streamed;
- estimated at over 4 million posts on Facebook.

Due to this, a huge amount of data is constantly being created, and even with cloud storage available from companies such as Amazon, Google, and Microsoft, data security becomes a major problem.

Big Data solutions must be secure, have high fault tolerance, be able to scale horizontally, and use replication to ensure data is not lost.

Big Data is the term for the vast amount of data we are constantly creating from a vast number of data sources. This data needs to be relevant, clean, and secure. There are at least challenges to storing Big Data.

- **Management** – there are few data exchange standards and thousands of data management tools.
- **Security** – authentication, access, and accounting are difficult to secure.
- **Unstructured data** – unstructured data is difficult to analyze and search.

16.3. Cassandra Advantages

Cassandra is an open source NoSQL distributed database management system. A NoSQL database is schema-free and does not use traditional data storage and retrieval methods such as relational databases, has a simple design, supports

horizontal scaling, and provides greater availability control. Cassandra can be fully distributed and deployed globally if needed, providing a decentralized database with no single point of failure. Cassandra is a distributed database management system developed by Facebook in 2008. One of the things that makes Cassandra fast is that it uses sequential reads and writes. This is convenient for work with time series data in IoT solutions. Instead of adding or deleting data in a file, a new file is created and old files are deleted.

Key features of Cassandra are next.

- **Data dissemination** – data can be replicated across multiple data centers.
- **Transaction support** – supports atomicity, consistency, isolation, durability (ACID).
- **Elastic scalability** – As more data is needed or more clients are added, more hardware can be added to scale as needed.
- **Fast linear scaling** – as the number of nodes increases in a cluster, throughput increases, supporting fast response.
- **Quick recording** – Cassandra performs fast writes while storing hundreds of terabytes of data.
- **Always on** – Cassandra is always available even during hardware and/or network failures.
- **Storage flexibility data** – unstructured, structured, and semi-structured data are supported.

Apache Hadoop defines HDFS as "the primary storage system used by Hadoop applications" that enables " reliable and fast computation."

The primary NameNode in the cluster governs the file system and data access. The cluster also has DataNodes, often a single physical machine, to handle the added storage.

Data is stored in files, divided into blocks, and distributed across DataNodes. HDFS copies these blocks to two additional servers by default.

Cassandra uses the Cassandra File System (CFS).

Each cluster node has the same peer-to-peer implementation.

Clusters store data in real time and analytical operations can be performed on this data (Fig. 16.3).

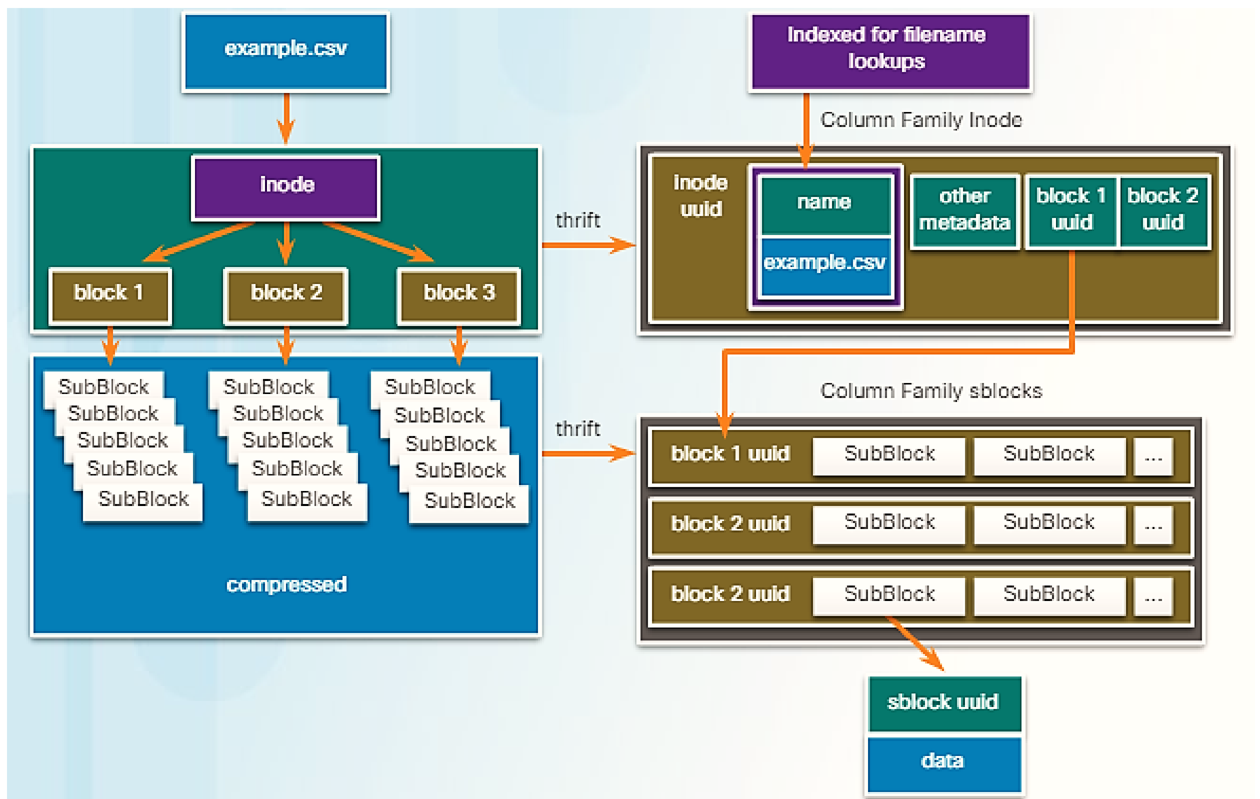


Fig. 16.3. Cassandra File System [1]

Built-in replication copies data across all real analytics and search nodes.

There are two families of columns, similar to RDBMS tables, that contain data.

Data in these column families is replicated across the cluster to ensure data protection. Column families are similar to the two main HDFS services.

Inode column family takes the place of the HDFS NameNode service.

SubBlocks column family takes the place of the HDFS DataNode service. These columns store the contents of any file.

Advantages of using CFS compared to HDFS are next.

• **Better accessibility** – no shared storage solution is required. The more nodes and clusters, the better the availability. This can also be improved by increasing the replication ratio, which determines how many nodes will receive replicated data.

- **Hardware support** – no special servers or special network devices are required for CFS.
- **Automatic recovery** – nodes, clusters, even data centers can fail, and data on nodes and clusters will remain available elsewhere.
- **Data integration** – all data written to Cassandra is replicated to both the analytics and search nodes. This allows analytics, search, and real-time tasks to run simultaneously without affecting each other.
- **Easier deployment** – clusters are easy to set up and can be up and running in just a few minutes.
- **Multiple data centers supported** – CFS can run a single database across multiple data centers. Tools are available to allow each data center to have a local copy of all data in the database. Analytical tasks can be run across multiple data centers simultaneously.

HDFS is an excellent choice when a Hadoop application needs a low-cost storage solution with a focus on data storage.

CFS can perform analytics on data coming from large applications that integrate into databases and DBMSs.

Apache Cassandra fully implements the principles of Availability and Partition tolerance, ensuring correct response to any request and ease of scaling.

Data consistency is considered a weak point of this distributed DBMS due to its decentralization, when different replicas of the same information are stored on many nodes. To eliminate this problem, a mechanism for configuring consistency levels is used.

Unlike the NoSQL database for Big Data, Apache HBase, all nodes in a Cassandra cluster are equal – clients can connect to any of them for writing and reading. Query execution begins with its coordination, in order to use the key and labels to determine which nodes in the cluster have the required data and send the query there. The node that performs the coordination is called **coordinator**, and the nodes that are selected to store the record with the given key are **replica nodes**. Physically, the coordinator can be one of the replica nodes.

Data availability directly depends on the consistency level of read and write operations, as it determines how many replica nodes can fail when confirming the successful execution of these operations. For example, if the replication level is less than the sum of the nodes from which the confirmation of the successful data write came and from which the read is being performed, that is, **a guarantee of strong consistency**.

Strong consistency guarantees that after writing a new value, it will always be read. Otherwise, a situation is possible when the DBMS returns stale data as a result of the read. To combat this, Apache Cassandra has an **eventual consistency mechanism**, which distributes data across replica nodes after the coordination wait ends. If not all replica nodes are available, then other means of data recovery will have to be used, in particular, read with correction and manually launched **anti-entropy node repair**.

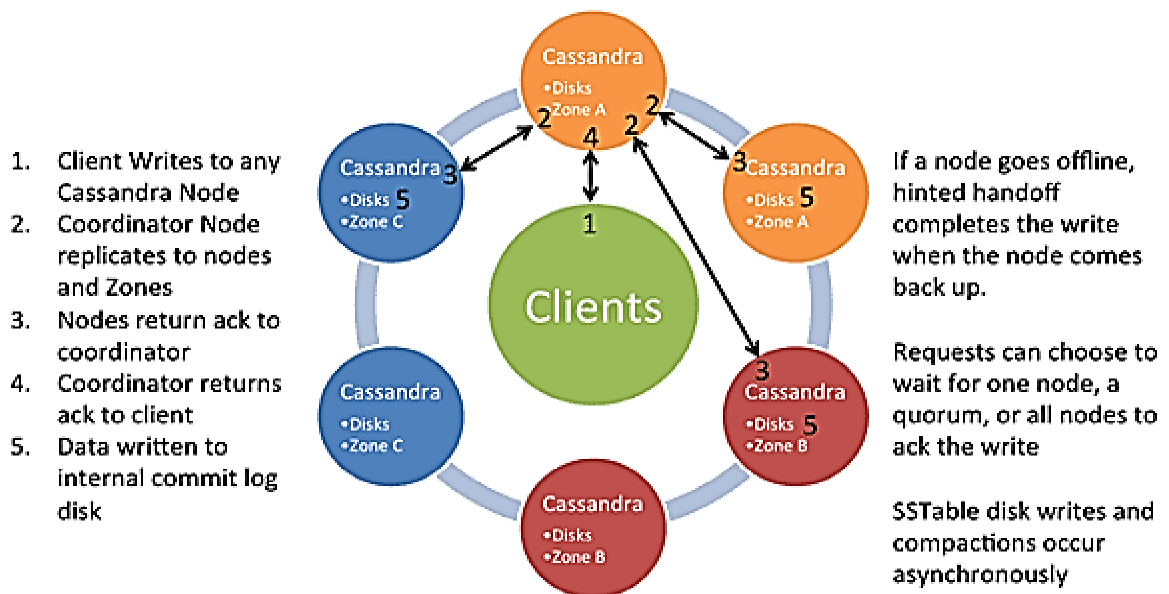


Fig. 16.4. Distributed data writing in an Apache Cassandra cluster [2]

When writing data, the consistency level determines the number of replica nodes that will be expected to confirm the successful completion of the operation – a signal that the data was successfully written. There are the following consistency levels for writing:

- **ANY (0)** – allows data to be written even if all replicas do not respond. The coordinator receives a response from any one replica or the data is persisted using a hinted handoff mechanism to the coordinator.
- **ONE (1)** – the coordinator sends requests to all replica nodes, but returns control to the user, waiting for confirmation from any first node;
- **TWO (2)** – the coordinator waits for confirmation from the first two nodes before returning control;
- **THREE (3)** – the coordinator waits for confirmation from the first three nodes before returning control;
- **QUORUM (4)** – the coordinator waits for confirmation of records from more than half of the replica nodes, namely *round (N/2)+1* , where *N* is the replication level;
- **ALL (5)** – the coordinator waits for confirmation from all replica nodes;
- **LOCAL_ QUORUM (6)** – the coordinator waits for confirmation from more than half of the replica nodes in the same data center as the coordinator. This eliminates delays associated with sending data to other data centers.
- **EACH_ QUORUM (7)** – the coordinator is waiting for confirmation from more than half of the replica nodes in each data center.

In the case of reading, the consistency level determines the number of replica nodes from which information will be read:

- **ONE (1)** – the coordinator sends requests to the nearest replica node, reading others for read repair with a probability pre-set in the configuration;
- **TWO (2)** – the coordinator sends requests to the two nearest nodes, choosing the value with the larger timestamp;
- **THREE (3)** – the coordinator sends requests to the three nearest nodes, choosing the value with the largest timestamp;
- **QUORUM (4)** – a quorum is being met, i.e. the coordinator sends requests to more than half of the replica nodes, namely *round (N/2)+1*, where *N* is the replication level;
- **ALL (5)** – the coordinator returns data after reading from all replica nodes;

- **LOCAL_QUORUM** (6) – a quorum of nodes is gathered in the data center where coordination takes place, and data with the latest timestamp is returned;
- **EACH_QUORUM** (7) – the coordinator returns data after quorum meetings in each of the data centers.

Summarizing the possible levels of consistency in Apache Cassandra, we can draw the following conclusions:

- **QUORUM** for reading and writing will ensure strict consistency and balance between the latency of these operations;
- Strong consistency will also be achieved when writing ALL and reading ONE. In this case, the data is read faster and with greater availability. All replica worker nodes will be required for the write.
- The reverse case (**write ONE, read ALL**) ensures strict consistency, the write will be faster and with greater availability, because confirmation of the successful completion of the operation will be required from only one node.
- In the absence of strict consistency requirements, we can speed up read and write operations and improve availability by setting lower consistency levels.

Apache Cassandra is a highly scalable, distributed NoSQL database designed for managing large volumes of structured data across many servers. It is particularly valued for its high availability and fault tolerance, making it ideal for applications that cannot afford to lose data or experience downtime.

One common use case for Cassandra is in real-time big data applications, such as IoT platforms. For example, smart home systems can use Cassandra to store time-series sensor data from devices like thermostats, lights, and alarms, ensuring fast write operations and consistent access across multiple devices and users.

Another example is in social media platforms, where user activity such as posts, likes, and messages needs to be recorded and retrieved instantly across millions of users. Cassandra's ability to handle high write throughput and replicate data across global data centers ensures that content is available to users with minimal delay and downtime. Cassandra is also widely used in e-commerce and recommendation systems. For instance, large online retailers can use it to store user session data,

transaction histories, and product catalogs, enabling personalized recommendations and real-time analytics. Its decentralized architecture ensures that the system continues operating even if some nodes fail, which is critical for online businesses.

Cassandra is a powerful solution for companies requiring distributed storage, high-speed writes, and horizontal scalability in mission-critical applications.

Conclusion to lecture 16

Distributed streaming platform differs from traditional message brokers by using transaction logs to transfer streaming data in real time between different systems and applications. Cassandra is an open source NoSQL distributed database system. Cassandra uses the Cassandra File System (CFS). Each node in the cluster has the same peer-to-peer implementation. The clusters store data in real time and perform analytics on that data.

Questions for reinforcement

1. What data reception problems do you know?
2. What is Kafka used for?
3. What is the purpose and benefits of Cassandra?
4. In what types of real-time systems is Cassandra most commonly used?
5. Why is Cassandra a good choice for applications requiring high write throughput?

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. What is Cassandra. URL: https://cassandra.apache.org/
3. About the Cassandra File System (CFS) – deprecated. URL: https://docs.datastax.com/en/dse/5.1/dse-dev/datastax_enterprise/ analytics/ cfsAbout.html

Lecture 17.
Apache Spark platform

Lecture plan

17.1. The problem of the computable function.
17.2. Spark technology.
17.3. Comparison of Spark and MapReduce.
17.4. Spark and sparklyr for working with big data in R.

17.1. The problem of the computable function

An important challenge faced by Big Data computing is the size of the data sets used in various fields. For example, genetic sequencing (the digital representation of the human genome) has become a subject of research in the scientific community in recent years. This is because high-performance computing (HPC), combined with the development of sequencing technology, has made it possible to sequence the entire human genome in 26 hours. The sequence of the entire human genome is about 200 GB of data and requires a huge amount of computing power to process it.

The amount of data and its analysis may exceed the most powerful HPC in the future. When this happens, HPC must either be scaled up by adding more processors and memory to the computer, or scaled up by adding more computers to the cluster and connecting them with high-speed connections. When the amount of data to be processed is huge, frequent data movements can significantly increase latency.

It is desirable to have a computing system that works to overcome the limitations of the storage system to minimize latency.

Analytics conducted on Big Data can offer new opportunities and unforeseen trends, providing a better overview of customers and the market as a whole.

Accurate customer analytics, fraud detection, and risk analysis are the benefits of big data analytics. These complex calculations require not only large data stores, but also low-latency, high-throughput streaming processing. This further increases the size and complexity of HPC analytics.

To reduce the latency of frequent I/O operations, the computation can be moved to the server where the data is located. Several small tasks are executed in many places to distribute the load. The MapReduce model can reduce the large number of I/O and network bottlenecks to some extent, but this is not the main purpose of MapReduce.

Computational latency is also a big data problem. Big data computation is partitioned to perform specific computations on different nodes. When one node is slower than the others, the response time increases. This forces the results of the overall work to wait for that node to perform its part before they can be implemented. Latency is a significant problem in large data centers, causing a significant impact on time-sensitive computations, such as data streams.

Also, work schedules can significantly increase latency. Often, large amounts of work are completed while other, smaller amounts of work are being performed.

Smaller tasks that have to wait for larger tasks to complete can cause delays. This can be a problem when some tasks need to be processed in real time.

17.2. Spark technology

Spark is distributed, open-source data processing engine used for big data. While the Hadoop platform has enabled many companies to successfully apply the MapReduce paradigm for distributed computing of huge amounts of data, each time a new task arises, new code for map and reduce operations must be written, which is inconvenient and time-consuming.

To solve this problem, Facebook engineers created Hive, a Hadoop-based database management system, in 2008. The main feature of Hive was its support for SQL-like queries against data stored in HDFS (this new SQL dialect was called Hive Query Language, HQL).

In 2009, the Spark research project was launched at the University of California, Berkeley, with the aim of improving the efficiency of distributed MapReduce computing and creating a universal platform for such computing.

Spark was published as an open source project in 2010 and was donated to the Apache Software Foundation in 2013. Spark uses in-memory caching to achieve fast performance by limiting disk reads and writes.

Apache Spark is an open source framework for implementing distributed processing of unstructured and weakly structured data, which is part of the Hadoop project ecosystem. Unlike the classic Hadoop core processor, which implements the two-level MapReduce concept with storage of intermediate data on drives, Spark works according to the paradigm of resident computing (*in-memory computing*), processes data in RAM, which allows for significant speed gains for some classes of tasks, in particular, the ability to access data loaded into memory multiple times makes the library attractive for machine learning algorithms.

Spark can use HDFS and achieve better performance than MapReduce. Spark supports programming languages such as Python, Scala, R, and Java. Spark also supports structured databases in SQL. With this variety of supported languages and systems, many different Spark solutions can be implemented.

Spark includes libraries that help create different types of applications.

- **Shark SQL** – a SQL library that efficiently supports complex analytics while remaining fault-tolerant.
- **Spark Streaming** – A stream processing library that scales, supports high throughput, and supports fault tolerance.
- **MLib** – a scalable, high-performance machine learning library.
- **GraphX** – a library that contains graph theory algorithms.

One reason Spark is so fast is that it uses Resilient Distributed Datasets (RDDs) to store input, output, and intermediate data in memory instead of on disk. This eliminates the cost of I/O. RDDs are resilient because they keep track of history in case they have to reconstruct themselves after a failure. They are distributed because they are spread across many nodes in a cluster. This allows for redundancy, as well as increased efficiency through parallel processing. When a request comes in from an application, Spark creates and loads data into an RDD. The data can come from any source, including HDFS, CFS, AWS S3, or even a SQL database. Once an RDD

is created, Spark can perform two different types of operations on an RDD (Fig. 17.1).

- *Transformations* are any manipulation of data, including mapping or filtering. Any transformation will result in the creation of a new RDD. The original RDD is not changed. This allows Spark to implement version control on the RDD. Transformations are only performed when they are needed, for example, by an action.
- *Actions* interact with the data but do not create changes. Examples include aggregating, counting, or retrieving a specific element in an RDD. When an action is requested, a new RDD is created on which the action is to be performed.

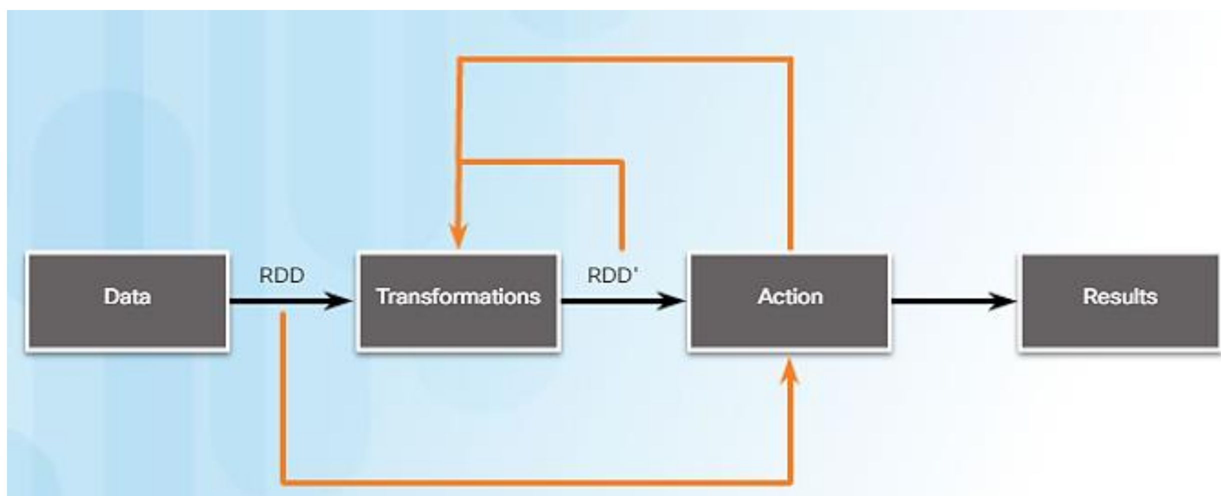


Fig. 17.1. Spark flow [1]

17.3. Comparison of Spark and MapReduce

Spark can run directly on a Hadoop instance, using HDFS for storage and YARN for cluster management.

Spark does not require Hadoop. We can use other storage solutions such as CFS or AWS S3, as well as other cluster managers such as Mesos.

Spark is platform independent because it supports many different technologies and programming languages. This is useful because there are so many different needs when it comes to big data solutions and analysis in so many different areas.

The approach to solving big data is to choose the right tools for the job. Sometimes a combination of Spark and MapReduce is the best solution for a particular job. Using Hadoop with MapReduce is still viable when performing batch processing with an application that uses HDFS exclusively, or existing applications that use this technology are already in production.

Spark has gained great popularity due to its performance, ease of administration, and the fact that it allows software to be built faster.

Consider several reasons to use Spark instead of MapReduce when creating big data software solutions.

- *Streaming data* – Spark is capable of processing huge amounts of data in real time. For example, data can come from mobile devices, social networks, or IoT sensors.
- *Heterogeneous data* – many big data solutions receive data from different sources.
- *Machine Learning* – with its built-in machine learning library, Spark can cater to a much larger audience than previous solutions.
- *Real-time applications* – in-memory processing allows Spark to return results much faster than MapReduce. This is important in all situations, but is essential when the application requires real-time results.
- *Less code* – Spark supports many different programming languages, which means less code needs to be written and maintained.
- *Developer experience* – Spark is much easier to learn than MapReduce. It results in more reliable code for the project much faster.

17.4. Spark and sparklyr for working with big data in R

The R ecosystem includes a wide range of packages that allow users to work with remote databases and perform scalable distributed computing using dedicated software platforms, or frameworks. All computations in R are performed in the computer's RAM. The size of many modern datasets (for example, those collected

by high-volume web applications and industrial systems) far exceeds the RAM available on a single computer. Such data is typically stored in a remote database or other type of storage. Depending on the task, we can use one of the following strategies to process such data with R.

1. *Creating a representative sample of limited size*. Almost all classical statistical methods assume that the researcher works with a representative sample of some general population. Therefore, to apply such methods, it is quite possible to randomly select several thousand or even hundreds of thousands of records from the database, and then perform the necessary analysis locally on the user's computer.

This approach is especially useful in the initial stages of a project, when we need to quickly familiarize with the properties of the data or create a prototype of the model. At the same time, the user will have access to the full range of additional packages available for R.

The disadvantage of this approach is that sometimes creating a representative sample is more difficult than it seems. When working with a sample, the researcher risks making incorrect conclusions about the properties of the general population.

2. *Data partitioning*. Solving a number of problems involves processing logically separated data sets, for example, corresponding to certain time periods, separate geographical areas, companies, users, etc. Such sets can be easily extracted from the database and then sequentially (sometimes in parallel) processed using R on the researcher’s local computer. In this strategy, all available data is subjected to analysis. These data must be indivisible into separate parts, which is not always possible or does not always make sense. Depending on the volume of data, this approach can take too much time and computational resources.

3. *Performing resource-intensive calculations on the database side*. Often, before performing statistical analysis or building a predictive model, it is necessary to apply operations such as filtering, grouping, aggregation, etc. to the data. Most databases cope with such operations perfectly, and therefore it makes sense to first perform such calculations on the database side, and then download the resulting smaller data set to the user's computer and perform its further processing using R.

This strategy also includes the ability to build some predictive models using the database's computational resources. For example, the **modeldb** package allows create linear regression models in this way. It is also worth mentioning the attempts of some companies to implement the ability to execute R scripts completely on the database side. In particular, this ability is available in Microsoft SQL Server. Depending on the database used, the functionality required by the user may sometimes be unavailable. Moreover, not all databases are equally efficient: executing resource-intensive queries can significantly slow down some of them, which , in turn , can cause other undesirable consequences (for example, slowing down the website served by such a database).

4. *Using specialized software platforms for working with big data*. If the strategies described above are not suitable for one reason or another, it is worth turning to specialized software platforms designed to organize and perform distributed computing on big data. The most famous and widely used among such platforms are Hadoop and Spark (both are part of the Apache Software Foundation projects and therefore are also often called Apache Hadoop and Apache Spark).

There are several packages in R (notably **sparklyr**) that provide a convenient interface for working with these platforms. Spark is one of the most widely used platforms for working with big data, characterized by high performance due to calculations performed in the RAM of a large number of computers combined into a single cluster, as well as due to efficient data transfer protocols over the network. The **sparklyr** package provides a convenient interface for working with Spark clusters from the R environment. In particular, it can be used to establish a connection to the cluster; perform data transformation, filtering, and aggregation operations using **dplyr syntax**; build predictive models using machine learning algorithms implemented in the MLlib library for Spark; work with other R packages that use Spark to perform distributed computing.

Thanks to packages like **sparklyr**, we can use R as a client that pushes compute tasks to a Spark cluster and then collects the results for further analysis in R itself. There are two ways to formally represent such compute tasks before sending them

to the cluster: either using SQL commands or using functions from the **dplyr package**. While SQL (more precisely, Spark SQL, which is based on the HiveQL dialect) allows formulate arbitrarily complex operations on data, in most standard cases dplyr **is** more convenient because it is familiar to most modern R users and has a more concise syntax.

The examples below use data from the nycflights13 package, which contains several tables describing 336,776 flights from New York airports in 2013. Let 's start a local Spark cluster and load the necessary tables into it [4]:

```
require(sparklyr)
require(nycflights13)
# connect to cluster:
sc <- spark_connect(master = "local", version = "2.3")
# load data into cluster:
flights_tbl <- copy_to(sc, flights, "flights") # flight data
airlines_tbl <- copy_to(sc, airlines, "airlines") # airline data
```

Let's assume that we are faced with the task of building a model that predicts the probability of a delayed flight arriving without delay (given that the departure is delayed by 15-30 minutes). We will consider this task as a case of binary classification: the feedback of interest to us takes the value 1 if the delayed flight arrived without delay, and 0 if not.

The process of building predictive models includes several steps, the first of which is data preparation and exploratory analysis. The exploratory analysis itself can also consist of several steps, such as identifying and eliminating problems with the quality of the analyzed data (missing observations, outliers, etc.), calculating descriptive statistics, identifying the most promising predictors for further inclusion in the model, etc.

Let's see how the commands from **dplyr package** can help with this.

Using the **dplyr** package, we can perform the following standard types of calculations on a Spark cluster:

- selection, filtering and aggregation of variables;

- using window functions;
- combining multiple tables using join operators;
- import calculation results from Spark into the R environment.

Dplyr commands include select(), filter(), mutate(), summarise() and arrange(). In addition, the group_by() command is used to perform group operations.

These and other dplyr commands can be combined into "chains" using the %>% operator from the **magrittr** package (it is loaded at the same time as dplyr and therefore does not need to be called separately). We count the total number of flights performed by each airline in 2013, and then select the 5 airlines with the largest number of flights:

```
require(dplyr)
flights_tbl %>%
group_by(carrier) %>%
summarise(N = n()) %>%
arrange(desc(N)) %>%
head(5)
## # Source:     spark<?> [?? x 2]
## # Ordered by: desc(N)
##   carrier     N
##   <chr>   <dbl>
## 1 UA      58665
## 2 B6      54635
## 3 EV      54173
## 4 DL      48110
## 5 AA      32729
```

Since Spark "understands" only SQL, in reality, all calculations specified in R using **dplyr** commands are automatically translated into SQL before being sent to the Spark cluster as follows:

select () -> SELECT

filter () -> WHERE

mutate () -> operators + , - , / , * , log , etc.

summarise () -> aggregation functions SUM , MIN , MAX , etc.

arrange () -> ORDER

group_by () -> GROUP BY

In addition, dplyr automatically translates the following basic R commands to SQL:

```
# Mathematical operators:
+, -, *, /, %%, ^
# Mathematical functions:
abs, acos, asin, asinh, atan, atan2, ceiling, cos, cosh, exp,
floor, log, log10, round, sign, sin, sinh, sqrt, tan, tanh
# Logical operators:
<, <=, !=, >=, >, ==, %in%
# Boolean operators:
&, &&, |, ||, !
# String functions:
paste, tolower, toupper, nchar
# Functions for converting variable type:
as.double, as.integer, as.logical, as.character, as.date
# Aggregation functions:
mean, sum, min, max, sd, var, cor, cov, n
```

The **show_query()** function from the **dplyr** package allows view the SQL query that is generated from the corresponding R code. For the example above, we get:

```
flights_tbl %>%
group_by(carrier) %>%
summarise(N = n()) %>%
arrange(desc(N)) %>%
head(5) %>%
```

```
show_query()
## <SQL>
## SELECT `carrier`, count(*) AS `N`
## FROM `flights`
## GROUP BY `carrier`
## ORDER BY `N` DESC
## LIMIT 5
```

As with databases, **dplyr** uses the principle of "lazy evaluation" when working with Spark. This means that all calculations on the Spark cluster are postponed until the last moment, until the result is needed. In addition, the final result of the calculations will be imported into the R environment only if the user explicitly requests it. For example, we could write the above command sequence as follows:

```
command_1 <- group_by(flights_tbl, carrier)
command_2 <- summarise(command_1, N = n())
command_3 <- arrange(command_2, desc(N))
command_4 <- head(command_3, n = 5)
```

None of these sequential commands will be executed until we request the result of the calculations. As mentioned earlier, the command_1, command_2 ... command_4 objects store only the information needed to connect to the Spark cluster, as well as instructions for the corresponding calculations.

"Requesting the result of calculations explicitly" refers to two situations: either displaying the result on the screen, or saving it to a local R object:

```
command_4
## # Source:     spark<?> [?? x 2]
## # Ordered by: desc(N)
##   carrier     N
##   <chr>   <dbl>
## 1 UA      58665
## 2 B6      54635
## 3 EV      54173
```

```
## 4 DL     48110
## 5 AA     32729
result <- collect(command_4)
result
## # A tibble: 5 x 2
##   carrier    N
##   <chr>  <dbl>
## 1 UA     58665
## 2 B6    54635
## 3 EV     54173
## 4 DL     48110
## 5 AA     32729
```

In the second case, we used the special collect() command from the dply package. In a sense, collect() is the opposite of copy_to(), which was used above to import data from R into Spark.

Dplyr package has several functions for performing standard JOIN operations on two tables: inner_join (), left_join (), right_join (), full_join (), semi_join (), nested_join (), and anti_join (). The following example performs a LEFT JOIN of the flights_tbl table with the airlines_tbl table on the carrier field:

```
flights_tbl %>%
left_join(airlines_tbl, by = "carrier") %>%
glimpse()
## Observations: ??
## Variables: 20
## Database: spark_connection
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 201...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, ...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 91...
```

```
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 85...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14...
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA...
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696,...
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N8...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA"...
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL"...
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719,...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, ...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0...
## $ time_hour      <dttm> 2013-01-01 10:00:00, 2013-01-01 ...
## $ name           <chr> "United Air Lines Inc.", "United ...
```

Although standard dplyr commands and some basic R functions are automatically converted to SQL, they will not be sufficient for developing more complex calculations on data using Spark.

Spark SQL is based on the Hive Query Language (HiveQL) and all of its features can be used in conjunction with **dplyr** commands. For example, to calculate the median value of the variable dep_delay (flight delay, min) from the table flights_tbl, we cannot use the basic R functions median () or quantile (), which would result in an error. We can use the Hive function percentile ():

```
flights_tbl %>%
summarise(median = percentile(dep_delay, 0.5))
## # Source: spark<?> [?? x 1]
##   median
##    <dbl>
## 1     -2
```

When translating R code to SQL, when dplyr encounters an unfamiliar function, it simply includes it in the SQL query "as is":

```
flights_tbl %>%
summarise(median = percentile(dep_delay, 0.5)) %>%
show_query()
```

```
## <SQL>
## SELECT percentile(`dep_delay`, 0.5) AS `median`
## FROM `flights`
```

Hive percentile() function allows calculate multiple percentiles at once. To do this, we need to pass it an array (array()) with the required percentile values:

```
flights_tbl %>%
  summarise(perc = percentile(dep_delay, array(0.25, 0.5, 0.75)))
# Source: spark<?> [?? x 1]
  perc
  <list>
1 <list [3]>
```

The result of the above command is a list with three values. To automatically extract these values from the list, we use the Hive explode() function:

```
flights_tbl %>%
summarise(perc = percentile(dep_delay, array(0.25, 0.5, 0.75))) %>%
mutate(perc = explode(perc))
## # Source: spark<?> [?? x 1]
##    perc
##   <dbl>
## 1    -5
## 2    -2
## 3    11
```

Let's apply the knowledge gained during exploratory analysis of data from the flights_tbl table. Let's find out if there are missing values in our data. There are several ways to do this. The missing values are counted for all columns of the flights_tbl table using the summarise_each() command from the **dplyr** package in combination with an anonymous function that specifies the calculation logic:

```
flights_tbl %>%
summarise_each(list(~sum(as.integer(is.na(.))))) %>%
glimpse
```

```
## Observations: ??
## Variables: 19
## Database: spark_connection
## $ year           <dbl> 0
## $ month          <dbl> 0
## $ day            <dbl> 0
## $ dep_time       <dbl> 8255
## $ sched_dep_time <dbl> 0
## $ dep_delay      <dbl> 8255
## $ arr_time       <dbl> 8713
## $ sched_arr_time <dbl> 0
## $ arr_delay      <dbl> 9430
## $ carrier        <dbl> 0
## $ flight         <dbl> 0
## $ tailnum        <dbl> 2512
## $ origin         <dbl> 0
## $ dest           <dbl> 0
## $ air_time       <dbl> 9430
## $ distance       <dbl> 0
## $ hour           <dbl> 0
## $ minute         <dbl> 0
## $ time_hour      <dbl> 124
```

As we can see, missing values do occur in some variables. The maximum number of missing values reaches 9430, which is less than 3% of the total number of observations in the table (336776). The dimensionality of the table can be found using the sdf_dim() command from the **sparklyr** package, which is an analogue of the basic R dim() function.

In the **sparklyr** package whose names begin with sdf_ (for "Spark data frame"), such as sdf_nrow (), sdf_ncol (), sdf_bind_rows (), sdf_pivot (). Since the proportion of missing values is small, we can remove the corresponding rows from the table without much risk of affecting the quality of further analysis. To do this, use the basic function na.omit():

```
flights_full <- flights_tbl %>% na.omit()
## * Dropped 9554 rows with 'na.omit' (336776 => 327222)
```

```
flights_full %>% sdf_dim()
## [1] 327222     19
```

Since we are interested in flights with delays of 15 to 30 minutes (inclusive), we need to filter the data accordingly. In parallel, we add a new column target with the values of the dependent variable:

```
flights <- flights_full %>%
  filter(dep_delay >= 15, dep_delay <= 30) %>%
  mutate(target = as.integer(arr_delay <= 0))
flights %>% sdf_dim()
## [1] 24507    20
```

We ended up with a table with 24507 rows and 20 columns. This is a small table and could be imported from Spark into R (using the collect() command) for further analysis right now. For the sake of demonstrating the capabilities of Spark with R, we will leave this table in the cluster's memory.

Let's find out which of the available variables correlate with the dependent variable target. It would be logical to expect that the probability of a delayed flight arriving on schedule is largely determined by the distance between the departure airport and the arrival airport (the distance column, expressed in miles). We calculate the median value of this distance for both classes as the dependent variable:

```
flights %>%
group_by(target) %>%
summarise(median_dist = percentile(distance, 0.5))
## # Source: spark<?> [?? x 2]
##  target median_dist
##   <int>       <dbl>
## 1      0         820
## 2      1        1089
```

Indeed, the greater the distance between airports, the greater the chance that a delayed flight will make up for lost time and arrive without delay.

Conclusion to lecture 17

Apache Spark is a unified, open-source analytics engine for large-scale data processing. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Apache Spark also supports structured databases. Apache Spark is a powerful open-source data processing framework known for its speed, scalability, and ease of use in big data analytics. It supports in-memory computing, which allows data to be processed much faster compared to traditional disk-based systems like Hadoop MapReduce. Spark also provides high-level APIs in Java, Scala, Python, and R, making it accessible to a wide range of developers. Apache Spark processes real-time data using Spark Streaming and Structured Streaming. It ingests data from sources like Kafka, splits it into micro-batches, and processes it quickly in memory. Structured Streaming simplifies this by treating data as a live table, enabling real-time analytics with fault tolerance.

Questions for reinforcement

1. What is the problem of the computational function?
2. Describe the features of Spark technology.
3. What is the difference between Spark and MapReduce?
4. What packages in R are used to work with Spark?
5. How does in-memory computing improve Spark's performance?
6. In which programming languages can you write Spark applications?
7. How is Spark used for real-time stream processing?

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. Apache Spark. URL: https://spark.apache.org/
3. Sparklyr. URL: https://spark.posit.co/
4. tidyverse/nycflights13. URL: https://github.com/tidyverse/nycflights13

Lecture 18.
Lambda and Kappa architectures for big data processing

Lecture plan

18.1. Lambda architecture.
18.2. Advantages and disadvantages of Lambda architecture.
18.3. Kappa - architecture.
18.4. Advantages and disadvantages of Kappa architecture.

18.1. Lambda - architecture

In an effort to bring the analysis of “historical” data closer to that of real-time data [1], the Lambda architecture was created (Fig. 18.1).

Lambda is a data processing architecture that uses both stream processing and batch processing to get an accurate view of both "live" data and batch data.

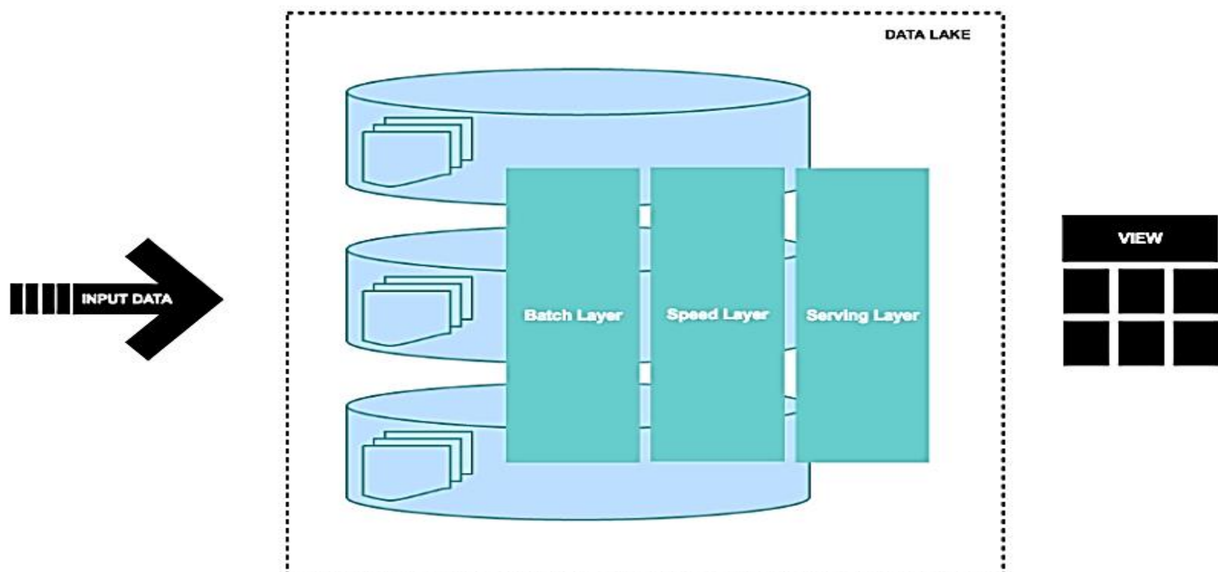


Fig. 18.1. Generalized Lambda architecture [2]

The Lambda architecture has four layers (Fig. 18.2).

Ingestion – this layer imports data. This can be data from many sources, including data streams.

Batch is data at rest. The data here is often built on a schedule and involves importing data from the streaming layer. Accuracy is more important than speed.

Stream is a complex layer that supports incremental updates. Low latency is a higher priority here than accuracy.

Presentation – this layer launches the operation and accepts all requests and uses the speed or packet layer.

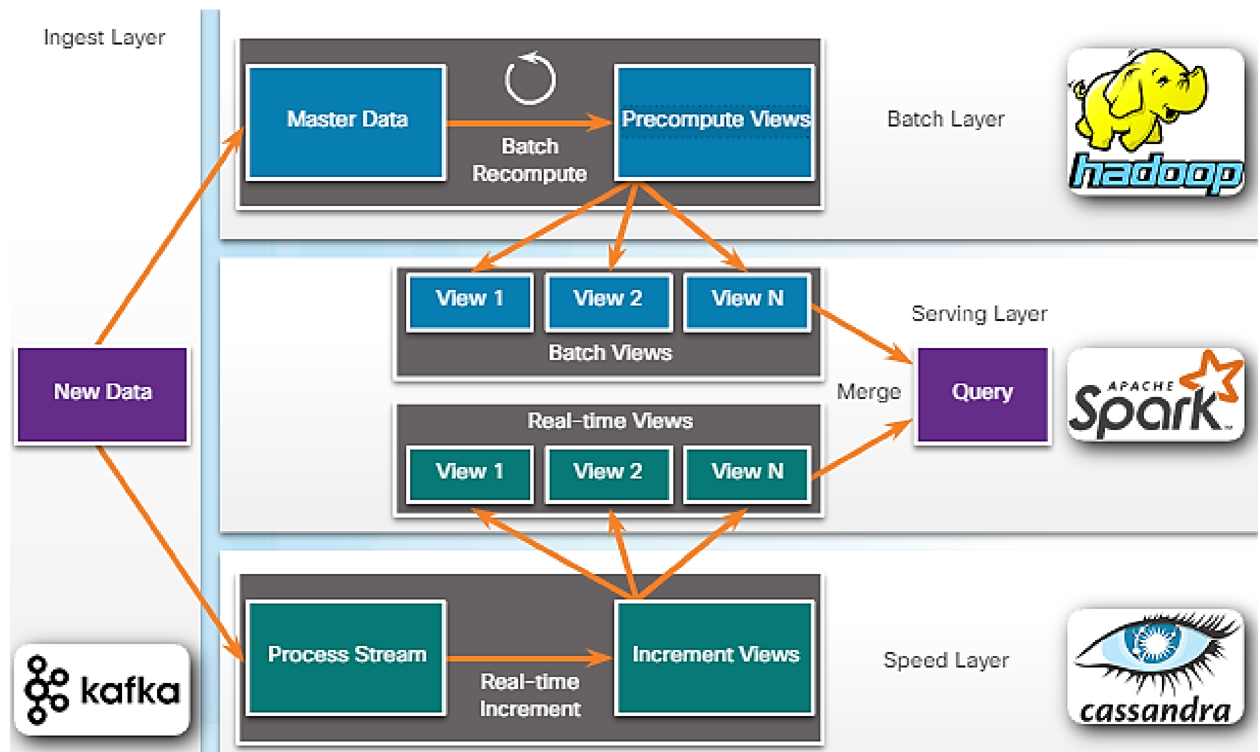


Fig. 18.2. Lambda technologies – architectures [1]

All incoming data is sent simultaneously to both the packet layer and the speed layer. The packet layer manages the underlying data set and pre-computes packet views that are continuously computed. The service layer indexes the packet views so that they can be queried.

The speed of the layer only applies to the latest data, which compensates for the latency of updates to the layer serving the data. Two different query results can be combined to form a new data type.

IoT is an ideal area for implementing Lambda architecture. Data is generated at high speed, datasets can be very large, and queries can be for both data at rest and data in motion.

Let's look at a real-life example of using Lambda architecture to visualize 171 bus routes in Los Angeles, California [1]. Lambda includes SACK (Spark, Akka, Cassandra, Kafka). This allows for real-time data analytics.

Akka is a free, open-source tool for building distributed and resilient message-driven applications [3].

Akka is used to retrieve the route information metadata and store it in Cassandra every 30 seconds. In this example, Akka uses a free REST API to query the data. The data is then stored in Cassandra. Subsequent queries are sent to Kafka. Spark is then used to read the vehicle information from Kafka. Once all this data is available, another API is used to visualize the bus positions using OpenStreetMap. The current bus positions are transmitted directly from Kafka to the map using a web conversation [3, 4].

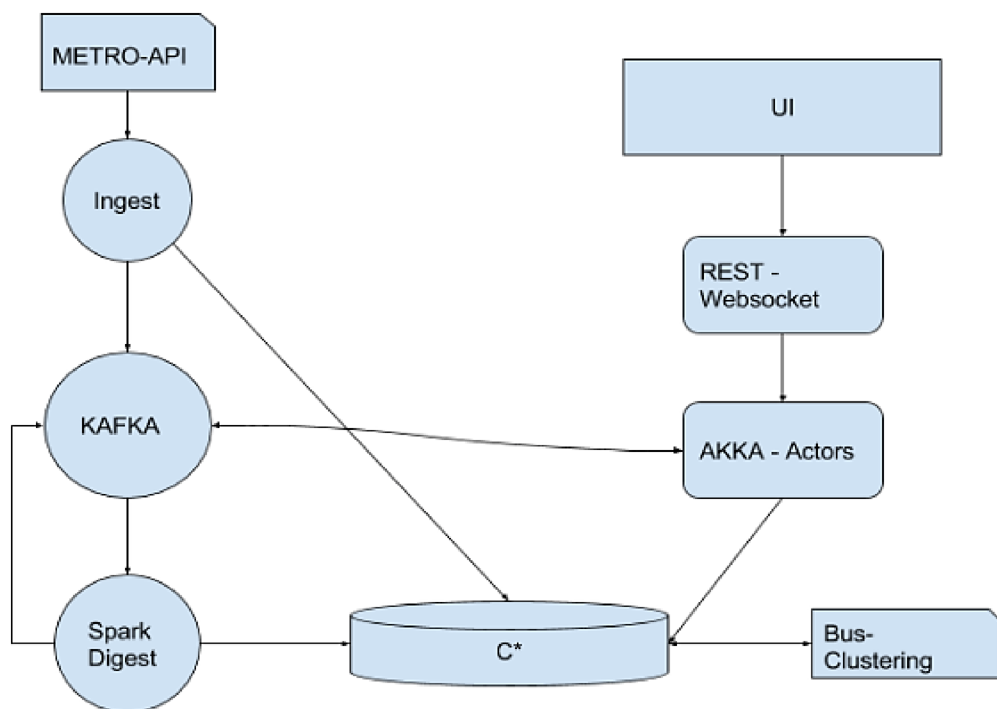


Fig. 18.3. IoT Analytics Platform Architecture (Ingest – Akka, Digest – Spark, UI – Javascript, Openstreetmap, Backend – Akka) [3]

This platform is not limited to IoT data. It is suitable for solutions where there is a lot of input data. It is very useful when there are multiple parallel input data streams. It is a better solution than traditional RDBMS which will not be able to provide real-time data.

Lambda architecture can be deployed for those enterprise data processing models where user requests must be processed using an immutable data store; fast responses are required, the system must be able to handle various updates in the form of new data streams; none of the stored records must be deleted, and this should allow updates and new data to be added to the database. Companies such as Twitter, Netflix, and Yahoo use this architecture to meet quality of service standards.

18.2. Advantages and disadvantages of Lambda architecture

Lambda architecture has the following *advantages* for Big Data systems:

- historical data storage at the batch level based on Hadoop Data Lake or other fault-tolerant distributed storage with a low probability of errors and failures;
- balance of speed and reliability;
- scalability.

The Lambda approach has the following *disadvantages*:

- impossibility of changing data analysis strategy "on the fly" – the ultimate consistency of data makes it impossible to send information back to the batch level. All calculations must be repeated, due to the binding to the storage, the data is difficult to transfer or reorganize;
- most tools focused on Lambda architecture are NoSQL and do not support SQL queries or other business intelligence tools;
- many disparate components that communicate with each other, which delays real-time computing, information processing logic is duplicated using different data structures, making overall management difficult. Development overhead also increases.

Such shortcomings are partially compensated by Apache Spark.

For fast real-time event processing without batch representation, another approach is more appropriate – the Kappa architecture (Fig. 18.4).

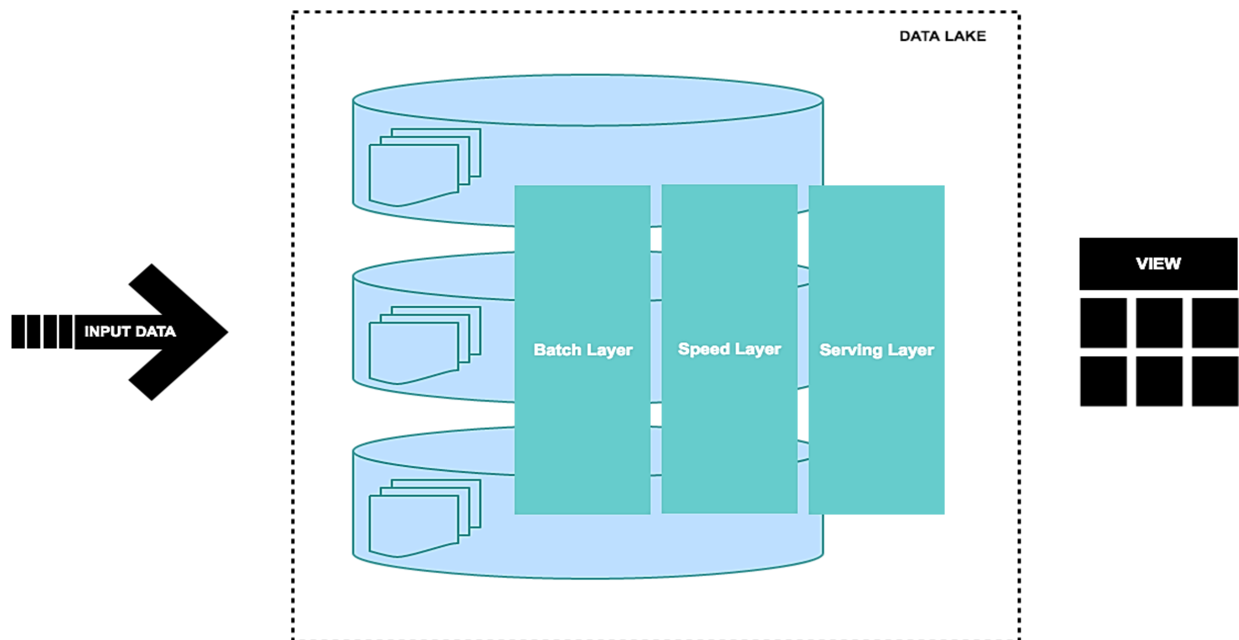


Fig. 18.4. Generalized Kappa architecture [3]

Some variants of applications for social networks, cloud-connected devices, and the Internet of Things use an optimized version of the Lambda architecture, which uses speed-layer services combined with a streaming layer to process data through a Data Lake.

18.3. Kappa – architecture

In 2014, Jay Kreps started a discussion where he pointed out some inconsistencies in the Lambda architecture, which subsequently led the big data world to another alternative architecture – Kappa, which used less code resources and was able to work well in certain enterprise scenarios.

The Kappa architecture can be deployed for those enterprise data processing models where:

- multiple events or data requests are queued for processing in the distributed file system storage or in history;

- the order of events and requests is not defined, streaming platforms can interact with the database at any time;
- processing terabytes of storage is required for each system node to support replication.

The above data scenarios are processed using the Apache Kafka platform, which provides computing speed, fault tolerance, and horizontal scalability. This allows for improved data flow management.

Balanced management of stream processors and databases allows applications to perform as expected by Kafka, keeps data in order for longer, and serves similar queries by relating them to the corresponding position in the stored log in real time.

LinkedIn and some other applications use this type of big data processing and benefit from storing large amounts of data to serve requests that are simply copies of each other.

The Kappa architecture cannot be considered a replacement for the Lambda architecture, but should be considered an alternative to be used in cases where active batch-level performance is not required to provide a standard quality of service.

18.4. Advantages and disadvantages of Kappa architecture

Consider the *advantages* of the Kappa architecture.

The Kappa architecture can be used to develop data systems that learn online and therefore do not require a batch layer:

- reprocessing is only required when the code changes;
- can be used for horizontally scalable systems;
- are required, as machine learning is performed in real time.

Let's look at the *disadvantages* of the Kappa architecture. The lack of a batch layer can lead to errors when processing data or when updating the database, requiring an exception handler to reprocess the data.

The choice between Lambda and Kappa architectures seems like a compromise. If we want an architecture that is more robust in updating the Data Lake, and also efficient in developing machine learning models to reliably predict new events, we should use the Lambda architecture, as it takes advantage of the batch layer and the speed of the layer to ensure fewer errors.

If we need to deploy a big data architecture using less expensive hardware and require it to process efficiently based on unique events that occur during execution, it makes sense to use the Kappa architecture for real-time data processing.

Conclusion to lecture 18

Lambda is a data processing architecture that uses both streaming and batch processing to get an accurate view of both live and batch data.

Benefits of Lambda architecture are next:

1. the batch layer of the Lambda architecture manages historical data using fault-tolerant distributed storage, which ensures a low probability of errors even if a system failure occurs;

2. it is a good balance of speed and reliability;

3. fault-tolerant and scalable architecture for data processing.

Disadvantages of Lambda architecture are next:

1. overhead and coding costs due to the use of complex processing;

2. repeated processing each batch cycle in, which is not beneficial in certain scenarios;

3. data modeled using the Lambda architecture is difficult to migrate or reorganize.

The Kappa architecture can be used to develop intelligent data systems that learn online and therefore do not require a batch layer, where reprocessing is only needed when the code changes, can be used for horizontally scaled systems where fewer resources are required, as machine learning is performed in real time.

Questions for reinforcement

1. Describe Lambda – a big data processing architecture.
2. Describe the advantages and disadvantages of the Lambda architecture.
3. Describe Kappa - a big data processing architecture.
4. Describe the advantages and disadvantages of the Kappa architecture.
5. Describe examples of using Lambda and Kappa architectures for Big Data.

List of recommended literature

1. Data Analytics Essentials. URL: https://www.netacad.com/courses/iot/big-data-analytics
2. IoT Analytics Platform. URL: https://blog.codecentric.de/en/2016/07/iot-analytics-platform/
3. Akka. URL: https://akka.io/
4. IoT-Analyse-Plattform: Floating Bus Data. URL: https://www.youtube.com/watch?v=VYxc-3ZRRL4&ab_channel=codecentricAG
5. A brief introduction to two data processing architectures — Lambda and Kappa for Big Data. URL: https://towardsdatascience.com/a-brief-introduction-to-two-data-processing-architectures-lambda-and-kappa-for-big-data-4f35c28005bb