МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ СПЕЦІАЛЬНОГО ЗВ'ЯЗКУ ТА ЗАХИСТУ ІНФОРМАЦІЇ

Д.М. ШАРАДКІН, І.Ю. СУБАЧ, А.В. МИКИТЮК

ІНСТРУМЕНТАЛЬНІ ЗАСОБИ PYTHON ДЛЯ МОДЕЛЮВАННЯ ТА СИСТЕМНОГО АНАЛІЗУ ЧАСОВИХ РЯДІВ ПРИ ВИРІШЕННІ ЗАДАЧ КІБЕРЗАХИСТУ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ СИСТЕМ

Навчальний посібник

Рекомендовано Вченою радою ІСЗЗІ КПІ ім. Ігоря Сікорського
для використання у навчальному процесі з підготовки
фахівців першого (бакалаврського) рівня вищої освіти
зі спеціальності 122 Комп'ютерні науки

Київ
ІСЗЗІ КПІ ім. Ігоря Сікорського
2023

УДК 004.942 (075.8)

Ш 25

Розглянуто Вченою радою
ІСЗЗІ КПІ ім. Ігоря Сікорського
(Протокол № 8 від 26.01.2023 р.)

Рецензент: *Д.М. Могилевич, д-р. тех. наук, проф.*

О.Г. Кофто, канд. тех. наук, с.н.с.

Шарадкін Д.М., Субач І.Ю., Микитюк А.В.

Ш25 Інструментальні засоби Python для моделювання та системного аналізу часових рядів при вирішенні задач кіберзахисту інформаційно-комунікаційних систем: навч. пос. / Шарадкін Д.М., Субач І.Ю., Микитюк А.В.; ІСЗЗІ КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2023.- 139 с.

ISBN 978-966-2577-18-1

Навчальний посібник знайомить курсантів (студентів) з засобами мови програмування Python та модулів його розширення, які використовуються при виконанні завдань математичного моделювання динамічних систем на основі часових рідів. Розглядаються питання, що виходять за рамки тем традиційних підручників з мови програмування та вимагають більш глибокого врахування особливостей створення програмних застосунків в предметних галузях практичного застосування вказаних задач. Детально та послідовно розглянуті методи роботи з типами даних, що використовуються для визначення та маніпулювання часовими мітками та інтервалами, і які є основою та обов'язковим компонентом будь якого часового ряду; інструментальні засоби первинної статистичної обробки даних та визначення параметрів вибірок даних; різноманітні засоби візуалізації даних та результатів їх дослідження, включно з представленням та аналізом гістограм. Обраний набір засобів дозволяє курсантам (студентам) перейти від вивчення теоретичних засад опису та моделювання систем до безпосереднього практичного їх аналізу. Розгляд тем супроводжується великою кількістю прикладів, що дозволяє курсантам (студентам) побачити специфіку використання математичного та статистичного апарату, визначити межі можливостей застосування відповідних програмних інструментів, звернути увагу на прикладний характер отриманих знань. Значну увагу приділено питанням використання отриманих навичок при вивчені задач технічної діагностики та задач, що пов'язані з кіберзахистом інформаційно-комунікаційних систем.

Навчальний посібник призначено для курсантів та студентів, що навчаються за спеціальністю 122 Комп'ютерні науки, а також для всіх, хто вивчає курси «Системний аналіз» та «Моделювання систем».

УДК:004.942 (075.8)

ISBN 978-966-2577-18-1

ЗМІСТ

ВСТУП

В сучасних умовах до спеціаліста в галузі інормаційних технологій пред'являються підвищені вимоги не тільки з точки зору його професійних якостей, але і з точки зору рівня володіння культурою та методикою проведення інженерно-технічного та наукового дослідження. Знання сучасних математичних методів аналізу та моделювання систем, вміння застосувати ці знання до вирішення прикладних задач, вільне володіння засобами програмування, які дозволяють довести розрахунки та дослідження до завершення та практичного використання – все це має бути присутнім в арсеналі сучасного спеціаліста.

Завдяки мережі інтернет на сьогоднішній день став доступним величезний обсяг інформації, в яких глибоко і детально описуються сучасні математичні методи дослідження систем. З іншого боку також існує така ж кількість джерел, що орієнтовані на вивчення мов та засобів програмування. Однак виявляється, що майже відсутня спеціальна література, з якої студенти та курсанти можуть набути вміння саме застосування універсальних засобів програмування до рішення математичними методами конкретних прикладних задач. Метою даного посібника є спроба заповнити цю прогалину, показати які саме програмні засоби та яким саме чином можуть слугувати основою для аналізу систем, об'єктів та процесів реального світу.

Навачльний посібник орієнтований на студентів та курсантів, що вже оволоділи навичками програмування, зокрема мовою програмування *Python*, набули уміння самостійного складання скриптів на цій мові хоча б в одному інтегрованому середовищі розробки (*Spyder, PyCharm, MS Visual Studio, Jupiter* тощо), ознайомлені з деякими додатковим модулями, що використовуються в інженерній та науковій роботі, зокрема з модулями *numpy* та *pandas*, та переходять до вивчення методів дослідження та моделювання систем, зокрема систем, що характеризуються статистичною невизначеністю да динамічністю поведінки. Опис самих методів аналізу та моделювання не входить в коло проблем, що розглядаються. Мета посібника – ознайомити читача саме з інструментами, що застосовуються при програмування на *Python* та можуть використовуватися при рішенні вказаних задач. Щоб сформувати у читача ширший професійний кругозір, підкреслити універсальність як самих методів так і інструментів для їх програмної підтримки, приклади, що розглядаються в посібнику, відносяться до різних прикладних галузей. Однак значна частина з цих прикладів тим чи іншим чином має відношення до систем, пов'язаних з

задачами забезпечення інформаційної безпеки. Це має закласти підвалини навичок використання представлених програмних інструментів і при рішенні інших задач в цій прикладній галузі.

В прикладах, наведених в посібнику, використовується версія 3.9 мови програмування *Python*. Навчальний посібник дає уявлення про роботу з інструментами таких модулів, що входять до екосистеми *Python*, як *time, datatime, matplotlib, sctatistics, scipy*. Окрему увагу приділяються засобам модулів *numpy* та *pandas*, які дають змогу форматування, маніпулювання, зберігання, обробки даних, що представляють часові ряди.

Основними джерелами інформації для дослідника, що займається застосуванням методів аналізу та моделювання часових рядів до вирішення прикладних задач, створенням відповідних програмних застосунків, були і залишаються інтернет-сайти з документацією з відповідних модулів розширення *Python*. Документацію з модулів, що розглянуті в посібнику, можна знайти в електронній формі за посиланнями:

модуль *time*:
https://docs.python.org/3/library/time.html
модуль *datetime*:
https://docs.python.org/3/library/datetime.html
модуль *statistics*:
https://docs.python.org/3/library/statistics.html
модуль *numpy*:
https://numpy.org/doc/stable/
модуль *pandas*:
https://pandas.pydata.org/docs/
модуль *scipy.stats*:
https://docs.scipy.org/doc/scipy/reference/stats.html
модуль *matplotlib*:
https://matplotlib.org/stable/index.html

Саме в цих джерелах міститься найбільш повний опис всіх можливостей, методів, параметрів, форматів тощо, які реалізовані в цих інструментах. Тому при виникненні будь яких неясностей, незрозумілостей, неоднозначностей трактування, помилок в програмному коді тощо дослідник має звертатися саме до документації з відповідного програмного інструменту. Однак досвід викладання показує, що студентам та курсантам часто досить важко в ній зорієнтуватися, зрозуміти не просто формати відповідних методів та функцій, а й набути навичок при рішенні конкретних прикладних задач. Навчальний

посібник, не заміняючи і не виключаючи необхідність подальшої роботи з документацією модулів, надає можливість читачу ознайомитися з основними конструкціями модулів, їх комбінаціями, зрозуміти основні принципи та ідеї, які були реалізовані авторами вказаних програмних інструментів.

Враховуючи, що наразі все більше студентів та курсантів мають достатній рівень володіння англійською мовою та виявляють бажання оволодівати сучасними знаннями використовуючи англомовні джерела інформації, в кінці посібника наведено короткий список відповідної літератури, що була надрукована за останні роки, та сконцентрована саме на питаннях застосування мови *Python* та його модулів при дослідженні часових рядів.

Оскільки багато термінів, що використовуються в програмуванні, запозичені з англійської мови, для деяких з них ще не виробилися сталі та – головне – прийнятні більшістю професійної спільноти терміни. Тому в посібнику для таких термінів ми будемо використовувати їх первинне, англійське позначення, а не переклад, чи мовну «кальку». Так наприклад, ми будемо писати «*DataFrame*», а не «об'єкт типу таблиця даних», чи «датафрейм»; «*Series*», а не «об'єкт типу послідовність», чи «об'єкт типу серія» тощо. Такий підхід є звичайним в професійній літературі і не має погіршити сприйняття читачем змісту викладеного матеріалу.

Навчальний посібник містить досить багато програмного коду, який читачу рекомендується самостійно відтворити та проаналізувати. Для спрощення перевірки результатів, при поданні програмного коду використовується окремий шрифт. Для позначення відповіді, яка формується при виконанні коду та яка виводиться при цьому в консоль, використовується спеціальний символ, а саме >>>> .

В навчальному посібнику досить багато інформації представлено в вигляді рисунків. Для зручності сприйняття рисунки нумеруються всередині глав. В главах 5 та 6, в яких рисунки є не ілюстрацією до тексту, а результатом роботи програми, рисунок розміщується безпосередно за кодом програми, одразу після символу >>>> .

Посібник покликаний дати читачу перший поштовх в напрямку практичного застосування засобів програмування до вирішення практичних задач дослідження даних. Подальший розвиток студентів та курсантів полягає в поглиблені як їх теоретичних знань, так і в розвитку своєї культури та навичок програмування, розширення арсеналу інструментальних засобів дослідження.

1. ЧАСОВІ РЯДИ – ІНСТРУМЕНТ МОДЕЛЮВАННЯ ДИНАМІЧНИХ СИСТЕМ

У найширшому визначенні ***часовий ряд – це послідовність упорядкованих у часі числових показників, що характеризують стан і зміни об'єкта, процесу або явища що досліджується, в конкретний момент часу.*** Приклади часових рядів ми зустрічаємо на кожному кроці: погодинна кількість переглядів сторінок веб-сайту, добова температура повітря у місті, рівень води у водоймі чи річці, середньомісячна кількість зареєстрованих звернень до служби підтримки, квартальний прибуток компанії, річні дані про населення країни, концентрація певної речовини у розчині під час протікання деякого хімічного процесу тощо. Кожен часовий ряд включає два обов'язкові елементи: по-перше – час і, по-друге, однозначно пов'язане з цим часом значення обраного показника (або, в разі багатовимірних даних – показників).

Час, інформація про який надається в часовому ряду, може визначатися як конкретний момент. Наприклад, в контексті технології IoT (Internet of Things – Інтернет Речей) у вигляді часових рядів представляються значення, що отримуються від різноманітних датчиків (наприклад – фізичні значення, такі як тиск, температура, електрична напруга, тощо). В таких часових рядах при характеристиці часу часто використовують термін «мітка часу», а сам часовий ряд називають «моментним часовим рядом».

В інших прикладних задачах можлива інша інтерпретація часового ряду, а саме як інтервального ряду. Інтервальний часовий ряд – це послідовність значень, у якій дані відносяться до результату, накопиченого за певний інтервал (проміжок) часу. Такі, наприклад, ряди показників обсягу продукції підприємства за місяцями року, кількості відпрацьованих людино-днів за окремими періодами (місяцями, кварталами, півріччями, роками) тощо. Часто моментний часовий ряд може бути легко перетворений на інтервальний шляхом об'єднання даних протягом деякого заданого проміжку часу. Перетворення одного типу ряду на інший можуть виконуватися у зв'язку з тим, що вихідне уявлення слабо піддається осмисленню та аналізу. Наприклад відомо, що інтернет-трафік представляє собою послідовність пакетів інформації певної довжини, що надходять у довільні моменти часу. Проте найчастіше найважливішим показником для задач моніторингу трафіку є його обсяг, тобто кількість інформації, що була передана через мережу за певний відрізок часу – мілісекунду,

секунду, годину тощо. В даному разі перетворення між моментною та інтервальною формою представлення часового ряду може бути виконане шляхом сумування обсягів усіх пакетів, зафіксованих у мережі за певний проміжок часу. Вихідний ряд при цьому є моментним рядом, а ряд, що отримується в результаті зазначеного перетворення – інтервальний.

Математично часовий ряд визначається як набір векторів $Y = \{\vec{y}(t)\}$, $t = 0,1,2, \ldots$ де t — моменти часу, в які спостерігаються відповідні значення ряду. Вектор $\vec{y}(t)$ розглядається як випадкова змінна. Вимірювання, подані в часовому ряду завжди (якщо це не обумовлено окремо) розташовані у відсортованому за хронологією їх отримання порядку. Самі виміри називають також значеннями, рівнями, спостереженнями чи точками ряду.

Переважна більшість часових рядів складаються з даних, що пов'язані з рівновіддаленими між собою моментами часу (т.з. еквідистантні часові ряди) або з інтервалами однакової величини, наприклад, погодинні вимірювання температури, щоденні підрахунки відвідувань веб-сайтів або місячні підсумкові об'єми переданого трафіку. Часові ряди також можуть бути нерівномірними та спорадичними, наприклад, дані з часовими мітками в журналі подій комп'ютерної системи, історія екстрених викликів служби 103 або моменти часу появи в мережі пакетів даних.

Таким чином, ***часові ряди це деяка модель, в результаті вивчення якої ми сподіваємося отримати знання про стан і поведінку самого об'єкта***. Дослідження часового ряду – це формалізована процедура, спрямована на те, щоб зрозуміти природу об'єкта, значення параметрів якого формують часовий ряд і висловити це розуміння у вигляді відповідної моделі, здатної описати його поведінку. Різноманіття прикладних задач, що зводяться до аналізу часових рядів дійсно вражає. Але найбільш захоплюючим є те, що вивчивши методи такого аналізу та програмні інструменти, які при цьому використовуються, перед дослідником відкривається величезний простір для їх застосування.

Якщо уявити, що дослідник знаходиться в певній часовій точці, структура часового ряду підказує, що задачі, які можуть розглядатися при його дослідженні, поділяються на дві групи. А саме – задачі, що стосуються часу, якій відносно положення дослідника знаходяться в минулому, і ті, які відносно дослідника знаходяться в майбутньому.

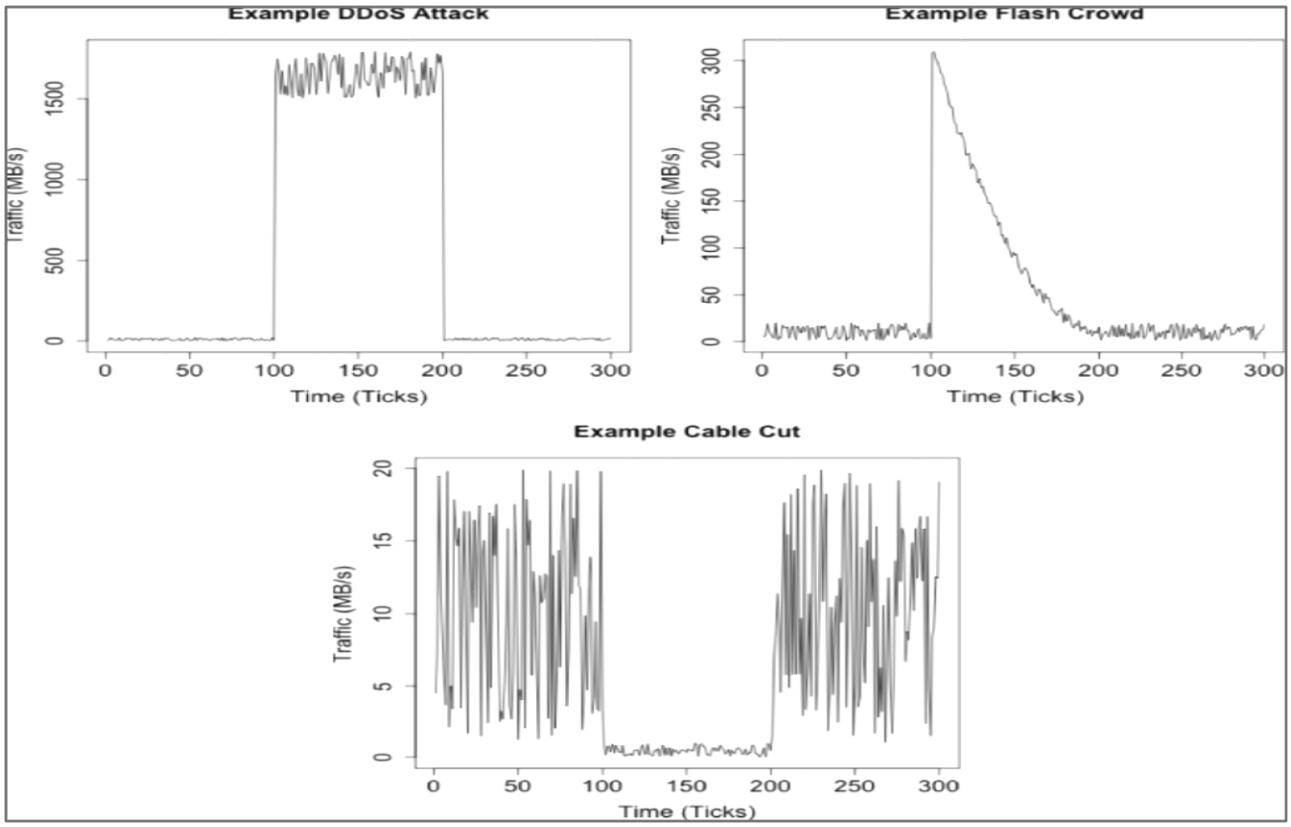


Рис. 1.1 – Приклади аномалій в даних моніторингу мережевого трафіку. (Джерело: Collins M. Network Security Through Data Analysis, O'Reilly Media, Inc.,2016.- p.252.)

Перший тип задач – це аналітика. Задачі цього типу полягають в спробі вивчення даних з минулого, та пошуку відповіді на запитання "що трапилося" та "чому це сталося"? Цей тип задач часто називають також задачами «виявлення аномалій», оскільки в такій постановці нас цікавлять моменти, коли об'єкт спостереження – а точніше часовий ряд, за яким ми спостерігаємо – в силу певних, як правило нам невідомих, причин суттєво змінює свою поведінку відносно звичайної («нормальної»).

Виявлення аномалій – одне з найважливіших завдань інтелектуального аналізу даних. Це завдання вирішується у багатьох важливих прикладних областях, таких як виявлення мережевих атак (Intrusion Detection), виявлення шахрайства (Fraud Detection), зокрема з кредитними картками, виявлення аномалій у медицині (Medical Anomaly Detection), виявлення деградації обладнання для запобігання його остаточної руйнації тощо. На Рис. 1.1 наведені деякі зразки часових рядів, за якими відслідковується стан об'єкта дослідження – в даному випадку інтернет-мережа. З малюнку зрозуміло, що відображені зміни

параметрів, за якими ведеться спостереження, зумовлені суттєвою зміною стану самого об’єкту, які в свою чергою можуть бути викликані як спробою зловмисного втручання в роботу мережі, так і відмовою обладнання мережі.

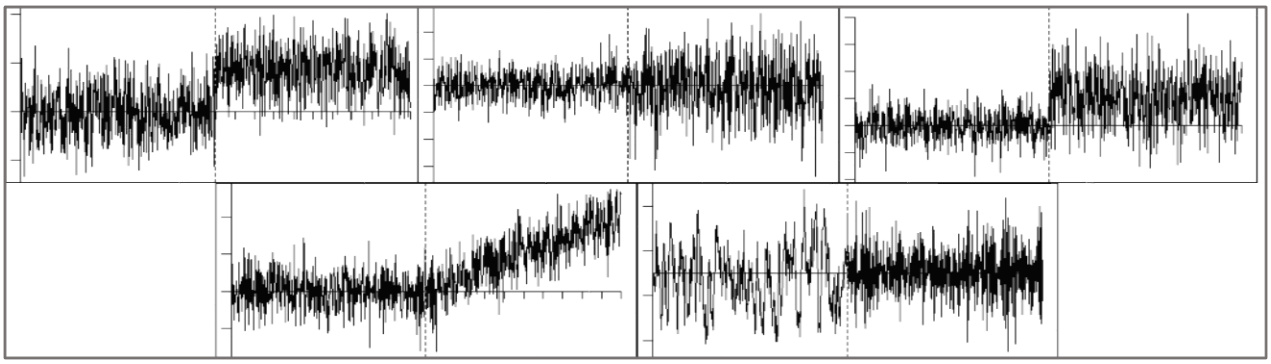


Рис. 1.2 – Деякі можливі прояви аномалій в часових рядах

Формальний аналіз часового ряду здатен виявити точки часу, в яких відбувається відчутна зміна параметрів часового ряду, а отже – зміна в поведінці самого об’єкту дослідження. Однак не завжди такі зміни можна виявити так просто. На Рис. 1.2 наведені зразки змін в рядах, виявити які, а тим більш точно вказати момент, в який розпочинається така зміна, вже не так легко, як в минулих прикладах.

На Рис. 1.3 представлені ще декілька прикладів аномалій. Людина здатна

Рис. 1.3 – Деякі можливі прояви аномалій в часових рядах

легко виявити такі аномальні ситуації візуально, аналізуючи графічне відображення часового ряду. Однак автоматизувати таке розпізнавання виявляється досить непростою задачею, особливо в загальній ситуації, коли ані опису часового ряду при «нормальній» поведінці об’єкту, ані тим більш опису всіх можливих відхилень-аномалій наперед задати навіть теоретично не виявляється можливим.

Основна задача другого типу – це задача *прогнозування майбутніх значень часового ряду* на основі значень ряду, отриманих в минулому. Це, мабуть, найвідоміше і найбільш ефектне застосування часових рядів. Прогнозування того, що відбудеться на фінансовому ринку завтра, обсягу товарів, які будуть продані протягом найближчого тижня, кількість поїздок Uber за наступні три години, об'єм інтернет-трафіку у провайдера та кількості кібератак на сайт в найближчий місяць, кількість людей, що захворіють на Covid наступного тижня – ось лише деякі приклади прогнозування, які можливо спробувати виконати за допомогою прогнозного дослідження часових рядів.

Аналіз і прогнозування часових рядів тісно пов'язані між собою. Як правило, при вирішенні однієї з цих задач використовуються результати, отримані при вирішенні іншої. І обидві ці задачі спираються на те, що перш, ніж приступити до їх вирішення, повинна бути побудована *модель поведінки об'єкта*.

На сьогоднішній день відомо досить велике число різноманітних моделей часових рядів, які використовуються при наукових дослідженнях і при вирішенні конкретних прикладних задач. Серед них можна згадати найпростіші моделі ковзаючого згладжування, моделі експоненціального згладжування, моделі з урахуванням сезонних складових, лінійні та нелінійні регресійні моделі, різноманітні авторегресійні моделі (ARMA,ARIMA,SARIMA тощо), моделі з використанням ланцюгів Маркова, нарешті моделі на основі генетичних алгоритмів та нейронних мереж різного типу. Вивчення цих моделей і їх застосування до вирішення задач аналізу та прогнозування розглядаються в відповідних курсах, що викладаються студентам та курсантам.

Побудові будь якої моделі часового (Рис. 1.4) ряду має передувати дослідження властивостей ряду, починаючи з визначення закону розподілу, якому підпорядковуються значення ряду. В разі якщо такий закон має теоретичне визначення – оцінка параметрів цього розподілу, точкова та інтервальна, визначення статистичних характеристик вибірки даних, визначення стаціонарності ряду, сезонності тощо. Зазначені дослідження носять абсолютно універсальний характери та ніяким чином не залежать ані від типу моделі яка буде будуватися, ані від того, чи дослідження виконується з метою пошуку аномалій в об'єкті, чи буде будуватися передбачення поведінки об'єкту в майбутні періоди, ані тим більше від того, в якій саме прикладній галузі отримані дані.

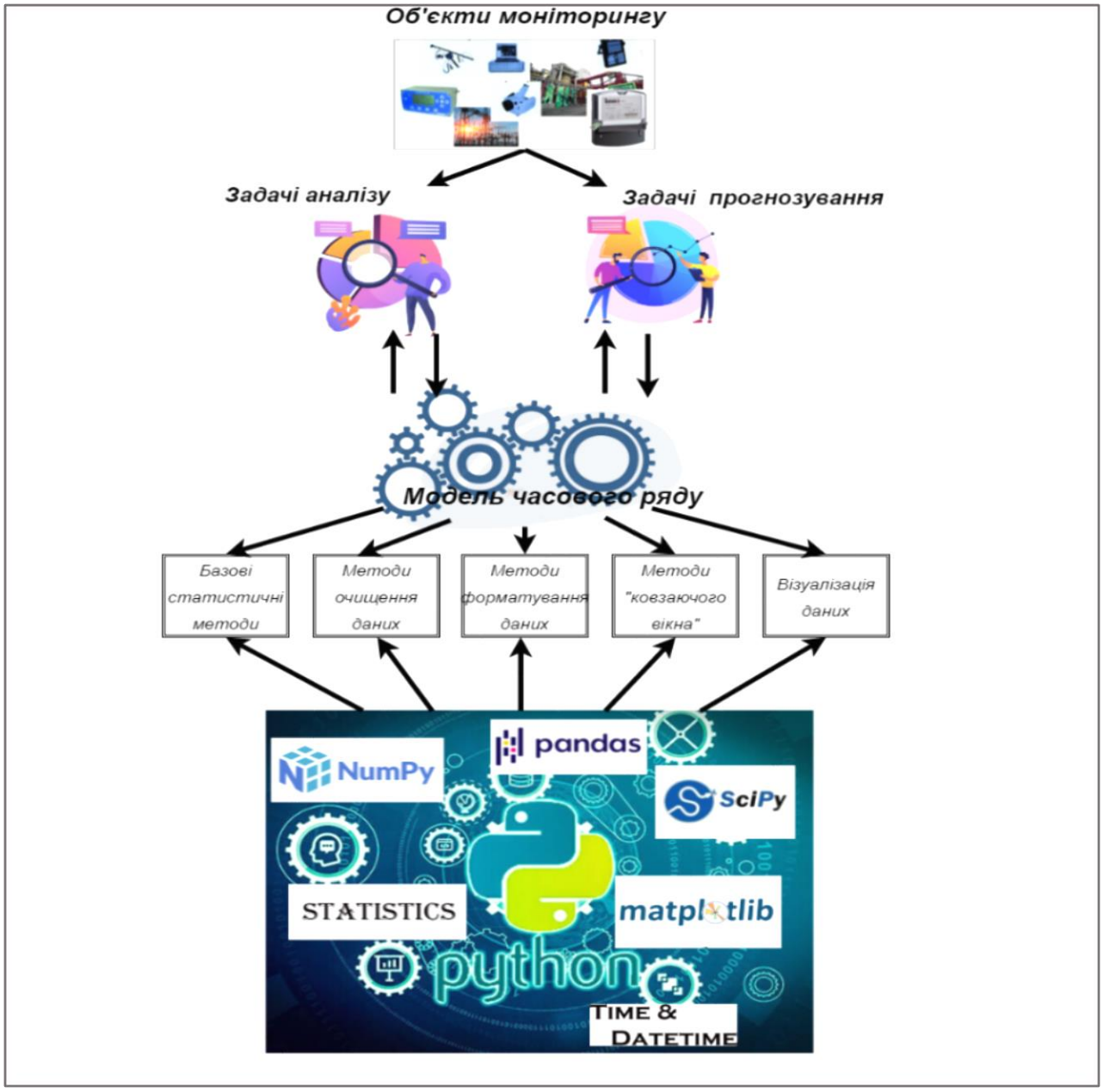


Рис. 1.4 – Узагальнюючий фреймворк дослідження та моделювання часових рядів

Окремою, допоміжною, але від того не менш значущою задачею є первинна обробка даних – очищення, структурування, перетворення необроблених даних у формат, необхідний для подальшої обробки програмними засобами. Працюючи з часовими рядами такі методи, як індексування на основі часу, повторна вибірка та ковзаючи вікна можуть допомогти моделювати, аналізувати та передбачувати значення рядів в часі. Крім того, впродовж всіх кроків дослідження виявляється корисним, а часто просто необхідним, забезпечити можливість візуального

відтворення інформації в різноманітних форматах. Іншими словами – як на початковому (часто говорять «розвідувальному) етапі роботи, так і під час подальшого поглибленого дослідження, дослідник має вільно володіти та використовувати відповідні засоби програмування.

Саме опису таких універсальних засобів, які використовуються як на початкових етапах аналізу даних часового ряду так і в подальшому складають основу інших, більш складних алгоритмів, присвячений основний об'єм даного навчального посібника. Набір програмних засобів та інструментів, які розглядаються в посібнику, відносяться до так званої «екосистеми *Python*», тобто спеціалізованих модулів (бібліотек), що мають програмний інтерфейс, за допомогою якого програміст отримує можливість безпосереднього звертання до них з програм написаних на мові програмування *Python*.

Нагадаємо, що *Python* – високорівнева інтерпретована мова програмування, на якій програмний код створюється відносно швидко та відносно легко. Програмуючі на *Python*, програміст може слідувати парадигмам процедурного або об'єктно-орієнтованого програмування. Наявність різноманітних бібліотек спрощує реалізацію складних процедур. Серед модулів розширення найбільш широко вживаних при дослідженні часових ряді слід зазначити наступні:

Time та *Datetime* – два бібліотеки, які зосередженні на представлені, форматуванні, обробці, перетворенні даних, що безпосередньо представляють собою сутності, що пов'язані з часом.

Statistics – бібліотека, що входить до базового комплекту *Python* та призначена для використання при програмній реалізації задач, пов'язаних з статистичною обробкою та дослідженням даних.

Numpy – бібліотека, яка використовується для наукових обчислень. Інструменти даної бібліотеки працюють з об'єктами типу «N -вимірний масив» і забезпечують основні математичні функціональні можливості, в також набір сервісних дій, таких як визначення розміру та розмірності масиву, визначення основних статистичних параметрів даних, мінімального та максимального значення елементів в масивах та його складових, тощо. Швидкість виконання арифметичних операцій роблять *numpy* незамінною складовою будь якого наукового, інженерного або технічного застосунку, а сам цей модуль використовуються в якості основи при побудові інших модулів екосистеми *Python*.

Pandas – ця бібліотека забезпечує високоефективні та водночас прості у використанні та представленні структури даних, такі як *Series* та *DataFrame*. Бібліотека розширює функціональність *Python* від простого збору та підготовки даних до аналізу даних. Водночас бібліотека забезпечує зручну та швидку роботу з даними, що представляють різні аспекти роботи з часом, що є основою аналізу часових рядів. Широке коло інструментів *pandas* можуть однаково добре застосовуються до аналізу будь-якого типу часових рядів, а гнучкі структури даних *pandas* можна застосовувати до даних часових рядів у будь-якій приклажній сфері, включаючи бізнес, науку, техніку, охорону здоров'я, кібербезпеку та багато інших.

Scipy – це бібліотека, яка використовується для наукових і технічних обчислень. Вона забезпечує функції оптимізації, обробки сигналів і зображень, інтеграції, інтерполяції та лінійної алгебри. Ця бібліотека використовується при реалізацію задач, пов'язаних з машинним навчанням. Бібліотека дуже потужна та розгалужена. В посібнику буде розглянуто лише ту її частину, яка безпосередньо використовується для реалізації задач первинного аналізу часових рядів.

Matplotlib – на сьогоднішній день ця бібліотека – безумовний лідер по використанню в якості засобу візуалізації даних у різних форматах, таких як графік, розкид, стовпчикова діаграма, гістограма тощо. Вона містить усі функціональні можливості, що стосуються графіків, необхідні від побудови різноманітних дизайнерських особливостей їх представлення.

Як вже зазначалося, в посібнику читачу буде надана початкова інформація щодо використання можливостей перерахованих бібліотек. Головною метою є навіть не перерахування методів, що входять до складу бібліотек та їх аргументів, а ознайомлення з базовими ідеями, які були закладені розробниками при їх створенні. Це дасть можливість подальшого самостійного вдосконалення читача в відповідності до шляху його ознайомлення з теоретичними методами дослідження та моделювання динамічних систем. В кінці посібника наведені посилання на інтернет-джерела, які містять офіційну документацію розглянутих бібліотек, а також на посилання на деякі літературні джерела з теми, що розглядається, і які мають допомогти читачеві в його подальшій навчальній та професійній діяльності.

Контрольні питання

1. Які явища та процеси реального світу описуються за допомогою їх моделей у вигляді часових рядів? Наведіть приклади.

2. Які основні задачі ставляться при дослідженні часових рядів?

3. В чому відмінність моментного та інтервального часових рядів?

4. Наведіть приклади проблем в галузі кібербезпеки, при вирішенні яких можуть використовуватися методи виявлення змін в поведінці моделей часового ряду.

2. АНАЛІЗ ЧАСОВИХ РЯДІВ В ЗАДАЧАХ МОНІТОРИНГУ ШИФРОВАНОГО МЕРЕЖЕВОГО ТРАФІКУ

Сьогодні інтернет стає невід'ємною складовою нашої цивілізації, основою її стрімкого розвитку та нових досягнень. Водночас, забезпечуючи значні системні, соціальні та інфраструктурні перетворення, поширення інтернет підсилює залежність всіх інституцій суспільства від безперебійної, коректної та безпечної роботи складових інфосфери, тобто призводить не лише до численних здобутків, а й породжує цілу низку проблем, зумовлених дедалі більшою вразливістю від стороннього, часто зловмисного, кібернетичного впливу. Комерційні, державні та некомерційні організації всіх видів сьогодні все частіше опиняються під кібератаками. Програми-вимагачі, шахрайство, крадіжка облікових даних, крадіжка ідентифікаційної інформації та інтелектуальної власності відбуваються щодня по всьому світу. Типова кібератака – це спроба зловмисників отримати доступ, змінити чи пошкодити комп'ютерну систему чи мережу несанкціонованим способом. Це систематичне, передбачуване та розраховане використання технологій для впливу на комп'ютерні мережі та системи з метою порушення роботи організацій та операцій, що залежать від них.

Одною з основних технологій забезпечення мережевої кібербезпеки довгий час залишалася технологія глибокого аналізу пакетів (Deep Packet Inspection – DPI), по суті – перевірка та фільтрація пакетів, що передаються мережею за їх змістом. DPI-системи аналізують не тільки заголовки пакетів, але і корисне навантаження кожного пакета. Пакети аналізуються на рівнях мережевої архітектури моделі OSI з другого і вище, в них намагаються знайти наперед задані та певним чином описані ознаки зловмисної або несанкціонованої мережевої активності. Системи захисту на основі DPI здатні виявляти і блокувати віруси, фільтрувати інформацію відповідно до заданих критеріїв, ідентифікувати та перевірити підозрілий і шкідливий трафік. У сфері мережевої безпеки методи перевірки мережі на основі сигнатур (тобто наперед заданих правил аналізу вмісту пакетів мережевого трафіку) можуть бути дуже ефективними. Але – тільки за умови, що подібні атаки відомі розробникам програмного забезпечення безпеки заздалегідь.

Раніше такий підхід працював достатньо ефективно. Правила безпеки, які припускали постійне доповнення та зміни, давали можливість відносно надійно захиститися від зловмисних дій. Але за останнє десятиріччя з'явилися нові

чинники, які значно впливають на застосування методів виявлення зловмисного втручання. Серед таких чинників виділимо як найбільш суттєві:

- швидке зростання об’єму трафіку і пропускних здібностей каналів зв’язку що призводить до необхідності розробки та використання алгоритмів зі зниженою обчислювальною складністю;

- використання різноманітних середовищ передачі – включаючи мобільні мережі, мережі об’єктів IoT, локальні, магістральні та навіть космічні канали зв’язку тощо;

- значне збільшення частки шифрованого трафіку, що призводить до неможливості використання засобів на основі аналізу вмісту.

Всі вказані чинники безумовно взаємопов’язані між собою та створюють нові виклики для розробників систем кіберзахисту. Але звернемо особливу увагу на останній з перерахованих чинників – розповсюдження шифрованого трафіку, та розглянемо проблеми, які при цьому з’являються.

Хоча перші спроби шифрування інтернет-трафіку з’явилися разом з самою інтернет мережею, до 2014 року більша частка трафіку передавалося у відкритому, нешифрованому вигляді. Це й давало можливість використовувати DPI з аналізом даних на різних рівнях мережевих протоколів, використовуючи правила безпеки, та приймати відповідні рішення. Крім того, відносно помірний об’єм трафіку мереж давав можливість проводити DPI (яка є досить ресурсовитратною технологією) практично в режимі реального часу.

Наразі потреба в шифруванні даних викликається цілим рядом причин, зокрема:

- розумінням того, що незашифровані з’єднання можуть прослуховуватися та аналізуватися не тільки службами захисту, але й недоброчинними хакерами, а отже, що останні можуть вкрасти конфіденційну інформацію;

- домінуючим поширенням WiFi-мереж, тобто появою у кіберзлочинців можливості відносно нескладного несанкціонованого підключення до таких мережі, а отже можливість моніторингу всієї мережевої активності користувача;

- тим, що маючи доступ до незашифрованого трафіку інтернет-провайдер може переглядати пакети даних користувачів, фіксувати сайти, які переглядаються та отримувати інформацію стосовно веб-програм, які використовуються. Завдяки цьому провайдери отримують інформацію для цілеспрямованого обмеження пропускної здатності з’єднання конкретного користувача. А також отримують можливість продавати дані щодо уподобань

користувачів рекламодавцям. За певних умов ця інформація може стати доступною певним державним органам, що в демократичних державах вважається суттєвим порушенням права конфіденційності особистої інформації;

-навіть в рамках локальної (корпоративної) мережі відсутність шифрування трафіку означає, що системні адміністратори також можуть відстежувати активність користувачів з метою запобігання як витоку інформації так і використанню певних заборонених корпоративними правилами безпеки джерел інформації: сайтів, месенджерів, соціальних мереж тощо.

Узагальнюючи можна сказати, *що шифрування – це засіб захисту від небажаного моніторингу онлайн-активності користувача та захист його особистої або корпоративної/комерційної інформації від несанкціонованого доступу до них стронніх осіб.* За даними Google, якщо в 2016 році обсяг зашифрованого трафіку становив 70%, то зараз цей показник сягнув 95%. Згідно прогнозів, до 2025 року практично весь мережевий трафік буде передаватися в зашифрованому вигляді.

З’явившись як засіб кіберзахисту та забезпечення конфіденційності користувача, шифрування трафіку досить швидко почало використовуватися і зловмисниками. Виявилося, що оскільки зашифрований трафік не може бути проаналізований системами захисту та виявлення вторгнень, а суб’єкти загроз продовжують розвивати свою тактику та методику (включаючи приховування атаки у зашифрованому трафіку), поширення шифрування трафіку водночас робить користувачів мереж знову вразливими для атак. Це концептуальне протиріччя ставить дуже складне завдання перед розробниками систем захисту: знайти баланс між наскрізною безпекою, збереженням конфіденційності кінцевих користувачів і водночас збором інформації з трафіку для виявлення можливих загроз, кращого розподілу та захисту ресурсів. Іншими словами – розробити такі способи виявлення шкідливої поведінки, які не знижують конфіденційність користувачів.

Позатим, незважаючи на те, що самі дані які передаються при їх шифруванні не піддаються аналізу за технологією DPI, інформація про заголовки пакетів, IP-адреси та порти джерела та отримувача інформації, статистичні властивості потоків трафіку, такі як інтервали затримки між моментами надходження пакетів, розмір пакетів, кількість пакетів за протоколами тощо, залишаються доступними, а отже можуть слугувати джерелом інформації для подальшого дослідження. Виходячи з того, що мережевий трафік є складним

динамічним процесом і водночас суперпозицією багатьох потоків з множинними взаємопов'язаними характеристиками, які генеруються різними протоколами, програмними застосунками, або навіть різними операційними системами, що встановлені на кінцевих пристроях, логічно припустити, що будь яка мережева атака повинна тим чи іншим чином змінити деякі ознаки мережевого трафіку. Отже, *наша прикладна задача зводиться до задачі вміння аналітично, за допомогою відповідних математичних та статистичних методів зафіксувати такі заміни і, зрештою, автоматизувати цей процес з використанням відповідних програмних інструментів*.

В загальному випадку процедура аналізу шифрованого трафіку може виконуватися не тільки з метою детектування шкідливого трафіку, але і для вирішення інших задач, пов'язаних з забезпеченням функціювання мереж, таких як підвищення якості обслуговування (різні типи даних, що передаються – відеотрафік, аудіотрафік, веб-трафік – мають різні вимоги до швидкості передачі інформації, затримки при передачі пакетів, втрат пакетів тощо), або блокуванням трафіку ресурсів, заборонених місцевим законодавством або корпоративними правилами (як варіант таких, що використовують P2P, VPN-з'єднання або TOR).

Однак між цими задачами існує суттєва різниця. При вирішенні задач, пов'язаних з продуктивністю роботи мережі мета полягає в пошуку відповіді на питання чи присутні вказані ознаки в мережевому трафіку, і прийняття рішення на підставі математичних та/або статистичних процедур визначення «впевненості» в такому висновку. На відміну від цього при детектуванні шкідливого мережевого трафіку попередня інформація щодо поведінки значень ознак як правило відсутня. Тим більше це характерно в випадку появи новітніх атак, т.з. «атак нульового для». Єдине, що може зробити дослідник за цих умов – виявити момент зміни ознак трафіку, які дають можливість з певним рівнем впевненості говорити, що в загальному мережевому трафіку з'явилася складова, яка нехарактерна для звичайної поведінки трафіку. Іншими словами – задача ставиться не як задача точного діагностування наявності факту, а тим більше – типу, мережевої атаки, а *як задача виявлення факту появи аномалій* в ознаках трафіку. Такі аномалії можуть виявитися як відображенням дійсної кіберзагрози, так і інформувати про інші зміни в функціюванні програмного або апаратного забезпечення мережі (наприклад – збою або виходу з ладу певних пристроїв). В рамках технології машинного навчання така задача називається задачею

виявлення аномалій (anomaly detection) або задачею *виявлення наявності точки зміни* (Change Point Detection - CPD) в поведінці об'єкту спостереження.

Для виконання процедури CPD необхідно обрати ознаки, за якими буде проводитися процедура визначення наявності такої зміни та алгоритм, який при цьому буде використовуватися. Різні дослідники виходячи з своїх цілей пропонують різні набори ознак, що характеризують мережевий трафік як об'єкт спостереження. Серед них – окрім вже зазначених вище первинних ознак зустрічаються і похідні ознаки, так як частота появи вхідних та вихідних пакетів, відношення вказаних величин, співвідношення розмірів вхідних та вихідних пакетів, тривалість сесій. Оскільки і сама задача ставиться в часі (зокрема – як визначення моменту виникнення змін в трафіку), і дані що аналізуються, характеризуються не окремим значенням, а прив'язкою даних до певного моменту часу, стає зрозумілим, що *часовий ряд як аналітична і математична модель найбільш точно пасує до використання в задачах аналізу поведінки мережі на основі даних, що описують мережевий трафік*.

Алгоритми CPD (Рис. 2.1) ґрунтуються на концепції *ковзаючого вікна*, яка полягає в порівнянні двох наборів значень параметра, який обраний для моніторингу, на двох інтервалах часу. При цьому інтервал часу, що починається раніше, разом з відповідними значеннями параметру, називають «базовим вікном», а інше вікно – «поточним вікном». В залежності від мети дослідження та методу аналізу розміри базового та поточного вікон (тобто кількість значень, які охоплює відповідний часовий ряд) можуть бути як однаковими так і різними. Моменти початку збору даних для цих наборів мають різнитися не менш ніж на одиницю, але можуть різнитися і на будь як великий проміжок часу. Сам алгоритм CPD є нескінченим циклом моніторингу (тобто після початку роботи закінчитися він може лише примусово). На першому кроці отримуються дані з базового вікна та вираховуються їх характеристики. На другому кроці отримуються дані поточного вікна і їх характеристики. На третьому кроці виконується порівняння характеристик базового та поточного вікон. Якщо відмінностей між вказаними характеристиками не виявлено, береться наступне поточне вікно та процедура продовжується з другого кроку. В разі, якщо між характеристиками виявлені значимі зміни (що є наслідком зміни в поведінці об'єкту моніторингу), алгоритм генерує відповідний сигнал для сповіщення особи, яка відповідальна за прийняття кінцевого рішення та виконання відповідних дій. Після цього поточне вікно приймається в якості нового базового

вікна (тобто подальші зміни в об’єкті будуть визначатися відносно цього нового базового вікна) і алгоритм продовжує роботу починаючи з першого кроку.

В якості характеристик, які отримуються з «сирих» даних базового та поточного вікон в найпростіших випадках використовуються точкові оцінки основних статистичних параметрів набору даних, зокрема середнього, дисперсії, медіани, рідше моментів третього і четвертого порядку, коефіцієнтів

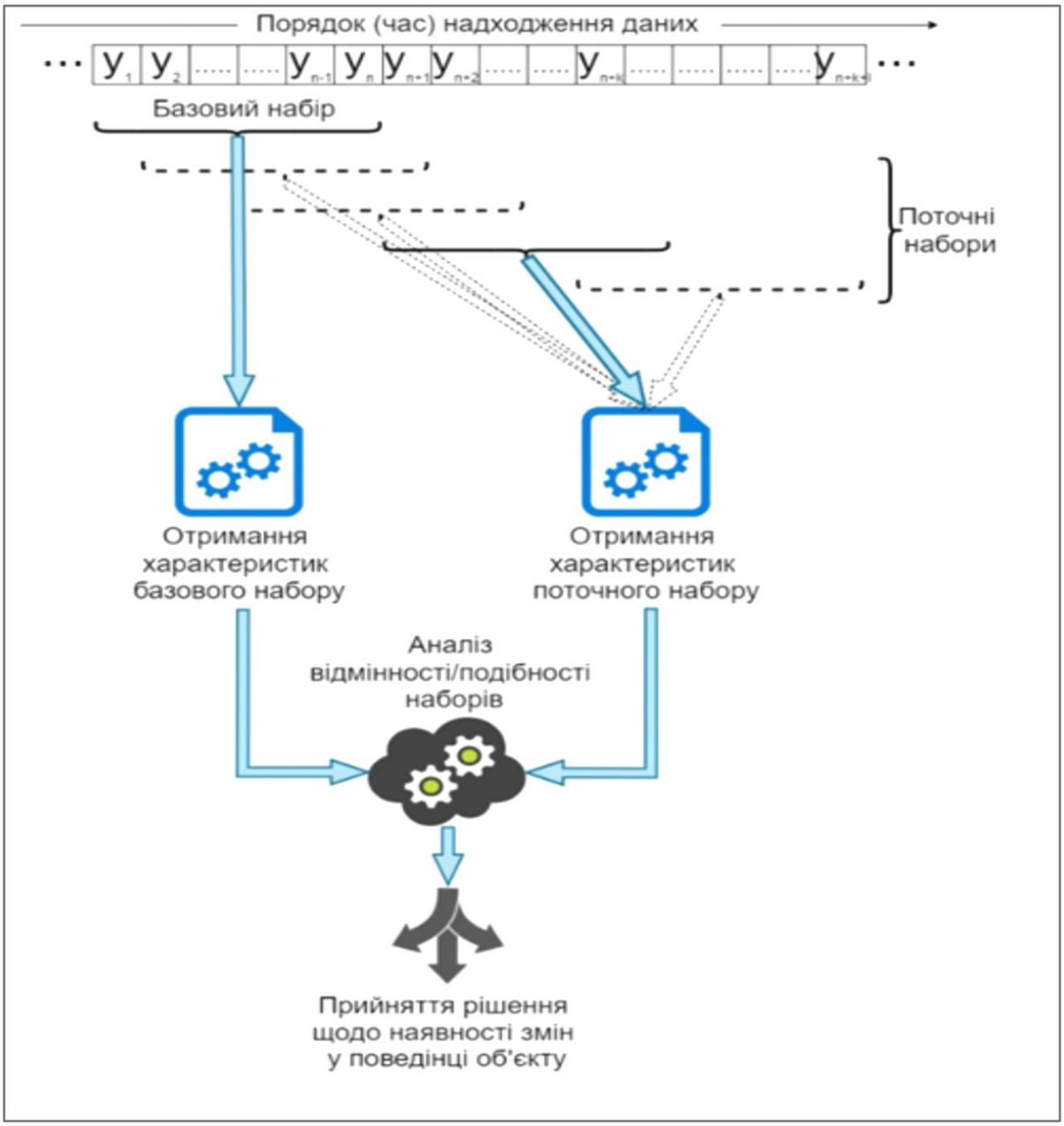


Рис. 2.1 – Алгоритм CPD

автокореляції. Популярність такого підходу зумовлена, зокрема, і тим, що обчислені точкові оцінки легко і головне – цілком зрозуміло для користувача можна відобразити на інформаційних панелях для візуальної перевірки. Крім того, для більшості з зазначених оцінок існують методики визначення їх p_value – статистичного параметру, що визначає на скільки отримані оцінки несумісні з гіпотезою незмінності поведінки об'єкту. Або, що рівнозначно, яка ймовірність помилитися, згенерувавши сигнал зміни в випадку, якщо такої зміни не відбулося. Вказана помилка відома як помилка I роду ("помилкове спрацювання"). Найбільш вживані статистичні тести, які використовуються для визначення відмінності даних двох вікон спостереження – t-тест Стьюдента, його непараметричний аналог –тест Вилкоксона-Манна-Уитні, які використовуються для визначення незмінності середнього значення наборів даних та/або їх медіани. Тест на рівність/відмінностей дисперсії у двох наборах можна виконати за допомогою критерію F-критерію Фішера або більш стійкого критерію Левена. Для виявлення тренду (зростаючого або спадаючого) в даних використовується критерій Аббе-Линника, а для виявлення тренду в дисперсіях – критерій Фостера-Стюарта.

Більш складні і більш точні – але водночас і більш ресурсовитратні, а отже і більш повільні – алгоритми використовують інші ознаки, отримані з даних базового та поточного вікон. Зокрема це може бути різноманітні алгоритми порівняння двох відповідних гістограм, алгоритми, які використовують поняття ентропії даних, алгоритми що застосовують фрактальний аналіз даних, а також, алгоритми що використовують аналітичні моделі часового ряду різної складності, від найпростіших регресійних, до моделей типу SARIMA, а потім певним чином порівнюють або самі отримані моделі, або аналізують відхилення даних поточного вікна від моделі побудованої на даних базового вікна.

Детальний опис конкретних алгоритмів CPD, порівняння їх між собою, виявлення сильних та слабких сторін, швидкості виявлення зміни відносно моменту її виникнення, та безпосередньо алгоритмічної ефективності виходять за рамки даного посібника. В наступних розділах будуть розглянуті ті інструментальні засоби різноманітних модулів екосистеми *Python*, які в поєднанні дадуть можливість читачу в подальшому перейти до вивчення відповідних реалізацій алгоритмів CPD та інших задач, що виникають при аналізі часових рядів в різноманітних прикладних наукових, технічних, інженерних, соціальних, медичних, фінансових тощо задачах.

Контрольні питання

1. Чому технологія глибокого аналізу пакетів перестає задовольняти потреби спеціалістів в галузі захисту інформації?

2. Чому шифрування мережевого трафіку стає загальновживаною практикою?

3. Які параметри мережевого трафіку та його елементів лишаються доступними для формалізованого аналізу при використанні шифрування трафіку?

4. Як задача виявлення аномалій в часових рядах може використовуватися при аналізі шифрованого мережевого трафіку?

5. Опишіть алгоритм ковзаючого вікні та можливість його застосування при виявленні аномалій в часових рядах.

3. ПРОГРАМНІ ЗАСОБИ *PYTHON* ТА МОДУЛІВ РОЗШИРЕННЯ ДЛЯ РОБОТИ З ЧАСОМ ТА ДАТАМИ

Часовий ряд є формою представлення даних, в якій основне семантичне навантаження покладено на інформацію про момент або інтервал, в який надійшли кожне з знечень вибірки, що аналізується. Таким чином, при аналізі часових рядів дослідник має оперувати програмними інструментами, які дозволяють просто і зручно опрацьовувати інформацію, пов'язану з часом або датою. Як в базовому *Python*, так і в пакетах, що побудовані на його основі, зокрема в *numpy* та *pandas*, існують декілька таких інструментів для організації роботи з часом та датами.

3.1. Модуль time

Модуль *time* зі стандартної бібліотеки *Python* містить декілька корисних методів для роботи з часом. З його допомогою можна отримувати інформацію про поточну дату і час з точністю до мілісекунди, виводити ці відомості в необхідному форматі, а також керувати ходом виконання програми, додаючи затримки по таймеру. Найпростіший виклик функції, що дозволяє визначити поточний час, та результат, що при цьому отримується, виглядає наступним чином:

```
import time
time.time()
>>>>
Out[1]: 1668789408.0911138
```

В даному разі час представлений в вигляді т.з. «системного часу». Цей час відповідає кількості секунд, що пройшли від моменту початку «епохи» і до поточного моменту виконання скрипту. Відправною точкою («епохою») для системного часу як правило вважається 1 січня 1970 року, 00 годин, 00 хвилин, 00 секунд, і для цього моменту лічильник секунд має нульове значення. Системний час представлений у *float*-форматі. Отримавши значення поточного часу в вигляді системного часу у разі необхідності секунди переводяться в години, дні, місяці і роки, надаючи користувачу можливість сформувати зручний для нього спосіб відображення часу.

Слід зазначити, що деякі системи теоретично можуть мати значення «початку епохи» відмінне від вказаного. Про всяк випадок не завадить хоча-б раз

впевнитися, яку саме дату ваш комп'ютер використовує в якості моменту «початку епохи». Це можна зробити за допомогою команди:

```
print(time.gmtime(0))
>>>>
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0,
tm_sec=0, tm_wday=3, tm_yday=1, tm_isdst=0)
```

Як бачимо, результат збігається з очікуваним.

Функція *time.ctime()* конвертує системний час представлений у секундах у дані типу «строка», що надає даним вигляду, більш прийнятному для сприйняття людиною.

```
print(time.ctime(1668789408.0911138))
>>>>
Fri Nov 18 18:36:48 2022
```

Дещо подібна до *time.ctime()* функція *time.localtime()* повертає не строковий об'єкт, а об'єкт спеціального типу – т.з. struct_time-об'єкт, робота з яким підтримується функціями з пакету time. Це дає можливість обробляти отриману інформацію методами, призначеними для виконання спеціальних операцій по обробці значень часу та дат.

```
loc_time=time.localtime()
print(loc_time)
>>>>
time.struct_time(tm_year=2022, tm_mon=11, tm_mday=18, tm_hour=22, tm_min=12,
tm_sec=5, tm_wday=4, tm_yday=322, tm_isdst=0)
```

В цьому об'єкті представлені (по своїм складовим) дата та час, а також день тижня, номер дня у році і вказівка на те, чи діє (tm_isdst=1) діє літній час, або що він не діє (tm_isdst= 0).

В наступному прикладі виділимо з усієї дати (точніше, з об'єкту, який репрезентує дату) тільки рік. Зауважте, що в даному прикладі дата визначається не функцією *time.localtime()*, як це робилося вище, а функцією *time.time()*, а сама функція *time.localtime()* лише перетворює *float*-представлення системного часу у struct_time-об'єкт.

```
s=time.time()
loc_time=time.localtime(s)
```

При необхідності можна отримати значення тільки одного атрибуту struct_time-об'єкту, наприклад року:

```
loc_time.tm_year
```

```
>>>>
2022
```

або отримати той самий результат, вказавши

```
loc_time[0]
>>>>
2022
```

Отримане значення (тобто – поточний рік на момент виконання скрипту) буде представлений в форматі *int*, а отже може застосовуватися в будь яких арифметичних або логічних операторах.

Значення описаних функцій також полягає і в тому, що при необхідності зберігання інформації (наприклад – в базі даних) простіше і економніше зберегти дату у вигляді числа, ніж попередньо перетворювати її на об'єкти *datetime* (який буде розглянутий далі). Звичайно, це також має свій недолік: якщо виявиться необхідним піклуватися зручністю інтерфейсу користувача, потрібно буде число що представляє дату, попередньо піддати обробці – звичайне протиріччя, яке виникає між ефективністю зберігання та ефективністю обробки даних в програмуванні.

За допомогою функції *time.strftime()* можна отримати рік, місяць, день, годину, хвилини, секунди окремо та повернути їх у вигляді строкової змінної. Далі ця змінна може бути, наприклад, виведена в консоль за допомогою функції *print()* або оброблена будь яким іншим способом, що використовується при роботі зі строковими змінними. В якості аргументу функція може приймати будь-яку змінну, що містить значення, які повертаються іншими функціями модуля *time*.

```
loc_time = time.localtime()
time_string = time.strftime("%m/%d/%Y, %H:%M:%S", loc_time)
print(time_string)
>>>>
11/18/2022, 21:29:30
```

Враховуючи, що в реальному житті використовується дуже багато різноманітних форматів представлення часу та дат, перший параметр вказаної функції дозволяє описати той з них, який треба використати в конкретній ситуації. Кодування сутностей, що складають час та дату, задаються наступними літералами:

- %Y – рік;

- %m – місяць;
- %d – день;
- %H – година;
- %M – хвилина;
- %S – секунда.

Насправді кількість допустимих літералів значно більша, нараховує декілька десятків елементів. Детально ознайомитися з ними можна в документації за посиланням: https://docs.python.org/3/library/time.html. Вказані засоби дають можливість в разі потреби зробити так, щоб результат включав, наприклад, тільки значення місяця та дня. Для цього достатньо просто записати в наведеній вище функції в якості значення першого аргументу літерал "%m/%d". Або, в разі потреби, навпаки: "%d/%m".

Якщо виявиться необхідним працювати з українськими назвами місяців та днів тижня, потрібно попередньо встановити відповідну локалізацію, для чого звернутися до функцій іншого допоміжного стандартного модулю *Python*:

```
import locale
locale.setlocale(locale.LC_ALL, "uk_UA")
print(time.strftime("%d-%B %Y. %H:%M:%S" , loc_time))
>>>>
18-листопад 2022. 22:17:30
```

Встановлення англійської або російської локалізації виконується наступними операторами:

```
locale.setlocale(locale.LC_ALL, "en")
locale.setlocale(locale.LC_ALL, "ru")
```

Функцією, яка в певному розумінні є зворотною до *strftime()*, є функція *time.strptime()*. Вона приймає два аргументи: строку що буде перетворена на дату і час, та опис формату, а повертає значення дати у вигляді *struct_time*-об’єкту.

```
st_time1=time.strptime("18-листопад 2022." ,"%d-%B %Y.")
st_time2=time.strptime("22:17:30" ,"%H:%M:%S")
print(st_time1)
print(st_time2)
>>>> time.struct_time(tm_year=2022, tm_mon=11, tm_mday=18, tm_hour=0, tm_min=0,
tm_sec=0, tm_wday=4, tm_yday=322, tm_isdst=-1)
time.struct_time(tm_year=1900, tm_mon=1, tm_mday=1, tm_hour=22, tm_min=17,
tm_sec=30, tm_wday=0, tm_yday=1, tm_isdst=-1)
```

Зверніть увагу, як саме заповнюються ті елементи об’єкту, що створюється в разі відсутності в вхідній строчці відповідної інформації.

Функція *time.sleep()* дає розробнику можливість призупинити виконання його скрипту на певну кількість секунд (часто кажуть – «зробити паузу»). Необхідність такої затримки може викликатися різними причинами. Як приклад – призупинити скрипт на час, що потрібен для виконання закриття файлу (по суті – запису файлу на диск, що може потребувати досить значного часу). В прикладі нижче показано, як саме можна зробити паузу (в даному разу на 10 секунд) в виконанні скрипта, а що саме буде виконуватися (та чи буде взагалі) під час утвореної паузи – визначається алгоритмом, який реалізується.

```
pause = 10
print("Почали... і не хвилюйтеся, ми встановили затримку в 10 секунд.")
time.sleep(pause)
print(str(pause) + " секунд пройшло.Продовжуємо роботу.")
```

3.2. Модуль datetime

На відміну від модуля *time*, що призначений в першу чергу для визначення поточного часу, існує також модуль *datetime*, який призначений для виконання певних маніпуляцій, логічних і арифметичні операції, порівнянь, обробки та ін. з даними, що безпосередньо зберігають значення часу та дати. Модуль datetime в Python має 5 основних підмодулів, кожен з яких реалізує відповідний клас об’єктів:

- *date* – для роботи з об’єктами-датами;
- *time* – для маніпулювання об’єктами які представляють час;
- *datetime* – це комбінація дати та часу;
- *timedelta* – дозволяє працювати з поняттям «тривалість часу», або «інтервал»;
- *tzinfo* – дозволяє працювати поняттям «часовий пояс».

З переліку видно – але ми звернемо на це спеціальну увагу – що дані для представлення інформації щодо дати та часу можуть мати різну семантику. Одні з них вказують певну дату та/або час, причому таке представлення може використовуватися у різних форматах, наприклад *«July 4»* , *«March 8, 2022 22:00», «31 December 2022, 23:59:59», «07/23/2022», «23/07/2023»* тощо. Такі дані ще називають «часовою міткою» або «міткою часу». Інші дані представляють величину проміжку між деякими моментами часу, наприклад *«1 year 3 month 5 days», «24 hours, 17 minutes, 56 seconds, 268 miliseconds», «3 weaks, 3 days, 3 hours, 3 minutes»* тощо. Про такі дані говорять як

про «інтервали». При обробці інформації можна, наприклад, до певної дати (мітки часу) додати «інтервал» і отримати іншу дату («мітку часу»). Можна додати один інтервал до іншого і отримати новий (більший за протяжністю) інтервал. А от скласти дві мітки часу семантично сенсу не має. Отже, в одному виразі можуть бути присутні дані обох типів, а реалізація модулю *datetime* як раз і забезпечує їх коректну обробку.

Імпортуємо модуль *datetime* і створимо об'єкт дати (екземпляр об'єкту класу *date*):

```
import datetime as dt
st_date=dt.date(year=2022, month=11, day=10)
print(st_date)
>>>>
2022-11-10
```

або скорочено, притримуючись порядку опису параметрів:

```
st_date=dt.date(2022,11,10)
```

Зазначимо, що отримане значення представлене в стандартному для модулю datetime форматі, а також, що це незмінюваний об'єкт (термінах опису об'єктів в *Python*). Спроба ввести некоректну дату, наприклад *dt.date(2022, 26, 3)* призведе до ситуації виключення типу *ValueError*.

Створемо об'єкт класу time (Зауваження – не слід путати об'єкти класу time модулю datetime і об'єкти, з якими працюють за допомогою функцій модулю time).

```
st_time= dt.time(hour=12, minute=45, second=16, microsecond=257)
print(st_time)
>>>>
12:45:16.000257
```

Цікаво зазирнути в менеджер змінних IDE і подивитися, на скільки сильно різняться представлення створених вище об'єктів *st_date* та *st_time*.

В разі, якщо виникає потреба представити мітку часу з точністю до дати, години, хвилини, секунди та мілісекунди, використовують об'єкти окремого класу *datetime* модулю *datetime*, який по суті є класом-наступником від класів *date* та *time*.

```
st_moment=dt.datetime(year=2022, month=11, day=10, hour=13,second=55)
print(st_moment)
>>>>
```

```
2022-11-10 13:00:55
```

Якщо при створенні об’єкту класу *datetime* ми задамо не всі параметри, то об’єкт все рівно буде створений.

```
st_moment1=dt.datetime(year=2022, month=11, day=10)
print(st_moment1)
>>>>
2022-11-10 00:00:00
```

Зрозуміло, що виключати параметри з опису можна тільки «послідовно», починаючи з останнього. Тобто – не можна пропустити тільки параметр *year*, або, наприклад, параметр *month*. Такі спроби призведуть до виникнення виключень типу «*TypeError*».

Окрема функція модулю *datetime*, а саме *datetime.combine()* створює об’єкт – «комбінацію» з часток, що відповідають даті та часу:

```
st_moment2=dt.datetime.combine(st_date, st_time)
>>>>
2022-11-10 12:45:16.000257
```

Звісно, можна і навпаки:

```
st_date2=st_moment2.date()          #відокремлюємо дату.
st_time2=st_moment2.time()          #відокремлюємо час.
print(st_date2,'\n',st_time2)
>>>>
2022-11-10
12:45:16.000257
```

Коректність подальшої роботи з утвореними змінними забезпечується тим, що об’єкт *st_date2* створюється як об’єкт класу *date*, а об’єкт *st_time2* – як об’єкт класу *time*.

Група методів дозволяє виділяти окремі компоненти мітки часу:

```
print('tuple:', st_moment1.timetuple())
print('Рік:', st_moment1.year)
print('Місяць :', st_moment1.month)
print('День :', st_moment1.day)
>>>>
tuple: time.struct_time(tm_year=2022, tm_mon=11, tm_mday=10, tm_hour=0, tm_min=0,
tm_sec=0, tm_wday=3, tm_yday=314, tm_isdst=-1)
Рік: 2022
Місяць : 11
День : 10
```

Отримати описові характеристики даних, що є часовими мітками, можна за допомогою методів:

```
print('Номер дня в році:', st_moment1.())
print('Номер дня в тижні:',st_moment1.weekday())
print('Номер дня в тижні:',st_date.weekday())
>>>>
Номер дня в році: 738469
Номер дня в тижні: 3
Номер дня в тижні: 3
```

(При визначенні номеру для в тижні приймається, що понеділку відповідає значення 0, вівторку – 1, тощо).

В разі, якщо потрібно отримати значення поточного часу, в *datetime* використовується функції *now()* або *today()*.

```
now = dt.datetime.now()
print(now)
>>>>
2022-11-19 18:47:19.518040
```

Відмінність між *datetime.now()* та *datetime.today()* полягає в тому, що перша з них в разі потреби дає можливість працювати не тільки з локальним часом, але використовувати так званий «час за Грінвічем».

Відомий нам метод *strftime()* має реалізацію і для об'єктів модулю *datetime*:

```
tod = dt.datetime.today()
tod.strftime("%d.%m.%Y")
>>>>
'19.11.2022'
tod.strftime("%H:%M:%S")
>>>>
'19:42:45'
```

Для зміни певних компонент опису дат та часу, представлених в форматі *datetime*, використовується метод *datetime.replace()*. Розглянемо приклад:

```
time_mk = dt.datetime(2015, 3, 21)
print(time_mk)
time_mk2 = time_mk.replace(year = 2022)
print(time_mk2)
time_mk2=time_mk2.replace(month=time_mk2.month+5)
print(time_mk2)
time_mk2=time_mk2.replace(hour=3,second=54)
print(time_mk2)
>>>>
```

```
2015-03-21 00:00:00
2022-03-21 00:00:00
2022-08-21 00:00:00
2022-08-21 03:00:54
```

В прикладі показано, як можна вирахувати нове значення параметру на основі старого його значення (*time_mk2.month+5*), однак такий спосіб досить кропіткий і потребує уваги програміста. Оскільки діапазони атрибутів визначаються відповідними природніми кордонами (наприклад – значення параметра *month* має лежати в діапазоні від 1 до 12, *hour* – від 0 до 23, *minute* – від 0 до 59 тощо), відповідальність за коректність змін покладається на програміста.

Метод *timestamp()* дозволяє перейти від об'єкту *datetime* до його представлення у вигляді секунд з початку епохи.

```
st_moment1=dt.datetime(year=2022, month=11, day=10, hour=13,second=55)
print(st_moment1)
sys_time=st_moment1.timestamp()
print(sys_time)
>>>>
2022-11-10 13:00:55
1668078055.0
```

Зворотня дія виконується за допомогою функції *fromtimestamp()*:

```
st_moment2=dt.datetime.fromtimestamp(sys_time)
print(st_moment2)
>>>>
2022-11-10 13:00:55
```

Одна з найбільш поширених задач при роботі з часом та даними – знаходження інтервалу між подіями, які відбувалися в певні моменти часу. Користуючись класом *datetime*, можна легко знаходити різницю між двома різними датами. Наступний приклад демонструє створення двох об'єктів. Виконуючи просту операцію знаходження різниці (віднімання) між об'єктами time_mk1 та time_mk2, створюється новий об'єкт time_int.

```
time_mk1 = dt.datetime.now()
time_mk2 = dt.datetime(2015, 3, 21)
time_int = time_mk1 – time_mk2
print(time_int)
>>>>
2800 days, 19:49:13.420862
```

Створений об’єкт *time_int* є об’єктом класу *timedelta*, що реалізований засобами модулю *datetime*. Екземпляр об’єкту *datetime.timedelta* зберігає значення інтервалу як поєднання *days, seconds* і *microseconds*, а решта переданих параметрів конструктор конвертує в ці одиниці, а саме:

- *millisecond* перетворюється на 1000 *microseconds*.
- *minute* перетворюється на 60 *seconds*.
- *hour* перетворюється на 3600 *seconds*.
- *week* перетворюється на 7 *days*.

Пояснимо це на прикладі. Явним чином створимо об’єкт класу *datetime.timedelta*:

```
delta = dt.timedelta(
 days=50,
 seconds=27,
 microseconds=10,
 milliseconds=29000,
 minutes=5,
 hours=8,
 weeks=2
 )
print(delta)
>>>>
64 days, 8:05:56.000010
```

При цьому слід по-перше, звернути увагу, що значення кількості днів в інтервалі були перераховані (50 днів+ 2 тижні=50+2*7=64 дні). По-друге, незважаючи на те, що функція *print()* для таких об’єктів робить відповідний перерахунок, в дійсності об’єкт зберігає лише три атрибути:

```
datetime.timedelta(days=64, seconds=29156, microseconds=10)
```

Отримати значення цих атрибутів можна звичайним чином:

```
days_int= delta.days
seconds_int= delta.seconds
microseconds_int= delta.microseconds
print(days_int)
print(seconds_int)
print(microseconds_int)
>>>>
64
29156
10
```

Отримані таким чином компоненти опису інтервалу часу представлені в вигляді змінних типу *int* і можуть використовуватися за звичайними правилами роботи з даними цього типу в мові програмування *Python*.

Також можна об'єкти-інтервали додавати один до одного, додавати об'єкти-інтервали до об'єктів часових міток (або віднімати від других перші), формуючи тим самим нові об'єкти. У наступному прикладі показано, як отримати нову дату маючи початкову дату та значення інтервалу до другої дати.

```
date1 = dt.datetime(2016, 12, 5)
delta1= dt.timedelta(weeks=8, days= 8, hours=2, minutes=5, seconds=17)
print(date1)
date2 = date1 + delta1
print(date2)
>>>>
2016-12-05 00:00:00
2017-02-07 02:05:17
```

Зверніть увагу, як коректно перераховані роки та місяці. Навіть якщо дані задані так, що прийдеться зважати на високосність року, відмінність кількості днів в місяці тощо, відповідальність за правильність перерахунків модуль *datetime* бере на себе.

Окрім вказаних, над об'єктами класу *timedelta* можуть виконуватися операції множення на ціле число (що збільшує інтервал в відповідну кількість разів) та навіть на дійсне число (що призводить до округлення результату), ділення інтервалу на інтервал, одномісна операція «мінус», тобто зміна знаків значень компонентів опису дати на протилежний), логічні операції порівняння тощо. До речі, якщо компоненти значень об'єкту від'ємні, це означає, що при додаванні такого інтервалу до певної дати, результуюча дата виявиться менша за першу, тобто – буде звернута «у минуле».

Нарешті метод *total_seconds()* дозволяє отримати значення інтервалу в секундах:

```
delta1= dt.timedelta(weeks=8, days= 8, hours=2, minutes=5, seconds=17)
delta_sec=delta1.total_seconds()
print(delta_sec)
>>>>
5537117.0
```

3.3. Робота з часом і датами в бібліотеці numpy

Основний недолік описаних вище об'єктів та засобів роботи з ними полягає в їх відносній повільності, яка стає особливо помітна при роботі з великими об'ємами даних. Тому в *numpy* використовується свій формат роботи з даними –

datetime64, який кодує дати як 64-бітові цілі числа і дозволяє скористатися всіма перевагами цього пакету.

На відміну від об’єктів, що використовуються модулями *time* та *datetime*, які підтримують лише роки в діапазоні від 1 до 9999 року нашої ери, *datetime64* допускає дати до нашої ери; роки до нашої ери дотримуються астрономічної угоди про нумерацію років, тобто рік 2 до нашої ери має номер -1, рік 1 до н. номер 0 тощо.

Створити дані в форматі *datetime64* можна по-різному:

```
n_dt1=np.datetime64(‘2023-02-25’)
print(n_dt1)
>>>>
2023-02-25

n_dt2=np.datetime64(‘2023-02’)
print(n_dt2)
>>>>
2023-02
```

Однак, в разі необхідності можливо задавши тільки рік та місяць, використовувати надалі в означенні дати і дні місяця. Про цьому використовують другий параметр при створюванні об’єкту, який задає (обмежує) точність представлення мітки часу:

```
n_dt3=np.datetime64(‘2023-02’, ‘D’)
print(n_dt3)
>>>>
2023-02-01
```

Наступний приклад показує створення мітки часу с точністю до хвилин:

```
n_dt4=np.datetime64(‘2017-12-03T03:30’)
print(n_dt4)
>>>>
2017-12-03T03:30
```

А ось так можна створити дані, що будуть представляти дані з точністю до мілісекунд:

```
n_dt5=np.datetime64(‘2025’,’ms’)
print(n_dt5)
>>>>
2025-01-01T00:00:00.000
```

Визначити поточну мітку часу з максимально допустимою точністю:

```
n_dt = np.datetime64(dt.datetime.now())
print(n_dt)
>>>>
2022-11-20T16:56:33.822641
```

Основні принади *numpy* виявляються при роботі з масивами даних, в контексті питань які розглядаються – з масивами даних, що представляють часові мітки:

```
n_arr1 = np.array([np.datetime64('2020-10-22'),
 np.datetime64('2020-10-23'),
 np.datetime64('2020-10-24')], dtype='datetime64[s]')
print(n_arr1)
>>>>
['2020-10-22T00:00:00' '2020-10-23T00:00:00' '2020-10-24T00:00:00']
```

Отримати значення або змінити значення окремого елемента масиву можна в звичайний для *numpy* спосіб:

```
elt=n_arr1[1]
print(elt)
>>>>
2020-10-23T00:00:00

n_arr1[1]=np.datetime64('2022-12-25')
print(n_arr1)
>>>>
['2020-10-22T00:00:00' '2022-12-25T00:00:00' '2020-10-24T00:00:00']
```

Зручним в роботі виявляється відома функція створювання *numpy*-масиву *numpy.arange()*. При роботі з мітками часу ця функція дозволяє створювати масив послідовних дат, що часто використовуються як відповідні мітки часу при дослідженні часових рядів.

```
n_arr2=np.arange('2022-10-22', '2022-11-07', dtype='datetime64[D]')
print(n_arr2)
>>>>
['2022-10-22' '2022-10-23' '2022-10-24' '2022-10-25' '2022-10-26'
 '2022-10-27' '2022-10-28' '2022-10-29' '2022-10-30' '2022-10-31'
 '2022-11-01' '2022-11-02' '2022-11-03' '2022-11-04' '2022-11-05'
 '2022-11-06']
```

Інший спосіб створення діапазону дат:

```
n_arr3=n_dt + np.arange(12)
print(n_arr3)
>>>>
['2022-11-20T16:56:33.822641' '2022-11-20T16:56:33.822642'
 '2022-11-20T16:56:33.822643' '2022-11-20T16:56:33.822644'
 '2022-11-20T16:56:33.822645' '2022-11-20T16:56:33.822646'
 '2022-11-20T16:56:33.822647' '2022-11-20T16:56:33.822648'
 '2022-11-20T16:56:33.822649' '2022-11-20T16:56:33.822650'
 '2022-11-20T16:56:33.822651' '2022-11-20T16:56:33.822652']
```

Об'єкт *datetime64* представляє окремий момент часу. Якщо дві мітки часу мають різні одиниці вимірювання, вони все одно можуть представляти той самий момент часу, що стає зрозумілим з наступного прикладу.

```
np.datetime64('2022') == np.datetime64('2022-01-01')
>>>>
True
```

Тут значення, що стоїть зліва від оператору порівняння спочатку приводиться до точності, що визначається значенням, що стоїть справа від оператору порівняння, а вже потім виконується сама операція логічного порівняння. Відповідно в наступному випадку отримаємо негативний результат виконання операція логічного порівняння:

```
np.datetime64('2022') == np.datetime64('2022-10')
>>>>
False
```

Дані, представлені в форматі *datetime64* можуть порівнюватися між собою за допомогою звичайних логічних операторів, при цьому більшим вважається та мітка часу, яка вказує на пізнішу дату.

```
np.datetime64('2022-09') >= np.datetime64('2022-10')
>>>>
False
np.datetime64('2022-09') <= np.datetime64('2022-10')
>>>>
True
```

Для роботою з часовими інтервалами в *numpy* використовується тип даних *timedelta64*. Аргументами для *timedelta64* є число, яке представляє кількість

одиниць, і одиниця дати/часу, наприклад (D)день, (M)місяць, (Y)рік, (h)година, (m)хвилина або (s)секунда.

```
n_dl1=np.timedelta64(1, 'D')
print(n_dl1)
>>>>
1 days

n_dl2=np.timedelta64(4056, 's')
print(n_dl2)
>>>>
4056 seconds
```

Дані у форматах *datetime64* та *timedelta64* можуть спільно використовуватися, комбінуючись у арифметичні вирази та надаючи можливість зручного обчислення дати й часу.

```
n_dl3=np.datetime64('2019-01-01') – np.datetime64('2018-01-01')
print(n_dl3)
>>>>
365 days

n_dl4=np.datetime64('2019-01-01','s')-np.datetime64('2018-01-01')
print(n_dl4)
>>>>
31536000 seconds

n_dt6=np.datetime64('2021-06-15T10:46') + np.timedelta64(12, 'h')
print(n_dt6)
>>>>
2021-06-15T22:46

res=np.timedelta64(3,'W') / np.timedelta64(1,'D')
print(res)
>>>>
21.0
```

тощо.

Одиниці *timedelta64* «Y» (роки) та «M» (місяці), обробляються спеціальним чином, оскільки неможливо визначити кількість днів у році, не знаючи, чи йдеться про високосний рік чи про звичайний, або про який саме місяць іде мова.

```
year_dl=np.timedelta64(1, 'Y')
month_dl=np.timedelta64(year_dl, 'M')
```

```
print(month_dl)
>>>>
12 months
```

А спроба виконання от такого оператору

```
month_dl=np.timedelta64(year_dl, 'D')
```

викличе виключення типу «*TypeError*».

numpy.busday_offset() – ще одна корисна функція, котра часто використовується для врахування контекста часових рядів, пов'язаних не тільки з вихідними, але й з святковими днями.

```
print(np.busday_offset('2021-02-17', 2))
print(np.busday_offset('2021-02-19', 2))
>>>>
2021-02-19
2021-02-23
```

Зверніть увагу, що в першому випадку функція відрахує і покаже дату, що віддалена від дати, яка задана в якості параметру саме на 2 дні. А от в другому – аж на 4 дні. Це відбулося тому, що два дні припали на суботу та неділю, які вважаються неробочими. На додачу до цього, функція *busday_count()* підраховує кількість робочих днів в проміжку часу.

```
print(np.busday_count(np.datetime64('2021-01-01'), np.datetime64('2022-01-01')))
>>>>
261
```

Тобто в 2021 році більше 100 календарних днів прийшлося на вихідні та святкові дні.

3.4. Робота з часом та датами в pandas

Використовуючи можливості *numpy* по обробці даних типу *datetime64* і *timedelta64*, а також спираючись на інші бібліотеки, що пов'язані з обробкою даних табличного типу, *pandas* пропонує велику кількість нових функцій для управління даними часових рядів.

Окремий об`єкт для роботи з часовими мітками (*Timestamp*) в *pandas* можливо створити наступним чином:

```
time_stamp = pd.Timestamp(dt.datetime(2022, 12, 13))
print(time_stamp)
>>>>
2022-12-13 00:00:00
```

Timestamp-об’єкт бібліотеки *pandas* має багато атрибутів, які можуть бути використані для того, щоб відобразити окремі компоненти часової мітки, такі як рік, місяць, день, година тощо, так само як це робилося при роботі з часом та даними у пакетах, які були розглянуті раніше:

```
print(time_stamp.year)
print(time_stamp.day)
>>>>
2022
13
```

Для відображення часових інтервалів в *pandas* використовується також спеціальний об’єкт «*Period*». Цей об’єкт має атрибут *freq* для зберігання інформації про частоту, яка буде використовуватися при створенні цього об’єкту. За замовчуванням встановлено щомісячну частоту, але можливо використати і іншу частоту, що і показано в наступному прикладі.

```
period = pd.Period(‘2023-01’)
period
>>>>
Period(‘2023-01’, ‘M’)

period1 = pd.Period(‘2023-01’,freq=‘D’)
period1
>>>>
Period(‘2023-01-01’, ‘D’)
```

При необхідності конвертувати параметр «частота» для вже існуючого об’єкту використовується метод *asfreq()*:

```
period2.asfreq(‘H’)
period2
>>>>
Period(‘2023-01’, ‘M’)
```

Об’єкти *pd.Timestamp* та *pd.Period* можуть взаємно конвертуватися один з одним:

```
period2.to_timestamp().to_period(‘M’)
```

При роботі з часовими рядами здебільшого потрібно створювати послідовність дат. Існує декілька варіантів створення такої послідовності. Перший полягає в використанні об’єктів *pandas* типу *Series*. Нагадаємо, що об’єкт типу *Series* модулю *pandas* фізично представляє собою впорядкований

набір пар «індекс-значення». При роботі з часовими рядами логічним вдається в якості індексу використовувати часові мітки, що семантично відображають час настання події або надходження відповідних значень даних:

```
values = [25, 50, 15, 67, 70, 9, 28, 30, 32, 12]
start = dt.datetime(2022, 3, 31)
dates = [start – dt.timedelta(days=x) for x in range(0, len(values))]
ts = pd.Series(values, index=dates)
print(ts)
>>>>
2022-03-31 25
2022-03-30 50
2022-03-29 15
2022-03-28 67
2022-03-27 70
2022-03-26 9
2022-03-25 28
2022-03-24 30
2022-03-23 32
2022-03-22 12
dtype: int64
```

Для роботи з індексом, що представляє часові мітки, *pandas* використовує спеціальний тип даних – *DatetimeIndex*. Побачити, як буде виглядати саме індекс серії в наведеному вище прикладі, можна наступним чином, звичайним при роботі з об’єктами в *pandas*:

```
ts.index
>>>>
DatetimeIndex([‘2022-03-31’, ‘2022-03-30’, ‘2022-03-29’, ‘2022-03-28’,
 ‘2022-03-27’, ‘2022-03-26’, ‘2022-03-25’, ‘2022-03-24’,
 ‘2022-03-23’, ‘2022-03-22’],
 dtype=‘datetime64[ns]’, freq=None)
```

Створимо ще один об’єкт типу *Series*, використовуючи ті самі значення міток часу, пов’язані з значеннями вибірки даних, що і для першого часового ряду:

```
values2 = [32, 54, 18, 61, 72, 19, 21, 33, 29, 17]
ts2 = pd.Series(values2, index=dates)
>>>>
2022-03-31 32
2022-03-30 54
2022-03-29 18
2022-03-28 61
```

```
2022-03-27 72
2022-03-26 19
2022-03-25 21
2022-03-24 33
2022-03-23 29
2022-03-22 17
dtype: int64
```

Над створеними об’єктами можливо виконання арифметичних операцій за правилами, визначеними для таких при роботі з *Series* в *pandas*.

```
print((ts + ts2)/2)
>>>>
2022-03-31 28.5
2022-03-30 52.0
2022-03-29 16.5
2022-03-28 64.0
2022-03-27 71.0
2022-03-26 14.0
2022-03-25 24.5
2022-03-24 31.5
2022-03-23 30.5
2022-03-22 14.5
dtype: float64
```

Так само як і при роботі з іншими об’єктами типу *Series*, значення індексів в *Series*, що представляють часові ряди, можуть не збігатися. В такому разі значення, результат яких визначити при виконанні арифметичних операцій не вдається, заміняється на значення *NaN*:

```
start3 = dt.datetime(2022, 3, 26)
dates3 = [start3 – dt.timedelta(days=x) for x in range(0, len(values))]
ts3 = pd.Series(values2, index=dates3)
print(ts+ts3)
>>>>
2022-03-17 NaN
2022-03-18 NaN
2022-03-19 NaN
2022-03-20 NaN
2022-03-21 NaN
2022-03-22 84.0
2022-03-23 93.0
2022-03-24 48.0
2022-03-25 82.0
2022-03-26 41.0
2022-03-27 NaN
```

```
2022-03-28 NaN
2022-03-29 NaN
2022-03-30 NaN
2022-03-31 NaN
Freq: D, dtype: float64
```

При роботі з часовими рядами індекс складається з послідовності часових міток. Створити таку послідовність можна в інший спосіб, а саме:

```
ind_ts = pd.date_range('30/10/2022', '11/08/2022')
print(ind_ts)
>>>>
DatetimeIndex(['2022-10-30', '2022-10-31', '2022-11-01', '2022-11-02',
 '2022-11-03', '2022-11-04', '2022-11-05', '2022-11-06',
 '2022-11-07', '2022-11-08'],
 dtype='datetime64[ns]', freq='D')
```

Зверніть увагу як справно *pandas* розбирає формат представлення дати. Дійсно, в цьому пакеті вбудовано досить потужний парсер дат, який здатен розібрати та, як правило, коректно представити дати, що задаються в різноманітних форматах, які зустрічаються в прикладних задачах. Для прикладу, наступний оператор дасть абсолютно ті самі результати, що і попередній:

```
ind_ts = pd.date_range('10.30.2022',np.datetime64("2022-11-08"))
```

Позатим, оскільки в деяких випадках навіть людині неможливо однозначно інтерпретувати дату (наприклад, 2022/11/7/ може означати і сьоме листопада 2022 року, і одинадцяте липня 2022 року), програмісту завжди краще впевнитися в тому, що парсер правильно розібрав дату та, у разі необхідності, спробувати або змінити представлення даних, або явно описати дані за допомогою літералів форматування, розглянутих раніше в цій главі.

В приладах вище при створенні індексу задавалися початкова та кінцева дати. Замість цього можливо задати тільки одну з вказаних дат та за допомогою параметра-ключового слова '*periods*' вказати кількість часових міток, які треба згенерувати.

```
ind_ts4 = pd.date_range(end='30/10/2022', periods=4)
print(ind_ts4)
>>>>
DatetimeIndex(['2022-10-27', '2022-10-28', '2022-10-29', '2022-10-30'],
dtype='datetime64[ns]', freq='D')
```

В разі, якщо часові мітки мають йти з періодом, відмінним від доби, вказати на це можливо за допомогою параметру *freq*. В якості значень він може приймати символьні літерали, наприклад «D» (календарний день), «W» (тиждень), «M» (закінчення місяця), «MS»(початок місяця), «Q» та «QS» (відповідно закінчення та початок каландарного кварталу), «A» та «AS» (закінчення та початок року), «H» (години), «T» (хвилини), «S» (секунди), «L» (мілісекунди), «U» (мікросекунди).

```
ind_ts5 = pd.date_range('2022-08-05', '2022-09-13', freq="W")
print(ind_ts5)
>>>>
DatetimeIndex(['2022-08-07', '2022-08-14', '2022-08-21', '2022-08-28',
 '2022-09-04', '2022-09-11'],
 dtype='datetime64[ns]', freq='W-SUN')
```

В цьому прикладі часові мітки створюються по датам неділь (по замовчуванню), однак день тижня можна також задати явно, за бажанням програміста, наприклад:

```
ind_ts6 = pd.date_range('2022-08-05', '2022-09-13', freq="W-Fri")
print(ind_ts6)
>>>>
DatetimeIndex(['2022-08-05', '2022-08-12', '2022-08-19', '2022-08-26',
 '2022-09-02', '2022-09-09'],
 dtype='datetime64[ns]', freq='W-FRI')
```

За допомогою множника можна створювати практично довільні інтервали між часовими мітками:

```
date= pd.date_range(start='1/1/2022', periods = 10, freq ='2.4D')
print(date)
>>>>
DatetimeIndex(['2022-01-01 00:00:00', '2022-01-03 09:36:00',
 '2022-01-05 19:12:00', '2022-01-08 04:48:00',
 '2022-01-10 14:24:00', '2022-01-13 00:00:00',
 '2022-01-15 09:36:00', '2022-01-17 19:12:00',
 '2022-01-20 04:48:00', '2022-01-22 14:24:00'],
 dtype='datetime64[ns]', freq='3456T')
```

Зверніть увагу, що в розумінні *pandas* «тиждень» та «7 днів» – це різні поняття. Якщо в якості виміру інтервалу між часовими мітками вказується тиждень («W») – спочатку вираховується початок тижня, а вже від нього

відраховується інтервали. Якщо в якості виміру вказується 7 днів («7D») інтервали вираховуються безпосередньо від заданої початкової дати.

```
date1 = pd.date_range('01-01-2021', periods = 4, freq ='W')
print(date1)
>>>>
DatetimeIndex(['2021-01-03', '2021-01-10', '2021-01-17', '2021-01-24'],
dtype='datetime64[ns]', freq='W-SUN')

date2 = pd.date_range('01-01-2021', periods = 4, freq ='7D')
print(date2)
>>>>
DatetimeIndex(['2021-01-01', '2021-01-08', '2021-01-15', '2021-01-22'],
dtype='datetime64[ns]', freq='7D')
```

Тепер розглянемо особливості використання часових даних в основних об'єктах пакету *pandas*, а саме в об'єктах типу *DataFrame*. Для початку створимо новий об'єкт вказаного типу за заповнимо його штучними даними.

```
date_rng = pd.date_range(start='1/1/2022', end='1/08/2022', freq='H')
df = pd.DataFrame(date_rng, columns=['Дата'])
df['Значення'] = np.random.randint(0,100,size=(len(date_rng)))
df.head()
```

Перші п'ять рядочків цього об'єкту виглядають так:

```
        Дата Значення
0 2022-01-01 00:00:00 86
1 2022-01-01 01:00:00 76
2 2022-01-01 02:00:00 80
3 2022-01-01 03:00:00 95
4 2022-01-01 04:00:00 86
```

Бачимо, що *DataFrame* індексується за індексом-вказівником, а часові мітки використовуються в якості значень одного з стовпчиків. Якщо ми хочемо надалі маніпулювати часовими рядами, нам знадобиться індексувати *DataFrame* за датою та часом. Конвертуємо індекс на індекс дати й часу, а потім ще раз подивимося, як виглядають перші елементи.

```
df['datetime'] = pd.to_datetime(df['Дата'])
df = df.set_index('datetime')
df.drop(['Дата'], axis=1, inplace=True)
df.head()
>>>>
```

```
                    Значення
datetime
2022-01-01 00:00:00 86
2022-01-01 01:00:00 76
2022-01-01 02:00:00 80
2022-01-01 03:00:00 95
2022-01-01 04:00:00 86
```

Тепер індекс *DataFrame* є індексом-міткою (*index-lable*) з значенням типу часової мітки.

Доволі часто в реальних задачах (наприклад в випадках, коли дані надходять з певного наперед підготовленого файлу) час може бути представлений у строковому (а не числовому) форматі. Тому для подальшої обробки прийдеться такі дані конвертувати з строкового формату на формат об'єкту типу часової мітки. Нагадаємо, що зробити це можна за допомогою, наприклад, функції *strptime()* опис якої був наданий раніше:

```
timestamp_date_rng= [dt.datetime.strptime(x,'%B-%d-%Y') for x in string_date_rng_2]
```

де *string_date_rng_2* – масив часових міток, які представлені в строковому форматі.

Повертаючись до нашого *DataFrame*-об'єкту, покажемо, як можна використати індекс для виконання операцій фільтрації даних за датою. Для прикладу, нехай потрібно відфільтрувати дані отримані лише 5 числа. Перші декілька рядків результату показано нижче:

```
                    Значення
datetime
2022-01-05 00:00:00 27
2022-01-05 01:00:00 0
2022-01-05 02:00:00 7
2022-01-05 03:00:00 63
2022-01-05 04:00:00 18
2022-01-05 05:00:00 17
2022-01-05 06:00:00 61
```

Звісно, аналогічного результату можна досягнути і з використанням безпосередньої вказівки дати, інформація про яку нас цікавить, наприклад:

```
df['2022-01-05']
```

Як і в звичайному *DataFrame,* індекси-мітки часу можуть використовуватися для створення зрізів:

```
df['2022-01-04 02:00':'2022-01-06 14:00']
>>>>
2022-01-04 02:00:00 89
2022-01-04 03:00:00 88
2022-01-04 04:00:00 72
2022-01-04 05:00:00 12
2022-01-04 06:00:00 39
 ...
2022-01-06 10:00:00 44
2022-01-06 11:00:00 8
2022-01-06 12:00:00 42
2022-01-06 13:00:00 95
```

При створенні *DataFrame* ми його заповнили даними з погодинною частотою. Однак в подальшому, при маніпулюванні даними ми можемо використати іншу бажану частоту відображення даних. Розберемо окремо, що відбувається при підвищенні частоти відображення даних та при зменшенні цієї частоти.

Щоб проілюструвати, що відбувається, коли ви збільшуєте частоту представлення даних у вибірці, створимо серії з відносно низькою (поквартальною) частотою для 2022 року з цілими значеннями 1–4. Коли вказується щоквартальна періодичність, за умовчанням *pandas* обирає грудень місяць для позначення кінця четвертого кварталу; при необхідності це значення можна змінити.

```
dates = pd.date_range(start='2022', periods=4, freq='Q')
data = range(1, 5)
quarterly = pd.Series(data=data, index=dates)
print(quarterly)
>>>>
2022-03-31 1
2022-06-30 2
2022-09-30 3
2022-12-31 4
Freq: Q-DEC, dtype: int64
```

Тепер збільшимо частоту відображення даних перетворивши частоту з квартальної на щомісячну за допомогою методу *asfreq()*. *Pandas* додає нові дати

закінчення місяця до *DateTimeIndex* між існуючими датами. У результаті між березнем і груднем виявиться кілька місяців із відсутніми даними.

```
monthly = quarterly.asfreq('M')
monthly = monthly.to_frame('baseline')
print(monthly)
>>>>
                baseline
2022-03-31 1.0
2022-04-30 NaN
2022-05-31 NaN
2022-06-30 2.0
2022-07-31 NaN
2022-08-31 NaN
2022-09-30 3.0
2022-10-31 NaN
2022-11-30 NaN
2022-12-31 4.0
```

Давайте порівняємо три способи, якими *pandas* пропонує заповнити пропущені значення під час підвищення дискретизації.

```
monthly['ffill'] = quarterly.asfreq('M', method='ffill')
monthly['bfill'] = quarterly.asfreq('M', method='bfill')
monthly['value'] = quarterly.asfreq('M', fill_value=0)
print(monthly)
>>>
 baseline ffill bfill value
2022-03-31 1.0 1 1 1
2022-04-30 NaN 1 2 0
2022-05-31 NaN 1 2 0
2022-06-30 2.0 2 2 2
2022-07-31 NaN 2 3 0
2022-08-31 NaN 2 3 0
2022-09-30 3.0 3 3 3
2022-10-31 NaN 3 4 0
2022-11-30 NaN 3 4 0
2022-12-31 4.0 4 4 4
```

Стає зрозумілим, що метод заповнення *ffill* («forward fill» – «заповнення вперед») поширює останнє перед відсутнім значенням на всі наступні відсутні значення, що йдуть підряд. Метод *bfill* («backfill» – «заповнення назад») заповнює попередні пропущені значення першими наступними заданими значеннями, а

fill_value просто замінює відсутні значення тими даними, що вказані в якості значення цього параметру.

Звернемося тепер до випадку пониження частоти відображення даних. Зазвичай таку дію поєднують з обчисленням деякої підсумкової статистики відносно саме нової частоти. Це дуже поширена операція, необхідність в якій може виникнути, наприклад, при обробці двох рядів, кожен з яких виміряний зі своєю частотою. Однак треба відразу відзначити, що в разі підвищення дискретизації представлення даних, в *DataFrame* мають бути створені нові рядки, а отже необхідно явно вказати, як саме заповнити або інтерполювати відсутні значення в цих рядках. При зменшенні дискретизації, кількість рядків в *DataFrame* зменшуються, а отже необхідно вказати, як саме агрегувати наявні дані.

Для виконання описаних дій *pandas* пропонує спеціальний метод *resemple()*, який змінює частоту даних часових рядів. Цей метод дотримується логіки, подібної до методу групування даних *groupby*: він групує дані, що відносяться до заданого нового періоду вибірки та застосовує відповідні дії отримання сумарної статистики до всіх елементів цієї групи.

Наприклад в наступних прикладах повернемося до створеного вище *DataFrame* df, в якому дані задані погодинно, та спочатку підрахуємо сумарне значення показника за добу (тобто – знизимо частоту представлення значень від погодинної до денної), а в другому прикладі – обчислимо середньогодинне значення даних в групах, які утворимо:

```
print(df.resample('D').sum())
>>>>
                Значення
datetime
2022-01-01 1367
2022-01-02 1318
2022-01-03 1351
2022-01-04 1241
2022-01-05 948
2022-01-06 1126
2022-01-07 1286
2022-01-08 67

print(df.resample('D').mean())
>>>>
```

```
                    Значення
datetime
2022-01-01 56.958333
2022-01-02 54.916667
2022-01-03 56.291667
2022-01-04 51.708333
2022-01-05 39.500000
2022-01-06 46.916667
2022-01-07 53.583333
2022-01-08 67.000000
```

Майже аналогічним чином можна створити статистику за принципом «ковзаючого вікна». Додаємо до нашого *DataFrame* новий стовпець та заповнимо його значеннями ковзаючої суми за 5 попередніх періодів:

```
df[‘Ковзаюче 5-денне середнє’] = df.rolling(5).sum()
df.head(10)
>>>>
                                Значення Ковзаюче 5-денне середнє
datetime
2022-01-01 00:00:00 59 NaN
2022-01-01 01:00:00 70 NaN
2022-01-01 02:00:00 85 NaN
2022-01-01 03:00:00 58 NaN
2022-01-01 04:00:00 71 343.0
2022-01-01 05:00:00 85 369.0
2022-01-01 06:00:00 21 320.0
2022-01-01 07:00:00 99 334.0
2022-01-01 08:00:00 94 370.0
2022-01-01 09:00:00 94 393.0
```

Зверніть увагу, що перші чотири (за шириною ковзаючого вікна мінус одиниця) значень нового стовпчика не отримують реальних значень (отримують значення-об’єкт NaN). Це відбувається саме тому, що для цих рядків даних визначити середнє за п’ять попередніх значень ряду неможливо (бо вони просто відсутні). Для заповнення таких відсутніх даних можна використати один з методів, що були згадані вище. Наприклад, створимо ще один стовпчик та перепишемо в нього дані, взявши їх з стовпчика «Ковзаюче 5-денне середнє» але замінимо при цьому відсутні значення на перше існуюче значення в цьому стовпчику:

```
df[‘Ковзаюче відновлене’] = df[‘Ковзаюче 5-денне середнє’].fillna(method=‘backfill’)
df.head(10)
```

```
>>>>
            Значення Ковзаюче 5-денне середнє Ковзаюче відновлене
datetime
2022-01-01 00:00:00 59 NaN 343.0
2022-01-01 01:00:00 70 NaN 343.0
2022-01-01 02:00:00 85 NaN 343.0
2022-01-01 03:00:00 58 NaN 343.0
2022-01-01 04:00:00 71 343.0 343.0
2022-01-01 05:00:00 85 369.0 369.0
2022-01-01 06:00:00 21 320.0 320.0
2022-01-01 07:00:00 99 334.0 334.0
2022-01-01 08:00:00 94 370.0 370.0
2022-01-01 09:00:00 94 393.0 393.0
```

3.5. Приклад обробки даних часового ряду, отриманого при моніторингу мережевого трафіку

При аналізі часових рядів здебільшого використовуються дві форми завдання значень часу – в вигляді часової мітки в звичному для людині форматі та в вигляді проміжку часу або від деякої умовної дати, яка відповідає часу надходження першого значення вибірки значень ряду, або у вигляді інтервалу часу від попередньої події. Як приклад можна привести відому програму моніторингу трафіку ***Wireshark***, в якій саме так – в інтервалах між першою і поточною датою та і інтервалах від попередньої дати – представлена інформація про час надходження чергового пакету даних. Отже, розглянемо, як засобами *Python* та *pandas* можна виконати трансформацію часових даних вказаних форм представлення. *DataFrame* містить інформацію про момент часу, коли відбувалася подія надходження пакету даних та розмір цього пакету. Початкові рядки DataFrame виглядають наступним чином:

```
TimeMark                    Values
2022-03-26 12:48:25.239026 294
2022-03-26 12:48:25.355425 505
2022-03-26 12:48:25.486731 227
2022-03-26 12:48:25.697087 125
2022-03-26 12:48:25.905349 366
2022-03-26 12:48:26.023998 379
2022-03-26 12:48:26.269897 33
2022-03-26 12:48:26.440253 121
2022-03-26 12:48:26.575350 268
2022-03-26 12:48:26.691990 214
…….
```

Для візуальної оцінки даних побудуємо відповідний графік (Рис. 3.1)

В разі необхідності визначити час в інтервалах відносно початку послідовності подій, додаємо до *DataFrame* новий стовпчик та заповнимо його даними, що отримуються в результаті відповідного перерахунку:

```
df2.insert(0,'Rel_Time',df2.index – df2.index.min())
df2.head(10)
>>>>
TimeMark                    Rel_Time                Values
2022-03-26 12:48:25.239026 0 days 00:00:00          294
2022-03-26 12:48:25.355425 0 days 00:00:00.116399 505
2022-03-26 12:48:25.486731 0 days 00:00:00.247705 227
2022-03-26 12:48:25.697087 0 days 00:00:00.458061 125
2022-03-26 12:48:25.905349 0 days 00:00:00.666323 366
2022-03-26 12:48:26.023998 0 days 00:00:00.784972 379
2022-03-26 12:48:26.269897 0 days 00:00:01.030871 33
2022-03-26 12:48:26.440253 0 days 00:00:01.201227 121
2022-03-26 12:48:26.575350 0 days 00:00:01.336324 268
2022-03-26 12:48:26.691990 0 days 00:00:01.452964 214
…….
```

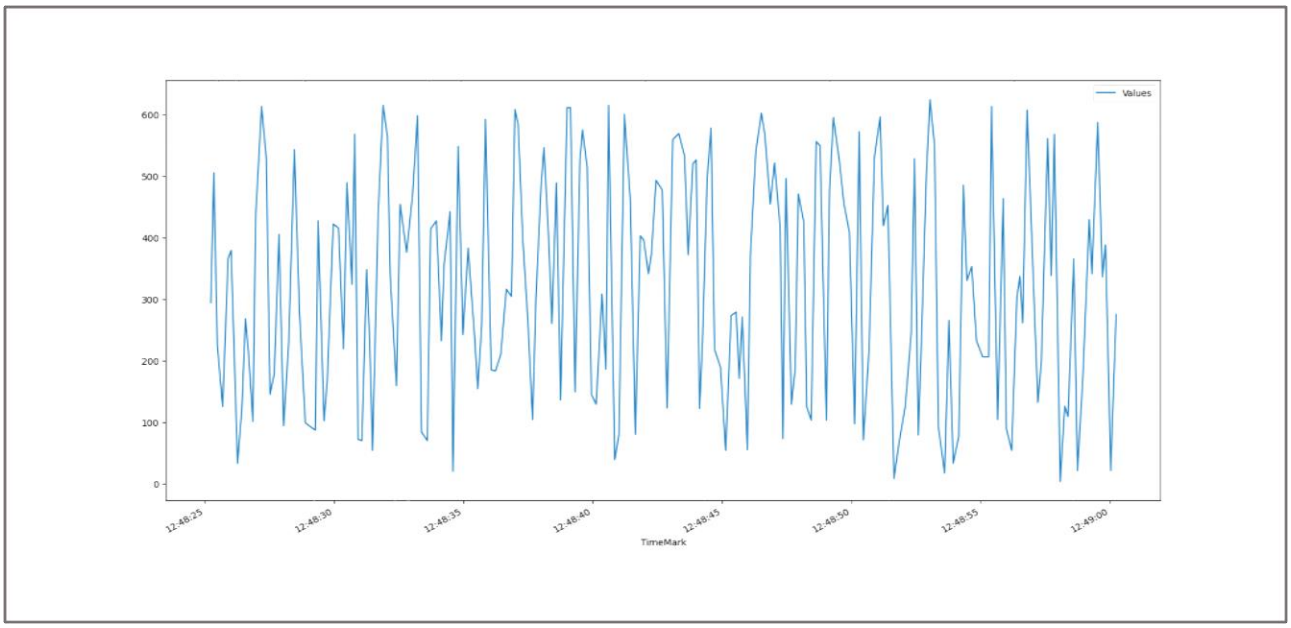


Рис. 3.1 – Приклад представлення параметрів мережевого трафіку у вигляді графіку часового ряду

Створений стовпчик *Rel_Time* можна перетворити на індекс. Якщо ми хочемо при цьому зберігати і початкову часову мітку в *DataFrame*, то зробити ці дії можна наступним чином:

```
df2['Mrk_Time']=df2.index
df2=df2.set_index('Rel_Time')
df2.head(10)
>>>>
Rel_Time                Values Mrk_Time
0 days 00:00:00 294             2022-03-26 12:48:25.239026
0 days 00:00:00.116399 505          2022-03-26 12:48:25.355425
0 days 00:00:00.247705 227          2022-03-26 12:48:25.486731
0 days 00:00:00.458061 125          2022-03-26 12:48:25.697087
0 days 00:00:00.666323 366          2022-03-26 12:48:25.905349
0 days 00:00:00.784972 379          2022-03-26 12:48:26.023998
0 days 00:00:01.030871 33           2022-03-26 12:48:26.269897
0 days 00:00:01.201227 121          2022-03-26 12:48:26.440253
0 days 00:00:01.336324 268          2022-03-26 12:48:26.575350
0 days 00:00:01.452964 214          2022-03-26 12:48:26.691990
```

Третій спосіб завдання часової мітки – в вигляді інтервалу від попереднього значення. В нашому *DataFrame* це можна зробити різними шляхами. Ми отримаємо бажані значення з початкової мітки часу:

```
df2['Diff_Time']=df2.Mrk_Time.diff()
df2[['Mrk_Time','Diff_Time']].head(10)
>>>>
Rel_Time                        Mrk_Time            Diff_Time
0 days 00:00:00         2022-03-26 12:48:25.239026 NaT
0 days 00:00:00.116399      2022-03-26 12:48:25.355425 0 days 00:00:00.116399
0 days 00:00:00.247705      2022-03-26 12:48:25.486731 0 days 00:00:00.131306
0 days 00:00:00.458061      2022-03-26 12:48:25.697087 0 days 00:00:00.210356
0 days 00:00:00.666323      2022-03-26 12:48:25.905349 0 days 00:00:00.208262
0 days 00:00:00.784972      2022-03-26 12:48:26.023998 0 days 00:00:00.118649
0 days 00:00:01.030871      2022-03-26 12:48:26.269897 0 days 00:00:00.245899
0 days 00:00:01.201227      2022-03-26 12:48:26.440253 0 days 00:00:00.170356
0 days 00:00:01.336324      2022-03-26 12:48:26.575350 0 days 00:00:00.135097
0 days 00:00:01.452964      2022-03-26 12:48:26.691990 0 days 00:00:00.116640
```

Нарешті покажемо, як маючи дані у вигляді «інтервал від попереднього» та значення початкового часу отримати системний час.

```
df2.Diff_Time[0]=dt.timedelta(0)
df2['Reculc_Time']=df2.Mrk_Time[0]+df2.Diff_Time.cumsum()
df2[['Mrk_Time','Reculc_Time']].head(10)
>>>>
Rel_Time                                Mrk_Time Reculc_Time
0 days 00:00:00         2022-03-26 12:48:25.176452 2022-03-26 12:48:25.176452
0 days 00:00:00.177157      2022-03-26 12:48:25.353609 2022-03-26 12:48:25.353609
```

```
0 days 00:00:00.300004    2022-03-26 12:48:25.476456 2022-03-26 12:48:25.476456
0 days 00:00:00.451650    2022-03-26 12:48:25.628102 2022-03-26 12:48:25.628102
0 days 00:00:00.643676    2022-03-26 12:48:25.820128 2022-03-26 12:48:25.820128
0 days 00:00:00.856271    2022-03-26 12:48:26.032723 2022-03-26 12:48:26.032723
0 days 00:00:00.983208    2022-03-26 12:48:26.159660 2022-03-26 12:48:26.159660
0 days 00:00:01.118069    2022-03-26 12:48:26.294521 2022-03-26 12:48:26.294521
0 days 00:00:01.231330    2022-03-26 12:48:26.407782 2022-03-26 12:48:26.407782
0 days 00:00:01.375297    2022-03-26 12:48:26.551749 2022-03-26 12:48:26.551749
```

Порівняння значень в стовпчиках *Mrk_Time* та *Reculc_Time* показує, що дані ми відновили абсолютно коректно.

Як видно, інформація, що представлена у *DataFrame* df2 надходила в довільні моменти часу. Часто виникає проблема привести цю інформацію до певних еквідистантних моментів, тобто розбити весь час спостереження на інтервали та підрахувати значення (суму, середнє тощо), що надходили в кожний з часових проміжків. Така задача виникає, наприклад, при переході від представлення трафіку мережі в формі – «момент надходження/об’єм пакету» до представлення того-ж трафіку у формі «інтервал надходження/сумарний об’єм інформації, що був переданий мережею за інтервал». Для прикладу, що розглядається, виберемо в якості довжини інтервалу одну секунду. Крім того, визначимо середнє значення об’єму трафіку за попередні 10 секунд, що дозволяє згладити окремі викиди, та дати усереднену картину поточного завантаження мережі. Відобразимо ці дані на графіку (Рис. 3.2) , причому щосекундний об’єм трафіку відобразимо у вигляді стовпчикової діаграми, а усереднений – у вигляді лінійного графіку (Детальний опис функцій побудови гістограм буде наведений в главі 6).

```
import matplotlib.pyplot as plt
df3=df2.resample('S').sum()
plt.bar(df3.index.astype(str),df3.Values,width=0.5)
plt.xticks(rotation = 45)
df3['Volume']=df3.rolling(10).mean()
plt.plot(df3.index.astype(str),df3.Volume,lw=3,color="red",ls='--')
```

В даному розділі розглянуті засоби відображення подій, що відбуваються в певні моменти часу, а також методи обробки значень, пов’язані з моментом їх

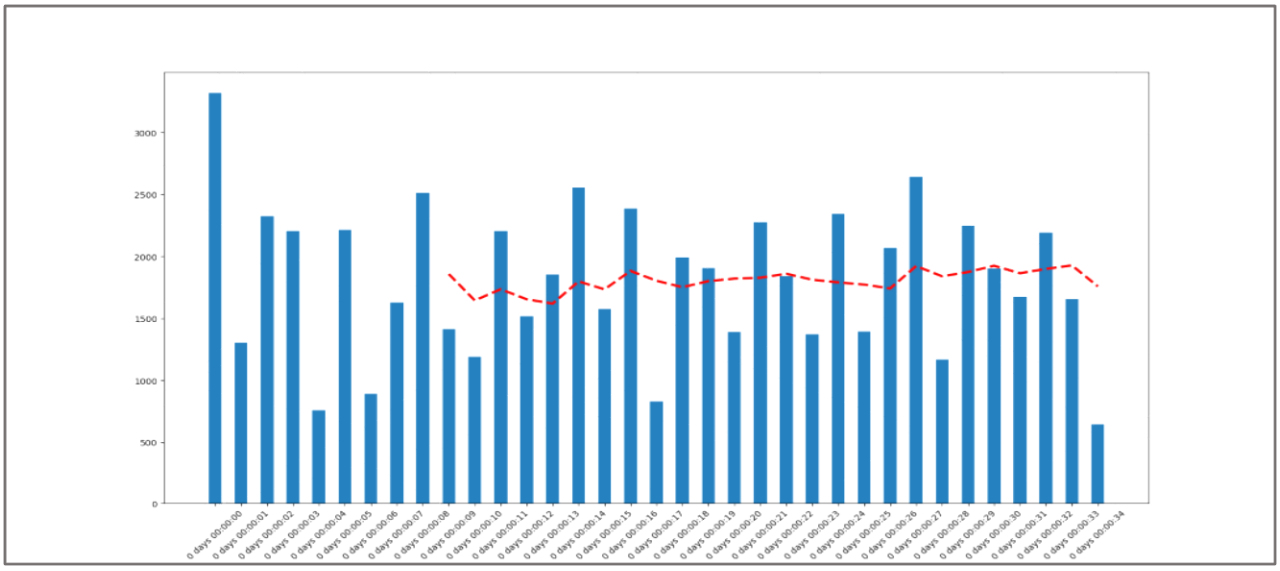


Рис. 3.2 – Різні форми представлення інформації про часовий ряд (на прикладі часового ряду мережевого трафіку)

отримання. Були вивчені об’єкти представлення моментів та інтервалів часу та спеціалізовані модулі для роботи з часом що використовуються при програмуванні на мові *Python*. Було показано, як за допомогою цих інструментів можливо виконати найбільш поширені маніпуляції з даними часового ряду, включаючи з зміною частоти відображення значень часового ряду, використання ковзаючих вікон тощо. Ці операції широко використовуються при поглибленому аналізу та моделюванні об’єктів, зокрема в задачах моніторингу, передбачення та виявлення змін в поведінці об’єктів реального світу.

Контрольні питання

1. Що таке «системний час» та чим він відрізняється від астрономічного часу?
2. Яке основне призначення модулю time зі стандартної бібліотеки *Python*?
3. В чому відмінність об’єктів типу класу *datetime.datetime* від об’єктів, які використовуються в модулі time?
4. Для виконання яких маніпуляцій над даними призначений об’єкт *datetime.timedelta* ?
5. Для чого в модулі pandas використовується об’єкт класу *DatetimeIndex*?
6. За допомогою яких програмних інструментів в модулі *pandas* реалізуються операції, що пов’язані з використанням ковзаючих вікон?

4. ЗАСОБИ *PYTHON* ДЛЯ ВИКОРИСТАННЯ В ЗАДАЧАХ ОПИСОВОЇ СТАТИСТИКИ

Описова статистика – це розділ статистики, який займається первинною обробкою емпіричних даних, їх візуалізацією та їх кількісним описом через основні статистичні показники. Основні статистичні показники, які використовуються для опису набору даних – це міри центральної тенденції та міри мінливості. До мір центральної тенденції включають середнє значення, медіану, моду, а до мір мінливості — стандартне відхилення (чи дисперсію), мінімальне та максимальне значення змінної, розмах, ексцес та коефіцієнт асиметрії.

Розподіли однієї змінної називаються одновимірними розподілами. Їх можна розділити на дискретні розподіли, де спостереження можуть приймати лише цілі значення (наприклад, кількість пакетів, що проходять мережею за певний проміжок часу); і неперервні розподіли, де змінні спостереження є плаваючими значеннями (наприклад, кількість байт інформації, що проходять мережею за певний проміжок часу).

4.1. Основні характеристики розподілів. Центр розподілу

4.1.1. Середнє значення вибірки

Середнє значення будь якої вибірки позначається як $\overline{x}$ і визначається за формулою:

$$\overline{x} = \frac{\sum_{i=1}^{N} x_i}{N}$$

де x_i -елементи вибірки, N - кількість елементів в вибірці.

Середнє значення вибірки легко вирахувати без звертання до будь яких спеціальних інструментів та додаткового імпортування бібліотек. Наприклад, одна з можливих реалізацій засобами виключно базового *Python* виглядає так:

```
list_x=[11,9,14,9,12,9,11,11,7,7,7]
i=0
sum=0
while i < len(list_x):
 sum+=list_x[i]
 i+=1
print(sum/i )
>>>>
```

```
9.727272727272727
```

Якщо дані представляють собою масив *numpy* – то все стає ще простіше:

```
arr_x=np.array([11,9,14,9,12,9,11,11,7,7,7])
mn=arr_x.sum() / len(arr_x)
print(mn)
```

Позатим існують спеціальні функції виконання цієї, а також інших базових операцій описової статистики, реалізовані в пакетах-надбудовах. З точку зору програміста слід розуміти, що зазвичай такі функції виконуються швидше, ніж реалізації, що написані на базових інструментах, здатні працювати з різними типами даних, реагувати на відповідні ситуації виключення тощо. Так, для обчислення середнього значення вибірки в пакеті *numpy* використовується метод *numpy.mean()*

```
import numpy as np
arr=np.arange(278,dtype=float)
mn=arr.mean()
print(mn)
>>>>
138.5
```

Реальні дані часто містять відсутні значення, які в багатьох випадках замінюються *nan* (*nan* розшифровується як «*Not-A-Number*»). Якщо до таких даних застосувати метод *numpy.mean()* – бажаного результату отримати не вдасться:

```
arr[10::44]=np.nan
mn=arr.mean()
print(mn)
>>>>
Nan
```

Тому для обрахування статистик масивів, які включають значення *nan*, що може скласти додаткові труднощі при самостійному програмуванні навіть найпростіших статистичних функцій, в numpy існує ряд функцій, які легко розпізнати – їх назва починається з «nan». Вони мають вбудовані алгоритми перевірки та «відсіювання» даних, обробити які неможливо.

```
mn=np.nanmean(arr)
print(mn)
>>>>
138.40959409594095
```

В numpy, в якому автори намагаються задовольнити якнайширше коло потреб дослідників та інженерів, існує ще одна функція, що має відношення до вказаної статистики, а саме функція визначення «зваженого середнього», або точніше «середнє арифметичне зважене». Ця статистика визначається формулою:

$$\bar{x} = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i}$$

Тут елементи набору значень $\{w_i\}$ носять назву «вагових коефіцієнтів». Якщо всі вагові коефіцієнти рівні між собою, то зважене середнє арифметичне буде дорівнювати середньому арифметичному. Зважене середнє арифметичне застосовується тоді, коли в дослідженні кожному елементу придають різну важливість при визначені результату. Призначення конкретних значень цим коефіцієнтам – задача позастатистична, обумовлена відповідними прикладними особливостями даних. (Наприклад, в фізиці саме через середнє зважене визначається середня швидкість тіла, що може змінювати свою швидкість, центр маси деякого тіла, або температура суміші кількох порцій однієї рідини з різними початковими температурами. В екології, при узгодженні рейтингів об'єктів експертами, вагові коефіцієнти часто використовуються для врахування досвіду кожного з експертів. Пересічна людина, купуючи певний товар оцінює його – може і підсвідомо – по деяким показникам, надаючи кожному з таких показників вагу виходячи з його важливості саме для цієї людини.)

Для розрахунку зваженого середнього в *numpy* використовується функція *avarege()*, параметр якої *weights* приймає значення вагових коефіцієнтів, які присвоюються кожному елементу вибірки, яка задається першим параметром. Наступний приклад показує спосіб використання вказаної функції та відмінність в результатах між розрахунком середнього арифметичного та зваженого з вказаними вагами середнього:

```
x = [6.0, 1, 2.5, 6, 25.0]
w = [0.1, 0.2, 0.3, 0.25, 0.15]
mean= np.mean(x)
wmean= np.average(x, weights=w)
print('Середнє арифметичне= {}, Зважене середнє ={}'.format(mean,wmean))
>>>>
Середнє арифметичне= 8.1, Зважене середнє =6.8
```

4.1.2. Медіана вибірки

Медіана – це значення, яке розділяє набір даних навпіл за кількістю елементів менших та більших за неї. Медіану можна також уявляти як значення, яке займає $N/2$-позицію за умови, що дані будуть відсортовані (або відранжовані). Медіану можна знайти за допомогою функції *np.median()*

```
np.random.seed(12345)
arr=np.random.randint(100,size=80)
mn=np.median(arr)
print(mn)
>>>> 42.0
```

Якщо в вхідній вибірці присутня парна кількість елементів, то за означенням медіана дорівнює середньому арифметичному двох серединних чисел у відранжованій послідовності.

Окрім зазначених методів з пакету *numpy*, в екосистемі *Python* існує окремий вбудований пакет *statistics*, що також має відповідні методи для обчислення статистичних параметрів вибірок. В деяких випадках методи та функії цього пакету дають змогу отримати результати, що відсутні в *numpy* та *scipy*. Наприклад, в цьому модулі існують дві спеціальні функції, що здатні знайти медіану (а точніше – пару елементів набору між якими знаходиться медіана) в випадку, коли в самому наборі присутня парна кількість значень

```
import statistics as st
md_l= st.median_low([1, 3, 5, 8, 11, 12])
md_h=st.median_high([1, 3, 5, 8, 11, 12])
print (‘md_l=‘,md_l,end=‘\n’)
print (‘md_h=‘,md_h,end=‘\n’)
>>>>> md_l= 5
>>>>>md_h= 8
```

4.1.3. Мода вибірки

Мода — це значення, яке найчастіше зустрічається в вибірці. Окремого методу обрахунку моди в *numpy* немає. І в разі потреби застосовують метод з іншого пакету-надбудови над *numpy*, а семе з модулю *scipy.stats*, який повертає значення моди, а також відображає кількість таких (що найчастіше зустрічаються) значень в вибірці.

```
import scipy.stats as sps
data = [1, 3, 4, 4, 7, 3, 4, 5, 8, 3, 4]
print (sps.mode(data))
>>>> ModeResult(mode=array([4]), count=array([4]))
```

Присутнія функція для визначення моди і в пакеті *statistics*:

```
np.random.seed(123)
arr=np.random.randint(20,size=1000)
mo=st.mode(arr)
print(mo)
>>>>
7
```

Зауважимо, що мода може обраховуватися не тільки для вибірок даних, що представлені в кількісних шкалах, але й до даних, що представлені в шкалі найменувань:

```
clrs=["red", "blue", "blue", "red", "green", "red", "red"]
print(st.mode(clrs))
>>>> ‘red’
```

Порівняйте отримані результати з результатами, що повертає відповідний метод з бібліотеки *scipy.stats*:

```
print(sps.mode(clrs)[0][0],sps.mode (clrs)[1][0])
>>>> red 4
```

В модулі *statistics* присутні навіть дві функції для пошуку моди. Перша, традиційна, дає одне значення, а друга – в разі якщо в вибірці присутні декілька елементів, які однаково часто зустрічаються серед елементів вибірки – повертає всі такі елементи. З прикладу різниця між цими функціями стає більш зрозумілою:

```
lt=[11,9,14,9,12,9,11,11,7,7,7]
print(st.mode(lt))
>>>> 11
print(st.multimode(lt))
>>>> [11, 9, 7]
```

Нарешті слід зазначити, що пакет *pandas*, який є надбудовою, наслідує майже всі статистичні функції *numpy* та *scipy*.

```
import pandas as pd
np.random.seed(123)
drr1 = np.random.randint (0, 15, 1000)
numbers = pd.Series(drr1)
print (‘Середнє ‘,numbers.mean())
print (‘Медіана ‘,numbers.median())
print (‘Мода ‘,numbers.mode()[0])
>>>>
Середнє 7.023
```

```
Медіана 7.0
Мода 7
```

4.2. Кількісні оцінки мінливості даних

Центральних метрик недостатньо для опису даних. Практично завжди потрібні метрики оцінки варіативності даних, які кількісно визначають розкид точок даних на множині можливих значень. Найпростіший показник розкиду – розмах, тобто різниця між найбільшим та найменшим значенням даних. Його можна знайти за допомогою методу *np.ptp()*:

```
range = np.ptp(drr1)
print('Розмах ',range )
>>>> Розмах 14
```

4.2.1. Квантилі та прецентилі

На Рис. 4.1 представлено співвідношення таких понять, як значення елементів набору даних (x), квантиль (Q), значення функції щільності розподілу (pdf) та значення кумулятивної функції розподілу (cdf). Всі ці характеристики набору даних тісно пов'язані між собою, а їх аналіз становить невід'ємну частину будь-якого дослідження.

Для обчислення відповідних характеристик набору даних використовуються наперед реалізовані функції. Так, значення квантилів можуть бути отримані засобами пакету *scipy.stats*.

```
np.random.seed(1935)
drr2 = np.random.normal(size=100000)
print(scs.mstats.mquantiles (drr2, prob = [0.25, 0.5, 0.75]))
>>>> [-0.67655052 -0.00317365 0.67523374]
```

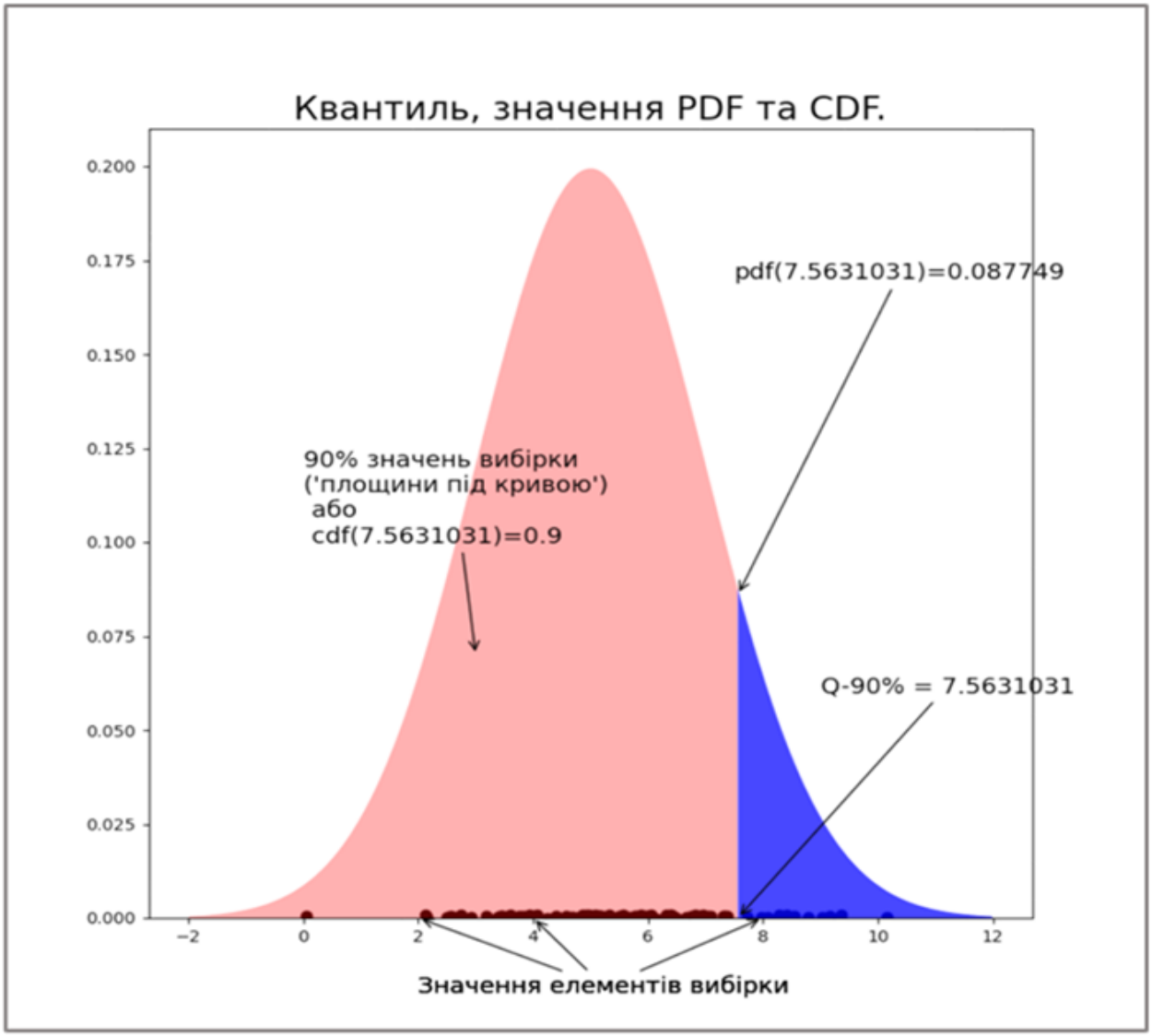


Рис. 4.1 – Основні елементи описової статистики, що використовуються для аналізу варіативногсті даних

З наведеного прикладу видно, що генератор нормально розподілених псевдовипадкових чисел, який використовується для генерації даних для прикладу, створює послідовність чисел, параметри якої досить точно збігаються з теоретичними значеннями для цього розподілу.

Вирішити зворотню задачу, тобто – по заданому числу знайти відсоток даних у наборі менших за це число, або – що тотожно – знайти значення кумулятивної функції розподілу для заданої точки (прецентилю), можна за допомогою функції:

```
print(scs.percentileofscore (drr2, 0.67523374))
>>>> 75.0
```

4.2.2. Дисперсія та стандартне відхилення

Як і математичне очікування, дисперсія є важливою характеристикою випадкової величини. Якщо математичне очікування відображає центр випадкової величини, то дисперсія є показником мінливості даних, тобто показує, наскільки далеко (в середньому) точки вибірки розташовані відносно середньоарифметичного значення вибірки. Говорячи про дисперсію необхідно розрізняти дисперсію генеральної сукупності та дисперсію вибірки з генеральної сукупності. Математично можна довести, що оцінкою дисперсії генеральної сукупності, яка забезпечує максимальне значення статистичної правдоподібності, є оцінка, що визначається наступним чином:

$$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N}$$

де μ- математичне очікування генеральної сукупності. Іншими словами дисперсія – це середній квадрат відхилень значень вибірки від математичного очікування. На практиці, під час аналізу реальних вибірок, математичне очікування, зазвичай, невідомо. Замість нього виникає бажання використати його оцінку – середнє арифметичне. В такому разі розрахунок дисперсії обчислюється за формулою:

$$s^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N} .$$

Однак визначене таким чином значення систематично занижує оцінку по відношенню до дійсного (і здебільшого невідомого нам) значення дисперсії генеральної сукупності, особливо при не дуже великих значеннях N. Тому таку оцінку називають «зміщеною оцінкою» дисперсії генеральної сукупності. На практиці це означає (і може бути досить просто змодельовано програмно), що якщо з деякої генеральної сукупності із заданою дисперсією, M разів вибирати вибірки з N елементів та обчислювати значення статистики за вказаною формулою для кожної з цих вибірок, то середнє значення цих M стандартних відхилень вибірки буде меншим за дійсну дисперсію генеральної сукупності. З теорії ймовірності відомо, що найкраща незміщена оцінка для дисперсії генеральної сукупності визначається як:

$$s^2 = \frac{\sum_{i=1}^{N} \left(x_i - \bar{x}\right)^2}{N-1},$$

і яку, щоб відрізнити від дисперсії описаної вище, називають вибірковою дисперсією.

Єдиною завадою для використання вибіркової дисперсії є те, що одиниця її вимірювання визначається як квадрат одиниці вимірювання значень самої вибірки. Іноді складно, або навіть неможливо дати прикладну інтерпретацію такої величини. Тому набагато частіше використовують похідну від величину, відому як «середньоквадратичне (стандартне) відхилення», та яка визначається як:

$$s = \sqrt{\frac{\sum_{i=1}^{N} \left(x_i - \bar{x}\right)^2}{N-1}} \ .$$

Дисперсію та середньоквадратичне відхилення досить просто вирахувати без будь яких додаткових інструментів:

```
n = len(x)
mean = sum(x) / n
var_ = sum((item – mean)**2 for item in x) / (n – 1)
std = math.sqrt(var)
```

Позатим практично всі пакети, які ми розглядаємо, мають у своєму складі засоби для визначення дисперсії та стандартних квадратних відхилень.

Модуль *statistics* має чотири різні функції, які допомагають розрахувати цей показники розкиду даних:

- *pvariance()* для обчислення дисперсії генеральної сукупності даних;
- *variance()* для розрахунку вибіркової дисперсії;
- *pstdev()* для обчислення стандартного відхилення генеральної сукупності (кореню квадратного від значення σ^2);
- *stdev()* для обчислення стандартного відхилення вибірки.

В модулі *numpy* застосовується дещо інший підхід. Тут існує дві окремі функції:

- *np.var()* – розрахунок дисперсії;
- *np.std()* – розрахунок стандартного відхилення.

Обидві ці функції мають додатковий параметр (*ddof*), який дозволяє вказувати, відносно якого об'єкту (а практично – за якою саме формулою) ведеться розрахунок – генеральної сукупності чи вибірки. В разі роботи з генеральною сукупністю цей параметр приймає значення *ddof=0* (це значення за замовчуванням), в разі роботи з вибіркою – значення *ddof=1*. Відмінність між отриманими значеннями особливо помітна при невеликих значеннях N та майже нівелюється при їх зростанні.

В наступному прикладі порівняємо відповідні статистики:

```
arr = np.arange(7,25)
print('Дисперсія для генеральної сукупності ',arr.var(ddof=0))
print('Оцінка дисперсії по виборці ',arr.var(ddof=1))
print('Стандартне відхилення для генеральної сукупності',arr.std(ddof=0))
print('Оцінка стандартного відхилення по виборці ',arr.std(ddof=1))
>>>>>
Дисперсія для генеральної сукупності 26.916666666666668
Оцінка дисперсії по виборці 28.5
Стандартне відхилення для генеральної сукупності 5.188127472091127
Оцінка стандартного відхилення по виборці 5.338539126015656
```

Слід звернути увагу, що аналогічні функції в пакеті *pandas* хоча і мають подібний інтерфейс, але використовують за замовченням значення параметру *ddof=1*.

4.3. Кількісні оцінки форми розподілу даних. Коефіцієнти скосу (skew) та ексцесу (kurtosis)

Коефіцієнти скосу (інша назва – коефіцієнт асиметрії) та ексцесу це показники, що характеризують геометричну форму розподілу. Асиметрія характеризує (Рис. 4.2) міру скошеності графіка ліворуч/праворуч відносно осі симетрії, а ексцес – міру його висоти. Наприклад, для вимірювань, які не можуть бути негативними (що зазвичай і відбувається в реальному світі, наприклад – час затримки між надходженнями пакетів в мережі, кількість відвідувачів сайту, що одночасно отримують до нього доступ тощо), ми можемо отримати вибірку значень, для якої характерна саме скошеність розподілу вліво. Протилежна, негативна асиметрія, зустрічається досить рідко. Вказані коефіцієнти можуть бути розраховані за формулами:

$$skew = \frac{\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^3}{N * s^3} \text{ та } kurt = \frac{\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^4}{N * s^4}$$

де s – стандартне среднеквадратичне відхилення.

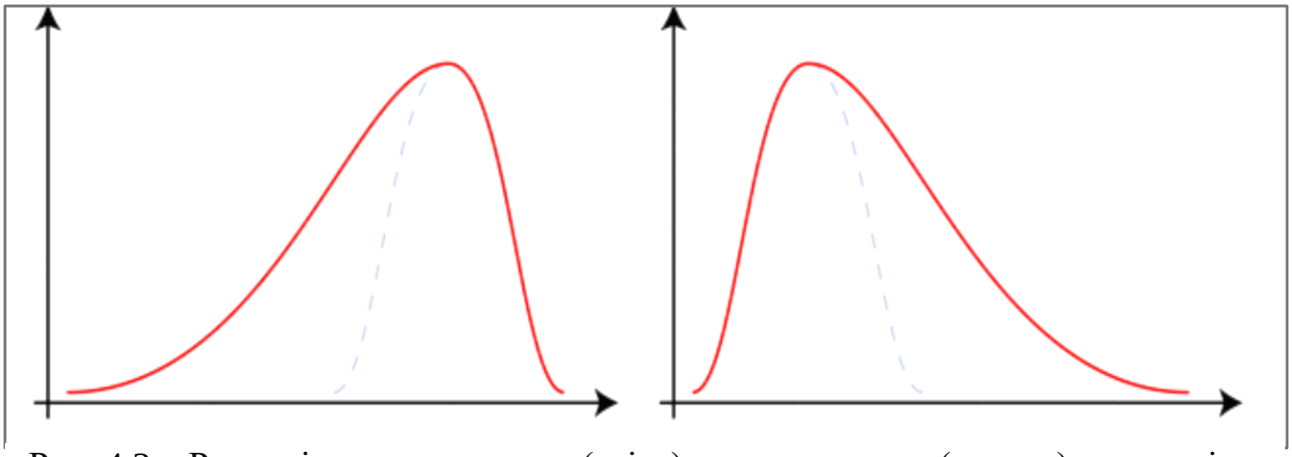


Рис. 4.2 – Розподіли з негативною (зліва) та позитивною (справа) асиметрією

На Рис. 4.3 показано як різняться значення коефіцієнтів скосу та ексцесу в даних, що розподілені за різними законами розподілу, хоча і мають деяку візуальну подібність в формі гістограм. Виявлення такої відмінності може слугувати при вирішенні задач технічної та медичної діагностики, інших задач пов'язаних з визначенням наявності точок змін в часовому ряду. Наприклад, адміністратор сайту або соціальної мережі може отримати інформацію про вік відвідувачів його ресурсу.

За цією інформацією можуть бути обчислені показники цього набору даних. Суттєва відмінність в цих показниках, визначених на різних проміжках часу може вказати на «омолодження», «старіння» аудиторії ресурсу, або взагалі на втрату цікавості користувачів до контенту ресурсу. В будь-якому разі це привід для коригування редакторської політики ресурсу.

Для отримання значень відповідних характеристик вибірок даних в пакеті *scipy.stats* доступні дві відповідні функції: *scs.skew()* та *scs.kurtosis()*. Аналогічні функції присутні і в пакеті Pandas: *df_drr3.skew()* та *df_drr3.kurtosis()*.

Між наведеними реалізаціями існує суттєва відмінність. В *scipy* по замовчуванню виконується т.з. поправка на зміщення коефіцієнтів скосу та ексцесу, в *pandas* – ні. Тому для того, щоб отримати однакові результати при використанні функцій з пакету *scipy.stats* треба за допомогою використання параметру *bias* явно відмовитися від введення вказаної поправки:

```
stats.skew(df_drr3,bias=False)
stats.kurtosis(df_drr3,bias=False)
```

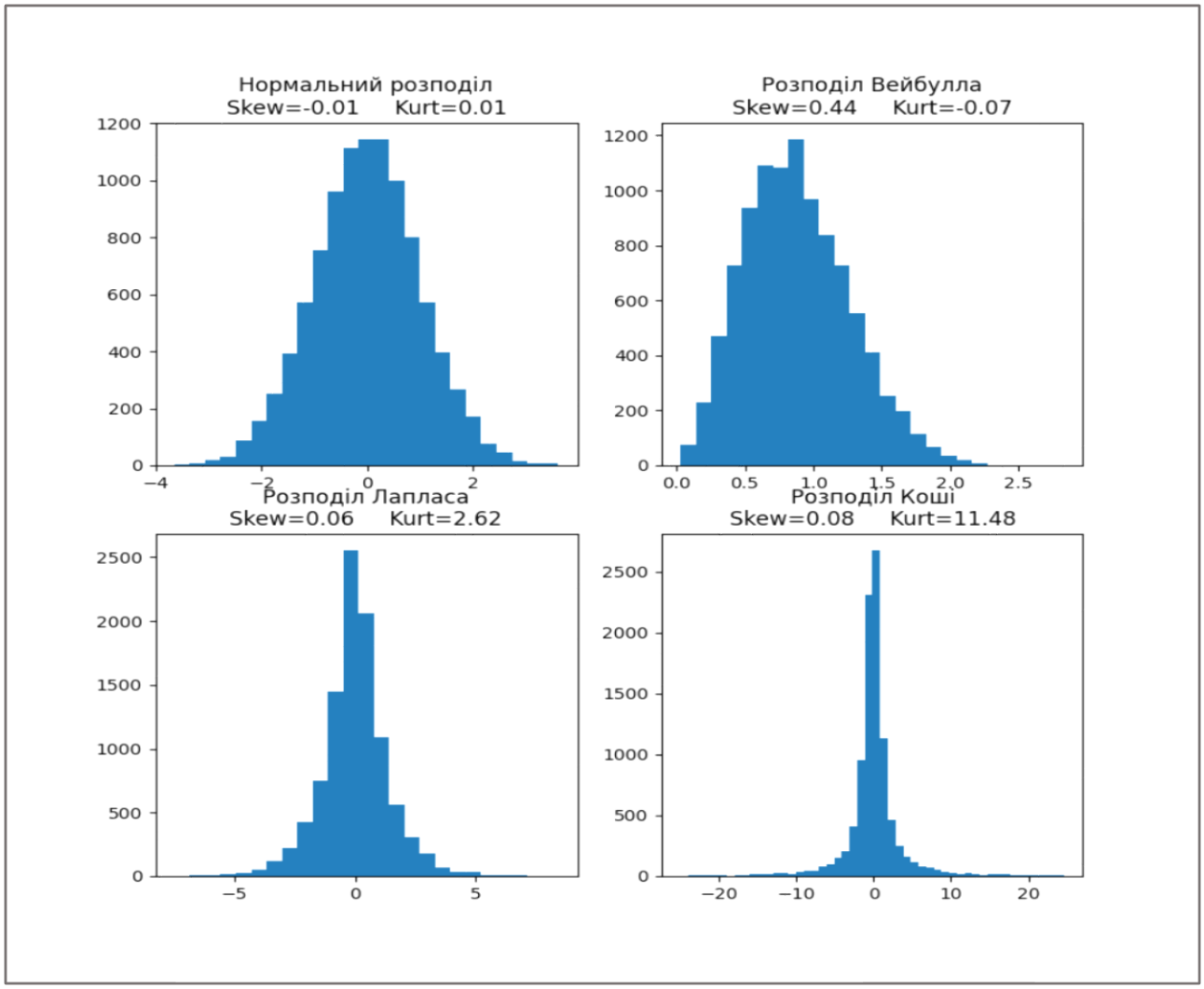


Рис. 4.3 – Залежність форми розподілу даних від значень коефіцієнтів скосу та ексцесу

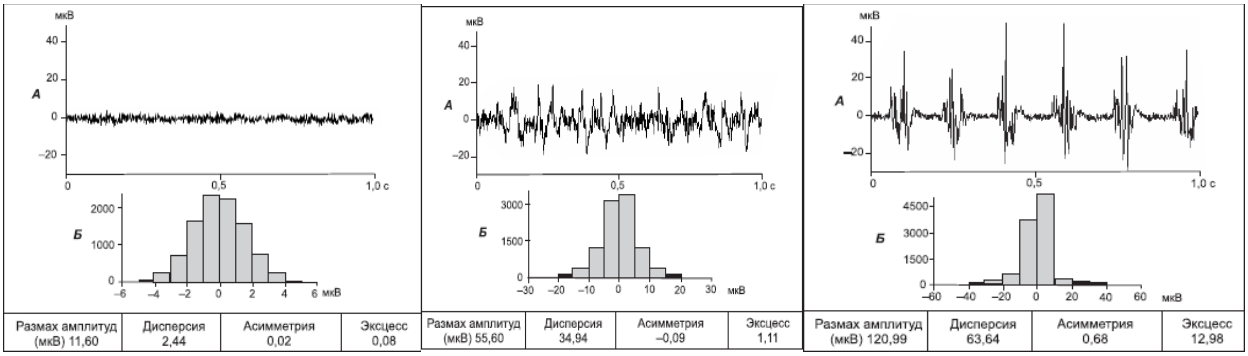


Рис. 4.4 – Приклад застосування описової статистики вибірок до вирішення задач практичної психіатрії. Джерело (http://www.mif-ua.com/archive/issue-22763/article-22791/)

Приклад наведений на Рис. 4.4 демонструє можливість використання розглянутих описових статистик даних для класифікації об’єктів (в даному випадку – пацієнтів), та вирішити відповідну діагностичну задачу практичної психіатрії.

Насам кінець покажемо приклад використання функцій з пакетів *scipy.ststs* та *pandas*, за допомогою яких можна отримати певну зведену інформацію про статистичні параметри вибірки:

```
drr3 = np.random.normal(size=100000)
print(scs.describe(drr3, ddof=1, bias=False))
>>>> DescribeResult(nobs=100000, minmax=(-4.357904697957778, 4.5250738922400044),
mean=-0.0011473676313138038, variance=0.9998934466783658, skewness=-
0.01199473007118635, kurtosis=0.012409207311517179)

drr4=pd.Series(drr3)
print(drr4.describe())
>>>>
count 100000.000000
mean -0.001147
std 0.999947
min -4.357905
25% -0.674788
50% 0.001009
75% 0.672834
max 4.525074
dtype: float64
```

Контрольні питання

1. З чим пов’язані вигоди використання модуля *numpy* при вирішенні задач визначення статистичних параметрів часових рядів?

2. В чому полягає відмінність дисперсії генеральної сукупності та дисперсії вибірки? Як операції визначення цих характеристик реалізовані в модулі *numpy*.

3. Що визначають коефіцієнти скосу та ексцесу?

5. ЗАСОБИ ВІЗУАЛІЗАЦІЇ В АНАЛІЗІ ДАНИХ

В практиці аналізу даних часто буває так, що дослідження не має сенсу навіть розпочинати, допоки дослідник не має можливості переглянути дані у візуальній формі, як-от діаграми та графіки. Вміння швидко візуалізувати вибірки даних та результати аналізу є важливою навичкою як у прикладній статистиці, так і в прикладному машинному навчанні.

Візуалізація даних – це спосіб графічного представлення даних та інформації. В іншому значенні це можна описати як переклад даних у візуальний контекст за допомогою діаграм, графіків, анімації, інфографіки тощо. Візуалізація може бути корисною під час вивчення та початково знайомства з набором даних, може допомогти з виявленням шаблонів, пошкоджених даних, викидів, що присутні у даних тощо. Візуалізацію можна використовувати для вираження та узагальнення інформації та демонстрації ключових зав'язків в даних у вигляді графіків і діаграм. Часто при представленні результатів для осіб, що мало обізнані з теорію аналізу даних, візуалізація може виявитися більш інформативною, ніж строго викладені математичні результати. Однак треба мати на увазі, що візуалізація – це в першу чергу *допоміжний* інструмент дослідника, а її результати можуть слугувати ілюстрацією до отриманих результатів але в жодному разі не замінювати собою науково обґрунтований їх математично-статистичний аналіз.

В цій главі розглядаються основні види діаграм та графіків, які можна використовувати, щоб краще зрозуміти дані, що досліджуються, та базові інструменти, які використовуються при цьому. Вказані інструменти здебільшого зосереджені в *matplotlib* – спеціальному пакеті екосистеми *Python*. Хоча в екосистемі *Python* існують і інші бібліотеки для візуалізації, однак на сьогоднішній день саме *matplotlib* має серед них безумовно найширше розповсюдження. Цей пакет надає досліднику надзвичайно широкі можливості пов'язані з візуалазацію. Крім того він є основою для багатьох інших бібліотек візуалізації та підтримує надбудови, що включені в пакети вищого рівня, зокрема – в *pandas*. Ми розглянемо лише базові, початкові можливості роботи з двомірними статичними графіками, які зосереджені в *pyplot* – окремому підмодулі, що входить до складу *matplotlib*. Даний модуль доцільно імпортувати до коду застосунку та присвоїти йому робочий нікнейм. Зазвичай вказану дію виконують за допомогою оператору:

```
import matplotlib.pyplot as plt
```

Далі розглянемо п’ять видів діаграм, а саме:

- лінійні діаграми (Line Plot);
- стовпчикові діаграми (Bar Chart);
- діаграми розкиду (Scatter Plot);
- діаграми «ящик з вусами» (Box and Whisker Plot), або «бох-плот-діаграми»;
- гістограми (Histogram Plot) – будуть розглянуті в наступній главі посібника.

Діаграми, графіки, їх окремі компоненти та складові, такі як осі, мітки, підписи, легенди, назви, коментарі тощо в *matplotlib* створюються шляхом виклику відповідних методів, що спрощує розуміння доволі складної структури та взаємодії між окремим складовими графіку. Всі ці можливості доступні через використання відповідного інтерфіейсу.

В *matplotlib* реалізовані два різновиди інтерфейсу розробника. Один з них успадковує стиль створювання графіків від добре відомого пакету MATLAB, який широко вживався в науковому та інженерно-технічному середовищі, а отже був досить популярним і звичним інструментом для спеціалістів з обробки даних. Інший, більш осучаснений інтерфейс, спирається на ідеї об’єктно-орієнтованого програмування, які все більше завойовують популярність як серед програмістів, так і серед спеціалістів по аналізу даних.

Багато хто вважає процедурний MATLAB-подібний API *matplotlib* більш простим у використанні, ніж об’єктно-орієнтований API, тому ми спочатку продемонструємо цей процедурний API.

5.1. Лінійна діаграма.

Лінійна діаграма широко використовується в практиці для представлення різноманітних даних, але основне призначення цього графіку – представлення спостережень, зібраних через певний проміжок часу (часовий ряд). При цьому вісь X графіку використовується для представлення регулярного часового інтервалу, або будь-яких послідовностей, в яких існує відношення порядку між спостереженнями. На осі ординат Y відображаються значення спостережень. Самі значення на площині лінійної діаграми з’єднується між собою лінією.

Лінійна діаграма створюється шляхом звертання до функцію *plot()*. Параметри цієї функції приймають посилання на дані значення елементів набору даних (вибірки). Розглянемо приклад створення такого графіку (Рис. 5.1).

```
x = np.linspace(0, 5, 11)
y = x**2
plt.plot(x, y, 'r')
>>>>
```

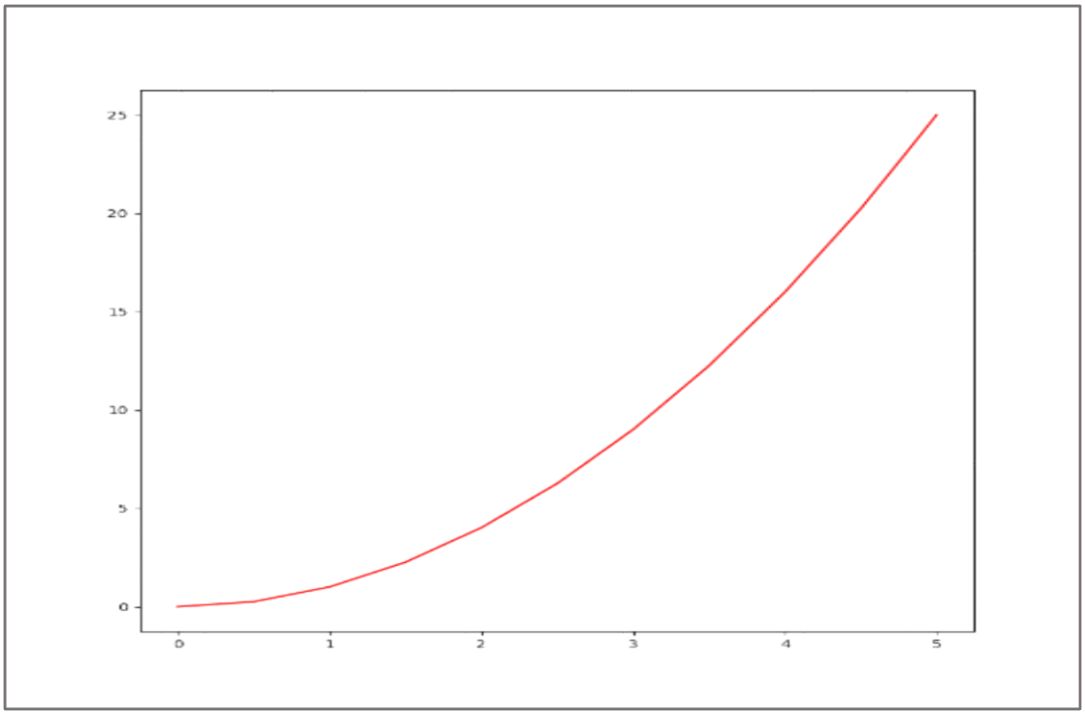


Рис. 5.1 – Приклад графіку, що створений функцією plt.plot()

Перші два оператора моделюють наш набір даних, в даному разі – функцію $y = x^2$. Оскільки кожна точка на графіку має бути вказана парою значень по осям X та Y, в цьому прикладі ми просто генеруємо деякі штучні значення – тобто два набори даних. Зв'язок між елементами цих наборів в прикладі встановлюється по однаковому індексу елементів в кожному з наборів. Для моделювання значень по осі X задаємо 11 значень, рівномірно віддалених одне від одного та таких, які вкладені в діапазон від 0 до 5 включно. Для цього використаємо фунцію *linspace()* з пакету *numpy*, та задаємо їй відповідні параметри, отримавши таким чином масив значень *[0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5.]*. Значення по осі Y змоделюємо як корінь квадратний від значень X.

Окрім параметрів значень по осям функція *plt.plot()* передбачає багато інших параметрів, з яких в наведеному прикладі використовується третій позиційний параметр, що задає бажаний колір графіку. Якщо пропустити

параметр кольору, *matplotlib* буде автоматично обирати для кожного графіку окремий колір. Результат виконання скрипту наведений на рисунку:

Багато елементів дизайну графіку задаються за допомогою виклику відповідних функцій та їх параметрів. Наприклад, доповнимо наш попередній графік підписами осей та назвою графіку, обравши різні шрифт та його розміри (Рис. 5.2). А також змінимо дизайн самого графіку вказавши бажаний стиль, товщину лінії, тип, розмір та колір маркерів:

```
plt.plot(x, y, ‘r’, linestyle =‘-’,lw=5,marker=‘o’,markersize=12,markerfacecolor=‘b’)
plt.xlabel(‘Ось X’,fontsize=20)
plt.ylabel(‘Ось Y’, fontsize=20,fontname="Comic Sans MS")
plt.title(‘Заголовок графика’, fontsize=36)
>>>
```

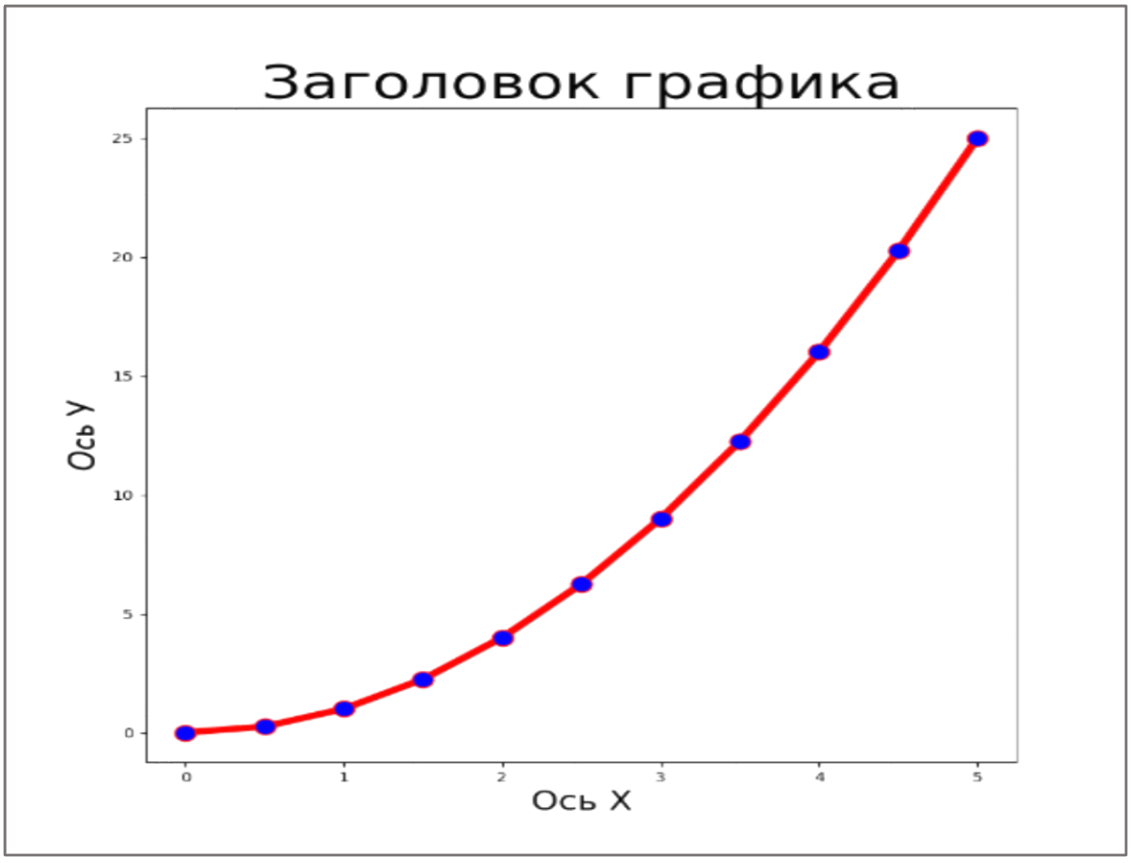


Рис. 5.2 – Приклад використвння стилів підписів та маркерів

Наведений приклад можна дещо скоротити за допомогою спеціального трюку, який полягає в одночасному завданні в третьому позиційному параметру функкції *plot()* відразу трьох елементів стилю графіку, а саме стилю лінії, яка використовується для зображення, типу маркеру для позначення значень, що з’єднується лінією на графіку та кольору самої лінії.

Так , запис

```
plt.plot(x, y, ‘r’, linestyle =‘-’,lw=5,marker=‘o’,markersize=12,markerfacecolor=‘b’)
```

можна скоротити до

```
plt.plot(x, y, ‘r-o ‘, lw=5 ,markersize=12,markerfacecolor=‘b’)
```

Для керування кольорами при відображення діаграм та графіків в бібліотеці *matplotlib* використовується досить складна та розгалужена система позначень. З всіма її особливостями можна ознайомитися в документації до пакета. Тут лише зазначимо, що найпростіший спосіб керування кольорами полягає в використанні умовного позначення у вигляді символьного літералу, прийнятого для основних кольорів, а саме:

"b" – для використання синього кольору;

"g"– для використання зеленого кольору;

"r" – для використання червогого кольору;

"c" – для використання блакитного кольору;

"m" – для використання фіолетового кольору;

"y" – для використання жовтого кольору;

"k" – для використання чорного кольору;

"w"– для використання білого кольору.

Стиль самої лінії лінійного графіку можна задати за допомогою стилю лінії, які, як було показано вище, можна сполучати з кодом кольору. Основні стилі наступні:

"-" – суцільна лінія;

"--" – штрихова лінія;

":" – пунктирна лінія;

"-." – штрих-пунктирна лінія;

"." – точкова лінія.

На одній діаграмі в разі потреби можна намалювати два та більше графіків. В наступному прикладі (Рис. 5.3) ми спочатку готуємо поточково два масиви – Y та Z – для представлення значень двох різних функцій, а потім виводимо їх. Принагідно звернемо на увагу на стилі представлення лінійних графіків. В дійсності їх існує величезна кількість, як и стилів представлення інших елементів графіку. Повний перелік з поясненням слід дивитися в поточній версії документації по пакету *matplotlib*.

```
x = [x*0.1 for x in range(100)]
x=np.array(x)
y = np.sin(x)
z=0.2*x-1
plt.plot(x, y, linestyle ='--', color='y')
plt.plot(x, z, lw=3, linestyle =':')
>>>>
```

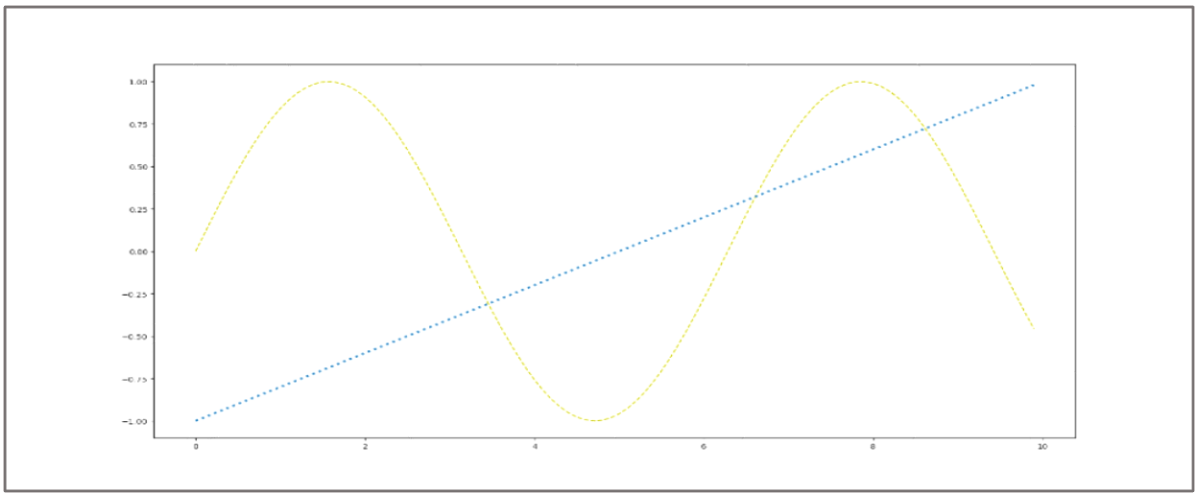


Рис. 5.3 – Два графіка на одній діаграмі

В ході аналізу даних може виникнути необхідність відобразити на одному малюнку відразу декілька графіків. Найпростіший спосіб зробити це полягає в використанні команди *plt.subplot()*. Ця функція приймає три параметри. Перші два задають кількість рядків та стовпчиків в розташуванні окремих діаграм, які треба відобразити на одному малюнку, а третій параметр – безпосередньо вказує, який з множини графіків буде описаний наступними командами пакету *matplotlib*. Розглянемо найпростіший приклад (Рис. 5.4), коли на одному малюнку треба відобразити два графіка та послідовно їх описати:

```
plt.subplot(1,2,1)
plt.plot(x, y, 'r--')
plt.subplot(1,2,2)
plt.plot(y, x, 'g*-')
>>>>
```

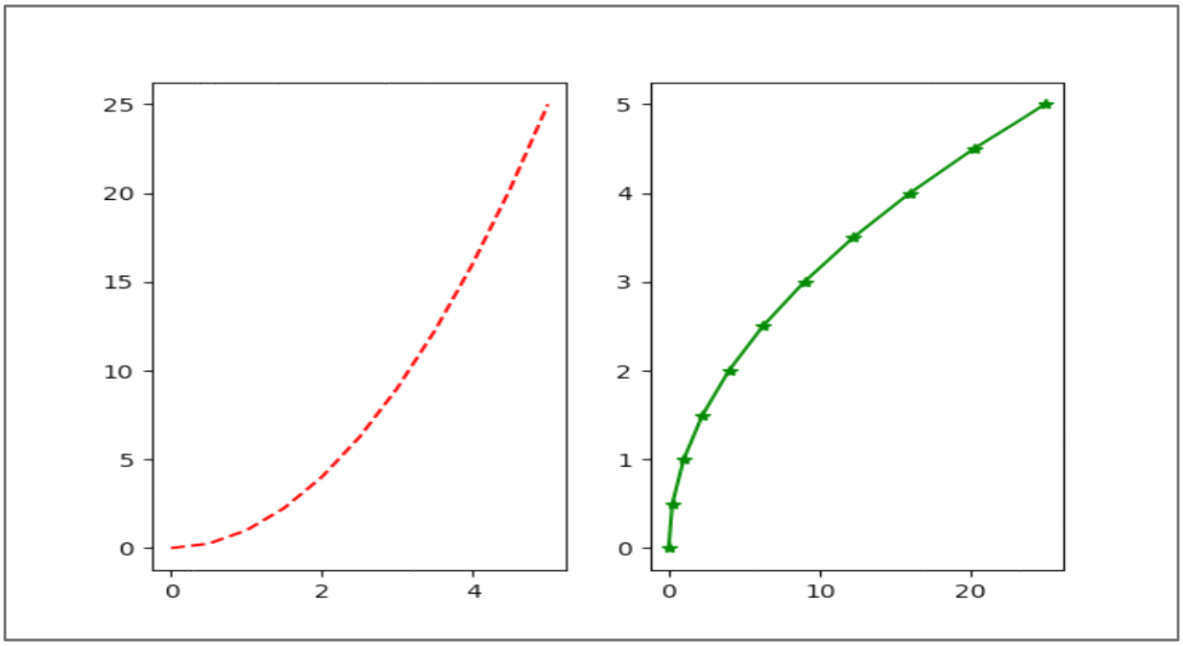


Рис.5.4 – Два графіка на одному малюнку

Перейдемо до ознайомлення з іншим, об’єктно-орієнтованим стилем інтерфейсу користувача для створення діаграм та графіків в *matplotlib*. Основна ідея об’єктно-орієнтованого програмування полягає в тому, щоб мати об’єкти, до яких можна застосовувати потрібні функції та дії. При застосуванні в *matplotlib* переваги такого підходу стають очевидними тоді, коли створюється більше ніж один малюнок або коли малюнок містить більше однієї діаграми.

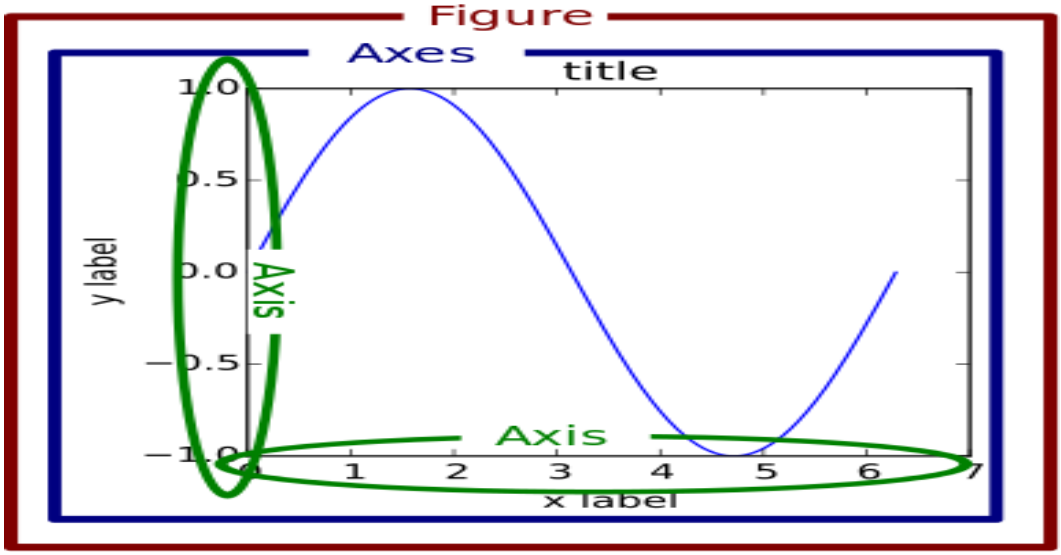


Рис. 5.5 – Елементи малюку в matplotlib

Щоб використовувати об’єктно-орієнтований API, ми починаємо так само, як і в попередньому випадку, з створення малюнку. Але замість створення нового глобального екземпляру малюнку (до якого ми зверталися за замочуванням) будемо зберігати посилання на кожний створений екземпляр об’єкту класу «малюнок» у змінній (традиційно її називають *fig* – від англійського *figure*), а з нього створюємо екземпляри класу *Axes*. (Примітка. *При вивченні matplotlib може виникнути складність з тлумаченням деяких термінів. В рамках пояснень різних сутностей пакету зустрічається два класи об’єктів з подібними назвами: Axes та Axis. Ця подібність здатна ввести в оману людину, що не відчуває тонкощів англійської мови, тим більш, що багато словників дають або однаковий переклад для обох термінів, або трактують один термін як форму множини іншого. Позатим, в matplotlib ці сутності кардинально відмінні. Якщо Axis традиційно позначає осі графіку чи діаграми, то Axes – позначає поле, область побудови, на якій відображається графік з усіма його компонентами, з осями включно. Рис.5.5 має дещо прояснити цю ситуацію. В україномовних документах ще не виробився сталий переклад терміну Axes у вказаному контексті, тому надалі ми будемо стисло перекладати цей термін як «діаграма» або «графік», маючи на увазі об’єкт, що відноситься до поля, на якому відображається візуалізація даних.*)

Наступний приклад виконує ті самі дії, що і наш перший приклад використання *matplotlib*, однак демонструє, як це робиться при використанні об’єктно-орієнтованого інтерфейса:

```
x = np.linspace(0, 5, 11)
y = x**2
fig = plt.figure()
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])
axes.plot(x, y, ‘r’)
axes.set_xlabel(‘Ось X’)
axes.set_ylabel(‘Ось Y’)
axes.set_title(‘Заголовок графіка’)
```

Метод *fig.add_axes()* створює графік в рамках малюнку, даючи можливість визначити його параметри, а саме – відповідно, відстань (в діапазоні від 0 до 1) від лівого та від нижнього краю малюнку, ширину та висоту графіку відносно відповідних характеристик малюнку. Вже знайомий нам метод *plot()* виконує безпосередню візуалізацію, інші три методи (в термінах ООП це не що інше, як сеттери для відповідних атрибутів об’єктів) задають підписи осей та назву

графіку. Крім вказаних, існує велика кількість інших сеттерів, як-то, наприклад *set_linestyle()* – для встановлення стилю ліній, *set_linewidth()* – для встановлення товщини линій графіку тощо. За інформацією про повний перелік атрибутів, якими можна керувати відповідними сеттерами, необхідно звернутись до документації пакету *matplotlib*.

Хоча для виконання опису бажаної візуалізації задіяно трохи більше коду, ніж і попередньому варіанті, перевага полягає в тому, що тепер ми отримуємо повний контроль над розміщенням діаграм на малюку, і можемо, наприклад, легко додати до нього більше однієї діаграми (Рис.5.6):

```
fig = plt.figure()
axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3])
# main figure
axes1.plot(x, y, ‘r’)
axes1.set_xlabel(‘Ось X’,fontsize=20)
axes1.set_ylabel(‘Ось Y’,fontsize=20)
axes1.set_title(‘Заголовок першого графіка’,fontsize=24)
# insert
axes2.plot(y, x, ‘g’)
axes2.set_xlabel(‘Ось Y’,fontsize=16)
axes2.set_ylabel(‘Ось X ‘,fontsize=16)
axes2.set_title(‘Заголовок другого графіка’,fontsize=20)
>>>>
```

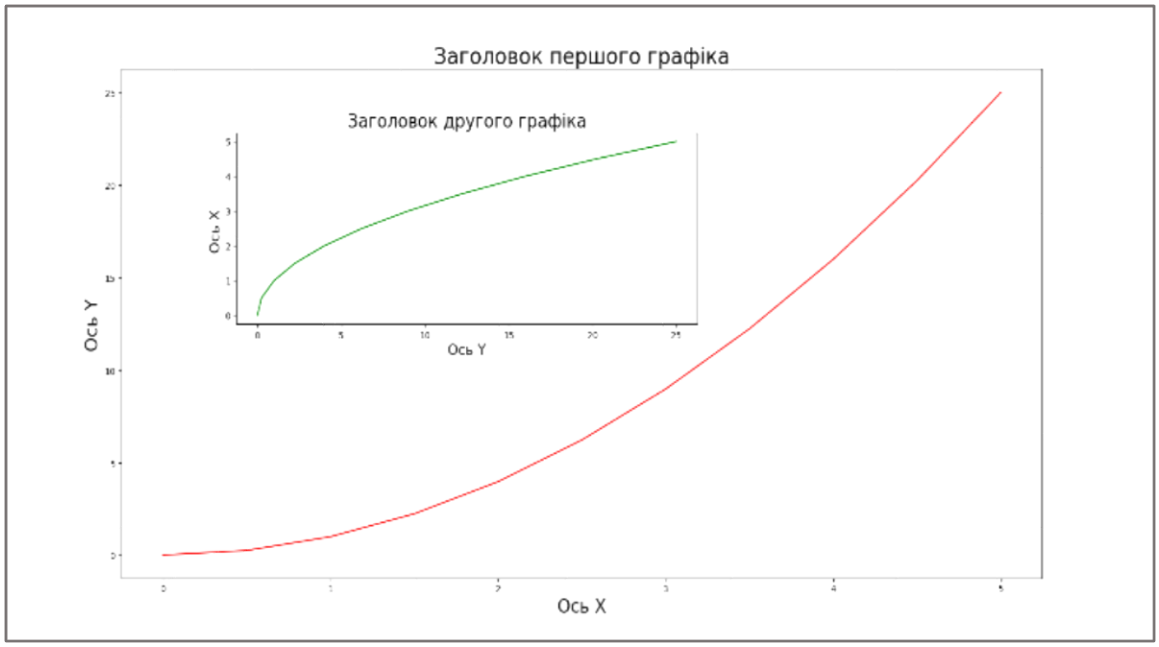


Рис. 5.6 – Діаграмі в іншій діаграмі

Якщо потреби чітко вказувати де саме мають бути розташовані наші діаграми в межах малюнку не виникає, можна скористатися одним з т.з. «менеджерів розташування» графіків, найбільш вживаним серед яких є менеджер *.subplots()*.(Примітка. *Ще одне термінологічне застереження. Зверніть увагу, що команда plt.subplot(), яка розгядалася раніше, і виклик менеджера plt.subplots() про який іде мова зараз – різні інструменти*).

В якості параметрів менеджер *subplots()* приймає два параметра – кількість рядків і кількість стовпчиків в розташуванні діаграм на малюнку (в разі створення малюнку з одною діаграмою параметри можна не задавати). В парі з методом *fig.tight_layout()* менеджер автоматично регулює положення діаграм на малюнку таким чином, щоб не виникало накладання діаграм одна на одну (Рис. 5.7).

```
fig, ax = plt.subplots(nrows=1, ncols=2)
for i in range(len(ax)):
 ax[i].plot(x, y, 'r')
 ax[i].set_xlabel('Ось X',fontsize=20)
 ax[i].set_ylabel('Ось Y',fontsize=20)
 ax[i].set_title('Заголовок {}-ого графіка'.format(i+1),fontsize=20)
fig.tight_layout()
>>>>
```

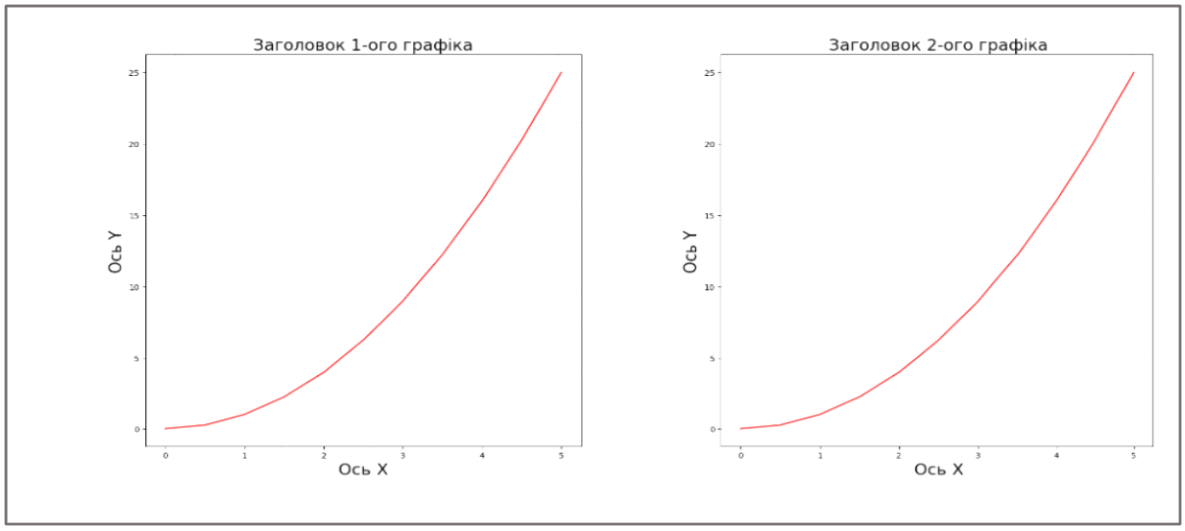


Рис. 5.7. – Приклад використання менеджера розташування

Для надання діаграмі більш інформативного дизайну можна явним чином керувати відображенням позначок на її осях (Рис.5.8). Для цього використовуються методи *set_xticks* і *set_yticks*, які в якості параметрів приймають списки значень. А методи *set_xticklabels* і *set_yticklabels* дозволяють замінити значення на такі, які містять власні, бажані досліднику, значення та/або відформатувати підписи бажаним чином:

```
fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(x, x**2, x, x**3, lw=2)
ax.set_xticks([1, 2, 3, 4, 5])
ax.set_xticklabels(['A', 'B', 'C', 'D', 'E'], fontsize=25)
yticks = [0, 50, 100, 150]
ax.set_yticks(yticks)
ax.set_yticklabels(["$%.1f$" % y for y in yticks], fontsize=18)
>>>>
```

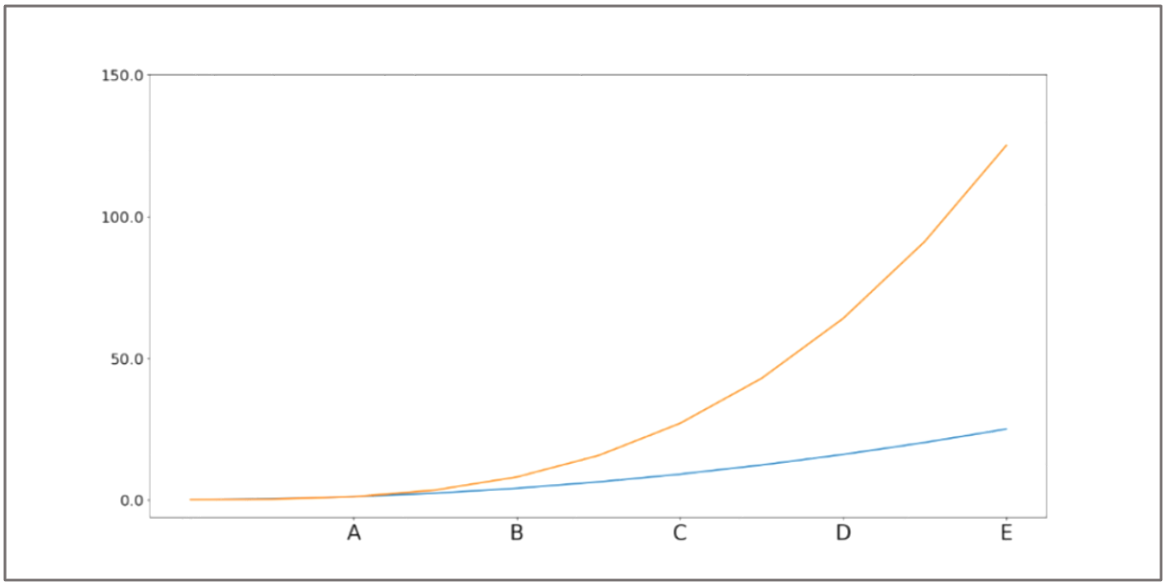


Рис.5.8 Приклад керування відображенням позначкок на осях графіку

За допомогою методу grid() в об'єкті-осі можна відображати лінії сітки на діаграмі, що може суттєво покращити її сприйняття людиною (Рис. 5.9). Можна також налаштувати зовнішній вигляд ліній сітки, використовуючи ті самі ключові аргументи, що й при відображенні самого графіку:

```
fig, ax = plt.subplots(1, 2, figsize=(10,3))
ax [0].plot(x, x**2, x, x**3, lw=2)
ax [0].grid(True)
ax [1].plot(x, x**2, x, x**3, lw=2)
ax [1].grid(color='g', alpha=0.5, ls='--', linewidth=2)
>>>>
```

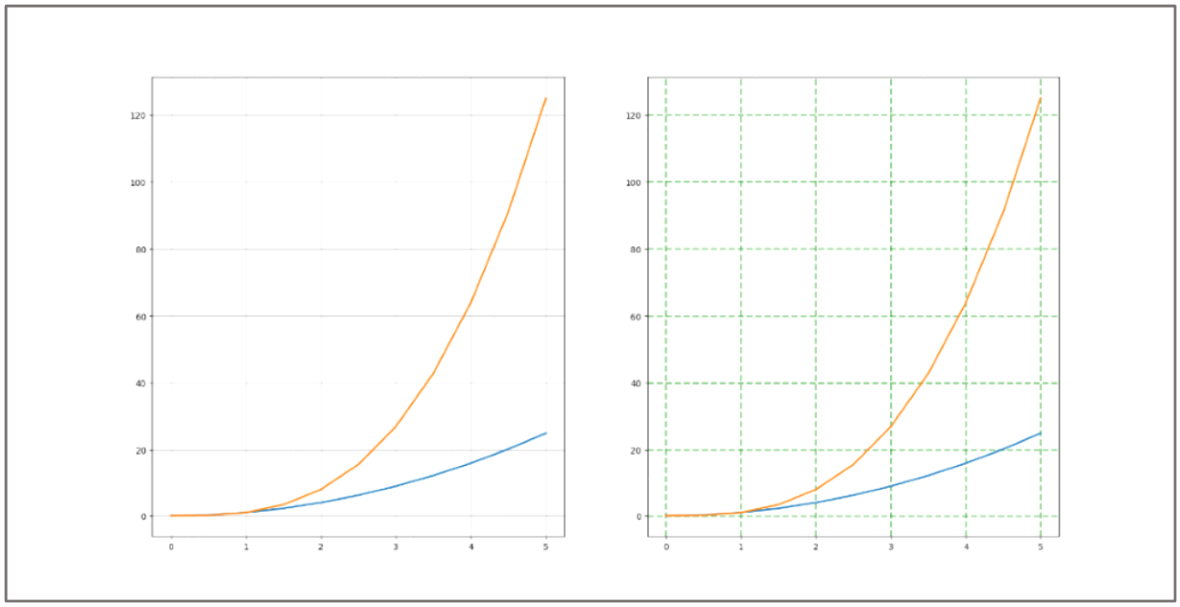


Рис. 5.9 – Приклад відображення сіток на діагамах

При спробах намалювати на одній діаграмі два графіки, які суттєво відрізняються за масштабам, *matplotlib* за замовчуванням буде обирати діапазон по осі Y таким, щоб на ньому відобразилися і найменші і найбільші значення для кожного з графіків. Це може привести до того, що зміни значень даних на графіку з меншими значеннями будуть практично непомітні.

Для демонстрації згенеруємо спеціальним чином два штучні набори даних, та спробуємо їх зобразити спочатку в єдиному масштабі (діаграма ax[0] – верхня на Рис. 5.10), а потім – з застосуванням для кожного з набору свого масштабу (діаграма ax[1] – нижня на Рис. 5.10). Зрозуміло, що верхній графік не надає досліднику ніякої інформації, з якої можливо було-б зробити висновки про наявність певного зв'язку між значеннями з наборів. В той жн час, нижній графік такий зв'язок між значеннями наборів показує чітко та однозначно.

```
np.random.seed(54647)
x= np.arange(0.01, 10.0, 0.01)
y1 = np.exp(t)
y2= np.sin(2 * np.pi * t)
y1=y1+y2*[1000]
fig, ax = plt.subplots(2,1)
ax[0].plot(x, y1, lw=2, color="blue")
ax[0].plot(x, y2, lw=2, color="red")

ax[1].plot(x, y1, lw=2, color="blue")
```

```
ax2 = ax[1].twinx()
ax2.plot(x, y2, lw=2, color="red")
>>>>
```

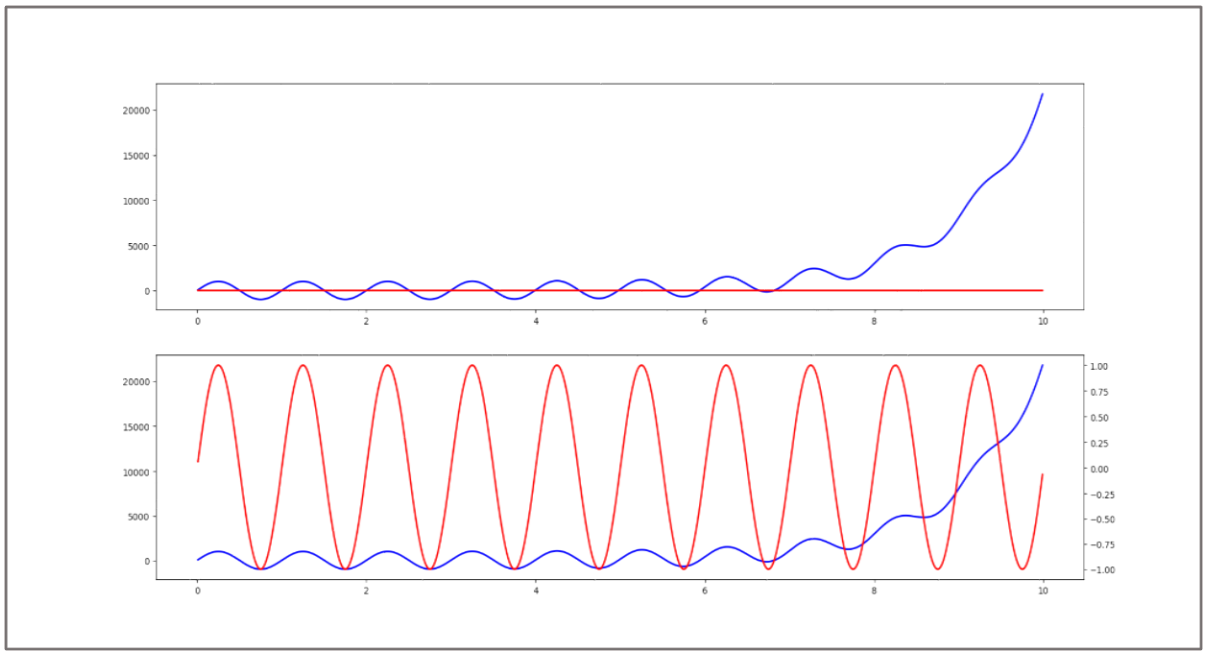


Рис. 5.10 – Приклад графіків з різними масштабами значень по осі Y

5.2. Лінійні діаграми при роботі з pandas

Візуалізація даних у вигляді лінійних діаграм є надзвичайно корисним інструментом аналізу даних, що представлені у вигляді часових рядів, особливо на початковому етапі дослідження. При такому дослідженні дані часто представлені в вигляді *pandas-DataFrame*. Тому не дивно, що розробники *pandas* потурбувалися про надання дослідникам набору методів та функцій, що спрощують побудову діаграм. Хоча цей набор інструментів є по суті надбудовою над *matplotlib* і відповідно дає можливість використання всіх можливостей останнього, в деяких випадках засоби візуалізації *pandas* дозволяють зробити відображення інформації простіше, ніж в самому *matplotlib*. А в деяких випадках – безпосередньо поєднати можливості *matplotlib* з можливостями спеціалізованих пакетів обробки та аналізу даних, спросивши не тільки процес візуалізації, але й сам процес дослідження даних.

В прикладі нижче показано використання лінійного графіка для візуалізації часових рядів. Було зібрано інформацію про кількість щоденних відвідувачів двох сайтів на протязі чотирьох років. Інформація представлена у вигляді *pandas-*

DataFrame. Скласти первинне уявлення про *DataFrame* можна за допомогою відомого метода *df.head()*, результати роботи якого доступні в консолі (Рис. 5.11):

```
Консоль 1/A
In [59]: df.head()
Out[59]:
            Сайт А.  Сайт В.
2017-01-01    40813    38374
2017-01-02    40558    37656
2017-01-03    40404    38454
2017-01-04    41201    37986
2017-01-05    39907    37418

In [60]:
```

Рис. 5.11 – Відображення на консолі перших значень DataFrame, що досліджується

Для візуалізації часових рядів, що представляють собою стовпчики *DataFrame*, застосуємо вбудований в *pandas* метод *df.plot()* (Рис. 5.12).

```
import pandas as pd
df.plot()
>>>>
```

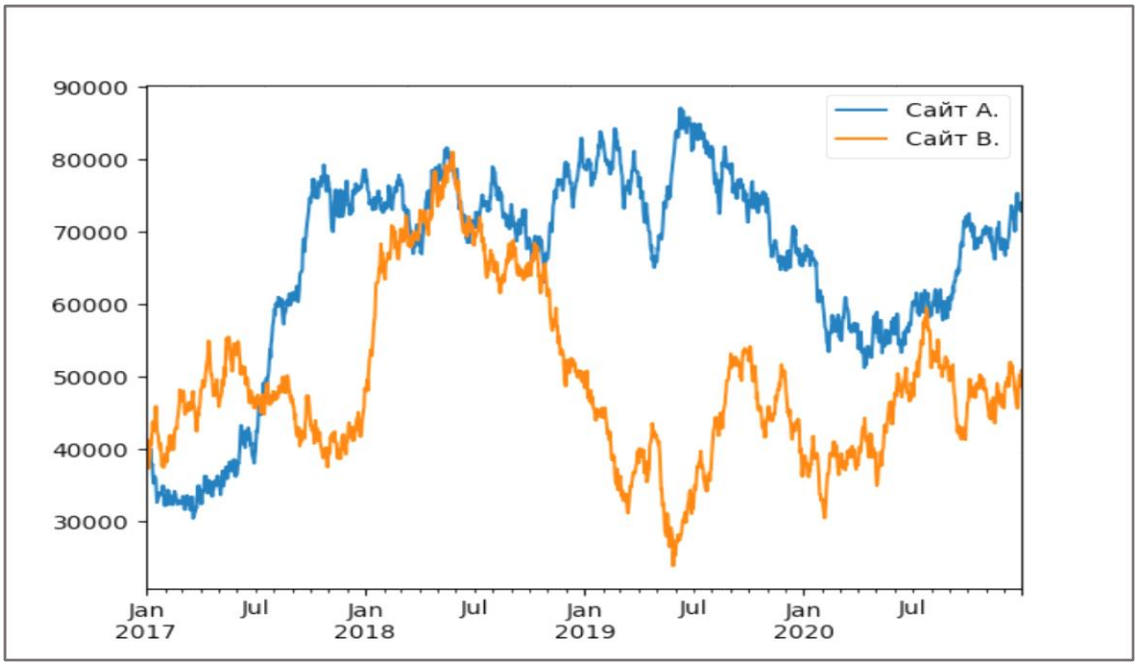


Рис. 5.12 – Візуалізація даних з DataFrame, що представляє інформацію про відвідування сайтів

Користуючись індексом *DataFrame* цей метод самостійно визначає тип значень, які відкладаються по осі X. В нашому випадку індекс представлений у вигляді дат. А стовпчики *DataFrame* використовуються в якості джерела значень

часових рядів, причому назви стовпчиків відображаються на легенді, яка супроводжує діаграму.

В разі необхідності побудувати графік часових рядів з використання деякого іншого стовпика з *DataFrame* в якості значень, що задаються по осі X, це можна зробити, безпосередньо задавши назву цього стовпчика як значення параметру x вказаної функції. В наступному прикладі (Рис. 5.13) припустимо, що ми бажаємо побудувати графік, вказавши по осі X т.з. «тіки», тобто номери днів, яким відповідають значення, що зберігаються в *DataFrame*. Початковий день – 1 січня 2017 року буде позначено значенням 1, 2 січня – значенням 2 тощо. Для цього створимо в *DataFrame* новий стовпчик, куди занесемо відповідні номери:

```
df['Номер дня спостереження']=pd.Series(np.arange(1,len(df)+1), index=df.index)
df.plot(x='Номер дня спостереження',y=['Сайт A.','Сайт B.'])
plt.xlabel('Номер дня спостереження',fontsize=20)
>>>>
```

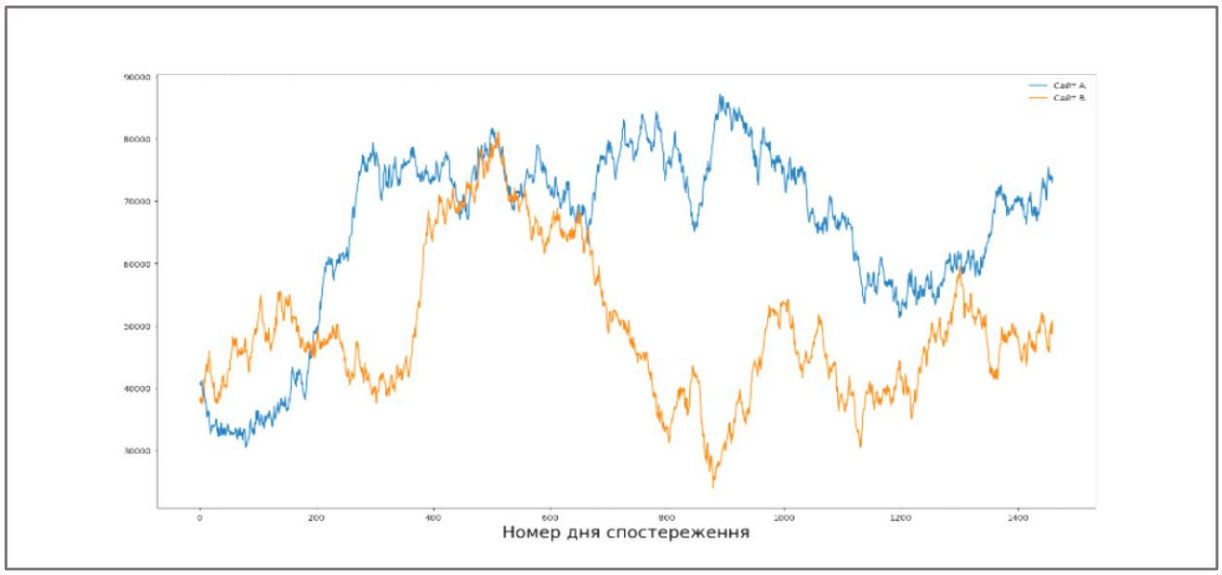


Рис. 5.13 – Відображення часового ряду в залежності від номера дня спостереження

Принагідно ми бачимо, як задаючи значення параметру у можна керувати відображенням лише деяких обраних стовпчиків *DataFrame*.

Однак часто буває так, що дані що отримані дослідником характеризуються саме номером відліку. Додатково надається інформація про мітку часу тільки першого відліку, а от графік бажано представити, використовуючи по осі X саме мітки часу (Рис. 5.14). Покажемо, як така задача може бути досить просто реалізована з використанням бібліотеки *datetime*, яка була розглянута в главі 3.

```
import matplotlib.pyplot as plt
import datetime
import numpy as np
import pandas as pd
df = pd.DataFrame({'date': np.array([datetime.datetime(2020, 1, i+1)
 for i in range(12)]),
 'sales': [6, 4, 4, 7, 8, 9, 14, 15, 9, 8, 4, 13]})
plt.plot(df.date, df.sales, linewidth=3)
plt.title('Кількість звернень в службу технічної підтримки')
plt.xlabel('Дата')
plt.ylabel('Кількість зафіксованих випадків')
>>>>
```

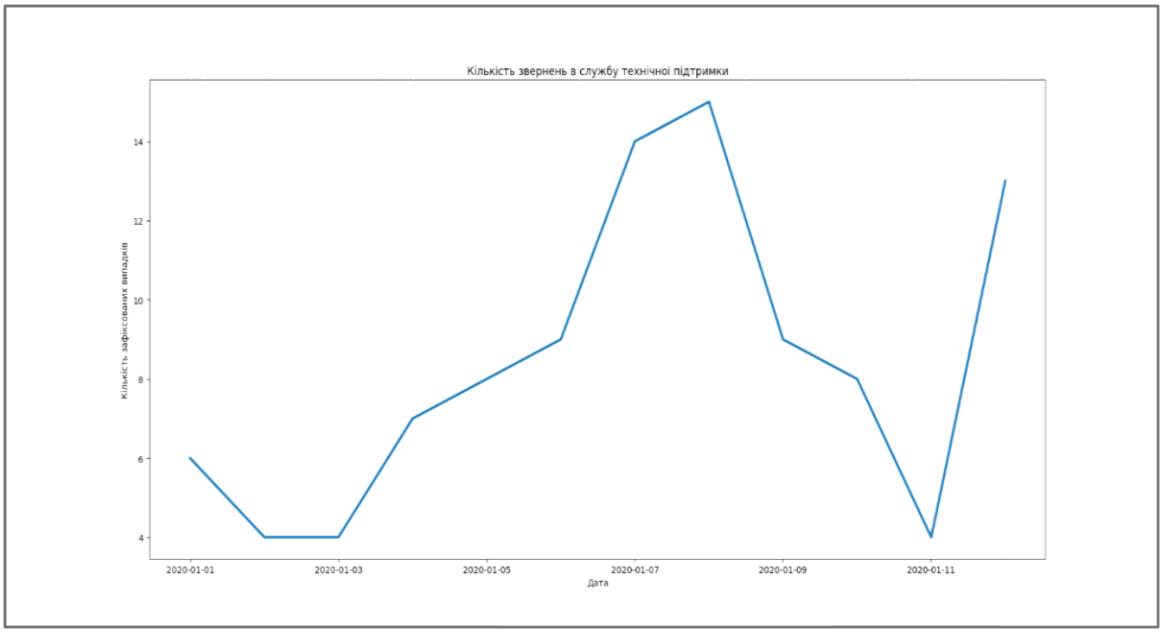


Рис. 5.14 – Побудова графіку за мітками часу, що розраховуються з допомогою функцій з бібліотеки *datetime*

Продемонструємо роботу с часовими рядами наступним прикладом.

В файлі *Kiyv-T-min.csv* зібрана інформація про середньодобову температуру в Києві з 1.01.2011 року по 31.12.2020. Прочитаємо цю інформацію (Рис. 5.15.).

```
import pandas as pd
import matplotlib.pyplot as plt
t_min = pd.read_csv('Kiyv-T-min.csv', header=0, delimiter=';')
t_min['Date'] =pd.to_datetime(t_min['Date'], format='%d.%m.%Y')
t_min.set_index('Date',drop=True, inplace=True)
t_min.head()
>>>>
```

```
Консоль 1/A
    ...: t_min.head()
Out[17]:
            T-Av
Date
2011-01-01  -3.0
2011-01-02  -0.4
2011-01-03  -3.0
2011-01-04  -4.3
2011-01-05  -6.8
```

Рис. 5.15 Відображення початкових даних файлу значень температур

Об’єкт *t_min* представляє собою найпростіший *DataFrame* з одним стовпчиком та індексом-датою, який можна вивести для візульного відображення (Рис. 5.16):

```
t_min.plot()
>>>>
```

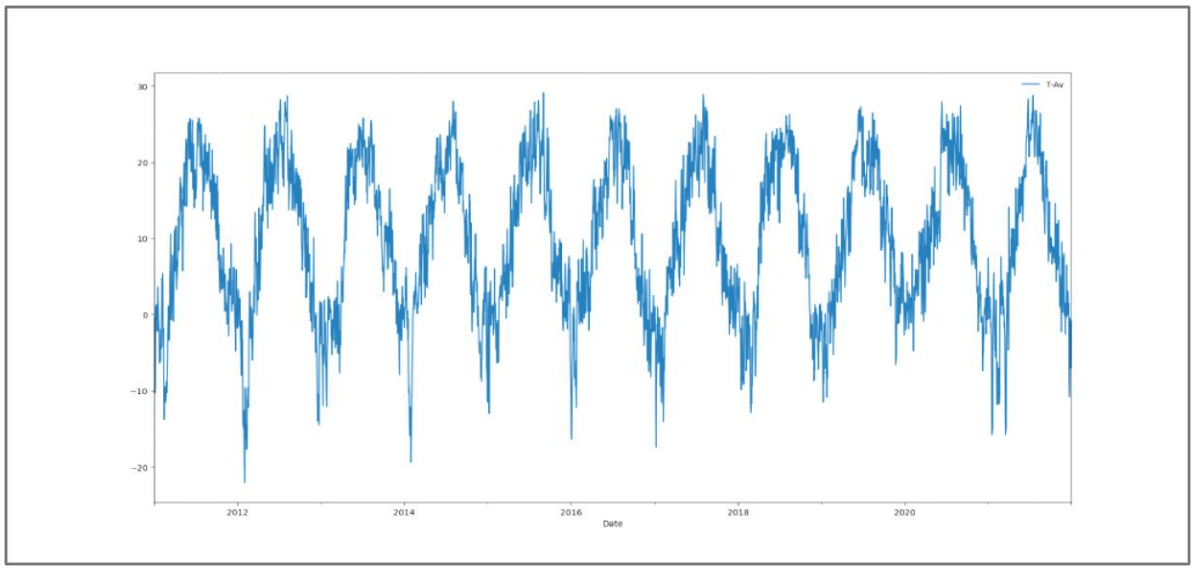


Рис. 5.16 – Відображення інформації з файлу температур за допомогою часового ряду

Оскільки в даному разі нам точно відомо, що це дані сезонні, ми можемо спробувати виконати порівняння температур поденно. Для цього виведемо дані кожного з років окремим графіком. Наведений нижче приклад показує, як це зробити.

Спочатку створимо робочий об’єкт-*DataFrame*, куди перенесемо значення температур так, щоб кожен стовпчик цього об’єкту представляв дані за один окремий рік. Для цього використаємо метод *groupby()* з бібліотеки *pandas*. Таким чином, в нашому *DataFrame* має бути 10 стовпиків (Рис. 5.17).

```
groups = t_min.groupby(pd.Grouper(freq='A'))
years = pd.DataFrame()
for name, group in groups:
 print(name.year)
 years[name.year] = group['T-Av'].values[:365]
years.head()
>>>>
```

Консоль 1/A

```
In [71]: years.head()
Out[71]:
   2011  2012  2013  2014  2015  2016  2017  2018  2019  2020
0  -3.0   0.5   1.2  -1.7  -3.5 -10.8  -0.1   1.5  -0.7   1.4
1  -0.4  -1.8  -2.7  -0.1   1.2 -15.9   1.0   2.1   0.4  -0.3
2  -3.0   2.2  -0.4   0.7   2.9 -16.3  -0.7   2.7  -0.5   0.5
3  -4.3   2.6   1.6   1.0   1.1 -16.4  -1.5   3.6  -2.0   1.0
4  -6.8   3.2   0.7   0.9  -0.6 -12.5  -1.3   2.4  -4.2   1.4
```

Рис. 5.17 – Початкові рядки DataFrame, після переофрмаутвання даних по роках спостереження

Тепер можна виводити кожний стовпчик окремою командою *matplotlib*, але ми покажемо (Рис.5.18), як засоби *pandas* спрощують цей процес.

```
years.plot()
>>>>
```

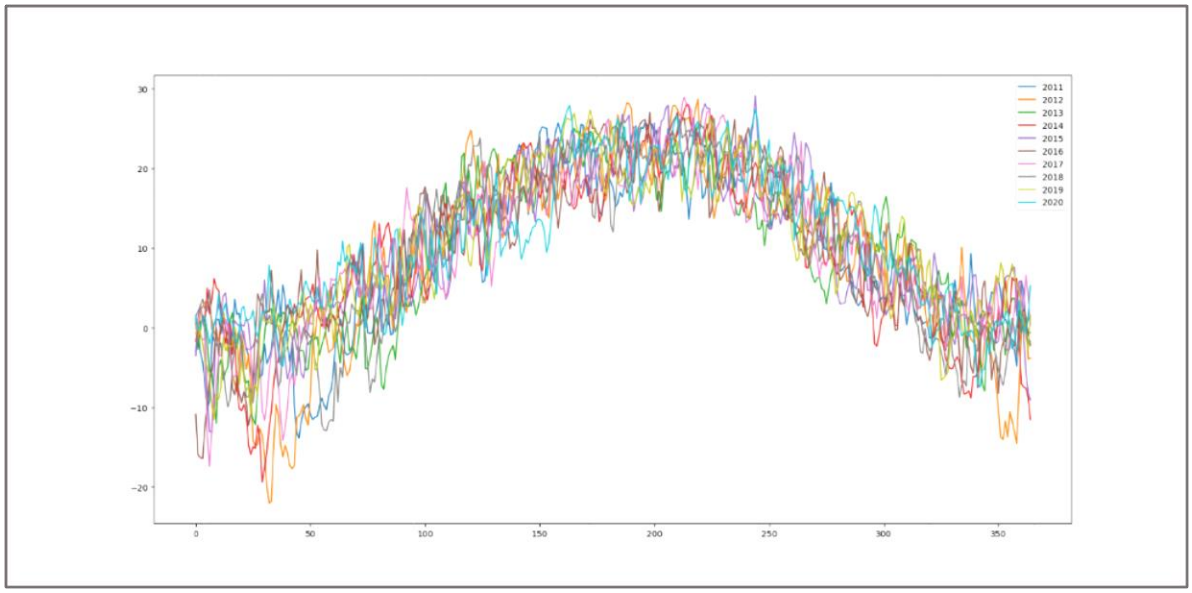


Рис. 5.18 – Графічне представлення інфорації залежності температури від дня спостереження

На графіку Рис. 5.18 видно, як температури змінювалися в кожному році в залежності від номеру дня в році. На наступному графіку (Рис. 5.19) дані по рокам

відображаються кожен на окремому графіку з збереженням відповідності номеру дня в році. Останній цикл призначений для впорядкування відображення легенди графіку.

```
years.plot(subplots=True)
for ax in plt.gcf().axes:
 ax.legend(loc=1)
 ax.set_xlim([0, 364])
```

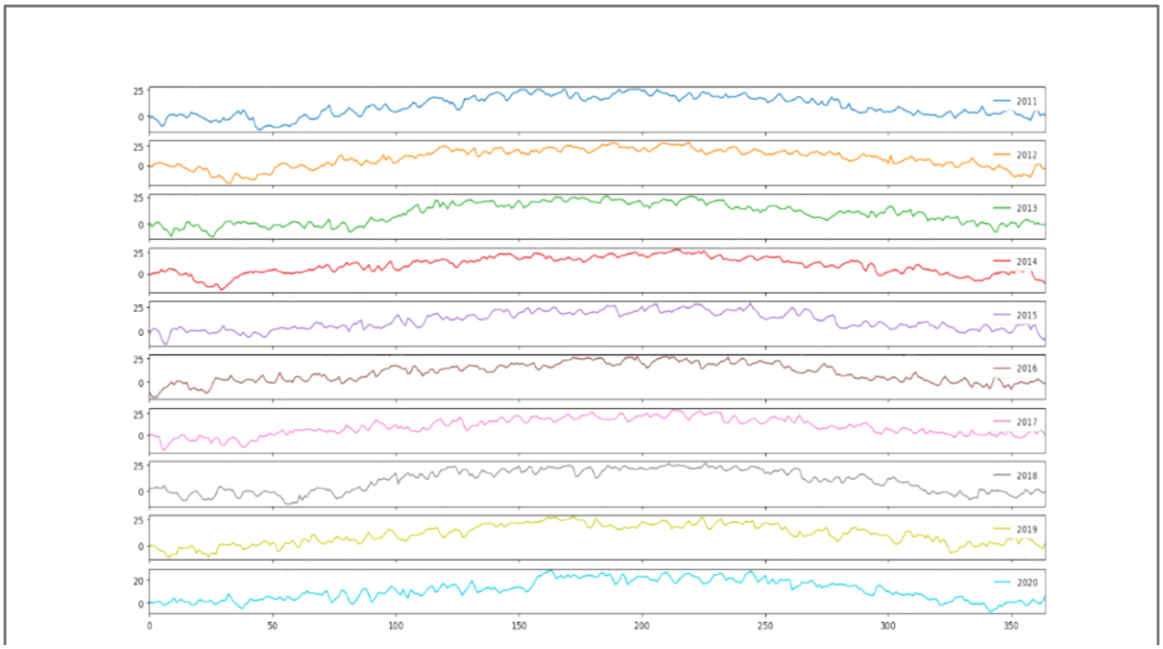


Рис. 5.19 – Розділене відображення значень часового ряду, по рокам та по дням спостереження

5.3. Стовпчикова гістограма

Стовпчикова гістограма зазвичай використовується для представлення відносних величин для кількох категорій. Ось X представляє деякі номінальні значення. Навіть якщо вони будуть задані в числовій формі, з точки зору *matplotlib* вони все одно будуть розглядатися як номінальні. Оскільки для номінальних значень неможливо задати ніякого відношення між елементами окрім тотожності або її заперечення, в будь якому випадку *matplotlib* розміщує об'єкти по осі X рівномірно. Значення по осі Y можуть інтерпретуватися як кількість даних кожної з категорій та ввідображаються у вигляді смуги від базової лінії до відповідного рівня на осі Y. Для відображення стовпчикової діаграми використовується функція *plt.bar()*. Параметр *width* цієї функції задає долю ширини відстані між двома послідовними значеннями по осі X, яку займає сам стовпчик. Використання методу *.xticks()* з параметром *rotation* дозволяє в разі потреби повернути підписи осей на відповідний кут (Рис. 5.20.).

```
np.random.seed(12102022)
x = [‘червоний’,’помаранчевий’,’жовтий’,’зелений’,
 ‘блакитний’,’синій’,’фіолетовий’]
y = np.random.randint(1, 20, size = 7)
plt.bar(x, y, width=0.5,color=‘blue’)
plt.xticks(rotation = 15)
```

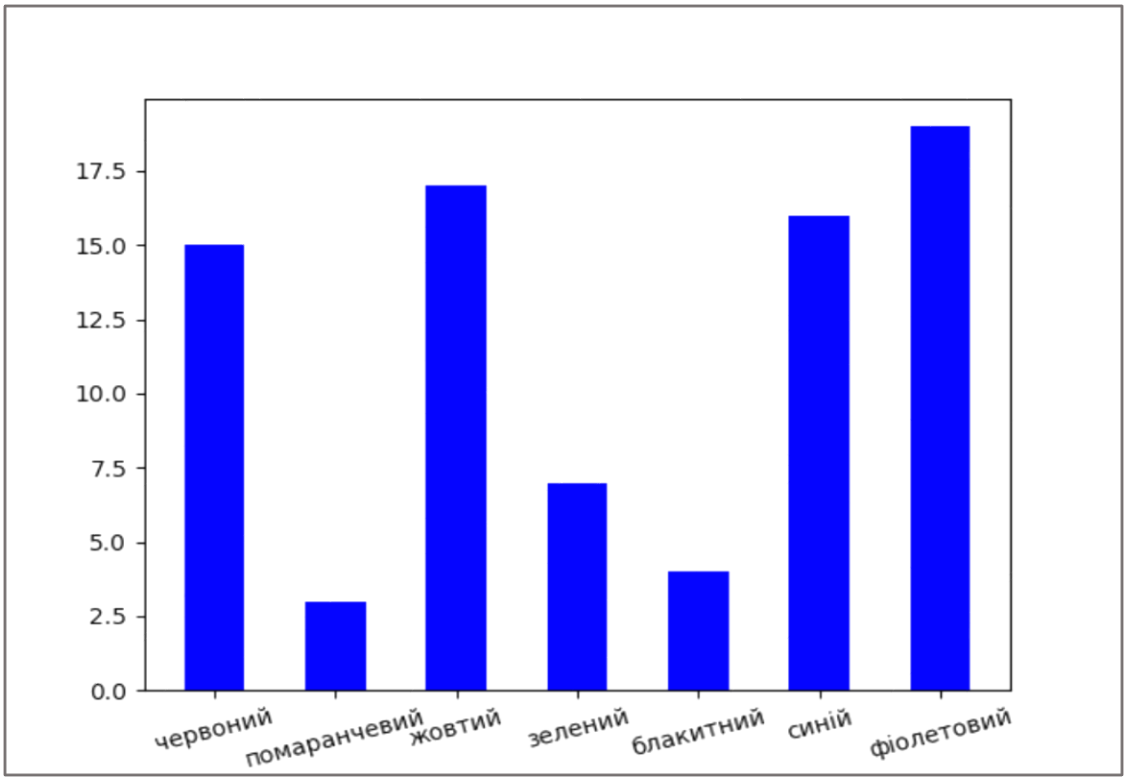


Рис. 5.20 – Форматування підписів при роботі з стовпчиковою діаграмою

Якщо підготувати спеціальний масив ідентифікаторів кольорів та вказати цей масив в якості значення параметру *color*, можна розфарбувати стовпчики діаграми в бажані кольори (Рис.5.21.)

```
color_rectangle = [‘r’,’orange’,’y’,’g’,’c’,’b’,’m’]
plt.bar(x, y, width=0.75,color=color_rectangle)
plt.xticks(rotation = 15)
>>>>
```

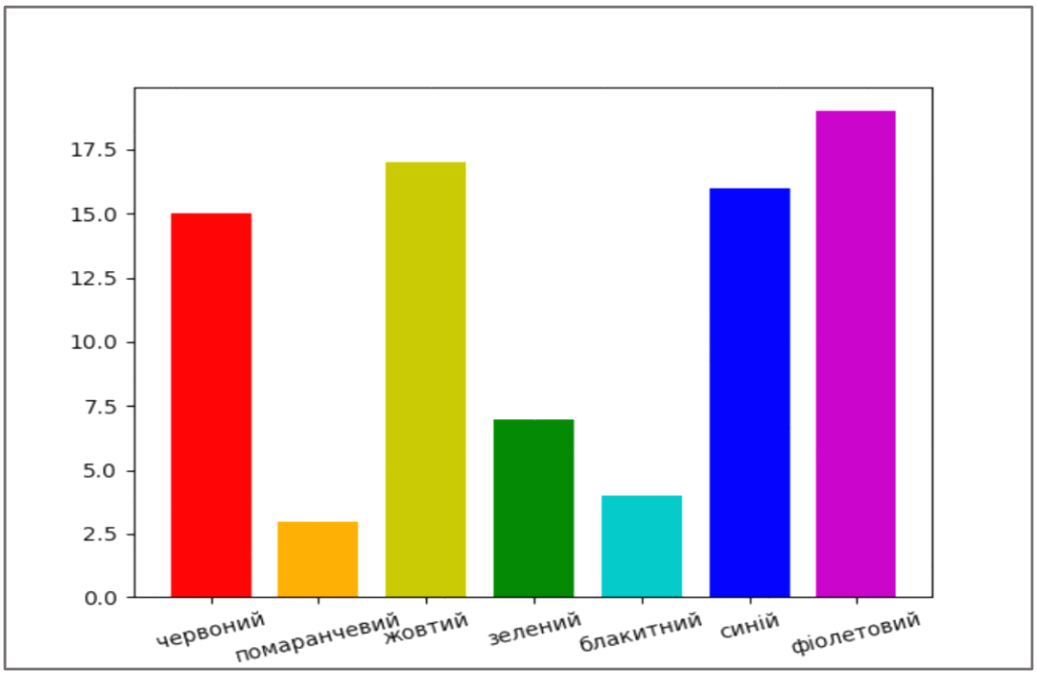


Рис. 5.21 – Використання кольорів при побудові стовпчикових діаграм

Якщо дані, які аналізуються, представляють собою багатовимірну величину, тобто кожному значенню по осі X відповідають декілька значень по осі Y, побудова графіку може виглядати як показоно на Рис. 5.22. В цьому прикладі згаданий вище параметр *width*, а також позиція відображення кожного стовпчика по осі X обирається таким чином, щоб не відбулася перекриття стовпчиків при їх візуалізації.

5.4. Діаграма розкиду

Діаграма розкиду (або «діаграма розсіювання») зазвичай використовується для узагальнення зв’язку між двома парними вибірками даних, тобто такими вибірками, в яких кожне значення в одній вибірці семантично пов’язане з одним і тільки одним значенням в іншій вибірці. Такі парні вибірки виникають тоді, коли для кожного спостереження було зареєстровано два показники, наприклад

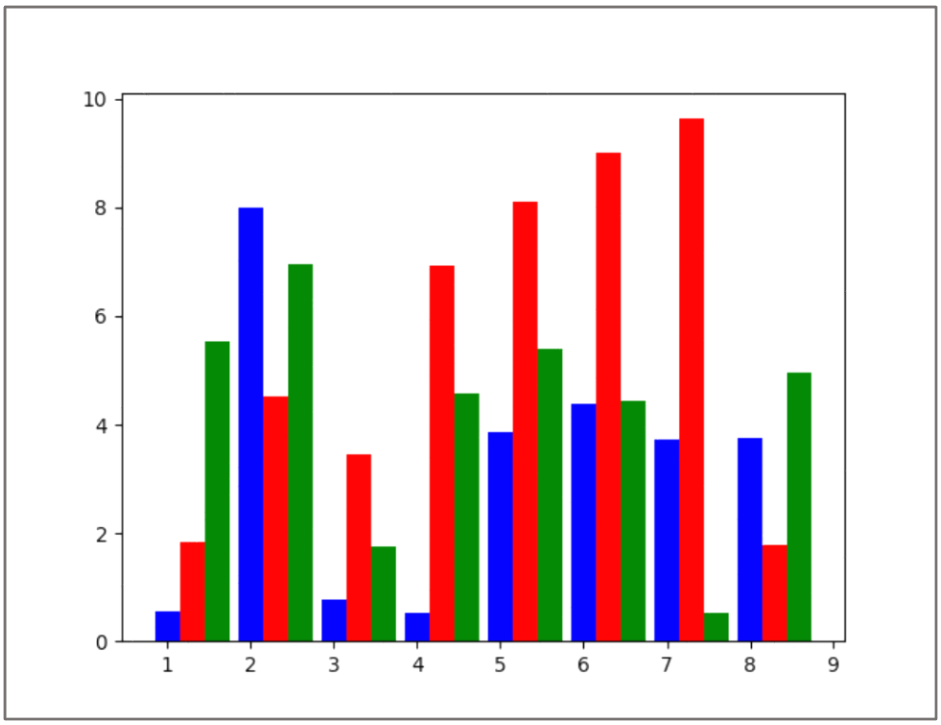


Рис. 5.22 – Приклад сумісного відображення декалькох стовпчикових діаграм

вага та зріст людини; кількість пакетів, що пройшли мережею за певний проміжок часу та об'єм інформації, шо пройшов мережею за той самий проміжок; показники систолічного та діастолічного тиску в кровоносній системі людини тощо. Графіки такого типу допомагають зрозуміти взаємозв'язки між даними з вибірок.

В діаграмі розкиду по осі X відображаються значення спостереження для першої вибірки, а на осі Y – значення спостереження для другої вибірки. Таким чином, кожній точці на цьому графіку відповідає одне спостереження. Діаграми розкиду можна створити, викликавши функцію *scatter()* і передавши в якості перших двох параметрів два масиви вибіркових даних.

В прикладі на Рис. 5.23 візалізується інформація про ріст та вагу деякої групу людей:

```
plt.scatter(height, weight)
plt.xlim([130,200])
plt.xlabel('Зріст (см)',fontsize=20)
plt.ylabel('Вага (кг)',fontsize=20)
>>>>
```

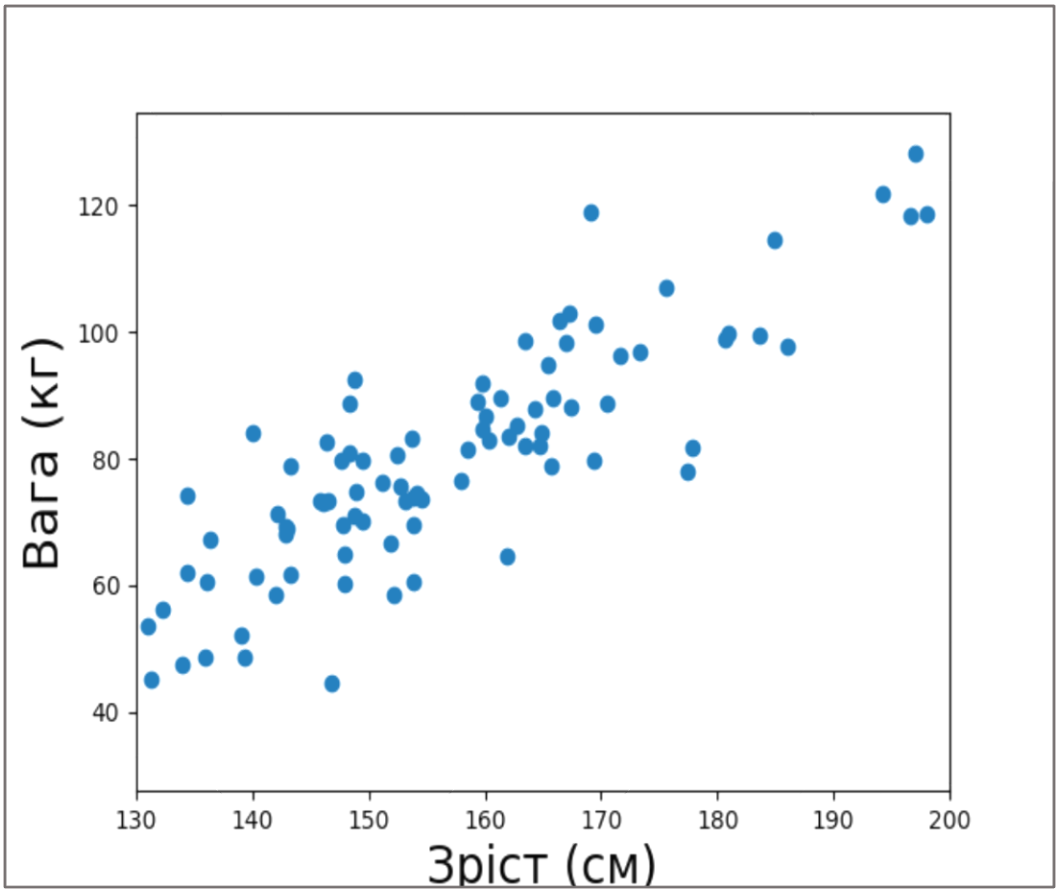


Рис. 5.23 – Приклад діаграми розкиду

В прикладі (Рис. 5.24) генеруються три штучні вибірки за різними законами розподілу. На графіку візуалізується залежність між першою парою вибірок (x та y), та між другою парою вибірок (x та z). Для підвищення інформативності для різних пар застосовуються різні кольори та форми точок, які відображаються на графіку.

```
x = np.random.randn(100)
y = np.random.randn(100)
z = np.random.uniform(0,1,100)
plt.scatter(x, y, marker='o')
plt.scatter(x, z, marker='*')
>>>>
```

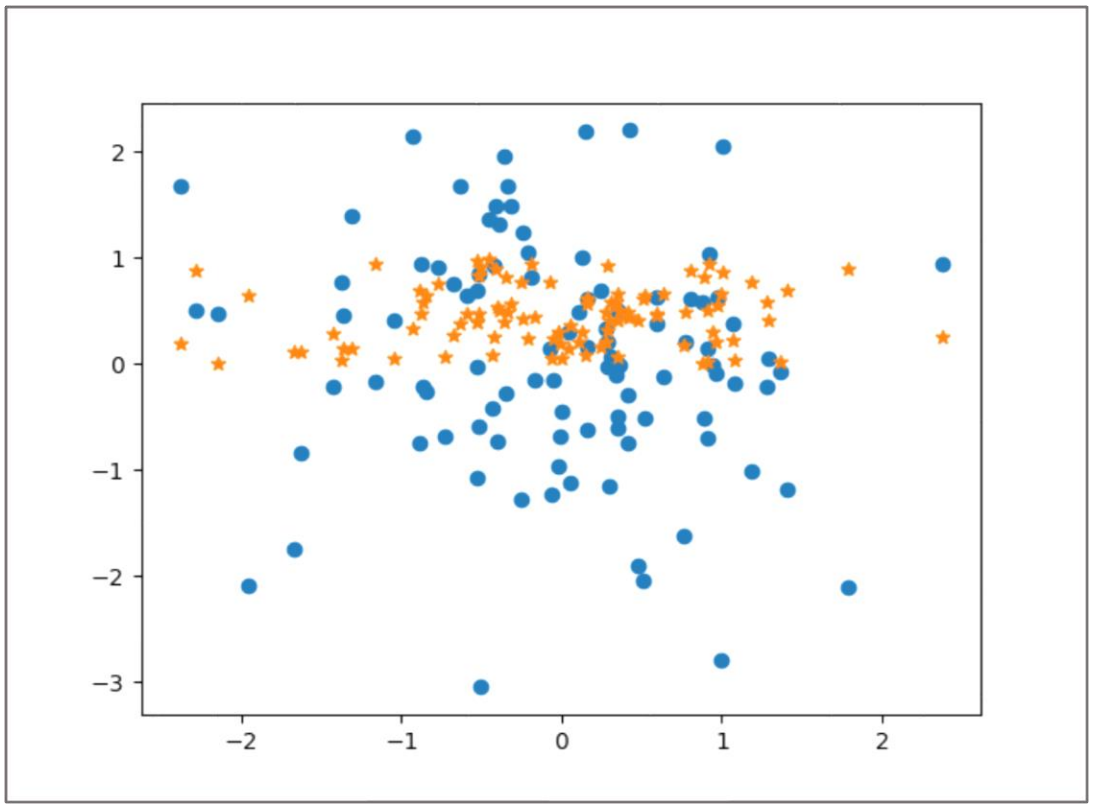


Рис. 5.24 – Відображення двох діаграм ризкиду на одному малюнку

Цікавою особливістю діаграм розкиду є можливість керування не тільки формою, але й кольором та/або розміром кожної точки графіку окремо. Отриману таким чином діаграму часто називають «бульбашковою» (Рис. 5.25). Її можна навіть уявити як діаграму з трьома змінними. У бульбашковій діаграмі значення третьої змінної представлено розміром бульбашки, що оточує точку даних, звідси й походить відповідна назва.

```
x = np.random.randn(100)
y = np.random.randn(100)
colors = np.random.rand(100)
sizes = 1000 * np.random.rand(100)
plt.scatter(x, y, c=colors, s=sizes, alpha=0.8)
>>>>
```

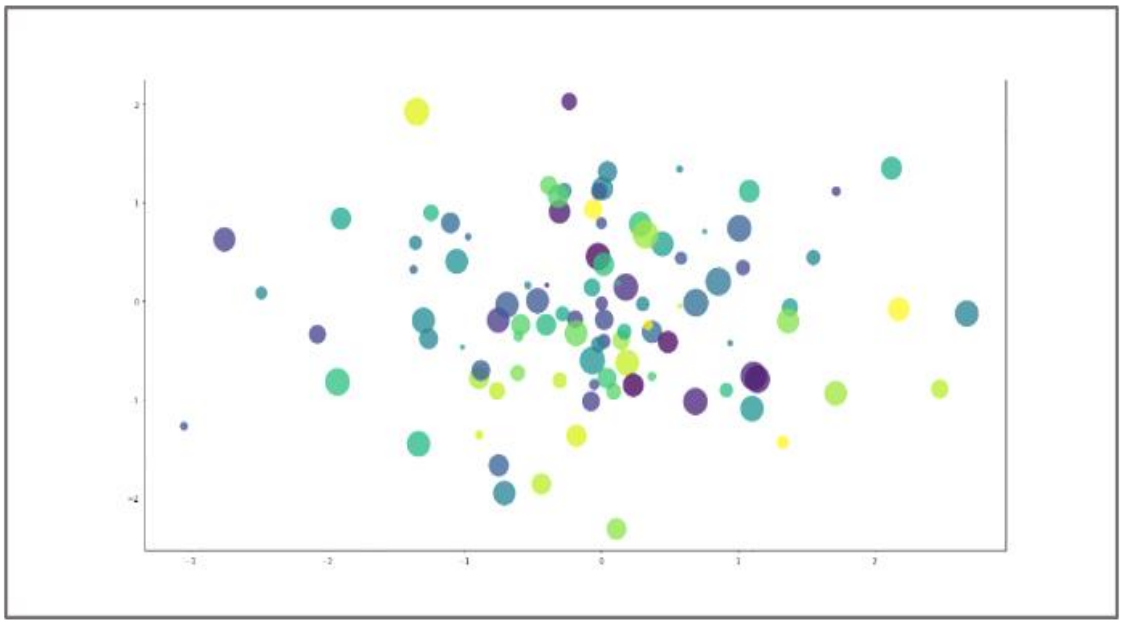


Рис. 5.25 – Приклад побудови бульбашкової діаграми

5.5. «Бокс-плот» діаграма, або «ящик з вусами»

Така назва походить від визначення зовнішньої схожості діаграми на «коробочки» та – при певній фантазії – «вуса», що від них відходять і зазвичай використовується для візаулізації узагальненої інформації про розподіл вибірки даних. Діаграма допомагає побачити середнє арифметичне значення вибірки, асиметрію, розкид значень вибірки та отримати деяку інформацію про викиди – елементи вибірки, поява яких надзвичайно рідкісна подія. Окрім вказаної назви,

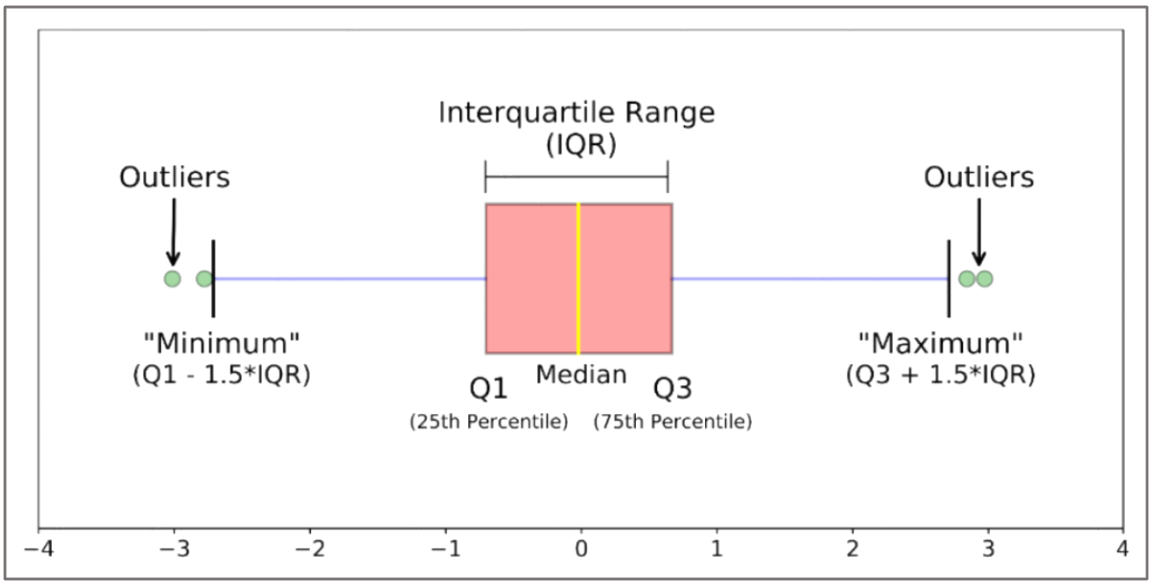


Рис. 5.26 – Елементи бокс-плот діаграми

часто застосовують і деякі синонімічні назви, як-то «діаграма розмаху», «коробковий графік» тощо (Рис. 5.26).

На бокс-плот діаграмі вісь X використовується для умовного представлення вибірки даних. Вісь ординат представляє значення спостереження. Прямокутник (з якого нас в дійсності цікавлять тільки його малі сторони, але для підвищення інформативності на графіку малюють саме прямокутник) позначає кордони, в межах яких знаходяться 50% значень з набору даних, починаючи від значення, що відповідає 25%-процентилю та закінчуючи значенням, що відповідає 75%-процентилю. Вказаний діапазон також називають міжквартельним розмахам, або IQR (InterQuartile Range). Медіана, або 50%-процентиль, малюється лінією, яка, зрозуміло, проходить десь в середині прямокутника.

Лінії, які називаються вусами, проходять від обох кінців прямокутника, і визначаються відносно 25%- та 75%-процентилів. Вони призначені для демонстрації діапазону значень у вбірці, які вважаються «звичайними». Спостереження за межами вусів можуть представляти собою викиди та зображуються маленькими колами.

«Ящик з вусами» будується за допомогою функції *plt.boxplot(),* яка в якості параметру приймає значення вибірки. В разі, коли на одному графіку виникає потреба одночасного представлення даних декількох вибірок, об’єкти-вибірки мають об’єднуватися в список, який і передається функції в якості параметру. Слід зазначити, що саме можливість такого одночасного відображення декількох вибірок на одному малюнку робить цей тип діаграм популярним серед дослідників. Вона дає можливість досить легко не тільки зрозуміти особливості закону розподілу кожної вибірки, але й провести порівняльний аналіз різних вибірок між собою. На Рис. 5.27 представлено бокс-плот діаграма, що створена для порівняльного аналізу набраних рейтингових балів курсантами різних учбових груп.

```
plt.boxplot([k21,k22,k23,k24])
plt.xticks([1, 2, 3, 4], ["Група К21", "Група К22", "Група К23", "Група К24"], fontsize=20)
plt.ylabel(‘Рейтингові бали’,fontsize=20)
```

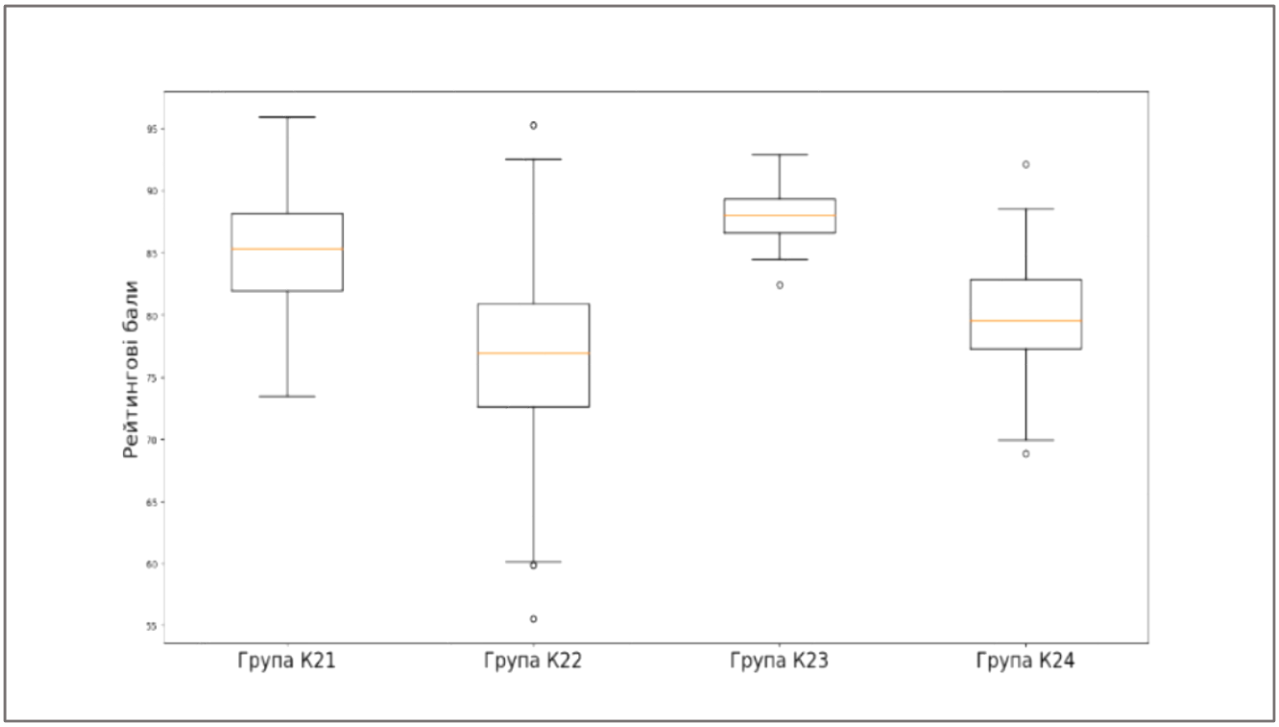


Рис. 5.27 – Приклад використання бокс-плот діаграм для порівняльного аналізу вибірок

Аналіз графіка дозволяє досить легко зробити висновок про групи більш і менш успішні в навчанні; звернути увагу на однорідність контингенту в кожній групі або навпаки, не те, що в групі присутні курсанти з досить різним рівнем підготовки; визначити де саме є курсанти, що відрізняються надзвичайно високими та навпаки, низькими оцінками тощо. Зрозуміло, що без використання графіку типу бокс-плот зробити висновки по якості успішності в різних групах було-б суттєво складніше.

Наступний приклад, що використовує вже відомий нам файл температур у місті Києві, будує графік, який демонструє відмінність розподілення температур в кожному місяці. Продемонструємо це на даних 2014 року. Будемо будувати графік (Рис. 5.28) не за допомогою засобів безпосередньо самого *matlotlib*, а використовуючи його надбудову в пакеті *pandas*.

```
t_min = pd.read_csv('Kiyv-T-min.csv', header=0, delimiter=';')
t_min['Date'] =pd.to_datetime(t_min['Date'], format='%d.%m.%Y')
t_min.set_index('Date',drop=True, inplace=True)
groups = t_min['2014'].groupby(pd.Grouper(freq='M'))
months = pd.concat([pd.DataFrame(x[1].values) for x in groups], axis=1)
months = pd.DataFrame(months)
```

```
months.columns = range(1,13)
months.boxplot()
>>>>
```

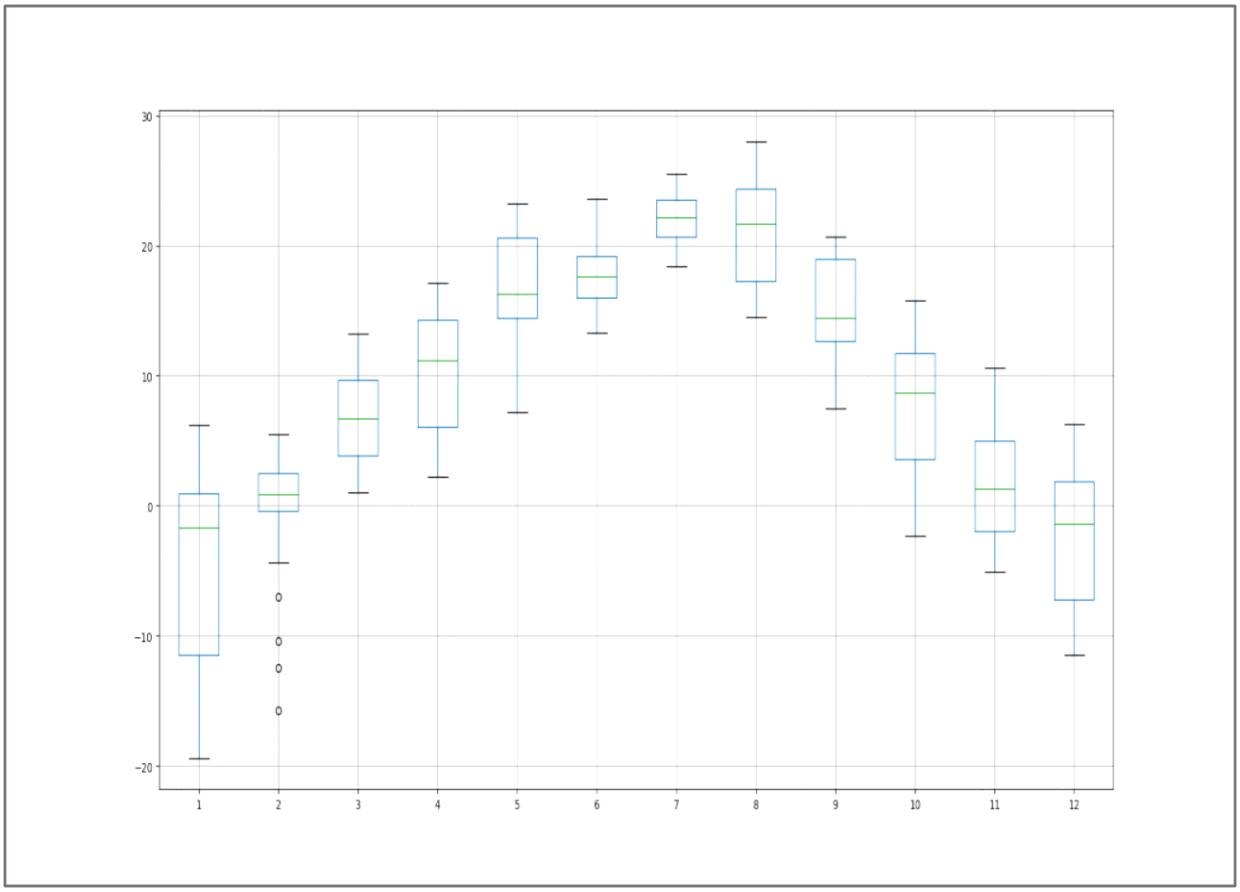


Рис. 5.28 – Аналіз помісячного розподілу температур за допомогою бокс-плот графіку

Треба визнати, що один правильно підібраний та побудований графік іноді здатен розповісти про поведінку нашого об’єкту спостереження більше, ніж багато рядків або сторінок простого пояснювального тексту.

Контрольні питання

1. В чому відмінність лінійної діаграми та діаграми розкиду; лінійної діаграми та стовпчикової діаграми?

2. З яких основних об’єктів складається об’єкт класу «малюнок» з модулю *matplotlib*?

3. Наведіть приклади використання «бульбашкової» діаграми розкиду. Коли такі діаграми має сенс використовувати?

4. Яка основна область використання «бокс-плот» діаграм?

6. ГІСТОГРАМИ. ЗАСОБИ ЇХ ПОБУДОВИ ТА АНАЛІЗУ

Після збору та первинної перевірки даних їх майже завжди необхідно представити так, щоб дослідник міг чітко зрозуміти основні особливості зібраної вибірки. Існує три способи подання даних: текстовий, табличний та графічний. Текстове представлення здебільшого використовується тоді, коли треба саме пояснити, семантично описати отримані дані, не турбуючись про формалізацію цього процесі. Однак дані, які представлені виключно в такому вигляді, для подальшої комп'ютерної обробки непридатні. Представлення даних в вигляді таблиць дозволяє детально переглянути дані, організувати їх ефективне зберігання та, можливо, пошук, але не дає узагальнюючого уявлення про поведінку всієї сукупності накопичених даних. Нарешті, графічне представлення даних з одного боку є привабливим способом зберігати певну семантичну інтерпретацію даних, з другої – дає можливість використати відповідні елементи формалізації представлення, та дати уявлення про всю вибірку в цілому. Графічне представлення може бути однаково легко зрозумілим і спеціалісту з обробки та аналізу даних, і спеціалісту в предметній галузі.

Гістограма – один із способів графічного представлення інформації про вибірку або набір даних. Зокрема, гістограма – це чудовий інструмент для швидкої оцінки розподілу ймовірностей, який інтуїтивно зрозумілий практично будь-якій аудиторії. Гістограма дозволяє зробити опис набору даних більш компактним ніж безпосередні забрані дані (про які часто говорять як про «сирі дані»), однак при цьому зберігаючи всі найважливіші характерні особливості. Гістограми дозволяють побачити, як розподілені значення змінних за інтервалами групування, тобто дають уявлення, як часто змінні приймають значення з різних діапазонів. Гістограма наочно показує, які значення чи діапазони значень досліджуваної змінної є найчастішими, наскільки сильно вони різняться між собою, як сконцентрована більшість спостережень навколо середнього, чи є розподіл симетричним чи ні, чи має він одну чи декілька мод, дають можливість візуально оцінити подібність розподілів, що спостерігаються, з теоретичними або очікуваними розподілами. Тобто гістограма дає якісну інформацію про розподіл, яка не може бути виражена якимось одним чисельним показником.

Наприклад, аналіз гістограм часу, який користувач проводить за читанням інформації розміщеної на сайті, і виявлені на гістограмі асиметричності

розподілу можуть підказати, що більшість користувачів, хоч і відвідують сайт, але не затримаються тут довго і лише мала частина лишається на тривалий час, тобто досить поглиблено ознайомиться з наданою інформацією. Для спеціаліста з маркетингу це може бути ознакою з одного боку недостатньою зацікваленістю відвідувачів інформацією, з іншого – ознакою наявності користувачів, на яких доцільно спрямувати спеціальні маркетингові заходи (тартетінг реклама). Наявність на гістограмі соціальних даних помітних «провалів» можуть нести важливу інформацію про соціальне розшарування групи покупців тощо. Гістограму можна розглядати як аналог функції щільності розподілу безперервної випадкової величини і за виглядом гістограми підібрати відповідний їй теоретичний закон (модель) розподілу.

Python і його пакети розширення для наукових та інженерних розрахунків пропонують кілька різних варіантів побудови гістограм. Розглянемо три з них:

• побудова гістограм на «чистому» *Python* без використання сторонніх бібліотек (самостійне створення спрощених функцій з нуля корисно як перший крок до розуміння складніших засобів, хоча скоріше за все навряд-чи буде використовуватися в реальних проектах);

• побудова гістограм за допомогою *numpy* для узагальнення основних даних;

• відображення отриманої гістограми за допомогою *matplotlib*.

У дослідників-початківців подекуди також виникають складнощі при спробах побудови гістограми на основі отриманих вибіркових даних, Тому розглянемо детально, як гістограми будуються, як можна інтерпретувати дані, представлені в вигляді гістограми та як гістограми використовуються при рішенні окремих задач з аналізу даних.

6.1. Гістограма. Неформальне визначення

Часто гістограмою називають графічне представлення даних у вигляді вертикальних прямокутників, висота яких пропорційна їх значенням. Строго кажучи таке визначення не є правильне, адже воно співпадає з визначенням стовпчикової діаграми. З іншої сторони така подібність підказує, що гістограма може розглядатися як окремий, специфічний вид стовпчикової діаграми. І дійсно, в статистиці та науці про дані вважають, що висота стовпчика такої стовпчикової діаграми пропорційна кількості появи конкретних значеннь серед множини всіх значень вибірки, і відповідно основи прямокутника вибираються не довільно, а

визначені довжиною інтервалу (інтервалів), на які розбивається множина можливих значень вибірки.

Наочно представити (Рис. 6.1) відповідність даних вибірки і побудованої на їх основі гістограми де після розбиття множини можливих значень вибірки

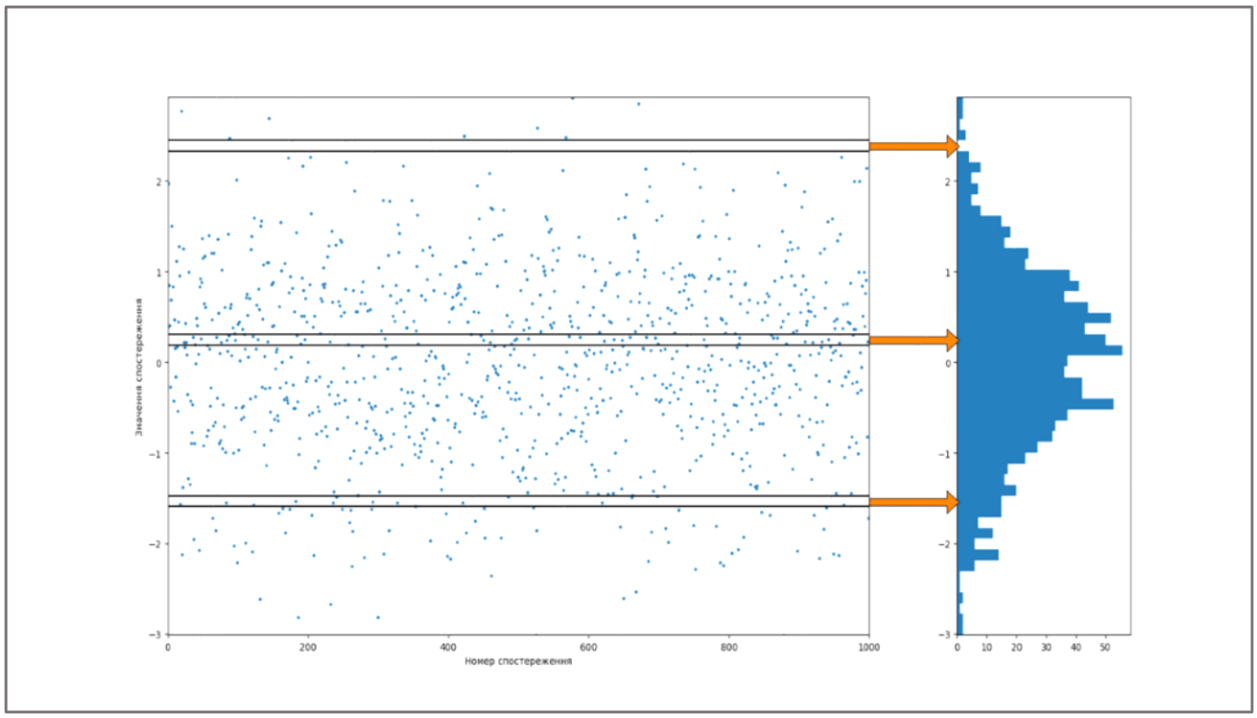


Рис. 6.1 – Неформальне пояснення процесу побудови гістограми

(вісь Y, ліва діаграма розкиду) на деяку кількість інтервалів, робиться підрахунок елементів вибірки, що потрапляють в кожний з цих інтервалів (горизонтальні «коридори» на лівій діаграмі. Щоб зберегти ясність на малюнку показано лише три таких коридора, хоча в дійсності так розбивається вся множина значень вибірки.). Саме ця кількість елементів, що потрапляє в кожний інтервал множини можливих значень, і відображається у вигляді стовпчика на гістограмі справа, причому висота стовпчика початково дорівнює вказаній кількості, але може легко бути перерахована в долю елементів в інтервалі по відношенню до загальної кількості елементів всієї вибірки (тобто – нормуватися до меж від 0 до 1). Зрозуміло, що при такому нормуванні форма діаграми не зміниться, а зміниться лише масштаб відповідної осі гістограми. Зауважимо, шо саме така нормалізація дає змогу порівнювати між собою діаграми вибірок з різною кількістю елементів в них. В теорії ймовірності доводиться, що при збільшенні до нескінченності кількості елементів вибірки та зменшенні довжини інтервалів,

гістограма (знана також як емпірична або вибіркова функція розподілу) перетворюється на функцію щільності розподілу.

6.2. Побудова гістограм за допомогою засобів базового Python

Зазначмо, що коли говорять про гістограму, більшість людей розмірковують про неї саме як про графічний об'єкт. Однак з точки зору дослідника, а тим більш з торчки зору програміста, що пише програму для створення та відображення гістограми, це в першу чергу об'єкт, що складається з двох елементів – списку інтервалів, на які розбивається множина можливих значень вибірки, та списку кількості об'єктів вибірки (або – в разі попереднього нормування – долі об'єктів), які потрапляють по кожного з інтервалів. В *Python* перший з цих об`єктів в свою чергу представляється послідовністю кордонів, які розбивають множину можливих значень на інтервалів. Відразу нагадаємо, що кількість таких кордонів завжди на одиницю більша ніж кількість самих інтервалів, причому найменший з цих кордонів –має бути не більший за мінімальне значення вибірки, а найбільший – не менший максимального значення.

Почнемо з створення допоміжної функції, яка буде рахувати, скільки разів певне значення зустрічається в вибірці. Для спрощення розглянемо випадок вибірки, значення якою можуть бути виключно цілими числами.

```
np.random.seed(12345)
left=1
right=21
n=100
ser=np.random.randint(left,right,n)
dict_of_elem=count_elements(ser)
print(dict_of_elem)
>>>>
3: 5, 6: 12, 2: 5, 5: 2, 10: 5, 18: 5, 15: 6, 17: 2, 19: 8, 12: 9, 14: 3, 11: 2, 7: 4, 8: 4, 1: 7, 16: 2,
4: 3, 20: 3, 9: 6, 13: 7}
```

Результатом роботи створеної функції є словник, кожен елемент якого зберігає значення елементу та кількість появ цього елементу в вибірці. Відсортуємо отриманий словник за значеннями ключа.

```
sorted_dict_of_elem=dict(sorted(dict_of_elem.items()))
print(sorted_dict_of_elem)
>>>>
{1: 7, 2: 5, 3: 5, 4: 3, 5: 2, 6: 12, 7: 4, 8: 4, 9: 6, 10: 5, 11: 2, 12: 9, 13: 7, 14: 3, 15: 6, 16: 2,
17: 2, 18: 5, 19: 8, 20: 3}
```

Припустимо, що потрібно побудувати гістограму, в якій буде 8 інтервалів. Визначимо кордини цих інтервалів.

```
n_bins=8
gr_bins=np.linspace(left,right-1,n_bins+1)
```

Тепер порахуємо, скільки елементів потрапляють в кожний інтервал. Не забуваємо, що оскільки інтервали визначаємо як напіввідкриті справа, то для останнього інтервалу треба зробити виключення, та включити в нього значення, що співпадають з його (інтервалу) правим кордоном.

```
cnt_bins=dict()
cnt=0
ib=1
for elem in sorted_dict_of_elem.keys():
 if elem==gr_bins[ib]:
 cnt_bins[ib-1]=cnt+sorted_dict_of_elem[elem]
 break
 if elem<gr_bins[ib]:
 cnt=cnt+sorted_dict_of_elem[elem]
 else:
 cnt_bins[ib-1]=cnt
 cnt=sorted_dict_of_elem[elem]
 ib=ib+1
```

Тепер можемо відобразити отримані результати графічно (Рис. 6.2), використавши для цього стовпчикову діаграму та функцію її створення з пакету *matplotlib*, яка розглядалася в главі 5:

```
bin_width=gr_bins[1]-gr_bins[0]
plt.bar(gr_bins[:-1],cnt_bins.values(),width=bin_width,align='edge')
>>>>
```

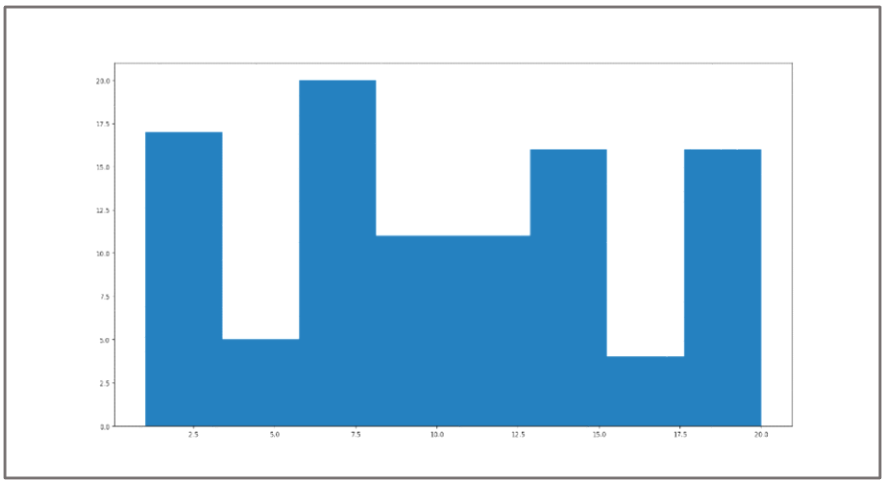


Рис. 6.2 – Побудова гістограми за допомогою стовпчикової діаграми

6.3. Гістограми та стовпчикові діаграми

Гістограма та стовпчикова діаграма – це два способи відображення даних у формі діаграми. Оскільки обидва використовують стовпці для відображення даних, початківцям часто важко їх розрізнити.

Гістограма є графічним представленням інофрмації у вигляді стовпців і служить в першу чергу для відображення частоти появи даних у вибірці. На відміну від цього стовпчикова діаграма є графічним представленням даних, яке використовує стовпці для порівняння різних категорій даних між собою. Оскільки відомо, що дані представлені (виміряні) в будь якій шкалі можуть бути послаблені до номінальної (категоріальної) шкали, при побудові стовпчикової діаграми в якості значень по осі X теоретично можна використовувати будь які дані, в тому числі і ті, що виміряні в числових шкалах. Це часто приводить до помилкового відображення даних, коли по осі X відкладаються, наприклад, номер елемента в вибірці, а по осі Y – саме значення цього елемента. Зрозуміло, що таке рішення буде помилковим. Достатньо замість стовпчикової використати, наприклад, діаграму розкиду, щоб зрозуміти це.

Крім того ще раз підкреслимо, що у гістограми має значення ширина інтервалів значення по осі X, яка може обиратися дослідником. Більше того, саме площа (а не висота) стовпця кожного інтервалу є характеристикою набору даних.

У стовпчикової діаграми ширина стовпця семантичного значення не має, а послідовність відображення значень за віссю X може бути абсолютно довільною. Досить часто ці стовпці впорядковуються по висоті виключно для кращого та більш зрозумілого відтворення інформації. У гістограм послідовність, в якій інтервали відображаються по осі X, суворо фіксована та її зміна загрожує спотворенням інформації про властивості даних. Взаємне розташування шпальт гістограми також несе важливу інформацію про властивості набору даних. Безглуздо говорити, наприклад, про колокоподібність, симетрію або скошеність набору даних, представлених стовпчиковою діаграмою, в той час, як такі характеристики гістограми здатні надати важливу інформацію.

Завершуючи зробимо висновок. Стовпчикова діаграма відображає самі значення даних, які обробляються. Гістограма – є однією з узагальнюючих характеристик даних, що отримана після їх обробки.

6.4. Використання питру для побудови гістограми

Як було зазначено, «ручна» побудова гістограми сьогодні має здебільшого методологічне значення. Пояснюється це тим, що більшість пакетів, що використовуються при науковій та інженерній обробці даних, мають відповідні засоби, які, як правило, працюють дуже ефективно, а також надають більше додаткових можливостей. Так, замість виконання циклу визначення кордонів інтервалів та для підрахунку кількості елементів вибірки, які потрапляють до кожного інтервалу, користувачам *numpy* пропонується скористатися функцією *histogram()*. Функція має параметри, перший з яких є обов'язковим, інші – необов'язковими. Формат цієї функції наступний:

```
numpy.histogram(data, bins=10, range=None, weights=None, density=None)
```

Перший параметр (*data*) – це посилання на масив, якій містить значення вибірки, яка досліджується.

Параметр *bins* задає кількість інтервалів на які треба розділити множину можливих значень вибірки. Однак, якщо в якості значення цього параметра буде переданий список або масив числових значень, функція буде використовувати їх як кордони інтервалів. Зрозуміло, що у такий спосіб дослідник має змогу в разі потреби створити інтервали різної ширини. Якщо-ж цей параметр не заданий, за замовчуванням буде створено 10 однакових за ширино інтервалів.

Параметр *range* дозволяє обмежувати вибірку, тобто використовувати для побудови гістограми не всі її значення, а лише ті, які обмежені зверху та знизу значеннями, які задаються двохелементним кортежем в якості значення цього параметру.

Параметр *weights* приймає в якості значень масив такої самої розмірності, як і параметр *data*. При підрахунку кількості потраплянь до інтервалу кожне значення вибірки додає до суми не одиницю (як зазвичай), а значення, що вказано в якості його «ваги».

Параметр *density* – необов'язковий. Якщо параметр отримує значення *True*, буде виконуватися нормалізація кількості елементів, що потрапляють в кожний з інтервалів. Тобто перед формуванням результатів порахована кількість елементів для кожного інтервалу буде поділена на кількість елементів в вибірці (або на суму ваг всіх елементів вибірки якщо був заданий пареметр *weights*).

Функція повертає два значення, а саме – масив, який містить значеня стовпиків гістограми, і масивив даних типу *float*, що містить кордони інтервалів.

Після застосування функції

```
hist, bin_edges = np.histogram(ser, bins=n_bins)
```

щоб впевнетися в тотожності результатів порівняємо їх з тими, що ми вже отримували раніше:

```
bin_edges==gr_bins
>>>>
array([ True, True, True, True, True, True, True, True, True])

hist ==[cnt_bins[key] for key in cnt_bins]
>>>>
array([ True, True, True, True, True, True, True, True])
```

Звісно, впевнитися можна і використавши для відображення попередньо отриманих за допомогою функцієї *np.histogram()* результатів стовпчикову діаграму пакету *matplotlib* та порівнявши отриманий графік з тим, що наведений вище.

```
plt.bar(bin_edges[:-1],hist,width=bin_width,align='edge')
```

Як буде виглядати гістограма при розбитті її, наприклад, на 6 інтервалів з не однаковою довжиною інтервалів, продемонструємо в наступному прикладі (Рис. 6.3):

```
hist, bin_edges = np.histogram(ser,bins=[0,1,3,10,12,20])
bin_width=bin_edges[1:]-bin_edges[:-1]
plt.bar(bin_edges[:-1],hist,width=bin_width,align='edge')
>>>>
```

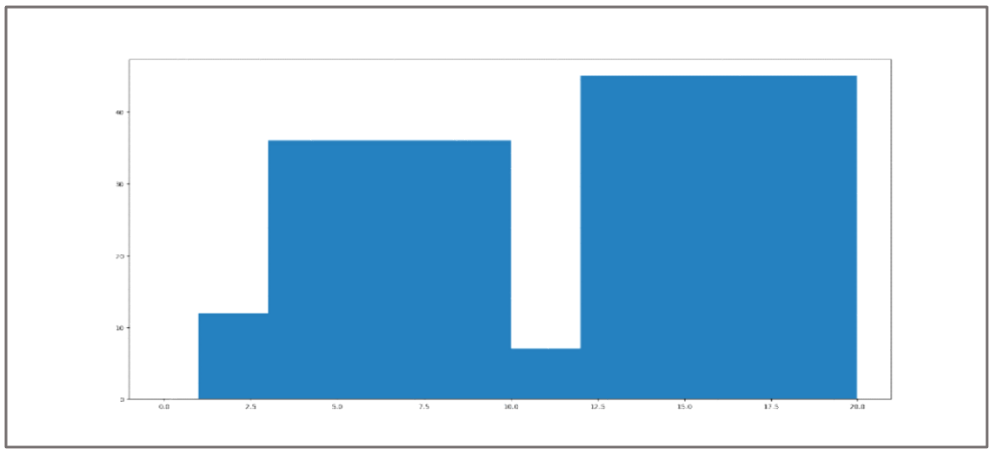


Рис. 6.3 – Гістограма з нерівними інтервалами

Іноді виникає необхідність побудувати гістограми для декількох вибірок, причому зробити це по однаковим інтервалам (Рис. 6.4). Для зручності виконання такого дослідження використовується метод *np.histogram_bin_edges()*. Об'єкт, що повертається цим методом, може бути в подальшому використаний у якості значень параметрів bins при побудові гістограми інших вибірок.

```
n_bins=12
shared_bins = np.histogram_bin_edges(ser, bins=n_bins)
bin_width=shared_bins[1]-shared_bins[0]
hist,bin_edges= np.histogram(ser, bins=n_bins, density=True)
plt.bar(shared_bins[:-1],hist,width=bin_width,align='edge',alpha=0.7)
ser2 = np.random.normal(loc=15, scale=3, size=5*n)
hist1, bin_edges = np.histogram(ser2 ,bins=shared_bins,density=True)
plt.bar(shared_bins[:-1],hist1,width=bin_width,align='edge',alpha=0.3)
```

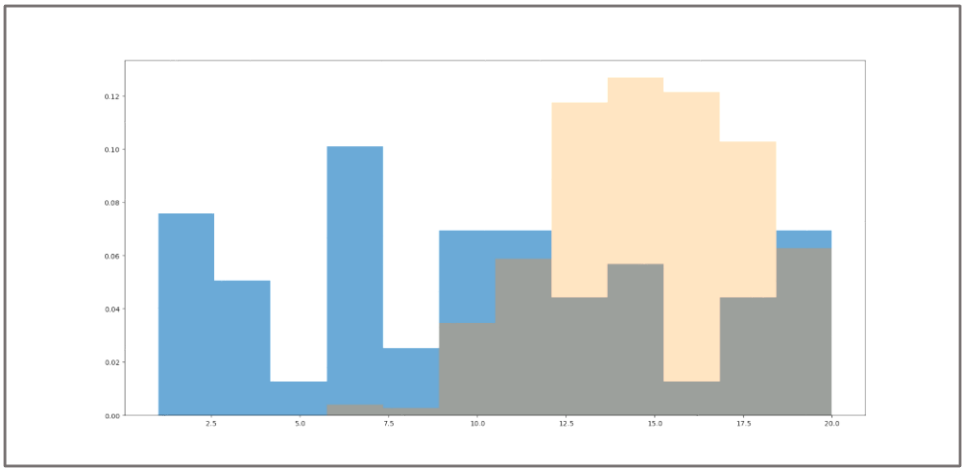


Рис. 6.4 – Дві гістограми на одному малюнку

Зверніть увагу, що в даному випадку порівняти гістограми, тобто відобразити їх в однаковому масштабі по осі Y незважаючи на те, що в першій вибірці в п’ять разів менше значень ніж в другій, дала змогу саме явна вказівка значення параметра *density=True* для обох гістограм.

Зрозуміло, що до побудованих гістограм ми можемо використовувати усі засоби форматування для надання нашим результатам кращого візуального вираження. Деякі такі можливості продемонстрвоані на Рис. 6.5.

```
hist, bin_edges = np.histogram(ser, density=True)
plt.bar(bin_edges[:-1], hist, width = bin_edges[1]-bin_edges[0],
 color='#0504aa',alpha=0.7,align='edge')
# plt.xlim(min(bin_edges), max(bin_edges))
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Значення елементів вибірки',fontsize=15)
plt.ylabel('Частота',fontsize=15)
plt.xticks(fontsize=15)
plt.title('Гістограма розподілу',fontsize=20)
>>>>
```

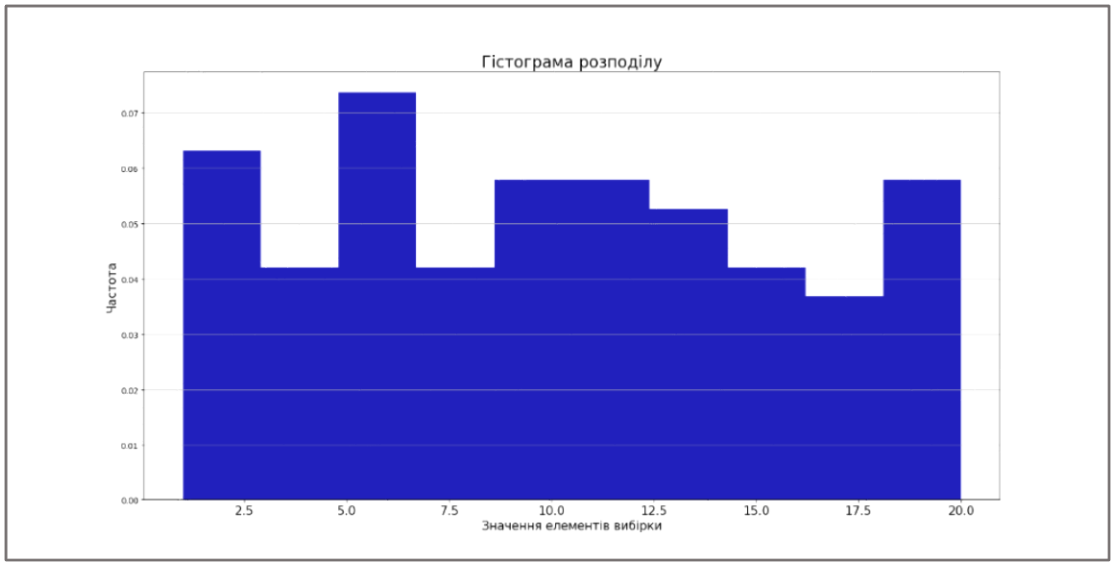


Рис. 6.5 – Приклад використання елементів форматування при побудові гістограм

6.5. Візуалізація гістограм за допомогою matplotlib та pandas

Строго кажучи, для виконання подальшої дослідницької роботи з даними, використання відповідних методів, процедур та алгоритмів даних, що надаються функцією *np.histogram()* виявляється досить. Однак якщо метою є саме візуальне представлення інформації – для виконання презентації, в учбових цілях, для первинного ознайомлення з отриманими даними тощо – доводиться поєднувати

цю функцію з засобами побудови діаграм, зокрема з функцією *plt.bar()* з пакету *matplotlib*. Тому *matplotlib* має в своєму складі і спеціальні засоби для візуалізації гістограм, а саме функцію *hist()*, яка по суті поєднує функцію побудови гістограми з пакета *numpy* та функцію візуального відображення отриманих результатів. Найпростіший спосіб створити гістограму за допомогою *matplotlib* – просто викликати функцію, передавши їй в якості параметру набір значень вибірки даних, що досліджується:

```
hst=plt.hist(ser)
```

Побудована в результаті виконення цієї функції діаграма нічим не відрізняється від діаграми, отриманої вище. Будучи комбінацією функцій *np.histogram()* та *plt.bar()*, функція виконує і підготовку даних для побудови гістограми і безпосередньо їх візуалізацію. В якості результату функція повертає об’єкт, який складається з трьохелементного кортежа, перший елемент якого є *numpy*-масив з значень, які визначають висоту стовпчиків гістограми, другий елемент – *numpy*-масив з значень, що відповідають кордонам інтервалів, а третій – посилання на об`єкт, що описує параметри самої стовпчикової гістограми. Функція може також приймати значення аргументів, деякі з яких знайомі з попереднього розгляду. Ключовим аргументом звісно є аргумент *bins*, який

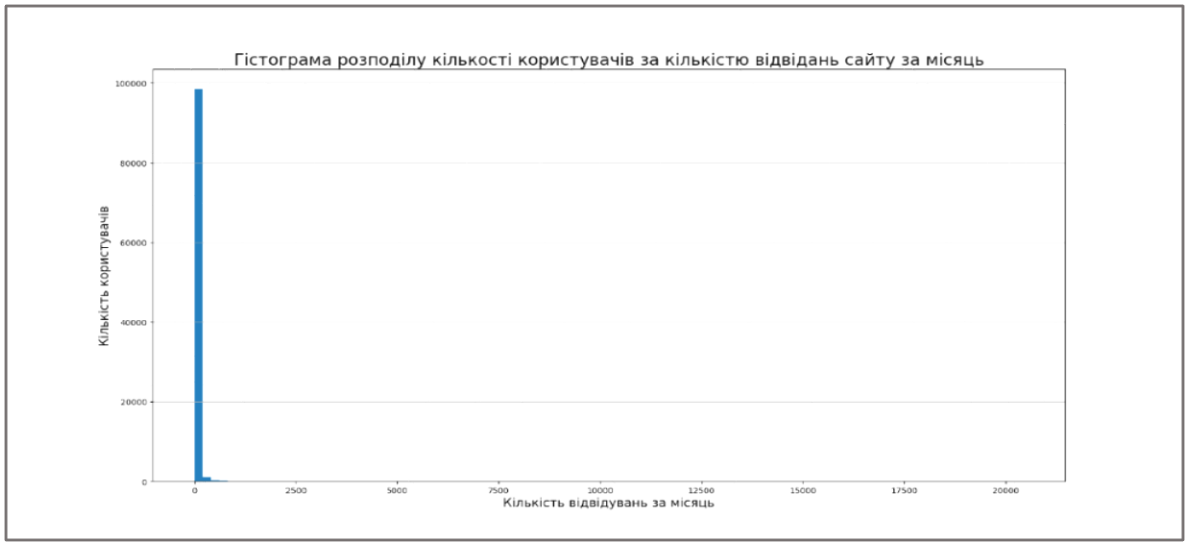


Рис. 6.6 – Візуалізація даних про кількість відвідувачів сайту у вигляді гістограми

визначає кількість інтервалів однакової ширини в діапазоні значень вибірки даних або кордони цих інтервалів при довільному діленні множини значень на інтервали.

Якщо при аналізі виявляється, що надані дані мають кілька інтервалів із значно більшою кількістю значень, що потрапляють до них, ніж більшість інших інтервалів, може виявитися корисним виконати візуалізацію з використанням логарифмічної шкали. Це можна зробити задавши значення аргументу *log=True*. В прикладі, наведеному на Рис.6.6. візуалізуються дані, що показують кількість відвідувань впродовж місяця веб-сайту, що надійшли з кожної окремої IP-адреси.

Хоча з діаграми зрозуміло, що більшість користувачів відвідують сайт певну розумну кількість разів на місяць, але отримати більш осмислену інформацію з цього графіку навряд чи вдасться. Використаємо можливість застосування логарифмічної шкали (Рис. 6.7).

```
plt.hist(number_of_visits, bins=100, log=True)
plt.grid(axis='y', alpha=0.75)
plt.xlabel('Кількість відвідувань за місяць',fontsize=15)
plt.ylabel('Кількість користувачів \n (логаріфмічна шкала)',fontsize=15)
plt.title('Гістограма розподілу кількості користувачів за кількістю відвідань сайту за місяць',fontsize=20)
>>>>
```

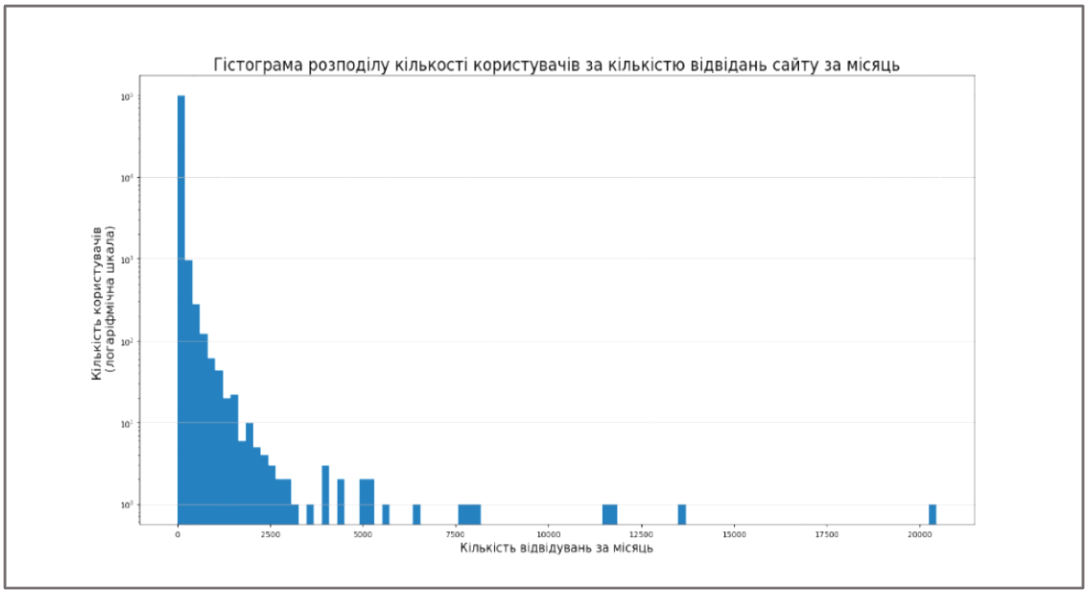


Рис. 6.7 – Візуалізація гістограми з використанням логаріфмічної шкали

Отриманий при цьому графік надає системному адміністратору можливість звернути увагу на наявність серед користувачів таких IP-адрес, кількість відвідань з яких важко пояснити інакше, як результатом дій ботів, скануванням портів, спробою перевірити сайт на вразливість до Dos-атак тощо. Подальші дії

можуть полягати, зокрема, в блокуванні цих IP адрес, що дозволить вберігти сайт від потенційної загрози.

Функція *plt.hist()* дає можливість одночасного відображення на одній картинці кількох гістограм. Це виявляється корисним, наприклад, при порівнянні поведінки однієї і тієї ж характеристики для різних підгруп. Згенеруємо три штучні набори даних та покажемо, як будуть виглядати їх гістограми на одній діаграмі (Рис. 6.8).

```
plt.figure(figsize=[10,8])
d2 = 0.3*np.random.randn(1000)
d3 = 0.3*np.random.randn(1000)
d4 = 0.3*np.random.randn(1000)
n, bins, patches = plt.hist([d2, d3, d4])
>>>>
```

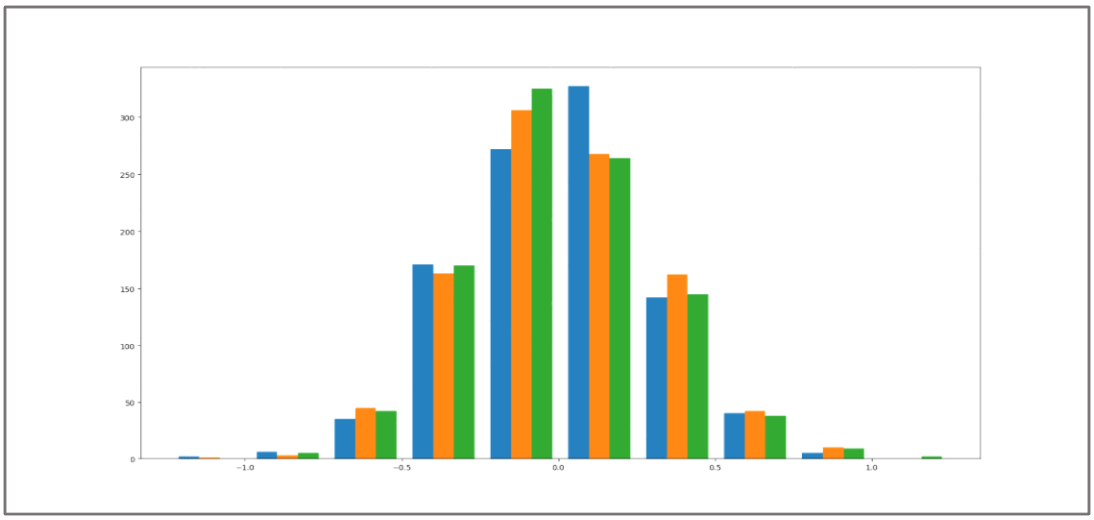
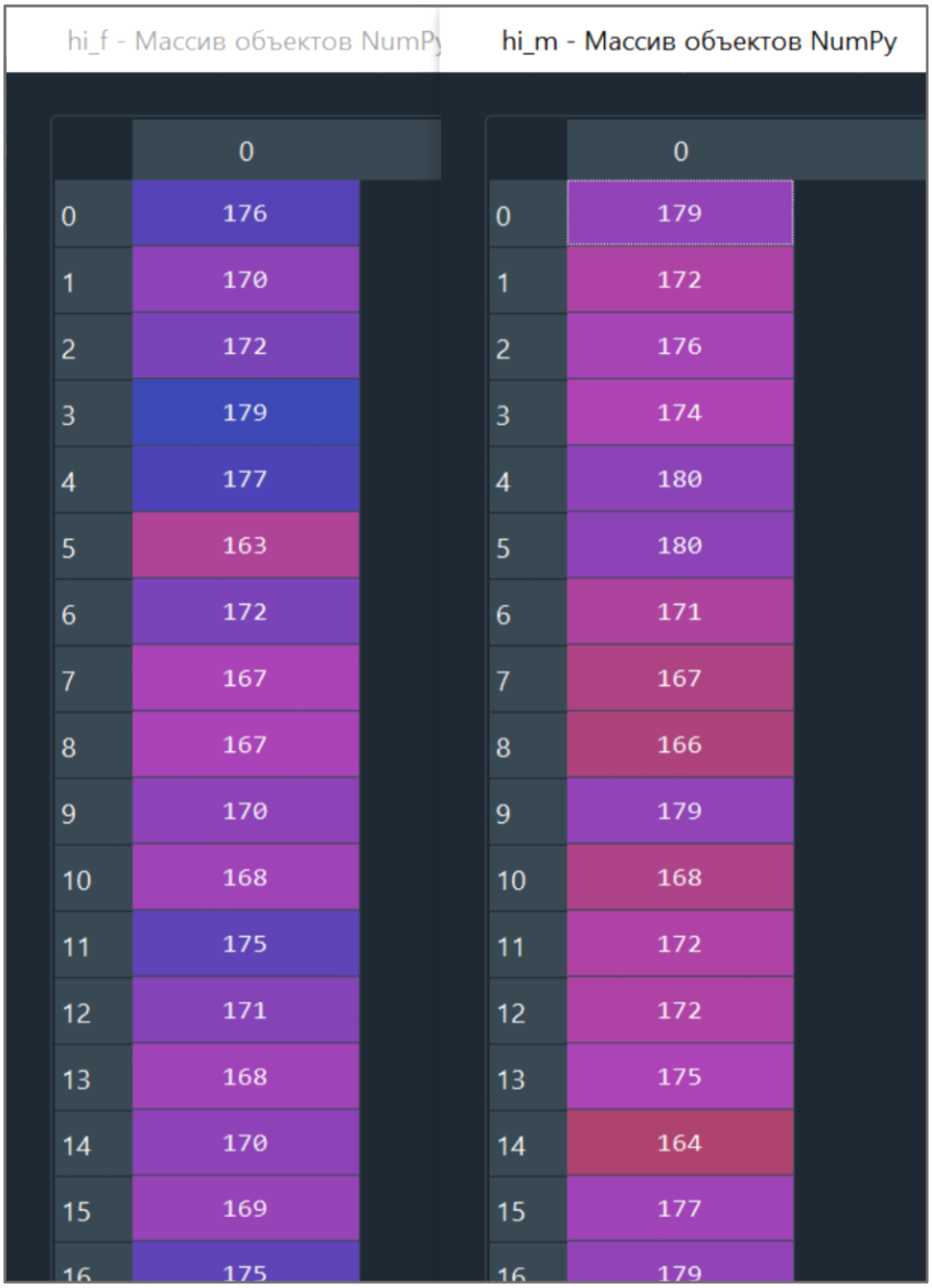


Рис. 6.8 – Відображення декількох гістограм на одній діаграмі

До речі, з даної гістограми добре видно, що хоча вибірки належать одній і тій самій генеральній сукупності, їх вигляд може досить помітно відрізнятися.

В наступному прикладі покажемо, як візуальний аналіз за допомогою гістограми може бути використаний, зокрема, для отримання попереднього висновку при порівнянні даних. Нехай ми маємо два набори данних, що надані для дослідження і відносно яких ми маємо зробити висновки про те, чи отримані ці дані з однієї генеральної сукупності. Уявимо, що досліднику невідоме джерело походження даних і віисновок можна зробити виключно по цим даним.

Зрозуміло, що візуальний аналіз самих даних (Рис.6.9) навряд чи допоможе в цьому.



hi_f - Массив объектов NumPy

	0
0	176
1	170
2	172
3	179
4	177
5	163
6	172
7	167
8	167
9	170
10	168
11	175
12	171
13	168
14	170
15	169
16	175

hi_m - Массив объектов NumPy

	0
0	179
1	172
2	176
3	174
4	180
5	180
6	171
7	167
8	166
9	179
10	168
11	172
12	172
13	175
14	164
15	177
16	179

Рис. 6.9 – Приклад даних для порівняння у табличному вигляді

Побудуємо гістограми, та порівняємо їх.

```
plt.hist(hi_f, bins=shared_bins,alpha=0.7,density=True)
plt.hist(hi_m, bins=shared_bins,alpha=0.7,density=True)
>>>>
```

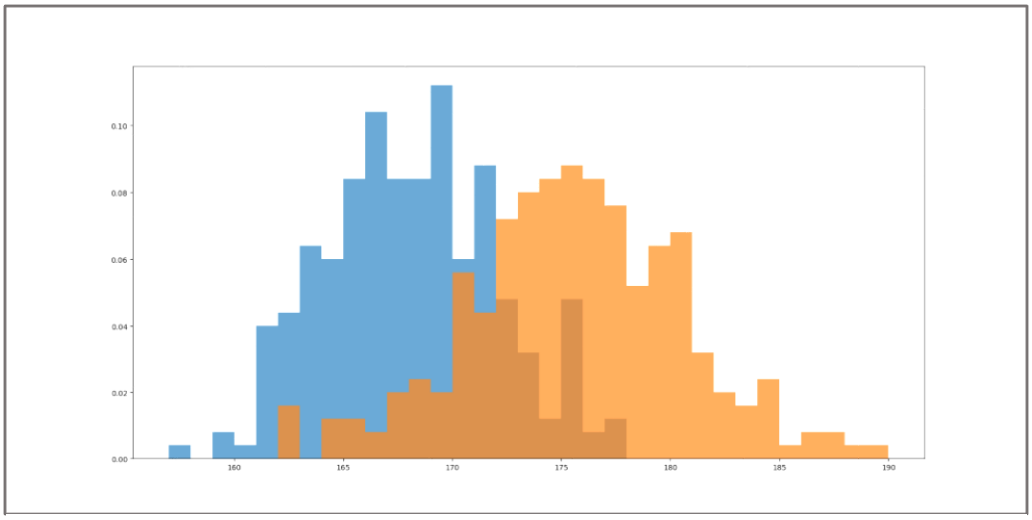


Рис. 6.10 – Порівняння вибірок даних за допомогою гістограм

З наведеної на Рис. 6.10. діаграми видно, що елементи вибірки *hi_m* явно, суттєво та беззаперечно зсунуті вправо (в сторону більших значень) відносно елементів вибірки *hi_f*, а також що розкид значень *hi_m* більший за розкид значень *hi_f*. І хоча в дійсності існують більш науково обґрунтовані методи, здатні прийняти рішення при значно менших відмінностях між вибірками та їх гістограмами, в даному випадку приведеної діаграми достатньо, щоб зрозуміти, що вести мову за те, що обидві вибірки отримані з однієї генеральної сукупності неможливо. (В даному разі це було очікувано, адже в дійсності вибірка *hi_m* представляє собою значення росту курсантів чоловічої статі, а *hi_f* – жіночої).

Функція *plt.hist()* має параметр, який дає змогу побудувати не тільки гістограму (яка є емпіричним аналогом функції щильності розподілу), але й емпиричну (кумулятивну) функцію розподілу. Цей тип функції також надає багато можливостей для подальшого аналізу і має ряд цікавих властивостей. Головною з яких є те, що вона є неспадаючою функцією, яка приймає значення від 0 зліва (теоретично, починаючи від $-\infty$, практично – від мінімального значення вибірки) і до значення 1 справа (теоретично до $+\infty$, практично – до максимального значення вибірки). Маючи при невеликій кількості спостережень та інтервалів сходинковий вигляд, графік цієї функції із збільшенням кількості спостережень стає гладкішим, а емпірична функція розподілу наближається до теоретичної функції розподілу генеральної сукупності чи певної теоретичної моделі розподілу. Емпіричні функції розподілу широко використовують у непараметричних статистичних критеріях (омега-квадрат, Колмогорова —

Смирнова тощо). Для прикладу побудуємо відповідний графік (Рис.6.11, синій графік) з вибірки *hi_f*, що використовувалася в минулому прикладі:

```
plt.hist(hi_f, cumulative=True, bins = 250, density=True,histtype='step',lw=2)
x=np.arange(hi_f.min(),hi_f.max(),0.01)
plt.plot(x,scs.norm.cdf(x,loc=168,scale=5),lw=2,ls='--')
>>>>
```

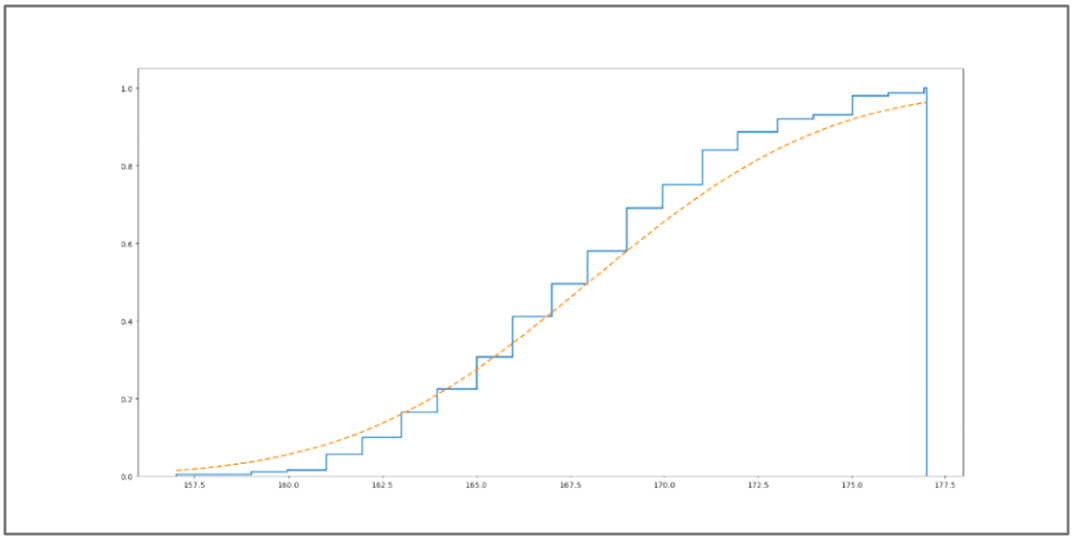


Рис. 6.11 – Емпірична та теоретична функції розподілу вибірки

На Рис. 6.11 також показаний теоретичний закон розподілу (червоний графік), параметри якого були разраховані, а саме – нормальний закон розподілу з середнім значенням 168 та стандартним відхиленням 5. Візуально можна констатувати задовільну відповідність значень наданої вибірки теоретичному закону розподілу, хоча звичайно, для науково обгрунтованої відповіді треба використати відповідний статистичний критерій, наприклад критерій Колмогорова-Смірнова, або критерій χ^2.

При роботі з об'єктами типу *DataFrame* пакету *pandas* генерувати гістограми ще простіше. Це можна зробити за допомогою спеціального методу, назва якого збігається з назвами розглянутих вище – *pandas.hist()*, але може бути застосований як по даних формату *Series*, так і безпосередньо до всіх даних *DataFrame*. В наступному прикладі (Рис. 6.12, 6.13 та 6.14) будуються гістограми, що показують розподіл курсантів за кількістю годин, які вони займаються самостійною роботою. Спочатку використовуючи дані відповідного *DataFrame* будуються гістограми по чотирьом дисциплінам учбового плану. Далі

створюється об’єкт типу *Series*, в якому зберігається сумарний час роботи кожного з курсантів і, нарешті, будується гістограма для цієї серії:

```
hm_work_time=pd.DataFrame({‘Математичний аналіз’:d1, ‘Фізика’:d2,
 ‘Програмування’:d3, ‘Теорія ймовірностей’:d4})
print(hm_work_time.head().round(1))
>>>>
```

Консоль 1/A

	Математичний аналіз	Фізика	Програмування	Теорія ймовірностей
0	14.8	15.0	21.2	25.7
1	16.5	10.0	14.8	19.9
2	10.0	19.1	17.7	17.2
3	10.9	12.2	22.2	16.6
4	8.1	16.5	15.5	13.3

Рис. 6.12 – Початкові дані DataFrame з даними для аналізу

```
hm_work_time.hist(density=True)
>>>>
```

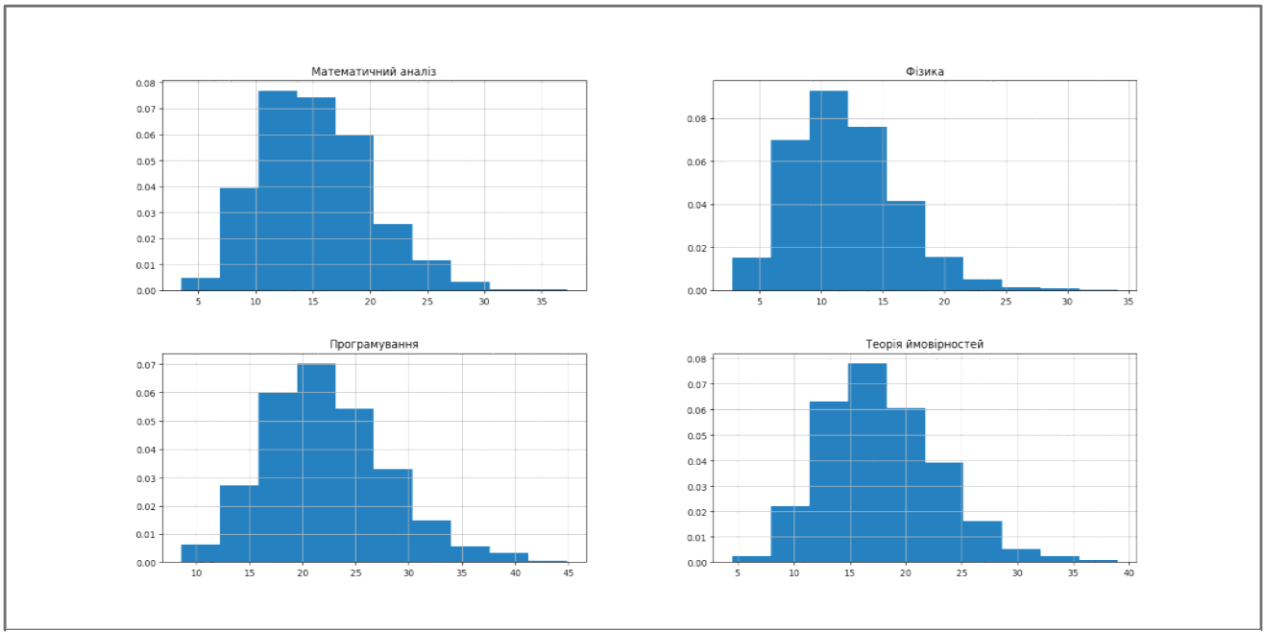


Рис. 6.13 – Гістограми розподілу часу занять по кожній з дисциплін

```
plt.figure()
hm_work_time_total=hm_work_time[‘Математичний
аналіз’]+hm_work_time[‘Фізика’]+hm_work_time[‘Програмування’]+hm_work_time[‘Тео
рія ймовірностей’]
hm_work_time_total.hist(grid=True, bins=20, rwidth=0.9,
 color=‘#607c8e’,density=True)
plt.title(‘Розподіл курсантів за часом самостійної роботи (за місяць)’)
```

```
plt.xlabel('Час на самостійну роботу')
plt.ylabel('Доля серед усіх курсантів')
plt.grid(axis='y', alpha=0.75)
>>>>
```

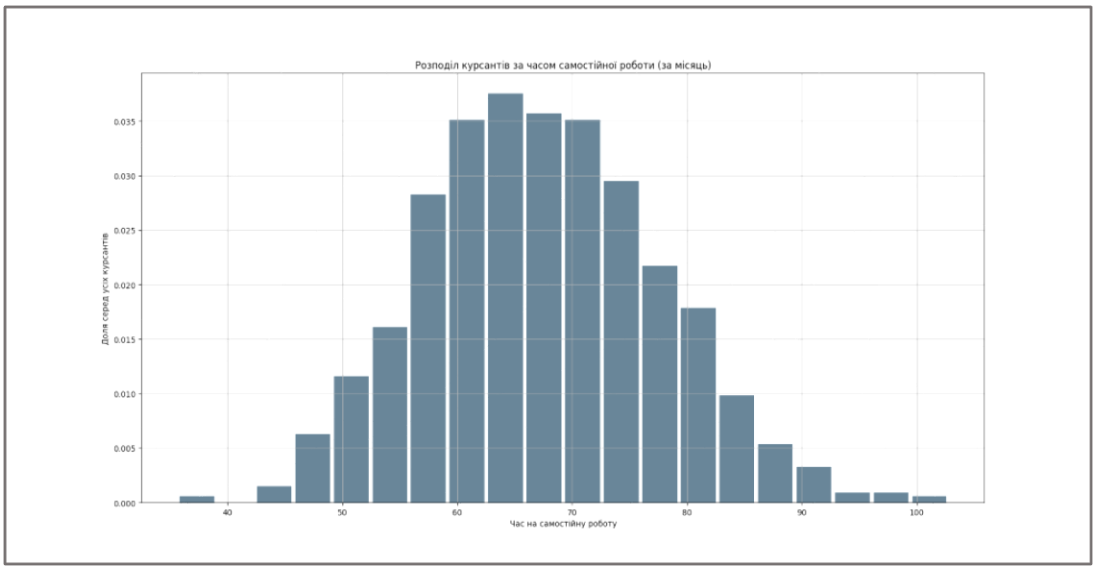


Рис.6.14 – Гістограма розподілу часу самостіних занять для всіх курсантів

Корисно звернути увагу як при побудові останньої гістограми використовується параметр *rwidth* (ширина стовпичка гістограми) для покращення візуального сприйняття інформації яка надається.

6.6. Вибір кількості інтервалів при побудові гістограм

В усіх прикладах, що розглядалися вище, при побудові гістограм застосовувалося така кількість інтервалів, яка обиралася функціями *np.histogram(), plt.hist() ma pd.hist()* по замовчуванню (як правило рівна 10), або ми задавали його довільним чином. Якщо метод дослідження передбачає подальший тонкий статистичний анализ даних, жодний з цих варіантів не може вважатися коректним.

Розглянемо наступний приклад (Рис.6.15). Представлені гістограми побудовані з однієї та тієї-ж вибірки *t* що містить 5000 елементів та представляють значення параметру (в даному разі – температуру) деякого технічного об'єкта. Однак всі гістограми використовують різну кількість інтервалів.

```
fig, ax = plt.subplots(1, 4, figsize=(10, 4))
fig.subplots_adjust(left=0.1, right=0.95, bottom=0.15)
for i, bins in enumerate([10, 30, 300, 3000 ]):
```

```
ax[i].hist(t, bins=bins, histtype='stepfilled')
ax[i].set_xlabel('t')
ax[i].set_ylabel('P(t)')
ax[i].set_title('plt.hist(t, bins={0})'.format(bins),
fontdict=dict(family='monospace'))
>>>>
```

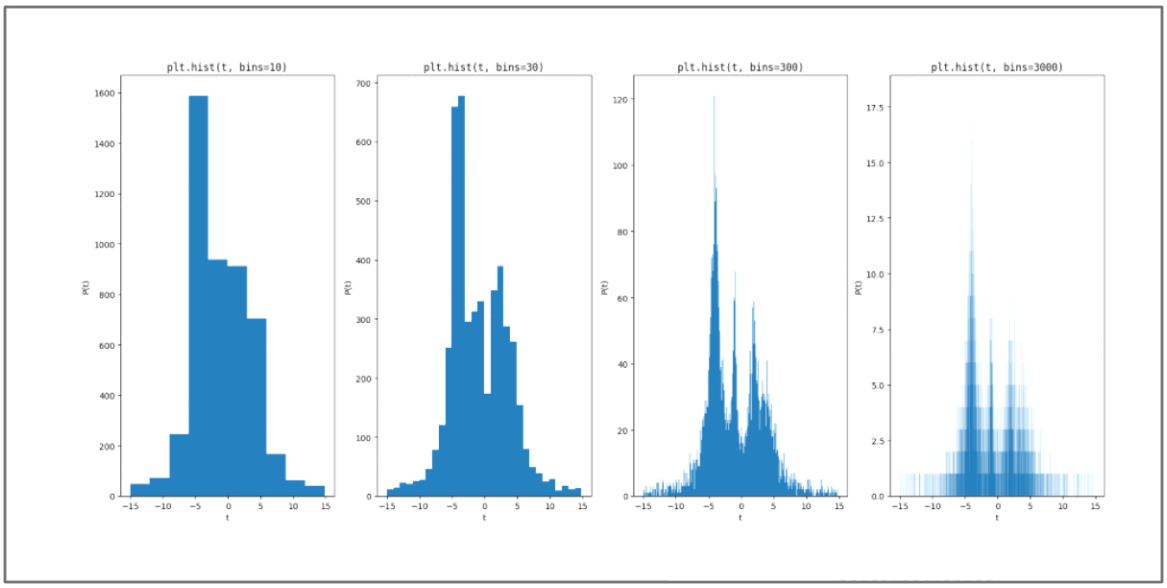


Рис. 6.15 – Гістограми набору даних при різній кількості інтервалів

Помінто, на скількі як отримані графіки різняться між собою. При цьому важко говорити, яка з обраних кількістей інтервалів є оптимальною. При занадто малих значеннях губиться інформація про форму вибірки, практично не діючи досліднику жодного уявлення навіть про кількість мод в ній. При занадто великій кількості інтервалів висота окремих стовпичків співставна з статистичною похибкою вибіркі і теж унеможливлює подальше дослідження.

Нажаль, слід зазначити, що жодного формально обґрунтованого методу, що дозволяє обрати оптимальну кількість інтервалів для довільної гістограми на сьогодні не існує. Випробуваний і застосовуваний більшістю вчених метод є метод «спроб та помилок», який намагається вибрати деяку розмумну з певної точки зору «золоту середину» між ситуаціями втрати інформативності гістограм, що наведені вище. Позатим для деяких випадків вдається знайти якщо не оптимальну кількість інтервалів, то таку, яка дає прийнятні для подальших досліджень результати. В пакеті *matplotlib* реалізовано 7 таких методів. Вибір метода виконується вказівкою відповідного ключового слову в якості значення параметру *bins*. З цих методів найбільш простим (і, віповідно, швидким, що досить важливо при великих обсягах вибірок) є метод метод *'sqrt'*, в якому

кількість інтервалів обирається за формулою $n\_bins = \sqrt{N}$, де N – кількість значень в вибірці. Метод *'rice'* визначає $n\_bins = 2 * \sqrt[3]{N}$. Метод виявляє тенденцію дещо переоцінювати кількість інтервалів, та не враховує величину розкиду даних. Метод *'sturges'* використовує вибір кількості інтервалів за формулою $n\_bins = \log_2 N + 1$. Ця оцінка передбачає, що дані розподілені за нормальним законом розподілу і є дещо консервативною для великих вибірок та для вибірок, що розподілені за іншими законами. Навпаки, метод *'scott'*, який обирає кількість інтервалів пропорційною стандартному відхиленню даних та зворотньо пропорційно кореню кубічному від їх кількості: $n\_bins = \sigma * \sqrt[3]{\frac{24 * \sqrt{\pi}}{N}}$ демонструє кращі результати саме на великих наборах даних. Крім того цей метод не стійкий до наявності викидів к даних. Метод *'fd'* (Freedman Diaconis Estimator – оцінка Фридмана-Диакониса) використовує оцінку кількості інтервалів за формулою $n\_bins = \frac{2(Q_{75} - Q_{25})}{\sqrt{N}}$, де Q_α – α-процентиль значень вибірки. Метод *'auto'* автоматично оберає серед методів *'sturges'* і *'fd'* той, який пропонує більшу кількість інтервалів.

Метод *'doane'* можна розглідати як покращений метод *'sturges'*, в якому враховуються можлива скошеність даних, що характерно для вибірок, розподілених не за нормальним законом розподілу.

$$n\_bins = 1 + \log_2 N + \log_2\left(1 + \frac{G}{\sigma_G}\right),$$

де $$G = mean\left[\left(\frac{X - \mu}{\sigma}\right)^3\right],$$

$$\sigma_G = \sqrt{\frac{6(N-2)}{(N+1)(N+3)}}.$$

На завершення покажемо, які значення обираються для тих самих даних різними методами визначення кількості інтервалі, та як при цьому видозмінюється вигляд гістограми (Рис. 6.16).

```
fig= plt.figure()
```

```
fig.subplots_adjust(left=0.2, right=0.8,bottom=0.15,hspace=0.4, wspace=0.2)
methods=['sqrt','rice','scott','fd','auto','sturges','doane']
for i in range(len(methods)):
 ax = fig.add_subplot(3,3,i+1)
 n, bins,patches =ax.hist(t, bins=methods[i], color='#0504aa', alpha=0.7,rwidth=0.75)
 ax.set_title('method=\'{0}\', bins={1} '.format(methods[i],len(bins)),
 fontdict=dict(family='monospace'))
>>>>
```

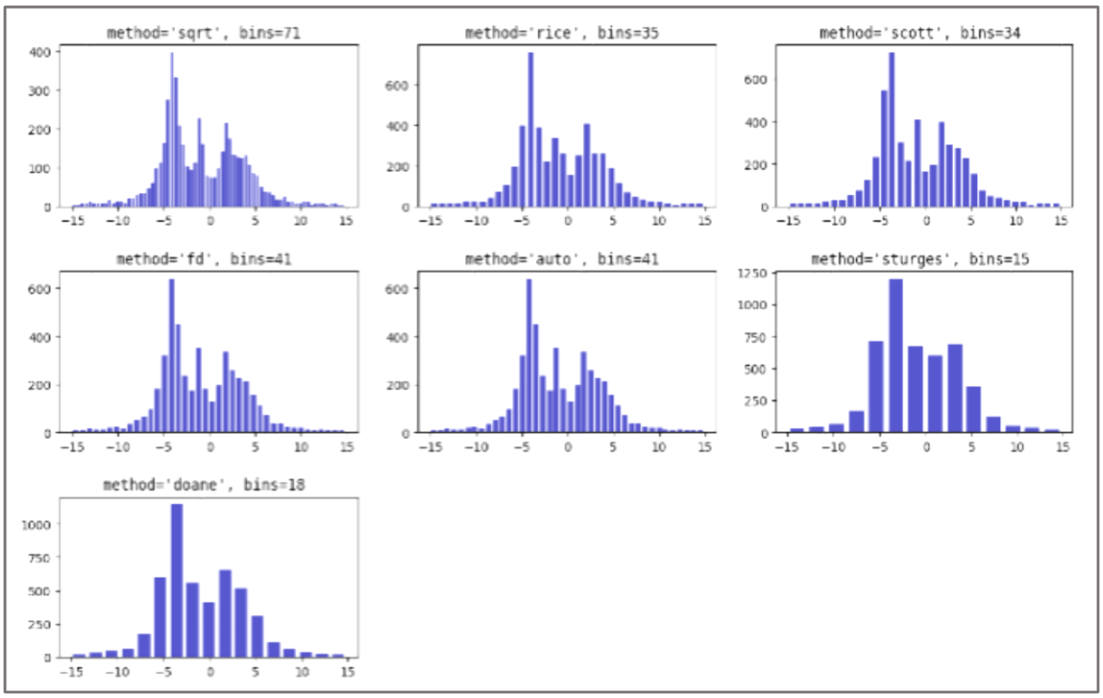


Рис. 6.16 – Гістограми набору даних при різних методах вибору кількості інтервалів

Контрольні питання

1. Що таке гістограма? Яким чином вона будується?
2. Яка відмінність гістограми від стовпчикової діаграми?
3. Яким чином гістограми можуть використовуватися для пошуку аномалій в часових рядах?
4. Як обирається кількість інтервалів при побудові гістограми? На що впливає такий вибір?

7. ВВЕДЕННЯ ТА ВИВЕДЕННЯ ДАНИХ ПРИ РОБОТІ З МОДУЛЕМ *PANDAS*

Розглянуті вище засоби аналізу та інструментального дослідження даних виходили з того, що самі дані тим чи іншим способом вже знаходяться в розпорядженні відповідного програмного застосунку. Однак в дійсності, як було сказано в главі 2, дані можуть потрапляти до застосунків аналізу з різних джерел та відповідно бути представлені в різних форматах.

Зазвичай операції, пов'язані з імпортування даних з різноманітних джерел та операції експортування даних для їх подальшої обробки в інших застосунках є досить нетривіальною задачею і часто займають час та інші ресурси програміста для її реалізації. Популярність *Python* немалою мірою ґрунтується на наявності та простоті набору інструментальних засобів, які він надає програмісту для виконання вказаних дій. Простий синтаксис взаємодії з файлами, інтуїтивно зрозумілі структури даних, зручні засоби пакування та розпакування інформації дозволяють досліднику концентруватися на питаннях глибинного аналізу даних, а не на питаннях формального опису операцій вводу-виводу. Звісно, це не заперечує той факт, що програміст має вільно володіти вказаним простим але ефективним набором інструментів. В поточній главі ми розглянемо деякі, найбільш вживані інструменти та методи їх ефективного використання.

7.1. Формат csv та робота з ним

Формат *csv* (Comma Separated Values) є одним із найпоширеніших форматів імпорту та експорту даних. Кожен рядок у файлі *csv* представляє окремий запис або рядок, який складається з окремих значень, розділених комами (так було на початку використання цього формату і саме через це він і отримав свою назву). З часом та відповідним розвитком потреб програмування до стандарту була додана можливість визначення символу, який використовується для розділення даних, самим програмістом. Це зробило описаний формат одним з найбільш універсальних та широковживаних інструментів зберігання та обміну інформації.

Для роботи з цим форматом в *Python* існує окремий спеціалізований модуль, який має назву *«csv»*. Однак при використанні модуля *pandas* стають доступними цілий ряд функцій для читання табличних даних, що представлені в форматі *csv*, та представлення отриманих даних у вигляді об'єкта *DataFrame*. Це суттєво знижує складність створення програмування застосунків. Ці функції перелічені у

наведеній нижче таблиці. Треба зазначити, що з них найчастіше використовуються функції *read_csv* і *read_table*.

Назва функції	Опис функції
read_csv	Завантажує дані в форматі «з роздільниками» з файлу або по заданій URL-адресі.
read_table	Завантажує дані в форматі «з роздільниками» з файлу або по заданій URL-адресі. За замовчуванням в якості роздільника використовується символ табуляції («\t»).
read_fwf	Читає дані у форматі з фіксованою шириною стовпців, роздільники ігноруються.
read_clipboard	Завантажує дані із буфера обміну. Часто використовується для перетворення на DataFrame дані отримані з веб-сторінки.

Припустимо, ми маємо файл, який містить деякі дані. Такий файл можна створити за допомогою будь-якого текстового редактора або шляхом передачі значень будь-якої програми для запису файлу *csv*. Розглянемо в якості прикладу вже відомий нам файл часового ряду температур у місті Києві, та подивимося, які саме інструменти має використовувати програміст, щоб досягти потрібних результатів.

```
import pandas as pd
t_min = pd.read_csv('Kiyv-T-min.csv', header=0, delimiter=';')
t_min.head(10)
>>>>
 Date T-Av
0 01.01.2011 -3.0
1 02.01.2011 -0.4
2 03.01.2011 -3.0
3 04.01.2011 -4.3
4 05.01.2011 -6.8
5 06.01.2011 -10.3
6 07.01.2011 -9.1
7 08.01.2011 -2.3
8 09.01.2011 0.4
9 10.01.2011 1.0
```

Окрім першого параметру, який задає або сам файл або URL-джерело даних (зверніть увагу наскільки спрощена робота для програміста – йому не потрібно враховувати жодних особливостей, що пов'язані з відмінністю фізичних джерел інформації), функція використовує також параметр , *delimiter*, що безпосередньо

вказує, який саме символ використовуються в файлі для розділення значень. Треба зауважити, шо роздільники поділяють самі значення в рядку, та не використовуються в кінці рядка.

Перший рядок файлу може містити назву колонок, але це не є обов'язковим. Параметр *header* пр значений для керування роботою з заголовком даних (по суті назвою стовпиків таблиці). Наприклад, в наведеному прикладі цей параметр отримує значення 0, тобто вказує на те, що заголовки (назви) стовпчиків записані в рядку файлу з індексом 0. Однак оскільки файл-джерело інформації є стороннім по відношенню до скрипта, який пишеться, дуже часто програміст не має впливу на представлення інформації в цьому файлі. Отже цілком можливою виглядає ситуація, коли в файлі назви стовпчиків даних відсутні. В такому разі прочитати його можна двома способами.

По-перше, можна доручити *pandas* самостійно виконати вибір імен стовпчиків за замовчуванням:

```
t_min1 = pd.read_csv('Kiyv-T-min.csv', header=None, delimiter=';')
t_min1.head(10)
>>>>
          0 1
0 01.01.2011 -3
1 02.01.2011 -0.4
2 03.01.2011 -3
3 04.01.2011 -4.3
4 05.01.2011 -6.8
5 06.01.2011 -10.3
6 07.01.2011 -9.1
7 08.01.2011 -2.3
8 09.01.2011 0.4
9 10.01.2011 1
```

(Примітка. Для коректної роботи останнього прикладу і отримання аналогічних результатів файл-джерело має бути попередньо модифікований, а саме в ньому має бути видалена строчка, що відповідає заголовку файлу. В разі, якщо таку дію не виконати, результатом роботи вказаної функції буде *DataFrame*, в якому перший рядок даних буде містити назви стовпчиків, що, звісно, буде помилкою).

По-друге, програміст має можливість задати бажані назви стовпчиків в *DataFrame*, який створюється, самостійно:

```
t_min2 = pd.read_csv('Kiyv-T-min.csv', names=['Дата спостереження','Температура'],
delimiter=';')
```

```
t_min2.head(10)
>>>>
Дата спостереження Температура
0 01.01.2011 -3.0
1 02.01.2011 -0.4
2 03.01.2011 -3.0
3 04.01.2011 -4.3
4 05.01.2011 -6.8
5 06.01.2011 -10.3
6 07.01.2011 -9.1
7 08.01.2011 -2.3
8 09.01.2011 0.4
9 10.01.2011 1.0
```

Припустимо також, що необхідно зробити так, щоб стовпчик *«Дата спостереження»* став індексом *DataFrame*. Цього можна досягти, задавши значення параметру *index_col*, в якому вказати, що індексом буде стовпець з номером 1 або з іменем *«Дата спостереження»:*

```
t_min2 = pd.read_csv('Kiyv-T-min.csv', names=['Дата спостереження','Температура'],
 delimiter=';', index_col='Дата спостереження')
t_min2.head(10)
>>>>
                    Температура
Дата спостереження
01.01.2011 -3.0
02.01.2011 -0.4
03.01.2011 -3.0
04.01.2011 -4.3
05.01.2011 -6.8
06.01.2011 -10.3
07.01.2011 -9.1
08.01.2011 -2.3
09.01.2011 0.4
10.01.2011 1.0
```

Досить частою, але в одночас складною для дослідника задачою є необхідність обробки даних в разі, коли у таблиці немає фіксованого роздільника, а для поділу полів використовуються пробіли або інші символи. У такому разі можна передати функції *read_table()* регулярний вираз замість роздільника. Розглянемо в якості прикладу текстовий файл, в якому семантично представлена інформація про кількість пакетів з відповідними протоколами, які були

зафіксовані в мережі на певних проміжках часу. Побачити, що саме і як саме записано в файлі можна, наприклад, найпростішим шляхом:

```
print(list(open('prot_test.txt')))
>>>>
['Protocols T1 T2 T3 \n', 'DNS 64 52 34\n', 'TCP 286 441 367\n', 'UDP 105 281 117\n',
'BitTorrent 34 12 22']
```

Видно, що у даному випадку поля розділені змінним числом пробілів. І хоча можна було б переформатовати дані вручну, простіше використати регулярний вираз «\s+»:

```
result=pd.read_table('prot_test.txt', sep='\s+')
print(result)
>>>>
 Protocols T1 T2 T3
0 DNS 64 52 34
1 TCP 286 441 367
2 UDP 105 281 117
3 BitTorrent 34 12 22
```

Продемонструємо ще один користний трюк, який може використовуватися програмістом для спрощення коду. Для цього трошки змінимо зміст нашого файлу, просто видаливши назву першого з стовпчиків («Protocols»):

```
print(list(open('prot_test.txt')))
>>>>
[' T1 T2 T3 \n', 'DNS 64 52 34\n', 'TCP 286 441 367\n', 'UDP 105 281 117\n', 'BitTorrent 34
12 22']
```

І знову звернемося до нашої функції з тими самими значеннями параметрів:

```
result=pd.read_table('prot_test.txt', sep='\s+')
print(result)
>>>>
          T1 T2 T3
DNS 64 52 34
TCP 286 441 367
UDP 105 281 117
BitTorrent 34 12 22
```

Результат змінився досить кардинально. Оскільки імен стовпців на одне менше ніж число стовпчиків, *read_table* інтерпретує таку ситуацію як неявну

вказівку використати стовпчик без назви в якості індексу результуючого об'єкту типу *DataFrame*.

У функції read_csv() та її аналогу – *read_table()* існують багато додаткових аргументів, які допомагають впоратися з широкою різноманітністю файлових форматів. Наприклад, параметр *skiprows* дозволяє пропустити вказані в вигляді індекса рядки при читанні інформації з файлу-джерела:

```
result1=pd.read_csv('prot_test.txt', sep='\s+', skiprows=[1,3])
print(result1)
>>>>
 T1 T2 T3
TCP 286 441 367
BitTorrent 34 12 22
```

Параметр *skiprows* виявляється дуже зручним інструментом, який використовується при необхідності «очищення» даних від сторонньої інформації. Так, при роботі з даними, що формуються сторонніми застосунками, в файлі з інформації може виявитися – наприклад – в вигляді окремих рядків назва таблиці, якісь зведені чи інформаційні позначки тощо. Саме параметр *skiprows* дає засоби легкого обминання такої інформації без необхідності виконання досить складного семантичного розбору змісту інформації файлу-джерела. Подібний до ного параметр *skip_footer* призначений для відмови від читання та обробки деякої кількості останніх рядків файла-джерела (якщо, наприклад, в них містилися прізвище людини, що підписала документ, остаточна зведена сума тощо).

Серед інших параметрів функцій *read_csv()* та *read_table()* зазначимо ті, які мають безпосереднє відношення до обробки значень часу та дат.

parse_dates – вказує на необхідність розібрати дані як дату та час; за замовченням приймає значення *False*. Якщо в якості параметру передається значення *True*, робиться спроба розібрати усі стовпці, що зчитуються з файлу-джерела. Можна також в якості значення цього параметру задати список стовпчиків, які слід об'єднати перед розбором (якщо, наприклад, час та дати задані у різних стовпчиках таблиці).

date_parser – в разі виникнення потреби відмовитися від застосування вбудованого парсеру часу та дат та використати з цією метою власну функцію, програміст може вказати ім'я цієї функції як значення цього параметру.

encoding – оскільки програміст часто не контролює процес створення файлу-джерела, а сам файл може створюватися в різноманітних системах

кодування, даний параметр вказує, яке саме кодування використання для представленні інформації у файлі. Наприклад, при використанні кодування тексту у *Unicode*, у варіанті *UTF-8*, даний параметр має прийняти значення *utf-8*.

squeeze – якщо виявилося, що файл-джерело містить таблицю, що включає один стовпчик, використання цього параметру вкаже, що результатом роботи функції має бути об'єкт типу *Series (*а не об'єкт типу *DataFrame,* як це відбувається за замовченням).

thousands –параметр, що дозволяє вказувати, який роздільник – кома чи точка – використовується при розділені разрядів в числах.

Зрозуміло, що крім засобів читання інфорації з файлу, існують і інструменті, призначені для запису інформації в файл формату *«csv».* Запис інформації, що представлена в форматі *DataFrame* в файл безпосередньо в форматі *csv* виконується за допомогою звертанням до методу *pandas.to_csv().* Наприклад, створений в прикладі вище об'єкт *result* може бути записаний у файл наступним чином:

```
result.to_csv('prot_test.csv')
```

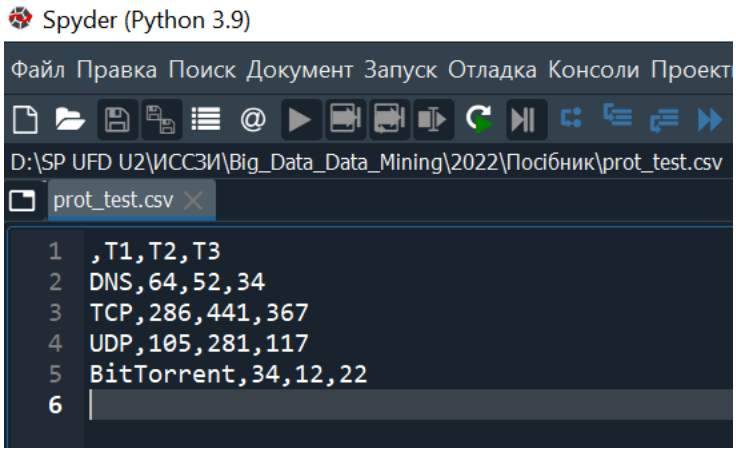


Рис. 7.1 – Приклад змісту файлу з даними розподілу мережевого тафіку за протоколами

Перевірити зміст створеного файлу можна за допомогою будь якого редактору текстових файлів, зокрема стандартного застосунку «*Блокнот*» що входить до складу *MS Windows*, застосунку *Notepad++* – безкоштовного поширеного і дуже зручного редактору текстових файлів, або навіть редактором, який входить до складу IDE, що використовується при створені скриптів, в тому числі і в системі *Spyder*. На Рис. 7.1. наведений скрин екрану комп'ютеру, що відображає вміст створеного файлу в редакторі IDE *Spyder*.

В прикладі за замовчуванням в якості роздільника використовувався символ «кома». Звісно, можна використати і інші символи. В наступному прикладі використаємо в якості роздільника символ «крапка з комою», і принагідно нагадаємо, як можна результат виконання операції виводу замість запису в файл перенаправляти в консоль *Python*-інтерпретатора, де цей результат можна побачити (Рис. 7.2).

```
import sys
result.to_csv(sys.stdout, sep=';')
>>>>
```

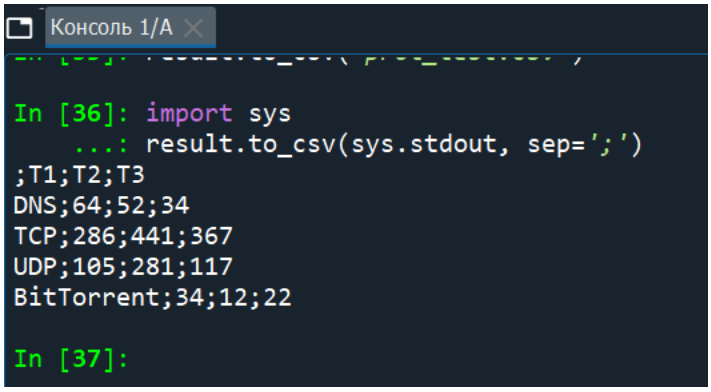


Рис. 7.2 – Консоль інтерпретатора при виконанні скрипту з перенаправленням результатів

В разі, якщо значення індексу не мають бути збережені (найбільш часто так може відбуватися в разі, коли дані індексуються за допомогою індекс-позиції) це вказується шляхом використання значення параметру index=False. Результат показаний на Рис. 7.3.

```
result.to_csv(sys.stdout, sep=';',index=False)
>>>>
```

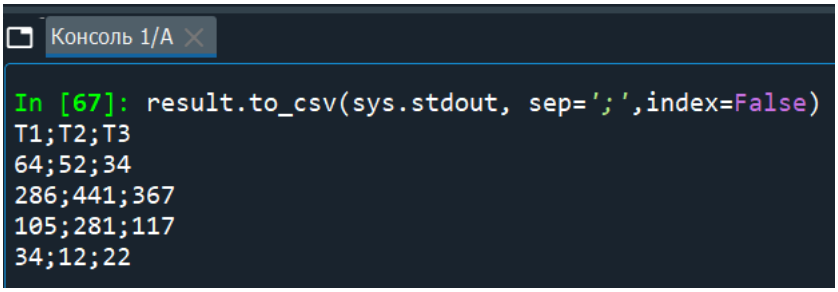


Рис.7.3 – Вміст файлу в разі відмови від збереження індексу *DataFrame*

У об’єктів типу *Series* також мається метод *to_csv()* для безпосереднього запису інформації в файл. Використовуючи інструменти, що розглядалися в попередніх главах, створимо об’єкт типу *Series*, заповнимо його випадковими числами, використаємо послідовність часових міток в якості індексу значень нашого об’єкту та запишемо отриману структуру у файл:

```
dates=pd.date_range('1/11/2022', periods=7)
print (dates)
>>>>
DatetimeIndex(['2022-01-11', '2022-01-12', '2022-01-13', '2022-01-14',
 '2022-01-15', '2022-01-16', '2022-01-17'],
 dtype='datetime64[ns]', freq='D')
time_series = pd.Series(np.random.rand(7), index=dates)
print (time_series)
>>>>
2022-01-11 0.929616
2022-01-12 0.316376
2022-01-13 0.183919
2022-01-14 0.204560
2022-01-15 0.567725
2022-01-16 0.595545
2022-01-17 0.964515
Freq: D, dtype: float64
time_series.to_csv('time_series.csv')
```

Тепер, виключно заради перевірки та знайомства з описаними вище можливостями, виконаємо зворотню дію, а саме читання даних з створеного файлу ‘time_series.csv’ та перетворення отриманих значень у об’єкт типу *Series*.

```
time_series_cont=pd.read_csv('time_series.csv',parse_dates=True,squeeze=True,
index_col=0)
print(time_series_cont)
>>>>
2022-01-11 0.929616
2022-01-12 0.316376
2022-01-13 0.183919
2022-01-14 0.204560
2022-01-15 0.567725
2022-01-16 0.595545
2022-01-17 0.964515
Name: 0, dtype: float64
```

7.2. Формат обміну даними JSON

Окрім формату *csv* в останні роки великої популярності набув інший формат обміну даних, а саме формат *JSON* (JavaScript Object Notation). Однією з головних особливостей цього формату є гнучкість, а також те, що структура даних у цьому форматі може бути не схожою на звичні таблиці, а отже може використовуватися для даних, семантична структура яких відмінна від традиційної таблиці. В останні роки використання цього формату стало дуже популярним серед програмістів, мабуть завдяки тому, що він став основним форматом обміну даними за протоколом *HTTP* між веб-сервером та браузером або іншим клієнтським застосунком. Цей формат має куди більшу гнучкість, ніж табличний формат типу *csv*.

Так само як і *csv*-файли, *JSON*-файли по суті представляють деяку «надбудову» над звичайними текстовими файлами. Якщо в *csv*-форматі така надбудова передбачає, що в файлі мають зберігатися дані, розділені між собою наперед визначеним символом, то структура JSON складніша. Дані, які представлені у форматі *JSON* (і зберігаються у файлі з розширенням *.json*) нагадують код опису об’єктів на *Python* з деякими незначними відмінностями (наприклад, забороняється ставити кому після останнього елемента списку, відсутнє значення позначається *null* тощо). Базовими типами є об’єкти (словники), масиви (списки), строки, числа, булеві значення та *null*. Ключ будь-якого об’єкта має бути строковим.

На *Python* існує кілька бібліотек для читання і запису *JSON*-даних. Ми розглянемо роботу вбудованими в *pandas* методами, що спрощують роботу з *JSON*-файлами при використанні *DataFrame* в якості відповідного об’єкту застосунку.

При роботі з *JSON* об’єкти, що використовуються програмою на мові *Python,* упаковуються у байтову послідовність, а сама операція такої упаковки носить назву *серіалізації*. А розпакування байтової послідовності в об’єкти мови *Python*, приведення послідовності байт назад до типів та структур *Python* — операцією *десеріалізації*. Саме послідовність байт забезпечує необхідну незалежність даних, що передаються, як від середовища передачі, так і від особливостей реалізації застосунків, що обмінюються інформацією за допомогою серелізованих потоків байтів.

Для того, щоб зрозуміти, як відбувається робота з *JSON*-об’єктом, звернемося до прикладів з попереднього розділу, та розглянемо ще раз об’єкт

result. Записуємо зміст цього об’єкту в файл та ознайомимося з змістом цього файлу за допомогою будь якого редактору текстових файлів.

```
result.to_json(‘result_json.json’)
>>>>
{"T1":{"DNS":64,"TCP":286,"UDP":105,"BitTorrent":34},"T2":{"DNS":52,"TCP":441,"UDP":28
1,"BitTorrent":12},"T3":{"DNS":34,"TCP":367,"UDP":117,"BitTorrent":22}}
```

Видно, що створений *JSON*-об’єкт представляє собою структуру, подібну до вкладених словників *Python*. Ключ кожного елемента – заголовок стовпчика об’єкту *DataFrame*, з якого створювався спеціалізований об’єкт, значення – інший словник, що складається з значень елементів відповідного стовпчика. При цьому ключі в цих вкладених словниках відповідають індексам нашого *DataFrame.* Відповідно, найпростіший спосіб прочитати дані, що вже представлені в вигляді *json*-файлу (об’єкту) - це прочитати їх за допомогою функції *pandas.read_json().* Виконаємо вказані дії та впевнимося, що серіалізація та десеріалізація відбулися коректно.

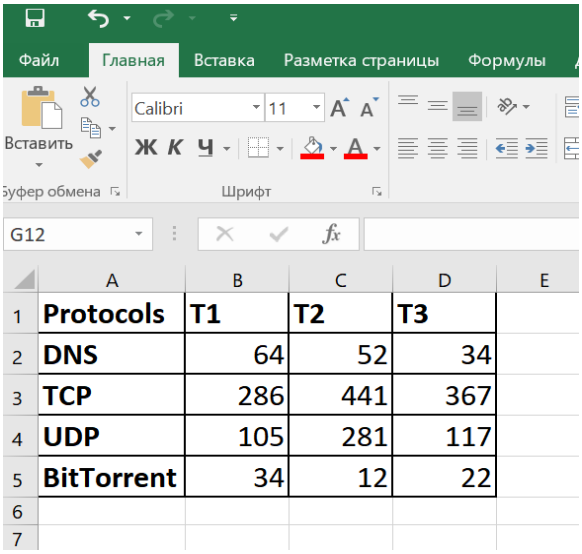
```
result_from_json=pd.read_json(‘result_json.json’)
print (result_from_json)
>>>>
              T1    T2    T3
DNS           64    52    34
TCP           286   441   367
UDP           105   281   117
BitTorrent    34    12    22
```

7.3. Формат xlsx та обмін даними з електронними таблицями

Ще один формат даних, який широко використовується при програмуванні на мові *Python* взагалі, та при роботі з модулем *pandas* зокрема – це формат, що використовується широковживаним застосунком *MS Excel*. Звернемося ще раз до файлу з інформацією про кількість пакетів певного протоколу, що пройшли мережею, але в даному разі попередньо трансформуємо файл так, щоб він був представлений в форматі *.xlsx.* Файли такого формату можуть бути відкриті та оброблені застосунком *MS Excel* та його клонами з пакетів *LibreOffice, OpenOffice, NeoOffice* тощо, і спроба прочитати вміст файлу безпосередньо базовими засобами *Python* для їх подальшої обробки призведе до помилки.

Якщо відкрити цей файл в *MS Excel* то можна побачити наступне(Рис. 7.4):

Застосування функції *pandas.read_excel()* дає можливість читання вмісту файлу в об’єкт типу *DataFrame.*



	A	B	C	D	E
1	**Protocols**	**T1**	**T2**	**T3**	
2	**DNS**	64	52	34	
3	**TCP**	286	441	367	
4	**UDP**	105	281	117	
5	**BitTorrent**	34	12	22	
6					
7					

Рис. 7.4 – Таблиця з даними в застосунку MS Excel

```
result_from_excel=pd.read_excel('prot_test.xlsx')
print(result_from_excel)
>>>>
 Protocols T1 T2 T3
0 DNS 64 52 34
1 TCP 286 441 367
2 UDP 105 281 117
3 BitTorrent 34 12 22
```

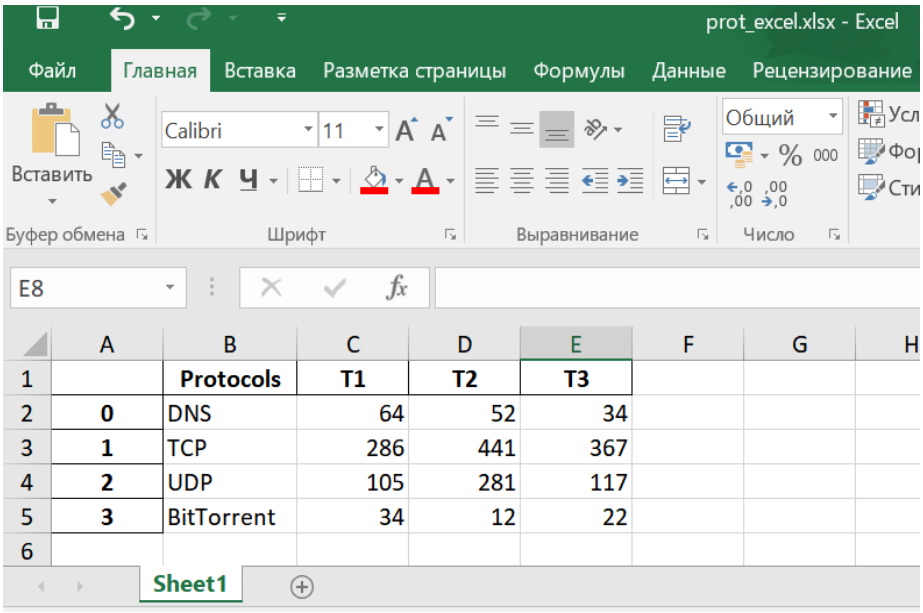
В результаті виконання коду було прочитано вміст першого аркушу вказаного файлу та виконано запис в об’єкт типу *DataFrame*, що створюються. В разі якщо файл-джерело інформації має в своєму складі декілька аркушів, програміст може явним чином вказати, з якого саме аркушу треба отримати інформацію. Для цього використовується параметр *sheet_name.* Крім того, оскільки іншої інформації не задано, по замовчуванню назви стовпчиків об’єкту *DataFrame* беруться з першого рядочка таблиці.

Як і розглянута вище функція *pd.read_csv()*, функція *pd.read_excel()* дозволяє явним чином задавати назви стовпчиків, типи даних для значень стовпчиків, та індекси *DataFrame*, що створюється. Всі параметри функції *pd.read_csv()* можуть використовуватися і в функції *pd.read_excel().*

Запис інформації в файл формату *.xlsx* виконується за допомогою методу *pd.to_excel()* об'єкту *DataFrame*. Наступний код записує дані, що були щойно отримані, в файл, який при цьому створюється.

```
result_from_excel.to_excel('prot_excel.xlsx')
```

Якщо спробувати "зазирнути" у створений файл за допомогою менеджеру файлів *IDE Spyder*, буде запущений (в разі його присутності на комп'ютері) застосунок *MS Excel*, до якого користувач і буде перенаправлений. Після виконання коду на екрані застосунку можна буде побачити наступне відображення вмісту створеного *.xlsx*-файлу. Для перевірки можна подивитися вміст створено в результаті виконання скрипту файлу (Рис. 7.5).



	A	B	C	D	E	F	G	H
1		**Protocols**	**T1**	**T2**	**T3**			
2	**0**	DNS	64	52	34			
3	**1**	TCP	286	441	367			
4	**2**	UDP	105	281	117			
5	**3**	BitTorrent	34	12	22			
6								

Рис. 7.5 – Вміст файлу, створеного в результаті виконання скрипту при його перегляді в застосунку MS Excel

Зауважте, що за замовченням дані записуються на аркуш з назвою «*sheet1*», навіть якщо у користувача встановлена не англомовна версія застосунку. Очікувано, назву аркушу можна явно задати, вказавши необхідне значення параметру *sheet_name*.

Якщо виникає необхідність записати більше, ніж один аркуш у файл, зробити це можливо з використанням *ExcelWriter*–об'єкту, обробка якого доступна в модулі *pandas*.

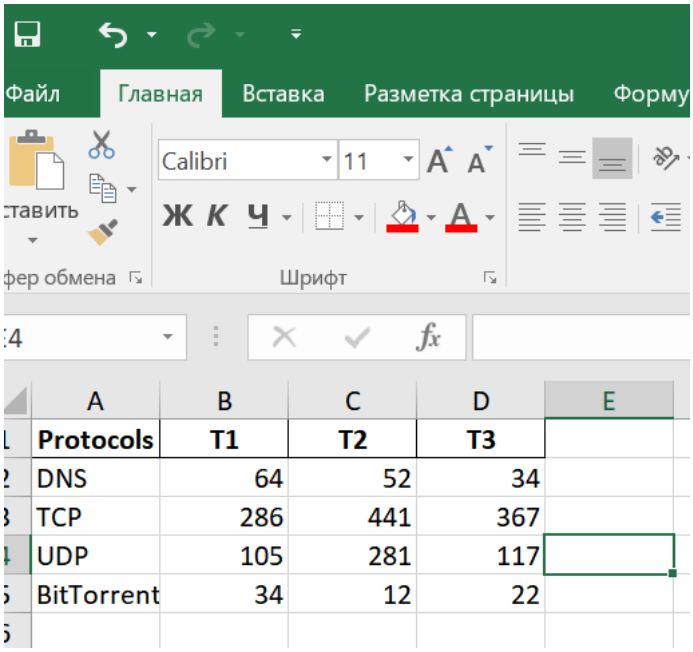
```
result_from_excel2 = result_from_excel.copy()
with pd.ExcelWriter('prot_excel.xlsx') as writer:
 result_from_excel.to_excel(writer, sheet_name='Аркуш_1')
 result_from_excel2.to_excel(writer, sheet_name='Аркуш_2')
```

Цей самий метод можна використати для додавання нового аркушу до файлу, який вже існує:

```
with pd.ExcelWriter('prot_excel.xlsx',mode='a') as writer:
 result_from_excel.to_excel(writer, sheet_name='Аркуш_3')
```

Якщо в якості значення параметру *index* задати *False*, як видно з Рис. 7.6 стовпчик індексу в файлі створюватися не буде:

```
result_from_excel.to_excel('prot_excel.xlsx',index=False)
>>>>
```



	A	B	C	D	E
1	**Protocols**	**T1**	**T2**	**T3**	
2	DNS	64	52	34	
3	TCP	286	441	367	
4	UDP	105	281	117	
5	BitTorrent	34	12	22	
6					

Рис. 7.6 – Вміст створеного файлу при відмові від використання індексу

Можна помітити, що вміст створеного файлу дещо відрізняється від вмісту файлу, який використовувався в якості джерела. По-перше, змінилася форматування аркушу, найбільш значимі зміни – відсутність кордонів комірок та зміна ширини та висоти стовпчика. Відновлення або заміна формату відображення таблиці не входить в можливості, що їх надає *pandas*. Для роботи з форматуванням таблиць *xlsx*-формату, а також для роботи з окремими комірками

таблиць використовуються спеціальні бібліотеки, зокрема *openpyxl*. Знайомство з цими бібліотеками не входить в круг завдань, що розглядаються в даному навчальному посібнику.

Контрольні питання

1. В чому відмінність даних, представлених в файлах формату *txt* від даних в форматі csv?

2. Чим відрізняються між собою представлення даних в форматах *csv* та *JSON* ?

3. Якими програмними інструментами можна прочитати вміст файлу, що представлений в форматі *xlsx*?

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Atwan T. Time Series Analysis with Python Cookbook / Atwan Tarek. – Birmingham, UK : Packt Publishing, Limited, 2022. – 630 p. ISBN: 978-1-80107-554-1.

2. Auffarth B. Machine Learning for Time-Series with Python: Forecast, predict, and detect anomalies with state-of-the-art machine learning methods / Auffarth Ben. – Birmingham, UK : Packt Publishing, Limited, 2021. – 371 p. ISBN 978-1-80181-962-6.

3. Brownlee J. Introduction to Time Series Forecasting with Python: How to Prepare Data and Develop Models to Predict the Future Series / Brownlee Jason. – UK : Machine Learning Mastery, 2020. – 365 p.

4. Haslwanter T. An Introduction to Statistics with Python / Haslwanter Thomas. – Switzerland : Springer International Publishing AG, 2016. – 285 p. ISBN 978-3-319-28315.

5. Johansson R. Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib / Johansson Robert. – New York, USA : Apress Media LLC, 2018. – 700p. ISBN 978-1-4842-4245-2.

6. Lazzeri F. Machine Learning for Time Series Forecasting with Python / Lazzeri Francesca. – Indianapolis, Indiana, USA : John Wiley & Sons, Inc., Indianapolis, Indiana, 2021. – 216 p. ISBN: 978-1-119-68236-3.

7. Mather B. Time Series with Python: How to Implement Time Series Analysis and Forecasting Using Python / Mather Bob. – USA : Publisher: Bob Mather, 2020. – 222 p. ISBN: 978-0-648-78308-4.

8. Peixeiro M. Time Series Forecasting in Python / Peixeiro Marco. – Shelter Island, NY, USA : Manning Publications Co., 2022. – 458 p. ISBN: 978-1-617-29988-9.

9. Vishwas B. Hands-on Time Series Analysis with Python: From Basics to Bleeding Edge Techniques / Vishwas B., Patel A. – New York, USA : Apress Media LLC, 2020. – 420 p. ISBN: 978-1-4842-5992-4.

ПРЕДМЕТНИЙ ПОКАЗЧИК