

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу**

«На правах рукопису»
УДК 004.942:519.254

До захисту допущено
Завідувач кафедри ММСА
_____ О.Л. Тимощук
«____» ____ 20__ р.

**Магістерська дисертація
на здобуття ступеня магістра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 "Системний аналіз"
на тему: «Методика формування, реалізації та ефективного управління
інвестиційним портфелем»**

Виконав:

Студент II курсу, групи КА-01мп
Бевз В'ячеслав Анатолійович

Керівник:

професор кафедри ММСА,
д.т.н., проф. Бідюк П.І.

Рецензент:

декан ФІОТ КПІ ім. Ігоря Сікорського,
д.т.н., проф. Теленик С.Ф.

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних посилань.

Студент _____

Київ
2021

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – другий (магістерський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри ММСА

_____ О.Л. Тимошук

«___» _____ 20__р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Бевзу В'ячеславу Анатолійовичу

1. Тема роботи «Методика формування, реалізації та ефективного управління інвестиційним портфелем», керівник роботи професор, д.т.н. Бідюк Петро Іванович, затверджені наказом по університету від «02» листопада 2021 р. № 3651-с
2. Термін подання студентом дисертації 10 грудня 2021 року.
3. Об'єкт дослідження: Аналіз інвестиційних якостей цінних паперів, економічні взаємовідносини учасників ринку цінних паперів, а також процеси, пов'язані з формуванням та управлінням інвестиційним портфелем.
4. Предмет дослідження: Інструментарій для розробки ресурсу і методи формування та управління інвестиційним портфелем.
5. Зміст роботи:
 - 1) Аналіз актуальності задачі моделювання та побудови ПП;
 - 2) Вибір типів математичних моделей для опису ПП;
 - 3) Побудова СППР для виконання обчислювальних експериментів.
6. Орієнтовний перелік графічного (ілюстративного) матеріалу:

- 1) Приклади результатів вагів побудованих портфелів (рис.);
- 2) Графіки оптимальних портфелів (рис.);
- 3) Таблиці у розділі стартап-проекту та порівняльні таблиці результатів.
- 4) Графіки тікетів (рис.);

7. Дата видачі завдання 01 вересня 2021 року.

Календарний план

№ з/п	Назва етапів виконання магістерській дисертації	Термін виконання етапів роботи	Примітка
1	Формулювання тематики (напрямку) дослідження.	01.09.2021 – 03.09.2021	Виконано
2	Аналіз актуальності задач стосовно тематики дослідження	04.09.2021 – 10.09.2021	Виконано
3	Формулювання задач дослідження	01.09.2021 – 10.09.2021	Виконано
4	Уточнення теми магістерській дисертації	20.09.2021	Виконано
5	Збір статичних даних, попередній аналіз даних	01.10.2021 – 11.10.2021	Виконано
6	Розробка програмного продукту	01.10.2021 – 18.11.2021	Виконано
7	Виконання обчислювальних експериментів, аналіз результатів	19.11.2021 – 30.11.2021	Виконано
8	Оформлення пояснювальної записки у цілому	01.12.2021 – 06.12.2021	Виконано
9	Підготовка презентації для захисту	06.12.2021 – 10.12.2021	Виконано
10	Попередній захист магістерській дисертації	13.12.2021 – 16.12.2021	Виконано
11	Захист магістерської дисертації	17.12.2021 – 24.12.2021	Виконано

Студент

(підпис)

В. А. Бевз
(ініціали, прізвище)

Керівник роботи

(підпис)

П. І. Бідюк
(ініціали, прізвище)

РЕФЕРАТ

Магістерська дисертація: 195 с., 42 рис., 2 табл., 1 додаток, 30 джерел.

Об'єкт дослідження: аналіз інвестиційних якостей цінних паперів, економічні взаємовідносини учасників ринку цінних паперів, а також процеси, пов'язані з формуванням та управлінням інвестиційним портфелем.

Мета роботи: Сформувати принципи інвестиційного портфеля для довгострокових і короткострокових фінансових завдань; оцінювання портфельів; розробка програмного забезпечення для виконання обчислювальних експериментів.

Метод дослідження: математичні моделі і методи аналізу процесів в економіці та фінансах.

В роботі проведений огляд існуючих методів прогнозування. Наведено моделі інтелектуального аналізу даних. Розглянуто і проаналізовано статистичні тести для аналізу.

Створено програмне забезпечення для інвесторів, в залежності від його фінансових цілей і підходу до інвестицій. Дозволяє вносити свої зміни в існуючий портфель і/або сформувати готовий інвестиційний портфель. В роботі порівняно методи оптимізацій.

Система реалізована на базі платформи PyCharm з використанням мови програмування Python та Jupyter, наведено приклади застосування програми для прогнозування та оптимізації ваг, ризиків, доходностей інвестиційних. Розглянуто шляхи можливого подальшого вдосконалення системи.

ІНВЕСТИЦІЙНИЙ ПОРТФЕЛЬ, ІНВЕСТИЦІЇ, АКЦІЇ, ЦІННІ ПАПЕРИ, BLACK LITTERMAN, ЕКОНОМІКА, ФОНДИ, ІНДЕКСИ, ВАРТІСТЬ, ПРИБУТКОВІСТЬ.

ABSTRACT

Master's thesis: 195 p., 42 fig., 2 tabl., 1 appendix, 30 sources.

Object of research: analysis of investment qualities of securities, economic relations of securities market participants, as well as processes related to the formation and management of the investment portfolio.

Objective: To form the principles of the investment portfolio for long-term and short-term financial tasks; portfolio valuation; development of software for computational experiments.

Method of research: mathematical models and methods of analysis of processes in economics and finance.

The paper reviews the existing forecasting methods. Models of data mining are presented. Statistical tests for analysis are considered and analyzed.

Software for investors has been created, depending on its financial goals and approach to investment. Allows you to make changes to the existing portfolio and / or create a ready-made investment portfolio. The methods of optimization are compared in the work.

The system is implemented on the basis of the PyCharm platform using the Python and Jupyter programming languages, examples of application of the program for forecasting and optimization of weights, risks, return on investment are given. Ways of possible further improvement of the system are considered.

INVESTMENT PORTFOLIO, INVESTMENTS, SHARES, SECURITIES, BLACK LITTERMAN, ECONOMY, FUNDS, INDICES, VALUE, PROFITABILITY.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ	8
ВСТУП	9
РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ФОРМУВАННЯ ІНВЕСТИЦІЙНОГО ПОРТФЕЛЯ	10
1.1 Обробка історичних цін	10
1.2 Розрахунок Дохідності	10
1.3 Розрахунок Коваріації	12
1.3.1 Середнє значення вибірки	13
1.3.2 Метод усадки Ledoit Wolf	14
1.4 Оптимізація середньої дисперсії	18
1.5 Загальний ефективний кордон	20
1.5.1 Ефективна напівваріантність	21
1.5.2 Efficient CVaR	23
1.5.3 Efficient CDaR	25
1.6 Ставлення стандартного відхилення до ризику	26
1.7 Модель Black-Litterman (BL)	27
1.8 Висновки до розділу	31
РОЗДІЛ 2 ОБГРУНТУВАННЯ ОБ'ЄКТІВ ІНВЕСТУВАННЯ	33
2.1 Аналіз інвестицій з точки зору потенційного ризику і прибутковості	33
2.1.1 Фундаментальний аналіз	33
2.1.2 Технічний аналіз	34
2.2 Аналіз об'єктів інвестування	35
2.3 Висновки до розділу	53
РОЗДІЛ 3 АНАЛІЗ І ПОБУДОВА ІНВЕСТИЦІЙНОГО ПОРТФЕЛЯ.	
ОСНОВНІ РЕЗУЛЬТАТИ	55
3.1 Моделі Повернення Ризику	55
3.2 Оптимізація Середньої Дисперсії	59
3.2.1 Дохідність портфеля	63
3.2.2 Long/short min variance	64

	7
3.2.3 Max Sharpe з обмеженнями по сектору	66
3.2.4 Максимізація прибутку для даного ризику з регуляризацією L2	69
3.2.5 Мінімізація ризику для даної прибутковості, ринково нейтральний (market- neutral)	73
3.2.6 Ефективна оптимізація напівдисперсій (semi-variance)	74
3.2.7 Ефективна оптимізація CVaR	77
3.2.8 Побудова графіка - без обмежень (plot - Unconstrained)	79
3.2.9 Побудова графіка - з обмеженнями (plot - Constrained)	81
3.3 Розподіл Black-Litterman	84
3.3.1 Попередні оцінки	90
3.3.2 Розподіл портфеля	93
3.4 Висновки до розділу	95
РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЕКТУ	97
4.1 Опис ідеї проекту	97
4.2 Технологічний аудит ідеї проекту	100
4.3 Аналіз ринкових можливостей запуску стартап-проекту	101
4.4 Розроблення ринкової стратегії проекту	109
4.5 Розроблення маркетингової програми стартап-проекту	112
4.6 Висновки до розділу	116
ВИСНОВКИ	118
ПЕРЕЛІК ПОСИЛАНЬ	120
ДОДАТОК А ЛІСТИНГ ПРОГРАМИ	123

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

MVO - оптимізації середньої дисперсії

CAGR - середнє геометричне значення

CAPM - Модель ціноутворення на капітальні активи

IR - Коефіцієнт попередньої інформації

CVaR - Умовна вартість, схильний до ризику

CDaR- Умовно просадка ризику

ВСТУП

Дана робота присвячена вивченню та створенню інвестиційного портфеля для пасивного інвестора (з використанням сучасної теорії портфеля), диверсифікації та ретельного вибору активів.

Актуальність дослідження. Актуальність даної теми підтверджується поточною ситуацією на фінансовому ринку України та фінансової неграмотності більшості людей. Інвестори весь час намагаються оптимізувати і збалансувати свої портфелі, щоб періоди криз проходили найбільш гладко, а за часів підйомів економіки максимально зберегти, а то і примножити свої вклади. Інвестори можуть використовувати методи математичного аналізу і способи прийняття рішення для вкладення грошових коштів, ґрунтуючись на попередніх фінансових документах, які можуть допомогти зорієнтуватися в процесах фінансових ринках.

Мета дослідження. Сформувати принципи інвестиційного портфеля для довгострокових і короткострокових фінансових завдань. Використовування західної теорії портфельного управління для українських сучасних реалій. Сформування принципів управління портфелем.

Об'єкт дослідження. Об'єктом дослідження є аналіз інвестиційних якостей цінних паперів, економічні взаємовідносини учасників ринку цінних паперів, а також процеси, пов'язані з формуванням та управлінням інвестиційним портфелем.

Предмет дослідження. Предметом дослідження є взаємовідносини, які виникають в результаті формування і управління інвестиційним портфелем цінних паперів для забезпечення функціонування та їх ефективності.

Метод дослідження. Теоретичний розділ даної роботи, побудований на основі аналізу основних положень теорії сучасного портфеля і операцій на ринку цінних паперів.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ФОРМУВАННЯ ІНВЕСТИЦІЙНОГО ПОРТФЕЛЯ

1.1 Обробка історичних цін

Оптимізація середньої дисперсії вимагає двох речей: очікуваного прибутку активів та коваріаційної матриці (або, загальніше, моделі ризику, що визначає кількісну оцінку ризику активу). Для підрахунку цих параметрів будемо використовувати `expectedReturns` та `riskModels` відповідно. Користувачу потрібно надати набір історичних цін на його активи. Це може бути або файл в форматі `csv` або можемо загрузити дані з ресурсу `Yahoo finance`. Індекс повинен складатися з дат або позначок часу, а кожен стовпець повинен представляти часовий ряд цін на актив. Також потрібно зауважити, що дані про ціни не обов'язково повинні бути щоденними, але частота повинна бути однаковою для всіх активів.

1.2 Розрахунок Дохідності

Після того як ми отримали історичні ціни в `pandas dataframe` можемо вибрати як будемо рахувати доходність між "Оцінка середньої прибутковості" та "Оцінка прибутку CAPM".

Оцінка середньої прибутковості

Ключові визначення:

1. Середня прибутковість (також відома як очікувана прибуток) - це приблизний прибуток або збиток, який інвестор очікує від портфеля інвестицій.

2. Він також може посилається на щомісячну прибутковість акцій або середнє значення розподілу ймовірностей можливих доходів.
3. Інвестор може розрахувати середню рентабельність інвестицій, враховуючи історичну прибутковість інвестицій або ймовірні норми прибутку за різних сценаріїв.
4. Розрахунок середньої дохідності може допомогти інвестору визначити кількісну залежність між ризиком портфеля цінних паперів та потенційним доходом.
5. Однак середня прибутковість не гарантує майбутньої норми прибутку і є лише одним інструментом, який інвестор повинен враховувати, оцінюючи інвестиції, перш ніж купувати їх.

Обчислюємо середньорічну (щоденну) історичну прибутковість від вхідних (добових) цін на активи. Використовуємо середнє геометричне значення (CAGR) - сукупний середньорічний темп зростання. Виражається у відсотках і показує, на скільки відсотків за рік приростає досліджуваний параметр.

Розраховується як:

$$CAGR(t_0, t_n) = \left(\frac{V(t_n)}{V(t_0)} \right)^{\frac{1}{t_n - t_0}} - 1,$$

де

$V(t_0)$ - початкове значення параметра;

$V(t_n)$ - кінцеве значення параметра;

$t_n - t_0$ - кількість років.

Здебільшого, це метод за замовчуванням. Він інтуїтивно зрозумілий і легко інтерпретується, проте оцінки піддаються великій невизначеності. Це проблема, особливо в контексті оптимізатора середньої дисперсії, який максимально збільшить кількість помилкових вхідних даних.

Оцінка прибутку CAPM.

Модель ціноутворення на капітальні активи (CAPM) описує зв'язок між систематичним ризиком та очікуваною прибутковістю активів, зокрема акцій. CAPM широко використовується у фінансах для ціноутворення на ризиковані цінні папери та створення очікуваної прибутковості активів, враховуючи ризик цих активів та вартість капіталу. Формула очікуваного прибутку активу з урахуванням його ризику

$$E R_i = R_f + \beta_i (E R_M - R_f),$$

де

$E R_i$ = очікувана віддача інвестицій;

R_f = безризикова ставка (за замовчуванням 0,02);

β_i = бета інвестицій;

$(E(R_M - R_f))$ = премія ринкового ризику.

1.3 Розрахунок Коваріації

На додаток до очікуваного прибутку, оптимізація середньої дисперсії вимагає моделі ризику, певного способу кількісної оцінки ризику активів. Найбільш часто використовуваною моделлю ризику є коваріаційна матриця, яка описує мінливість активів та їх співзалежність. Це важливо, оскільки один із принципів диверсифікації полягає в тому, що ризик можна зменшити, зробивши багато некорельованих ставок (кореляція - це лише нормалізована коваріація), див. рис.1.1

$$\text{cov}(\hat{A}) = \sigma_u^2 (X^T X)^{-1} = \begin{bmatrix} \sigma_{a_1}^2 & \sigma_{a_1 a_2} & \dots & \sigma_{a_1 a_j} & \dots & \sigma_{a_1 a_m} \\ \sigma_{a_2 a_1} & \sigma_{a_2}^2 & \dots & \sigma_{a_2 a_j} & \dots & \sigma_{a_2 a_m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \sigma_{a_j a_1} & \sigma_{a_j a_2} & \dots & \sigma_{a_j}^2 & \dots & \sigma_{a_j a_m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \sigma_{a_m a_1} & \sigma_{a_m a_2} & \dots & \sigma_{a_m a_j} & \dots & \sigma_{a_m}^2 \end{bmatrix}$$

Рисунок 1.1 - Приклад коваріаційної матриці

Багато в чому предмет моделей ризику є набагато важливішим, ніж очікуваний прибуток, оскільки історична дисперсія, як правило, є набагато більш стійкою статистикою, ніж середня історична прибутковість. Фактично, дослідження Kritzman et al. (2010) припускає, що портфелі мінімальних дисперсій, сформовані шляхом оптимізації без надання очікуваної прибутковості, насправді набагато краще працюють за вибіркою.

Проблема, однак, полягає в тому, що на практиці у нас немає доступу до матриці коваріацій (так само, як у нас немає доступу до очікуваної віддачі) - єдине, що ми можемо зробити, це зробити оцінки на основі минулих даних. Найбільш простий підхід-це просто розрахувати вибіркoву коваріаційну матрицю на основі історичних результатів, але порівняно недавні (після 2000 р.) Дослідження показують, що існують набагато більш надійні статистичні оцінки коваріаційної матриці.

Розрахунок Коваріації здійснюється двома методами: “Середнє значення вибірки”, та “Метод усадки Ledoit Wolf”.

1.3.1 Середнє значення вибірки

Обчислює щорічну вибіркoву матрицю коваріацій (щоденних) повернень активів.

Це підхід за замовчуванням. Записи у вибірковій матриці коваріацій (яку ми позначаємо як S) є вибірковими коваріаціями між i -м та j -м активами (діагоналі складаються з дисперсій). Хоча вибірка коваріаційна матриця є неупередженою оцінкою коваріаційної матриці, тобто $E(S)=\Sigma$, на практиці вона страждає від помилки в специфікації та відсутності надійності. Це особливо проблематично при оптимізації середньої дисперсії, оскільки оптимізатор може надавати додаткову довіру помилковим значенням.

1.3.2 Метод усадки Ledoit Wolf

Метод усадки Ledoit Wolf : Обчислює оцінку усадки Ледойта-Вольфа для конкретної цілі усадки.

Оптимізація середньої дисперсії (MVO), запропонована Марковіцем, дає теоретичні гарантії та може бути кодована для дуже ефективної роботи, але потребує хороших вихідних даних. Зокрема, для цього потрібна хороша модель ризику, тобто хороша оцінка коваріації.

Вибіркова коваріація є вибором за замовчуванням, але часто має коефіцієнти з надзвичайними помилками, які є особливо небезпечними для MVO, оскільки оптимізатор, ймовірно, здійснюватиме великі розподіли на основі цих коефіцієнтів. Одним із можливих поліпшень є переміщення екстремальних значень до центру в процесі, який називається усадка.

Портфель інвестора можна розглядати як позицію у еталоні (з ваговим вектором w_B) та додаткову позицію в певних акціях - активні ваги x (також вектор). Очевидно, тому що ваги портфеля (w_p) сума до одного, маємо

$$w_p = w_B + x$$

Очікувана надлишкова прибутковість акцій визначається середньою прибутковістю портфеля за вирахуванням прибутковості з контрольної точки.

$$\alpha = \mu - w_B^T \mu$$

Проблему оптимізації можна викласти так (де Σ позначає матрицю коваріацій та $\mathbf{1}$ є відповідним вектором одиниць):

$$\begin{array}{ll} \underset{x}{\text{minimize}} & x^T \Sigma x \\ \text{subject to} & x^T \alpha \geq g \\ & x^T \mathbf{1} = 0 \\ & x \geq -w_B \\ & x \leq c\mathbf{1} - w_B \end{array}$$

Обмеження цієї оптимізації відповідно інтерпретуються так:

Активний прибуток має перевищувати певну цільову вигоду g , отож

$$x^T \alpha \geq g$$

Активні ваги повинні в суммі довінювати нулю (для w_p та w_B)

Портфоліо може бути лише Long, тому

$$w_p \geq 0 \Rightarrow x \geq -w_B$$

Максимальний розмір позиції - c (наприклад, ми не можемо мати більше 10% даного запасу), тому

$$w_p \leq c\mathbf{1}.$$

Оцінка усадки вибіркової коваріаційної матриці:

Вибіркову коваріацію легко обчислити і вона є неупередженою оцінкою, але, як правило, виникає велика кількість помилок оцінки, особливо коли точок даних про ціну менше, ніж є запасів (тобто $N \geq T$).

З іншого боку, існують високоструктуровані оцінювачі, такі як однофакторна модель Шарпа (Шарп 1963), які можуть бути дуже упередженими, але мають невелику похибку оцінки.

Принцип усадки полягає в компромісі між деякими високоструктурованими оцінювачами F (також відома як мета усадки) та іншим неструктурованим оцінювачем.

У разі коваріації неструктурований оцінювач є однозначно матрицею коваріації вибірки S .

Компроміс між цими двома оцінками визначається коефіцієнтом усадки δ .

Тоді остаточна оцінка набуває вигляду:

$$\delta F + (1 - \delta)S, 0 \leq \delta \leq 1$$

Мета усадки.

Ціль скорочення має включати лише невелику кількість вільних параметрів (високоструктурованих), але також дає певну інформацію щодо характеру оцінюваної кількості.

У моїй роботі використовується модель постійної кореляції , яка встановлює, що всі кореляції парні імена дорівнюють середнім значенням усіх вибірових кореляцій. Якщо $s_{i,j}^2$ позначає вибірову коваріацію між двома запасами, матрицю цільової усадки F потім подається:

$$\begin{aligned} f_{i,i} &= s_{i,i} \\ f_{i,j} &= \bar{r} \sqrt{s_{i,i} s_{j,j}} \end{aligned}$$

Емпірична оцінка.

Створюється кілька контрольних показників різних розмірів, де вага індексу визначається ринковою капіталізацією.

Щоб імітувати кваліфікованого активного інвестора, випадковий шум додається до реалізованої надмірної віддачі (тобто за допомогою ретроспективності), нормалізованої таким чином, що коефіцієнт попередньої інформації (IR) становить приблизно 1,5.

Використовуючи ефективний кордон ефективності з постійною кореляційною усадкою, та коефіцієнт попередньої інформації (IR) аналіз обчислюється за 50 випробуваннями та порівнюється з іншими моделями ризику:

- вибірккову коваріаційну матрицю;
- скорочення з однофакторною метою Шарпа;
- багатфакторні моделі.

Результати показують, що усадка перевершує Середнє значення вибірки у всіх сценаріях.

Усадка з постійною кореляцією буряків однофакторна усадка для

$$N \leq 225$$

Дослідження показують, що додавання обмеження на дисперсію портфеля, тобто

$$\sigma_p^2 = w_p^T \Sigma w_p,$$

підвищує загальну ефективність.

1.4 Оптимізація середньої дисперсії

Оптимізація середньої дисперсії базується на класичній роботі Гаррі Марковіца 1952 р. , яка очолила трансформацію управління портфелем з мистецтва на науку. Ключовий висновок полягає в тому, що, поєднуючи активи з різними очікуваними прибутками та нестабільністю, можна прийняти рішення про математично оптимальний розподіл.

Математична оптимізація є загалом дуже складною проблемою, особливо коли ми маємо справу зі складними цілями та обмеженнями. Однак опуклі проблеми оптимізації- це добре зрозумілий клас проблем, які виявляються неймовірно корисними для фінансів. Опукла проблема має такий вигляд:

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimise}} & f(\mathbf{x}) \\ \text{subject to} & g_i(\mathbf{x}) \leq 0, i = 1, \dots, m \\ & A\mathbf{x} = \mathbf{b}, \end{array}$$

де $\mathbf{x} \in \mathbb{R}^n$, і $f(\mathbf{x}), g_i(\mathbf{x})$ є опуклими функціями.

Як показано у визначенні опуклої задачі, нам необхідно визначити дві речі: ціль оптимізації та обмеження оптимізації. Наприклад, класична проблема оптимізації портфеля полягає в тому, щоб мінімізувати ризик за умови обмеження повернення (тобто портфель повинен повертати більше, ніж певну суму). З точки зору впровадження, немає особливої різниці між ціллю та обмеженням. Розглянемо подібну проблему, яка полягає в тому, щоб максимізувати прибуток за умови обмеження ризику - тепер роль ризику та прибутку помінялися місцями.

Якщо $\mathbf{w}$ є ваговим вектором акцій з очікуваними прибутками μ , тоді дохідність портфеля дорівнює вазі кожної акції, помноженій на її прибуток, тобто $\mathbf{w}^T \mu$. Ризик портфеля з точки зору коваріаційної матриці Σ надається $\mathbf{w}^T \Sigma \mathbf{w}$.

Оптимізацію портфеля можна розглядати як проблему опуклої оптимізації, і рішення можна знайти за допомогою квадратного програмування. Якщо ми позначимо цільове повернення як μ^* , точна постановка проблеми довгострокової оптимізації портфеля виглядає наступним чином:

$$\begin{array}{ll} \underset{w}{\text{minimise}} & w^T \Sigma w \\ \text{subject to} & w^T \mu \geq \mu^* \\ & w^T \mathbf{1} = 1 \\ & w_i \geq 0 \end{array}$$

Якщо ми змінимо цільову прибутковість, ми отримаємо інший набір ваг (тобто інший портфель) - набір усіх цих оптимальних портфелів називається ефективним кордоном.

Коефіцієнт Шарпа-це дохідність портфеля, що перевищує безризикову ставку, на одиницю ризику (мінливість).

$$SR = \frac{RP - R_f}{\sigma}$$

Це особливо важливо, оскільки він вимірює прибутковість портфеля з поправкою на ризик. Тому на практиці, замість того, щоб намагатися мінімізувати мінливість для певної цільової віддачі (згідно з Марковіцем 1952 р.), Часто має сенс просто знайти портфоліо, яке максимально збільшує коефіцієнт Шарпа. Реалізовано в методі `maxSharpe()`. Здебільшого ваги після оптимізатора можуть бути крихітними. І тому щоб цього уникнути можемо використати `cleanWeights()`. Цей метод, який скорочує крихітні ваги до нуля, а решту округляє. Після чого можемо дізнатись через метод `portfolioPerformance()` параметри нашого портфеля як: Очікуваний річний прибуток, Річна мінливість, Коефіцієнт Шарпа.

Регуляризація L2.

Як обговорювалося в вище, оптимізація середньої дисперсії часто призводить до того, що багато ваг є незначними, тобто ефективний портфель не включає більшість активів. Це очікувана поведінка, але це може бути небажаним, якщо інвестору потрібна певна кількість активів у його портфелі.

Для того, щоб змусити оптимізатор середньої дисперсії створити більш несуттєві ваги, ми додаємо до всіх цільових функцій те, що можна вважати «штрафом за малі ваги», що параметризується γ (gamma). Розглядаючи, наприклад, задачу мінімальної дисперсії, ми маємо:

$$\underset{w}{\text{minimise}} \{w^T \Sigma w\} \longrightarrow \underset{w}{\text{minimise}} \{w^T \Sigma w + \gamma w^T w\}$$

Потрібно зауважити, що $w^T w$ є такою ж, як сума квадратів ваг. Цей термін зменшує кількість незначних ваг, оскільки він має мінімальне значення, коли всі ваги розподілені порівну, і максимальне значення в граничному випадку, коли весь портфель віднесений до одного активу. Називається регуляризацією L2, тому що вона має абсолютно таку ж форму, як термін регуляризації L2 у машинному навчанні, хоча і дещо іншу мету (у ML вона використовується для утримання малих ваг, тоді як тут вона використовується для збільшення їх).

1.5 Загальний ефективний кордон

Методи оптимізації середньої дисперсії, описані раніше, можна використовувати, коли у нас є вектор очікуваної віддачі та коваріаційна матриця. Метою та обмеженнями буде деяке поєднання прибутковості портфеля та нестабільності портфеля.

Однак, можливо, інвестору захочеться побудувати ефективний кордон, див. рис.1.2, для абсолютно іншого типу моделі ризику (такої, що не залежить від коваріаційних матриць), або оптимізувати мету, не пов'язану з дохідністю портфеля (наприклад, помилка відстеження). Було реалізовано кілька популярних альтернатив, що забезпечує підтримку нестандартних проблем оптимізації.

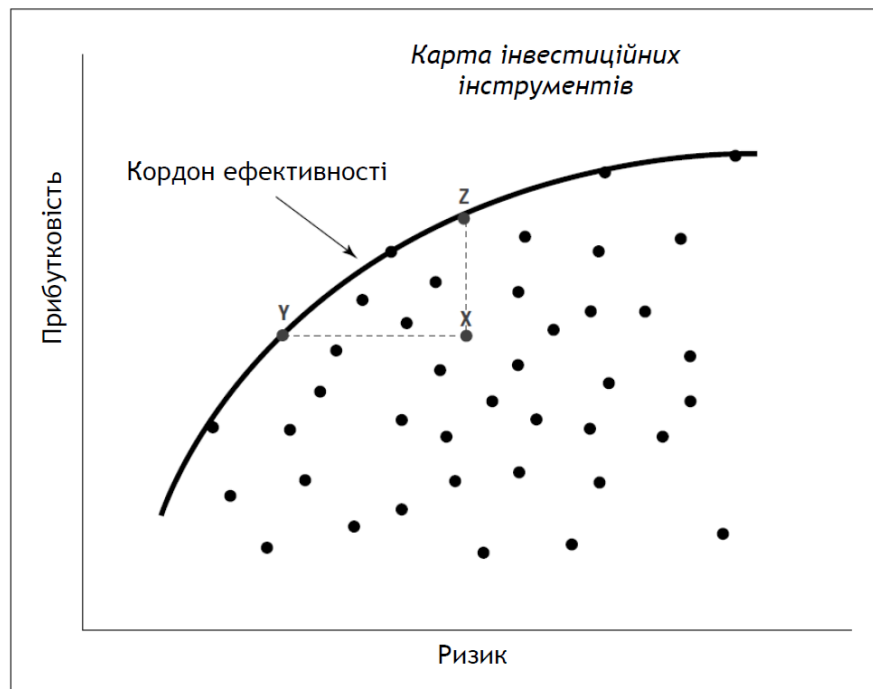


Рисунок 1.2 - Приклад межі ефективності

Кожна точка на цій діаграмі являє собою різне можливе портфоліо, причому чим ближче до кордону ефективності, тим «кращий» портфель (з точки зору коефіцієнта Шарпа). Чорна лінія є ефективним кордоном.

1.5.1 Ефективна напівваріантність

Замість того, щоб карати волатильністю, оптимізація середньої напівваріації прагне лише покарати зменшення мінливості, оскільки може бути бажаною зростання нестабільності.

Існує два підходи до проблеми оптимізації середньої напівваріації. Перший полягає у використанні евристичного (тобто «швидкого та брудного») рішення: вдаючи, що напівваріаційна матриця, є типовою коваріаційною матрицею, і робимо стандартну оптимізацію середньої дисперсії. Можна показати, що це не дає портфеля, ефективного у просторі середньої напівваріації (хоча це може бути досить хорошим наближенням).

На щастя, можна написати оптимізацію середньої напівваріації у вигляді опуклої задачі (хоча і з багатьма змінними), яку можна вирішити, щоб дати «точне» рішення. Наприклад, щоб максимізувати віддачу для цільової напівваріації s^* (тривалий час) ми вирішили б таку проблему:

$$\begin{array}{ll} \underset{w}{\text{maximise}} & w^T \mu \\ \text{subject to} & n^T n \leq s^* \\ & Bw - p + n = 0 \\ & w^T \mathbf{1} = 1 \\ & n \geq 0 \\ & p \geq 0. \end{array}$$

Тут B – це $T \times N$ (B в масштабі) матриця надлишкових доходів. Можуть бути додані додаткові обмеження лінійної рівності та випуклі обмеження нерівності.

Розроблена можливість оптимізувати ефективні межі напівваріантності за допомогою Efficient Semivariance класу. Efficient Semivariance успадковує від класу ефективного кордону, тому має ті самі корисні методи, але знаходить портфоліо на межі середньої напівваріації.

Об'єкти Efficient Semivariance дозволяють оптимізувати на кордоні середньої напівваріації. Це може бути актуальним для користувачів, яких більше турбує відхилення від нижньої сторони.

Створені методи:

1. “Efficient return” - мінімізують напіввідхилення для заданої цільової віддачі.
2. “Efficient risk” - максимальна віддача для цільового напіввідхилення (нижнє стандартне відхилення). Отриманий портфель матиме напіввідхилення менше цільового (але не буде гарантовано аналогічним).

1.5.2 Efficient CVaR

Умовна вартість, схильний до ризику (The conditional value-at-risk), інакше очікуваний дефіцит, є популярною мірою залишкового ризику. CVaR можна розглядати як середнє значення втрат, що виникають у "дуже погані дні", де "дуже погано" визначається кількісно за допомогою параметра β .

Наприклад, якщо обчислити CVaR до 10% для $\beta=0.95$, ми можемо бути впевненими на 95%, що найгірший середньодобовий збиток становитиме 3%. Інакше кажучи, CVaR - це середнє значення всіх втрат, настільки серйозних, що вони тільки відбуваються $(1-\beta)\%$ часу.

Хоча CVaR є досить інтуїтивно зрозумілою концепцією, для її математичного формулювання потрібно багато нових позначень. Ми прийнемо такі позначення:

- w для вектора ваг портфеля;
- r для вектора повернення активів (щоденно) з розподілом ймовірностей $p(r)$;
- $L(w, r) = -w^T r$ за втрату портфеля;
- α для портфеля з ризиком (VaR) з упевненістю β .

Тоді CVaR можна записати так:

$$CVaR(w, \beta) = \frac{1}{1 - \beta} \int_{L(w, r) \geq \alpha(w)} L(w, r) p(r) dr.$$

Це неприємний вираз для оптимізації, оскільки ми по суті інтегруємо значення VaR. Ключове розуміння Рокафеллара та Урясева (2001) [3] полягає в тому, що ми можемо еквівалентно оптимізувати таку опуклу функцію:

$$F_{\beta}(w, \alpha) = \alpha + \frac{1}{1 - \beta} \int [-w^T r - \alpha]^+ p(r) dr,$$

де $[x]^+ = \max(x, 0)$.

Автори доводять, що мінімізація $F_{\beta}(w, \alpha)$ над усім w, α мінімізує CVaR. Припустимо, у нас є вибірка щоденних доходів T (вони можуть бути історичними або імітованими). Інтеграл у виразі стає сумою, тому задача оптимізації CVaR зводиться до лінійної програми:

$$\begin{array}{ll} \underset{w, \alpha}{\text{minimise}} & \alpha + \frac{1}{1 - \beta} \frac{1}{T} \sum_{i=1}^T u_i \\ \text{subject to} & u_i \geq 0 \\ & u_i \geq -w^T r_i - \alpha. \end{array}$$

Це формулювання вводить нову змінну для кожної точки даних (подібно до ефективної напівваріантності), тому ми можете зіткнутися з проблемами продуктивності для довгих кадрів даних, що повертаються. У той же час, ми повинні прагнути надати вибірку даних, достатньо велику, щоб включати події хвоста.

1.5.3 Efficient CDaR

Умовно просадка ризику (CDaR) є більш екзотичної мірою ризику хвоста. Він намагається полегшити проблеми з ефективною напівваріантністю та ефективним CvAr тим, що враховує часовий проміжок зменшення вартості матеріалу. CDaR можна розглядати як середнє значення втрат, що виникають у "дуже погані періоди", де "дуже погано" визначається кількісно за допомогою параметра β . Просадка визначається як різниця у складному поверненні до попереднього піку.

По -іншому, CDaR - це середнє значення всіх просадок, настільки серйозних, що вони тільки відбуваються $(1-\beta)\%$ часу. Коли $\beta=1$, то CDaR - це просто максимальна просадка.

Хоча просадка є досить інтуїтивно зрозумілою концепцією, для її математичного формулювання потрібно багато нових позначень. Ми прийнемо такі позначення:

- w для вектора ваг портфеля;
- r для вектора сукупної дохідності активів (щоденно) з розподілом ймовірностей $p(r(t))$;
- $D(w, r, t) = \max_{\tau < t} (w^T r(\tau)) - w^T r(t)$ для вилучення портфеля;
- α для зменшення портфеля (DaR) з упевненістю β .

Потім CDaR можна записати так:

$$CDaR(w, \beta) = \frac{1}{1 - \beta} \int_{D(w, r, t) \geq \alpha(w)} D(w, r, t) p(r(t)) dr(t).$$

Це неприємний вираз для оптимізації, оскільки ми по суті інтегруємо значення VaR. Ключове розуміння Чехлова, Рокафеллара та Уряєва (2005)

полягає в тому, що ми можемо еквівалентно оптимізувати опуклу функцію, яку можна перетворити на лінійну задачу (так само, як і для CVaR).

1.6 Ставлення стандартного відхилення до ризику

При інвестуванні стандартне відхилення використовується як індикатор волатильності ринку і, таким чином, ризику. Чим непередбачуваніша цінова дія і ширший діапазон, тим більше ризик. Цінні папери, прив'язані до асортименту, або ті, які не відходять далеко від своїх можливостей, не вважаються великим ризиком. Це тому, що можна припустити — з відносною впевненістю — що вони продовжують вести себе так само. Цінний папір з дуже великим торговим діапазоном і тенденцією до різкого зростання, різкого розвороту або розриву є набагато більш ризикованим, що може означати більші втрати. Але пам'ятайте, що ризик не обов'язково поганий у світі інвестицій. Чим ризикованіше безпека, тим більший потенціал у неї для виплати.

Чим вище стандартне відхилення, тим ризикованіші інвестиції.

При використанні стандартного відхилення для вимірювання ризику на фондовому ринку основне припущення полягає в тому, що більшість цінової діяльності йде за зразком нормального розподілу. При нормальному розподілі окремі значення потрапляють в межах одного стандартного відхилення середнього значення вище або нижче 68% часу. Значення знаходяться в межах двох стандартних відхилень 95% часу.

Наприклад, на акції із середньою ціною 45 доларів США і стандартним відхиленням 5 доларів можна з 95% впевненістю припустити, що наступна ціна закриття залишається між 35 і 55 доларами. Однак ціна різко падає або стрибки за межами цього діапазону в 5% випадків. Акції з високою мінливістю, як

правило, мають високе стандартне відхилення, тоді як відхилення стабільної акції блакитних фішок зазвичай досить низьке.

1.7 Модель Black-Litterman (BL)

Модель Black-Litterman (BL) [1] використовує байєсівський підхід до розподілу активів. В Зокрема, він поєднує в собі попередню оцінку прибутковості (наприклад, the market-implied returns) з видом на певних активах, щоб зробити задню оцінку очікуваної прибутковості.

Перевагами цього є:

1. Ви можете надати уявлення лише про підмножину активів, і BL суттєво поширюватиме їх, беручи до уваги коваріацію з іншими активами.
2. Ви можете бути впевненими у своїх поглядах.
3. Використання результатів повернення Блек-Літтермана призводить до набагато більш стабільних портфелів, ніж використання середньо-історичної віддачі.

По суті, Блек-Літтерман розглядає сам вектор очікуваної віддачі як величину, яку потрібно оцінити. Формула Блек-Літтермана наведена нижче:

$$E(R) = [(\tau\Sigma)^{-1} + P^T\Omega^{-1}P]^{-1}[(\tau\Sigma)^{-1}\Pi + P^T\Omega^{-1}Q]$$

де

$E(R)$ – вектор очікуваної прибутковості $N \times 1$;

N – кількість активів;

Q - вектор переглядів $K \times 1$;

P - це матриця вибору $K \times N$, яка відображає уявлення до всесвіту активів. По суті, він повідомляє моделі, який вид відповідає якому активу (активам);

Ω – матриця невизначеності переглядів $K \times K$;
 P – вектор $N \times 1$ попередніх очікуваних прибутків;
 Σ є матрицею коваріацій $N \times N$ повернення активів;
 τ - це скалярна константа налаштування.

Хоча формула здається досить громіздкою, виявляється, що формула просто являє собою середнє зважене між попередньою оцінкою прибутку та переглядами, де зважування визначається впевненістю до переглядів і параметра τ .

Аналогічно ми можемо обчислити апостеріорну оцінку коваріаційної матриці:

$$\hat{\Sigma} = \Sigma + [(\tau \Sigma)^{-1} + P^T \Omega^{-1} P]^{-1}$$

Можемо вважати попередню оцінкою за замовчуванням за відсутності будь-якої інформації. Black and Litterman (1991) дають уявлення про те, що природним вибором для цього попереднього є ринкова оцінка прибутку, яка вбудована в ринкову капіталізацію активу.

Кожен актив у ринковому портфелі вносить певний ризик для портфеля. Стандартна теорія передбачає, що інвестори повинні отримати компенсацію за ризик, який вони беруть на себе, тому ми можемо віднести до кожного активу очікувану компенсацію (тобто попередню оцінку прибутку). Це кількісно виражається премією за ризик, що передбачається на ринку, яка є надлишковим доходом ринку, поділеним на його дисперсію:

$$\delta = \frac{R - R_f}{\sigma^2}$$

Для розрахунку ринкової прибутковості ми використовуємо таку формулу:

$$\Pi = \delta \sum w_{mkt}$$

Тут, w_{mkt} позначає коефіцієнти ринкової капіталізації. Ця формула обчислює загальну суму ризику, внесеного активом, і множить її на ринкову ціну ризику, в результаті чого отримуємо ринковий вектор прибутку.

Як це пряцює.

У моделі Блек-Літтермана користувачі можуть надавати абсолютні або відносні уявлення. Абсолютні точки зору - це твердження типу: "AAPL поверне 10%" або "XOM знизиться на 40%". Відносні погляди, з іншого боку, — це твердження на кшталт «GOOG перевершить FB на 3%».

Ці подання повинні бути вказані у векторі Q і відображено у всесвіті активів за допомогою матриці вибору P . Нижче наведено короткий приклад цього.

Скажімо, наш Всесвіт визначається упорядкованим списком: SBUX, GOOG, FB, AAPL, BAC, JPM, T, GE, MSFT, XOM. Ми хочемо представити чотири погляди на ці 10 активів, два абсолютних та два відносних:

1. SBUX знизиться на 20% (абсолютно).
2. MSFT зросте на 5% (абсолютно).
3. GOOG перевершує FB на 10%.
4. BAC і JPM будуть перевершувати T і GE на 15%.

Відповідний вектор переглядів утворюється шляхом взяття наведених вище чисел і розміщення їх у стовпець:

$Q = \text{np.array}([-0.20, 0.05, 0.10, 0.15]).\text{reshape}(-1, 1).$

Матриця довіри та тау

Матриця довіри - це діагональна коваріаційна матриця, що містить дисперсії кожного виду. Одна евристика для розрахунку Ω це означає, що це пропорційно до дисперсії попередників. Це розумно - кількість, яка багато рухається, важче передбачити!

За замовчуванням має значення:

$$\Omega = \tau * P \Sigma P^T$$

В якості альтернативи можемо реалізувати методу Ідзорека. Це дозволяє вказати невизначеність погляду інвестора у відсотках довіри.

Вага після обробки

Після того як оптимальні ваги були створені, часто необхідно виконати певну постобробку, перш ніж їх можна буде використовувати на практиці. Зокрема, інвестор, ймовірно, використовує методи оптимізації портфеля для створення розподілу портфеля – списку тикерів і відповідних цілих кількостей, які він можете придбати у брокера.

Однак непросто перетворити безперервні вагові коефіцієнти (вихідні дані за допомогою будь-якого з методів оптимізації) у дійсний розподіл. Наприклад, скажімо, що у інвестора є 10 000 доларів, які б він хотів би виділити. Якщо ми помножимо вагові коефіцієнти на загальну вартість портфеля, то в результаті отримаємо доларову суму кожного активу. Отже, якщо оптимальна вага для Apple становить 0,15, нам потрібні акції Apple на суму 1500 доларів. Однак акції Apple поставляються в окремих одиницях (190 доларів США на момент написання статті), тому ми не зможемо купити акції точно за 1500 доларів США. Найкраще, що ми можемо зробити, це купити таку кількість акцій, яка наблизить нас до бажаної вартості долара.

Є спосіб вирішення цієї проблеми: за допомогою простого алчного алгоритму.

Жадібний алгоритм.

Greedy portfolio() триває у двох «раундах». У першому раунді ми купуємо якомога більше акцій за кожен актив, не перевищуючи бажаної ваги. У прикладі

Apple, $1500/190 \approx 7.89$, тому ми купуємо 7 акцій вартістю 1330 доларів. Після повторного перегляду всіх активів у нас залишиться багато грошей (оскільки ми завжди округлюємо).

У другому раунді ми обчислюємо, наскільки поточні ваги відхиляються від існуючих ваг для кожного активу. Ми хотіли, щоб Apple сформувала 15% портфеля (загальною вартістю 10 000 доларів США), але ми купили лише акції Apple на суму 1330 доларів, тому існує відхилення 0.15–0.133. Деякі активи будуть мати більший відхилення від ідеалу, тому ми спочатку придбаємо їх акції. Потім ми повторюємо процес, завжди купуючи акції активу, чия поточна вага найбільш віддалена від ідеальної ваги. Хоча цей алгоритм не гарантує оптимального рішення, виявляється, що він дозволяє нам генерувати дискретні асигнування з дуже невеликою кількістю грошей (наприклад, 12 доларів, залишених на портфелі 10 000 доларів США).

1.8 Висновки до розділу

Конкретні фінансові рішення завжди мають бути спрямовані на майбутнє, і ця умова є причиною, що зумовлює те, що багато параметрів невідомі, що мають одне значення, але відомий лише спектр можливостей. Ось чому, коли існує невизначеність або ризик, необхідно приймати рішення, які потребують нових принципів регулювання інформації та методу прийняття рішення.

В умовах ринку багато завдань з оптимізації вирішуються не тільки для того, щоб досягти граничних оптимізуючих розмірів - прибутку, частини ринкових та подібних цінностей, гарантуючи ту чи іншу вартість. Проаналізовано принципи та досягнуто методу формування та розрахунку інвестиційного портфеля, адекватного стохастичному характеру інвестиційного доходу.

- Це дуже важливий момент, особливо фінансовий, момент управління фінансами - це управління активами та пасивами та, насамперед, формування цих пасивів та оптимального портфеля активів;
- Враховуючи постійно зростаючий вплив ризику та невизначеності на результати підприємницької чи індивідуальної діяльності, а також збільшення кількості інструментів управління ризиками, дуже важливо правильно організувати бухгалтерський облік та інше інформаційне регулювання, що дозволяє приймати раціональні рішення щодо ризиків. можливість управління.

РОЗДІЛ 2 ОБГРУНТУВАННЯ ОБ'ЄКТІВ ІНВЕСТУВАННЯ

2.1 Аналіз інвестицій з точки зору потенційного ризику і прибутковості

2.1.1 Фундаментальний аналіз

Фундаментальний аналіз може бути одним із інструментів, який допоможе вам вибрати акції для купівлі, утримання та продажу, залежно від того, що саме ви хочете поставити компаніям у вашому портфелі.

Фундаментальний аналіз має два елементи: розрив чисел, який стосується аналізу фінансових аспектів компанії та визначення того, чи збігаються такі фактори, як зростання доходів, боргів та доходів, щоб сформувати уявлення про майбутнє, яке вам подобається.

Іншу частину фундаментального аналізу важче виміряти - нематеріальні елементи, які також впливають на результати діяльності компанії, такі як її управління та стратегія.

Для кого підходить фундаментальний аналіз?

Фундаментальний аналіз вимагає багато досліджень і може вимагати більше часу та зусиль, ніж технічний аналіз. Однак якщо інвестор хоче інвестувати на довгострокову перспективу, дослідження дуже важливі.

За допомогою фундаментального аналізу ви дійсно розкриваєте і зосереджуєтесь на тому, чи шукаєте ви бізнес, яким задоволені. З іншого боку, технічний аналіз, як правило, більше використовується трейдерами, які, ймовірно, будуть більше зосереджені на часових рамках днів або навіть хвилин, а не інвесторами, які будуть зосереджені на тому, що акції будуть поставлятися роками.

Іншими словами, як інвестор ви купите акцію, тому що вважаєте, що вона має потенціал для збільшення вартості, і вам доведеться використати фундаментальний аналіз, щоб сказати вам це. Однак трейдер, який використовує технічний аналіз, буде більше піклуватися про те, щоб заробити на короткострокових змінах ціни.

2.1.2 Технічний аналіз

Багато інвесторів аналізують акції на основі їх фундаментальних показників – таких як дохід, оцінка або галузеві тенденції – але фундаментальні фактори не завжди відображаються на ринковій ціні. Технічний аналіз має на меті передбачити рух цін шляхом вивчення історичних даних, головним чином ціни та обсягу.

Це допомагає трейдерам та інвесторам подолати розрив між внутрішньою вартістю та ринковою ціною, використовуючи такі методи, як статистичний аналіз та поведінкова економіка. Технічний аналіз допомагає трейдерам орієнтуватися на те, що найімовірніше станеться з огляду на минулу інформацію. Більшість інвесторів для прийняття рішень використовують як технічний, так і фундаментальний аналіз.

Як правило, існує два різні способи підходу до технічного аналізу: підхід зверху вниз і підхід знизу вгору . Часто короткострокові трейдери використовують підхід зверху вниз, а довгострокові інвестори — знизу вгору.

Основні кроки для початку роботи з технічним аналізом:

1. Вибрати стратегію або розробити систему торгівлі.

Перший крок – визначити стратегію або розробити торгову систему. Наприклад, трейдер-початківець може прийняти рішення дотримуватися стратегії кросовера ковзної середньої , де він відстежуватиме дві ковзаючі середні (50-денну та 200-денну) за певним рухом ціни акцій.

Для цієї стратегії, якщо короткострокова 50-денна ковзаюча середня перевищує довгострокову 200-денну ковзну середню, це вказує на зростання ціни та генерує сигнал на покупку . Для сигналу продажу є зворотне.

2. Визначити цінні папери.

Не всі акції або цінні папери підходять для вищевказаної стратегії, яка ідеально підходить для високоліквідних і нестабільних акцій замість неліквідних або стабільних акцій. Різні акції або контракти також можуть

вимагати різного вибору параметрів-у цьому випадку різні ковзні середні, такі як 15-денна та 50-денна ковзна середня.

3. Знайти правильного посередника.

Потрібно отримати правильний торговий рахунок, який підтримує вибраний тип цінних паперів (наприклад, звичайні акції, пенні, ф'ючерси, опціони тощо). Він повинен пропонувати необхідну функціональність для відстеження та моніторингу вибраних технічних показників, зберігаючи при цьому низькі витрати, щоб уникнути поглинання прибутку. Для наведеної вище стратегії працював би базовий рахунок із ковзними середніми на діаграмах свічників.

4. Відстеження та моніторинг угод.

Трейдери можуть вимагати різних рівнів функціональності залежно від своєї стратегії. Наприклад, денним трейдерам знадобиться маржинальний рахунок, який надає доступ до котирування рівня II і видимість маркет-мейкерів. Але для нашого прикладу вище, базовий обліковий запис може бути кращим як варіант з меншою вартістю.

5. Використовування додаткового програмного забезпечення або інструментів.

Можливо, для підвищення продуктивності потрібні інші функції. Деяким трейдерам можуть знадобитися мобільні сповіщення або доступ до торгівлі на ходу, в той час як інші можуть використовувати автоматизовані торгові системи для здійснення торгів від свого імені.

2.2 Аналіз об'єктів інвестування

MSFT

Корпорація Майкрософт є технологічною компанією. Компанія розробляє, ліцензує і підтримує ряд програмних продуктів, послуг та пристроїв.

Сегменти компанії включають продуктивність і бізнес-процеси, інтелектуальне хмара та інші персональні обчислення. Продукти компанії включають операційні системи; додатки для підвищення продуктивності на різних пристроях; серверні додатки; додатки для бізнес-рішень; інструменти управління робочим столом і сервером; засоби розробки програмного забезпечення; відеоігри, а також навчання і сертифікація системних інтеграторів і розробників.

Компанія розробляє, виробляє і продає пристрої, включаючи персональні комп'ютери (ПК), планшети, ігрові та розважальні консолі, телефони, інші інтелектуальні пристрої та супутні аксесуари, які інтегруються з його хмарними пропозиціями.

Вона пропонує ряд послуг, включаючи хмарні рішення, які надають клієнтам програмне забезпечення, послуги, платформи і контент, а також надає підтримку рішень та консультаційні послуги.

На Рис. 2.1 Зображено історичні ціни на MSFT

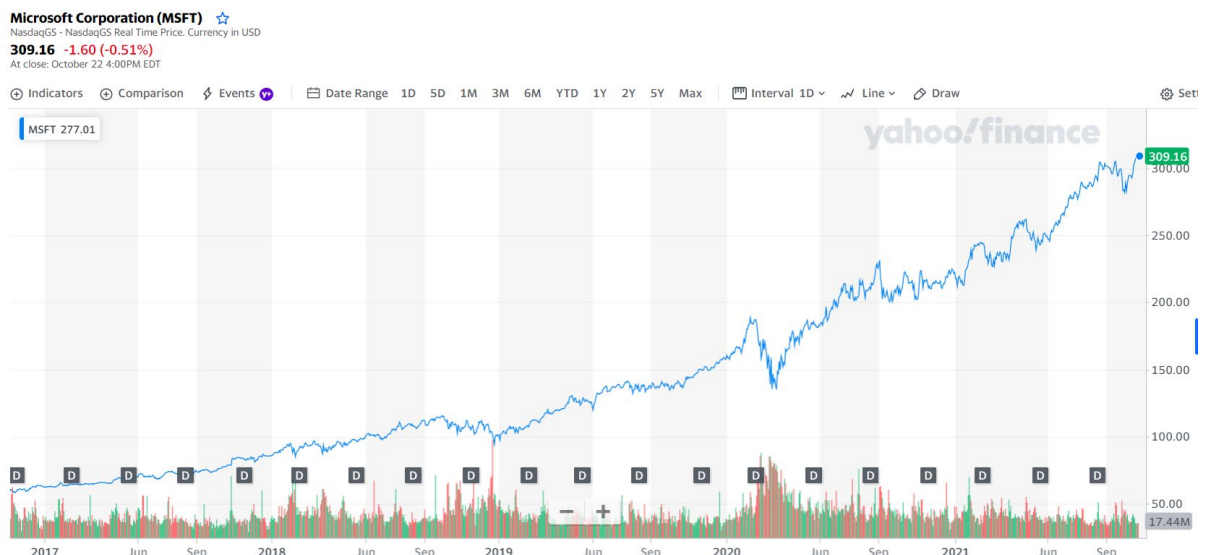


Рисунок 2.1 - Історичні ціни на MSFT

AMZN

Амазон.ком, Інк. пропонує широкий спектр продуктів і послуг через свої веб-сайти. Продукти компанії включають товари і контент, які вона купує для перепродажу у постачальників і пропонує сторонні продавці.

Компанія також виробляє і продає електронні пристрої. Вона працює в трьох сегментах: Північна Америка, Міжнародний і Амазон Веб Сервісес (AWS). Її продукти AWS включають аналітику, Амазон Атена, Амазон Клауд серч, Амазон OMP, Амазон Еластик серч Сервіс, Амазон Кінесіс, Амазон Менедж стрімінга для апашей Кафка, Амазон Редшіфт, Амазон КвікСайт, AWS Дата Піплайн, AWS Гли і AWS Лейк Формейшн.

Рішення AWS включають машинне навчання, аналітику і озера даних, Інтернет речей, бессерверной обчислення, контейнери, корпоративні програми і сховище. Крім того, Компанія надає такі послуги, як реклама.

На Рис. 2.2 Зображено історичні ціни на AMZN



Рисунок 2.2 - Історичні ціни на AMZN

КО

Coca-Cola Co. — це компанія з виробництва безалкогольних напоїв, яка займається виробництвом, реалізацією та продажем безалкогольних напоїв, які включають газовані безалкогольні напої, воду, воду та спортивні напої, соки, молочні та рослинні напої, чай і кава та енергетичні напої.

Його бренди включають Coca-Cola, Diet Coke, Coca-Cola Zero, Fanta, Sprite, Minute Maid, Georgia, Powerade, Del Valle, Schweppes, Aquarius, Minute Maid Pulpy, Dasani, Simply, Glaceau Vitaminwater, Bonaqua, Gold Peak, Fuze Чай, вода Glaceau Smartwater та Ice Dew. Він працює в таких сегментах: Європа, Близький Схід та Африка, Латинська Америка, Північна Америка, Азіатсько-Тихоокеанський регіон, Розливні інвестиції та Global Ventures. Компанію заснував Аса Гріггс Кендлер у 1886 році, а штаб-квартира знаходиться в Атланті, штат Джорджія.

На Рис. 2.3 Зображено історичні ціни на КО



Рисунок 2.3 - Історичні ціни на КО

МА

Мастеркард Інкорпорейтед - технологічна компанія, яка об'єднує споживачів, фінансові установи, продавців, уряду і підприємства по всьому світу, дозволяючи їм використовувати електронні форми оплати.

Компанія дозволяє користувачам здійснювати платежі, створюючи ряд платіжних рішень і послуг, використовуючи свої бренди, в тому числі МастерКард, Маестро та Циррус. Компанія пропонує ряд продуктів і рішень для підтримки платіжних продуктів, які клієнти можуть запропонувати власникам карт.

Послуги компанії полегшують транзакції в її базовій мережі між власниками рахунків, продавцями, фінансовими установами, підприємствами, урядами та іншими організаціями на ринках по всьому світу. Продукти компанії включають споживчі кредити і дебет, передплачені і комерційні. Компанія також надає інтегровані пропозиції.

На Рис. 2.4 Зображено історичні ціни на МА



Рисунок 2.4 - Історичні ціни на МА

COST

Оптова корпорація Costco має оптові склади для членства в багатьох країнах. Компанія продає всі види продуктів харчування, автомобільні приналежності, іграшки, обладнання, спортивні товари, ювелірні вироби, електроніку, одяг, засоби для здоров'я та краси, а також інші товари.

На Рис. 2.5 Зображено історичні ціни на COST



Рисунок 2.5 - Історичні ціни на COST

LUV

Southwest Airlines Co. займається управлінням пасажирської авіакомпанії. Він пропонує допоміжні послуги, такі як рання реєстрація на приїзд, покращений інтернат, а також перевезення домашніх тварин і неповнолітніх без супроводу.

Фірма працює в штатах США, окрузі Колумбія, Співдружності Пуерто - Ріко, Мексиці, Ямайці, Багамських Островах, Арубі, Домініканській Республіці, Коста -Ріці, Белізі, Кубі, Кайманових островах, а також у Терксах і Кайкосі. Компанія була заснована Ролліном У. Кінгом та Гербертом Д. Келлехером 15 березня 1967 року і має штаб -квартиру в Далласі, Техас.

На Рис. 2.6 Зображено історичні ціни на LUV

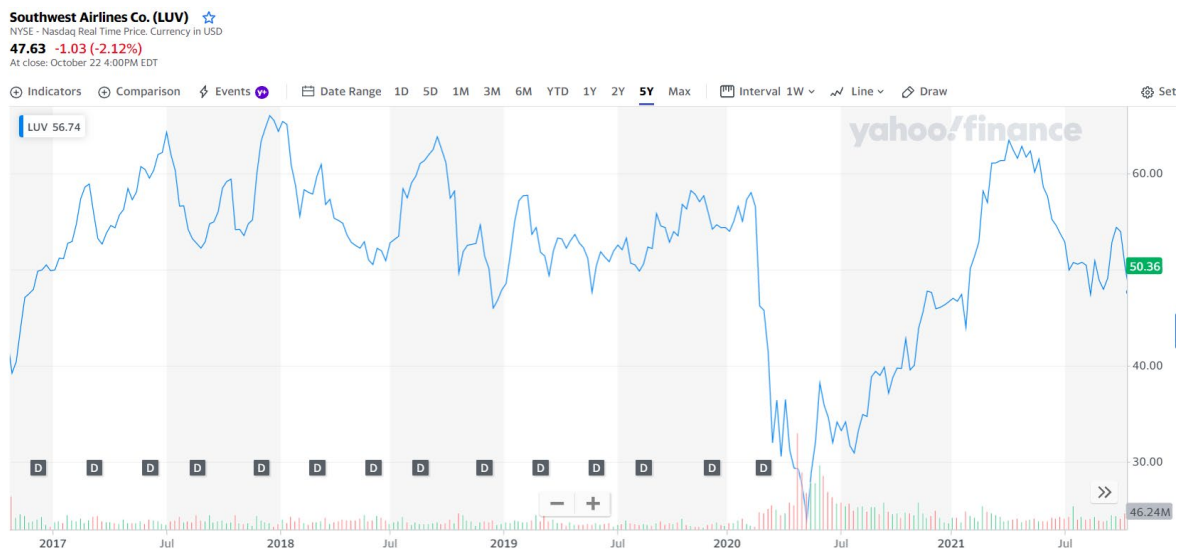


Рисунок 2.6 - Історичні ціни на LUV

ХОМ

Корпорація Ексон Мобіл займається енергетичним бізнесом. Компанія займається розвідкою, видобутком, транспортуванням і продажем сирої нафти і природного газу, а також виробництвом, транспортуванням і продажем нафтопродуктів.

Компанія також виробляє і продає нафтохімічні продукти, включаючи олефіни, ароматичні вуглеводні, поліетилен і поліпропіленові пластики, а також ряд спеціалізованих продуктів. Сегменти компанії включають Апстрім, Давнстрім, а також хімічну промисловість, корпоративне право та фінансування.

Сегмент Апстрім займається розвідкою і видобутком сирої нафти і природного газу. Давнстрім займається виробництвом і продажем нафтопродуктів. Хімічний сегмент займається виробництвом і продажем продуктів нафтохімії. Компанія веде розвідку і розробку проектів в США, Канаді / Південній Америці, Європі, Африці.

На Рис. 2.7 Зображено історичні ціни на ХОМ



Рисунок 2.7 - Історичні ціни на ХОМ

PFE

Пфайзер Інк. є глобальною біофармацевтичною компанією, заснованою на дослідженнях. Компанія займається відкриттям, розробкою і виробництвом товарів для здоров'я. В її глобальний портфель входять ліки і вакцини.

Компанія управляє своїми комерційними операціями через два бізнес-сегмента: Пфайзер Інновейтів Хелс (ІН) і Пфайзер Ессентіал Хелс (ЕН). ІН спеціалізується на розробці і комерціалізації ліків і вакцин. Терапевтичні області ІН включають внутрішню медицину, вакцини, онкологію, запалення і імунологію, рідкісні захворювання і охорону здоров'я.

ЕН включає в себе застарілі бренди, фірмові дженерики, непатентовані стерильні ін'єкційні продукти, біоаналогів і інфузійні системи. ЕН також включає в себе організацію досліджень і розробок (R & D), а також контрактне виробництво.

На Рис. 2.8 Зображено історичні ціни на PFE



Рисунок 2.8 - Історичні ціни на PFE

JPM

JPMorgan Chase & Co. є фінансовою холдинговою компанією. Надає фінансові та інвестиційні банківські послуги. Фірма пропонує широкий спектр інвестиційних банківських продуктів та послуг на всіх ринках капіталу, включаючи консультування з питань корпоративної стратегії та структури, залучення капіталу на ринках акцій та боргів, управління ризиками, ринок майнових цінних паперів та похідних інструментів, а також посередництво та дослідження.

Він працює через такі сегменти: споживчий та громадський банк, корпоративний та інвестиційний банк, комерційний банк, а також управління активами та багатствами. Сегмент споживчого та громадського банківського обслуговування обслуговує споживачів і бізнес за допомогою персональних послуг у відділеннях банків і через банкомат, онлайн-, мобільний та телефонний банкінг.

Сегмент «Корпоративний та інвестиційний банк» пропонує набір інвестиційних банків, маркетинг, першочергове посередництво, а також казначейські та цінні папери та послуги для глобальної клієнтської бази корпорацій, інвесторів, фінансових установ, державних та муніципальних утворень.

Сегмент комерційного банкінгу надає послуги США та її багатонаціональним клієнтам, включаючи корпорації, муніципалітети, фінансові установи та некомерційні організації.

На Рис. 2.9 Зображено історичні ціни на JPM



Рисунок 2.9 - Історичні ціни на JPM

UNH

UnitedHealth Group, Inc. займається наданням медичних послуг, програмним забезпеченням та консультаційними послугами щодо даних. Він працює через такі сегменти: UnitedHealthcare, OptumHealth, OptumInsight та OptumRx.

Сегмент UnitedHealthcare використовує можливості Optum для координації догляду за пацієнтами, покращення доступності медичної допомоги, аналізу тенденцій витрат, управління пільгами в аптеках, ефективнішої роботи з постачальниками медичних послуг та створення більш простого досвіду споживачів.

Сегмент OptumHealth надає медичні послуги бізнесу, який обслуговує широкий ринок охорони здоров'я, включаючи платників, медичних працівників, роботодавців, уряд, компанії з наук про життя та споживачів.

Сегмент OptumInsight зосереджений на даних та аналітиці, технологіях та інформації, щоб допомогти основним учасникам галузі охорони здоров'я. Сегмент OptumRx надає послуги з аптечного обслуговування. Компанія була заснована Річардом Т. Берком у січні 1977 року і має штаб -квартиру в Міннетонці, штат Міннесота.

На Рис. 2.10 Зображено історичні ціни на UNH



Рисунок 2.10 - Історичні ціни на UNH

ACN

Accenture Plc займається наданням управлінських консультацій, технологій та аутсорсингом. Вона працює за такими сегментами: комунікації, медіа та технології; Фінансові послуги; охорона здоров'я та державна служба; Продукція; Ресурси; та інші.

Сегмент комунікацій, медіа та технологій обслуговує компанії з комунікацій, засобів масової інформації, високих технологій та програмного забезпечення та платформ за рахунок прискорення та реалізації цифрової трансформації, розробки комплексних та галузевих рішень та підвищення ефективності та результатів бізнесу.

Послуги компанії включають допомогу клієнтам досягти нового зростання шляхом переходу до моделей на основі даних та платформ, оптимізації їх структури витрат, збільшення інновацій продуктів та бізнес-моделей, а також диференціації та розширення цифрового досвіду для своїх клієнтів.

Сегмент фінансових послуг обслуговує банківську індустрію, ринки капіталу та страхування, вирішуючи проблеми зростання, витрат і

прибутковості, консолідацію галузі, зміни в нормативно-правових актах та потребу постійно адаптуватися до нових цифрових технологій.

Сегмент "Охорона здоров'я та державна служба" обслуговує платників та постачальників медичних послуг, а також державні департаменти та установи, організації державних служб, навчальні заклади та некомерційні організації, надаючи інформацію та пропозиції, включаючи консультаційні послуги та цифрові рішення.

Сегмент Продукція обслуговує: групу споживчих товарів, роздрібну торгівлю та туристичні послуги; Промислова група; та наук про життя. Компанія допомагає клієнтам покращити їхню ефективність у сфері дистрибуції та продажів та маркетингу; в дослідженнях і розробках і виробництві; а також у таких ділових функціях, як фінанси, людські ресурси, закупівлі та ланцюжок поставок, одночасно використовуючи технології.

Сегмент ресурсів обслуговує хімічну, енергетичну, лісову продукцію, металообробну промисловість, комунальні підприємства та суміжні галузі, працюючи над розробкою та впровадженням інноваційних стратегій, вдосконаленням операцій, керуванням складними ініціативами щодо змін та інтеграцією цифрових технологій. Сегмент «Інший» представляє нарахування пенсії.

Компанія була заснована в 1989 році і має штаб -квартиру в Дубліні, Ірландія. керувати комплексними ініціативами щодо змін та інтегрувати цифрові технології. Сегмент «Інший» представляє нарахування пенсії. Компанія була заснована в 1989 році зі штаб-квартирою в Дубліні, Ірландія. керувати ініціативами комплексних змін та інтегрувати цифрові технології.

Інший сегмент являє собою плату за пенсійне врегулювання. Компанія була заснована в 1989 році зі штаб-квартирою в Дубліні, Ірландія.

На Рис. 2.11 Зображено історичні ціни на ACN

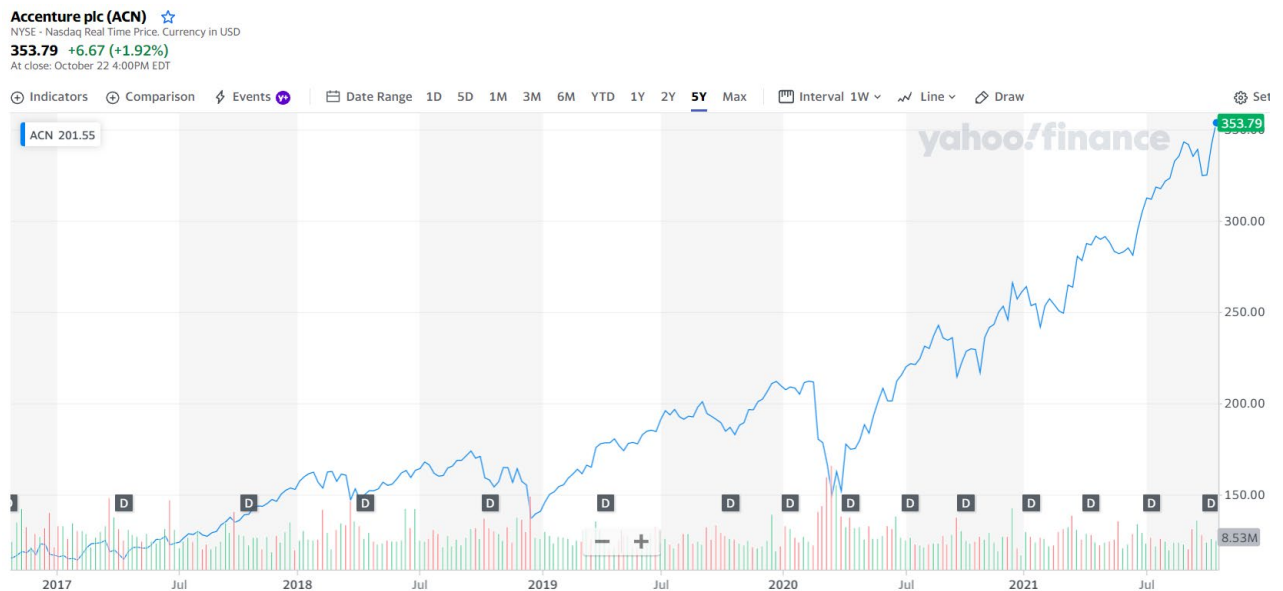


Рисунок 2.11 - Історичні ціни на ACN

DIS

Walt Disney Co. - це різноманітне міжнародне сімейне розважальне та медіа -підприємство. Він працює через такі сегменти: медіа-мережі, парки, досвід і продукти, студійні розваги та напрямки для споживачів і міжнародні (DTCI).

Сегмент медіа-мереж включає мережі кабельного та мовного телебачення, телевізійне виробництво та розповсюдження, внутрішні телевізійні станції, радіомережі та станції. Сегмент «Парки, досвід та продукти» володіє та управляє курортом Уолта Діснея у Флориді; курорт Діснейленд у Каліфорнії; Aulani, Disney Resort & Spa на Гавайях; Disney Vacation Club; круїзна лінія Disney; та Пригоди Діснея. Сегмент Studio Entertainment виробляє та купує живі та анімаційні фільми, контент прямо до відео, музичні записи та живі сценічні вистави.

Цей сегмент розповсюджує фільми насамперед під прапорами Walt Disney Pictures, Pixar, Marvel, Lucasfilm та Touchstone. Сегмент DTCI надає ліцензії на торгові назви, персонажі та візуальні та літературні властивості

компанії різним виробникам, розробникам ігор, видавцям і роздрібним продавцям по всьому світу.

Він також розробляє та видає ігри, в першу чергу для мобільних платформ, а також книги, журнали та комікси. Цей сегмент також розповсюджує фірмові товари безпосередньо через роздрібні, онлайн- та оптові підприємства. Компанія була заснована Уолтером Еліасом Діснеєм 16 жовтня 1923 року і має штаб -квартиру в Бербанку, Каліфорнія. торгові назви, персонажі та візуальні та літературні властивості для різних виробників, розробників ігор, видавців та роздрібних продавців у всьому світі.

Він також розробляє та видає ігри, насамперед для мобільних платформ, а також книги, журнали та комікси. Цей сегмент також поширює брендові товари безпосередньо через роздрібний, онлайн та оптовий бізнес. Компанія була заснована Уолтером Еліасом Діснеєм 16 жовтня 1923 року і має штаб -квартиру в Бербанку, Каліфорнія. s торгові назви, персонажі та візуальні та літературні властивості різним виробникам, розробникам ігор, видавцям та роздрібним торговцям у всьому світі.

Він також розробляє та видає ігри, в першу чергу для мобільних платформ, а також книги, журнали та комікси. Цей сегмент також поширює брендові товари безпосередньо через роздрібний, онлайн та оптовий бізнес. Компанія була заснована Уолтером Еліасом Діснеєм 16 жовтня 1923 року і має штаб-квартиру в Бербанку, Каліфорнія.

На Рис. 2.12 Зображено історичні ціни на DIS



Рисунок 2.12 - Історичні ціни на DIS

GILD

Gilead Sciences, Inc. - біофармацевтична компанія, яка займається дослідженнями, розробкою та комерціалізацією лікарських засобів у сферах незадоволеної медичної потреби.

Основними напрямками діяльності фірми є вірус імунодефіциту людини, синдром набутого імунодефіциту, захворювання печінки, гематологія, онкологія, а також запальні та респіраторні захворювання.

Він пропонує протівірусні продукти під брендами Harvoni, Genvoya, Epclusa, Truvada, Atripla, Descovy, Stribild, Viread, Odefsey, Complera/Eviplera, Sovaldi, Vosevi. Компанія була заснована Майклом Л. Ріорданом 22 червня 1987 року і має штаб-квартиру у Фостер-Сіті, Каліфорнія.

На Рис. 2.13 Зображено історичні ціни на GILD



Рисунок 2.13 - Історичні ціни на GILD

F

Ford Motor Co. займається виробництвом, розповсюдженням та продажем автомобілів. Він працює за трьома наступними сегментами: автомобільний, мобільний та кредитний Ford.

Автомобільний сегмент займається розробкою, виробництвом, маркетингом та обслуговуванням автомобілів Ford, автомобілів Lincoln.

Сегмент Mobility включає в себе Ford Smart Mobility LLC та бізнес автономних транспортних засобів. Сегмент Ford Credit включає бізнес Ford Credit на консолідованій основі, що насамперед стосується фінансування та оренди транспортних засобів.

Компанія була заснована Генрі Фордом 16 червня 1903 року зі штаб-квартирою в Дірборні, штат Мічиган.

На Рис. 2.14 Зображено історичні ціни на F



Рисунок 2.14 - Історичні ціни на F

TSLA

Компанія Tesla, Inc. займається проектуванням, розробкою, виробництвом та продажем повністю електричних транспортних засобів, систем виробництва та зберігання енергії. Він також пропонує центри технічного обслуговування автомобілів, станцію нагнітання та можливість самостійного керування автомобілем.

Компанія працює в таких сегментах: автомобільне виробництво та виробництво та зберігання енергії. Автомобільний сегмент включає проектування, розробку, виробництво та продаж електромобілів. Сегмент виробництва та зберігання енергії включає проектування, виготовлення, встановлення, продаж та оренду стаціонарних продуктів зберігання енергії та систем сонячної енергії та продаж електроенергії, виробленої її системами сонячної енергії, споживачам.

Вона розробляє продукти для зберігання енергії для використання в будинках, комерційних приміщеннях та на комунальних підприємствах.

Компанію заснували Джеффри Б. Штраубель, Ілон Рів Маск, Мартін Еберхард.

На Рис. 2.15 Зображено історичні ціни на TSLA



Рисунок 2.15 - Історичні ціни на TSLA

2.3 Висновки до розділу

Під час прийняття інвестиційних рішень багато інвесторів використовують як фундаментальний, так і технічний аналіз, оскільки технічний аналіз допомагає заповнити прогалини у знаннях. Розвиваючи розуміння технічного аналізу, трейдери та інвестори можуть покращити свої довгострокові прибутки з поправкою на ризик, але важливо розуміти та практикувати ці методи, перш ніж вкладати реальний капітал, щоб уникнути дорогих помилок.

Головні ідеї:

1. Розуміння обґрунтування та логіки, що лежить в основі технічного аналізу.
2. Повторне тестування торгових стратегій, щоб побачити, як би вони діяли в минулому.
3. Практикувати торгівлю на демо-рахунку перед здійсненням реального капіталу.
4. Усвідомлення обмежень технічного аналізу, щоб уникнути дорогих невдач і несподіванок.

5. Бути продуманим та гнучким щодо масштабованості та майбутніх вимог.
6. Намагаючись оцінити особливості торгового рахунку, подавши запит на безкоштовну пробну версію.
7. Починати з малого на початку і розширювати, коли набуваєте досвіду.

РОЗДІЛ 3 АНАЛІЗ І ПОБУДОВА ІНВЕСТИЦІЙНОГО ПОРТФЕЛЯ.

ОСНОВНІ РЕЗУЛЬТАТИ

3.1 Моделі Повернення Ризику

У цьому розділі ми порівнюємо, наскільки добре різні моделі ризику прогнозують коваріаційну матрицю поза вибіркою (out-of-sample covariance matrix), і наскільки добре різні моделі прибутку прогнозують прибутки поза вибіркою (out-of-sample returns).

В цьому підрозділу беремо дані з файлу data/stock_prices.csv і будуємо Коваріацію sample, див. рис. 3.1 та Коваріацію future, див. рис. 3.2

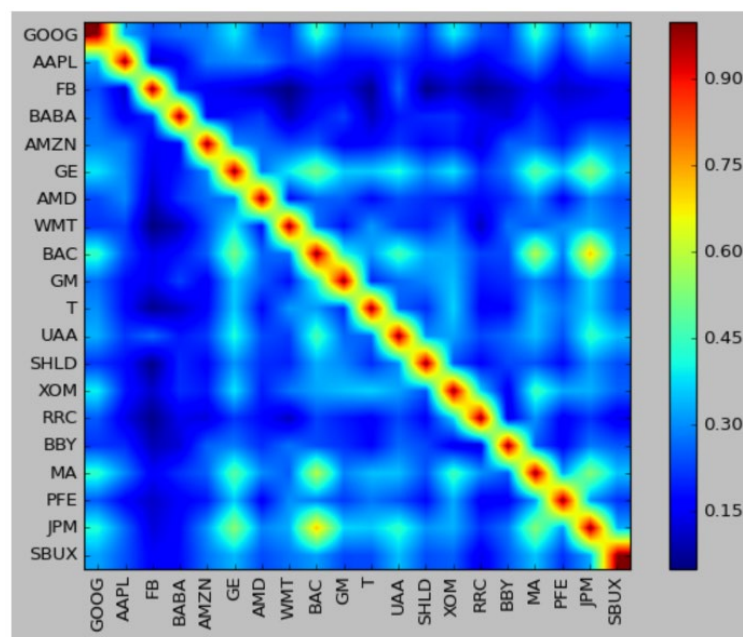


Рисунок 3.1 - Коваріація sample

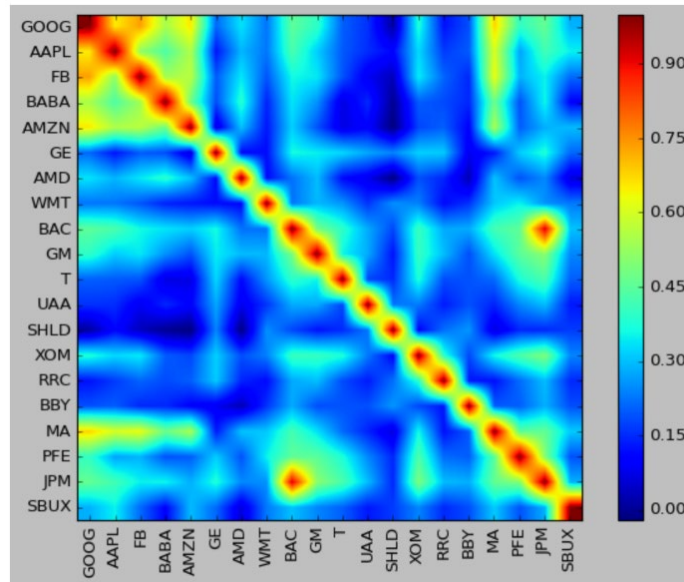


Рисунок 3.2 - Коваріація future

Ми бачимо, що візуально вибіркова коваріація не фіксує деякі нові особливості коваріаційної матриці, наприклад, високо корельовану групу акцій FAANG (Meta (FB), Amazon (AMZN), Apple (AAPL), та Alphabet (GOOG)).

Можливо, ми зможемо покращити це, використовуючи експоненціально зважену коваріаційну матрицю, яка надає більшої ваги останнім даним. Ми також можемо подивитися, як кожна модель прогнозує майбутню дисперсію, див рис. 3.3

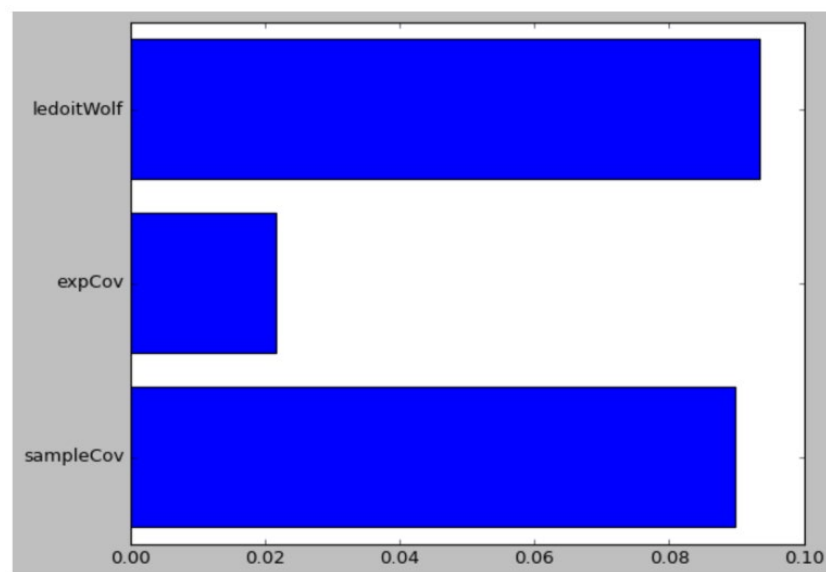


Рисунок 3.3 - Дисперсії помилок

Ми бачимо, що матриця експоненційної коваріації є набагато кращою оцінкою майбутньої дисперсії порівняно з іншими моделями. Його середня абсолютна похибка становить 2%, що насправді дуже добре. Давайте візуально порівняємо експоненційну матрицю cov, див. рис. 3.4, з реалізованою майбутньою матрицею cov, див. рис. 3.5:

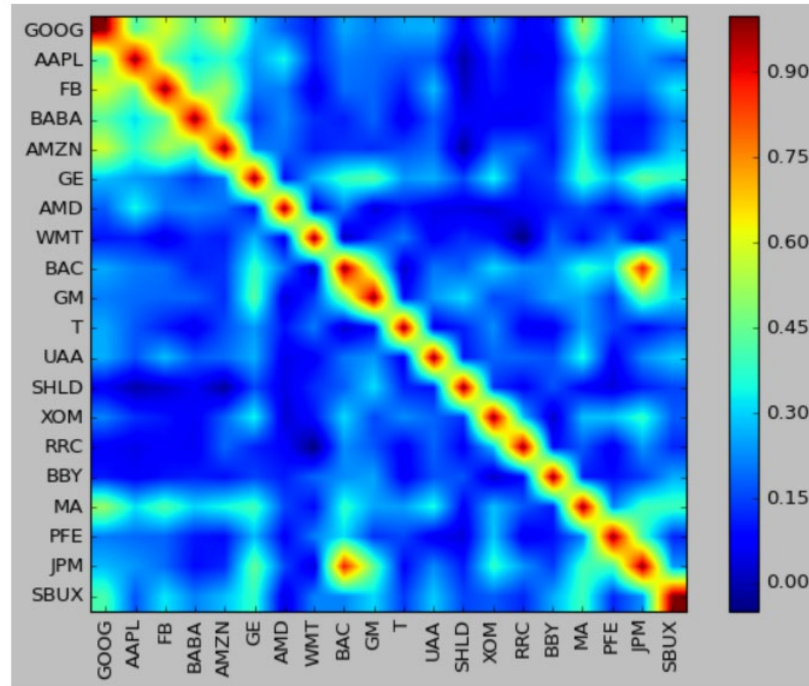


Рисунок 3.4 - Коваріація Ехр

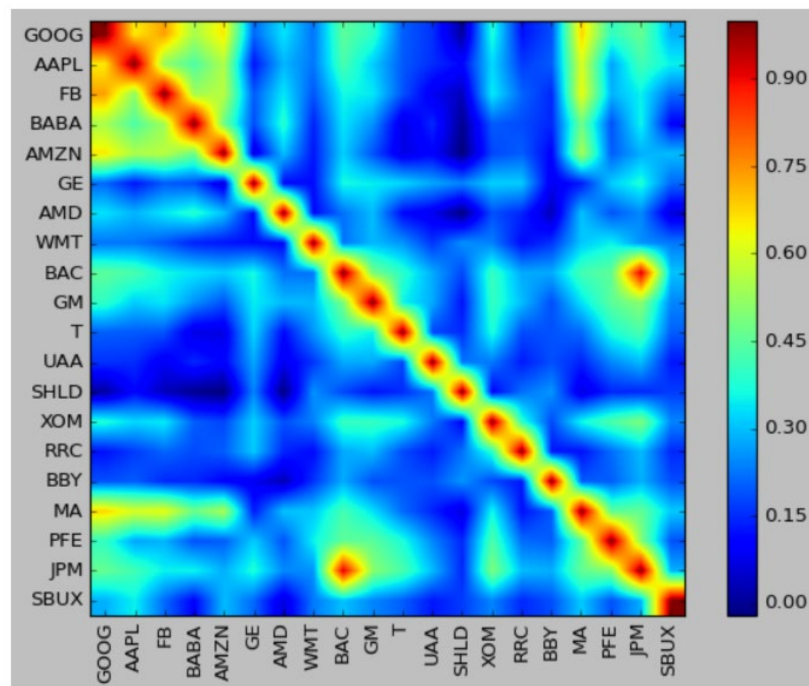


Рисунок 3.5 - Коваріація Майбутня

А як щодо доходності? Будемо вибирати між "Оцінка середньої прибутковості", "Експоненційною" та "Оцінка прибутку CAPM", див. рис. 3.6

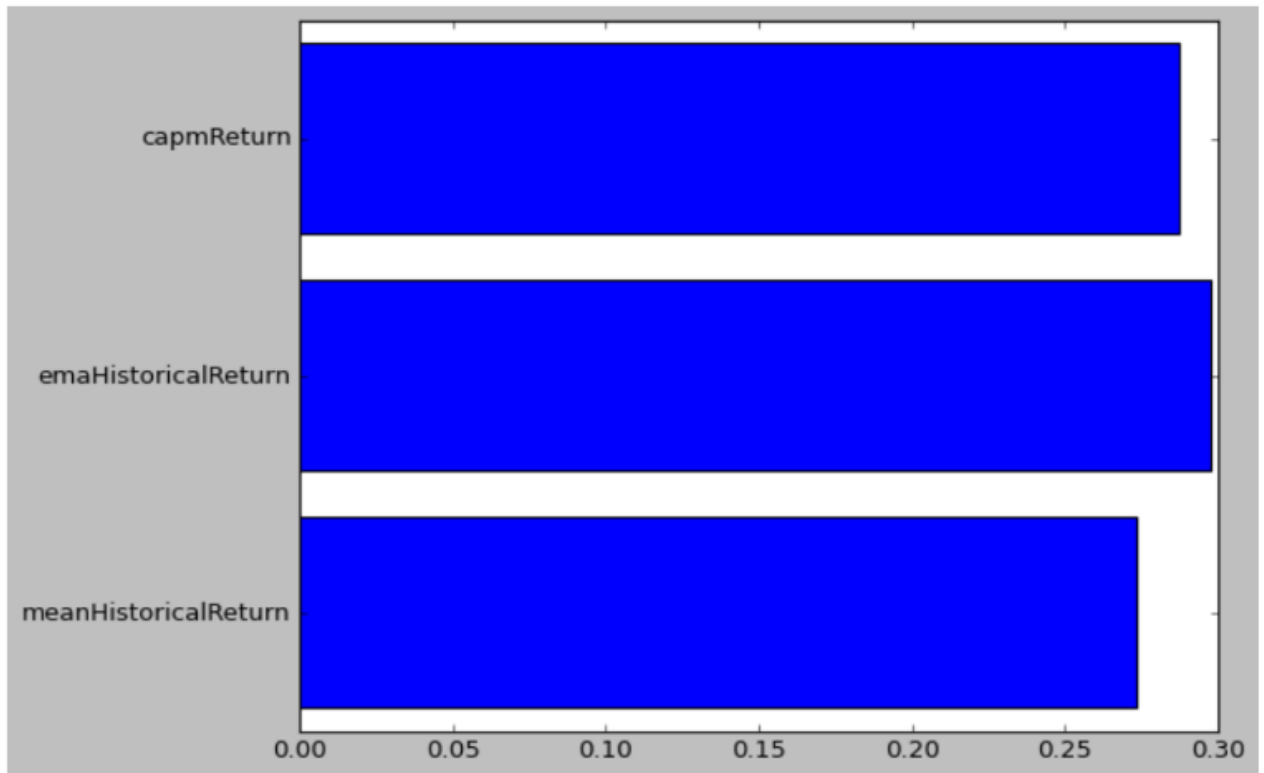


Рисунок 3.6 - Дохідність

Оцінка прибутку Eхr трохи краще, ніж інші, але покращення майже непомітне. Також бачимо, що середні абсолютні відхилення перевищують 25%, що означає, що якщо наші очікувані річні прибутки становлять 10%, в середньому реалізована річна прибутковість може становити від 15% втрати до 35% прибутку. Це величезний діапазон, який бажано зоптимізувати.

Перевіримо ступінь згоди, див. рис. 3.7

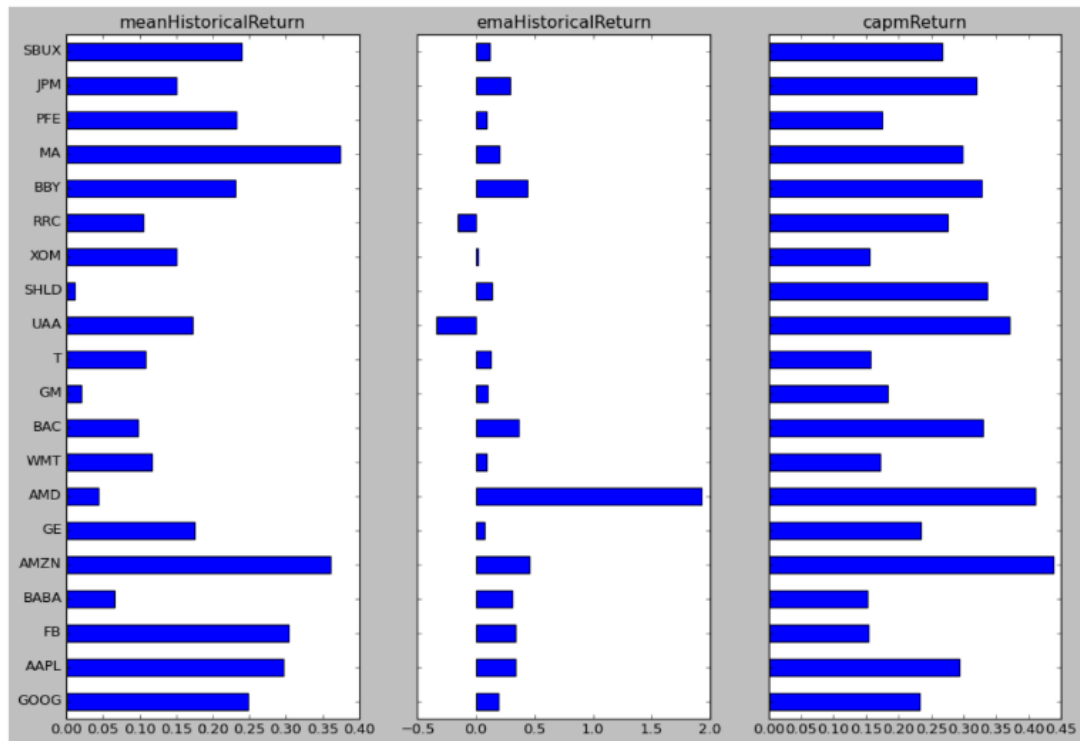


Рисунок 3.7 - Ступінь згоди

Хороша новина полягає в тому, що ми бачимо гарну ступінь згоди, окрім ехр

3.2 Оптимізація Середньої Дисперсії

У цьому підрозділі ми працюємо над кількома прикладами, що демонструють можливості щодо середньої дисперсії.

Я обговорю, які, на мою думку, мають бути нашими параметрами «за замовчуванням», ґрунтуючись на досвіді оптимізації портфелів.

Для початку нам потрібен список тикерів. Деякі люди просто надають весь світ акцій, але я не думаю, що це гарна ідея, оскільки оптимізація портфеля значно відрізняється від вибору активів. Я б запропонував від 10 до 50 акцій як

відправну точку.

Деякі з речей, які ми охоплюємо:

- Завантаження даних;
- Розрахунок та візуалізація коваріаційної матриці;
- Оптимізація long/short портфеля для мінімізації загальної дисперсії;
- Оптимізація портфеля для максимізації коефіцієнта Шарпа, з урахуванням галузевих обмежень;
- Оптимізація портфеля для максимізації прибутку для певного ризику, з урахуванням обмежень сектора, з метою регуляризації L2;
- Оптимізація ринково нейтрального портфеля для мінімізації ризику для заданого рівня прибутку;
- Оптимізація вздовж межі середньої напівваріації (mean-semiVariance frontier);
- Оптимізація вздовж межі середнього CVaR (mean-CVaR frontier);
- Побудова ефективної межі:
 - 1) Простий (з використанням CLA).
 - 2) Стриманий.
 - 3) Складні сюжети.

Downloading data

Для завантаження даних ми будемо використовувати `yfinance`, чудову бібліотеку, яка надає безкоштовні дані про ціни від Yahoo Finance, без ключа API.

Завантажуємо дані тікети на максимальний період:

"MSFT", "AMZN", "KO", "MA", "COST",
 "LUV", "XOM", "PFE", "JPM", "UNH",
 "ACN", "DIS", "GILD", "F", "TSLA"

Приклад деяких тікетів з цінами за 5 днів, див. таб. 3.1 ., та рис. 3.8:

Таблиця 3.1 - Приклад деяких тікетів з цінами за 5 днів

	ACN	AMZN	COST	DIS	F	GILD	JPM	KO	LUV
Date									
2021-11-26	353.600006	3504.560059	546.130005	148.110001	19.750000	70.269997	161.929993	53.316540	44.959999
2021-11-29	365.820007	3561.570068	554.880005	147.809998	19.670000	70.650002	161.229996	54.160000	44.540001
2021-11-30	357.399994	3507.070068	539.380005	144.899994	19.190001	68.930000	158.830002	52.450001	44.400002
2021-12-01	360.140015	3443.719971	529.840027	142.149994	19.580000	68.930000	157.889999	52.299999	42.549999
2021-12-02	366.015015	3479.423340	527.820007	145.772400	19.940201	69.320000	161.029999	53.049999	44.110001

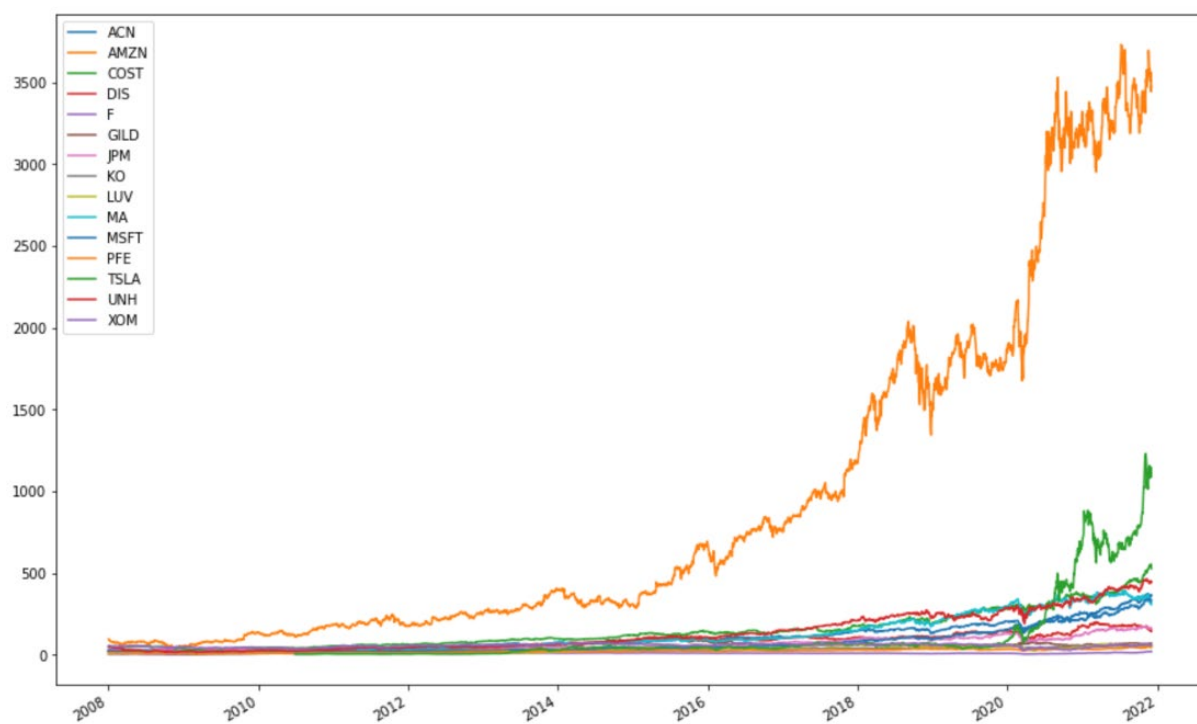


Рисунок 3.8 - Графік тікетів від 2008-01-01

Будуємо коваріаційну матрицю, див. рис. 3.9:

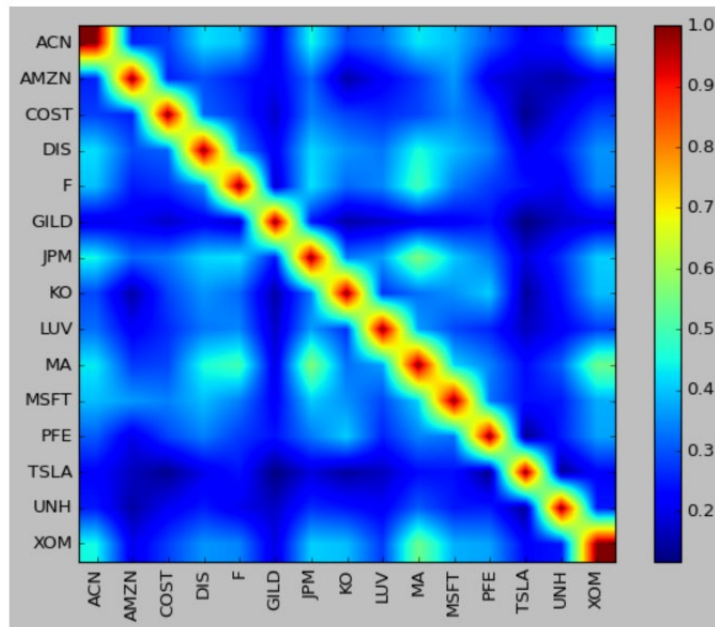


Рисунок 3.9 - Вибіркова коваріаційна матриця

З причин, які обговорювались в теорії, вибіркова коваріаційна матриця не повинна бути нашим вибором за замовчуванням. Я думаю, що кращим варіантом є усадка Ledoit-Wolf, яка зменшує екстремальні значення в матриці коваріації. На Рис. 3.10 ми бачимо, що за межами діагоналі менше яскравих плям:

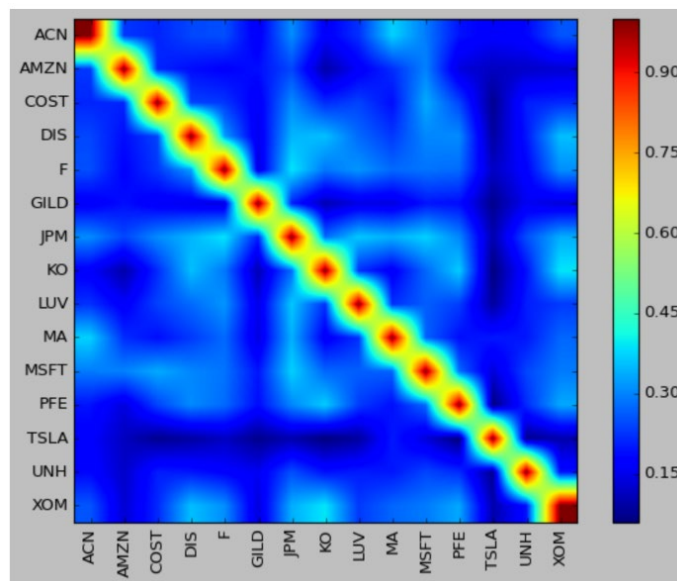


Рисунок 3.10 - Коваріаційна матриця Ledoit-Wolf

3.2.1 Дохідність портфеля

Як написано в розділі 1, часто буває поганою ідеєю надавати прибутки, використовуючи просту оцінку, як-от середнє минулого прибутку. Якщо у нас немає власного методу оцінки прибутку, дослідження показують, що нам краще не надавати очікувану прибутковість – можемо просто знайти портфель `minVolatility()` або скористатися HRP.

Однак у цьому прикладі ми будемо використовувати показники CAPM, які мають бути трохи більш стабільними, ніж середня історична прибутковість за замовчуванням. В першому підрозділі Моделі Повернення Ризику була надана додаткова інформація.

На рисунку 3.11 наведена дохідність тікетів

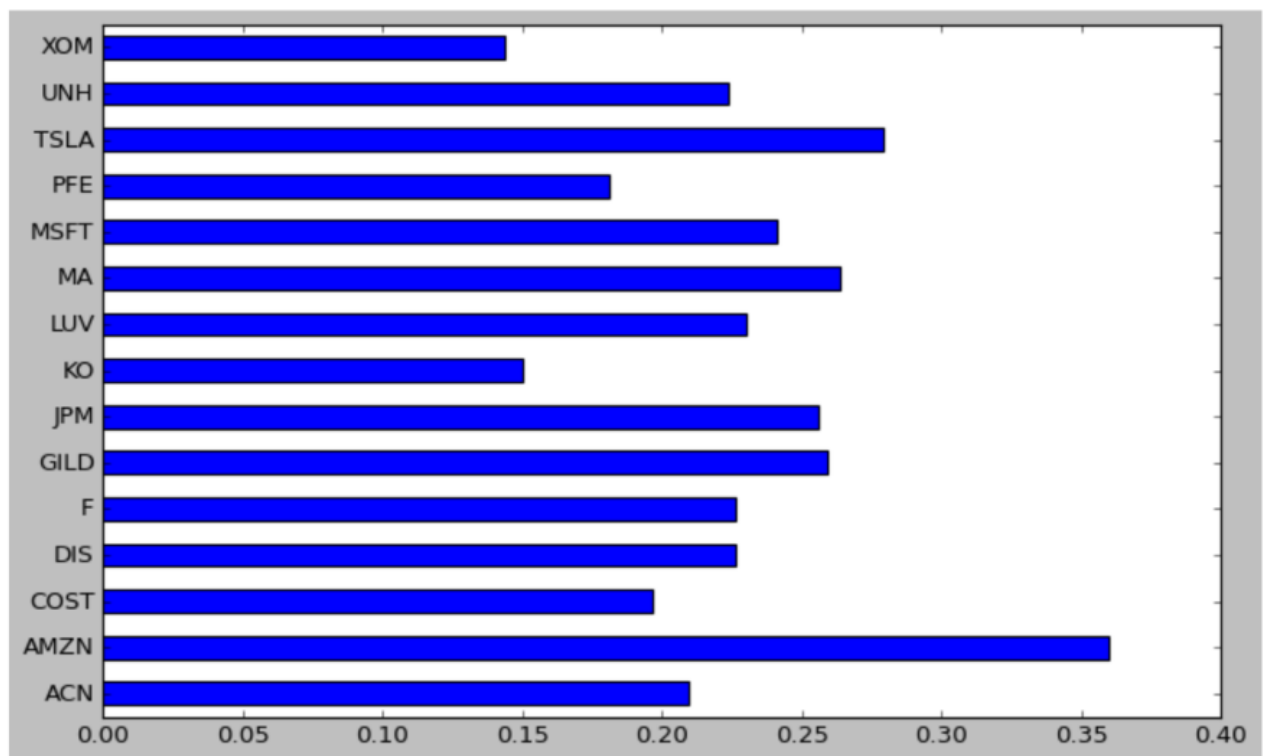


Рисунок 3.11 - Дохідність тікетів

ACN	0.209499
AMZN	0.360018
COST	0.196313
DIS	0.226497
F	0.226293
GILD	0.259228
JPM	0.255662
KO	0.150383
LUV	0.230029
MA	0.263939
MSFT	0.241181
PFE	0.181125
TSLA	0.279299
UNH	0.223879
XOM	0.143430

3.2.2 Long/short min variance

Тепер ми створюємо long/short портфель з метою мінімізації дисперсії. Існує чимало досліджень, які демонструють, що ці портфелі з глобальною мінімальною дисперсією (GMV) перевершують портфелі, оптимізовані за середньою дисперсією.

На Рис. 3.12 зображено Long Short портфель з метою мінімізації дисперсії

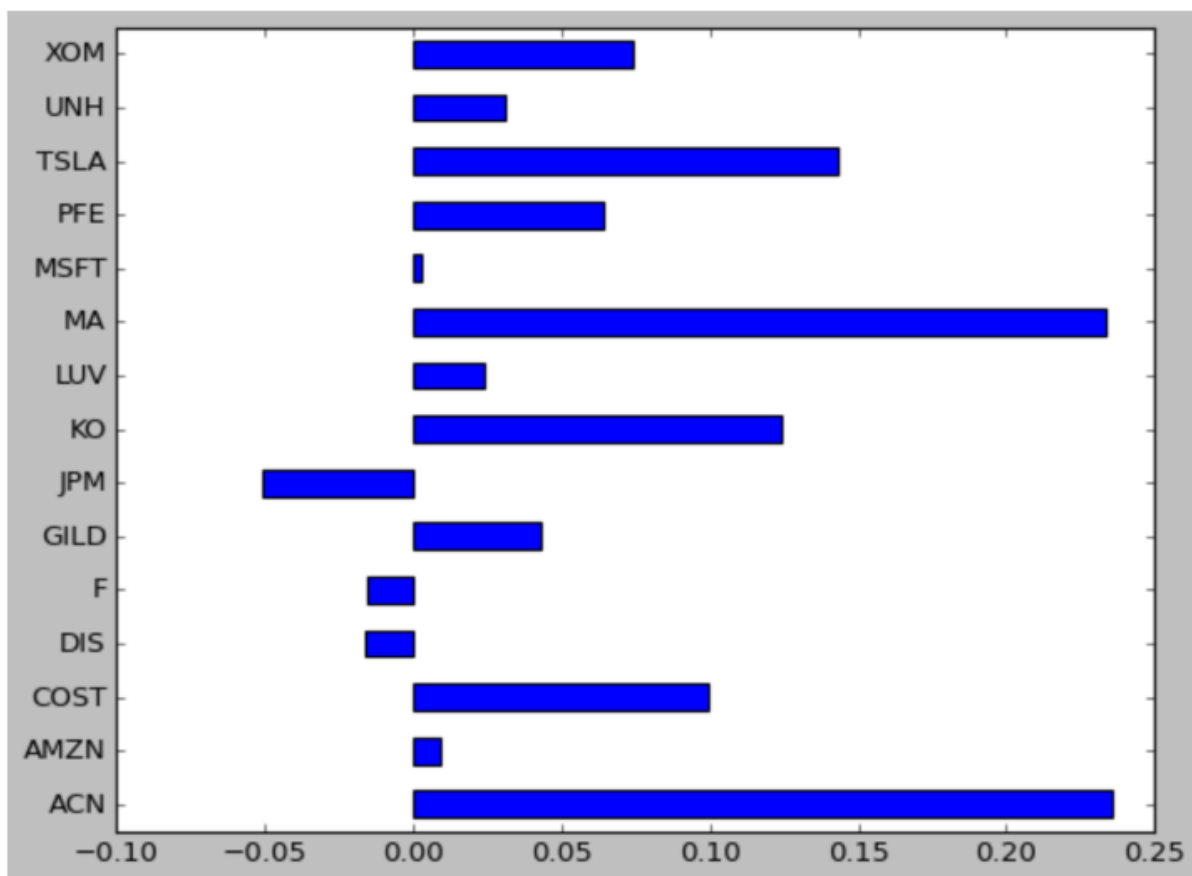


Рисунок 3.12 - Long Short портфель з метою мінімізації дисперсії

Ваги портфеля

('ACN', 0.23552),
 ('AMZN', 0.00892),
 ('COST', 0.09915),
 ('DIS', -0.01657),
 ('F', -0.01589),
 ('GILD', 0.04309),
 ('JPM', -0.05052),
 ('KO', 0.1241),
 ('LUV', 0.02389),
 ('MA', 0.23324),
 ('MSFT', 0.00256),
 ('PFE', 0.06432),

```
('TSLA', 0.14351),
('UNH', 0.0309),
('XOM', 0.07375)])
```

Ми можемо отримати швидку індикацію ефективності портфеля таким чином. Зауважу, що це приблизна оцінка і може мати дуже мало схожості з фактичною ефективністю портфеля

Annual volatility: 11.8%

3.2.3 Max Sharpe з обмеженнями по сектору

Ми можемо розглянути можливість максимізації коефіцієнта Шарпа. Це теоретично дає оптимальний портфель з точки зору ризиків-прибутків.

У цьому підрозділу ми створюємо довготривалий портфель maxSharp, але також включаємо секторні обмеження. Секторні обмеження вимагають трьох речей: sectorMapper, наші межі sectorNiwer і наші межі sectorUpper

```
sectorMapper = {
    "MSFT": "Tech",
    "AMZN": "Consumer Discretionary",
    "KO": "Consumer Staples",
    "MA": "Financial Services",
    "COST": "Consumer Staples",
    "LUV": "Aerospace",
    "XOM": "Energy",
    "PFE": "Healthcare",
    "JPM": "Financial Services",
    "UNH": "Healthcare",
    "ACN": "Misc",
```

```
"DIS": "Media",
"GILD": "Healthcare",
"F": "Auto",
"Tesla": "Auto"
```

```
sectorLower= {
    "Consumer Staples": 0.1, # принаймні 10% до основних продуктів
    "Tech": 0.05 # принаймні 5% до техніки
    # Для всіх інших секторів буде вважатися, що нижньої межі немає
}
```

```
sectorUpper = {
    "Tech": 0.2,
    "Aerospace":0.1,
    "Energy": 0.1,
    "Auto":0.15
}
```

Потім ми налаштовуємо оптимізатор і додаємо наші обмеження. Ми можемо використовувати `ef.addObjective()`, щоб додати інші обмеження. Наприклад, скажімо, що на додаток до вищезгаданих обмежень сектору, я спеціально хочу:

- 10% портфеля в AMZN
- Менше 5% мого портфеля в TSLA

Після використання оптимізатора отримуємо ваги, див. рис. 3.13:

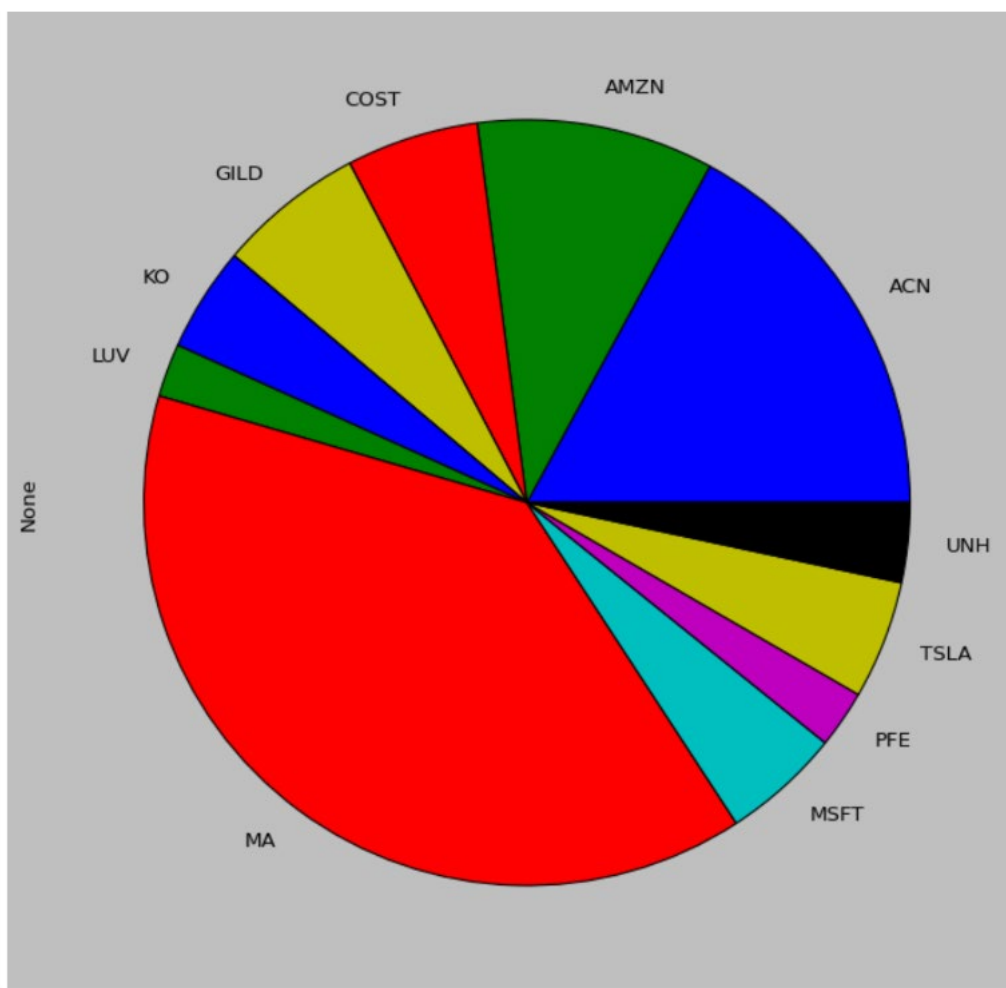


Рисунок 3.13 - Ваги

('ACN', 0.17073),
 ('AMZN', 0.1),
 ('COST', 0.05586),
 ('DIS', 0.0),
 ('F', 0.0),
 ('GILD', 0.06183),
 ('JPM', 0.0),
 ('KO', 0.04414),
 ('LUV', 0.02242),
 ('MA', 0.38709),
 ('MSFT', 0.05),
 ('PFE', 0.02437),

```
('TSLA', 0.05),
('UNH', 0.03357),
('XOM', 0.0)])
```

Ми можемо відразу побачити, що наші явні обмеження були задоволені, і можемо перевірити всі обмеження сектора:

```
Tech: 0.050
Consumer Staples: 0.100
Consumer Discretionary: 0.100
Financial Services: 0.387
Energy: 0.000
Healthcare: 0.120
Aerospace: 0.022
Misc: 0.171
Media: 0.000
Auto: 0.050
```

3.2.4 Максимізація прибутку для даного ризику з регуляризацією L2

Давайте уявімо, що ми багато подумали про свою толерантність до ризику і вирішили, що не можемо прийняти нічого більше, ніж 15% волатильності. Ми можемо використати це для створення портфеля, який максимізує прибуток для даного ризику (з тими ж застереженнями щодо очікуваної прибутковості).

Отримуємо:

```
OrderedDict([('ACN', 0.0),
('AMZN', 0.19117),
```

('COST', 0.1),
 ('DIS', 0.0),
 ('F', 0.0),
 ('GILD', 0.0627),
 ('JPM', 0.0),
 ('KO', 0.0),
 ('LUV', 0.0),
 ('MA', 0.44614),
 ('MSFT', 0.05),
 ('PFE', 0.0),
 ('TSLA', 0.15),
 ('UNH', 0.0),
 ('XOM', 0.0)])

9/15 tickers have zero weight

Наш портфель:

Expected annual return: 27.6%

Annual volatility: 15.0%

Sharpe Ratio: 1.71

Хоча цей портфель здається, що він відповідає нашим цілям, нас може турбувати той факт, що багатьом тикерам присвоєно нульову вагу. По суті, оптимізатор "переповнює" надані нами дані - ви, швидше за все, отримаєте кращі результати, застосувавши певний рівень диверсифікації. Один із способів зробити це — використовувати **регуляризацію L2** — по суті, додаючи штраф на кількість майже нульових ваг.

Отримуємо:

OrderedDict([('ACN', 0.04391),
 ('AMZN', 0.20663),
 ('COST', 0.08899),
 ('DIS', 0.01364),

```
(('F', 0.0),
 ('GILD', 0.08383),
 ('JPM', 0.0471),
 ('KO', 0.01101),
 ('LUV', 0.03044),
 ('MA', 0.24018),
 ('MSFT', 0.05),
 ('PFE', 0.0),
 ('TSLA', 0.15),
 ('UNH', 0.03427),
 ('XOM', 0.0))]
```

3/15 tickers have zero weight

Ми можемо налаштувати значення гами, щоб вибрати кількість ненульових тікерів. Більша гамма тягне ваги портфеля, див. рис. 3.14, до рівного розподілу.

```
OrderedDict([('ACN', 0.05499),
```

```
 ('AMZN', 0.13025),
 ('COST', 0.06148),
 ('DIS', 0.0635),
 ('F', 0.06176),
 ('GILD', 0.07987),
 ('JPM', 0.07808),
 ('KO', 0.03852),
 ('LUV', 0.06528),
 ('MA', 0.08226),
 ('MSFT', 0.07083),
 ('PFE', 0.04079),
 ('TSLA', 0.08824),
 ('UNH', 0.06219),
```

('XOM', 0.02196)])

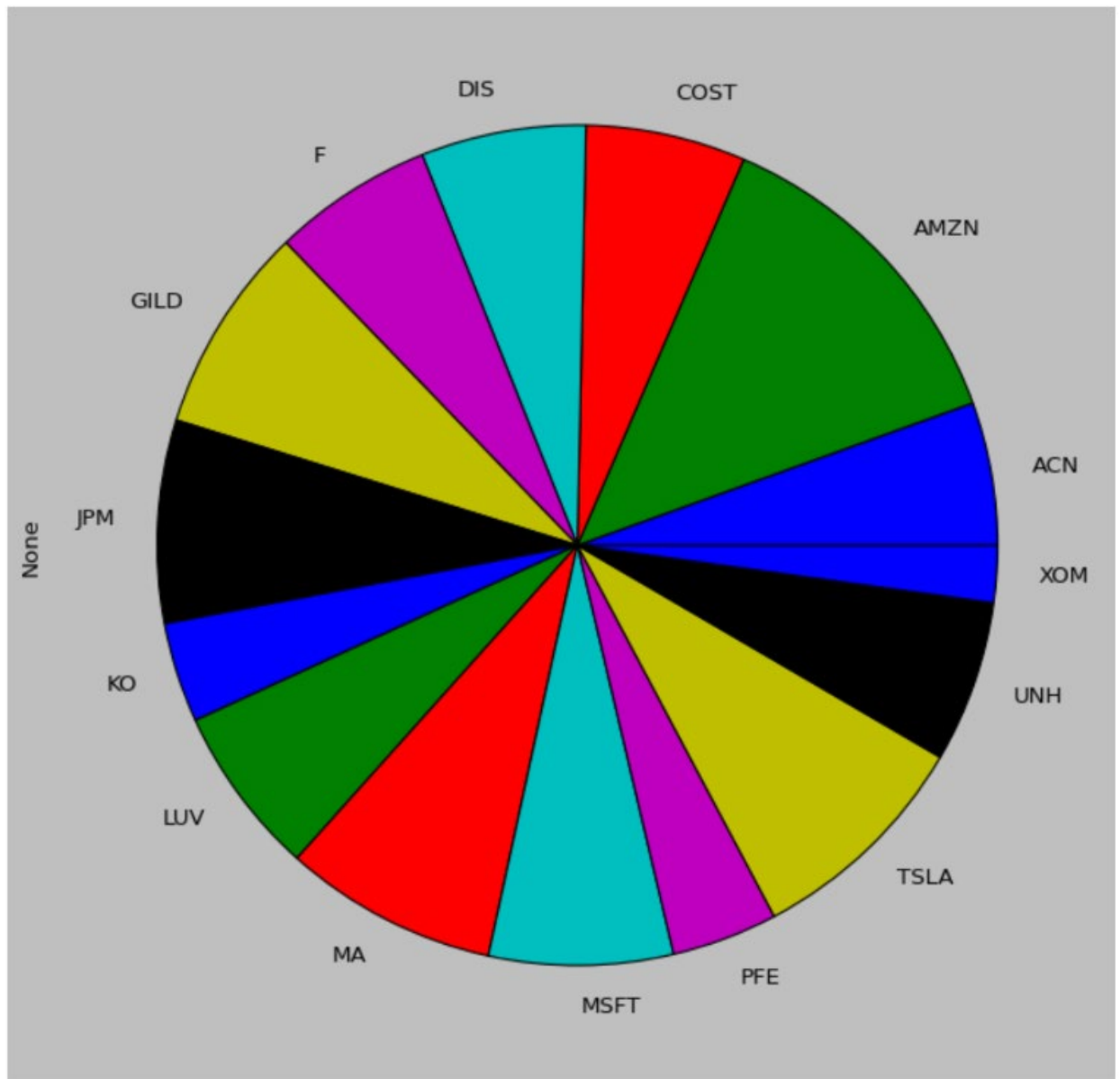


Рисунок 3.14 – Ваги

Expected annual return: 24.8%

Annual volatility: 14.8%

Sharpe Ratio: 1.54

Отриманий портфель все ще має волатильність менше, ніж наш ліміт у 15%. Коефіцієнт Шарпа у вибірці знизився, але цей портфель є набагато надійнішим для реальних інвестицій.

3.2.5 Мінімізація ризику для даної прибутковості, ринково нейтральний(market- neutral)

Натомість ми можемо опинитися в ситуації, коли маємо певну необхідну норму прибутку (можливо, ми є пенсійним фондом, який потребує 7% прибутку на рік), але ми хотіли б мінімізувати ризик. Крім того, припустимо, що ми хотіли б, щоб наш портфель був нейтральним до ринку в тому сенсі, що він однаково схильний до long або short сторони.

Не повинно бути обмежень ваги, щоб дозволити шорт

```
OrderedDict([('ACN', -0.03835),
             ('AMZN', 0.2214),
             ('COST', -0.06082),
             ('DIS', -0.00209),
             ('F', -0.00395),
             ('GILD', 0.04975),
             ('JPM', 0.04795),
             ('KO', -0.14077),
             ('LUV', 0.00181),
             ('MA', 0.06405),
             ('MSFT', 0.02035),
             ('PFE', -0.08482),
             ('TSLA', 0.08942),
             ('UNH', -0.00859),
             ('XOM', -0.15536)])
```

На рис. 3.15 зображено ваги без обмежень

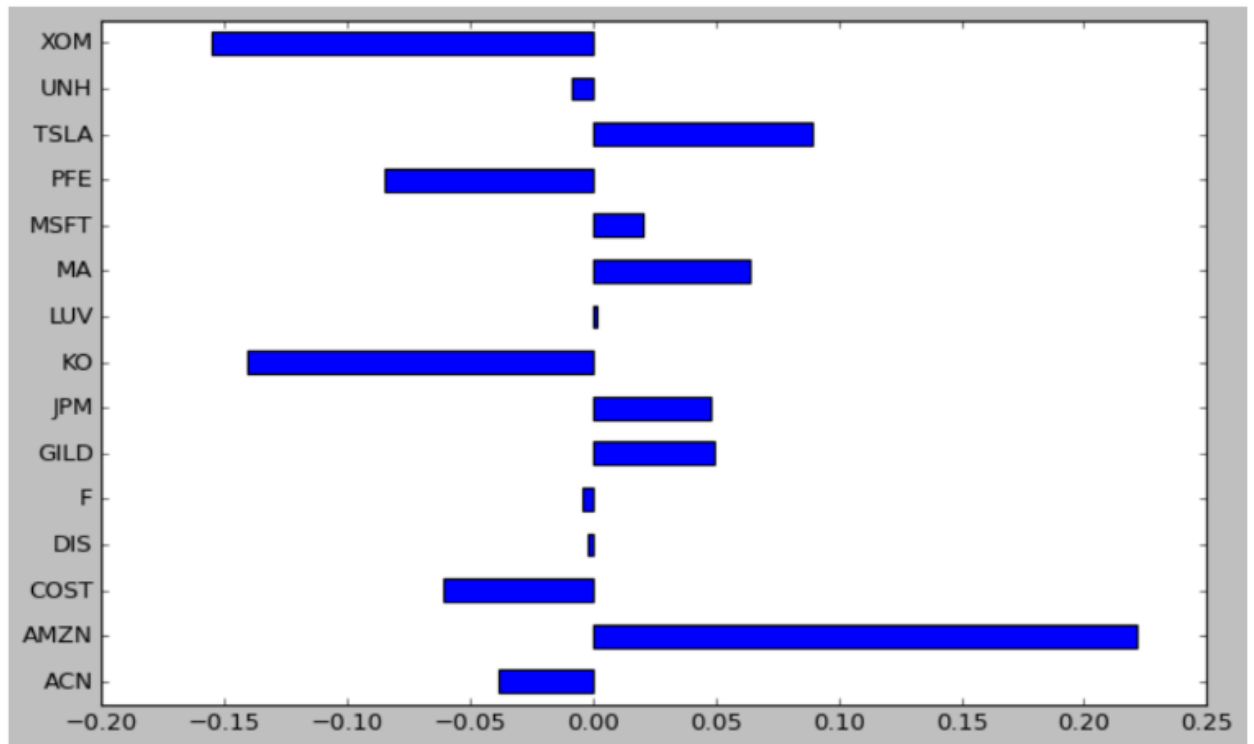


Рисунок 3.15 - Ваги без обмежень

Expected annual return: 7.0%

Annual volatility: 10.6%

Sharpe Ratio: 0.47

Net weight: -0.00

3.2.6 Ефективна оптимізація напівдисперсій (semi-variance)

У цьому прикладі ми зведемо до мінімуму напівваріантність портфеля (тобто волатильність знизу) за умови обмеження прибутковості (цільове значення 20%).

Насправді є два способи зробити це. Перший – це «інтуїтивно зрозумілий». Ми обчислюємо напівковаріаційну матрицю і передаємо її в EfficientFrontier (так само, як ми зробили б для експоненціальної матриці cov або зменшеної матриці Ледуа-Вольфа (Ledoit-Wolf shrunk matrix)).

На рис. 3.16 зображено ваги для інтуїтивно зрозумілого способу

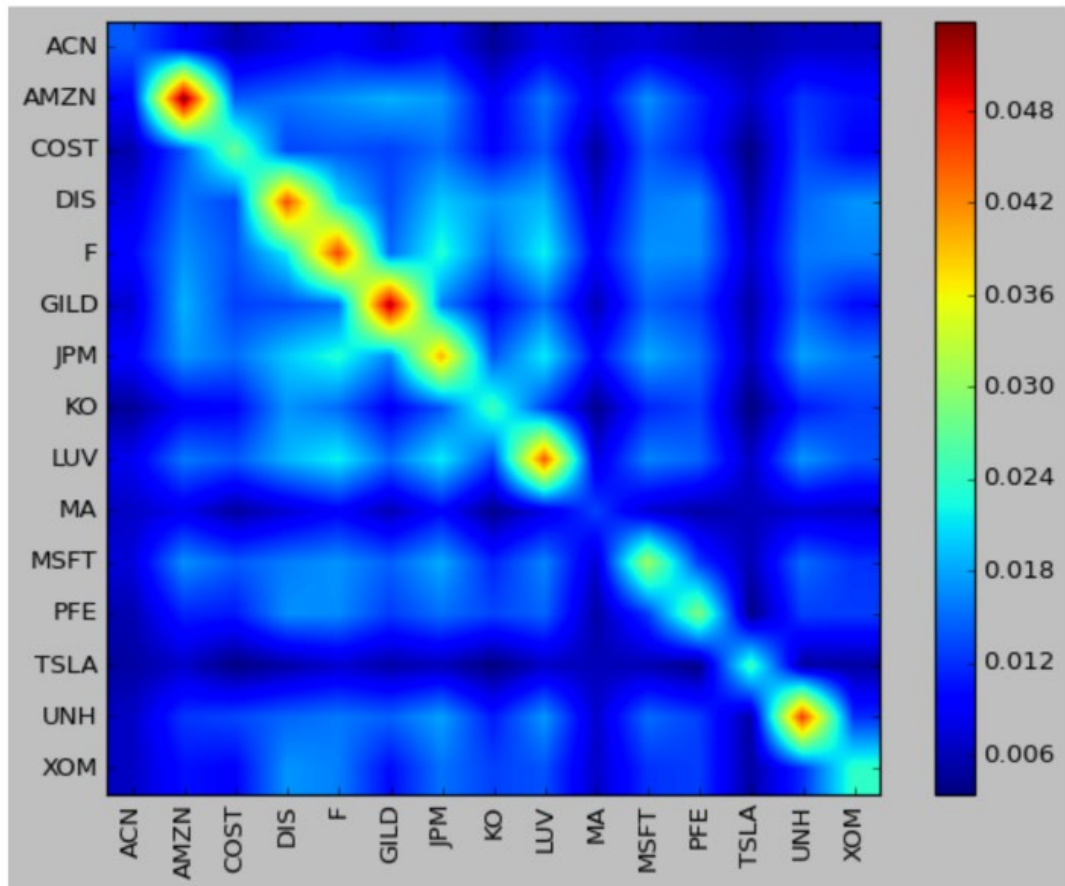


Рисунок 3.16 - Ваги

```
OrderedDict([('ACN', 0.26201),
             ('AMZN', 0.0),
             ('COST', 0.08926),
             ('DIS', 0.0),
             ('F', 0.0),
             ('GILD', 0.0086),
             ('JPM', 0.0),
             ('KO', 0.13019),
```

```
(('LUV', 0.0),
 ('MA', 0.3035),
 ('MSFT', 0.0),
 ('PFE', 0.04926),
 ('TSLA', 0.14987),
 ('UNH', 0.0),
 ('XOM', 0.00731]))
```

Expected annual return: 20%

Annual volatility: 8.9%

Sharpe Ratio: 1.73

Однак це рішення не є по-справжньому оптимальним у просторі середніх напівваріацій. Щоб зробити оптимізацію належним чином, ми повинні використовувати клас `EfficientsemiVariance`. Для цього потрібно спочатку обчислити прибуток і відкинути NaN.

```
OrderedDict([('ACN', 0.0),
 ('AMZN', 0.0888),
 ('COST', 0.30691),
 ('DIS', 0.0),
 ('F', 0.0),
 ('GILD', 0.07746),
 ('JPM', 0.0),
 ('KO', 0.30533),
 ('LUV', 0.0),
 ('MA', 0.0),
 ('MSFT', 0.0),
 ('PFE', 0.18291),
 ('TSLA', 0.0),
 ('UNH', 0.03858),
 ('XOM', 0.0)])
```

Щоб порівняти це з евристичним рішенням, я скористаюся швидким хаком: заміню `es.weights` на `ef.weights` і знову запускаю `es.portfolioPerformance`. В реальному житті так краще не робити.

Expected annual return: 20.7%

Annual semi-deviation (дохідність в найгіршому сценарії): 14.0%

Sharpe Ratio: 1.47

Ми бачимо, що евристичний метод має значно нижчий коефіцієнт Шарпа і набагато вищу напівваріантність.

3.2.7 Ефективна оптимізація CVaR

У цьому прикладі ми знайдемо портфель, який максимізує прибуток за умови обмеження CVaR.

Перш ніж це зробити, давайте спочатку обчислимо 95%-CVaR для портфеля max-sharp.

```
OrderedDict([('ACN', 0.0),
             ('AMZN', 0.0888),
             ('COST', 0.30691),
             ('DIS', 0.0),
             ('F', 0.0),
             ('GILD', 0.07746),
             ('JPM', 0.0),
             ('KO', 0.30533),
             ('LUV', 0.0),
             ('MA', 0.0),
             ('MSFT', 0.0),
             ('PFE', 0.18291),
             ('TSLA', 0.0),
```

('UNH', 0.03858),

('XOM', 0.0)])

Expected annual return: 25.4%

Annual volatility: 12.8%

Sharpe Ratio: 1.83

VaR: -2.11%

CVaR: -3.26%

Це значення CVaR означає, що наші середні втрати в найгірші 5% днів становитимуть -3,26%. Скажімо, це було за межами нашої зони комфорту (для портфеля \$100 000 це означало б втрати \ \$3260 за день).

Давайте спочатку побудуємо портфель з мінімальним CVaR:

OrderedDict([('ACN', 0.0),

('AMZN', 0.00788),

('COST', 0.30927),

('DIS', 0.0),

('F', 0.0),

('GILD', 0.07831),

('JPM', 0.0),

('KO', 0.36116),

('LUV', 0.0),

('MA', 0.0),

('MSFT', 0.0),

('PFE', 0.2205),

('TSLA', 0.0),

('UNH', 0.00972),

('XOM', 0.01315)])

Expected annual return: 18.2%

Conditional Value at Risk: 2.12%

Ми значно знизили CVaR, але ціною значного зниження прибутку. Ми можемо використовувати `efficientRisk`, щоб максимізувати прибуток для цільового ризику. Скажімо, 2,5% CVaR є прийнятним.

```
OrderedDict([('ACN', 0.0),
             ('AMZN', 0.29068),
             ('COST', 0.26255),
             ('DIS', 0.0),
             ('F', 0.0),
             ('GILD', 0.14348),
             ('JPM', 0.0),
             ('KO', 0.05767),
             ('LUV', 0.0),
             ('MA', 0.0),
             ('MSFT', 0.0),
             ('PFE', 0.14892),
             ('TSLA', 0.0),
             ('UNH', 0.0967),
             ('XOM', 0.0)])
```

Expected annual return: 25.1%

Conditional Value at Risk: 2.50%

Тепер ми маємо вище прибутковість (25,1% проти 18,2%), але з трохи вищим ризиком (2,50% CVaR проти 2.12%).

3.2.8 Побудова графіка - без обмежень (plot - Unconstrained)

Щоб побудувати безобмежений ефективний кордон, найпростіше використовувати алгоритм критичної лінії (critical line algorithm)

```
OrderedDict([('ACN', 0.15398),  
             ('AMZN', 0.0703),  
             ('COST', 0.05482),  
             ('DIS', 0.0),  
             ('F', 0.0),  
             ('GILD', 0.05632),  
             ('JPM', 0.0),  
             ('KO', 0.04193),  
             ('LUV', 0.02208),  
             ('MA', 0.32395),  
             ('MSFT', 0.01466),  
             ('PFE', 0.03307),  
             ('TSLA', 0.19663),  
             ('UNH', 0.03225),  
             ('XOM', 0.0)])
```

Expected annual return: 25.2%

Annual volatility: 12.7%

Sharpe Ratio: 1.82

На Рис. 3.17 Зображено приклад побудови графіка - без обмежень

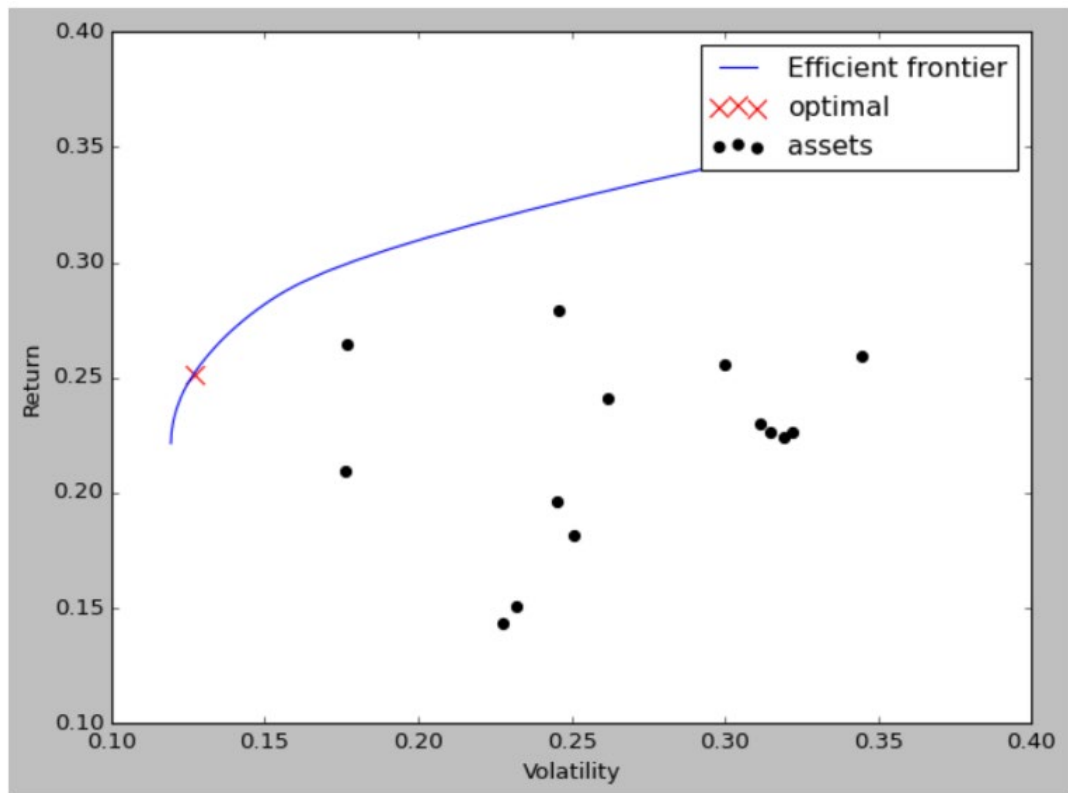


Рисунок 3.17 - Приклад побудови графіка - без обмежень

3.2.9 Побудова графіка - з обмеженнями (plot - Constrained)

У цьому прикладі ми намалюємо ефективний кордон, що відповідає портфелям з обмеженням впливу MSFT, AMZN і TSLA (наприклад, можливо, ми хочемо уникнути великих технологій).

Модуль `plot` дозволяє нам передати `'ax'`, поверх якого додаються графіки.

Це дозволяє побудувати складні сюжети.

У цьому прикладі ми побудуємо ефективну границю, а також 10 000 змодельованих портфелів.

Щоб створити смодельовані портфелі, ми будемо вибирати випадкові weight з розподілу Дирихле (вони вже нормалізовані):

Sample portfolio returns:

[0.23084438 0.22265398 0.22472077 ... 0.2626649 0.22374024 0.22090538]

Sample portfolio volatilities:

0 0.139204

1 0.157438

2 0.166383

3 0.133365

4 0.178652

...

9995 0.152468

9996 0.155733

9997 0.147559

9998 0.151746

9999 0.138154

Length: 10000, dtype: float64

На Рис. 3.18 Зображено приклад побудови графіка – з обмеженнями

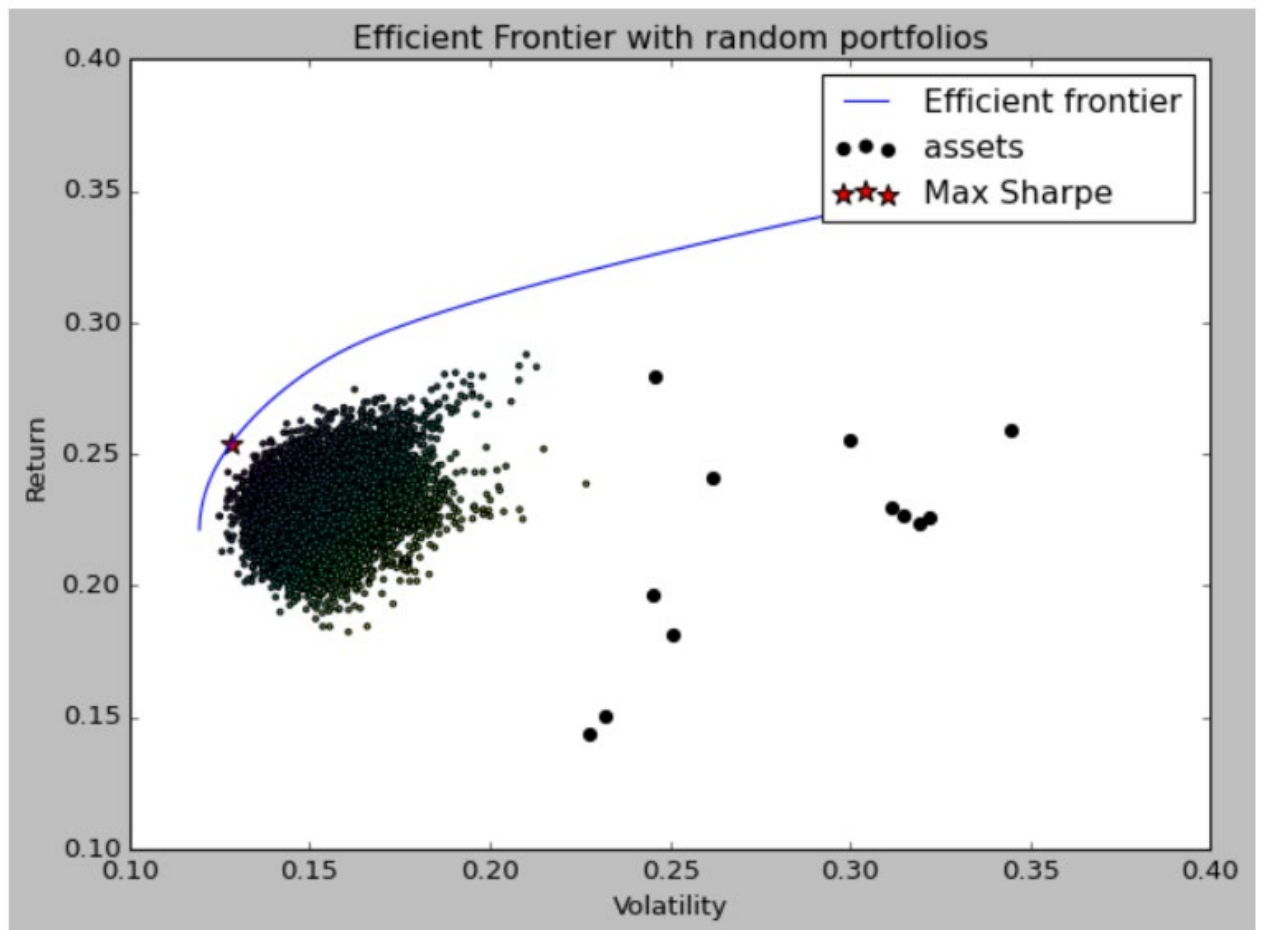


Рисунок 3.18 - Приклад побудови графіка – з обмеженнями

Наш портфель:

```
OrderedDict([('ACN', 0.14553),
             ('AMZN', 0.07664),
             ('COST', 0.05073),
             ('DIS', 0.00165),
             ('F', 0.0),
             ('GILD', 0.05769),
             ('JPM', 0.0),
             ('KO', 0.03263),
             ('LUV', 0.02272),
             ('MA', 0.33277),
             ('MSFT', 0.01619),
             ('PFE', 0.02962),
```

('TSLA', 0.20134),

('UNH', 0.03248),

('XOM', 0.0)])

Expected annual return: 25.4%

Annual volatility: 12.8%

Sharpe Ratio: 1.83

3.3 Розподіл Black-Litterman

Метод Блека-Літтермана – дуже ефективний спосіб конвертувати наші погляди на доходність активів, а також нашу невпевненість у цих поглядах у портфелі. Опис теорії привів в першому розділі дисертації

У цій книзі ми розглядаємо:

- Загрузка даних по методу Блека-Літтермана;
- Побудова вектора априорної доходності на основі ринкової рівноваги;
- Два способи побудови матриці невизначеності;
- Комбінування Блека-Літтермана з оптимізацією середньої дисперсії;
- Загрузка даних.

Для доповнення до цінових даних для побудови ринкового потрібна ринкова капіталізація.

Будемо використовувати тікети, див. рис. 3.19:

[«MSFT», «AMZN», «NAT», «BAC», «DPZ», «DIS», «KO», «MCD», «COST», «SBUX»]

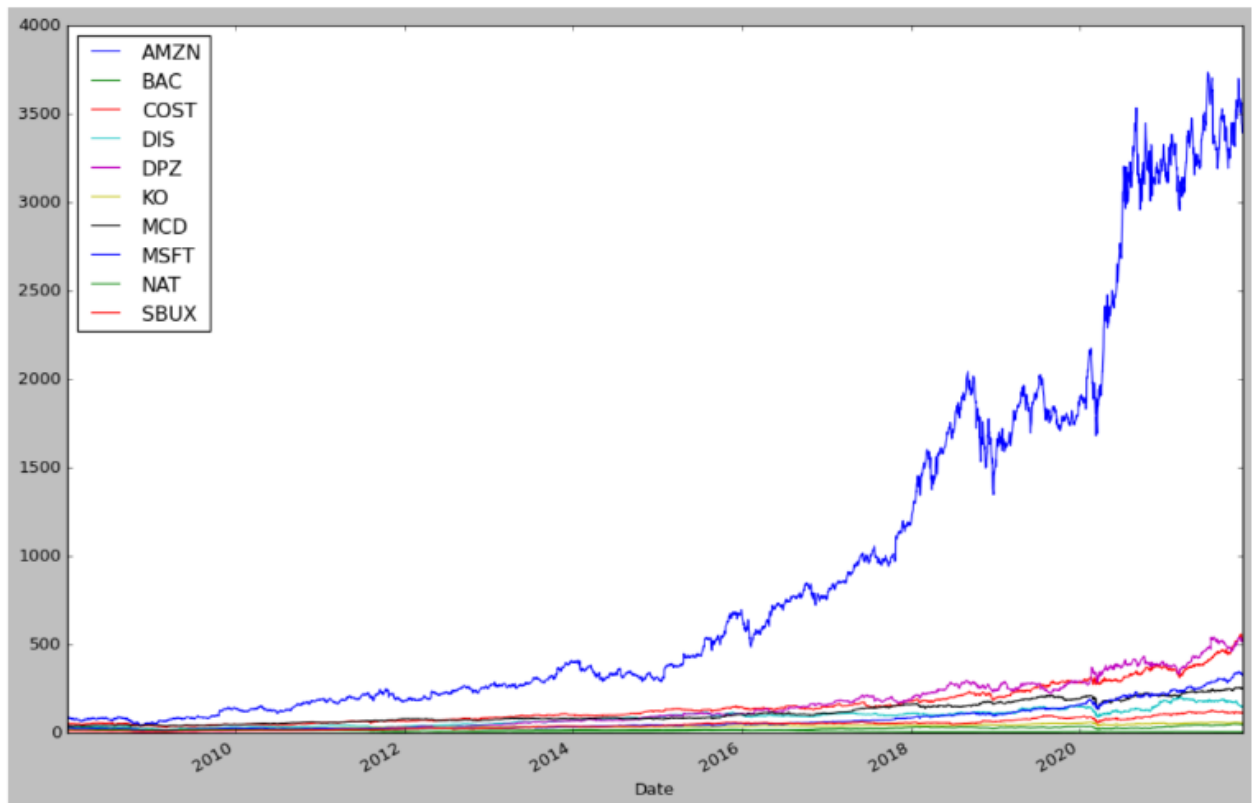


Рисунок 3.19 - Графік тікетів від 2008-01-01

Капіталізація.

'MSFT': 2473804431360,
 'AMZN': 1719125278720,
 'NAT': 309535200,
 'BAC': 359035568128,
 'DPZ': 18971713536,
 'DIS': 265778249728,
 'KO': 231261749248,
 'MCD': 186310590464,
 'COST': 233693970432,
 'SBUX': 130718121984}

Далі рахуємо передбачуваний ринковий параметр неприйняття (тобто ринкова ціна ризику) виходячи з ринкових цін. Наприклад, якщо над ринком

надлишкова доходність 10% на рік. При 5% дисперсії параметр неприйняття ризику дорівнює 2, тобто нам необхідно отримати 2-кратну компенсацію

$$\delta = \frac{R - R_f}{\sigma^2}$$

2.7695498654681727

На рис. 3.20 зображена коваріаційна матриця Black-Litterman

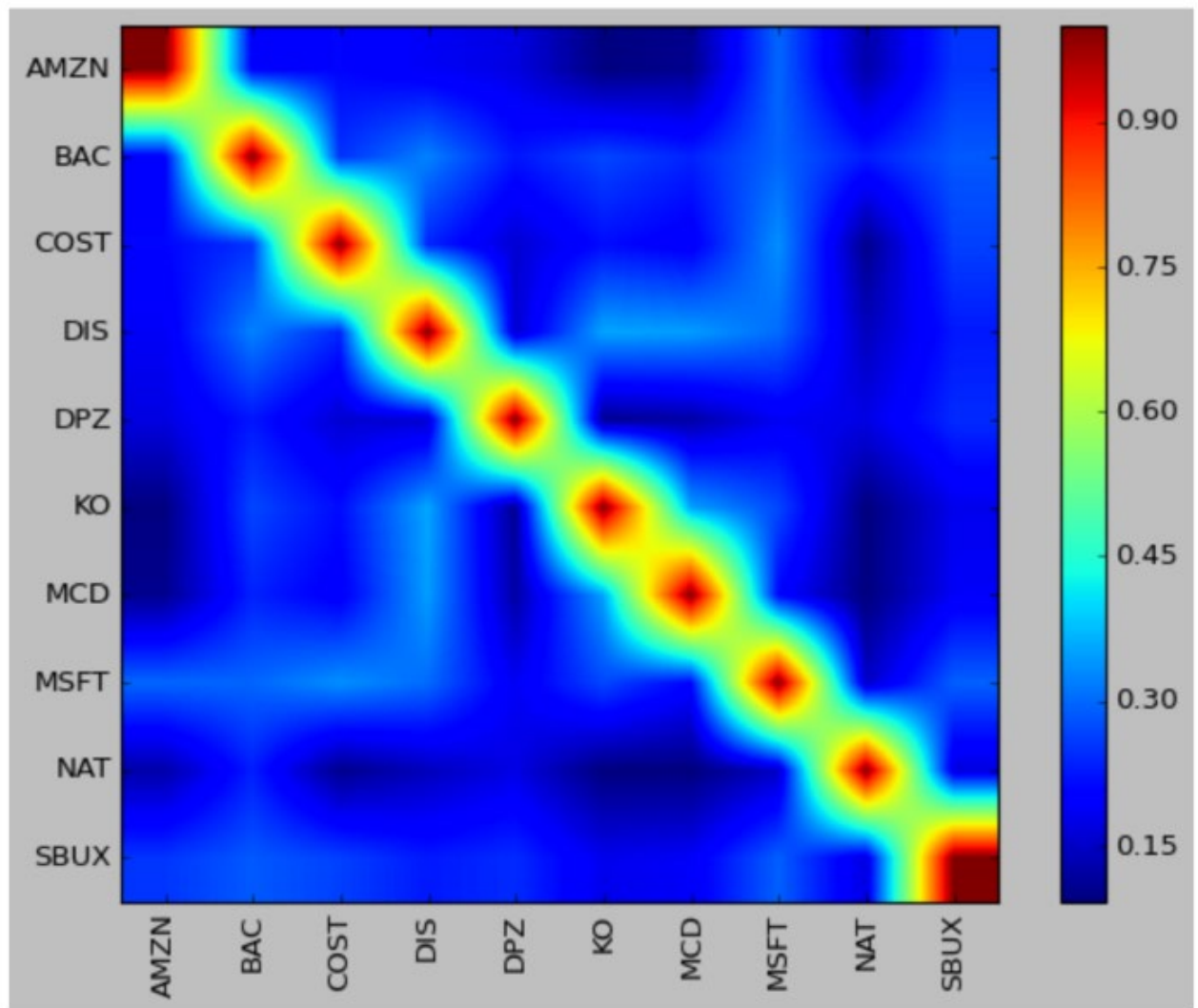


Рисунок 3.20 - Коваріаційна матриця Black-Litterman

Далі обчислюємо попередню оцінку прибутковості, див. рис. 3.21, передбачувану ринковими вагами.

Іншими словами, враховуючи внесок кожного активу у ризик ринку портфель, скільки ми очікуємо отримати компенсацію?

Отримуємо:

AMZN 0.184110
 BAC 0.108541
 COST 0.079417
 DIS 0.097353
 DPZ 0.049185
 KO 0.064613
 MCD 0.069164
 MSFT 0.143231
 NAT 0.053498
 SBUX 0.082697

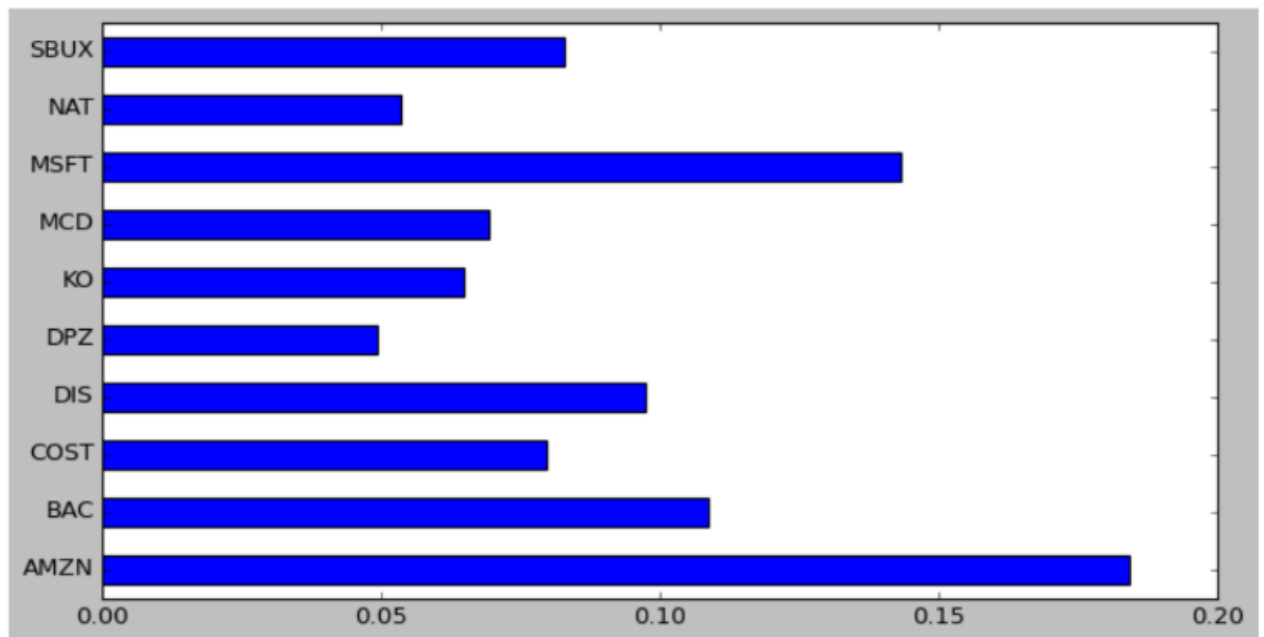


Рисунок 3.21 - Попередня оцінка прибутковості

У методі BL уявлення задаються через матрицю P (матрицю вибору) і вектор Q .

Q містить величину кожного уявлення, тоді як P зіставляє уявлення з активами, яким вони належать.

Якщо ми надаємо ****absolute views**** (тобто оцінку повернення для кожного активу), то нам не потрібно турбуватися про P та Q, ми можемо просто передати свої перегляди у вигляді словника.

Нам не обов'язково використовувати перегляди всіх активів

```
viewdict = {
    "AMZN": 0.10,
    "BAC": 0.30,
    "COST": 0.05,
    "DIS": 0.05,
    «DPZ»: 0.20,
    «KO»: -0.05, # Я гадаю, що Coca-Cola впаде на 5%
    «MCD»: 0.15,
    «MSFT»: 0.10,
    «NAT»: 0.50, # але низька достовірність, яка буде відображена пізніше
    «SBUX»: 0.10
```

Перегляд достовірності

У цьому розділі ми пропонуємо два способи побудови матриці невизначеності.

Перший відомий як метод Ідзорека.

Це дозволяє нам вказати вектор/список процентної достовірності, див. рис. 3.22.

```
Confidences = [
    0.6,
    0.4,
    0.2,
    0.5,
    0.7, # впевнений в dominos
    0.7, # впевнений в КО
    0.7,
```

0.5,
0.1,
0.4
]

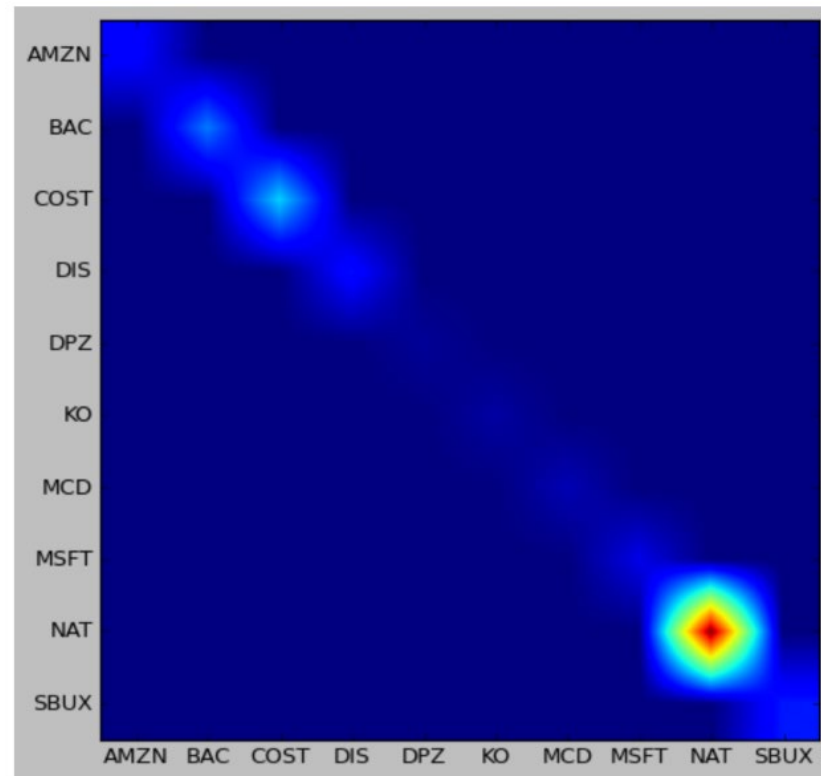


Рисунок 3.22 - Матриця довіри

```
array([0.00455272, 0.00893248, 0.01203356, 0.00495883, 0.00081642,  
0.00115707, 0.00173526, 0.00342652, 0.03609055, 0.00535626])
```

Потрібно звернути увагу, як NAT, якому ми надавали найменше довері, а також він має найвищу невизначеність.

Замість того, щоб ввести достовірність, ми можемо розрахувати матрицю невизначеності на пряму, показав 1 довірливий інтервал стандартного відхилення, тобто кордони, які, як ми вважаємо, будуть містити істинну доходність у 68% випадків.

Це може бути простіше, чим уявляти кілька вільних відсотків упевненості
intervals = [

(0, 0.25),
 (0.1, 0.4),
 (-0.1, 0.15),
 (-0.05, 0.1),
 (0.15, 0.25),
 (-0.1, 0),
 (0.1, 0.2),
 (0.08, 0.12),
 (0.1, 0.9),
 (0, 0.3)

]

Отримуємо дисперсії:

[0.015625, 0.022500000000000006, 0.015625, 0.0056250000000000015,
 0.0025000000000000005, 0.

3.3.1 Попередні оцінки

Враховуючи вихідні дані, ми можемо обчислити апостеріорну оцінку доходності (posterior estimate of returns)

AMZN 0.156831
 BAC 0.136345
 COST 0.061731
 DIS 0.072841
 DPZ 0.102431
 KO 0.007294
 MCD 0.103989
 MSFT 0.103975

NAT 0.061132

SBUX 0.082812

Ми можемо візуалізувати, як це співвідноситься з попередніми та нашими поглядами, див таб.3.2 та рис. 3.23:

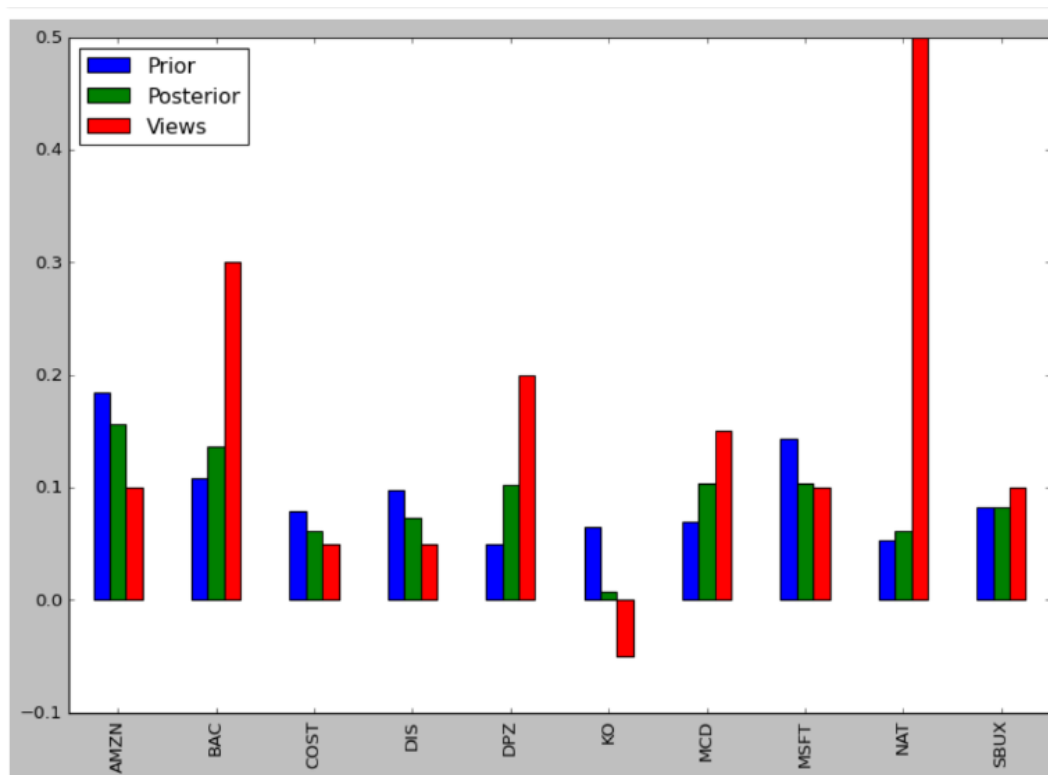


Рисунок 3.23 - Оцінка доходності prior, posterior та views

Таблиця 3.2 - Оцінка доходності prior, posterior та views

	Prior	Posterior	Views
AMZN	0.184110	0.156831	0.10
BAC	0.108541	0.136345	0.30
COST	0.079417	0.061731	0.05
DIS	0.097353	0.072841	0.05
DPZ	0.049185	0.102431	0.20
KO	0.064613	0.007294	-0.05
MCD	0.069164	0.103989	0.15
MSFT	0.143231	0.103975	0.10
NAT	0.053498	0.061132	0.50
SBUX	0.082697	0.082812	0.10

Зверніть увагу, що posterior часто знаходиться між попереднім (prior) та переглядом (views).

Це підтверджує той факт, що метод BL по суті є байєсовським середньозваженим значенням апіорних і уявлень, де вага визначається достовірністю.

Аналогічну, але менш інтуїтивну процедуру можна використовувати для отримання оцінки апостеріорної коваріації, див. рис. 3.24:

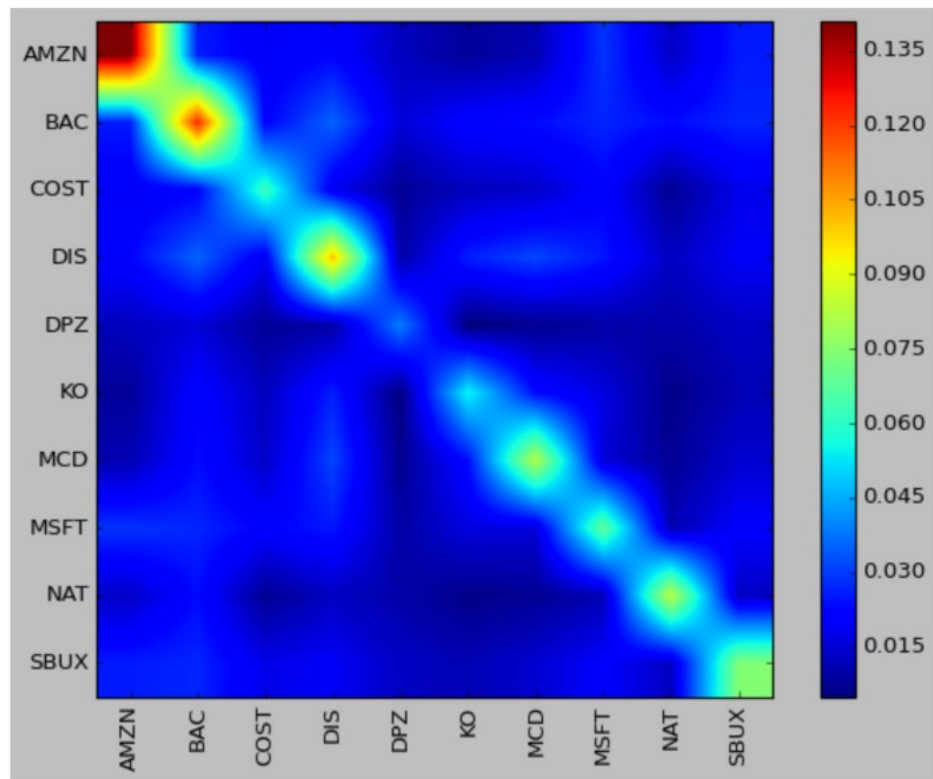


Рисунок 3.24 - Оцінка апостеріорної коваріації:

3.3.2 Розподіл портфеля

Тепер, коли ми побудували апостеріорну оцінку Блека-Літтермана, ми можемо перейти до використання будь-якого оптимізатора, описаного в попередніх розділах. Використаємо `ef.addObjective(objectiveFunctions.L2Reg)` (щоб було чим менше нульових портфелів) та `ef.maxSharpe()`

Отримуємо:

OrderedDict([('AMZN', 0.19754),

('BAC', 0.16477),

('COST', 0.05443),

('DIS', 0.06489),

('DPZ', 0.13068),

('KO', 0.0),

('MCD', 0.12469),

('MSFT', 0.12138),

('NAT', 0.05553),

('SBUX', 0.0861)])

На Рис. 3.25 Зображена оцінка апостеріорної коваріації

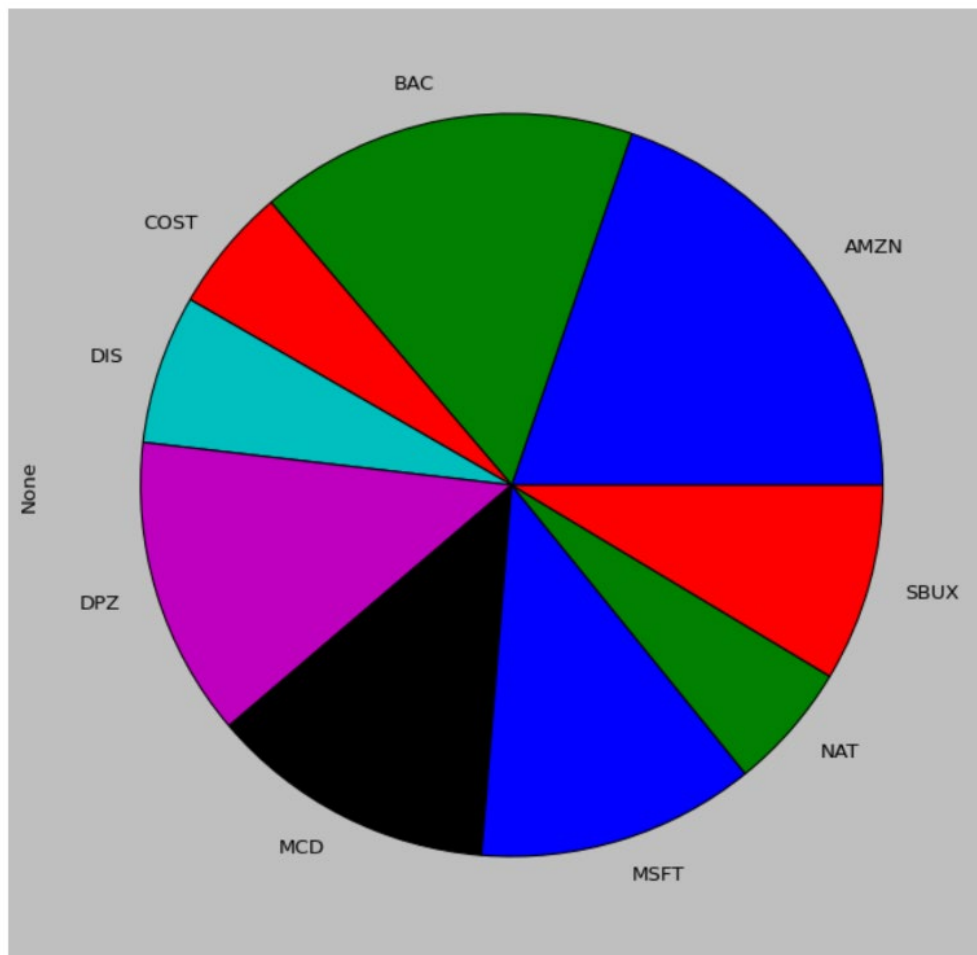


Рисунок 3.25 - Оцінка апостеріорної коваріації

Якби у нас було 20000\$, то потрібно було б купити стільки акцій для нашого портфеля (використанням цілісного програмування)

Залишки: \$74.45

```
{'AMZN': 1,
 'BAC': 76,
 'COST': 2,
 'DIS': 9,
 'DPZ': 5,
 'MCD': 11,
 'MSFT': 8,
 'NAT': 618,
 'SBUX': 16}
```

3.4 Висновки до розділу

Скажімо, ми провели тести, і результати не вражають. Що варто спробувати?

Можемо спробувати модель ієрархічного паритету ризиків— яка, здається, надійно перевершує оптимізацію середньої дисперсії поза вибіркою.

Можемо використовувати модель Блека-Літтермана, щоб побудувати більш стабільну модель очікуваного доходу. Крім того, просто потрібно відмовитися від очікуваного доходу!

Існує велика кількість досліджень, які свідчать про те, що портфелі з мінімальною дисперсією (`ef.minVolatility()`) постійно перевершують портфелі з максимальним коефіцієнтом Шарпа поза вибіркою (навіть якщо вони виміряні

коефіцієнтом Шарпа), через складність прогнозування очікуваної прибутковості.

Можемо спробувати різні моделі ризику: відомо, що моделі усадки мають кращі числові властивості порівняно з коваріаційною матрицею вибірки.

Можемо додати нові цілі або обмеження. Налаштувати параметр регуляризації L2, щоб побачити, як диверсифікація впливає на продуктивність.

РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

4.1 Опис ідеї проекту

Все починається з ідеї. Ідея має вирішувати певні задачі кардинально новим або оригінальним шляхом та підвищувати ефективність виконання.

Ідея проекту є розробка зручного та багатофункціонального додатку для інвесторів, які бажають аналізувати можливі портфелі, та знаходити саме ті, які задовольняють потреби.

Використання такої програми мінімізує помилки, дасть змогу контролювати, порівнювати різні методи оптимізації. Економить багато часу, оскільки знаходиться в єдиному місці. У таблиці 4.1 описано зміст ідеї стартап-проекту.

Таблиця 4.1 - Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Ідеєю стартап-проекту є розробка додатку для оптимізації портфелів що представлені даними котрі беруться з YahooFinance. Використовуються різні моделі для побудова різноманітних портфелів від менш ризикованих, до портфелів з більшою дохідністю. Порівняння портфелів за критеріями якості. Вибір найкращої моделі та побудова прогнозу за обраною. Користувач матиме змогу скоригувати параметри портфеля або обрати іншу модель для прогнозу.	Застосування в фінансових фондах для проведення інтелектуального дослідження можливих варіантів оптимізації портфелів.	Автоматичний розрахунок, що зменшує інтелектуальне навантаження на користувача.
	Застосування в приватному бізнесі для вироблення стратегії, прогнозування прибутків або втрат, оцінка ризиків.	Безліч способів прогнозування
		Збереження портфелів, що дає можливість поділитися з іншими інвесторами
		Можливість автоматичного оновлення прогнозу.

На сучасному ринку є багато продуктів які вирішують подібні задачі.

Всі вони конкурують між собою та мають певні переваги та недоліки. Важливо створити продукт, який розширить функціонал аналогів, буде технологічно стабільним та швидким.

Всі вони пропонують схожі послуги, тому для узагальненого порівняння поєднуємо їх концепції.

У таблиці 4.2 наведений аналіз потенційних технічних та економічних переваг ідеї порівняно з деякими відомими пропозиціями.

Таблиця 4.2 - Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Характерис тика	Концепція		W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент			
1.	Вартість	1000\$	2000\$			доступ ність
2.	Пробний період	1 місяць	1 місяць		тестування системи	
3.	Вибір платформи	На власному ПК, потрібен лише доступ до WI-FI	Лише в хмарі			можлив ість керуват и ресурса ми
4.	Інтеграція з іншими системами	немає	JSON	Реалізаці я в нових версіях		

Перелік сильних та слабких сторін дає змогу оцінити потреби користувачів конкурентоспроможність ідеї.

Проаналізувавши ринок можна зробити висновок, що користувачі потребують зрозумілої та доступної для досягнення поставлених цілей. Тому стартап-проект має узагальнити функціонал аналогів та спростити інтерфейс.

4.2 Технологічний аудит ідеї проекту

Сучасна індустрія інформаційних технологій налічує в собі велику кількість технологій та способів реалізації продукту. Ідеєю стартап-проекту є створення додатку, котрий можливо розвернути на сервері. Через обмеженість ресурсів потрібно обрати безкоштовні варіанти. Тому найкращим варіантом буде розробка додатку за допомогою технологій Jupiter на мові програмування Python. Проект буде розміщений на локальному сервері Windows Server та використовувати базу даних Yahoo Finance. У таблиці 4.3 наведено детальний опис доступності обраних технологій.

Таблиця 4.3 - Технологічна здійсненність ідеї проекту

Ідея проекту	Технології реалізації	Наявність технологій	Доступність технологій
Середовище розробки	JetBrains PyCharm 2019.2.4	Розробник Microsoft	Є безкоштовна версія
Веб додаток	Веб сервер: Jupiter	Розробник Microsoft	Безкоштовна
Збереження історії оптимізації	База даних: CookBook Jupiter	Розробник Microsoft	Безкоштовна
Складні обчислення	Оперативна пам'ять	Наявна	Ціна залежить від об'єму

Для розробки програмного продукту застосовуються популярні технології, які дозволяють проводити складні обчислення. Встановлення продукту на сервер дозволяє легко керувати ресурсами та виправляти помилки.

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Для того, щоб спланувати шляхи розвитку проекту потрібно правильно визначити ринкові можливості та ринкові загрози. Оцінка стану ринкового середовища може бути корисною під час впровадження проекту на ринку та для аналізу загроз, які можуть перешкодити реалізації проекту.

У таблиці 4.4 наведений аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку.

Таблиця 4.4 - Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	5
2	Загальний обсяг продаж, грн/ум. од	200000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	40

Як бачимо з таблиці, ринок інформаційних послуг в Україні та й у світі стрімко розвивається, немає обмежень для входу і є привабливим реалізації нових проектів.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи.

Результати наведені у таблиці 4.5.

Таблиця 4.5 - Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Автоматизація оптимізації портфелів, пошук найкращих варіантів	Фонди, Банки, Державні установи, Компанії, Біржі	Різні підходи до прогнозування та оптимізацій. Деяким портібно більше дохідності, деяким менше ризику.	Зрозумілий користувацький інтерфейс, швидкість роботи програми, можливість обрати кращі методи оптимізації, можливість переглянути історію портфелів.

Після визначення потенційних груп клієнтів переходимо до аналізу ринкового середовища.

У таблиці 4.6 наведено фактори, що перешкоджають ринковому впровадженню проекту.

У таблиці 4.7 наведено фактори, що йому сприяють. Фактори в таблиці подані в порядку зменшення значущості.

Таблиця 4.6 - Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Нестача технічних ресурсів	Поганий вплив на швидкість роботи програми, можливі неточні прогнози оптимізації, обмеження завантаження великого об'єму даних.	Заміна серверів на більш нові, додавання ресурсів пам'яті та встановлення потужних процесорів.
2	Нестабільна економічна ситуація в країні	Встановлення високого податку на продаж продукту.	Підвищення ціни на продукт в умовах кризи, вихід на глобальний ринок.
3	Конкуренція	На ринку з'являються нові продукти, більш вигідні користувачам	Встановлення реферальної програми, знижки для великого бізнесу, швидка реакція підтримки.
4	Зміна вимог користувача	Необхідність в новому функціоналі	Постійне оновлення та розширення функціоналу згідно нових вимог ринку.

Таблиця 4.7 - Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Зростання попиту на інформаційні продукти	Все більше компаній та інвесторів потребуватимуть програмні продукти.	Залучення нових інвесторів, розвиток реклами.
2	Глобалізація ринку	Нові ринки збуту, міжнародні клієнти.	Переклад інтерфейсу на популярні мови такі як англійська, китайська, встановлення цін у різних валютах.
3	Поява нових технологій	Нові можливості для обробки даних.	Використання нових технологій у своєму продукті.
4	Винайдення нових методів прогнозування	Математичні досягнення для точних прогнозів.	Включення в свій продукт найбільш точних методів та їх комбінація.

Ступеневий аналіз конкуренції на ринку представлений у таблиці 4.8. Він представлений у вигляді згрупованих, найбільш цікавих для нашого проекту показників, що впливають на процеси.

Таблиця 4.8 - Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - чиста	Незалежна конкуренція між компаніями, встановлення самостійної ціни, що не контролюється ринком та курсом валют.	Створення унікального продукту, підтримка якості.
2. За рівнем конкурентної боротьби - міжнародний	Інформаційні продукти не обмежуються країною тому іноземні конкуренти становлять загрозу.	Моніторинг появи нових компаній, встановлення ціни, що не перевищує конкурентів.
3. За галузевою ознакою - міжгалузева	Застосування програми у різних галузях діяльності.	Розвиток продукту для застосування у таких сферах як економіка, екологія, медицина, політика.
4. Конкуренція за видами товарів - товарно-родова	Можлива поява нових продуктів з додатковим функціоналом	У створенні продукту враховувати недоліки конкурентів
5. За характером конкурентних переваг - нецінова	Впровадження нових технологій для розвитку продукту	Пошук нових швидких та якісних технологій для зменшення собівартості продукту.

Продовження таблиці 4.8

6.	За інтенсивністю - не марочна	Торгова марка має мінімальний вплив	Розвиток реклами, участь у конференціях.
----	-------------------------------	-------------------------------------	--

Відомий маркетолог М. Пропер наводив канонічні параметри, за якими аналізувались галузі конкуренції.

Для нашого проекту наведено та проаналізовано наступні параметри (таблиця 4.9).

Таблиця 4.9 - Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Investing.com	Investopedia		Фонди, Банки, Державні установи, Компанії, Біржі	Програмні продукти створені в наукових цілях
Висновки	Вузькопрофільні рішення, немає можливості збереження результатів, високо затратні.	Висока конкуренція на ринку, легких вхід, є можливість виділятися.		Користувачі прагнуть мати зручний інтерфейс, швидко та стабільну роботу програми.	Впровадження нових методів, можливість користувачу самому коригувати моделі.

Конкурентна ситуація на ринку дає можливість виходу нових продуктів, та вони мають бути не гіршими конкурентів та розширювати функціонал. Щоб бути конкурентоспроможним на ринку продукт має мати такі сильні сторони, як зручний інтерфейс, швидкість обробки даних, низька ціна, висока якість оптимізації. У таблиці 4.10 описано фактори конкурентоспроможності.

Таблиця 4.10 - Фактори конкурентоспроможності

№ п/п	Фактор	Обґрунтування
1	Актуальність даних	При кожному запуску автоматичне оновлення даних (або по запиту користувача)
2	Вартість продукту	Низька ціна збільшує кількість потенційних клієнтів.
3	Кросплатформність	Можливість встановлення продукту на власні сервери (майбутні розробки) та робота з хмарними технологіях.
4	Унікальний графічний інтерфейс	Розробка простого впізнаваного інтерфейсу, графічне представлення даних.
5	Математична база	Використання найпопулярніших методів оптимізацій, додавання нових, можливість коригування методів користувачами.

За визначеними факторами конкурентоспроможності проведено аналіз сильних та слабких сторін проекту (таблиця 4.11).

Таблиця 4.11 - Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	Фактор	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні						
			-3	-2	-1	0	1	2	3
1	Вартість продукту	19						*	
2	Кросплатформність	17					*		
3	Унікальний графічний інтерфейс	9			*				
4	Математична база	13				*			
5	Актуальність даних	15					*		

Фактори ризику та фактори можливості це фактори, що не реалізовані на ринку, та мають деяку ймовірність здійснення. Їх перелік складається шляхом

аналізу факторів можливостей та загроз маркетингового середовища. Сильні та слабкі сторони, а також можливості і загрози, наведені у таблиці 4.12 як результати SWOT-аналізу проекту.

Таблиця 4.12 - SWOT-аналіз проекту

Сильні сторони: - Автоматизація прогнозування - Застосування швидких новітніх технологій - Використання в різних сферах діяльності - Зручний інтерфейс	Слабкі сторони: - Малий бізнес; - Відсутність клієнтів; - Відсутність інвестицій;
Можливості: - Заохочення малого та середнього бізнесу; - Молоді та висококваліфіковані кадри; - Інтегрування процесів прогнозування в інші сфери діяльності - Застосування нових методів	Загрози: - Конкуренція - Висока ціна для локального бізнесу - Зміна потреб користувачів

За результатами SWOT-аналізу можна зробити висновок, що можливостей проекту більше ніж загроз, тому ризики виходу на ринок мінімальні. Проведемо аналіз впровадження стартап-проекту на ринок з точки зору ресурсів та термінів. У таблиці 4.13 наведено альтернативи ринкового впровадження програмного продукту.

Таблиця 4.13 - Альтернативи ринкового впровадження

Альтернатива поведінки	ринкової	Ймовірність отримання ресурсів	Строки реалізації
Безкоштовна публікація продукту, заробіток на рекламі		30%	6 місяців
Продаж підписки на продукт з безкоштовним пробним періодом з обмеженим функціоналом, розширення функціоналу з преміум підпискою		90%	20 місяців
Продаж програми On-premises		40%	24 місяці

Всі три альтернативи є популярними на ринку. Перший варіант підходить для маловідомих стартапів для пошуку цільової аудиторії. Третій варіант є найскладнішим технічно, потрібна підготовка серверів у користувачів. Тому було обрано другу альтернативу ринкового впровадження. Вона є більш швидкою у реалізації та проста в отриманні ресурсів. Також шляхом безкоштовного пробного доступу є можливість залучення потенційних клієнтів.

4.4 Розроблення ринкової стратегії проекту

Міра сформованості ринку інформаційних технологій є високою, тому є можливість обрати одну з трьох стратегій конкурентної боротьби: концентрованого маркетингу, стратегію диференційованого маркетингу, стратегію масового маркетингу. Для виходу продукту на ринок потрібно обрати найбільш доцільну стратегію.

Проведемо опис цільових груп потенційних споживачів (таблиця 4.14).

Таблиця 4.14 - Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Просто та входу у сегмент
1	Фонди	Висока	Високий	Сильна	Просто
2	Банки	Помірна	Високий	Слабка	Складно
3	Фінансові біржі	Висока	Високий	Сильна	Просто
4	Підприємства регіонального характеру	Слабка	Помірний	Слабка	Складно
5	Державні установи,	Помірна	Помірний	Сильна	Складно
Які цільові групи обрано:1,2,3					

Продукт є універсальним, це означає, що його можна застосовувати в різних галузях. Тому в якості цільових груп обрано три групи, які включають в себе Фонди, біржі та банки. Програма буде універсальною для всіх сегментів ринку, тому використовувати будемо стратегію масового маркетингу.

Оберемо базову стратегію розвитку для роботи з обраними сегментами ринку.

Стратегією конкурентної поведінки було обрано стратегію лідера.

У таблиці 4.15 визначені базові стратегії розвитку. У таблиці 4.16 визначені базові стратегії конкурентної поведінки.

Таблиця 4.15 - Базова стратегія розвитку

№ п/п	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Спеціальні пропозиції Фондам, біржам та банкам. Участь в конференціях. Безкоштовна підписка студентам та науковцям.	Можливість купити підписку на обмежений функціонал, який цікавить користувача. Безкоштовних пробний період.	Розвиток рекламної компанії, підтримка програмного продукту, надання безкоштовних підписок навчальним установам.

Таблиця 4.16 - Базова стратегія конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Ні, але має унікальні ідеї які спрощують розуміння	Обидва варіанти	Повторення функціоналу конкурентів та додавання унікального.	Стратегія лідера

Наступним кроком після визначення стратегії конкурентної поведінки є визначення вимог до продукту від цільової аудиторії. На основі вимог визначаємо стратегії позиціонування (таблиця 4.17).

Таблиця 4.17 - Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Надійний продукт, низька ціна, постійне розширення функціоналу, зручний інтерфейс.	Стратегія масового маркетингу	Позиція порівняння власного продукту з продуктами конкурентів, створення унікальних рішень та методів	Точність результатів, швидкість обрахунків, глобальний ринок

В даному підрозділі було порівняно ринкові стратегії конкурентів. Було опрацьовано групи потенційних користувачів та обрано стратегію конкурентної поведінки - стратегію лідера. Також було визначено стратегію розвитку та позиціонування.

4.5 Розроблення маркетингової програми стартап-проекту

Велике значення в аналізі та розвитку стартап-проекту займає визначення ключових переваг концепції потенційного товару. Формування маркетингової

концепції товару впливає з попереднього аналізу конкурентоспроможності продукту.

Концепція товару – опис характеристик товару, які необхідні споживачу та набір переваг перед подібними продуктами. У таблиці 4.18 наведено визначення ключових переваг концепції потенційного продукту.

Таблиця 4.18 - Ключові переваги потенційного продукту

№ п/п	Потреба	Вигода	Ключові переваги перед конкурентами
1	Точність оптимізації	Більш точніші результати, і більше диверсифікації	Всі результати відображаються відразу після зміни методів оптимізації і пошуку портфеля
2	Швидка обробка даних	Економія часу на побудові моделей	Можливість працювати з декількома процесами одночасно в режимі багато поточності
3	Актуальні дані	Використання нових даних з бірж	Коригування прогнозу після оновлення даних, найбільш актуальна інформація
4	Простий інтерфейс	Зручний інтерфейс з графічним відображенням результатів	Робота з програмою без додаткових інструкцій чи базової математичної підготовки

Сучасна маркетингова модель є трирівневою. На першому рівні необхідно уточнити ідею продукту, на другому рівні його фізичні особливості та на третьому рівні – особливості виходу продукту на ринок. У таблиці 4.19 наведено опис маркетингової моделі.

Таблиця 4.19 - Опис маркетингової моделі

Рівень товару	Сутність та складові
I. Товар за задумом	Програмний продукт для оптимізації портфелів
II. Товар у реальному виконанні	Програмний продукт з оновленням актуальних даних, зручним інтерфейсом, новими моделями.
III. Товар із підкріпленням	Програмний продукт з автоматизацією аналізу доходності та ризиків.

При встановленні ціни на продукт стартап-проекту потрібно встановити нижчу ціну, ніж у конкурентів, для того, щоб переманити свою цільову аудиторію та завоювати довіру користувачів. Також потрібно врахувати доходи цільової аудиторії.

Експертним методом було проведено аналіз меж встановлення ціни, результати наведені у таблиці 4.20.

Таблиця 4.20 - Визначення меж встановлення ціни

Рівень цін на товари замітники	Рівень цін на товари аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1000\$ - 1800\$	3000\$-10000\$	Цільова група – великий бізнес та інвестори з високим рівнем доходів	1000\$-3000\$ або підписка 500\$/міс

Після запуску проекту потрібно бути готовими до того, що перший час він не буде приносити прибуток. На входження в ринок та довіру користувачів потрібен час. Також наш продукт матиме безкоштовний пробний період. За цей час необхідно правильно визначити систему збуту продукту.

У таблиці 4.21 наведено формування системи збуту продукту.

Таблиця 4.21 - Система збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Міжнародні державні організації	Прямі постачання товару без посередників, дотримання комерційної таємниці	канал нульового рівня	Участь у конференціях та бізнес тренінгах, розсилка на пошту
2	Великий та середній бізнес	Ведення бухгалтерської звітності по продажу, дотримання комерційної таємниці	канал нульового рівня	Участь у конференціях та бізнес тренінгах, розсилка на пошту

Завершуємо підготовку маркетингової програми розробкою концепції маркетингових комунікацій. При розробці акцентуємо увагу на специфіці поведінки клієнтів та обрану основу для позиціонування. У таблиці 4.22 наведено концепція маркетингових комунікацій.

Таблиця 4.22 - Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Відповідальні особи державних установ та бізнесу відвідують конференції по нових програмах.	Інтернет, семінари, конференції	Порівняння продукту з аналогами	Завоювання довіри, бонуси для інвесторів, великого бізнесу, реферальні бонуси, технічна підтримка	Плануєте повну автоматизацію пошуку інвестиційних портфелів для знаходження ідеальних? Маємо для Вас гарну пропозицію.

В даному підрозділі було розроблено маркетингову стратегію для просування продукту на ринку.

4.6 Висновки до розділу

В даному розділі було розроблено план стартап-проекту на основі дослідження. Було встановлено, що продукт є конкурентоспроможним, адже кількість компаній, які потребують продукт такого типу зростає щороку.

Проаналізувавши аналоги ідеї на ринку можна зробити висновок, що розробка програмного забезпечення за даним проектом є доцільною та економічно вигідною. Фінансово технічний напрямок розвивається з неабиякими темпами, тому для розвитку програмного продукту обрана стратегія лідера. Програмний продукт має включати в себе весь функціонал аналогів, щоб користувачу було зручно переключитися на один додаток який повністю буде його задовольняти.

ВИСНОВКИ

Конкретні фінансові рішення завжди мають бути спрямовані на майбутнє, і ця умова є причиною, що зумовлює те, що багато параметрів невідомі, що мають одне значення, але відомий лише спектр можливостей. Ось чому, коли існує невизначеність або ризик, необхідно приймати рішення, які потребують нових принципів регулювання інформації та методу прийняття рішення.

В дисертаційній роботі розглянуто методи інтелектуального аналізу та оптимізації даних, критерії якості портфелів та прогнозів. Проведений огляд методів інтелектуального аналізу даних: Метод усадки Ledoit Wolf, середня історична доходність, експоненційно зважена середня історична доходність, Оцінка доходності CAPM, Модель Black-Litterman (BL), регуляризацією L2, Max Sharpe з обмеженнями по сектору.

Новизною дисертаційної роботи є побудова множини моделей та методів оптимізації для вибраних тікетів з використанням результатів класичної теорії Марковіца, регуляризацією L2, моделі Black-Litterman, та Max Sharpe з обмеженнями по сектору.

Спроектована і реалізована програмно оригінальна СППР для моделювання і прогнозування ваг інвестиційних портфелів. Проект ґрунтується на принципах системного аналізу: ієрархічність, оптимальність, адаптивність, функціональна повнота (тобто працює автономно без інших систем).

В умовах ринку багато завдань з оптимізації вирішуються не тільки для того, щоб досягти граничних оптимізуючих розмірів - прибутку, частини ринкових та подібних цінностей, гарантуючи ту чи іншу вартість.

Проаналізовано принципи та досягнуто методу формування та розрахунку інвестиційного портфеля, адекватного стохастичному характеру інвестиційного доходу.

Враховуючи постійно зростаючий вплив ризику та невизначеності на результати підприємницької чи індивідуальної діяльності, а також збільшення кількості інструментів управління ризиками, дуже важливо правильно організувати бухгалтерський облік та інше інформаційне регулювання, що дозволяє приймати раціональні рішення щодо ризиків.

Розвиваючи розуміння технічного аналізу, трейдери та інвестори можуть покращити свої довгострокові прибутки з поправкою на ризик, але важливо розуміти та практикувати ці методи, перш ніж вкладати реальний капітал, щоб уникнути дорогих помилок.

Існує велика кількість досліджень, які свідчать про те, що портфелі з мінімальною дисперсією постійно перевершують портфелі з максимальним коефіцієнтом Шарпа поза вибіркою (навіть якщо вони виміряні коефіцієнтом Шарпа), через складність прогнозування очікуваної прибутковості.

Обов'язково пробуємо різні моделі ризику: відомо, що моделі усадки мають кращі числові властивості порівняно з коваріаційною матрицею вибірки.

Налаштовуємо параметр регуляризації L_2 , щоб побачити, як диверсифікація впливає на продуктивність.

Всі ці рекомендації в майбутньому дозволять грамотно управляти запропонованим портфелем фінансових інвестицій і збільшити його прибутковість.

Знайдена структура оптимального портфеля в подальшому може використовуватися при укрупненні або подальшій диверсифікації портфеля цінних паперів, при включенні в нього нових фінансових інструментів (акцій).

В майбутньому рекомендується включити облігації надійних емітентів і єврооблігації, які характеризуються вагомими поточними виплатами купонного доходу або включити в портфель опціони.

ПЕРЕЛІК ПОСИЛАНЬ

1. D. Kritzman, Page & Turkington. In defense of optimization: The fallacy of 1/N. Financial Analysts Journal, 2010. 31-39с. Режим доступу: <https://www.tandfonline.com/doi/abs/10.2469/faj.v66.n2.6> [Дата звернення: 07 10 2021].
2. Y. Chen. Shrinkage Algorithms for MMSE Covariance Estimation, IEEE Transactions on Signals Processing, 2010. 5016-5029с. Режим доступу: <https://webee.technion.ac.il/Sites/People/YoninaEldar/104.pdf> [Дата звернення: 10 09 2021].
3. Аскинадзи В. М. Инвестиции. Москва: Юрайт, 2017. 422с.
4. Вітлінський В.В.. Аналіз, оцінка і моделювання економічного ризику. Київ: ДЕМІУР, 1996. 212 с.
5. T. Idzorek. A step-by-step guide to the Black-Litterman model: Incorporating user-specified confidence levels. In: Forecasting Expected Returns in the Financial Markets. Elsevier Ltd, 2007. 17–38с. Режим доступу: https://faculty.fuqua.duke.edu/~charvey/Teaching/BA453_2006/Idzorek_onBL.pdf [Дата звернення: 15 15 2021].
6. Е.Ф. Жуков. Ценные бумаги и фондовые рынки: Учебное пособие для вузов. Москва: Банки и биржи. ЮНИТИ, 1995. 224 с.
7. Д. В. Кузнецов. Инвестиционный менеджмент. Москва: Юрайт, 2018. 276с.
8. И. А. Бланк. Инвестиционный менеджмент. Киев :МП «ИТЕМ» ЛТД «Юнайгед Лондон Трейд Лимигед», 1995. 448 с.
9. Л. М. Борщ. Інвестиції в Україні: стан. Проблеми і перспективи. - 2-ге вид., стер. Киев: Т-во «Знання», КОО, 2003. 318 с.
10. И. Я. Лукасевич. Инвестиции. - Москва: Вузовский учебник: ИНФРА-М, 2018, 413 с. Режим доступу: <http://znanium.com/catalog/product/958774>. [Дата звернення: 25 11 2021].

11. М. В. Скрипниченко. Портфельные инвестиции: Учебное пособие. - Санкт-Петербург: Университет ИТМО, 2016. 40 с. Режим доступа: <http://books.ifmo.ru/file/pdf/1914.pdf>. [Дата звернення: 17 11 2021].
12. P. D. Kaplan. "What's Wrong with Multiplying by the Square Root of Twelve". Режим доступа: (www.morningstar.com). [Дата звернення: 19 10 2021].
13. Б.И. Алехин. Рынок ценных бумаг. Введение в фондовые операции. Москва: Финансы и статистика, 199. 160с.
14. А.Н. Буренин. Рынок ценных бумаг и производных финансовых инструментов: Учебное пособие. Москва: 1 Федеративная Книготорговая Компания, 1998. 352 с.
15. Ю.Ф. Касимов. Основы теории оптимального портфеля ценных бумаг. Москва: Информационноиздательский дом «Филинь», 1998. 144 с.
16. Б Л. Луців. Банківська дальність у сфері інвестицій. Тернопіль: економічна думка, Карт-бланш, 2001. 320 с.
17. Я.М. Миркин. Ценные бумаги и фондовый рынок. Москва: Перспектива, 1995. 550 с.
18. В.В. Оскольський. Ринок цінних паперів в Україні: погляд через призму діяльності Української фондової біржі. Київ: УФБ, 2001. 216 с.
19. Л.О. Примостка. Банківський менеджмент. Хеджування фінансових ризиків. Київ : КНЕУ, 1998, 108 с.
20. Про цінні папери та фондовий ринок. Закону України від 23.02.2006р. № 3480-IV Режим доступа: <https://zakon.rada.gov.ua/laws/show/3480-15#Text> [Дата звернення: 18 08 2021].
21. В.И.Колесникова . Ценные бумаги: Учебник / под ред. В.С.Торкановского. Москва: Финансы и статистика, 1999. 416 с.
22. У. Шарп, Г.Александр, Дж. Бэйли. Инвестиции: Пер. с англ. Москва: ИНФРА-М, 2001. 1028 с.
23. С.Айвазян, В.С. Мхитарян. Прикладная статистика и основы эконометрики. Москва: Юнити, 1998. 1000 с.

24. Г. Бенджамін, Дж. Цвейг. Розумний інвестор. Стратегія вартісного інвестування. Київ: Наш формат, 2019, 544 с.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Файл 1_Моделі_Повернення_Ризику.ipynb

```
### md
```

1_Models_Risk_Return

У цьому розділі ми порівнюємо, наскільки добре різні моделі ризику прогнозують коваріаційну матрицю поза вибіркою (out-of-sample covariance matrix), і наскільки добре різні моделі прибутку прогнозують прибутки поза вибіркою (out-of-sample returns).

```
###
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import portfOpt
from portfOpt import riskModels, expectedReturns, plot
portfOpt.__version__
```

```
###
```

```
df = pd.read_csv("data/stock_prices.csv", parse_dates=True, index_col="date")
pastDf, futureDf = df.iloc[:-250], df.iloc[-250:]
sampleCov = riskModels.sampleCov(pastDf)
futureCov = riskModels.sampleCov(futureDf)
```

```
plot.plotCovariance(sampleCov, plotCorrelation=True)
plot.plotCovariance(futureCov, plotCorrelation=True)
plt.show()
```

```
###
```

```
sampleCov
futureCov
```

```
### md
```

Ми бачимо, що візуально вибіркова коваріація не фіксує деякі нові особливості коваріаційної матриці, наприклад, високо корельовану групу акцій FAANG (Meta (FB), Amazon (AMZN), Apple (AAPL), Netflix (NFLX), and Alphabet (GOOG)). Можливо, ми зможемо покращити це, використовуючи експоненціально зважену коваріаційну матрицю, яка надає більшій ваги останнім даним. Ми також можемо подивитися, як кожна модель прогнозує майбутню дисперсію.

```
###
```

```
futureVariance = np.diag(futureCov)
meanAbsErrors = []
```

```

riskMethods = [
    "sampleCov",
    "expCov",
    "ledoitWolf",
]

for method in riskMethods:
    S = riskModels.riskMatrix(df, method=method)
    variance = np.diag(S)
    meanAbsErrors.append(np.sum(np.abs(variance - futureVariance)) / len(variance))

xrange = range(len(meanAbsErrors))
plt.barh(xrange, meanAbsErrors)
plt.yticks(xrange, riskMethods)
plt.show()

```

md

Ми бачимо, що матриця експоненційної коваріації є набагато кращою оцінкою майбутньої дисперсії порівняно з іншими моделями.

Його середня абсолютна похибка становить 2%, що насправді дуже добре.

Давайте візуально порівняємо експоненційну матрицю cov з реалізованою майбутньою матрицею cov:

###

```

expCov = riskModels.expCov(pastDf)
plot.plotCovariance(expCov, plotCorrelation=True)
plot.plotCovariance(futureCov, plotCorrelation=True)
plt.show()

```

md

Returns

А як щодо доходності? Будемо вибирати між "Оцінка середньої прибутковості" та "Оцінка прибутку CAPM".

###

```

futureRets = expectedReturns.meanHistoricalReturn(futureDf)
meanAbsErrors = []
returnMethods = [
    "meanHistoricalReturn",
    "emaHistoricalReturn",
    "capmReturn",
]

for method in returnMethods:
    mu = expectedReturns.returnModel(pastDf, method=method)
    meanAbsErrors.append(np.sum(np.abs(mu - futureRets)) / len(mu))

```

```

xrange = range(len(meanAbsErrors))
plt.barh(xrange, meanAbsErrors)
plt.yticks(xrange, returnMethods)
plt.show()

```

###

```
print(meanAbsErrors)
```

md

Оцінка прибутку Exp трохи краще, ніж інші але покращення майже непомітне. Також бачимо, що середні абсолютні відхилення перевищують 25%, що означає, що якщо наші очікувані річні прибутки становлять 10%, в середньому реалізована річна прибутковість може становити від 15% втрати до 35% прибутку. Це величезний діапазон, який бажано зоптимізувати.

```
###
```

```
fig, axs = plt.subplots( 1, len(returnMethods),sharey=True, figsize=(15,10))
```

```
for i, method in enumerate(returnMethods):
    mu = expectedReturns.returnModel(pastDf, method=method)
    axs[i].set_title(method)
    mu.plot.barh(ax=axs[i])
```

```
### md
```

Хороша новина полягає в тому, що ми бачимо гарну ступінь згоди, окрім exp

```
###
```

Файл: 2_Оптимізація_Середньої_Дисперсії.ipynb

```
### md
```

Mean-variance optimization

У цій книзі ми працюємо над кількома прикладами, що демонструють можливості щодо середньої дисперсії.

Я обговорю, які, на мою думку, мають бути вашими параметрами «за замовчуванням», ґрунтуючись на досвіді оптимізації портфелів.

Для початку нам потрібен список тикерів. Деякі люди просто надають весь світ акцій, але я не думаю, що це гарна ідея – оптимізація портфеля значно відрізняється від вибору активів. Я б запропонував від 10 до 50 акцій як відправну точку.

Деякі з речей, які ми охоплюємо:

Завантаження даних

Розрахунок та візуалізація коваріаційної матриці

Оптимізація long/short портфеля для мінімізації загальної дисперсії

Оптимізація портфеля для максимізації коефіцієнта Шарпа, з урахуванням галузевих обмежень

Оптимізація портфеля для максимізації прибутку для певного ризику, з урахуванням обмежень сектора, з метою регуляризації L2

Оптимізація ринково нейтрального портфеля для мінімізації ризику для заданого рівня прибутку

Оптимізація вздовж межі середньої напівваріації (mean-semiVariance frontier)

Оптимізація вздовж межі середнього CVaR (mean-CVaR frontier)

Побудова ефективної межі:

Простий (з використанням CLA)

Стриманий

Складні сюжети

Downloading data

Для завантаження даних ми будемо використовувати `yfinance`, чудову бібліотеку, яка

надає безкоштовні дані про ціни від Yahoo Finance, без ключа API.

```

#%%

import cvxopt
import cvxpy
import yfinance as yf
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

#%%

tickers = ["MSFT", "AMZN", "KO", "MA", "COST",
           "LUV", "XOM", "PFE", "JPM", "UNH",
           "ACN", "DIS", "GILD", "F", "TSLA"]

#%%

ohlcv = yf.download(tickers, period="max")

#%%

prices = ohlcv["Adj Close"].dropna(how="all")
prices.tail()

#%%

prices[prices.index >= "2008-01-01"].plot(figsize=(15,10));

#%% md

## Calculating the covariance matrix

#%%

import portfOpt
portfOpt.__version__

#%%

from portfOpt import riskModels
from portfOpt import plot

sampleCov = riskModels.sampleCov(prices, frequency=252)
sampleCov

#%%

plot.plotCovariance(sampleCov, plotCorrelation=True);

#%% md

З причин, які обговорюються в дипломі, вибіркова коваріаційна матриця не повинна бути
нашим вибором за замовчуванням.
Я думаю, що кращим варіантом є усадка Ledoit-Wolf, яка зменшує екстремальні значення в
матриці коваріації.
На зображенні нижче ми бачимо, що за межами діагоналі менше яскравих плям:

#%%

S = riskModels.CovShrinkage(prices).ledoitWolf()

```

```
plot.plotCovariance(S, plotCorrelation=True);
```

```
### md
```

Return estimation

Як написано в дипломі, часто буває поганою ідеєю надавати прибутки, використовуючи просту оцінку, як-от середнє минулого прибутку.
Якщо у нас немає власного методу оцінки прибутку, дослідження показують, що нам краще не надавати очікувану прибутковість
– можемо просто знайти портфель `minVolatility()` або скористатися `HRP`.

Однак у цьому прикладі ми будемо використовувати показники CAPM, які мають бути трохи більш стабільними,
ніж середня історична прибутковість за замовчуванням.
В першому блокноті `1-RiskReturnModels.ipynb` додаткова інформація.

```
###
```

```
from portfOpt import expectedReturns
```

```
mu = expectedReturns.capmReturn(prices)
mu
```

```
###
```

```
mu.plot.barh(figsize=(10,6));
```

```
### md
```

Long/short min variance

У цьому розділі ми створюємо long/short портфель з метою мінімізації дисперсії.
Існує чимало досліджень, які демонструють, що ці портфелі з глобальною мінімальною дисперсією (GMV) перевершують портфелі, оптимізовані за середньою дисперсією.

```
###
```

```
from portfOpt import EfficientFrontier
```

```
###
```

```
S = riskModels.CovShrinkage(prices).ledoitWolf()
```

```
# У цьому випадку нам не потрібно надавати очікувані доходи
ef = EfficientFrontier(None, S, weightBounds=(None, None))
ef.minVolatility()
weights = ef.cleanWeights()
weights
```

```
###
```

```
pd.Series(weights).plot.barh();
```

```
### md
```

Ми можемо отримати швидку індикацію ефективності портфеля таким чином.
Зауважу, що це приблизна оцінка і може мати дуже мало схожості з фактичною ефективністю портфоліо!

```
###
```

```
ef.portfolioPerformance(verbose=True);
```

```
###
```

```
prices
```

```
### md
```

Max Sharpe with sector constraints

Ми можете розглянути можливість максимізації коефіцієнта Шарпа.
Це теоретично дає оптимальний портфель з точки зору ризиків-прибутків.

У цьому розділі ми створюємо довготривалий портфель max-sharp, але також включаємо секторні обмеження. Секторні обмеження вимагають трьох речей: ``sectorMapper``, ваші межі ``sectorLower`` і ваші межі ``sectorUpper``

```
###
```

```
sectorMapper = {
  "MSFT": "Tech",
  "AMZN": "Consumer Discretionary",
  "KO": "Consumer Staples",
  "MA": "Financial Services",
  "COST": "Consumer Staples",
  "LUV": "Aerospace",
  "XOM": "Energy",
  "PFE": "Healthcare",
  "JPM": "Financial Services",
  "UNH": "Healthcare",
  "ACN": "Misc",
  "DIS": "Media",
  "GILD": "Healthcare",
  "F": "Auto",
  "TSLA": "Auto"
}

sector_lower = {
  "Consumer Staples": 0.1, # принаймні 10% до основних продуктів
  "Tech": 0.05 # принаймні 5% до техніки
  # Для всіх інших секторів буде вважатися, що нижньої межі немає
}

sectorUpper = {
  "Tech": 0.2,
  "Aerospace": 0.1,
  "Energy": 0.1,
  "Auto": 0.15
}
```

```
### md
```

Потім ми налаштовуємо оптимізатор і додаємо наші обмеження. Ми можемо використовувати ``ef.addObjective()``, щоб додати інші обмеження. Наприклад, скажімо, що на додаток до вищезгаданих обмежень сектору, я спеціально хочу:

- 10% портфеля в AMZN
- Менше 5% мого портфеля в TSLA

```
###
```

```

mu = expectedReturns.capmReturn(prices)
S = riskModels.CovShrinkage(prices).ledoitWolf()

ef = EfficientFrontier(mu, S) # weightBounds automatically set to (0, 1)
ef.addSectorConstraints(sectorMapper, sector_lower, sectorUpper)

amzn_index = ef.tickers.index("AMZN")
ef.addConstraint(lambda w: w[amzn_index] == 0.10)

tsla_index = ef.tickers.index("TSLA")
ef.addConstraint(lambda w: w[tsla_index] <= 0.05)

ef.addConstraint(lambda w: w[10] >= 0.05)

ef.maxSharpe()
weights = ef.cleanWeights()

###

weights

###

pd.Series(weights).plot.pie(figsize=(10,10));

### md

Ми можемо відразу побачити, що наші явні обмеження були задоволені,
і можемо перевірити всі обмеження сектора наступним чином:

###

# Цикл  $O(N^2)$  не є гарною ідеєю на співбесіді з кодуванням :)
for sector in set(sectorMapper.values()):
    total_weight = 0
    for t,w in weights.items():
        if sectorMapper[t] == sector:
            total_weight += w
    print(f"{sector}: {total_weight:.3f}")

### md

## Максимізація прибутку для даного ризику з регуляризацією L2
Давайте уявимо, що ми багато подумали про свою толерантність до ризику і вирішили,
що не можемо прийняти нічого більше, ніж 15% волатильності.
Ми можемо використати це для створення портфеля, який максимізує прибуток для даного
ризикy
(з тими ж застереженнями щодо очікуваної прибутковості).

###

ef = EfficientFrontier(mu, S)
ef.addSectorConstraints(sectorMapper, sector_lower, sectorUpper)
ef.efficientRisk(targetVolatility=0.15)
weights = ef.cleanWeights()
weights

###

num_small = len([k for k in weights if weights[k] <= 1e-4])
print(f"{num_small}/{len(ef.tickers)} tickers have zero weight")

```

```
###
```

```
ef.portfolioPerformance(verbose=True);
```

```
### md
```

Хоча цей портфель здається, що він відповідає нашим цілям, нас може турбувати той факт, що багатьом тикерам присвоєно нульову вагу. По суті, оптимізатор "переповнює" надані нами дані - ви, швидше за все, отримаєте кращі результати, застосувавши певний рівень диверсифікації. Один із способів зробити це – використовувати ****регуляризацію L2**** – по суті, додаючи штраф на кількість майже нульових ваг.

```
###
```

```
from portfOpt import objectiveFunctions
```

```
# Ми завжди повинні створювати новий ефективний об'єкт кордону
ef = EfficientFrontier(mu, S)
ef.addSectorConstraints(sectorMapper, sector_lower, sectorUpper)
ef.addObjective(objectiveFunctions.L2Reg, gamma=0.1)
# gamma є параметром налаштування
ef.efficientRisk(0.15)
weights = ef.cleanWeights()
weights
```

```
###
```

```
num_small = len([k for k in weights if weights[k] <= 1e-4])
print(f"{num_small}/{len(ef.tickers)} tickers have zero weight")
```

```
### md
```

Ми можемо налаштувати значення гами, щоб вибрати кількість ненульових тікерів. Більша гамма тягне ваги портфеля до рівного розподілу.

```
###
```

```
ef = EfficientFrontier(mu, S)
ef.addSectorConstraints(sectorMapper, sector_lower, sectorUpper)
ef.addObjective(objectiveFunctions.L2Reg, gamma=1)
# gamma є параметром налаштування
ef.efficientRisk(0.15)
weights = ef.cleanWeights()
weights
```

```
###
```

```
pd.Series(weights).plot.pie(figsize=(10, 10));
```

```
###
```

```
ef.portfolioPerformance(verbose=True);
```

```
### md
```

Отриманий портфель все ще має волатильність менше, ніж наш ліміт у 15%. Коефіцієнт Шарпа у вибірці знизився, але цей портфель є набагато надійнішим для реальних інвестицій.

```
### md
```

Мінімізація ризику для даної прибутковості, ринково нейтральний(*market- neutral*)

Натомість ми можемо опинитися в ситуації, коли маємо певну необхідну норму прибутку (можливо, ми є пенсійним фондом, який потребує 7% прибутку на рік), але ми хотіли б мінімізувати ризик.

Крім того, припустимо, що ми хотіли б, щоб наш портфель був нейтральним до ринку в тому сенсі,

що він однаково схильний до long або short сторони.

```
###
```

```
# Не повинно бути обмежень ваги, щоб дозволити шорт
```

```
ef = EfficientFrontier(mu, S, weightBounds=(None, None))
```

```
ef.addObjective(objectiveFunctions.L2Reg)
```

```
ef.efficientReturn(targetReturn=0.07, marketNeutral=True)
```

```
weights = ef.cleanWeights()
```

```
weights
```

```
###
```

```
ef.portfolioPerformance(verbose=True);
```

```
###
```

```
pd.Series(weights).plot.barh(figsize=(10,6));
```

```
###
```

```
print(f"Net weight: {sum(weights.values()):.2f}")
```

```
### md
```

Ефективна оптимізація напівдисперсій (*semi-variance*)

У цьому прикладі ми зведемо до мінімуму напівваріантність портфеля (тобто волатильність знизу)

за умови обмеження прибутковості (цільове значення 20%).

Насправді є два способи зробити це. Перший – це «інтуїтивно зрозумілий».

Ми обчислюємо напівковаріаційну матрицю і передаємо її в `EfficientFrontier`

(так само, як ми зробили б для експоненціальної матриці cov або зменшеної матриці Ледуа-Вольфа (Ledoit-Wolf shrunk matrix)).

```
###
```

```
semicov = riskModels.semiCovariance(prices, benchmark=0)
```

```
plot.plotCovariance(semicov);
```

```
###
```

```
ef = EfficientFrontier(mu, semicov)
```

```
ef.efficientReturn(0.2)
```

```
weights = ef.cleanWeights()
```

```
weights
```

```
###
```

```
ef.portfolioPerformance(verbose=True);
```

```
### md
```

Однак це рішення не є по-справжньому оптимальним у просторі середніх напівваріацій. Щоб зробити оптимізацію належним чином, ми повинні використовувати клас `EfficientSemivariance`. Для цього потрібно спочатку обчислити прибуток і відкинути NaN.

```
###
```

```
returns = expectedReturns.returnsFromPrices(prices)
returns = returns.dropna()
```

```
###
```

```
from portfOpt import EfficientSemivariance
```

```
es = EfficientSemivariance(mu, returns)
es.efficientReturn(0.2)
es.portfolioPerformance(verbose=True);
```

```
### md
```

Щоб порівняти це з евристичним рішенням, я скористаюся швидким хакем: заміню `es.weights` на `ef.weights` і знову запускаю `es.portfolioPerformance`. В реальному житті так краще не робити.

```
###
```

```
es.weights = ef.weights
es.portfolioPerformance(verbose=True);
```

```
### md
```

Ми бачимо, що евристичний метод має значно нижчий коефіцієнт Шарпа і набагато вищу напівваріантність.

```
### md
```

Ефективна оптимізація CVaR

У цьому прикладі ми знайдемо портфель, який максимізує прибуток за умови обмеження CVaR.

Перш ніж це зробити, давайте спочатку обчислимо 95%-CVaR для портфеля max-sharp.

```
###
```

```
returns = expectedReturns.returnsFromPrices(prices).dropna()
returns.head()
```

```
###
```

```
ef = EfficientFrontier(mu, S)
ef.maxSharpe()
weightArr = ef.weights
ef.portfolioPerformance(verbose=True);
```

```
###
```

```
# Обчислення CVaR
```

```
portfolioRets = (returns * weightArr).sum(axis=1)
portfolioRets.hist(bins=50);
```

```
###
```

```
# VaR
```

```
var = portfolioRets.quantile(0.05)
cvar = portfolioRets[portfolioRets <= var].mean()
print("VaR: {:.2f}%".format(100*var))
print("CVaR: {:.2f}%".format(100*cvar))
```

```
### md
```

Це значення CVaR означає, що наші середні втрати в найгірші 5% днів становитимуть - 3,35%.

Скажімо, це було за межами нашої зони комфорту (для портфеля \$100 000 це означало б втрати \$3350 за день).

Давайте спочатку побудуємо портфель з мінімальним CVaR:

```
###
```

```
from portfOpt import EfficientCVar
```

```
ec = EfficientCVar(mu, returns)
ec.minCvar()
ec.portfolioPerformance(verbose=True);
```

```
### md
```

Ми значно знизили CVaR, але ціною значного зниження прибутку.

Ми можемо використовувати `efficientRisk`, щоб максимізувати прибуток для цільового ризику.

Скажімо, 2,5% CVaR є прийнятним.

```
###
```

```
from portfOpt import EfficientCVar
```

```
ec = EfficientCVar(mu, returns)
ec.efficientRisk(targetCvar=0.025)
ec.portfolioPerformance(verbose=True);
```

```
### md
```

Тепер ми маємо вище прибутковість (25,1% проти 18,2%), але з трохи вищим ризиком хвоста (2,50% CVaR проти 1,82%).

```
### md
```

Побудова графіка - без обмежень (plot - Unconstrained)

Щоб побудувати безобмежений ефективний кордон, найпростіше використовувати алгоритм критичної лінії (critical line algorithm)

```
###
```

```
from portfOpt import CriticalLineAlgorithm
from portfOpt import plot
cla = CriticalLineAlgorithm(mu, S)
cla.maxSharpe()
cla.portfolioPerformance(verbose=True);
```

```
###
```

```
ax = plot.plotEfficientFrontier(cfa, showfig=False)
```

```
### md
```

Побудова графіка - з обмеженнями (plot - Constrained)

У цьому прикладі ми намалюємо ефективний кордон, що відповідає портфелям з обмеженням впливу MSFT, AMZN і TSLA (наприклад, можливо, ми хочемо уникнути великих технологій).

```
###
```

```
import cvxpy as cp
```

```
mu = expectedReturns.capmReturn(prices)
```

```
S = riskModels.CovShrinkage(prices).ledoitWolf()
```

```
ef = EfficientFrontier(mu, S,)
```

```
bigTechIndices = [t in {"MSFT", "AMZN", "TSLA"} for t in tickers]
```

```
ef.addConstraint(lambda w: cp.sum(w[bigTechIndices]) <= 0.3)
```

```
### md
```

Прежде чем вызывать любую функцию оптимизации, мы должны передать это модулю построения графика:

```
###
```

```
ax = plot.plotEfficientFrontier(ef, ef_param="risk",
                                ef_param_range=np.linspace(0.12, 0.4, 50),
                                showfig=False);
```

```
### md
```

Complex plots

Модуль `plot` позволяет вам передать `ax`, поверх которого добавляются графики. Это позволяет строить сложные сюжеты.

В этом примере мы построим эффективную границу, а также 10 000 смоделированных портфелей.

Чтобы сгенерировать смоделированные портфели, мы будем выбирать случайные веса из распределения Дирихле (они уже нормализованы):

```
###
```

```
nSamples = 10000
```

```
w = np.random.dirichlet(np.ones(len(mu)), nSamples)
```

```
rets = w.dot(mu)
```

```
stds = np.sqrt((w.T * (S @ w.T)).sum(axis=0))
```

```
sharpes = rets / stds
```

```
print("Sample portfolio returns:", rets)
```

```
print("Sample portfolio volatilities:", stds)
```

```
### md
```

Обратите внимание, что приведенный выше код эквивалентен генерации образцов с помощью

цикла for (только более эффективно).

```
###

# Постройка эффективной границы с помощью Monte Carlo sim
ef = EfficientFrontier(mu, S)

fig, ax = plt.subplots()
plot.plotEfficientFrontier(ef, ax=ax, show_assets=False)

# Нахождение и построение портфолио касания
ef.maxSharpe()
retTangent, stdTangent, _ = ef.portfolioPerformance()
ax.scatter(stdTangent, retTangent, marker="*", s=100, c="r", label="Max Sharpe")

# Построение случайных портфелей
ax.scatter(stds, rets, marker=".", c=sharpes, cmap="viridis_r")

# Формат
ax.set_title("Efficient Frontier with random portfolios")
ax.legend()
plt.tight_layout()
plt.show()

###
```

Файл 3_Black_Litterman.ipynb

md

Распределение Black-Litterman

Метод Блэка-Литтермана - очень эффективный способ конвертировать наши взгляды на доходность активов, а также нашу неуверенность в этих взглядах в портфель.

Описание теории привел в дипломе

В этой книге мы рассмотрим:

- Загрузка данных по методу Блэка-Литтермана.
- Построение вектора априорной доходности на основе рыночного равновесия
- Два способа построения матрицы неопределенности
- Комбинирование Блэка-Литтермана с оптимизацией средней дисперсии

Загрузка данных

В дополнение к ценовым данным для построения рыночного априора требуется рыночная капитализация.

```
###

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
```

```

###

tickers = ["MSFT", "AMZN", "NAT", "BAC", "DPZ", "DIS", "KO", "MCD", "COST", "SBUX"]

###

ohlcv = yf.download(tickers, period="max")
#ohlcv = yf.download(tickers, start="2015-02-02", end="2021-12-01")
prices = ohlcv["Adj Close"]
prices.tail()

###

marketPrices = yf.download("SPY", period="max")["Adj Close"]
marketPrices.head()

###

mcaps = {}
for t in tickers:
    stock = yf.Ticker(t)
    mcaps[t] = stock.info["marketCap"]
mcaps

### md

```

Построение предыдущего

```

###

import portfOpt
portfOpt.__version__

###

from portfOpt import blackLitterman, riskModels
from portfOpt import BlackLittermanModel, plot

S = riskModels.CovShrinkage(prices).ledoitWolf()
delta = blackLitterman.marketImpliedRiskAversion(marketPrices)
delta

###

plot.plotCovariance(S, plotCorrelation=True);

###

marketPrior = blackLitterman.marketImpliedPriorReturns(mcaps, delta, S)
marketPrior

###

marketPrior.plot.barh(figsize=(10,5));

### md

```

Просмотр

В методе BL представления задаются через матрицу P (матрицу выбора) и вектор Q. Q содержит величину каждого представления, в то время как P сопоставляет представления с активами, которым они принадлежат.

Если мы предоставляем ***absolute views*** (то есть оценку возврата для каждого актива), то нам не нужно беспокоиться о P и Q, мы можем просто передать свои просмотры в виде словаря.

```
###
```

```
# Нам не обязательно предоставлять просмотры всех активов
```

```
viewdict = {
    "AMZN": 0.10,
    "BAC": 0.30,
    "COST": 0.05,
    "DIS": 0.05,
    "DPZ": 0.20,
    "KO": -0.05, # Я думаю, Coca-Cola упадет на 5%
    "MCD": 0.15,
    "MSFT": 0.10,
    "NAT": 0.50, # но низкая достоверность, которая будет отражена позже
    "SBUX": 0.10
}
```

```
bl = BlackLittermanModel(S, pi=marketPrior, absoluteViews=viewdict)
```

```
### md
```

Black-Litterman также допускает относительные просмотры, например, мы думаем, что актив A превзойдет актив B на 10%.

Если мы хотим включить их, то нам придется построить P и Q самостоятельно.

Объяснение этого так же дано в дипломе

```
### md
```

Просмотр достоверности

В этом разделе мы предлагаем два способа построения матрицы неопределенности.

Первый известен как метод Идзорека.

Это позволяет нам указать вектор / список процентной достоверности.

```
###
```

```
confidences = [
    0.6,
    0.4,
    0.2,
    0.5,
    0.7, # уверен в домино
    0.7,
    0.7,
    0.5,
    0.1,
    0.4
]
```

```
###
```

```
bl = BlackLittermanModel(S, pi=marketPrior, absoluteViews=viewdict, omega="idzorek",
viewConfidences=confidences)
```

```
###
```

```
fig, ax = plt.subplots(figsize=(7,7))
im = ax.imshow(bl.omega)
```

```
# Мы хотим показать все тикеты
ax.set_xticks(np.arange(len(bl.tickers)))
ax.set_yticks(np.arange(len(bl.tickers)))
```

```
ax.set_xticklabels(bl.tickers)
ax.set_yticklabels(bl.tickers)
plt.show()
```

```
###
```

```
np.diag(bl.omega)
```

```
### md
```

Обратите внимание, как NAT, которому мы дали наименьшее доверие, также имеет наивысшую неопределенность.

Вместо того, чтобы вводить достоверность, мы можем рассчитать матрицу неопределенности напрямую,

указав 1 доверительный интервал стандартного отклонения, то есть границы, которые, по нашему мнению,

будут содержать истинную доходность в 68% случаев.

Это может быть проще, чем придумывать несколько произвольный процент уверенности.

```
###
```

```
intervals = [
    (0, 0.25),
    (0.1, 0.4),
    (-0.1, 0.15),
    (-0.05, 0.1),
    (0.15, 0.25),
    (-0.1, 0),
    (0.1, 0.2),
    (0.08, 0.12),
    (0.1, 0.9),
    (0, 0.3)
]
```

```
###
```

```
variances = []
for lb, ub in intervals:
    sigma = (ub - lb)/2
    variances.append(sigma ** 2)
```

```
print(variances)
omega = np.diag(variances)
```

```
### md
```

Предварительные оценки

Учитывая исходные данные, мы можем вычислить апостериорную оценку доходности (posterior estimate of returns)

```
###
```

```
# Мы используем ярлык для автоматического вычисления предполагаемого рынка
bl = BlackLittermanModel(S, pi="market", marketCaps=mcaps, riskAversion=delta,
    absoluteViews=viewdict, omega=omega)
```

```
###
```

```
# Предварительная оценка доходности
```

```
retBl = bl.blReturns()
```

```
retBl
```

```
### md
```

Мы можем визуализировать, как это соотносится с предыдущими и нашими взглядами:

```
###
```

```
retsDf = pd.DataFrame([marketPrior, retBl, pd.Series(viewdict)],  
                      index=["Prior", "Posterior", "Views"]).T
```

```
retsDf
```

```
###
```

```
retsDf.plot.bar(figsize=(12,8));
```

```
### md
```

Обратите внимание, что posterior часто находится между предыдущим (prior) и просмотром (views).

Это подтверждает тот факт, что метод BL по сути представляет собой байесовское средневзвешенное значение априорных и представлений, где вес определяется достоверностью.

Аналогичную, но менее интуитивную процедуру можно использовать для получения оценки апостериорной ковариации:

```
###
```

```
SB1 = bl.blCov()
```

```
plot.plotCovariance(SB1);
```

```
### md
```

Распределение портфеля

Теперь, когда мы построили апостериорную оценку Блэка-Литтермана, мы можем перейти к использованию любого из оптимизаторов, описанных в предыдущих книгах.

```
###
```

```
from portfOpt import EfficientFrontier, objectiveFunctions
```

```
###
```

```
ef = EfficientFrontier(retBl, SB1)  
ef.addObjective(objectiveFunctions.L2Reg)  
ef.maxSharpe()  
weights = ef.cleanWeights()  
weights
```

```
###
```

```
pd.Series(weights).plot.pie(figsize=(10,10));
```

```
###
```

```

from portfOpt import DiscreteAllocation

da = DiscreteAllocation(weights, prices.iloc[-1], totalPortfolioValue=20000)
alloc, leftover = da.lpPortfolio()
print(f"Остатки: ${leftover:.2f}")
alloc

###

```

Файл: 4_Ієрархічний_Паритет_Ризиків.ipynb

md

Hierarchical Risk Parity (Иєрархический паритет рисков)

HRP - это современный метод оптимизации портфеля, вдохновленный машинным обучением.

Идея состоит в том, что, изучая иєрархическую структуру рынка, мы можем лучше диверсифицировать.

В этой книге мы рассмотрим:

- Загрузка данных для HRP
- Использование HRP для поиска портфеля с минимальной дисперсией
- Построение дендрограмм

md

Загрузка данных

HRP требует только исторической доходности

###

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
import portfOpt

portfOpt.__version__

###

tickers = ["BLK", "BAC", "AAPL", "TM", "WMT",
           "JD", "INTU", "MA", "UL", "CVS",
           "DIS", "AMD", "NVDA", "PBI", "TGT"]
#ohlcv = yf.download(tickers, period="max")
ohlcv = yf.download(tickers, start="2015-02-02", end="2021-12-01")
prices = ohlcv["Adj Close"]
prices.tail()

###

```

```

from portfOpt import expectedReturns

```

```

rets = expectedReturns.returnsFromPrices(prices)
rets.tail()

```

```
### md
```

Оптимизация HRP

HRP использует совершенно другой бэкэнд.
В будущем будет возможность добавления передатать ограничения или указать целевую функцию.

```
###
```

```
from portfOpt import HRPopt
```

```
###
```

```
hrp = HRPopt(rets)
hrp.optimize()
weights = hrp.cleanWeights()
weights
```

```
###
```

```
pd.Series(weights).plot.pie(figsize=(10, 10));
```

```
###
```

```
hrp.portfolioPerformance(verbose=True);
```

```
### md
```

Построение

Построить дендрограмму (древовидную диаграмму) на основе иерархической структуры доходности активов очень просто.

```
###
```

```
from portfOpt import plot
```

```
plot.plotDendrogram(hrp);
```

```
###
```

Файл `_init_.py`

```
from .CriticalLineAlgorithm import CriticalLineAlgorithm
from .discreteAllocation import getLatestPrices, DiscreteAllocation
from .efficient_frontier import EfficientFrontier, EfficientSemivariance,
EfficientCVaR, EfficientCDaR
from .hierarchicalPortfolio import HRPopt
from .riskModels import CovShrinkage
from .blackLitterman import (
    marketImpliedPriorReturns,
    marketImpliedRiskAversion,
    BlackLittermanModel,
)
```

```
__version__ = "1.4.2"

__all__ = [
    "marketImpliedPriorReturns",
    "marketImpliedRiskAversion",
    "BlackLittermanModel",
    "CriticalLineAlgorithm",
    "getLatestPrices",
    "DiscreteAllocation",
    "EfficientFrontier",
    "EfficientSemivariance",
    "EfficientCVaR",
    "HRPOpt",
    "CovShrinkage",
]
```

Файл `baseOptimizer.py`

"""

Модуль `baseOptimizer` содержит родительские классы `BaseOptimizer`, из которых оптимизаторы унаследуют. `BaseConvexOptimizer` - это базовый класс для всех cvxpy (и scipy).

оптимизация.

Кроме того, мы определяем общую функцию полезности `PortfolioPerformance` для оценить доходность и риск для данного набора весов портфеля.

"""

```
import collections
import json
import warnings
import numpy as np
import pandas as pd
import cvxpy as cp
import scipy.optimize as sco
from . import objectiveFunctions
from . import exclusion
```

class `BaseOptimizer`:

"""

Методы:

- `setWeights()` создает `self.weights` (`np.ndarray`) из dict весов
- `cleanWeights()` округляет веса и обрезает почти до нуля.
- `saveWeightsToFile()` сохраняет веса в csv, json или txt.

"""

```
def __init__(self, n_assets, tickers=None):
```

"""

```
:param n_assets: number of assets
:type n_assets: int
:param tickers: name of assets
:type tickers: list
"""
```

```
self.nAssets = n_assets
if tickers is None:
    self.tickers = list(range(n_assets))
else:
    self.tickers = tickers
self.riskFreeRate = None
```

```

# Outputs
self.weights = None

def makeOutputWeights(self, weights=None):
    """
    Служебная функция для вывода значения веса из атрибута веса (np.array). Если
    нет переданных аргументов, то использовать self.tickers и self.weights. Если
    передан один аргумент, предположим это альтернативный массив весов, поэтому использовать self.tickers и аргумент.
    """
    if weights is None:
        weights = self.weights

    return collections.OrderedDict(zip(self.tickers, weights))

def setWeights(self, inputWeights):
    """
    Служебная функция для установки атрибута весов (np.array) из пользовательского
    ввода
    """
    self.weights = np.array([inputWeights[ticker] for ticker in self.tickers])

def cleanWeights(self, cutoff=1e-4, rounding=5):
    """
    Вспомогательный метод очистки исходных весов, установка любых весов,
    абсолютное значение которых значения ниже порогового значения до нуля, а остальные округляются.
    """
    if self.weights is None:
        raise AttributeError("Weights not yet computed")
    cleanWeights = self.weights.copy()
    cleanWeights[np.abs(cleanWeights) < cutoff] = 0
    if rounding is not None:
        if not isinstance(rounding, int) or rounding < 1:
            raise ValueError("rounding must be a positive integer")
        cleanWeights = np.round(cleanWeights, rounding)

    return self.makeOutputWeights(cleanWeights)

def saveWeightsToFile(self, filename="weights.csv"):
    """
    Утилита для сохранения Weights в текстовый файл.
    """
    cleanWeights = self.cleanWeights()

    ext = filename.split(".")[1]
    if ext == "csv":
        pd.Series(cleanWeights).to_csv(filename, header=False)
    elif ext == "json":
        with open(filename, "w") as fp:
            json.dump(cleanWeights, fp)
    elif ext == "txt":
        with open(filename, "w") as f:
            f.write(str(dict(cleanWeights)))
    else:
        raise NotImplementedError("Only supports .txt .json .csv")

class BaseConvexOptimizer(BaseOptimizer):
    """

```

BaseConvexOptimizer содержит множество частных переменных для использования *cvxpy*. Например, неизменная переменная оптимизации для весов хранится как *self.w*. Непосредственное взаимодействие с этими переменными напрямую обескуражен.

Methods:

- `addObjective()` добавляет (выпуклую) цель к задаче оптимизации
- `addConstraint()` добавляет ограничение к проблеме оптимизации
- `convexObjective()` решает для общей выпуклой цели с линейными ограничениями
- `nonconvexObjective()` решает общую невыпуклую цель с использованием *scipy* *backend*.

Это склонно к застреванию в локальных минимумах и, как правило, * не * рекомендуется.

- `setWeights()` создает *self.weights* (*np.ndarray*) из диктата весов
- `cleanWeights()` округляет веса и обрезает почти до нуля.
- `saveWeightsToFile()` сохраняет веса в *csv*, *json* или *txt*.

```
def __init__(
    self,
    nAssets,
    tickers=None,
    weightBounds=(0, 1),
    solver=None,
    verbose=False,
    solverOptions=None,
):
    super().__init__(nAssets, tickers)

    # Оптимизационные переменные
    self.w = cp.Variable(nAssets)
    self.objective = None
    self.additionalObjectives = []
    self.constraints = []
    self.lowerBounds = None
    self.upperBounds = None
    self.mapBoundsToConstraints(weightBounds)

    self.opt = None
    self.solver = solver
    self.verbose = verbose
    self.solverOptions = solverOptions if solverOptions else {}

def mapBoundsToConstraints(self, testBounds):
    """
    Преобразовует границы ввода в форму, приемлемую для cvxpy, и добавляет в
    список ограничений.
    """
    # Если это коллекция правильной длины, предположим, что все они являются
    # границами.
    if len(testBounds) == self.nAssets and not isinstance(
        testBounds[0], (float, int)
    ):
        bounds = np.array(testBounds, dtype=np.float)
        self.lowerBounds = np.nan_to_num(bounds[:, 0], nan=-np.inf)
        self.upperBounds = np.nan_to_num(bounds[:, 1], nan=np.inf)
    else:
        # В противном случае это должна быть пара.
        if len(testBounds) != 2 or not isinstance(testBounds, (tuple, list)):
            raise TypeError(
                "testBounds must be a pair (lower bound, upper bound) "
```

```

        "OR a collection of bounds for each asset"
    )
    lower, upper = testBounds

    # Замените значения None соответствующими +/- 1
    if np.isscalar(lower) or lower is None:
        lower = -1 if lower is None else lower
        self.lowerBounds = np.array([lower] * self.nAssets)
        upper = 1 if upper is None else upper
        self.upperBounds = np.array([upper] * self.nAssets)
    else:
        self.lowerBounds = np.nan_to_num(lower, nan=-1)
        self.upperBounds = np.nan_to_num(upper, nan=1)

    self.constraints.append(self.w >= self.lowerBounds)
    self.constraints.append(self.w <= self.upperBounds)

def solveCvxpyOptProblem(self):
    """
    Вспомогательный метод для решения проблемы cvxpy и проверки вывода,
    после определения целей и ограничений
    """
    try:
        self.opt = cp.Problem(cp.Minimize(self.objective), self.constraints)

        if self.solver is not None:
            self.opt.solve(
                solver=self.solver, verbose=self.verbose, **self.solverOptions
            )
        else:
            self.opt.solve(verbose=self.verbose, **self.solverOptions)
    except (TypeError, cp.DCPError) as e:
        raise exclusion.OptimizationError from e

    if self.opt.status not in {"optimal", "optimal_inaccurate"}:
        raise exclusion.OptimizationError(
            "Solver status: {}".format(self.opt.status)
        )
    self.weights = self.w.value.round(16) + 0.0 # +0.0 удаляет ноль со знаком
    return self.makeOutputWeights()

def addObjective(self, newObjective, **kwargs):
    """
    Добавьте новый член в целевую функцию. Этот член должен быть выпуклым,
    и построен из атомарных функций cvxpy.

    Пример:

    def L1_norm(w, k=1):
        return k * cp.norm(w, 1)

    ef.addObjective(L1_norm, k=2)
    """
    self.additionalObjectives.append(newObjective(self.w, **kwargs))

def addConstraint(self, newConstraint):
    """
    Добавьте новое ограничение к проблеме оптимизации. Это ограничение должно
    удовлетворять правилам DCP,
    т.е. быть либо ограничением линейного равенства, либо ограничением выпуклого

```

неравенства.

Примеры::

```

        ef.addConstraint(lambda x : x[0] == 0.02)
        ef.addConstraint(lambda x : x >= 0.01)
        ef.addConstraint(lambda x: x <= np.array([0.01, 0.08, ..., 0.5]))
    """
    if not callable(newConstraint):
        raise TypeError("New constraint must be provided as a lambda function")
    self.constraints.append(newConstraint(self.w))

def addSectorConstraints(self, sectorMapper, sectorLower, sectorUpper):
    """
    Добавляет ограничения на сумму весов различных групп активов.
    Чаще всего это будут ограничения сектора, например, подверженность портфеля
    tech.
    """
    if np.any(self.lowerBounds < 0):
        warnings.warn(
            "Sector constraints may not produce reasonable results if shorts are
allowed."
        )
    for sector in sectorUpper:
        isSector = [sectorMapper[t] == sector for t in self.tickers]
        self.constraints.append(cp.sum(self.w[isSector]) <= sectorUpper[sector])
    for sector in sectorLower:
        isSector = [sectorMapper[t] == sector for t in self.tickers]
        self.constraints.append(cp.sum(self.w[isSector]) >= sectorLower[sector])

def convexObjective(self, customObjective, weightsSumToOne=True, **kwargs):
    """
    Оптимизируйте пользовательскую выпуклую целевую функцию. Ограничения следует
    добавлять с помощью
    ``ef.addConstraint ()``. Аргументы оптимизатора должны передаваться как
    аргументы ключевого слова.
    """
    self.objective = customObjective(self.w, **kwargs)

    for obj in self.additionalObjectives:
        self.objective += obj

    if weightsSumToOne:
        self.constraints.append(cp.sum(self.w) == 1)

    return self.solveCvxpyOptProblem()

def nonconvexObjective(
    self,
    customObjective,
    objectiveArgs=None,
    weightsSumToOne=True,
    constraints=None,
    solver="SLSQP",
    initialGuess=None,
):
    """
    Оптимизируйте некоторую целевую функцию, используя scipy backend. Это может
    поддерживать невыпуклые цели и нелинейные ограничения, но может застрять
    на локальных минимумах.
    """

```

```

# Дезинфекция входных данных
if not isinstance(objectiveArgs, tuple):
    objectiveArgs = (objectiveArgs,)

# Сделайте scipy границы
boundArray = np.vstack((self.lowerBounds, self.upperBounds)).T
bounds = list(map(tuple, boundArray))

if initialGuess is None:
    initialGuess = np.array([1 / self.nAssets] * self.nAssets)

# Построить ограничения
finalConstraints = []
if weightsSumToOne:
    finalConstraints.append({"type": "eq", "fun": lambda x: np.sum(x) - 1})
if constraints is not None:
    finalConstraints += constraints

result = sco.minimize(
    customObjective,
    x0=initialGuess,
    args=objectiveArgs,
    method=solver,
    bounds=bounds,
    constraints=finalConstraints,
)
self.weights = result["x"]
return self.makeOutputWeights()

def portfolioPerformance(
    weights, expectedReturns, covMatrix, verbose=False, riskFreeRate=0.02
):
    """
    После оптимизации рассчитайте (и при необходимости распечатайте)
    производительность оптимального
    портфолио. В настоящее время рассчитывает ожидаемую доходность, волатильность и
    коэффициент Шарпа.
    """
    if isinstance(weights, dict):
        if isinstance(expectedReturns, pd.Series):
            tickers = list(expectedReturns.index)
        elif isinstance(covMatrix, pd.DataFrame):
            tickers = list(covMatrix.columns)
        else:
            tickers = list(range(len(expectedReturns)))
        newWeights = np.zeros(len(tickers))

        for i, k in enumerate(tickers):
            if k in weights:
                newWeights[i] = weights[k]
            if newWeights.sum() == 0:
                raise ValueError("Weights add to zero, or ticker names don't match")
    elif weights is not None:
        newWeights = np.asarray(weights)
    else:
        raise ValueError("Weights is None")

    sigma = np.sqrt(objectiveFunctions.portfolioVariance(newWeights, covMatrix))

    if expectedReturns is not None:

```

```

mu = objectiveFunctions.portfolioReturn(
    newWeights, expectedReturns, negative=False
)

sharpe = objectiveFunctions.sharpeRatio(
    newWeights,
    expectedReturns,
    covMatrix,
    riskFreeRate=riskFreeRate,
    negative=False,
)
if verbose:
    print("Expected annual return: {:.1f}%".format(100 * mu))
    print("Annual volatility: {:.1f}%".format(100 * sigma))
    print("Sharpe Ratio: {:.2f}".format(sharpe))
return mu, sigma, sharpe
else:
    if verbose:
        print("Annual volatility: {:.1f}%".format(100 * sigma))
    return None, sigma, None

```

Файл blackLitterman.py

"""

Модуль *blacklitterman* содержит класс *BlackLittermanModel*, который генерирует апостериорные оценки ожидаемой доходности с учетом априорной оценки и предоставленной пользователем

Просмотры. Кроме того, определены две функции полезности, которые вычисляют:

- предварительная оценка прибыли, предполагаемая рынком (market-implied prior estimate of returns)
- рыночный параметр неприятия риска (market-implied risk-aversion parameter)

"""

```

import sys
import warnings
import numpy as np
import pandas as pd
from . import baseOptimizer

```

```

def marketImpliedPriorReturns(
    marketCaps, riskAversion, covMatrix, riskFreeRate=0.02
):

```

"""

Вычислите предварительную оценку доходности, предполагаемую рыночными весами. Другими словами, учитывая вклад каждого актива в риск рынка портфель, сколько мы ожидаем получить компенсацию?

.. математика ::

$\Pi = \Delta \Sigma w_{\text{mkt}}$

"""

```

if not isinstance(covMatrix, pd.DataFrame):
    warnings.warn(
        "If covMatrix is not a dataframe, market cap index must be aligned to covMatrix",
        RuntimeWarning,
    )
mcaps = pd.Series(marketCaps)
mktWeights = mcaps / mcaps.sum()
#  $\Pi$  - это избыточная доходность, поэтому для получения прибыли необходимо

```

добавить `riskFreeRate`.

```
return riskAversion * covMatrix.dot(mktWeights) + riskFreeRate
```

```
def marketImpliedRiskAversion(marketPrices, frequency=252, riskFreeRate=0.02):
```

```
    """
```

Рассчитайте предполагаемый рыночный параметр неприятия риска (т. Е. Рыночную цену риска)

исходя из рыночных цен. Например, если на рынке избыточная доходность 10% в год. при 5% дисперсии параметр неприятия риска равен 2, т. е. вам необходимо получить 2-кратную компенсацию дисперсия.

```
    .. математика ::
```

```
    
$$\Delta = \frac{R - R_f}{\sigma^2}$$

```

```
    """
```

```
    if not isinstance(marketPrices, (pd.Series, pd.DataFrame)):
        raise TypeError("Please format marketPrices as a pd.Series")
```

```
    rets = marketPrices.pct_change().dropna()
```

```
    r = rets.mean() * frequency
```

```
    var = rets.var() * frequency
```

```
    return (r - riskFreeRate) / var
```

```
class BlackLittermanModel(baseOptimizer.BaseOptimizer):
```

```
    """
```

Объект BlackLittermanModel (наследуемый от BaseOptimizer) содержит требует конкретный формат ввода, определяющий предварительные просмотры, неопределенность во взглядах,

и матрица выбора для отображения представлений во вселенной активов. Затем мы можем вычислить

апостериорные оценки доходности и ковариации. Были предоставлены вспомогательные методы

по возможности указать значения по умолчанию.

- ``defaultOmega()`` - просмотр неопределенности, пропорциональной дисперсии актива

- ``idzorekMethod()`` - преобразовать просмотры, указанные в процентах, в неопределенности BL

- ``blReturns()`` - апостериорная оценка доходности

- ``blCov()`` - апостериорная оценка ковариации

- ``blWeights()`` - веса, подразумеваемые апостериорными возвратами

- ``portfolioPerformance()`` рассчитывает ожидаемую доходность, волатильность и коэффициент Шарпа для выделенного портфеля.

- ``setWeights()`` создает self.weights (np.ndarray) из диктата весов

- ``cleanWeights()`` округляет веса и обрезает почти до нуля.

- ``saveWeightsToFile()`` сохраняет веса в csv, json или txt.

```
    """
```

```
def __init__(
```

```
    self,
```

```
    covMatrix,
```

```
    pi=None,
```

```
    absoluteViews=None,
```

```
    Q=None,
```

```
    P=None,
```

```
    omega=None,
```

```
    viewConfidences=None,
```

```
    tau=0.05,
```

```

riskAversion=1,
**kwargs
):
    if sys.version_info[1] == 5: # pragma: без обложки
        warnings.warn(
            "When using python 3.5 you must explicitly construct the Black-
Litterman inputs"
        )

    # Сохранять необработанные фреймы данных
    self.rawCovMatrix = covMatrix

    # Инициализировать базовый оптимизатор
    if isinstance(covMatrix, np.ndarray):
        self.covMatrix = covMatrix
        super().__init__(len(covMatrix), list(range(len(covMatrix))))
    else:
        self.covMatrix = covMatrix.values
        super().__init__(len(covMatrix), covMatrix.columns)

    # Дезинфекция входных данных
    if absoluteViews is not None:
        self.Q, self.P = self.parseViews(absoluteViews)
    else:
        self.setQP(Q, P)
    self.setRiskAversion(riskAversion)
    self.setPi(pi, **kwargs)
    self.setTau(tau)
    # Убедитесь, что все размеры работают
    self.checkAttributeDimensions()

    self.setOmega(omega, viewConfidences)

    # Частные посредники
    self.tauSigmaP = None
    self._A = None

    self.posteriorRets = None
    self.posteriorCov = None

def parseViews(self, absoluteViews):
    """
    Учитывая набор (dict или серию) абсолютных представлений, построить
    соответствующий вектор представлений и матрицу выбора. Взгляды должны
    быть подмножеством тикеров в ковариационной матрице.

    {"AAPL": 0.20, "GOOG": 0.12, "XOM": -0.30}

    """
    if not isinstance(absoluteViews, (dict, pd.Series)):
        raise TypeError("views should be a dict or pd.Series")
    # Принуждение к сериям
    views = pd.Series(absoluteViews)
    k = len(views)

    Q = np.zeros((k, 1))
    P = np.zeros((k, self.nAssets))

    for i, view_ticker in enumerate(views.keys()):
        try:
            Q[i] = views[view_ticker]
            P[i, list(self.tickers).index(view_ticker)] = 1

```

```

        except ValueError:
            # Можно было бы сделать это умнее, просто пропустить
            raise ValueError("Providing a view on an asset not in the universe")
    return Q, P

def setQP(self, Q, P):
    if isinstance(Q, (pd.Series, pd.DataFrame)):
        self.Q = Q.values.reshape(-1, 1)
    elif isinstance(Q, np.ndarray):
        self.Q = Q.reshape(-1, 1)
    else:
        raise TypeError("Q must be an array or dataframe")

    if isinstance(P, pd.DataFrame):
        self.P = P.values
    elif isinstance(P, np.ndarray):
        self.P = P
    elif len(self.Q) == self.nAssets:
        # Если предоставляется представление для каждого актива, P по умолчанию
        # в единичную матрицу.
        self.P = np.eye(self.nAssets)
    else:
        raise TypeError("P must be an array or dataframe")

def setPi(self, pi, **kwargs):
    if pi is None:
        warnings.warn("Running Black-Litterman with no prior.")
        self.pi = np.zeros((self.nAssets, 1))
    elif isinstance(pi, (pd.Series, pd.DataFrame)):
        self.pi = pi.values.reshape(-1, 1)
    elif isinstance(pi, np.ndarray):
        self.pi = pi.reshape(-1, 1)
    elif pi == "market":
        if "marketCaps" not in kwargs:
            raise ValueError(
                "Please pass a series/array of market caps via the marketCaps
keyword argument"
            )
        marketCaps = kwargs.get("marketCaps")
        riskFreeRate = kwargs.get("riskFreeRate", 0)

        marketPrior = marketImpliedPriorReturns(
            marketCaps, self.riskAversion, self.rawCovMatrix, riskFreeRate
        )
        self.pi = marketPrior.values.reshape(-1, 1)
    elif pi == "equal":
        self.pi = np.ones((self.nAssets, 1)) / self.nAssets
    else:
        raise TypeError("pi must be an array or series")

def setTau(self, tau):
    if tau <= 0 or tau > 1:
        raise ValueError("tau should be between 0 and 1")
    self.tau = tau

def setRiskAversion(self, riskAversion):
    if riskAversion <= 0:
        raise ValueError("riskAversion should be a positive float")
    self.riskAversion = riskAversion

def setOmega(self, omega, viewConfidences):
    if isinstance(omega, pd.DataFrame):

```

```

        self.omega = omega.values
    elif isinstance(omega, np.ndarray):
        self.omega = omega
    elif omega == "idzorek":
        if viewConfidences is None:
            raise ValueError(
                "To use Idzorek's method, please supply a vector of percentage "
                "confidence levels for each view."
            )
        if not isinstance(viewConfidences, np.ndarray):
            try:
                viewConfidences = np.array(viewConfidences).reshape(-1, 1)
                assert viewConfidences.shape[0] == self.Q.shape[0]
                assert np.issubdtype(viewConfidences.dtype, np.number)
            except AssertionError:
                raise ValueError(
                    "viewConfidences should be a numpy 1D array or vector with the
same length "
                    "as the number of views."
                )

        self.omega = BlackLittermanModel.idzorekMethod(
            viewConfidences,
            self.covMatrix,
            self.pi,
            self.Q,
            self.P,
            self.tau,
            self.riskAversion,
        )
    elif omega is None or omega == "default":
        self.omega = BlackLittermanModel.defaultOmega(
            self.covMatrix, self.P, self.tau
        )
    else:
        raise TypeError("self.omega must be a square array, dataframe, or string")

K = len(self.Q)
assert self.omega.shape == (K, K), "omega must have dimensions KxK"

def checkAttributeDimensions(self):
    """
    Вспомогательный метод, гарантирующий, что все атрибуты, созданные
инициализатором
иметь правильные размеры, чтобы в дальнейшем избежать ошибок линейной алгебры.
    """
    N = self.nAssets
    K = len(self.Q)
    assert self.pi.shape == (N, 1), "pi must have dimensions Nx1"
    assert self.P.shape == (K, N), "P must have dimensions KxN"
    assert self.covMatrix.shape == (N, N), "covMatrix must have shape NxN"

@staticmethod
def defaultOmega(covMatrix, P, tau):
    """
    Если матрица неопределенности омега не указана, рассчитываем по методу
He and Litterman (1999), так что соотношение омега/тау пропорционально
дисперсия просмотра портфолио.
    """
    return np.diag(np.diag(tau * P @ covMatrix @ P.T))

@staticmethod

```

```

def idzorekMethod(viewConfidences, covMatrix, pi, Q, P, tau, riskAversion=1):
    """
    Используйте метод Идзорека для создания матрицы неопределенности, заданной
    пользователем.
    процентное доверие. Мы используем решение в замкнутой форме, описываемое
    формулой
    Jay Walters in The Black-Litterman Model in Detail (2014).
    """
    viewOmegas = []
    for viewIdx in range(len(Q)):
        conf = viewConfidences[viewIdx]

        if conf < 0 or conf > 1:
            raise ValueError("View confidences must be between 0 and 1")

        # Специальный обработчик, чтобы избежать деления на ноль.
        # Если значение равно нулю, вернуть очень большое число как
        неопределенность
        if conf == 0:
            viewOmegas.append(1e6)
            continue

        PView = P[viewIdx].reshape(1, -1)
        alpha = (1 - conf) / conf # formula (44)
        omega = tau * alpha * PView @ covMatrix @ PView.T # формула (41)
        viewOmegas.append(omega.item())

    return np.diag(viewOmegas)

def blReturns(self):
    """
    Рассчитайте апостериорную оценку вектора доходности,
    учитывая взгляды на некоторые активы.
    """

    if self.tauSigmaP is None:
        self.tauSigmaP = self.tau * self.covMatrix @ self.P.T

    # Решите линейную систему Ax = b, чтобы избежать инверсии
    if self._A is None:
        self._A = (self.P @ self.tauSigmaP) + self.omega
        b = self.Q - self.P @ self.pi
        postRets = self.pi + self.tauSigmaP @ np.linalg.solve(self._A, b)
        return pd.Series(postRets.flatten(), index=self.tickers)

def blCov(self):
    """
    Вычислить апостериорную оценку ковариационной матрицы,
    учитывая взгляды на некоторые активы. По материалам He and Litterman (2002).
    Предполагается, что омега диагональная. Если это не так,
    пожалуйста, установите omegaInv вручную.
    """

    if self.tauSigmaP is None:
        self.tauSigmaP = self.tau * self.covMatrix @ self.P.T
    if self._A is None:
        self._A = (self.P @ self.tauSigmaP) + self.omega

    b = self.tauSigmaP.T
    M = self.tau * self.covMatrix - self.tauSigmaP @ np.linalg.solve(self._A, b)
    posterior_cov = self.covMatrix + M
    return pd.DataFrame(posterior_cov, index=self.tickers, columns=self.tickers)

```

```

def blWeights(self, riskAversion=None):
    """
    Вычислите веса, подразумеваемые апостериорной доходностью, учитывая
    рыночная цена риска. Технически это применимо к любому
    оценка ожидаемой прибыли, и на самом деле это особый случай
    оптимизации среднего отклонения

    .. математика ::

        
$$w = (\Delta \Sigma)^{-1} E(R)$$

    """
    if riskAversion is None:
        riskAversion = self.riskAversion

    self.posteriorRets = self.blReturns()
    A = riskAversion * self.covMatrix
    b = self.posteriorRets
    rawWeights = np.linalg.solve(A, b)
    self.weights = rawWeights / rawWeights.sum()
    return self.makeOutputWeights()

def optimize(self, riskAversion=None):
    """
    Псевдоним для blWeights для согласованности с другими методами.
    """
    return self.blWeights(riskAversion)

def portfolioPerformance(self, verbose=False, riskFreeRate=0.02):
    """
    После оптимизации рассчитайте (и при необходимости распечатайте)
    производительность оптимального
    портфолио. В настоящее время рассчитывает ожидаемую доходность, волатильность
    и коэффициент Шарпа.
    Этот метод использует апостериорные возвращения BL и ковариационную матрицу.
    """
    if self.posteriorCov is None:
        self.posteriorCov = self.blCov()
    return baseOptimizer.portfolioPerformance(
        self.weights,
        self.posteriorRets,
        self.posteriorCov,
        verbose,
        riskFreeRate,
    )

```

Файл CriticalLineAlgorithm.py

```

"""
Модуль CriticalLineAlgorithm содержит класс CriticalLineAlgorithm, который
генерирует оптимальные портфели с использованием реализованного алгоритма критической
линии
Маркос Лопес де Прадо и Дэвид Бейли. (Marcos Lopez de Prado and David Bailey.)
"""

```

```

import numpy as np
import pandas as pd
from . import baseOptimizer

```

```

class CriticalLineAlgorithm(baseOptimizer.BaseOptimizer):

```

```

def __init__(self, expectedReturns, covMatrix, weightBounds=(0, 1)):

    self.mean = np.array(expectedReturns).reshape((len(expectedReturns), 1))
    self.expectedReturns = self.mean.reshape((len(self.mean),))
    self.covMatrix = np.asarray(covMatrix)

    # Bounds
    if len(weightBounds) == len(self.mean) and not isinstance(
        weightBounds[0], (float, int)
    ):
        self.lB = np.array([b[0] for b in weightBounds]).reshape(-1, 1)
        self.uB = np.array([b[1] for b in weightBounds]).reshape(-1, 1)
    else:
        if isinstance(weightBounds[0], (float, int)):
            self.lB = np.ones(self.mean.shape) * weightBounds[0]
        else:
            self.lB = np.array(weightBounds[0]).reshape(self.mean.shape)
        if isinstance(weightBounds[1], (float, int)):
            self.uB = np.ones(self.mean.shape) * weightBounds[1]
        else:
            self.uB = np.array(weightBounds[1]).reshape(self.mean.shape)

    self.w = [] # решение
    self.lS = [] # лямбды
    self.g = [] # гамма
    self.f = [] # свободный вес

    self.frontierValues = None # результат вычисления границы эффективности

    if isinstance(expectedReturns, pd.Series):
        tickers = list(expectedReturns.index)
    else:
        tickers = list(range(len(self.mean)))
    super().__init__(len(tickers), tickers)

    @staticmethod
    def _infnone(x):
        """
        Вспомогательный метод для сопоставления None с плавающей бесконечностью.
        """
        return float("-inf") if x is None else x

    def _init_algo(self):
        # Инициализировать алгоритм
        # 1) Сформировать структурированный массив
        a = np.zeros((self.mean.shape[0]), dtype=[("id", int), ("mu", float)])
        b = [self.mean[i][0] for i in range(self.mean.shape[0])] # dump array into
list
        # заполнить структурированный массив
        a[:] = list(zip(list(range(self.mean.shape[0])), b))
        # 2) Сортировка структурированного массива
        b = np.sort(a, order="mu")
        # 3) Первый свободный вес
        i, w = b.shape[0], np.copy(self.lB)
        while sum(w) < 1:
            i -= 1
            w[b[i][0]] = self.uB[b[i][0]]
            w[b[i][0]] += 1 - sum(w)
        return [b[i][0]], w

    def computeBi(self, c, bi):

```

```

if c > 0:
    bi = bi[1][0]
if c < 0:
    bi = bi[0][0]
return bi

def computeW(self, covarFInv, covarFB, meanF, wB):
    # 1) вычислить гамму
    onesF = np.ones(meanF.shape)
    g1 = np.dot(np.dot(onesF.T, covarFInv), meanF)
    g2 = np.dot(np.dot(onesF.T, covarFInv), onesF)
    if wB is None:
        g, w1 = float(-self.ls[-1] * g1 / g2 + 1 / g2), 0
    else:
        onesB = np.ones(wB.shape)
        g3 = np.dot(onesB.T, wB)
        g4 = np.dot(covarFInv, covarFB)
        w1 = np.dot(g4, wB)
        g4 = np.dot(onesF.T, w1)
        g = float(-self.ls[-1] * g1 / g2 + (1 - g3 + g4) / g2)
    # 2) вычислить веса
    w2 = np.dot(covarFInv, onesF)
    w3 = np.dot(covarFInv, meanF)
    return -w1 + g * w2 + self.ls[-1] * w3, g

def computeLambda(self, covarFInv, covarFB, meanF, wB, i, bi):
    # 1) C
    onesF = np.ones(meanF.shape)
    c1 = np.dot(np.dot(onesF.T, covarFInv), onesF)
    c2 = np.dot(covarFInv, meanF)
    c3 = np.dot(np.dot(onesF.T, covarFInv), meanF)
    c4 = np.dot(covarFInv, onesF)
    c = -c1 * c2[i] + c3 * c4[i]
    if c == 0: # pragma: no cover
        return None, None
    # 2) BI
    if type(bi) == list:
        bi = self.computeBi(c, bi)
    # 3) Лямбда
    if wB is None:
        # Все бесплатные активы
        return float((c4[i] - c1 * bi) / c), bi
    else:
        onesB = np.ones(wB.shape)
        l1 = np.dot(onesB.T, wB)
        l2 = np.dot(covarFInv, covarFB)
        l3 = np.dot(l2, wB)
        l2 = np.dot(onesF.T, l3)
        return float((1 - l1 + l2) * c4[i] - c1 * (bi + l3[i])) / c, bi

def getMatrices(self, f):
    # Срез covarF, covarFB, covarB, meanF, meanB, wF, wB
    covarF = self.reduceMatrix(self.covMatrix, f, f)
    meanF = self.reduceMatrix(self.mean, f, [0])
    b = self.getB(f)
    covarFB = self.reduceMatrix(self.covMatrix, f, b)
    wB = self.reduceMatrix(self.w[-1], b, [0])
    return covarF, covarFB, meanF, wB

def getB(self, f):
    return self.diffLists(list(range(self.mean.shape[0])), f)

```

```

@staticmethod
def diffLists(list1, list2):
    return list(set(list1) - set(list2))

@staticmethod
def reduceMatrix(matrix, listX, listY):
    # Сократите матрицу до предоставленного списка строк и столбцов
    if len(listX) == 0 or len(listY) == 0:
        return
    matrix_ = matrix[:, listY[0] : listY[0] + 1]
    for i in listY[1:]:
        a = matrix[:, i : i + 1]
        matrix_ = np.append(matrix_, a, 1)
    matrix__ = matrix_[listX[0] : listX[0] + 1, :]
    for i in listX[1:]:
        a = matrix__[i : i + 1, :]
        matrix__ = np.append(matrix__, a, 0)
    return matrix__

def _purge_num_err(self, tol):
    # Устранение нарушений ограничений неравенства (связанных с плохо обусловленной матрицей COV)
    i = 0
    while True:
        flag = False
        if i == len(self.w):
            break
        if abs(sum(self.w[i]) - 1) > tol:
            flag = True
        else:
            for j in range(self.w[i].shape[0]):
                if (
                    self.w[i][j] - self.lB[j] < -tol
                    or self.w[i][j] - self.uB[j] > tol
                ): # прагма: без обложки
                    flag = True
                    break
        if flag is True:
            del self.w[i]
            del self.ls[i]
            del self.g[i]
            del self.f[i]
        else:
            i += 1

def _purge_excess(self):
    # Устранение нарушений выпуклой оболочки
    i, repeat = 0, False
    while True:
        if repeat is False:
            i += 1
        if i == len(self.w) - 1:
            break
        w = self.w[i]
        mu = np.dot(w.T, self.mean)[0, 0]
        j, repeat = i + 1, False
        while True:
            if j == len(self.w):
                break
            w = self.w[j]
            mu_ = np.dot(w.T, self.mean)[0, 0]
            if mu < mu_:

```

```

        del self.w[i]
        del self.ls[i]
        del self.g[i]
        del self.f[i]
        repeat = True
        break
    else:
        j += 1

def goldenSection(self, obj, a, b, **kargs):
    # Метод золотого сечения. Максимум, если kargs ['minimum'] == False is passed
    tol, sign, args = 1.0e-9, 1, None
    if "minimum" in kargs and kargs["minimum"] is False:
        sign = -1
    if "args" in kargs:
        args = kargs["args"]
    numIter = int(np.ceil(-2.078087 * np.log(tol / abs(b - a))))
    r = 0.618033989
    c = 1.0 - r
    # Инициализировать
    x1 = r * a + c * b
    x2 = c * a + r * b
    f1 = sign * obj(x1, *args)
    f2 = sign * obj(x2, *args)
    # Цикл
    for i in range(numIter):
        if f1 > f2:
            a = x1
            x1 = x2
            f1 = f2
            x2 = c * a + r * b
            f2 = sign * obj(x2, *args)
        else:
            b = x2
            x2 = x1
            f2 = f1
            x1 = r * a + c * b
            f1 = sign * obj(x1, *args)
    if f1 < f2:
        return x1, sign * f1
    else:
        return x2, sign * f2

def _eval_sr(self, a, w0, w1):
    # Оцените SR портфеля в выпуклой комбинации
    w = a * w0 + (1 - a) * w1
    b = np.dot(w.T, self.mean)[0, 0]
    c = np.dot(np.dot(w.T, self.covMatrix), w)[0, 0] ** 0.5
    return b / c

def _solve(self):
    # Вычислить точки поворота, свободные наборы и веса
    f, w = self._init_algo()
    self.w.append(np.copy(w)) # магазин решение
    self.ls.append(None)
    self.g.append(None)
    self.f.append(f[:])
    while True:
        # 1) случай а): связана одна свободная гиря
        l_in = None
        if len(f) > 1:
            covarF, covarFB, meanF, wB = self.getMatrices(f)

```

```

covarF_inv = np.linalg.inv(covarF)
j = 0
for i in f:
    l, bi = self.computeLambda(
        covarF_inv, covarFB, meanF, wB, j, [self.lB[i], self.uB[i]]
    )
    if CriticalLineAlgorithm._infnone(l) >
CriticalLineAlgorithm._infnone(l_in):
        l_in, i_in, bi_in = l, i, bi
        j += 1
# 2) случай б): свободный ограниченный вес
l_out = None
if len(f) < self.mean.shape[0]:
    b = self.getB(f)
    for i in b:
        covarF, covarFB, meanF, wB = self.getMatrices(f + [i])
        covarF_inv = np.linalg.inv(covarF)
        l, bi = self.computeLambda(
            covarF_inv,
            covarFB,
            meanF,
            wB,
            meanF.shape[0] - 1,
            self.w[-1][i],
        )
        if (self.ls[-1] is None or l < self.ls[-1]) and l >
CriticalLineAlgorithm._infnone(
            l_out
        ):
            l_out, i_out = l, i
if (l_in is None or l_in < 0) and (l_out is None or l_out < 0):
    # 3) вычислить решение с минимальной дисперсией
    self.ls.append(0)
    covarF, covarFB, meanF, wB = self.getMatrices(f)
    covarF_inv = np.linalg.inv(covarF)
    meanF = np.zeros(meanF.shape)
else:
    # 4) решите лямбда
    if CriticalLineAlgorithm._infnone(l_in) >
CriticalLineAlgorithm._infnone(l_out):
        self.ls.append(l_in)
        f.remove(i_in)
        w[i_in] = bi_in # установить значение на правильной границе
    else:
        self.ls.append(l_out)
        f.append(i_out)
        covarF, covarFB, meanF, wB = self.getMatrices(f)
        covarF_inv = np.linalg.inv(covarF)
# 5) вычислить вектор решения
wF, g = self.computeW(covarF_inv, covarFB, meanF, wB)
for i in range(len(f)):
    w[f[i]] = wF[i]
self.w.append(np.copy(w)) # store решение
self.g.append(g)
self.f.append(f[:])
if self.ls[-1] == 0:
    break
# 6) Очистить точки поворота
self._purge_num_err(10e-10)
self._purge_excess()

def maxSharpe(self):

```

```

"""
Увеличьте коэффициент Шарпа.
"""
if not self.w:
    self._solve()
# 1) Вычислить локальный портфель максимального SR между любыми двумя
соседними поворотными точками
w_sr, sr = [], []
for i in range(len(self.w) - 1):
    w0 = np.copy(self.w[i])
    w1 = np.copy(self.w[i + 1])
    kargs = {"minimum": False, "args": (w0, w1)}
    a, b = self.goldenSection(self._eval_sr, 0, 1, **kargs)
    w_sr.append(a * w0 + (1 - a) * w1)
    sr.append(b)

self.weights = w_sr[sr.index(max(sr))].reshape((self.nAssets,))
return self.makeOutputWeights()

def minVolatility(self):
    """
    Минимизируйте волатильность.
    """
    if not self.w:
        self._solve()
    var = []
    for w in self.w:
        a = np.dot(np.dot(w.T, self.covMatrix), w)
        var.append(a)
    # return min(var)**.5, self.w[var.index(min(var))]
    self.weights = self.w[var.index(min(var))].reshape((self.nAssets,))
    return self.makeOutputWeights()

def efficientFrontier(self, points=100):
    """
    Эффективно вычислите всю эффективную границу
    """
    if not self.w:
        self._solve()

    mu, sigma, weights = [], [], []
    # удалите 1, чтобы избежать дублирования
    a = np.linspace(0, 1, points // len(self.w))[:-1]
    b = list(range(len(self.w) - 1))
    for i in b:
        w0, w1 = self.w[i], self.w[i + 1]
        if i == b[-1]:
            # включить 1 в последней итерации
            a = np.linspace(0, 1, points // len(self.w))
        for j in a:
            w = w1 * j + (1 - j) * w0
            weights.append(np.copy(w))
            mu.append(np.dot(w.T, self.mean)[0, 0])
            sigma.append(np.dot(np.dot(w.T, self.covMatrix), w)[0, 0] ** 0.5)

    self.frontierValues = (mu, sigma, weights)
    return mu, sigma, weights

def setWeights(self, _):
    # Переопределяет родительский метод, поскольку setWeights ничего не делает.
    raise NotImplementedError("setWeights does nothing for CriticalLineAlgorithm")

```

```

def portfolioPerformance(self, verbose=False, riskFreeRate=0.02):
    """
    После оптимизации рассчитайте (и при необходимости распечатайте)
    производительность оптимального
    портфолио. В настоящее время рассчитывает ожидаемую доходность, волатильность
    и коэффициент Шарпа.
    """
    return baseOptimizer.portfolioPerformance(
        self.weights,
        self.expectedReturns,
        self.covMatrix,
        verbose,
        riskFreeRate,
    )

```

Файл discreteAllocation.py

```

"""
Модуль «Discrete_allocation» содержит класс «DiscreteAllocation», который
предлагает несколько методов для генерации дискретного распределения портфеля из
непрерывных весов.
"""

```

```

import collections
import numpy as np
import pandas as pd
import cvxpy as cp
from . import exclusion

```

```

def getLatestPrices(prices):
    """
    Вспомогательный инструмент, который извлекает самые последние цены на активы из
    фрейма данных
    цены на активы, необходимые для дискретного распределения.
    """
    if not isinstance(prices, pd.DataFrame):
        raise TypeError("prices not in a dataframe")
    return prices.ffill().iloc[-1]

```

```

class DiscreteAllocation:

```

```

    """

```

```

    Сгенерируйте дискретное распределение портфеля из непрерывных весов

```

```

- ``greedyPortfolio()`` - использует жадный алгоритм
- ``lpPortfolio()`` - использует линейное программирование
    """

```

```

def __init__(
    self, weights, latestPrices, totalPortfolioValue=10000, shortRatio=None
):
    if not isinstance(weights, dict):
        raise TypeError("weights should be a dictionary of {ticker: weight}")
    if any(np.isnan(val) for val in weights.values()):
        raise ValueError("weights should have no NaNs")
    if (not isinstance(latestPrices, pd.Series)) or any(np.isnan(latestPrices)):
        raise TypeError("latestPrices should be a pd.Series with no NaNs")
    if totalPortfolioValue <= 0:
        raise ValueError("totalPortfolioValue must be greater than zero")
    if shortRatio is not None and shortRatio < 0:

```

```

        raise ValueError("shortRatio must be non-negative")

# Отбросьте компании с незначительным весом. Используйте кортеж, потому что
порядок имеет значение.
self.weights = list(weights.items())
self.latestPrices = latestPrices
self.totalPortfolioValue = totalPortfolioValue
if shortRatio is None:
    self.shortRatio = sum((-x[1] for x in self.weights if x[1] < 0))
else:
    self.shortRatio = shortRatio

@staticmethod
def removeZeroPositions(allocation):
    """
    Функция полезности для удаления нулевых позиций (т.е. когда акции не
    покупаются)
    """
    return {k: v for k, v in allocation.items() if v != 0}

def allocationRmseError(self, verbose=True):
    """
    Служебная функция для расчета и печати ошибки RMSE между дискретизированными
    гури и непрерывные гури. RMSE использовался вместо MAE, потому что мы
    хотиме наказать за большие отклонения.
    """
    portfolioVal = 0
    for ticker, num in self.allocation.items():
        portfolioVal += num * self.latestPrices[ticker]

    sse = 0 # сумма квадратных ошибок
    for ticker, weight in self.weights:
        if ticker in self.allocation:
            allocationWeight = (
                self.allocation[ticker] * self.latestPrices[ticker] /
portfolioVal
            )
        else:
            allocationWeight = 0
        sse += (weight - allocationWeight) ** 2
        if verbose:
            print(
                "{:} allocated {:.3f}, desired {:.3f}".format(
                    ticker, allocationWeight, weight
                )
            )
    rmse = np.sqrt(sse / len(self.weights))
    print("Allocation has RMSE: {:.3f}".format(rmse))
    return rmse

def greedyPortfolio(self, reinvest=False, verbose=False):
    """
    Преобразование непрерывных весов в дискретное распределение портфеля
    используя жадный итеративный подход.
    """
    # Сортировать по убыванию веса
    self.weights.sort(key=lambda x: x[1], reverse=True)

    # Если в портфолио есть shorts
    if self.weights[-1][1] < 0:
        longs = {t: w for t, w in self.weights if w >= 0}
        shorts = {t: -w for t, w in self.weights if w < 0}

```

```

# Сделайте их суммой в один
longTotalWeight = sum(longs.values())
shortTotalWeight = sum(shorts.values())
longs = {t: w / longTotalWeight for t, w in longs.items()}
shorts = {t: w / shortTotalWeight for t, w in shorts.items()}

# Построение дискретных распределений только для длительного использования
для каждого
    shortVal = self.totalPortfolioValue * self.shortRatio
    longVal = self.totalPortfolioValue
    if reinvest:
        longVal += shortVal

    if verbose:
        print("\nAllocating long sub-portfolio...")
    da1 = DiscreteAllocation(
        longs, self.latestPrices[longs.keys()], totalPortfolioValue=longVal
    )
    longAlloc, longLeftover = da1.greedyPortfolio()

    if verbose:
        print("\nAllocating short sub-portfolio...")
    da2 = DiscreteAllocation(
        shorts,
        self.latestPrices[shorts.keys()],
        totalPortfolioValue=shortVal,
    )
    shortAlloc, shortLeftover = da2.greedyPortfolio()
    shortAlloc = {t: -w for t, w in shortAlloc.items()}

    # Combine and return
    self.allocation = longAlloc.copy()
    self.allocation.update(shortAlloc)
    self.allocation = self.removeZeroPositions(self.allocation)

    return self.allocation, longLeftover + shortLeftover

# В противном случае портфель будет только Long, и мы продолжаем использовать
жадный алгоритм.
availableFunds = self.totalPortfolioValue
sharesBought = []
buyPrices = []

# First round
for ticker, weight in self.weights:
    price = self.latestPrices[ticker]
    # Попытка купить меньшее целое количество акций, которое может равняться
    нулю.
    nShares = int(weight * self.totalPortfolioValue / price)
    cost = nShares * price
    # Поскольку все веса > 0 (Long only), мы всегда округляем в меньшую сторону
    nShares

    # поэтому стоимость всегда <= простая взвешенная доля стоимости портфеля,
    # поэтому мы не можем исчерпать средства только здесь.
    assert cost <= availableFunds, "Unexpectedly insufficient funds."
    availableFunds -= cost
    sharesBought.append(nShares)
    buyPrices.append(price)

# Второй раунд
while availableFunds > 0:

```

```

# Вычислить эквивалентные непрерывные веса акций, которые
# уже куплены
currentWeights = np.array(buyPrices) * np.array(sharesBought)
currentWeights /= currentWeights.sum()
idealWeights = np.array([i[1] for i in self.weights])
deficit = idealWeights - currentWeights

# Попытка купить актив, текущие веса которого отклоняются сильнее всего.
idx = np.argmax(deficit)
ticker, weight = self.weights[idx]
price = self.latestPrices[ticker]

# Если мы не можем позволить себе этот актив, ищем следующий по величине
дефицит, который мы
# можно купить.
counter = 0
while price > availableFunds:
    deficit[idx] = 0 # мы больше не можем покупать актив на idx
    idx = np.argmax(deficit) # найти следующий наиболее отклоняющийся
актив

# Если любое из этих условий выполнено, мы прерываем оба цикла while
# отсюда повторяющийся оператор ниже
if deficit[idx] < 0 or counter == 10:
    break

    ticker, weight = self.weights[idx]
    price = self.latestPrices[ticker]
    counter += 1

if deficit[idx] <= 0 or counter == 10: # pragma: без обложки
    # Dirty solution to break out
    break

# Покупайте по одной акции за раз
sharesBought[idx] += 1
availableFunds -= price

self.allocation = self.removeZeroPositions(
    collections.OrderedDict(zip([i[0] for i in self.weights], sharesBought))
)

if verbose:
    print("Funds remaining: {:.2f}".format(availableFunds))
    self.allocationRmseError(verbose)
return self.allocation, availableFunds

def lpPortfolio(self, reinvest=False, verbose=False, solver="GLPK_MI"):
    """
    Преобразование непрерывных весов в дискретное распределение портфеля
    с использованием целочисленного программирования.

    """
    if any([w < 0 for _, w in self.weights]):
        longs = {t: w for t, w in self.weights if w >= 0}
        shorts = {t: -w for t, w in self.weights if w < 0}

        # Сделайте их суммой в один
        longTotalWeight = sum(longs.values())
        shortTotalWeight = sum(shorts.values())
        longs = {t: w / longTotalWeight for t, w in longs.items()}
        shorts = {t: w / shortTotalWeight for t, w in shorts.items()}

```

*# Построение дискретных распределений только для длительного использования
для каждого*

```

shortVal = self.totalPortfolioValue * self.shortRatio
longVal = self.totalPortfolioValue
if reinvest:
    longVal += shortVal

if verbose:
    print("\nAllocating long sub-portfolio:")
da1 = DiscreteAllocation(
    longs, self.latestPrices[longs.keys()], totalPortfolioValue=longVal
)
longAlloc, longLeftover = da1.lpPortfolio()

if verbose:
    print("\nAllocating short sub-portfolio:")
da2 = DiscreteAllocation(
    shorts,
    self.latestPrices[shorts.keys()],
    totalPortfolioValue=shortVal,
)
shortAlloc, shortLeftover = da2.lpPortfolio()
shortAlloc = {t: -w for t, w in shortAlloc.items()}

# Объедините и верните
self.allocation = longAlloc.copy()
self.allocation.update(shortAlloc)
self.allocation = self.removeZeroPositions(self.allocation)
return self.allocation, longLeftover + shortLeftover

p = self.latestPrices.values
n = len(p)
w = np.fromiter([i[1] for i in self.weights], dtype=float)

# Целочисленное распределение
x = cp.Variable(n, integer=True)
# Оставшиеся доллары
r = self.totalPortfolioValue - p.T @ x

# Настроить линейную программу
eta = w * self.totalPortfolioValue - cp.multiply(x, p)
u = cp.Variable(n)
constraints = [eta <= u, eta >= -u, x >= 0, r >= 0]
objective = cp.sum(u) + r

opt = cp.Problem(cp.Minimize(objective), constraints)

if solver is not None and solver not in cp.installed_solvers():
    raise NameError("Solver {} is not installed.".format(solver))
opt.solve(solver=solver)

if opt.status not in {"optimal", "optimal_inaccurate"}: # pragma: no cover
    raise exclusion.OptimizationError("Please try greedyPortfolio")

vals = np.rint(x.value).astype(int)
self.allocation = self.removeZeroPositions(
    collections.OrderedDict(zip([i[0] for i in self.weights], vals))
)

if verbose:
    print("Funds remaining: {:.2f}".format(r.value))

```

```

        self.allocationRmseError()
    return self.allocation, r.value

```

Файл exclusion.py

```

"""
В ``exceptions`` модульные домики пользовательских исключений. В настоящее время
реализовано:

```

```

- OptimizationError
"""

```

```

class OptimizationError(Exception):

```

```

    """
    Когда процедура оптимизации дает сбой - обычно это означает
    что cvxpy не вернул флаг
    """

```

```

    def __init__(self, *args, **kwargs):
        defaultMessage = (
            "Please check your objectives/constraints or use a different solver."
        )
        super().__init__(defaultMessage, *args, **kwargs)

```

Файл expectedReturns.py

```

"""
Модуль expectedReturns предоставляет функции для оценки ожидаемой доходности
активов, что является обязательным входом при оптимизации среднего отклонения.

По соглашению, результаты этих методов предполагают * годовую(annual) * доходность.
Предполагается, что
* суточные(daily)* цены указаны, хотя на самом деле функции не зависят
на период времени (просто измените параметр ``частота(frequency)``). Цены на активы
должны быть указаны как
фрейм данных pandas

```

Все функции обрабатывают данные о ценах в процентные данные, прежде чем
расчет их соответствующих оценок ожидаемой прибыли.

В настоящее время реализовано:

- общая функция модели возврата (general return model function) , позволяющая запускать любую модель возврата из одной функции.
- средняя историческая доходность (mean historical return)
- экспоненциально взвешенная средняя историческая доходность (exponentially weighted mean historical return)
- Оценка доходности CAPM (CAPM estimate of returns)

Кроме того, сделаны вспомогательные функции для преобразования прибыли в цены и наоборот.

```

"""

```

```

import warnings
import pandas as pd
import numpy as np

```

```

#+
def returnsFromPrices(prices, logReturns=False):
    """

```

Рассчитайте возврат с учетом цен.

```
"""
if logReturns:
    return np.log(1 + prices.pct_change()).dropna(how="all")
else:
    return prices.pct_change().dropna(how="all")
```

```
def pricesFromReturns(returns, logReturns=False):
```

Рассчитайте псевдо-цены с учетом доходности. Это не настоящие цены, потому что все начальные цены установлены на 1, но при прохождении они ведут себя так, как задумано к любому методу

```
"""
if logReturns:
    ret = np.exp(returns)
else:
    ret = 1 + returns
ret.iloc[0] = 1 # установить псевдоцену первого дня
return ret.cumprod()
```

```
def returnModel(prices, method="meanHistoricalReturn", **kwargs):
```

*Вычислите оценку будущей доходности, используя модель доходности, указанную в ``методе``.
Должен быть одним из:*

```
- ``meanHistoricalReturn``
- ``emaHistoricalReturn``
- ``capmReturn``
```

```
"""
if method == "meanHistoricalReturn":
    return meanHistoricalReturn(prices, **kwargs)
elif method == "emaHistoricalReturn":
    return emaHistoricalReturn(prices, **kwargs)
elif method == "capmReturn":
    return capmReturn(prices, **kwargs)
else:
    raise NotImplementedError("Return model {} not implemented".format(method))
```

```
def meanHistoricalReturn(prices, returnsData=False, compounding=True, frequency=252):
```

*Рассчитайте среднегодовую (дневную) историческую доходность на основе исходных (дневных) цен на активы.
Используйте ``сложение(compounding)`` для переключения между средним геометрическим по умолчанию (CAGR) и среднее арифметическое.*

```
"""
if not isinstance(prices, pd.DataFrame):
    warnings.warn("prices are not in a dataframe", RuntimeWarning)
    prices = pd.DataFrame(prices)
if returnsData:
    returns = prices
else:
```

```

        returns = returnsFromPrices(prices)
    if compounding:
        return (1 + returns).prod() ** (frequency / returns.count()) - 1
    else:
        return returns.mean() * frequency

def emaHistoricalReturn(
    prices, returnsData=False, compounding=True, span=500, frequency=252
):
    """
    Рассчитайте экспоненциально взвешенное среднее значение (дневной) исторической
    доходности, получив
    больший вес для более свежих данных.
    """
    if not isinstance(prices, pd.DataFrame):
        warnings.warn("prices are not in a dataframe", RuntimeWarning)
        prices = pd.DataFrame(prices)
    if returnsData:
        returns = prices
    else:
        returns = returnsFromPrices(prices)

    if compounding:
        return (1 + returns.ewm(span=span).mean().iloc[-1]) ** frequency - 1
    else:
        return returns.ewm(span=span).mean().iloc[-1] * frequency

def capmReturn(
    prices,
    marketPrices=None,
    returnsData=False,
    riskFreeRate=0.02,
    compounding=True,
    frequency=252,
):
    """
    Вычислите оценку возврата, используя модель ценообразования капитальных активов.
    Согласно CAPM,
    доходность активов равна рыночной доходности плюс кодировка термина:  $\beta$ 
    бета`
    относительный риск актива.

    .. математика ::

        
$$R_i = R_f + \beta_i (E(R_m) - R_f)$$

    """
    if not isinstance(prices, pd.DataFrame):
        warnings.warn("prices are not in a dataframe", RuntimeWarning)
        prices = pd.DataFrame(prices)
    if returnsData:
        returns = prices
        marketReturns = marketPrices
    else:
        returns = returnsFromPrices(prices)
        if marketPrices is not None:
            marketReturns = returnsFromPrices(marketPrices)
        else:
            marketReturns = None
    # Используйте равновзвешенный набор данных в качестве прокси для рынка

```

```

if marketReturns is None:
    # Добавьте рыночную доходность справа и вычислите выборочную матрицу
    ковариаций
    returns["mkt"] = returns.mean(axis=1)
else:
    marketReturns.columns = ["mkt"]
    returns = returns.join(marketReturns, how="left")

# Вычислить ковариационную матрицу для нового фрейма данных (включая рынки)
    cov = returns.cov()
# Крайний правый столбец матрицы cov - это ковариации для рынка.
    betas = cov["mkt"] / cov.loc["mkt", "mkt"]
    betas = betas.drop("mkt")
# Найдите среднюю рыночную доходность за заданный период времени
if compounding:
    mktMeanRet = (1 + returns["mkt"]).prod() ** (
        frequency / returns["mkt"].count()
    ) - 1
else:
    mktMeanRet = returns["mkt"].mean() * frequency

# CAPM formula
return riskFreeRate + betas * (mktMeanRet - riskFreeRate)

```

Файл hierarchicalPortfolio.py

"""

Модуль ``hierarchicalPortfolio`` стремится реализовать одно из последних достижений в Оптимизация портфеля - применение моделей иерархической кластеризации при распределении.

Все иерархические классы имеют API, похожий на API EfficientFrontier, хотя, поскольку многие иерархические модели в настоящее время не поддерживают разные цели, фактическое происхождение с вызовом optimize ().

В настоящее время реализовано:

- HRPopt реализует портфель иерархического паритета рисков (HRP). Код воспроизводится с разрешение от Маркоса Лопеса де Прадо (2016) (Marcos Lopez de Prado (2016)).

"""

```

import collections
import numpy as np
import pandas as pd
import scipy.cluster.hierarchy as sch
import scipy.spatial.distance as ssd

from . import baseOptimizer, riskModels

```

```

class HRPopt(baseOptimizer.BaseOptimizer):

```

"""

Объект HRPopt (наследуемый от BaseOptimizer) создает иерархический портфель с паритетом рисков.

"""

```

def __init__(self, returns=None, covMatrix=None):

```

```

if returns is None and covMatrix is None:
    raise ValueError("Either returns or covMatrix must be provided")

if returns is not None and not isinstance(returns, pd.DataFrame):
    raise TypeError("returns are not a dataframe")

self.returns = returns
self.covMatrix = covMatrix
self.clusters = None

if returns is None:
    tickers = list(covMatrix.columns)
else:
    tickers = list(returns.columns)
super().__init__(len(tickers), tickers)

@staticmethod
def getClusterVar(cov, clusterItems):
    """
    Вычислить дисперсию на кластер
    """

    covSlice = cov.loc[clusterItems, clusterItems]
    weights = 1 / np.diag(covSlice) # Обратные веса дисперсии
    weights /= weights.sum()
    return np.linalg.multi_dot((weights, covSlice, weights))

@staticmethod
def getQuasiDiag(link):
    """
    Сортировать сгруппированные элементы по расстоянию
    """

    return sch.to_tree(link, rd=False).pre_order()

@staticmethod
def rawHrpAllocation(cov, orderedTickers):
    """
    Учитывая кластеры, вычислите портфель, который минимизирует риск за счет
    рекурсивного обхода иерархического дерева сверху.
    """

    w = pd.Series(1, index=orderedTickers)
    clusterItems = [orderedTickers] # инициализировать все элементы в одном
    кластере

    while len(clusterItems) > 0:
        clusterItems = [
            i[j:k]
            for i in clusterItems
            for j, k in ((0, len(i) // 2), (len(i) // 2, len(i)))
            if len(i) > 1
        ] # двухсекционный
        # Для каждой пары оптимизируйте локально.
        for i in range(0, len(clusterItems), 2):
            firstCluster = clusterItems[i]
            secondCluster = clusterItems[i + 1]
            # Сформируйте портфель обратной дисперсии для этой пары
            firstVariance = HRPOpt.getClusterVar(cov, firstCluster)
            secondVariance = HRPOpt.getClusterVar(cov, secondCluster)
            alpha = 1 - firstVariance / (firstVariance + secondVariance)
            w[firstCluster] *= alpha # weight 1

```

```

        w[secondCluster] *= 1 - alpha # weight 2
    return w

def optimize(self, linkageMethod="single"):
    """
    Постройте иерархический портфель с паритетом рисков, используя иерархическую
    кластеризацию Scipy.
    """
    if linkageMethod not in sch._LINKAGE_METHODS:
        raise ValueError("linkageMethod must be one recognised by scipy")

    if self.returns is None:
        cov = self.covMatrix
        corr = riskModels.covToCorr(self.covMatrix).round(6)
    else:
        corr, cov = self.returns.corr(), self.returns.cov()

    # Вычислить матрицу расстояний с исправлением ClusterWarning согласно
    https://stackoverflow.com/questions/18952587/

    # это может избежать некоторых неприятных проблем с плавающей запятой
    matrix = np.sqrt(np.clip((1.0 - corr) / 2.0, a_min=0.0, a_max=1.0))
    dist = ssd.squareform(matrix, checks=False)

    self.clusters = sch.linkage(dist, linkageMethod)
    sortIx = HRPOpt.getQuasiDiag(self.clusters)
    orderedTickers = corr.index[sortIx].tolist()
    hrp = HRPOpt.rawHrpAllocation(cov, orderedTickers)
    weights = collections.OrderedDict(hrp.sort_index())
    self.setWeights(weights)
    return weights

def portfolioPerformance(self, verbose=False, riskFreeRate=0.02, frequency=252):
    """
    После оптимизации рассчитайте (и при необходимости распечатайте)
    производительность оптимального
    портфолио. В настоящее время рассчитывает ожидаемую доходность, волатильность
    и коэффициент Шарпа.
    при условии ежедневного возврата
    """
    if self.returns is None:
        cov = self.covMatrix
        mu = None
    else:
        cov = self.returns.cov() * frequency
        mu = self.returns.mean() * frequency

    return baseOptimizer.portfolioPerformance(
        self.weights, mu, cov, verbose, riskFreeRate
    )

```

Файл objectiveFunctions.py

```

"""
Модуль objectiveFunctions обеспечивает цели оптимизации, включая фактические
целевые функции, вызываемые методами оптимизации объекта EfficientFrontier.
Эти методы в первую очередь предназначены для внутреннего использования во время
оптимизации, и для каждого из них требуется
другая подпись (поэтому они не были учтены в классе).
По очевидным причинам любая целевая функция должна принимать ``weights``.

```

в качестве аргумента, а также должен иметь по крайней мере один из вариантов «expectedReturns» или «covMatrix».

Целевые функции либо вычисляют цель с учетом множества весов, либо они вернуть `cvxpy * expression *`, когда веса являются `cvxpy.Variable`. Таким образом, та же цель функция может использоваться как внутри для оптимизации, так и извне для вычисления цели с учетом веса. `objectiveValue()` автоматически выбирает одно из двух вариантов поведения.

`` objectiveFunctions `` по умолчанию - цели для минимизации. В случаях целей которые явно должны быть максимизированы (например, коэффициент Шарпа, доходность портфеля), целевая функция фактически возвращает отрицательное количество, поскольку минимизация отрицательного значения эквивалентна максимальному положительный. Это поведение контролируется необязательным аргументом `negative = True`.

В настоящее время реализовано:

- Дисперсия портфеля (Portfolio variance) (т. Е. Квадрат волатильности)
- Возврат портфеля (Portfolio return)
- коэффициент Шарпа (Sharpe ratio)
- L2-регуляризация (L2 regularisation) (минимизация снижает ненулевые веса)
- Квадратичная полезность (Quadratic utility)
- ожидаемая (квадратная) ошибка отслеживания (Ex-ante)
- Ошибка отслеживания постфактум (в квадрате) (Ex-post)

"""

```
import numpy as np
import cvxpy as cp
```

```
def objectiveValue(w, obj):
```

"""

Вспомогательный метод для возврата значения целевой функции или целевая функция как объект `cvxpy` в зависимости от того, `w` - переменная `cvxpy` или массив `np`.

"""

```
if isinstance(w, np.ndarray):
    if np.isscalar(obj):
        return obj
    elif np.isscalar(obj.value):
        return obj.value
    else:
        return obj.value.item()
else:
    return obj
```

```
def portfolioVariance(w, covMatrix):
```

"""

Рассчитает общую дисперсию портфеля (т. Е. Квадратную волатильность).

"""

```
variance = cp.quad_form(w, covMatrix)
return objectiveValue(w, variance)
```

```
def portfolioReturn(w, expectedReturns, negative=True):
```

"""

Рассчитает (отрицательную) среднюю доходность портфеля

```

"""
sign = -1 if negative else 1
mu = w @ expectedReturns
return objectiveValue(w, sign * mu)

def sharpeRatio(w, expectedReturns, covMatrix, riskFreeRate=0.02, negative=True):
    """
    Вычисляет (отрицательный) коэффициент Шарпа портфеля
    """
    mu = w @ expectedReturns
    sigma = cp.sqrt(cp.quad_form(w, covMatrix))
    sign = -1 if negative else 1
    sharpe = (mu - riskFreeRate) / sigma
    return objectiveValue(w, sign * sharpe)

def L2Reg(w, gamma=1):
    """
    Регуляризация L2, то есть:  $\gamma \|w\|^2$ , для увеличения количества
    ненулевых весов.

    Пример::

        ef = EfficientFrontier(mu, S)
        ef.addObjective(objectiveFunctions.L2Reg, gamma = 2)
        ef.minVolatility()
    """
    L2Reg = gamma * cp.sum_squares(w)
    return objectiveValue(w, L2Reg)

def quadraticUtility(w, expectedReturns, covMatrix, riskAversion, negative=True):
    """
    Квадратичная функция полезности, например:  $\mu - \frac{1}{2} \delta w^T \Sigma w$ .
    """
    sign = -1 if negative else 1
    mu = w @ expectedReturns
    variance = cp.quad_form(w, covMatrix)

    utility = mu - 0.5 * riskAversion * variance
    return objectiveValue(w, sign * utility)

def exAnteTrackingError(w, covMatrix, benchmarkWeights):
    """
    Рассчитывает (квадрат) ожидаемой ошибки отслеживания, т. Е.
    :  $(w - w_b)^T \Sigma (w - w_b)$ .
    """
    relativeWeights = w - benchmarkWeights
    trackingError = cp.quad_form(relativeWeights, covMatrix)
    return objectiveValue(w, trackingError)

def exPostTrackingError(w, historicReturns, benchmarkReturns):
    """
    Вычисляет (квадрат) ошибки отслеживания постфактум, например:  $\text{Var}(r - r_b)$ .
    """
    if not isinstance(historicReturns, np.ndarray):
        historicReturns = np.array(historicReturns)

```

```

if not isinstance(benchmarkReturns, np.ndarray):
    benchmarkReturns = np.array(benchmarkReturns)

x_i = w @ historicReturns.T - benchmarkReturns
mean = cp.sum(x_i) / len(benchmarkReturns)
trackingError = cp.sum_squares(x_i - mean)
return objectiveValue(w, trackingError)

```

Файл plot.py

```

"""
Модуль plot содержит все функции для создания различных графиков.

В настоящее время реализовано:

- plotCovariance - построить корреляционную матрицу
- plotDendrogram - строить иерархические кластеры в портфолио
- plotEfficientFrontier - построить эффективную границу из EfficientFrontier или
объекта CriticalLineAlgorithm
- plotWeights - гистограмма весов

"""
import copy
import numpy as np
from . import riskModels, exclusion
from . import EfficientFrontier, CriticalLineAlgorithm
import scipy.cluster.hierarchy as sch
import warnings

try:
    import matplotlib.pyplot as plt

    plt.style.use("classic")
except (ModuleNotFoundError, ImportError): # pragma: no cover
    raise ImportError("Please install matplotlib via pip or poetry")

def plotIO(**kwargs):
    """
    Вспомогательный метод, позволяющий при желании сохранить фигуру в файл.
    """
    filename = kwargs.get("filename", None)
    showfig = kwargs.get("showfig", False)
    dpi = kwargs.get("dpi", 300)

    plt.tight_layout()
    if filename:
        plt.savefig(fname=filename, dpi=dpi)
    if showfig: # pragma: без обложки
        plt.show()

def plotCovariance(covMatrix, plotCorrelation=False, showTickers=True, **kwargs):
    """
    Создает базовый график ковариационной (или корреляционной) матрицы, учитывая
    ковариационную матрицу.
    """
    if plotCorrelation:
        matrix = riskModels.covToCorr(covMatrix)
    else:
        matrix = covMatrix

```

```

fig, ax = plt.subplots()

cax = ax.imshow(matrix)
fig.colorbar(cax)

if showTickers:
    ax.set_xticks(np.arange(0, matrix.shape[0], 1))
    ax.set_xticklabels(matrix.index)
    ax.set_yticks(np.arange(0, matrix.shape[0], 1))
    ax.set_yticklabels(matrix.index)
    plt.xticks(rotation=90)

plotIO(**kwargs)

return ax

def plotDendrogram(hrp, ax=None, showTickers=True, **kwargs):
    """
    Строит кластеры в виде дендрограммы.
    """
    ax = ax or plt.gca()

    if hrp.clusters is None:
        warnings.warn(
            "hrp param has not been optimized. Attempting optimization.",
            RuntimeWarning,
        )
        hrp.optimize()

    if showTickers:
        sch.dendrogram(hrp.clusters, labels=hrp.tickers, ax=ax, orientation="top")
        ax.tick_params(axis="x", rotation=90)
        plt.tight_layout()
    else:
        sch.dendrogram(hrp.clusters, no_labels=True, ax=ax)

    plotIO(**kwargs)

    return ax

def plotCla(cla, points, ax, showAssets):
    """
    Вспомогательная функция для построения эффективной границы из объекта
    CriticalLineAlgorithm
    """
    if cla.weights is None:
        cla.maxSharpe()
        optimalRet, optimalRisk, _ = cla.portfolioPerformance()

    if cla.frontierValues is None:
        cla.efficientFrontier(points=points)

    mus, sigmas, _ = cla.frontierValues

    ax.plot(sigmas, mus, label="Efficient frontier")
    ax.scatter(optimalRisk, optimalRet, marker="x", s=100, color="r", label="optimal")

    if showAssets:
        ax.scatter(
            np.sqrt(np.diag(cla.covMatrix)),

```

```

        cla.expectedReturns,
        s=30,
        color="k",
        label="assets",
    )
    return ax

```

```
def efDefaultReturnsRange(ef, points):
```

```
    """
```

```
    Вспомогательная функция для генерации диапазона доходов от GMV, возвращаемых к
    максимальной (ограниченная) доходность
    """
```

```

    efMinvol = copy.deepcopy(ef)
    efMaxret = copy.deepcopy(ef)

    efMinvol.minVolatility()
    minRet = efMinvol.portfolioPerformance()[0]
    maxRet = efMaxret.maxReturn()
    return np.linspace(minRet, maxRet - 0.0001, points)

```

```
def plotEf(ef, efParam, efParamRange, ax, showAssets):
```

```
    """
```

```
    Вспомогательная функция для построения эффективной границы из объекта
    EfficientFrontier
    """
```

```
    mus, sigmas = [], []
```

```
    # Создание портфеля для каждого значения efParamRange
```

```

    for paramValue in efParamRange:
        efI = copy.deepcopy(ef)

```

```
        try:
```

```
            if efParam == "utility":
                efI.maxQuadraticUtility(paramValue)
```

```
            elif efParam == "risk":
                efI.efficientRisk(paramValue)
```

```
            elif efParam == "return":
                efI.efficientReturn(paramValue)
```

```

            else:
                raise NotImplementedError(
                    "efParam should be one of {'utility', 'risk', 'return'}"
                )

```

```
        except exclusion.OptimizationError:
```

```
            continue
```

```
        except ValueError:
```

```

            warnings.warn(
                "Could not construct portfolio for paramter value {:.3f}".format(
                    paramValue
                )
            )

```

```

        ret, sigma, _ = efI.portfolioPerformance()
        mus.append(ret)
        sigmas.append(sigma)

```

```
ax.plot(sigmas, mus, label="Efficient frontier")
```

```
if showAssets:
```

```

    ax.scatter(
        np.sqrt(np.diag(ef.covMatrix)),

```

```

        ef.expectedReturns,
        s=30,
        color="k",
        label="assets",
    )
    return ax

def plotEfficientFrontier(
    opt,
    efParam="return",
    efParamRange=None,
    points=100,
    ax=None,
    showAssets=True,
    **kwargs
):
    """
    Построение эффективной границы на основе объекта CLA или EfficientFrontier.
    """
    ax = ax or plt.gca()

    if isinstance(opt, CriticalLineAlgorithm):
        ax = plotCla(opt, points, ax=ax, showAssets=showAssets)
    elif isinstance(opt, EfficientFrontier):
        if efParamRange is None:
            efParamRange = efDefaultReturnsRange(opt, points)

        ax = plotEf(opt, efParam, efParamRange, ax=ax, showAssets=showAssets)
    else:
        raise NotImplementedError("Please pass EfficientFrontier or
CriticalLineAlgorithm object")

    ax.legend()
    ax.set_xlabel("Volatility")
    ax.set_ylabel("Return")

    plotIO(**kwargs)
    return ax

def plotWeights(weights, ax=None, **kwargs):
    """
    plot веса портфеля в виде горизонтальной гистограммы
    """
    ax = ax or plt.gca()

    desc = sorted(weights.items(), key=lambda x: x[1], reverse=True)
    labels = [i[0] for i in desc]
    vals = [i[1] for i in desc]

    y_pos = np.arange(len(labels))

    ax.barh(y_pos, vals)
    ax.set_xlabel("Weight")
    ax.set_yticks(y_pos)
    ax.set_yticklabels(labels)
    ax.invertYaxis()

    plotIO(**kwargs)
    return ax

```

Файл riskModels.py

```

import warnings
import numpy as np
import pandas as pd
from .expectedReturns import returnsFromPrices

def isPositiveSemidefinite(matrix):
    try:
        # Significantly more efficient than checking eigenvalues
        # (stackoverflow.com/questions/16266720)
        np.linalg.cholesky(matrix + 1e-16 * np.eye(len(matrix)))
        return True
    except np.linalg.LinAlgError:
        return False

def fixNonpositiveSemidefinite(matrix, fix_method="spectral"):
    if isPositiveSemidefinite(matrix):
        return matrix

    warnings.warn(
        "The covariance matrix is non positive semidefinite. Amending eigenvalues."
    )

    # Собственное разложение
    q, V = np.linalg.eigh(matrix)

    if fix_method == "spectral":
        # Remove negative eigenvalues
        q = np.where(q > 0, q, 0)
        # Reconstruct matrix
        fixed_matrix = V @ np.diag(q) @ V.T
    elif fix_method == "diag":
        min_eig = np.min(q)
        fixed_matrix = matrix - 1.1 * min_eig * np.eye(len(matrix))
    else:
        raise NotImplementedError("Method {} not implemented".format(fix_method))

    if not isPositiveSemidefinite(fixed_matrix): # pragma: no cover
        warnings.warn(
            "Could not fix matrix. Please try a different risk model.", UserWarning
        )

    # Восстановить ярлыки, если они есть
    if isinstance(matrix, pd.DataFrame):
        tickers = matrix.index
        return pd.DataFrame(fixed_matrix, index=tickers, columns=tickers)
    else:
        return fixed_matrix

def riskMatrix(prices, method="sampleCov", **kwargs):
    if method == "sampleCov":
        return sampleCov(prices, **kwargs)
    elif method == "semiCovariance" or method == "semiVariance":
        return semiCovariance(prices, **kwargs)
    elif method == "expCov":
        return expCov(prices, **kwargs)
    elif method == "ledoitWolf" or method == "ledoit_wolf_constantVariance":
        return CovShrinkage(prices, **kwargs).ledoitWolf()
    else:

```

```

        raise NotImplementedError("Risk model {} not implemented".format(method))

def sampleCov(prices, returns_data=False, frequency=252, log_returns=False, **kwargs):

    if not isinstance(prices, pd.DataFrame):
        warnings.warn("data is not in a dataframe", RuntimeWarning)
        prices = pd.DataFrame(prices)
    if returns_data:
        returns = prices
    else:
        returns = returnsFromPrices(prices, log_returns)
    return fixNonpositiveSemidefinite(
        returns.cov() * frequency, kwargs.get("fix_method", "spectral")
    )

def
semiCovariance(prices, returns_data=False, benchmark=0.000079, frequency=252, log_returns=
False, **kwargs):
    if not isinstance(prices, pd.DataFrame):
        warnings.warn("data is not in a dataframe", RuntimeWarning)
        prices = pd.DataFrame(prices)
    if returns_data:
        returns = prices
    else:
        returns = returnsFromPrices(prices, log_returns)
    drops = np.fmin(returns - benchmark, 0)
    T = drops.shape[0]
    return fixNonpositiveSemidefinite(
        (drops.T @ drops) / T * frequency, kwargs.get("fix_method", "spectral")
    )

def pairExpCov(X, Y, span=180):
    covariation = (X - X.mean()) * (Y - Y.mean())
    # Экспоненциально взвешиваем ковариацию и берем среднее
    if span < 10:
        warnings.warn("it is recommended to use a higher span, e.g 30 days")
    return covariation.ewm(span=span).mean().iloc[-1]

def expCov(prices, returns_data=False, span=180, frequency=252, log_returns=False,
**kwargs):

    if not isinstance(prices, pd.DataFrame):
        warnings.warn("data is not in a dataframe", RuntimeWarning)
        prices = pd.DataFrame(prices)
    assets = prices.columns
    if returns_data:
        returns = prices
    else:
        returns = returnsFromPrices(prices, log_returns)
    N = len(assets)

    # Перебираем матрицу, заполняя записи попарным exp cov
    S = np.zeros((N, N))
    for i in range(N):
        for j in range(i, N):
            S[i, j] = S[j, i] = pairExpCov(
                returns.iloc[:, i], returns.iloc[:, j], span
            )
    cov = pd.DataFrame(S * frequency, columns=assets, index=assets)

    return fixNonpositiveSemidefinite(cov, kwargs.get("fix_method", "spectral"))

```

```

def covToCorr(cov_matrix):

    if not isinstance(cov_matrix, pd.DataFrame):
        warnings.warn("covMatrix is not a dataframe", RuntimeWarning)
        cov_matrix = pd.DataFrame(cov_matrix)

    Dinv = np.diag(1 / np.sqrt(np.diag(cov_matrix)))
    corr = np.dot(Dinv, np.dot(cov_matrix, Dinv))
    return pd.DataFrame(corr, index=cov_matrix.index, columns=cov_matrix.index)

def corrToCov(corr_matrix, stdevs):

    if not isinstance(corr_matrix, pd.DataFrame):
        warnings.warn("corr_matrix is not a dataframe", RuntimeWarning)
        corr_matrix = pd.DataFrame(corr_matrix)

    return corr_matrix * np.outer(stdevs, stdevs)

class CovShrinkage:

    def __init__(self, prices, returns_data=False, frequency=252, log_returns=False):

        try:
            from sklearn import covariance

            self.covariance = covariance
        except (ModuleNotFoundError, ImportError):
            raise ImportError("Please install scikit-learn via pip or poetry")

        if not isinstance(prices, pd.DataFrame):
            warnings.warn("data is not in a dataframe", RuntimeWarning)
            prices = pd.DataFrame(prices)

        self.frequency = frequency

        if returns_data:
            self.X = prices.dropna(how="all")
        else:
            self.X = returnsFromPrices(prices, log_returns).dropna(how="all")

        self.S = self.X.cov().values
        self.delta = None # shrinkage constant

    def formatAndAnnualize(self, raw_cov_array):

        assets = self.X.columns
        cov = pd.DataFrame(raw_cov_array, index=assets, columns=assets) *
self.frequency
        return fixNonpositiveSemidefinite(cov, fix_method="spectral")

    def shrunkCovariance(self, delta=0.2):

        self.delta = delta
        N = self.S.shape[1]
        # Shrinkage target
        mu = np.trace(self.S) / N
        F = np.identity(N) * mu
        # Shrinkage
        shrunkCov = delta * F + (1 - delta) * self.S
        return self.formatAndAnnualize(shrunkCov)

```

```

def ledoitWolf(self, shrinkageTarget="constantVariance"):
    """
    Рассчитывает оценку усадки Ледуа-Вольфа для конкретной
    цели усадки. """

    if shrinkageTarget == "constantVariance":
        X = np.nan_to_num(self.X.values)
        shrunkCov, self.delta = self.covariance.ledoit_wolf(X)
    elif shrinkageTarget == "singleFactor":
        shrunkCov, self.delta = self.ledoit_wolfSingleFactor()
    elif shrinkageTarget == "constantCorrelation":
        shrunkCov, self.delta = self.ledoit_wolfConstantCorrelation()
    else:
        raise NotImplementedError(
            "Shrinkage target {} not recognised".format(shrinkageTarget)
        )

    return self.formatAndAnnualize(shrunkCov)

def ledoit_wolfSingleFactor(self):

    X = np.nan_to_num(self.X.values)

    # De-mean returns
    t, n = np.shape(X)
    Xm = X - X.mean(axis=0)
    xmkt = Xm.mean(axis=1).reshape(t, 1)

    # compute sample covariance matrix
    sample = np.cov(np.append(Xm, xmkt, axis=1), rowvar=False) * (t - 1) / t
    betas = sample[0:n, n].reshape(n, 1)
    varmkt = sample[n, n]
    sample = sample[:n, :n]
    F = np.dot(betas, betas.T) / varmkt
    F[np.eye(n) == 1] = np.diag(sample)

    # compute shrinkage parameters
    c = np.linalg.norm(sample - F, "fro") ** 2
    y = Xm ** 2
    p = 1 / t * np.sum(np.dot(y.T, y)) - np.sum(sample ** 2)

    # r is divided into diagonal
    # and off-diagonal terms, and the off-diagonal term
    # is itself divided into smaller terms
    rdiag = 1 / t * np.sum(y ** 2) - sum(np.diag(sample) ** 2)
    z = Xm * np.tile(xmkt, (n,))
    v1 = 1 / t * np.dot(y.T, z) - np.tile(betas, (n,)) * sample
    roff1 = (
        np.sum(v1 * np.tile(betas, (n,)).T) / varmkt
        - np.sum(np.diag(v1) * betas.T) / varmkt
    )
    v3 = 1 / t * np.dot(z.T, z) - varmkt * sample
    roff3 = (
        np.sum(v3 * np.dot(betas, betas.T)) / varmkt ** 2
        - np.sum(np.diag(v3).reshape(-1, 1) * betas ** 2) / varmkt ** 2
    )
    roff = 2 * roff1 - roff3
    r = rdiag + roff

    # compute shrinkage constant
    k = (p - r) / c
    delta = max(0, min(1, k / t))

```

```

# compute the estimator
shrunkCov = delta * F + (1 - delta) * sample
return shrunkCov, delta

def ledoit_wolfConstantCorrelation(self):

    X = np.nan_to_num(self.X.values)
    t, n = np.shape(X)

    S = self.S # sample cov matrix

    # Constant correlation target
    var = np.diag(S).reshape(-1, 1)
    std = np.sqrt(var)
    _var = np.tile(var, (n,))
    _std = np.tile(std, (n,))
    r_bar = (np.sum(S / (_std * _std.T)) - n) / (n * (n - 1))
    F = r_bar * (_std * _std.T)
    F[np.eye(n) == 1] = var.reshape(-1)

    # Estimate pi
    Xm = X - X.mean(axis=0)
    y = Xm ** 2
    pi_mat = np.dot(y.T, y) / t - 2 * np.dot(Xm.T, Xm) * S / t + S ** 2
    pi_hat = np.sum(pi_mat)

    # Theta matrix, expanded term by term
    term1 = np.dot((Xm ** 3).T, Xm) / t
    help_ = np.dot(Xm.T, Xm) / t
    help_diag = np.diag(help_)
    term2 = np.tile(help_diag, (n, 1)).T * S
    term3 = help_ * _var
    term4 = _var * S
    theta_mat = term1 - term2 - term3 + term4
    theta_mat[np.eye(n) == 1] = np.zeros(n)
    rho_hat = sum(np.diag(pi_mat)) + r_bar * np.sum(
        np.dot((1 / std), std.T) * theta_mat
    )

    # Estimate gamma
    gamma_hat = np.linalg.norm(S - F, "fro") ** 2

    # Compute shrinkage constant
    kappa_hat = (pi_hat - rho_hat) / gamma_hat
    delta = max(0.0, min(1.0, kappa_hat / t))

    # Compute shrunk covariance matrix
    shrunkCov = delta * F + (1 - delta) * S
    return shrunkCov, delta

```

In Folder `efficient_frontier`

Файл `__init__.py`

"""

Модуль `EffectiveFrontier` содержит класс `EfficientFrontier` и его потомки, которые генерируют оптимальные портфели для различных возможных целевых функций и параметров.

"""

```

from .efficientFrontier import EfficientFrontier
from .efficientCvar import EfficientCVaR
from .efficientSemivariance import EfficientSemivariance
from .efficientCdar import EfficientCDaR

```

```

__all__ = [
    "EfficientFrontier",
    "EfficientCVaR",
    "EfficientSemivariance",
    "EfficientCDaR",
]

```

Файл `efficientCdar.py`

```

"""

```

Подмодуль ``effectiveCdar`` содержит класс EfficientCDaR, который формирует портфели вдоль границы среднего CDaR (условной просадки под угрозой).

```

"""

```

```

import warnings
import numpy as np
import cvxpy as cp

```

```

from .. import objectiveFunctions
from .efficientFrontier import EfficientFrontier

```

```

class EfficientCDaR(EfficientFrontier):

```

```

    """

```

Класс EfficientCDaR позволяет оптимизировать по границе среднего CDaR, используя формулировка Chekhlov, Ursayev and Zabarankin (2005).

```

- ``minCdar()`` минимизирует CDaR
- ``efficientRisk()`` минимизирует CDaR для заданной целевой доходности
- ``efficientReturn()`` minimises CDaR for a given target return
- ``addObjective()`` добавляет (выпуклую) цель к задаче оптимизации
- ``addConstraint()`` добавляет (линейное) ограничение к задаче оптимизации

- ``portfolioPerformance()`` рассчитывает ожидаемую доходность и CDaR портфеля.
- ``setWeights()`` создает self.weights (np.ndarray) из диктата весов
- ``cleanWeights()`` округляет веса и обрезает почти до нуля.
- ``saveWeightsToFile()`` сохраняет веса в csv, json или txt.
"""

```

```

def __init__(
    self,
    expectedReturns,
    returns,
    beta=0.95,
    weightBounds=(0, 1),
    solver=None,
    verbose=False,
    solverOptions=None,
):
    super().__init__(
        expectedReturns=expectedReturns,
        covMatrix=np.zeros((len(expectedReturns),) * 2), # dummy
        weightBounds=weightBounds,
        solver=solver,
        verbose=verbose,
        solverOptions=solverOptions,
    )

```

```

)

self.returns = self.validateReturns(returns)
self.beta = self.validateBeta(beta)
self.alpha = cp.Variable()
self.u = cp.Variable(len(self.returns) + 1)
self.z = cp.Variable(len(self.returns))

def validateBeta(self, beta):
    if not (0 <= beta < 1):
        raise ValueError("beta must be between 0 and 1")
    if beta <= 0.2:
        warnings.warn(
            "Warning: beta is the confidence-level, not the quantile. Typical
values are 80%, 90%, 95%.",
            UserWarning,
        )
    return beta

def minVolatility(self):
    raise NotImplementedError("Please use minCdar instead.")

def maxSharpe(self, riskFreeRate=0.02):
    raise NotImplementedError("Method not available in EfficientCDaR.")

def maxQuadraticUtility(self, riskAversion=1, marketNeutral=False):
    raise NotImplementedError("Method not available in EfficientCDaR.")

def minCdar(self, marketNeutral=False):
    """
    Сведите к минимуму CDaR портфеля (дополнительные пояснения см. В
документации).
    """
    self.objective = self.alpha + 1.0 / (
        len(self.returns) * (1 - self.beta)
    ) * cp.sum(self.z)

    for obj in self.additionalObjectives:
        self.objective += obj

    self.constraints += [
        self.z >= self.u[1:] - self.alpha,
        self.u[1:] >= self.u[:-1] - self.returns.values @ self.w,
        self.u[0] == 0,
        self.z >= 0,
        self.u[1:] >= 0,
    ]

    self.makeWeightSumConstraint(marketNeutral)
    return self.solveCvxpyOptProblem()

def efficientReturn(self, targetReturn, marketNeutral=False):
    """
    Минимизировать CDaR для заданной целевой доходности.
    """
    self.objective = self.alpha + 1.0 / (
        len(self.returns) * (1 - self.beta)
    ) * cp.sum(self.z)

    for obj in self.additionalObjectives:
        self.objective += obj

```

```

self.constraints += [
    self.z >= self.u[1:] - self.alpha,
    self.u[1:] >= self.u[:-1] - self.returns.values @ self.w,
    self.u[0] == 0,
    self.z >= 0,
    self.u[1:] >= 0,
]

ret = self.expectedReturns.T @ self.w
self.constraints.append(ret >= targetReturn)

self.makeWeightSumConstraint(marketNeutral)
return self.solveCvxpyOptProblem()

def efficientRisk(self, targetCdar, marketNeutral=False):
    """
    Увеличьте отдачу от целевого CDaR.
    Итоговый портфель будет иметь CDaR меньше целевого.
    (но не гарантируется равное).
    """
    self.objective = objectiveFunctions.portfolioReturn(
        self.w, self.expectedReturns
    )
    for obj in self.additionalObjectives:
        self.objective += obj

    cdar = self.alpha + 1.0 / (len(self.returns) * (1 - self.beta)) * cp.sum(
        self.z
    )
    self.constraints += [
        cdar <= targetCdar,
        self.z >= self.u[1:] - self.alpha,
        self.u[1:] >= self.u[:-1] - self.returns.values @ self.w,
        self.u[0] == 0,
        self.z >= 0,
        self.u[1:] >= 0,
    ]

    self.makeWeightSumConstraint(marketNeutral)
    return self.solveCvxpyOptProblem()

def portfolioPerformance(self, verbose=False):
    """
    После оптимизации рассчитайте (и при необходимости распечатайте)
    производительность оптимального
    портфель, а именно: ожидаемая доходность, CDaR
    """
    mu = objectiveFunctions.portfolioReturn(
        self.weights, self.expectedReturns, negative=False
    )

    cdar = self.alpha + 1.0 / (len(self.returns) * (1 - self.beta)) * cp.sum(
        self.z
    )
    cdarVal = cdar.value

    if verbose:
        print("Expected annual return: {:.1f}%".format(100 * mu))
        print("Conditional Drawdown at Risk: {:.2f}%".format(100 * cdarVal))

    return mu, cdarVal

```

Файл `efficientCvar.py`

"""

Подмодуль ``effectiveCvar`` содержит класс `EfficientCVaR`, который формирует портфели вдоль границы среднего CVaR.

"""

```
import warnings
import numpy as np
import cvxpy as cp

from .. import objectiveFunctions
from .efficientFrontier import EfficientFrontier
```

```
class EfficientCVaR(EfficientFrontier):
```

"""

Класс `EfficientCVaR` позволяет оптимизировать по границе среднего значения CVaR, используя

формулировка Rockafellar and Ursayev (2001).

Methods:

```
- ``minCvar()`` минимизирует CVaR
- ``efficientRisk()`` максимизирует доход для заданного CVaR
- ``efficientReturn()`` минимизирует CVaR для заданной целевой доходности
- ``addObjective()`` добавляет (выпуклую) цель к задаче оптимизации
- ``addConstraint()`` добавляет ограничение к проблеме оптимизации

- ``portfolioPerformance()`` вычисляет ожидаемую доходность и CVaR портфеля.
- ``setWeights()`` создает self.weights (np.ndarray) из диктата весов
- ``cleanWeights()`` округляет веса и обрезает почти до нуля.
- ``saveWeightsToFile()`` сохраняет веса в csv, json или txt.
"""
```

```
def __init__(
    self,
    expectedReturns,
    returns,
    beta=0.95,
    weightBounds=(0, 1),
    solver=None,
    verbose=False,
    solverOptions=None,
):
    super().__init__(
        expectedReturns=expectedReturns,
        covMatrix=np.zeros((returns.shape[1],) * 2), # dummy
        weightBounds=weightBounds,
        solver=solver,
        verbose=verbose,
        solverOptions=solverOptions,
    )

    self.returns = self.validateReturns(returns)
    self.beta = self.validateBeta(beta)
    self.alpha = cp.Variable()
    self.u = cp.Variable(len(self.returns))

    @staticmethod
    def validateBeta(beta):
        if not (0 <= beta < 1):
```

```

        raise ValueError("beta must be between 0 and 1")
    if beta <= 0.2:
        warnings.warn(
            "Warning: beta is the confidence-level, not the quantile. Typical
values are 80%, 90%, 95%.",
            UserWarning,
        )
    return beta

def minVolatility(self):
    raise NotImplementedError("Please use minCvar instead.")

def maxSharpe(self, riskFreeRate=0.02):
    raise NotImplementedError("Method not available in EfficientCVar.")

def maxQuadraticUtility(self, riskAversion=1, marketNeutral=False):
    raise NotImplementedError("Method not available in EfficientCVar.")

def minCvar(self, marketNeutral=False):
    """
    Сведите к минимуму CVaR портфеля (подробные объяснения в дипломе).
    """
    self.objective = self.alpha + 1.0 / (
        len(self.returns) * (1 - self.beta)
    ) * cp.sum(self.u)

    for obj in self.additionalObjectives:
        self.objective += obj

    self.constraints += [
        self.u >= 0.0,
        self.returns.values @ self.w + self.alpha + self.u >= 0.0,
    ]

    self.makeWeightSumConstraint(marketNeutral)
    return self.solveCvxpyOptProblem()

def efficientReturn(self, targetReturn, marketNeutral=False):
    """
    Минимизируйте CVaR для заданной целевой доходности.
    """
    self.objective = self.alpha + 1.0 / (
        len(self.returns) * (1 - self.beta)
    ) * cp.sum(self.u)

    for obj in self.additionalObjectives:
        self.objective += obj

    self.constraints += [
        self.u >= 0.0,
        self.returns.values @ self.w + self.alpha + self.u >= 0.0,
    ]

    ret = self.expectedReturns.T @ self.w
    self.constraints.append(ret >= targetReturn)

    self.makeWeightSumConstraint(marketNeutral)
    return self.solveCvxpyOptProblem()

def efficientRisk(self, targetCvar, marketNeutral=False):
    """
    Увеличивает отдачу от целевой CVaR.
    """

```

```

        Результирующий портфель будет иметь CVaR меньше целевого.
        (но не гарантируется равное).
        """
        self.objective = objectiveFunctions.portfolioReturn(
            self.w, self.expectedReturns
        )
        for obj in self.additionalObjectives:
            self.objective += obj

        cvar = self.alpha + 1.0 / (len(self.returns) * (1 - self.beta)) * cp.sum(
            self.u
        )
        self.constraints += [
            cvar <= targetCvar,
            self.u >= 0.0,
            self.returns.values @ self.w + self.alpha + self.u >= 0.0,
        ]

        self.makeWeightSumConstraint(marketNeutral)
        return self.solveCvxpyOptProblem()

    def portfolioPerformance(self, verbose=False):
        """
        После оптимизации рассчитайте (и при необходимости распечатайте)
        производительность оптимального
        портфель, а именно: ожидаемая доходность, CVaR
        """
        mu = objectiveFunctions.portfolioReturn(
            self.weights, self.expectedReturns, negative=False
        )

        cvar = self.alpha + 1.0 / (len(self.returns) * (1 - self.beta)) * cp.sum(
            self.u
        )
        cvarVal = cvar.value

        if verbose:
            print("Expected annual return: {:.1f}%".format(100 * mu))
            print("Conditional Value at Risk: {:.2f}%".format(100 * cvarVal))

        return mu, cvarVal

```

Файл `efficientFrontier.py`

```

"""
Подмодуль ``effectiveFrontier`` содержит класс EfficientFrontier, который генерирует
классические оптимальные портфели по средней дисперсии для множества целей и
ограничений
"""

```

```

import warnings
import numpy as np
import pandas as pd
import cvxpy as cp

from .. import objectiveFunctions, baseOptimizer

class EfficientFrontier(baseOptimizer.BaseConvexOptimizer):
    """
    Объект EfficientFrontier (наследуемый от BaseConvexOptimizer) содержит несколько

```

методов оптимизации, которые могут быть вызваны (соответствующие разным целям функции) с различными параметрами.

- `minVolatility()` оптимизирует для минимальной волатильности
- `maxSharpe()` оптимизирует максимальный коэффициент Шарпа (также известный как портфель касания)
- `maxQuadraticUtility()` максимизирует квадратичную полезность при некотором избегании риска.
- `efficientRisk()` максимизирует прибыль для заданного целевого риска
- `efficientReturn()` минимизирует риск для заданной целевой доходности
- `addObjective()` добавляет (выпуклую) цель к задаче оптимизации
- `addConstraint()` добавляет ограничение к проблеме оптимизации
- `convexObjective()` решает для общей выпуклой цели с линейными ограничениями
- `portfolioPerformance()` рассчитывает ожидаемую доходность, волатильность и коэффициент Шарпа для оптимизированное портфолио.
- `setWeights()` создает `self.weights` (`np.ndarray`) из `dict weights`
- `cleanWeights()` округляет веса и обрезает почти до нуля.
- `saveWeightsToFile()` сохраняет веса в `csv`, `json` или `txt`.

```
def __init__(
    self,
    expectedReturns,
    covMatrix,
    weightBounds=(0, 1),
    solver=None,
    verbose=False,
    solverOptions=None,
):
    # Входы
    self.covMatrix = EfficientFrontier.validateCovMatrix(covMatrix)
    self.expectedReturns = EfficientFrontier.validateExpectedReturns(
        expectedReturns
    )

    if self.expectedReturns is None:
        numAssets = len(covMatrix)
    else:
        numAssets = len(expectedReturns)

    # Лейблы
    if isinstance(expectedReturns, pd.Series):
        tickers = list(expectedReturns.index)
    elif isinstance(covMatrix, pd.DataFrame):
        tickers = list(covMatrix.columns)
    else: # использовать целочисленные метки
        tickers = list(range(numAssets))

    if expectedReturns is not None and covMatrix is not None:
        if covMatrix.shape != (numAssets, numAssets):
            raise ValueError("Covariance matrix does not match expected returns")

    super().__init__(
        len(tickers),
        tickers,
        weightBounds,
        solver=solver,
        verbose=verbose,
```

```

        solverOptions=solverOptions,
    )

    @staticmethod
    def validateExpectedReturns(expectedReturns):
        if expectedReturns is None:
            return None
        elif isinstance(expectedReturns, pd.Series):
            return expectedReturns.values
        elif isinstance(expectedReturns, list):
            return np.array(expectedReturns)
        elif isinstance(expectedReturns, np.ndarray):
            return expectedReturns.ravel()
        else:
            raise TypeError("expectedReturns is not a series, list or array")

    @staticmethod
    def validateCovMatrix(covMatrix):
        if covMatrix is None:
            raise ValueError("covMatrix must be provided")
        elif isinstance(covMatrix, pd.DataFrame):
            return covMatrix.values
        elif isinstance(covMatrix, np.ndarray):
            return covMatrix
        else:
            raise TypeError("covMatrix is not a dataframe or array")

    def validateReturns(self, returns):
        """
        Вспомогательный метод для проверки ежедневных доходов (необходим для некоторых
        эффективных границ)
        """
        if not isinstance(returns, (pd.DataFrame, np.ndarray)):
            raise TypeError("returns should be a pd.DataFrame or np.ndarray")

        returnsDf = pd.DataFrame(returns)
        if returnsDf.isnull().values.any():
            warnings.warn(
                "Removing NaNs from returns",
                UserWarning,
            )
            returnsDf = returnsDf.dropna(axis=0, how="any")

        if self.expectedReturns is not None:
            if returnsDf.shape[1] != len(self.expectedReturns):
                raise ValueError(
                    "returns columns do not match expectedReturns. Please check your
                    tickers."
                )

        return returnsDf

    def makeWeightSumConstraint(self, isMarketNeutral):
        """
        Вспомогательный метод для ограничения суммы веса. Если рынок нейтрален,
        проверьте веса, указанные в конструкторе.
        """
        if isMarketNeutral:
            # Check and fix bounds
            portfolioPossible = np.any(self.lowerBounds < 0)
            if not portfolioPossible:
                warnings.warn(

```

```

        "Market neutrality requires shorting - bounds have been amended",
        RuntimeError,
    )
    self.mapBoundsToConstraints((-1, 1))
    # Delete original constraints
    del self.constraints[0]
    del self.constraints[0]

    self.constraints.append(cp.sum(self.w) == 0)
else:
    self.constraints.append(cp.sum(self.w) == 1)

def minVolatility(self):
    """
    Минимизирует волатильность.
    """
    self.objective = objectiveFunctions.portfolioVariance(
        self.w, self.covMatrix
    )
    for obj in self.additionalObjectives:
        self.objective += obj

    self.constraints.append(cp.sum(self.w) == 1)
    return self.solveCvxpyOptProblem()

def maxReturn(self, return_value=True):
    """
    Вспомогательный метод для максимального дохода. Это не должно использоваться
    для оптимизации портфеля.
    """
    self.objective = objectiveFunctions.portfolioReturn(
        self.w, self.expectedReturns
    )

    self.constraints.append(cp.sum(self.w) == 1)

    res = self.solveCvxpyOptProblem()

    # Ограничения очистки, так как это вспомогательный метод
    del self.constraints[-1]

    if return_value:
        return -self.opt.value
    else:
        return res

def maxSharpe(self, riskFreeRate=0.02):
    """
    Максимизирует коэффициент Шарпа. Результат также называется портфелем касания,
    поскольку это портфель, для которого линия рынка капитала касается границы
    эффективности.

    Это задача выпуклой оптимизации после подстановки определенной переменной.
    """
    if not isinstance(riskFreeRate, (int, float)):
        raise ValueError("riskFreeRate should be numeric")
    self.riskFreeRate = riskFreeRate

    # maxSharpe требует от нас преобразования переменной.
    # Здесь мы рассматриваем w как преобразованную переменную.
    self.objective = cp.quad_form(self.w, self.covMatrix)
    k = cp.Variable()

```

```

# Примечание: цели не масштабируются на k. Следовательно, есть тонкие различия
# между тем, как эти цели работают для maxSharpe и minVolatility
if len(self.additionalObjectives) > 0:
    warnings.warn(
        "maxSharpe transforms the optimization problem so additional
objectives may not work as expected."
    )
for obj in self.additionalObjectives:
    self.objective += obj

newConstraints = []
# Необходимо перестроить ограничения
for constr in self.constraints:
    if isinstance(constr, cp.constraints.nonpos.Inequality):
        # Либо первый, либо второй элемент - это выражение
        if isinstance(
            constr.args[0], cp.expressions.constants.constant.Constant
        ):
            newConstraints.append(constr.args[1] >= constr.args[0] * k)
        else:
            newConstraints.append(constr.args[0] <= constr.args[1] * k)
    elif isinstance(constr, cp.constraints.zero.Equality):
        newConstraints.append(constr.args[0] == constr.args[1] * k)
    else:
        raise TypeError(
            "Please check that your constraints are in a suitable format"
        )

# Преобразовал выпуклую задачу maxSharpe:
self.constraints = [
    (self.expectedReturns - riskFreeRate).T @ self.w == 1,
    cp.sum(self.w) == k,
    k >= 0,
] + newConstraints

self.solveCvxpyOptProblem()
# Обратное преобразование
self.weights = (self.w.value / k.value).round(16) + 0.0
return self.makeOutputWeights()

def maxQuadraticUtility(self, riskAversion=1, marketNeutral=False):
    """
    Максимизируйте данную квадратичную полезность, то есть:
     $\max_w w^T \mu - \frac{\Delta^2 w^T \Sigma w}{2}$ 
    """
    if riskAversion <= 0:
        raise ValueError("risk aversion coefficient must be greater than zero")

    self.objective = objectiveFunctions.quadraticUtility(
        self.w, self.expectedReturns, self.covMatrix, riskAversion=riskAversion
    )
    for obj in self.additionalObjectives:
        self.objective += obj

    self.makeWeightSumConstraint(marketNeutral)
    return self.solveCvxpyOptProblem()

def efficientRisk(self, targetVolatility, marketNeutral=False):
    """
    Увеличьте отдачу от целевого риска. Полученный портфель будет иметь
    волатильность.

```

```

        меньше целевого (но не гарантированно равным).
        """
        if not isinstance(targetVolatility, (float, int)) or targetVolatility < 0:
            raise ValueError("targetVolatility should be a positive float")

        globalMinVolatility = np.sqrt(1 / np.sum(np.linalg.inv(self.covMatrix)))

        if targetVolatility < globalMinVolatility:
            raise ValueError(
                "The minimum volatility is {:.3f}. Please use a higher
targetVolatility".format(
                    globalMinVolatility
                )
            )

        self.objective = objectiveFunctions.portfolioReturn(
            self.w, self.expectedReturns
        )
        variance = objectiveFunctions.portfolioVariance(self.w, self.covMatrix)

        for obj in self.additionalObjectives:
            self.objective += obj

        self.constraints.append(variance <= targetVolatility ** 2)
        self.makeWeightSumConstraint(marketNeutral)
        return self.solveCvxpyOptProblem()

    def efficientReturn(self, targetReturn, marketNeutral=False):
        """
        Рассчитайте «портфель Марковица», минимизируя волатильность для заданной
целевой доходности.
        """
        if not isinstance(targetReturn, float) or targetReturn < 0:
            raise ValueError("targetReturn should be a positive float")
        if targetReturn > self.maxReturn():
            raise ValueError(
                "targetReturn must be lower than the maximum possible return"
            )

        self.objective = objectiveFunctions.portfolioVariance(
            self.w, self.covMatrix
        )
        ret = objectiveFunctions.portfolioReturn(
            self.w, self.expectedReturns, negative=False
        )

        for obj in self.additionalObjectives:
            self.objective += obj

        self.constraints.append(ret >= targetReturn)
        self.makeWeightSumConstraint(marketNeutral)
        return self.solveCvxpyOptProblem()

    def portfolioPerformance(self, verbose=False, riskFreeRate=0.02):
        """
        После оптимизации рассчитайте (и при необходимости распечатайте)
производительность оптимального
портфолио. В настоящее время рассчитывает ожидаемую доходность, волатильность
и коэффициент Шарпа.
        """
        if self.riskFreeRate is not None:
            if riskFreeRate != self.riskFreeRate:

```

```

        warnings.warn(
            "The riskFreeRate provided to portfolioPerformance is different"
            " to the one used by maxSharpe. Using the previous value.",
            UserWarning,
        )
        riskFreeRate = self.riskFreeRate

    return baseOptimizer.portfolioPerformance(
        self.weights,
        self.expectedReturns,
        self.covMatrix,
        verbose,
        riskFreeRate,
    )

```

Файл `efficientSemivariance.py`
 """

Подмодуль ``effectiveSemivariance`` содержит класс `EfficientSemivariance`, который генерирует портфели вдоль границы средней полувариантности.

"""

```

import numpy as np
import cvxpy as cp

```

```

from .. import objectiveFunctions
from .efficientFrontier import EfficientFrontier

```

```

class EfficientSemivariance(EfficientFrontier):

```

```

    """
    Объекты EfficientSemivariance допускают оптимизацию по границе средней
    полувариантности.
    Это может быть актуально для пользователей, которых больше беспокоит обратное
    отклонение.

    - ``minSemivariance()`` минимизирует полувариантность портфеля (отклонение в
    сторону понижения)
    - ``maxQuadraticUtility()`` максимизирует "квадратичную полезность обратной
    стороны" при некотором избегании риска.
    - ``efficientRisk()`` максимизирует доход для заданного целевого полуотклонения
    - ``efficientReturn()`` минимизирует полуотклонение для заданной целевой
    доходности
    - ``addObjective()`` добавляет (выпуклую) цель к задаче оптимизации
    - ``addConstraint()`` добавляет ограничение к проблеме оптимизации
    - ``convexObjective()`` решает для общей выпуклой цели с линейными ограничениями

    - ``portfolioPerformance()`` рассчитывает ожидаемую доходность, полуотклонение и
    коэффициент Сортино для
    оптимизированное портфолио.
    - ``setWeights()`` создает self.weights (np.ndarray) из диктата весов
    - ``cleanWeights()`` округляет веса и обрезаает почти до нуля.
    - ``saveWeightsToFile()`` сохраняет веса в csv, json или txt.
    """

```

```

def __init__(
    self,
    expectedReturns,
    returns,
    frequency=252,
    benchmark=0,
    weightBounds=(0, 1),
    solver=None,

```

```

        verbose=False,
        solverOptions=None,
    ):
        # Создание экземпляра родителя
        super().__init__(
            expectedReturns=expectedReturns,
            covMatrix=np.zeros((returns.shape[1],) * 2),
            weightBounds=weightBounds,
            solver=solver,
            verbose=verbose,
            solverOptions=solverOptions,
        )

        self.returns = self.validateReturns(returns)
        self.benchmark = benchmark
        self.frequency = frequency
        self.T = self.returns.shape[0]

    def minVolatility(self):
        raise NotImplementedError("Please use minSemivariance instead.")

    def maxSharpe(self, riskFreeRate=0.02):
        raise NotImplementedError("Method not available in EfficientSemivariance")

    def efficientRisk(self, targetSemideviation, marketNeutral=False):
        """
        Увеличьте доходность от целевого полуотклонения (нижнее стандартное
        отклонение).
        Полученный портфель будет иметь полуотклонение меньше целевого.
        (но не гарантируется равное).
        """
        self.objective = objectiveFunctions.portfolioReturn(
            self.w, self.expectedReturns
        )
        for obj in self.additionalObjectives:
            self.objective += obj

        p = cp.Variable(self.T, nonneg=True)
        n = cp.Variable(self.T, nonneg=True)

        self.constraints.append(
            self.frequency * cp.sum(cp.square(n)) <= (targetSemideviation ** 2)
        )
        B = (self.returns.values - self.benchmark) / np.sqrt(self.T)
        self.constraints.append(B @ self.w - p + n == 0)
        self.makeWeightSumConstraint(marketNeutral)
        return self.solveCvxpyOptProblem()

    def efficientReturn(self, targetReturn, marketNeutral=False):
        """
        Сведите к минимуму полуотклонение для заданной целевой доходности.
        """
        if not isinstance(targetReturn, float) or targetReturn < 0:
            raise ValueError("targetReturn should be a positive float")
        if targetReturn > np.abs(self.expectedReturns).max():
            raise ValueError(
                "targetReturn must be lower than the largest expected return"
            )

        p = cp.Variable(self.T, nonneg=True)
        n = cp.Variable(self.T, nonneg=True)
        self.objective = cp.sum(cp.square(n))
        for obj in self.additionalObjectives:

```

```

        self.objective += obj

    self.constraints.append(
        cp.sum(self.w @ self.expectedReturns) >= targetReturn
    )
    B = (self.returns.values - self.benchmark) / np.sqrt(self.T)
    self.constraints.append(B @ self.w - p + n == 0)
    self.makeWeightSumConstraint(marketNeutral)
    return self.solveCvxpyOptProblem()

def portfolioPerformance(self, verbose=False, riskFreeRate=0.02):
    """
    После оптимизации рассчитайте (и при необходимости распечатайте)
    производительность оптимального
    портфель, а именно: ожидаемая доходность, полуоплата, коэффициент Сортино
    (expected return, semideviation, Sortino ratio.)
    """
    mu = objectiveFunctions.portfolioReturn(
        self.weights, self.expectedReturns, negative=False
    )

    portfolioReturns = self.returns @ self.weights
    drops = np.fmin(portfolioReturns - self.benchmark, 0)
    semivariance = np.sum(np.square(drops)) / self.T * self.frequency
    semiDeviation = np.sqrt(semivariance)
    sortinoRatio = (mu - riskFreeRate) / semiDeviation

    if verbose:
        print("Expected annual return: {:.1f}%".format(100 * mu))
        print("Annual semi-deviation: {:.1f}%".format(100 * semiDeviation))
        print("Sortino Ratio: {:.2f}".format(sortinoRatio))

    return mu, semiDeviation, sortinoRatio

```