Финансовый инжиниринг

2022/23 учебный год

Инженерный тур

Общая информация

Участникам командного практического тура предлагается разработать полностью завершённое web-приложение на языке программирования Python для построения оптимального портфеля ценных бумаг по модели вложения инвестиций американского экономиста, лауреата Нобелевской премии Гарри Марковица.

Сюжет задачи

Финансовый рынок — сложная система, требующая для грамотного использования всех его возможностей принятия множества решений, — от определения оптимальной пропорции долей распределения вкладываемых сумм между доступным набором фондовых активов до хеджирования портфельных инвестиционных рисков. В условиях пандемийных нагрузок растёт интерес российских граждан к диверсификации своих накоплений, в том числе с использованием инвестиционных web-приложений. Инвестиционные приложения дают возможность пользователям самостоятельно без брокеров определять стратегию и свободно выбирать инструменты инвестиций. В этой связи перспективным является разработка приложений, которые позволяют построить вероятностные распределения к наблюдаемым данным о доходностях предполагаемого инвестиционного портфеля и определить наиболее целесообразное направление инвестиционного вложения. Такие web-приложения могут быть очень эффективны в поиске вероятностных связей между биржевыми тикерами и прогнозами векторов движения котировок акций.

Требования к команде и компетенциям участников

Количество участников в команде: 3–4.

- Инженер-программист и одновременно проджект-менеджер, реализующий парсинг данных, их предобработку, математическую модель Гарри Марковица для формирования инвестиционного портфеля и контролирующий весь проект.
- Инженер-программист, выполняющий backend-разработку web-приложения, в том числе его БД.
- Инженер-программист, выполняющий frontend-разработку интерфейса web-приложения.
- DevOps-инженер, реализующий deploy приложения.

Для успешного выполнения командного тура участники команды должны обладать следующими компетенциями:

- базовый уровень знания ЯП Python;
- понимание процесса парсинга различных данных с нескольких сайтов в режиме реального времени;

- наличие умений работы с БД PostgreSQL;
- понимание процесса предобработки и очистки данных и наличие умений работы с библиотекой ЯП Python `pandas`;
- понимание процесса frontend-разработки web-приложения и наличие умений работы с фреймворком Streamlit;
- понимание процесса backend-разработки web-приложения и наличие умений, навыков работы с фреймворками ЯП Python `NumPy`, `SymPy`, `SciPy`;
- понимание процесса внешнего развёртывания (deploy) разработанного web-приложения, наличие умений, навыков работы с контейнером Docker и репозиторием GitHub.

Оборудование и программное обеспечение

Для решения командной инженерной задачи участникам необходим компьютерный класс со следующими минимальными характеристиками. Компьютеры:

- оперативная память не менее 4 Gb;
- процессор не менее чем на 2 ядра и частотой не менее 1,6 Ghz;
- объём памяти видеокарты не менее 512 Mb;
- жёсткий диск любого объема (интерфейс Sata);
- наличие Usb-разъема на лицевой стороне системного блока (в верхней части);
- мышь и клавиатура должна быть Usb;
- диагональ монитора не менее 17 дюймов, жидкокристаллическая.

Системное ПО: ОС Windows 10.0, пакет MS Office, антивирус Касперского, программы записи видео с экрана, браузер: Google Chrome.

Для разработки web-приложения необходимо наличие следующего ПО:

Наименование	**Описание**
PyCharm IDE https://www.jetbrains.com/pycharm/download	Pycharm — интерактивная среда компании Jetbrains для написания кода на ЯП Python IDE, позволяющая программировать в наиболее комфортной среде разработчика
Jupyter Notebook 2 https://jupyter.org/install	Jupyter Notebook — интерактивная среда, которая может быть использована как альтернативная для работы с ЯП Python
Pip Package 3 https://pip.pypa.io/en/stable/cli/pip_download/	PIP используется для быстрой установки любого пакета Python с лёгкой функцией поиска
БД PostgreSQL https://www.postgresql.org/download/	Используется для хранения данных до и после предобработки и очистки
Фреймворк Streamlit https://pypi.org/project/streamlit/	Используется для frontend-разработки web-приложения

Наименование	Описание
Библиотека ЯП Python `pandas` https://pandas.pydata.org/	Используется для предобработки и очистки данных
Фреймворки ЯП Python `NumPy`, `MathPlotLib` https://numpy.org/install/ https://pypi.org/project/matplotlib/	Используются для выполнения вычислений над матрицами данных и визуализации полученных результатов
Фреймворки ЯП Python `SymPy`, `SciPy` https://pypi.org/project/sympy/ https://scipy.org/install/	Для решения вычислительных задач по модели Гарри Марковица
Фреймворк Docker https://www.docker.com/products/docker-desktop/	Используется на этапе deploy приложения
Репозиторий GitHub	Используется для публикации кода и поддержки контроля версий

Описание задачи

Разработать web-приложение на ЯП Python для построения оптимального портфеля ценных бумаг по модели Гарри Марковица.

С этой целью выполнить следующие этапы, **каждый из которых должен быть реализован функционально в виде отдельной подпрограммы или модуля**.

Этап 1

Разработать парсер для записи исторических — за 1 год — биржевых тикеров с web-ресурса, а именно акции 1 уровня с Московская Биржа/Биржевая информация (moex.com) в формате `.csv` в созданную пользовательскую базу данных PostgreSQL. База данных PostgreSQL должна поддерживать регистрацию и авторизацию пользователей, а также обеспечивать механизм ввода/вывода актуальных тикеров биржевых данных.

Этап 2

Произвести, используя библиотеку ЯП Python `pandas`, предобработку полученных в процессе парсинга с сайта финансовых данных: заполнение пропусков, коррекция выбросов, сохранить обработанные данные в БД PostgreSQL.

Этап 3

Используя фреймворк Streamlit, произвести frontend-разработку web-приложения в виде аналога робо-консультанта с возможностью априорного определения критериев парсинга и визуализации выбранных с сайта данных.

Этап 4

Реализовать backend-функции web-приложения в виде построения оптимального инвестиционного портфеля акций фондового рынка по модели Гарри Марковица, используя фреймворки ЯП Python `NumPy`, `SymPy`, `SciPy`.

Этап 5

Выполнить deploy разработанного web-приложения, упаковав его в контейнер Docker и сделать публикацию проекта в репозитории GitHub.

Система оценивания

Сдаваемый для оценки экспертным жюри полностью завершённый проект должен содержать 7 (семь) артефактов в виде электронных файловых документов:

1. Листинг программы парсера данных.
2. Листинг программы для создания пользовательской базы данных PostgreSQL, регистрации и авторизации пользователей, обеспечения механизма ввода/вывода.
3. Файл с выводом актуальных тикеров биржевых данных из БД PostgreSQL после их предобработки и очистки.
4. Листинг программы, реализующей интерфейс взаимодействия с пользователем web-приложения в виде робо-консультанта, с возможностью определения критериев парсинга и визуализации выбранных с сайта данных.
5. Листинг программы для построения оптимального инвестиционного портфеля акций фондового рынка по модели Гарри Марковица.
6. Листинг программы, обеспечивающей deploy разработанного web-приложения через контейнер Docker с прилагаемой в листинге ссылкой на публикацию проекта в репозитории GitHub.
7. PowerPoint-презентация проекта, включающая в себя 5 (пять) слайдов:
   - 7.1. название web-приложения и состав команды с расшифровкой командных ролей;
   - 7.2. перечень блоков web-приложения, которые полностью были реализованы командой и блоков, которые команда не успела реализовать;
   - 7.3. визуализацию работы интерфейса взаимодействия с пользователем web-приложения в виде робо-консультанта;
   - 7.4. визуализацию построения оптимального инвестиционного портфеля акций фондового рынка по модели Гарри Марковица;
   - 7.5. результаты deploy разработанного web-приложения с указанием ссылки на публикацию проекта в репозитории GitHub.

Команда-победитель определяется экспертным жюри по качеству реализации обозначенных этапов финальной задачи (см. Описание задачи), а также наличию в сданном проекте указанных выше артефактов. В частности, из 100 баллов присуждается:

- 0–15 баллов — за разработку парсера, а также за эффективность парсинга данных;
- 0–10 баллов — за качественную предобработку и очистку данных;
- 0–15 баллов — за разработку пользовательской базы данных PostgreSQL с возможностью регистрации и авторизации пользователей, обеспечения механизма ввода/вывода данных;
- 0–15 баллов — за frontend-разработку web-приложения, в том числе за реализацию интерфейса робо-консультанта;
- 0–15 баллов — за корректную реализацию математической модели Гарри Марковица;
- 0–15 баллов — за деплой web-приложения, в том числе за публикацию проекта в репозитории GitHub;
- 0–15 баллов — за конструктивную презентацию разработанного web-приложения.

За индивидуальный предметный тур:

- участники индивидуального предметного тура получают 0–200 баллов, из которых: 0–100 баллов за предметный тур по математике и 0–100 баллов за предметный тур по информатике.

Итоговый балл участника равен:

$$\textit{Математика} \times 15\% + \textit{Информатика} \times 15\% + \textit{Командный балл} \times 70\%.$$

Решение задачи

Решение задачи программно построено в виде последовательных вложений: основного модуля (листинг `main.py`), модуля расчёта оптимальных инвестиций по модели Марковица (листинг `Markowitz.py`) и модуля получения максимального дохода при минимально возможном риске (листинг `optimizer.py`). Проверка работоспособности разработанных web-приложений производилась на тестовом примере парсинга данных с с Московская Биржа/Биржевая информация (`moex.com`). При разработке были использованы следующие фреймворки ЯП Python:

- `numpypandas`;
- `cvxpy`;
- `streamlit`;
- `matplotlib`;
- `seaborn`;
- `psycopg2-binary`;
- `apimoex`;
- `PyPortfolioOpt`.

Типовая структура проекта web-приложения

```
![Go Build](https://github.com/u1ug/Emalak-invest/actions/workflows/docker-image.yml/
↪  badge.svg)

### Структура проекта
* config - модуль для чтения конфигурационного файла
* moex - модуль для получения данных Московской биржи через <a
↪  href="https://pypi.org/project/apimoex/">API</a>
* postgres - модуль для работы с СУБД PostgreSQL
* config.json - файл конфигурации проекта
* Dockerfile - файл для сборки Docker контейнера
* main.py - точка входа Frontend приложения
* markowitz.py - реализация модели Марковица
* optimizer.py - оптимизация портфеля

### Сборка и запуск проекта
git clone https://github.com/u1ug/Emalak-invest
<br>
cd Emalak-invest
<br>
docker build -t emalak .
<br>
docker run -p 28003:28003 emalak

### Альтернативный вариант сборки и запуска
docker pull ghcr.io/u1ug/emalak-invest:latest
<br>
docker run -p 28003:28003 ghcr.io/u1ug/emalak-invest

### Web-приложение
http://92.100.158.108:28003/

### Docker контейнер
https://github.com/users/u1ug/packages/container/package/emalak-invest
```

Основной пользовательский модуль web-приложения main.py

```python
import streamlit as st
from postgres.storage import register, login
from markowitz import minimize_risk, maximize_return, getMaxReturn, getMinRisk,
↪  getStocksData, getMinReturn, getMaxRisk
import datetime
import pandas as pd

# hashed_passwords = stauth.Hasher(['abc', 'def']).generate()
st.set_page_config(page_icon=":money_with_wings:", page_title="EMALAK Invest",
↪  layout="centered", initial_sidebar_state="collapsed")

def LogInClicked(username, password):
    if login(username, password):
        st.session_state['loggedIn'] = True
        st.session_state['username'] = username
        st.session_state['password'] = password
        #print('loggedIn')
    else:
        #st.session_state['loggedIn'] = False
        st.error('Неверный пароль или имя пользователя')

```

```
20  def RegisterClicked(username, password):
21      if register(username, password):
22          st.session_state['loggedIn'] = True
23          st.session_state['username'] = username
24          st.session_state['password'] = password
25          #print('loggedIn')
26      else:
27          #st.session_state['loggedIn'] = False
28          st.error('Неверный пароль или имя пользователя')
29
30  def goToRegistration():
31      st.session_state.status = 'register'
32
33  def show_login_page():
34      if st.session_state['loggedIn'] == False and st.session_state.status ==
        ↪  'login':
35          username = st.text_input(label="username", label_visibility='hidden',
            ↪  value="", placeholder="Имя пользователя")
36          password = st.text_input(label="password", label_visibility='hidden',
            ↪  value="", type="password", placeholder="Пароль")
37          col1, col2 = st.columns(2)
38          with col1:
39              st.button("Войти", on_click=lambda: LogInClicked(username, password))
40          with col2:
41              st.button("Зарегистрироваться", on_click=lambda: goToRegistration())
42
43  def LogOutClicked():
44      st.session_state['loggedIn'] = False
45
46  def show_main_page():
47      st.title('EMALAK Invest')
48      st.title('Здравствуйте, ' + st.session_state['username'])
49      col1, col2 = st.columns([1,1,])
50      with col1:
51          fromDate = st.date_input('От', value=datetime.date(2018, 1, 1),
            ↪  min_value=datetime.date(2015, 1, 1))
52      with col2:
53          toDate = st.date_input('До', min_value=fromDate)
54      if ('stocks' not in st.session_state or 'fromDate' not in st.session_state or
        ↪  'toDate' not in st.session_state) or (st.session_state.fromDate !=
        ↪  fromDate or st.session_state.toDate != toDate):
55          st.session_state.fromDate = fromDate
56          st.session_state.toDate = toDate
57          st.session_state.stocks = getStocksData(start=fromDate, end=toDate)
58
59          min_risk = float(getMinRisk(st.session_state.stocks)) * 100 + 0.005
60          max_risk = float(getMaxRisk(st.session_state.stocks)) * 100
61          st.session_state.max_risk = max_risk
62          st.session_state.min_risk = min_risk
63          st.session_state.target_risk = min_risk
64
65          max_return = float(getMaxReturn(st.session_state.stocks)) * 100 - 0.005
66          min_return = float(getMinReturn(st.session_state.stocks)) * 100 + 0.005
67          st.session_state.min_return = min_return
68          st.session_state.max_return = max_return
69          st.session_state.target_return = max_return
70      if 'max_risk' not in st.session_state or (st.session_state.fromDate !=
        ↪  fromDate or st.session_state.toDate != toDate):
71          min_risk = float(getMinRisk(st.session_state.stocks)) * 100 + 0.005
72          max_risk = float(getMaxRisk(st.session_state.stocks)) * 100
```

```
73          st.session_state.max_risk = max_risk
74          st.session_state.min_risk = min_risk
75      if 'min_return' not in st.session_state or (st.session_state.fromDate !=
        ↪  fromDate or st.session_state.toDate != toDate):
76          max_return = float(getMaxReturn(st.session_state.stocks)) * 100 - 0.005
77          min_return = float(getMinReturn(st.session_state.stocks)) * 100 + 0.005
78          st.session_state.min_return = min_return
79          st.session_state.max_return = max_return
80          st.session_state.target_return = max_return
81      if 'target_risk' not in st.session_state or (st.session_state.fromDate !=
        ↪  fromDate or st.session_state.toDate != toDate):
82          st.session_state.target_risk = st.session_state.min_risk
83      if 'target_return' not in st.session_state or (st.session_state.fromDate !=
        ↪  fromDate or st.session_state.toDate != toDate):
84          st.session_state.target_return = st.session_state.max_return
85
86      profile = st.radio('Профиль', ['Максимизация доходности', 'Минимизация
        ↪  риска'], index = 0)
87      if profile == 'Максимизация доходности':
88              st.session_state.target_risk = st.slider('Максимальный
                ↪  риск',min_value=st.session_state.min_risk,
                ↪  max_value=st.session_state.max_risk, step=0.01,
                ↪  value=st.session_state.target_risk)
89      elif profile == 'Минимизация риска':
90              st.session_state.target_return = st.slider('Минимальная
                ↪  доходность',min_value=st.session_state.min_return,
                ↪  max_value=st.session_state.max_return, step=0.01,
                ↪  value=st.session_state.target_return)
91
92      clicked = st.button('Расчитать')
93      if clicked:
94              if profile == 'Максимизация доходности':
95                  st.write('Расчет максимизации доходности при заданном риске в
                    ↪  {:10.2f}%...'.format(st.session_state.target_risk))
96                  port = maximize_return(stocks=st.session_state.stocks,
                    ↪  target_risk=(st.session_state.target_risk / 100))
97                  perf = port.portfolio_performance()
98                  weights = port.clean_weights()
99                  weights_cpy = weights.copy()
100                 for key, value in weights_cpy.items():
101                     if value <= 0:
102                         del weights[key]
103                 del weights_cpy
104                 expected_annual_return = perf[0]
105                 annual_risk = perf[1]
106                 col1, col2 = st.columns(2)
107                 col1.metric("Доходность", "%.2f" % (expected_annual_return * 100)
                    ↪  + "%")
108                 col2.metric("Риск", "%.2f" % (annual_risk * 100) + "%")
109                 weights = pd.DataFrame(weights, columns=weights.keys(),
                    ↪  index=["Доля в портфеле (%)"])
110                 st.table(weights)
111             elif profile == 'Минимизация риска':
112                 st.write('Расчет минимизации риска при заданной доходности в
                    ↪  {:10.2f}%...'.format(st.session_state.target_return))
113                 port = minimize_risk(stocks=st.session_state.stocks,
                    ↪  target_return=(st.session_state.target_return / 100))
114                 perf = port.portfolio_performance()
115                 weights = port.clean_weights()
116                 weights_cpy = weights.copy()
```

```
                for key, value in weights_cpy.items():
                    if value <= 0:
                        del weights[key]
                del weights_cpy
                expected_annual_return = perf[0]
                annual_risk = perf[1]
                col1, col2 = st.columns(2)
                col1.metric("Доходность", "%.2f" % (expected_annual_return * 100)
                ↪  + "%")
                col2.metric("Риск", "%.2f" % (annual_risk * 100) + "%")
                weights = pd.DataFrame(weights, columns=weights.keys(),
                ↪  index=["Доля в портфеле (%)"])
                st.table(weights)

def show_register_page():
        if st.session_state['loggedIn'] == False and st.session_state.status ==
        ↪  'register':
            username = st.text_input(label="username", key="regusername",
            ↪  label_visibility='hidden', value="", placeholder="Имя
            ↪  пользователя")
            password = st.text_input(label="password",
            ↪  key="regpassword",label_visibility='hidden', value="",
            ↪  type="password", placeholder="Пароль")
            st.button("Зарегистрироваться", key="regbutton", on_click=lambda:
            ↪  RegisterClicked(username, password))


if 'loggedIn' not in st.session_state:
    st.session_state['loggedIn'] = False
    st.session_state.status = 'login'
    show_login_page()
else:
    if st.session_state['loggedIn']:
        show_main_page()
    else:
        if 'status' not in st.session_state:
            st.session_state.status = 'login'
            show_login_page()
        else:
            if st.session_state.status == 'login':
                show_login_page()
            elif st.session_state.status == 'register':
                show_register_page()
```

Вызываемый в main.py модуль расчёта оптимальных инвестиций по модели Марковица Markowitz.py

```
import yfinance as yf
import numpy as np
import pandas as pd
from optimizer import Optimizer
from postgres.storage import load_ticker_history

# Получение данных по ценам акций
def getStocksData(start, end):
    #tickers = ['LKOH.ME','GMKN.ME', 'DSKY.ME', 'NKNC.ME', 'MTSS.ME', 'IRAO.ME',
    ↪  'SBER.ME', 'AFLT.ME']
    #df_stocks= yf.download(tickers, start=start, end=end)['Adj Close']
    #df_stocks.head()
```

```
    #nullin_df = pd.DataFrame(df_stocks,columns=tickers)
    #nullin_df.isnull().sum()
    return load_ticker_history()

# Получение минимального дохода
def getMinReturn(stocks):
    mu = get_mu(stocks)
    sigma = get_sigma(stocks)
    ef = Optimizer(mu, sigma, weight_bounds=(0,1))
    ef.efficient_return(float(0))
    return ef.portfolio_performance()[0]

# Получение максимально возможного риска портфеля
def getMaxRisk(stocks):
    mu = get_mu(stocks)
    sigma = get_sigma(stocks)
    ef = Optimizer(mu, sigma, weight_bounds=(0,1))
    ef.efficient_risk(float(1))
    return ef.portfolio_performance()[1]

# Годовая доходность
def get_mu(prices):
    frequency = 252 # TODO
    if not isinstance(prices, pd.DataFrame):
        print("prices are not in a dataframe")
        prices = pd.DataFrame(prices)
    returns = prices.pct_change().dropna(how="all")
    return (1 + returns).prod() ** (frequency / returns.count()) - 1

def _is_positive_semidefinite(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 fix_nonpositive_semidefinite(matrix):
    print('a')
    if _is_positive_semidefinite(matrix):
        return matrix
    print('b')

    # Eigendecomposition
    q, V = np.linalg.eigh(matrix)

    # Remove negative eigenvalues
    q = np.where(q > 0, q, 0)
    # Reconstruct matrix
    fixed_matrix = V @ np.diag(q) @ V.T

    if not _is_positive_semidefinite(fixed_matrix):  # pragma: no cover
        print("Could not fix matrix.")

    # Rebuild labels if provided
    if isinstance(matrix, pd.DataFrame):
        tickers = matrix.index
        return pd.DataFrame(fixed_matrix, index=tickers, columns=tickers)
    else:
```

```
        return fixed_matrix

# Cov
def get_sigma(prices):
    frequency = 252 # TODO
    if not isinstance(prices, pd.DataFrame):
        print("data is not in a dataframe")
        prices = pd.DataFrame(prices)
    returns = prices.pct_change().dropna(how="all")
    return fix_nonpositive_semidefinite(returns.cov() * frequency)

# Получение минимально возможного риска портфеля
def getMinRisk(stocks):
    mu = get_mu(stocks)
    sigma = get_sigma(stocks)
    ef = Optimizer(mu, sigma, weight_bounds=(0,1))
    ef.min_volatility()
    return ef.portfolio_performance()[1]

# Получение максимально возможного риска портфеля
def getMaxReturn(stocks):
    return max(get_mu(stocks).values)

# Минимальный риск при заданной доходности
def minimize_risk(stocks, target_return: float):
    mu = get_mu(stocks)
    sigma = get_sigma(stocks)
    ef = Optimizer(mu, sigma, weight_bounds=(0,1))
    minrisk=ef.efficient_return(target_return)
    return ef

# Максимальная доходность для заданного риска
def maximize_return(stocks, target_risk: float):
    mu = get_mu(stocks)
    sigma = get_sigma(stocks)
    ef = Optimizer(mu, sigma, weight_bounds=(0,1))
    maxret=ef.efficient_risk(target_risk)
    return ef

#port = minimize_risk(getStocksData(start='2018-01-01', end='2020-12-31'),
↪  target_return=0.25)
#pwt=port.clean_weights()
#print("Weights", pwt)
#print("Portfolio performance:")
#print(port.portfolio_performance())
```

Вызываемый в Markowitz.py модуль получения максимального дохода при минимально возможном риске optimizer.py

```
import collections
import copy
from collections.abc import Iterable
from typing import List

import numpy as np
import pandas as pd
import cvxpy as cp
import scipy.optimize as sco

```

```python
# Выпуклая оптимизация с cvxpy
class Optimizer():
    def __init__(
        self,
        expected_returns,
        cov_matrix,
        weight_bounds=(0, 1),
        solver_options=None,
    ):
        self.cov_matrix = cov_matrix
        self.expected_returns = expected_returns
        self._max_return_value = None

        if self.expected_returns is None:
            num_assets = len(cov_matrix)
        else:
            num_assets = len(expected_returns)

        if isinstance(expected_returns, pd.Series):
            tickers = list(expected_returns.index)
        elif isinstance(cov_matrix, pd.DataFrame):
            tickers = list(cov_matrix.columns)
        else:  # int labels
            tickers = list(range(num_assets))
        self.tickers = tickers

        # Optimization variables
        self.n_assets = len(tickers)
        self._w = cp.Variable(len(tickers))
        self._objective = None
        self._additional_objectives = []
        self._constraints = []
        self._lower_bounds = None
        self._upper_bounds = None
        self._opt = None
        self._solver = None
        self._solver_options = solver_options if solver_options else {}
        self._map_bounds_to_constraints(weight_bounds)

    def clean_weights(self, cutoff=0.01, rounding=5):
        clean_weights = self.weights.copy()
        clean_weights[np.abs(clean_weights) < cutoff] = 0
        if rounding is not None:
            clean_weights = np.round(clean_weights, rounding)

        return self._make_output_weights(clean_weights)

    def _make_output_weights(self, weights=None):
        if weights is None:
            weights = self.weights
        return collections.OrderedDict(zip(self.tickers, weights))

    def _solve_cvxpy_opt_problem(self):
        try:
            if self._opt is None:
                self._opt = cp.Problem(cp.Minimize(self._objective),
                ↪  self._constraints)
                self._initial_objective = self._objective.id
                self._initial_constraint_ids = {const.id for const in
                ↪  self._constraints}
```

```
69              self._opt.solve(
70                  solver=self._solver, verbose=False, **self._solver_options
71              )
72
73          except (TypeError, cp.DCPError) as e:
74              raise e
75          self.weights = self._w.value.round(16) + 0.0  # +0.0 removes signed zero
76          return self._make_output_weights()
77
78      def add_constraint(self, new_constraint):
79          self._constraints.append(new_constraint(self._w))
80
81      def min_volatility(self):
82          self._objective = portfolio_variance(
83              self._w, self.cov_matrix
84          )
85          for obj in self._additional_objectives:
86              self._objective += obj
87
88          self.add_constraint(lambda w: cp.sum(w) == 1)
89          return self._solve_cvxpy_opt_problem()
90
91      def _max_return(self, return_value=True):
92          self._objective = portfolio_return(
93              self._w, self.expected_returns
94          )
95
96          self.add_constraint(lambda w: cp.sum(w) == 1)
97
98          res = self._solve_cvxpy_opt_problem()
99
100         if return_value:
101             return -self._opt.value
102         else:
103             return res
104
105     def is_parameter_defined(self, parameter_name: str) -> bool:
106         is_defined = False
107         objective_and_constraints = (
108             self._constraints + [self._objective]
109             if self._objective is not None
110             else self._constraints
111         )
112         for expr in objective_and_constraints:
113             params = [
114                 arg for arg in _get_all_args(expr) if isinstance(arg,
                    ↪  cp.Parameter)
115             ]
116             for param in params:
117                 if param.name() == parameter_name and not is_defined:
118                     is_defined = True
119         return is_defined
120
121     def efficient_risk(self, target_volatility):
122         if not isinstance(target_volatility, (float, int)) or target_volatility <
            ↪  0:
123             raise ValueError("Заданный риск должен быть float и >= 0")
124
125         global_min_volatility = np.sqrt(1 /
            ↪  np.sum(np.linalg.pinv(self.cov_matrix)))
```

```
126
127         if target_volatility < global_min_volatility:
128             raise ValueError("Минимальный риск равен {:.3f}. Используйте более
                ↪  высокий заданный риск".format(global_min_volatility))
129
130         update_existing_parameter = self.is_parameter_defined("target_variance")
131         if update_existing_parameter:
132             self.update_parameter_value("target_variance", target_volatility**2)
133         else:
134             self._objective = portfolio_return(
135                 self._w, self.expected_returns
136             )
137             variance = portfolio_variance(self._w, self.cov_matrix)
138
139             for obj in self._additional_objectives:
140                 self._objective += obj
141
142             target_variance = cp.Parameter(
143                 name="target_variance", value=target_volatility**2, nonneg=True
144             )
145             self.add_constraint(lambda _: variance <= target_variance)
146             self.add_constraint(lambda w: cp.sum(w) == 1)
147         return self._solve_cvxpy_opt_problem()
148
149     def efficient_return(self, target_return):
150         if not isinstance(target_return, float):
151             raise ValueError("Заданная доходность должна быть float")
152         if not self._max_return_value:
153             a = self.deepcopy()
154             self._max_return_value = a._max_return()
155         if target_return > self._max_return_value:
156             raise ValueError("Заданная доходность должна быть меньше чем
                ↪  максимально возможная")
157
158         update_existing_parameter = self.is_parameter_defined("target_return")
159         if update_existing_parameter:
160             self._validate_market_neutral()
161             self.update_parameter_value("target_return", target_return)
162         else:
163             self._objective = portfolio_variance(
164                 self._w, self.cov_matrix
165             )
166             ret = portfolio_return(
167                 self._w, self.expected_returns, negative=False
168             )
169
170             for obj in self._additional_objectives:
171                 self._objective += obj
172
173             target_return_par = cp.Parameter(name="target_return",
                ↪  value=target_return)
174             self.add_constraint(lambda _: ret >= target_return_par)
175             self.add_constraint(lambda w: cp.sum(w) == 1)
176         return self._solve_cvxpy_opt_problem()
177
178     def _map_bounds_to_constraints(self, test_bounds):
179         # If it is a collection with the right length, assume they are all bounds.
180         if len(test_bounds) == self.n_assets and not isinstance(
181             test_bounds[0], (float, int)
182         ):
```

```
            bounds = np.array(test_bounds, dtype=float)
            self._lower_bounds = np.nan_to_num(bounds[:, 0], nan=-np.inf)
            self._upper_bounds = np.nan_to_num(bounds[:, 1], nan=np.inf)
        else:
            lower, upper = test_bounds

            # Replace None values with the appropriate +/- 1
            if np.isscalar(lower) or lower is None:
                lower = -1 if lower is None else lower
                self._lower_bounds = np.array([lower] * self.n_assets)
                upper = 1 if upper is None else upper
                self._upper_bounds = np.array([upper] * self.n_assets)
            else:
                self._lower_bounds = np.nan_to_num(lower, nan=-1)
                self._upper_bounds = np.nan_to_num(upper, nan=1)

        self.add_constraint(lambda w: w >= self._lower_bounds)
        self.add_constraint(lambda w: w <= self._upper_bounds)

    def deepcopy(self):
        self_copy = copy.copy(self)
        self_copy._additional_objectives = [
            copy.copy(obj) for obj in self_copy._additional_objectives
        ]
        self_copy._constraints = [copy.copy(con) for con in
        ↪  self_copy._constraints]
        return self_copy

    def portfolio_performance(self, risk_free_rate=0.02):
        if isinstance(self.weights, dict):
            if isinstance(self.expected_returns, pd.Series):
                tickers = list(self.expected_returns.index)
            elif isinstance(self.cov_matrix, pd.DataFrame):
                tickers = list(self.cov_matrix.columns)
            else:
                tickers = list(range(len(self.expected_returns)))
            new_weights = np.zeros(len(tickers))

            for i, k in enumerate(tickers):
                if k in self.weights:
                    new_weights[i] = self.weights[k]
        else:
            new_weights = np.asarray(self.weights)

        sigma = np.sqrt(portfolio_variance(new_weights, self.cov_matrix))

        if self.expected_returns is not None:
            mu = portfolio_return(new_weights, self.expected_returns,
            ↪  negative=False)
            return mu, sigma
        else:
            return None, sigma

def _get_all_args(expression: cp.Expression) -> List[cp.Expression]:
    if expression.args == []:
        return [expression]
    else:
        return list(_flatten([_get_all_args(arg) for arg in expression.args]))

def _flatten(l: Iterable) -> Iterable:
```

```
    # Helper method to flatten an iterable
    for el in l:
        if isinstance(el, Iterable) and not isinstance(el, (str, bytes)):
            yield from _flatten(el)
        else:
            yield el

def _objective_value(w, obj):
    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 portfolio_variance(w, cov_matrix):
    variance = cp.quad_form(w, cov_matrix)
    return _objective_value(w, variance)

def portfolio_return(w, expected_returns, negative=True):
    sign = -1 if negative else 1
    mu = w @ expected_returns
    return _objective_value(w, sign * mu)
```

Модуль для парсинга данных moex.py

```
import pandas as pd
import apimoex
import requests
from datetime import date
from json import load
from postgres.storage import *
# получение исторических данных московской биржи по тикеру
def get_historical(ticker):
    with requests.Session() as session:
        data = apimoex.get_board_history(session, ticker)
    df = pd.DataFrame(data)
    df = df.drop('BOARDID', axis=1)
    df = df.dropna()
    df['unix_time'] = pd.to_datetime(df.TRADEDATE).astype(int) / 10**9
    return df

# обрезать датафрейм в обпределённом временном диапазоне
def cut_historical(df, start_date, end_date):
    mask = (df['TRADEDATE'] > start_date) & (df['TRADEDATE'] <= end_date)
    return df.loc[mask]
# получение исторических данных из бд по тикеру
def load_historical(tickers, start_date, end_date):
    conn = connect()
    out = pd.DataFrame()
    frames = []
    for ticker in tickers:
        df = load_ticker_history(conn, ticker, start_date, end_date)
        frames.append(df)
    out['date'] = frames[0]['TRADEDATE']

```

```
    for ticker, df in zip(tickers, frames):
        out[ticker] = df['CLOSE']
    out = out.fillna(out.mean())
    out = out.dropna(axis=1, how='all')
    out = out.set_index('date')
    return out
```

Модуль для работы с БД PostgreSQL storage.py

```
import psycopg2
import pandas as pd
from psycopg2.extras import NamedTupleCursor
from json import load
# чтение файла конфигурации
def read_config(path):
    cfg = open(path, 'r')
    return load(cfg)

cfg = read_config('config.json')
# подключение к БД
def connect():
    global cfg
    conn = psycopg2.connect(
    host=cfg.get('host'),
    password=cfg.get('password'),
    user=cfg.get('user'),
    port = cfg.get('port'),
    database = cfg.get('database')
    )
    return conn
# создание таблицы
def create_table(conn,name, columns):
    cur = conn.cursor()
    request = f"create table {name} ({columns})"
    cur.execute(request)
    cur.execute("commit")
    cur.close()
# удаление таблицы
def drop_table(conn, name):
    cur = conn.cursor()
    cur.execute(f"drop table {name}")
    cur.execute("commit")
    cur.close()
# получение информации из таблицы
def query(conn, request):
    cur = conn.cursor()
    cur.execute(request)
    out = cur.fetchall()
    cur.close()
    return out
# добавить исторические данные в бд
def insert_ticker_history(conn, ticker, df):
    cur = conn.cursor()
    for _, row in df.iterrows():
        request = f"INSERT INTO {ticker} (TRADEDATE, CLOSE, VOLUME, VALUE,
        ↪  unix_time) VALUES('{row[0]}', {row[1]}, {row[2]}, {row[3]}, {row[4]})"
        cur.execute(request)
        cur.execute('commit')
    cur.close()
```

```python
# получение исторических данных
def load_ticker_history(conn, ticker, start_human_readable, end_human_readable):
    start_unix_time = float((pd.to_datetime([start_human_readable]).astype(int) / 10**9)[0])
    end_unix_time = float((pd.to_datetime([end_human_readable]).astype(int) / 10**9)[0])
    out = pd.DataFrame(columns=['TRADEDATE', 'CLOSE', 'VOLUME', 'VALUE', 'unix_time'], data=query(conn, f"select * from {ticker} where unix_time >= {start_unix_time} and unix_time <= {end_unix_time}"))
    return out
# создание таблицы с пользователями
def init_table():
    conn = connect()
    with conn.cursor() as curs:
        curs.execute("create table if not exists users (username VARCHAR(50) PRIMARY KEY, password VARCHAR(36) NOT NULL);")
    conn.commit()
    conn.close()
# вход пользователя
def login(username, password):
    print('login', username, password)
    init_table()
    conn = connect()
    realPass = ''
    try:
        with conn.cursor(cursor_factory=NamedTupleCursor) as curs:
            curs.execute("SELECT password FROM users WHERE username=%s", (username,))
            sgbdrn = curs.fetchone()
            if sgbdrn is None:
                conn.close()
                return False
            realPass = sgbdrn[0]
    except e:
        print(e)
        conn.Close()
        return False
    conn.close()

    return password == realPass
# регистрация пользоавтеля
def register(username, password):
    print('register', username, password)
    init_table()
    conn = connect()
    try:
        with     conn.cursor(cursor_factory=NamedTupleCursor) as curs:
            curs.execute("INSERT INTO users (username, password) VALUES (%s, %s)", (username, password))
    except e:
        print(e)
        conn.close()
        return False
    conn.commit()
    conn.close()
    return True
```

Тестовый пример для проверки работы web-приложения

https://disk.yandex.ru/i/8PSjXo2k_zIGsg.

Материалы для подготовки

До финального командного тура с целью сплачивания команд и образовательной подготовки участников рекомендуется обязательное проведение специализированного хакатона по контейнеризации в Docker и работе с ключевыми для задачи фреймворками ЯП Python: Pandas, Streamlit, NumPy, SymPy, SciPy, cvxpy, Matplotlib, seaborn, БД PostgreSQL, а также по модели формирования оптимального портфеля ценных бумаг на базе теории Гарри Марковица.