In [1]:
```
import pandas as pd
import numpy as np
import requests as req
import datetime as dt
import time
from scipy.stats import norm, binom
import statsmodels.api as sm
```

Pet проект.

Данная тетрадь представляет из себя набор функций на основании которых можно составить стратегию для торгового бота или фундамент для обучения текстовых моделей

Представлены функции для загрузки данных с Московской биржи:

- Цены акции
- Индекс MOEX
- Кривая бескупонной доходности

Представлены функции для расчета:

- бета коэффициента
- альфа коэффициента
- event study

Загрузка данных

Константы для функций

In [2]:
```
# Определение констант
SECID = 'SBER'
INDEXID = 'IMOEX'
END_DATE = dt.datetime.today().strftime('%Y-%m-%d') # Окно заканчивается сегодняшней ценой закрытия
START_DATE = (dt.datetime.today() - dt.timedelta(days=365)
              ).strftime('%Y-%m-%d') #Окно начинается год назад
EVENT_DATE = (dt.datetime.today() - dt.timedelta(days=10)
              ).strftime('%Y-%m-%d')  # Дата события 10 дней назад
```

Загрузка данных акций

Для загрузки данных с API применяются три основные функции: **flatten**, **get_moex_stock_data**, **get_moex_data_and_prepare**

Описание функций

- flatten используется для преобразования вложенного JSON-объекта, полученного из MOEX API, в плоский список словарей.

Параметры:

- j (dict): JSON-объект, полученный из MOEX API. Этот объект должен содержать блок данных с определёнными столбцами и значениями.
- blockname (str): Имя ключа в JSON-объекте, который содержит необходимые данные. Обычно этот ключ указывает на блок данных, например, “history”.

- Функция get_moex_stock_data Функция возвращает список словарей, где каждый словарь представляет одну запись с данными о ценах акций.

Параметры:

- secid (str): Идентификатор ценной бумаги.
- start_date (str or datetime.date): Начальная дата в формате “YYYY-MM-DD” или объект datetime.date.
- end_date (str or datetime.date): Конечная дата в формате “YYYY-MM-DD” или объект datetime.date.
- engine (str): Движок рынка. По умолчанию ‘stock’.
- market (str): Рынок. По умолчанию ‘shares’.
- board (str): Торговая площадка. По умолчанию ‘TQBR’.

- get_moex_data_and_prepare - извлекает данные о ценах акций с MOEX API за заданный период времени.

Параметры:

- secid (str): Идентификатор ценной бумаги.
- start_date (str or datetime.date): Начальная дата в формате “YYYY-MM-DD” или объект datetime.date.
- end_date (str or datetime.date): Конечная дата в формате “YYYY-MM-DD” или объект datetime.date.

Функция использует **get_moex_stock_data** для получения данных и преобразует их в DataFrame с ценами акций и доходностью.

Функции

In [3]:
```
# Функция для извлечения данных с JSON объекта
def flatten(j: dict, blockname: str):
    columns = j[blockname]['columns']
    return [{k: r[I] for I, k in enumerate(columns)} for r in j[blockname]['data']]

# Функция для запроса тикера и извлечения данных с JSON объекта
def get_moex_stock_data(secid, start_date, end_date, engine='stock', market='shares', board='TQBR'):
    # Преобразование дат в объекты datetime, если они являются строками
    if isinstance(start_date, str):
        start_date = dt.datetime.strptime(start_date, "%Y-%m-%d").date()
    if isinstance(end_date, str):
        end_date = dt.datetime.strptime(end_date, "%Y-%m-%d").date()
```

```
    all_data = []
    current_start_date = start_date

    while current_start_date <= end_date:
        current_end_date = current_start_date + dt.timedelta(days=99)
        if current_end_date > end_date:
            current_end_date = end_date

        # Запрос к MOEX
        url = f"https://iss.moex.com/iss/history/engines/{engine}/markets/{market}/boards/{board}/securities/{secid}.json?from={current_s
        try:
            r = req.get(url)
            r.encoding = 'utf-8'
            j = r.json()

            # Преобразование ответа и добавление данных в список all_data
            flattened_data = flatten(j, 'history')
            all_data.extend(flattened_data)

            # Переход к следующему периоду
            current_start_date = current_end_date + dt.timedelta(days=1)

            # Добавление задержки, чтобы избежать отправки слишком большого количества запросов за короткий период
            time.sleep(1)

        except req.exceptions.RequestException as e:
            print(f"Запрос не удался: {e}")
            break

    return all_data

# Функция для обработки данных и упрощения данных
def get_moex_data_and_prepare(secid, start_date, end_date):
    data = get_moex_stock_data(secid, start_date, end_date)
    df = pd.DataFrame(data)
    df['TRADEDATE'] = pd.to_datetime(df['TRADEDATE'])
    df.set_index('TRADEDATE', inplace=True)
    df = df[['CLOSE']].asfreq("B").fillna(method='ffill')
    df.columns = [f'{secid}_Stock_Price']

    # Добавление столбца с доходностью и удаление первой строки
    df[f'{secid}_Daily_Return'] = df[f'{secid}_Stock_Price'].pct_change()
    df.dropna(inplace=True)

    return df
```

Пример использования

In [4]:
```
ticker_df = get_moex_data_and_prepare(SECID, START_DATE, END_DATE)
```

In [5]:
```
display(ticker_df.tail(10))
display(ticker_df.describe().T)
mean_ticker_return = round(ticker_df[f'{SECID}_Daily_Return'].mean(), 3)
print(f'Средняя доходность {SECID}:{mean_ticker_return}')
```

TRADEDATE	SBER_Stock_Price	SBER_Daily_Return
2024-05-28	318.22	0.003564
2024-05-29	320.38	0.006788
2024-05-30	316.63	-0.011705
2024-05-31	313.11	-0.011117
2024-06-03	310.95	-0.006899
2024-06-04	316.49	0.017816
2024-06-05	314.72	-0.005593
2024-06-06	313.08	-0.005211
2024-06-07	319.90	0.021784
2024-06-10	317.28	-0.008190

	count	mean	std	min	25%	50%	75%	max
SBER_Stock_Price	259.0	277.494324	22.571803	235.670000	260.985000	273.770000	295.510000	323.540000
SBER_Daily_Return	259.0	0.001069	0.011017	-0.043058	-0.004702	0.000678	0.005507	0.072818

```
Средняя доходность SBER:0.001
```

Загрузка данных индекса

Описание функций

- get_moex_index_data возвращает список словарей, где каждый словарь представляет одну запись с данными об индексах.

Параметры:

- indexid (str): Идентификатор индекса.
- start_date (str or datetime.date): Начальная дата в формате “YYYY-MM-DD” или объект datetime.date.
- end_date (str or datetime.date): Конечная дата в формате “YYYY-MM-DD” или объект datetime.date.
- engine (str): Движок рынка. По умолчанию ‘stock’.
- market (str): Рынок. По умолчанию ‘index’.
- board (str): Торговая площадка. По умолчанию ‘SNDX’.

- get_moex_index_data_and_prepare - извлекает данные о котировках индекса из словаря за заданный период времени.

Параметры:

- indexid (str): Идентификатор индекса.
- start_date (str or datetime.date): Начальная дата в формате “YYYY-MM-DD” или объект datetime.date.
- end_date (str or datetime.date): Конечная дата в формате “YYYY-MM-DD” или объект datetime.date.

Функция использует **get_moex_index_data** для получения данных и преобразует их в DataFrame с ценами индекса и доходностью.

Функции

In [6]:
```
def get_moex_index_data(indexid, start_date, end_date, engine='stock', market='index', board='SNDX'):
    # Преобразование дат в объекты datetime, если они являются строками
    if isinstance(start_date, str):
        start_date = dt.datetime.strptime(start_date, "%Y-%m-%d").date()
    if isinstance(end_date, str):
        end_date = dt.datetime.strptime(end_date, "%Y-%m-%d").date()

    all_data = []
    current_start_date = start_date

    while current_start_date <= end_date:
        current_end_date = current_start_date + dt.timedelta(days=99)
        if current_end_date > end_date:
            current_end_date = end_date

        # Запрос к MOEX
        url = f"https://iss.moex.com/iss/history/engines/{engine}/markets/{market}/boards/{board}/securities/{indexid}.json?from={current
        try:
            r = req.get(url)
            r.encoding = 'utf-8'
            j = r.json()

            # Преобразование ответа и добавление данных в список all_data
            flattened_data = flatten(j, 'history')
            all_data.extend(flattened_data)

            # Переход к следующему периоду
            current_start_date = current_end_date + dt.timedelta(days=1)

            # Добавление задержки, чтобы избежать отправки слишком большого количества запросов за короткий период
            time.sleep(1)

        except req.exceptions.RequestException as e:
            print(f"Запрос не удался: {e}")
            break

    return all_data

# Функция для получения данных MOEX и их подготовки
def get_moex_index_data_and_prepare(indexid, start_date, end_date):
    data = get_moex_index_data(indexid, start_date, end_date)
    df = pd.DataFrame(data)
    df['TRADEDATE'] = pd.to_datetime(df['TRADEDATE'])
    df.set_index('TRADEDATE', inplace=True)
    df = df[['CLOSE']].asfreq("B").fillna(method='ffill')
    df.columns = [f'{indexid}_Index_Price']

    # Добавление столбца с доходностью и удаление первой строки
    df[f'{indexid}_Daily_Return'] = df[f'{indexid}_Index_Price'].pct_change()
    df.dropna(inplace=True)

    return df
```

Пример использования

In [7]:
```
# Пример использования функции для индекса IMOEX
index_df = get_moex_index_data_and_prepare(INDEXID, START_DATE, END_DATE)
```

In [8]:
```
#Отразим результаты
display(index_df.tail(10))
display(index_df.describe().T)
mean_index_return = round(index_df[f'{INDEXID}_Daily_Return'].mean() * 252, 3)
print(f'Средняя доходность {INDEXID}: {mean_index_return}')
```

TRADEDATE	IMOEX_Index_Price	IMOEX_Daily_Return
2024-05-28	3302.91	0.001118
2024-05-29	3318.03	0.004578
2024-05-30	3282.18	-0.010805
2024-05-31	3217.19	-0.019801
2024-06-03	3141.42	-0.023552
2024-06-04	3186.93	0.014487
2024-06-05	3212.24	0.007942
2024-06-06	3192.38	-0.006183
2024-06-07	3233.22	0.012793
2024-06-10	3180.94	-0.016170

	count	mean	std	min	25%	50%	75%	max
IMOEX_Index_Price	259.0	3182.241583	169.916104	2757.130000	3099.315000	3191.050000	3266.880000	3501.890000
IMOEX_Daily_Return	259.0	0.000580	0.007456	-0.028641	-0.002446	0.001248	0.004778	0.022554

```
Средняя доходность IMOEX: 0.146
```

Загрузка безрисоквой ставки

Описание функций

get_rf_moex используется для получения данных о безрисковой ставке с MOEX API на заданную дату.

Параметры:

- date (str or datetime.date): Дата запроса в формате “YYYY-MM-DD” или объект datetime.date.
- engine (str): Движок рынка. По умолчанию ‘stock’.

get_risk_free_rate используется для извлечения значения безрисковой ставки на заданную дату и для заданного периода с MOEX API.

Параметры:

- date (str or datetime.date): Дата запроса в формате “YYYY-MM-DD” или объект datetime.date.
- period (float): Период безрисковой ставки в годах. По умолчанию 10 лет

Функции

In [9]:
```
#Функция для получения json файла
def get_rf_moex(date, engine='stock'):
    # Проверяем, является ли дата строкой, и преобразуем её в объект datetime.date, если это так
    if isinstance(date, str):
        date = dt.datetime.strptime(date, "%Y-%m-%d").date()

    # Формируем URL для запроса к API MOEX
    url = f"https://iss.moex.com/iss/engines/{engine}/zcyc.json?date={date}&iss.meta=off"

    try:
        # Отправляем GET-запрос к API MOEX
        r = req.get(url)
        # Проверяем, успешен ли запрос
        if r.status_code != 200:
            print(f"HTTP Error: {r.status_code}")
            return None

        # Устанавливаем кодировку ответа
        r.encoding = 'utf-8-sig'
        # Преобразуем ответ в формат JSON
        j = r.json()
        return j

    except req.exceptions.RequestException as e:
        # Обрабатываем исключения, связанные с запросом
        print(f"Request failed: {e}")
        return None

#Функция включающая загрузку и извлечения из файла JSON безрисковой ставки
def get_risk_free_rate(date, period=10):
    # Получаем данные с MOEX с помощью функции get_rf_moex
    response_json = get_rf_moex(date)
    if not response_json:
        return None

    # Извлекаем данные по доходностям
    yearyields_data = response_json.get('yearyields', {})
    columns = yearyields_data.get('columns', [])
    data = yearyields_data.get('data', [])

    # Проходимся по строкам данных
    for row in data:
        # Формируем словарь yield_info, где ключами являются названия столбцов
        yield_info = {columns[I]: row[I] for I in range(len(columns))}
        # Проверяем, соответствует ли текущий период заданному
        if yield_info.get('period') == period:
            # Возвращаем значение доходности, округленное до трех знаков после запятой
            return round(yield_info.get('value') / 100, 3)

    # Возвращаем None, если доходность за заданный период не найдена
    return None
```

Пример использования

In [10]:
```
risk_free_rate = get_risk_free_rate('2024-01-10')
print(f"10 летняя доходность по бескупоновым облигациям: {risk_free_rate}")
```

```
10 летняя доходность по бескупоновым облигациям: 0.118
```

Событийный анализ

Расчет метрик alpha & beta

Описание функций

calculate_beta используется для расчета бета коэффициента, который измеряет чувствительность доходности акций относительно доходности рыночного индекса.

Параметры:

- ticker_df (pd.DataFrame): DataFrame с данными о ценах акций, включая дневную доходность.
- index_df (pd.DataFrame): DataFrame с данными об индексе, включая дневную доходность.
- ticker_id (str): Идентификатор ценной бумаги.

- index_id (str): Идентификатор индекса.
- window (int): Окно для расчета бета коэффициента в днях. По умолчанию 90.

Функция объединяет данные акций и индекса, берет последние данные в указанном окне и использует линейную регрессию для расчета бета коэффициента.

calculate_alpha используется для расчета альфа коэффициента, который измеряет избыточную доходность акций по сравнению с ожидаемой доходностью, рассчитанной на основе бета коэффициента.

Параметры:

- ticker_df (pd.DataFrame): DataFrame с данными о ценах акций, включая дневную доходность.
- index_df (pd.DataFrame): DataFrame с данными об индексе, включая дневную доходность.
- ticker_id (str): Идентификатор ценной бумаги.
- index_id (str): Идентификатор индекса.
- beta (float): Предварительно рассчитанный бета коэффициент для ценной бумаги.
- risk_free_rate (float): Безрисковая ставка доходности. По умолчанию 0.0.
- window (int): Окно для расчета альфа коэффициента в днях. По умолчанию 90.

Функция объединяет данные акций и индекса, берет последние данные в указанном окне, вычисляет среднюю доходность акций и индекса, и на основе этих данных рассчитывает альфа коэффициент.

Функции

In [11]:
```python
# Функция для расчета бета коэффициента
def calculate_beta(ticker_df, index_df, ticker_id, index_id, window=90):
    # Объединение данных на основе даты
    merged_df = pd.merge(ticker_df, index_df, left_index=True, right_index=True, suffixes=('_ticker', '_index'))

    # Взятие только последних данных в окне
    returns_ticker = merged_df[f'{ticker_id}_Daily_Return'].tail(window)
    returns_index = merged_df[f'{index_id}_Daily_Return'].tail(window)

    # Добавление константы для выполнения линейной регрессии
    X = sm.add_constant(returns_index)

    # Создание и подгонка модели
    model = sm.OLS(returns_ticker, X).fit()

    # Извлечение бета коэффициента
    beta = round(model.params[1], 3)

    return beta

def calculate_alpha(ticker_df, index_df, ticker_id, index_id, beta, window=90):
    # Объединение данных на основе даты
    merged_df = pd.merge(ticker_df, index_df, left_index=True, right_index=True, suffixes=('_ticker', '_index'))

    # Взятие только последних данных в окне
    returns_ticker = merged_df[f'{ticker_id}_Daily_Return'].tail(window)
    returns_index = merged_df[f'{index_id}_Daily_Return'].tail(window)

    # Средняя доходность акций и индекса
    mean_ticker_return = returns_ticker.mean()
    mean_index_return = returns_index.mean()

    # Получение безрисковой ставки доходности для последней даты в окне
    risk_free_rate = get_risk_free_rate(returns_ticker.index[-1]) / 365

    if risk_free_rate is None:
        print("Не удалось получить безрисковую ставку доходности")
        return None

    # Расчет рыночной премии
    market_premium = mean_index_return - risk_free_rate

    # Расчет альфа коэффициента
    alpha = round(mean_ticker_return - risk_free_rate - beta * market_premium, 3)

    return alpha
```

Пример использования

Бета-коэффициент

- **Бета** Коэффициент наклона линии регрессии, показывающий чувствительность доходности актива к доходности рынка.

- **beta** = 1: Доходность актива изменяется в точности как доходность рынка. Актив имеет средний рыночный риск.
- **beta** > 1: Актив более волатильный и рискованный по сравнению с рынком. Например, **beta** = 1.5 означает, что если рынок вырастет на 1%, доходность актива вырастет на 1.5%, и наоборот.
- **beta** < 1: Актив менее волатильный и менее рискованный по сравнению с рынком. Например, **beta** = 0.5 означает, что если рынок вырастет на 1%, доходность актива вырастет на 0.5%, и наоборот.
- **beta** = 0: Доходность актива не зависит от доходности рынка. Такие активы имеют минимальный рыночный риск.
- **beta** < 0: Актив движется в противоположном направлении от рынка. Например, **beta** = -1 означает, что если рынок вырастет на 1%, доходность актива упадет на 1%, и наоборот.

Альфа-коэффициент

- **Альфа** Интерсепт, показывающий избыточную доходность актива после учета влияния рынка.

- **alpha** > 0: Актив обеспечивает избыточную доходность по сравнению с ожидаемой доходностью на основе его риска. Это указывает на превосходную управленческую эффективность.
- **alpha** = 0: Актив обеспечивает доходность, соответствующую ожидаемой доходности на основе его риска. Это указывает на то, что актив работает так, как предсказывается моделью.

- **alpha** < 0: Актив обеспечивает доходность ниже ожидаемой на основе его риска. Это указывает на недостаточную управленческую эффективность.

In [12]:
```
# Расчет коэффициента бета
beta = calculate_beta(ticker_df, index_df, SECID, INDEXID, window=90)

# Расчет коэффициента альфа
alpha = calculate_alpha(ticker_df, index_df, SECID, INDEXID, beta, window=90)

print("Бета коэффициент:", beta)
print("Альфа коэффициент:", alpha)
```

```
Бета коэффициент: 0.736
Альфа коэффициент: 0.001
```

Событийный анализ (Event study)

Описание функции

event_study используется для проведения исследования событий (event study), которое анализирует влияние определенного события на цены акций.

Параметры:

- ticker_id (str): Идентификатор ценной бумаги.
- index_id (str): Идентификатор индекса.
- event_date (datetime.date): Дата события.
- estimation_window (int): Окно оценки в днях до события. По умолчанию 90.
- event_window (int): Окно события в днях до и после даты события. По умолчанию 5.

Функция извлекает данные о ценах акций и индекса за период, включающий окна оценки и события, рассчитывает бета и альфа коэффициенты на основе окна оценки (до события), и вычисляет нормальную и аномальную доходность акций в окне события. Возвращает аномальную доходность и кумулятивную аномальную доходность (CAR).

Функции:

In [13]:
```
#Функция для выполнения событийного анализа
def event_study(ticker_id, index_id, event_date, estimation_window=90, event_window=5):
    event_date = dt.datetime.strptime(event_date, '%Y-%m-%d').date()
    start_date = (event_date - dt.timedelta(days=estimation_window + event_window)).strftime('%Y-%m-%d')
    end_date = (event_date + dt.timedelta(days=event_window)).strftime('%Y-%m-%d')
    ticker_df = get_moex_data_and_prepare(ticker_id, start_date, end_date)
    index_df = get_moex_index_data_and_prepare(index_id, start_date, end_date)

    # Определение периода оценки и периода события
    estimation_df = ticker_df[:event_date].tail(estimation_window)
    event_df = ticker_df[event_date - dt.timedelta(days=event_window):event_date + dt.timedelta(days=event_window)]

    index_estimation_df = index_df[:event_date].tail(estimation_window)
    index_event_df = index_df[event_date - dt.timedelta(days=event_window):event_date + dt.timedelta(days=event_window)]

    # Расчет бета и альфа коэффициентов
    beta = calculate_beta(estimation_df, index_estimation_df, ticker_id, index_id, window=estimation_window)
    alpha = calculate_alpha(estimation_df, index_estimation_df, ticker_id, index_id, beta, window=estimation_window)

    # Расчет нормальной и аномальной доходности
    normal_returns = alpha + beta * index_event_df[f'{index_id}_Daily_Return']
    abnormal_returns = event_df[f'{ticker_id}_Daily_Return'] - normal_returns

    # Расчет кумулятивной аномальной доходности (CAR)
    CAR = abnormal_returns.cumsum()

    # Создание DataFrame с результатами
    results_df = pd.DataFrame({
        f'Факт_доходность_{ticker_id}': event_df[f'{ticker_id}_Daily_Return'],
        f'{index_id}_факт_доходность': index_event_df[f'{index_id}_Daily_Return'],
        'Прогноз_доходности': normal_returns,
        'AR': abnormal_returns,
        'CAR': CAR})

    results_df.index.name = 'Date'
    results_df = results_df.round(3)

    return results_df, beta, alpha, estimation_df
```

Пример использования

In [14]:
```
results_df, beta, alpha, estimation_df = event_study(SECID, INDEXID, EVENT_DATE)
display(results_df.tail(10))
print(f"Бета коэффициент: {beta}")
print(f"Альфа коэффициент: {alpha}")
```

Date	Факт_доходность_SBER	IMOEX_факт_доходность	Прогноз_доходности	AR	CAR
2024-05-27	-0.012	-0.029	-0.019	0.007	0.007
2024-05-28	0.004	0.001	0.002	0.002	0.009
2024-05-29	0.007	0.005	0.004	0.003	0.011
2024-05-30	-0.012	-0.011	-0.007	-0.005	0.006
2024-05-31	-0.011	-0.020	-0.013	0.002	0.008
2024-06-03	-0.007	-0.024	-0.016	0.009	0.017
2024-06-04	0.018	0.014	0.011	0.007	0.023
2024-06-05	-0.006	0.008	0.007	-0.012	0.011
2024-06-06	-0.005	-0.006	-0.003	-0.002	0.009

```
Бета коэффициент: 0.703
Альфа коэффициент: 0.001
```

Данный метод может быть развит для тренировки ИИ в целях оценки влияния новостного контента на котировки акций.

Есть несколько возможных причин, по которым фактическая доходность (**Actual Return**) значительно ниже прогнозируемой доходности (**Predicted Return**):

- **Непредвиденные события:** Вне зависимости от основного события, могли произойти другие непредвиденные события, влияющие на цену акций. Это могут быть новости, изменения в макроэкономических условиях или другие факторы, влияющие на рынок.
- **Ошибки модели:** Прогнозируемая доходность основана на модели линейной регрессии, которая может не учитывать все факторы, влияющие на цену акций. Возможно, выбранная модель недостаточно точно описывает реальность.
- **Изменение чувствительности:** Бета и альфа коэффициенты, рассчитанные за период оценки, могут не быть стабильными и изменяться со временем. Например, чувствительность акций к изменениям на рынке могла измениться после события.
- **Период анализа:** Возможно, период оценки (estimation window) был недостаточно длинным или выбран неудачно, что привело к неверной оценке коэффициентов модели.