# Задание

Для заданного набора данных (по Вашему варианту) постройте модели классификации или регрессии (в зависимости от конкретной задачи, рассматриваемой в наборе данных). Для построения моделей используйте методы 1 и 2 (по варианту для Вашей группы). Оцените качество моделей на основе подходящих метрик качества (не менее двух метрик). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей? Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д.

| Метод 1 | Метод 2 |
|---|---|
| Линейная/логистическая регрессия | Градиентный бустинг |

Набор данных: https://www.kaggle.com/datasets/dhruvildave/currency-exchange-rates

# Ход выполнения работы

## Текстовое описание набора данных

Это исторический набор данных о курсах валют.

Этот набор данных включает в себя в общей сложности 113 валют с обменной стоимостью в различных сочетаниях. Его можно использовать для различных задач, таких как анализ временных рядов и прогнозирование. В наборе данных примерно 1,4 миллиона строк и 7 столбцов.

В датасете есть 2 главных столбца: `slug` и `currency`

Столбец `slug` имеет форму «ABC/PQR», где ABC и PQR — стандартные активные коды валют согласно ISO 4217. валюта содержит код валюты, который является единицей значения в таблицах. Например, если slug — `JPY/INR`, а валюта — INR, это означает, что

$$ 1 JPY = x INP $$
Столбцы:

- `slug` – сравнительная стоимость
- `date` – дата курса валют
- `open` – курс на момент открытия торгов
- `high` – максимальный курс в день торгов
- `low` – минимальный курс в день торгов
- `close` – Курс на момент закрытия торгов
- `currency` – валюта, на которую производится обмен

## Основные характеристики набора данных

Подключаем все необходимые библиотеки

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import matplotlib_inline
import matplotlib.pyplot as plt
from IPython.display import Image
from io import StringIO
import graphviz
import pydotplus
from sklearn.model_selection import train_test_split
%matplotlib inline
%matplotlib inline
sns.set(style="ticks")
from IPython.display import set_matplotlib_formats
matplotlib_inline.backend_inline.set_matplotlib_formats("retina")
```

Подключаем Dataset

In [2]:
```
data = pd.read_csv('forex.csv', sep=",")
```

Размер набора данных

In [3]:
```
data.shape
```

Out[3]:
```
(1453035, 7)
```

Типы колонок

In [4]:
```
data.dtypes
```

Out[4]:
```
slug         object
date         object
open        float64
high        float64
low         float64
close       float64
currency     object
dtype: object
```

Проверяем, есть ли пропущенные значения

In [5]:
```
data.isnull().sum()
```

Out[5]:
```
slug        0
date        0
open        0
high        0
low         0
close       0
currency    0
dtype: int64
```

В наборе данных нет пропусков, следовательно, их обрабатывать не нужно.

Оставим для анализа только первые 2000 строк набора данных

In [6]:
```
data_2000 = data.head(2000)
```

Первые 5 строк датасета

In [7]:
```
data_2000.head(5)
```

Out[7]:

| | slug | date | open | high | low | close | currency |
|---|---|---|---|---|---|---|---|
| **0** | GBP/EGP | 2001-04-10 | 5.58090 | 5.5947 | 5.5947 | 5.5947 | EGP |
| **1** | GBP/EGP | 2001-06-04 | 5.47517 | 5.4939 | 5.4939 | 5.4939 | EGP |
| **2** | GBP/EGP | 2001-08-01 | 5.67990 | 5.6543 | 5.6543 | 5.6543 | EGP |
| **3** | GBP/EGP | 2002-07-29 | 7.21700 | 7.2170 | 7.2170 | 7.2170 | EGP |
| **4** | GBP/EGP | 2003-01-02 | 7.42429 | 7.3899 | 7.3899 | 7.3899 | EGP |

In [8]:
```
total_count = data_2000.shape[0]
print('Всего строк: {}'.format(total_count))
```

```
Всего строк: 2000
```

Настройка отображения графиков

In [9]:
```
# Задание формата графиков для сохранения высокого качества PNG
from IPython.display import set_matplotlib_formats
matplotlib_inline.backend_inline.set_matplotlib_formats("retina")
# Задание ширины графиков, чтобы они помещались на A4
pd.set_option("display.width", 70)
```

# Обрабогтка данных

### Кодирование категориальных признаков

Преобразуем имена, страны, ... в числовые зеачения (label encoding)

In [10]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

In [11]:
```
data_digit = data_2000.copy()
le = LabelEncoder()
    # "slug"
le.fit(data_digit.slug.drop_duplicates())
data_digit.slug = le.transform(data_digit.slug)
    # "date"
le.fit(data_digit.date.drop_duplicates())
data_digit.date = le.transform(data_2000.date)
    # "currency"
le.fit(data_digit.currency.drop_duplicates())
data_digit.currency = le.transform(data_digit.currency)
```

### Масштабирование данных

In [12]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [13]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data_digit)
data_scalled = data_digit.copy()
data_scalled[data_scalled.columns] = sc1_data
data_scalled
```

Out[13]:

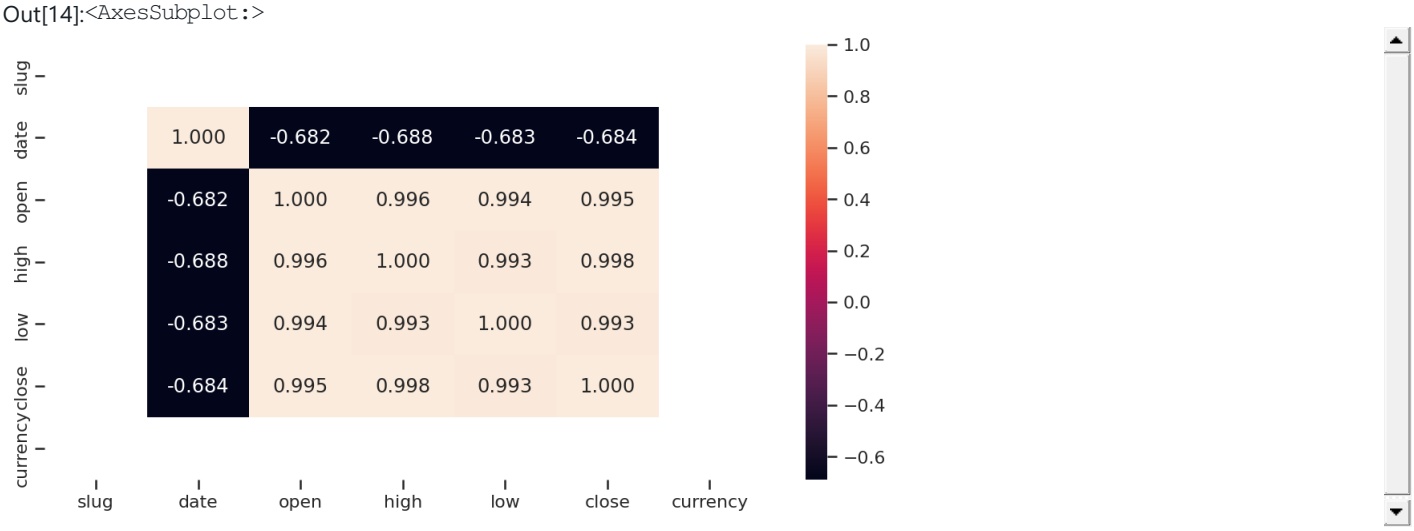
| | slug | date | open | high | low | close | currency |
|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.000000 | 0.015993 | 0.015187 | 0.015519 | 0.015300 | 0.0 |
| **1** | 0.0 | 0.000500 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **2** | 0.0 | 0.001001 | 0.030969 | 0.024167 | 0.024696 | 0.024347 | 0.0 |
| **3** | 0.0 | 0.001501 | 0.263481 | 0.259616 | 0.265292 | 0.261547 | 0.0 |
| **4** | 0.0 | 0.002001 | 0.294837 | 0.285667 | 0.291912 | 0.287792 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **1995** | 0.0 | 0.997999 | 0.610004 | 0.610809 | 0.608813 | 0.609402 | 0.0 |
| **1996** | 0.0 | 0.998499 | 0.611199 | 0.610703 | 0.614386 | 0.610358 | 0.0 |
| **1997** | 0.0 | 0.998999 | 0.609958 | 0.613777 | 0.617358 | 0.609311 | 0.0 |
| **1998** | 0.0 | 0.999500 | 0.616054 | 0.617363 | 0.622515 | 0.615352 | 0.0 |
| **1999** | 0.0 | 1.000000 | 0.617340 | 0.616851 | 0.611646 | 0.616384 | 0.0 |

2000 rows × 7 columns

Построим кореляционную матрицу

In [14]:
```
ig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data_scalled.corr(method='pearson'), ax=ax, annot=True, fmt='.3f')
```

Out[14]:<AxesSubplot:>




# Предсказание целевого признака

Предскажем значение целевого признака `close` .

### Разделение выборки на обучающую и тестовую

In [15]:
```
X = data_scalled.drop(columns='close')
Y = data_scalled['close']
```

Входные данные:

In [16]:
```
X.head()
```

Out[16]:

| | slug | date | open | high | low | currency |
|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.000000 | 0.015993 | 0.015187 | 0.015519 | 0.0 |
| **1** | 0.0 | 0.000500 | 0.000000 | 0.000000 | 0.000000 | 0.0 |
| **2** | 0.0 | 0.001001 | 0.030969 | 0.024167 | 0.024696 | 0.0 |
| **3** | 0.0 | 0.001501 | 0.263481 | 0.259616 | 0.265292 | 0.0 |
| **4** | 0.0 | 0.002001 | 0.294837 | 0.285667 | 0.291912 | 0.0 |

Выходные данные

In [17]:
```
Y.head()
```

Out[17]:
```
0    0.015300
1    0.000000
2    0.024347
3    0.261547
4    0.287792
Name: close, dtype: float64
```

In [18]:
```
X_train,  X_test,  Y_train,  Y_test = train_test_split(X,  Y, random_state = 2022, test_size = 0.1)
```

Входные параметры обучающей выборки

In [19]:
```
X_train.head()
```

Out[19]:

| | slug | date | open | high | low | currency |
|---|---|---|---|---|---|---|
| **1964** | 0.0 | 0.982491 | 0.587117 | 0.582679 | 0.580884 | 0.0 |
| **1510** | 0.0 | 0.755378 | 0.491335 | 0.490124 | 0.494234 | 0.0 |
| **228** | 0.0 | 0.114057 | 1.000000 | 0.994576 | 0.989992 | 0.0 |
| **1189** | 0.0 | 0.594797 | 0.645748 | 0.651083 | 0.632662 | 0.0 |
| **1889** | 0.0 | 0.944972 | 0.644054 | 0.640008 | 0.644840 | 0.0 |

Входные параметры тестовой выборки

In [20]:
```
X_test.head()
```

Out[20]:

| | slug | date | open | high | low | currency |
|---|---|---|---|---|---|---|
| **1018** | 0.0 | 0.509255 | 0.803050 | 0.810007 | 0.794922 | 0.0 |
| **1295** | 0.0 | 0.647824 | 0.382665 | 0.396905 | 0.368308 | 0.0 |
| **643** | 0.0 | 0.321661 | 0.767351 | 0.767067 | 0.708396 | 0.0 |
| **1842** | 0.0 | 0.921461 | 0.616538 | 0.616595 | 0.600206 | 0.0 |
| **1669** | 0.0 | 0.834917 | 0.531360 | 0.532235 | 0.536528 | 0.0 |

Выходные параметры обучающей выборки

In [21]:
```
Y_train.head()
```

Out[21]:
```
1964    0.586224
1510    0.490126
228     0.982241
1189    0.629544
1889    0.643372
Name: close, dtype: float64
```

Выходные параметры тестовой выборки

In [22]:
```
Y_test.head()
```

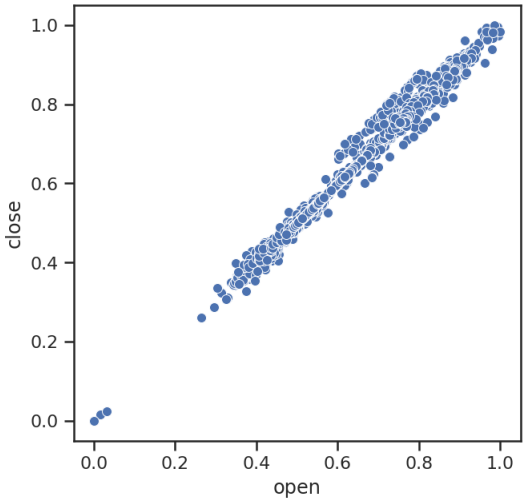
Out[22]:
```
1018    0.809657
1295    0.370350
643     0.759718
1842    0.615656
1669    0.535648
Name: close, dtype: float64
```

### Линейная регрессия

In [23]:
```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_blobs
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [24]:
```
fig, ax = plt.subplots(figsize=(5,5))
sns.scatterplot(ax=ax, x=X['open'], y=Y)
```

Out[24]:<AxesSubplot:xlabel='open', ylabel='close'>




Обучим линейную регрессию

In [25]:
```
reg1 = LinearRegression().fit(X, Y)
```

In [26]:
```
Y_pred_1 = reg1.predict(X_test)
```

Проверим результат на 2 метриках

In [27]:
```
mean_absolute_error(Y_test, Y_pred_1), mean_squared_error(Y_test, Y_pred_1)
```

Out[27]:(0.007128937368944646, 0.00013529152997477576)

### Градиентный бустинг

In [28]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
```

Обучим регрессор на 4 деревьях

In [29]:
```
ab1 = AdaBoostRegressor(n_estimators=4, random_state=2022)
ab1.fit(X, Y)
```

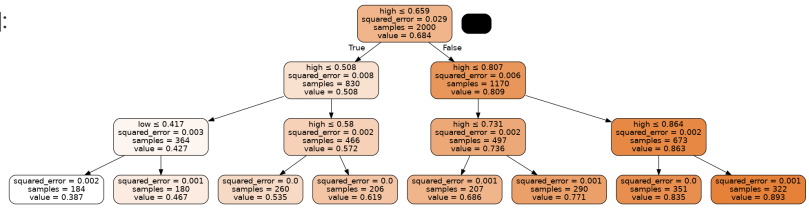
Out[29]:AdaBoostRegressor(n_estimators=4, random_state=2022)

Визуализируем обучающие деревья

In [30]:
```
# Визуализация дерева
def get_png_tree(tree_model_param, feature_names_param):
    dot_data = StringIO()
    export_graphviz(tree_model_param, out_file=dot_data, feature_names=feature_names_param,
                    filled=True, rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph.create_png()
```

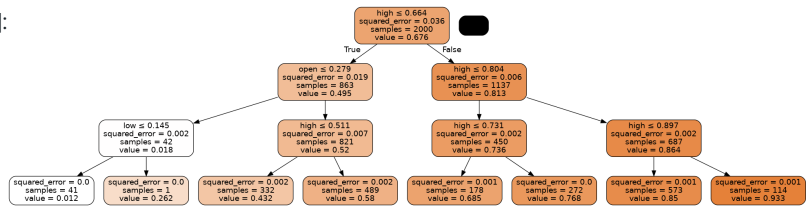
In [31]:
```
Image(get_png_tree(ab1.estimators_[0], X.columns), width="500")
```

Out[31]:



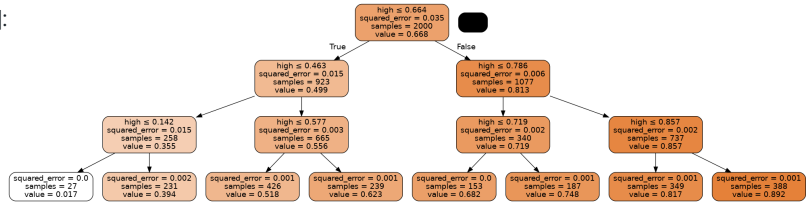

In [32]:
```
Image(get_png_tree(ab1.estimators_[1], X.columns), width="500")
```

Out[32]:



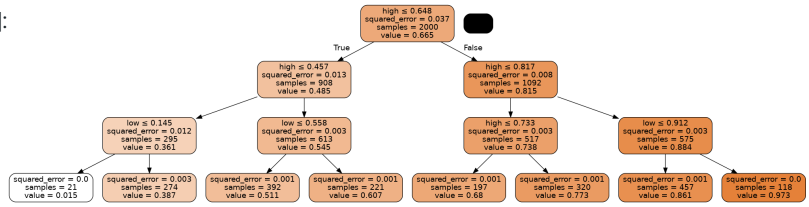

In [33]:
```
Image(get_png_tree(ab1.estimators_[2], X.columns), width="500")
```

Out[33]:




In [34]:
```
Image(get_png_tree(ab1.estimators_[3], X.columns), width="500")
```

Out[34]:




In [35]:
```
regressor = AdaBoostRegressor(n_estimators=4, random_state=2022)
regressor.fit(X_train, Y_train)
y_pred = regressor.predict(X_test)
```

In [36]:
```
print('Mean Absolute Error:', mean_absolute_error(Y_test, y_pred))
print('Mean Squared Error:', mean_squared_error(Y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(Y_test, y_pred)))
```

```
Mean Absolute Error: 0.018349098326094767
Mean Squared Error: 0.0005958052928923877
Root Mean Squared Error: 0.024409123148781638
```

Как видно, линейная регрессия показало более лучшие результаты, чем градиентный бустинг.

In [ ]: