# Импорт библиотек

from math import \* # вкл. в себя ключает математические функции и константы

import itertools # используется для работы с итерациями

import pandas as pd # для анализа данных

import seaborn as sns # для визуализации данных на основе Matplotlib

import numpy as np # для работы с многомерными массивами

import matplotlib.pyplot as plt # для создания визуализаций

import warnings # модуль для управления предупреждениями (сделан вывод более чистым)

warnings.filterwarnings("ignore")

# Чтение данных таблицы

df\_d=pd.read\_excel('C:/Users/юзер/Desktop/Курсовая/Таблицы датасеты/Доллар табл.xlsx')

# Создаем индекс из даты

df\_d['data'] = pd.to\_datetime(df\_d['data'])

# Устанавливаем data как индекс

df\_d.set\_index('data', inplace=True)

# Сортировка датафреймов по индексу (по датам) в порядке возрастания

df\_d.sort\_index(ascending=True, inplace=True)

df\_d.info() # Выводит общую информацию о датафрейме

df\_d.describe() # Выводит статистические характеристики числовых столбцов

'''

В процессе анализа столбцов таблицы дадасета выявлено, что есть пропущенные значения в столбцах ключевой ставки (key interest rate) и инфляции (inflation). Поэтому, пропущенные значения будут заполнены методом ffill + интерполяцией для ключ. ставки и минтерполяцией для инфляции

'''

df\_d['key interest rate'].fillna(method='bfill', inplace=True) # заполнение первым известным значением

df\_d['key interest rate'] = df\_d['key interest rate'].interpolate(method='linear') # интерполяция для ключевой ставки

df\_d['inflation'].fillna(method='bfill', inplace=True)

df\_d['inflation'] = df\_d['inflation'].interpolate(method='linear') # интерполяция для инфляции

df\_d.head(20) # вывод 1-ых 20 значений датафрейма

# смотрим информацию о датафрейме после замены пропущ. значений

df\_d.info()

df\_d.describe()

df\_d = df\_d.drop(columns=['nominal']) # удаляем столбец с номиналом (там везде 1)

df\_d = df\_d.drop(columns=['name of cur']) # удаляем столбец с наименованием валюты (везде 'доллар')

'''

Проверим значение столбцов на выбросы

В дальнейшем у нас будет 2 датафрейма:

df\_d1 - для работы с значениями без выбросов

df\_d2 - выбросы остаются

'''

df\_d1 = df\_d.copy()

df\_d2 = df\_d.copy()

# Готовим "обрезанные" датафреймы для проверки

df\_d\_check1 = df\_d1[['curs usd', 'gold', 'silver', 'palladium', 'brent (in usd)', 'curs eur']] # проверяем цены

df\_d\_check2 = df\_d1[['key interest rate', 'inflation']] # проверяем ключ. ставку и инфляцию

# Графики 6 в 1. Для начала изобразим динамику изменения цены (для курса доллара, цены на драг. металлы, нефть, курс евро)

plt.figure(figsize=(15, 10))

for i, col in enumerate(df\_d\_check1.columns, 1):

plt.subplot(2, 3, i)

sns.lineplot(x=df\_d\_check1.index, y=df\_d\_check1[col])

plt.title(f'Динамика {col}')

plt.xlabel('Дата')

plt.ylabel('Цена')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

# Динамика изменения ключевой ставки и инфляции

plt.figure(figsize=(15, 10))

for i, col in enumerate(df\_d\_check2.columns, 1):

plt.subplot(2, 1, i)

sns.lineplot(x=df\_d\_check2.index, y=df\_d\_check2[col])

plt.title(f'Динамика {col}')

plt.xlabel('Дата')

plt.ylabel('Размер')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

'''

Мы получили динамику изменения цен, инфляции и т.п. Теперь строим графики ящики с усами, чтоб визуально ознакомится с выбросами в нашем наборе данных

'''

# Создаем общую фигуру с размером 15x10

plt.figure(figsize=(15, 10))

# Первые 2 графика (для ключевой ставки и инфляции)

for i, col in enumerate(df\_d\_check2.columns, 1):

plt.subplot(2, 4, i)

sns.boxplot(x=df\_d\_check2[col])

plt.title(f'Ящик с усами для {col}')

# Остальные 6 графиков (для цен)

for i, col in enumerate(df\_d\_check1.columns, 1):

plt.subplot(2, 4, i + 2)

sns.boxplot(x=df\_d\_check1[col])

plt.title(f'Ящик с усами для {col}')

plt.tight\_layout()

plt.show()

# Проверка выбросов для курса доллара

Mcd = df\_d1['curs usd'].mean() # ср. знач

Scd = df\_d1['curs usd'].std() # cр. кв. отклонение

print(f"Значение курса доллара не считается выбросом на интервале ({Mcd-3\*Scd}, {Mcd+3\*Scd})")

# Проверка выбросов для цены на золото

Mg = df\_d1['gold'].mean()

Sg = df\_d1['gold'].std()

print(f"Значение цены золота не считается выбросом на интервале ({Mg-3\*Sg}, {Mg+3\*Sg})")

# Проверка выбросов для цены на серебро

Mss = df\_d1['silver'].mean()

Sss = df\_d1['silver'].std()

print(f"Значение цены серебра не считается выбросом на интервале ({Mss-3\*Sss}, {Mss+3\*Sss})")

# Проверка выбросов для цены на палладий

Mpp = df\_d1['palladium'].mean()

Spp = df\_d1['palladium'].std()

print(f"Значение цены палладия не считается выбросом на интервале ({Mpp-3\*Spp}, {Mpp+3\*Spp})")

# Проверка выбросов для цены на нефть Brent

Mb = df\_d1['brent (in usd)'].mean()

Sb = df\_d1['brent (in usd)'].std()

print(f"Значение цены на нефть Brent не считается выбросом на интервале ({Mb-3\*Sb}, {Mb+3\*Sb})")

# Проверка выбросов для курса евро

Mce = df\_d1['curs eur'].mean()

Sce = df\_d1['curs eur'].std()

print(f"Значение курса доллара не считается выбросом на интервале ({Mce-3\*Sce}, {Mce+3\*Sce})")

# Проверка выбросов для кл. ставки

Mkir = df\_d1['key interest rate'].mean()

Skir = df\_d1['key interest rate'].std()

print(f"Значение ключевой ставки не считается выбросом на интервале ({Mkir-3\*Skir}, {Mkir+3\*Skir})")

# Проверка выбросов для инфляции

Minf = df\_d1['inflation'].mean()

Sinf = df\_d1['inflation'].std()

print(f"Значение инфляции не считается выбросом на интервале ({Minf-3\*Sinf}, {Minf+3\*Sinf})")

# Проверка выбросов методом IQR

# Проверка выбросов для курса доллара

Q1\_usd = df\_d1['curs usd'].quantile(0.25)

Q3\_usd = df\_d1['curs usd'].quantile(0.75)

IQR\_usd = Q3\_usd - Q1\_usd

print(f"Значение курса доллара не считается выбросом на интервале ({Q1\_usd-1.5\*IQR\_usd}, {Q3\_usd+1.5\*IQR\_usd})")

# Проверка выбросов для цены на золото

Q1\_gold = df\_d1['gold'].quantile(0.25)

Q3\_gold = df\_d1['gold'].quantile(0.75)

IQR\_gold = Q3\_gold - Q1\_gold

print(f"Значение цены золота не считается выбросом на интервале ({Q1\_gold-1.5\*IQR\_gold}, {Q3\_gold+1.5\*IQR\_gold})")

# Проверка выбросов для цены на серебро

Q1\_silver = df\_d1['silver'].quantile(0.25)

Q3\_silver = df\_d1['silver'].quantile(0.75)

IQR\_silver = Q3\_silver - Q1\_silver

print(f"Значение цены серебра не считается выбросом на интервале ({Q1\_silver-1.5\*IQR\_silver}, {Q3\_silver+1.5\*IQR\_silver})")

# Проверка выбросов для цены на палладий

Q1\_palladium = df\_d1['palladium'].quantile(0.25)

Q3\_palladium = df\_d1['palladium'].quantile(0.75)

IQR\_palladium = Q3\_palladium - Q1\_palladium

print(f"Значение цены палладия не считается выбросом на интервале ({Q1\_palladium-1.5\*IQR\_palladium}, {Q3\_palladium+1.5\*IQR\_palladium})")

# Проверка выбросов для цены на нефть Brent

Q1\_brent = df\_d1['brent (in usd)'].quantile(0.25)

Q3\_brent = df\_d1['brent (in usd)'].quantile(0.75)

IQR\_brent = Q3\_brent - Q1\_brent

print(f"Значение цены на нефть Brent не считается выбросом на интервале ({Q1\_brent-1.5\*IQR\_brent}, {Q3\_brent+1.5\*IQR\_brent})")

# Проверка выбросов для курса евро

Q1\_eur = df\_d1['curs eur'].quantile(0.25)

Q3\_eur = df\_d1['curs eur'].quantile(0.75)

IQR\_eur = Q3\_eur - Q1\_eur

print(f"Значение курса евро не считается выбросом на интервале ({Q1\_eur-1.5\*IQR\_eur}, {Q3\_eur+1.5\*IQR\_eur})")

# Проверка выбросов для кл. ставки

Q1\_keyrate = df\_d1['key interest rate'].quantile(0.25)

Q3\_keyrate = df\_d1['key interest rate'].quantile(0.75)

IQR\_keyrate = Q3\_keyrate - Q1\_keyrate

print(f"Значение ключевой ставки не считается выбросом на интервале ({Q1\_keyrate-1.5\*IQR\_keyrate}, {Q3\_keyrate+1.5\*IQR\_keyrate})")

# Проверка выбросов для инфляции

Q1\_inf = df\_d1['inflation'].quantile(0.25)

Q3\_inf = df\_d1['inflation'].quantile(0.75)

IQR\_inf = Q3\_inf - Q1\_inf

print(f"Значение инфляции не считается выбросом на интервале ({Q1\_inf-1.5\*IQR\_inf}, {Q3\_inf+1.5\*IQR\_inf})")

# Заменяем выбросы в цене золота на значение медианы

# Относительно остальных драг. металлов, золото обладает наибольшей стабильностью в плане изменения цены

# Поэтому все выбросы в цене золота заменены на медиану

lower\_gold = Q1\_gold - 1.5 \* IQR\_gold

upper\_gold = Q3\_gold + 1.5 \* IQR\_gold

gold\_median = df\_d1['gold'].median() # медиана

df\_d1['gold'] = np.where( # Заменяем выбросы в цене золота на значение медианы

(df\_d1['gold'] < lower\_gold) | (df\_d1['gold'] > upper\_gold), # если значение меньше нижней или больше верхней границы

gold\_median, # то заменяем его на медиану

df\_d1['gold'] # иначе оставляем как есть

)

# Заменем выбросы в цене серебра

# Серебро более волатильно (нестабильно), чем золото. Его цена чаще резко меняется из-за промышленного спроса

df\_d1['silver'] = df\_d1['silver'].clip(Q1\_silver - 1.5\*IQR\_silver, Q3\_silver + 1.5\*IQR\_silver) # значения за границами как бы "подрезаются"

# все значения меньше нижней границы (Q1 - 1.5\*IQR) заменяются на эту границу

# все значения больше верхней границы (Q3 + 1.5\*IQR) заменяются на эту границу

# значения внутри границ остаются без изменений

# Аналогичная (почти), что и у серебра, замена выбросов в цене палладия

# Палладий очень нестабилен нестабилен по сравнению с серебром и золотом

# Резкие взлеты и падения палладия – его естественное поведение из-за дефицита на рынке

df\_d1['palladium'] = df\_d1['palladium'].clip(Q1\_palladium - 1\*IQR\_palladium, Q3\_palladium + 1\*IQR\_palladium)

# Такие жесткие границы (1\*IQR вместо стандартных 1.5\*IQR) защищают от перекосов, но не "убивают" волатильность

# Усредняем выбросы в значениях ключ. ставки и инфляции

for col in ['key interest rate', 'inflation']:

Q1\_n = df\_d1[col].quantile(0.25) # 1 квартиль

Q3\_n = df\_d1[col].quantile(0.75) # 3 квартиль

IQR\_n = Q3\_n - Q1\_n # межквартильный размах

low\_bound\_n = Q1\_n - 1.5 \* IQR\_n # нижняя граница

up\_bound\_n = Q3\_n + 1.5 \* IQR\_n # верхняя граница

df\_d1[col] = df\_d1[col].clip(low\_bound\_n, up\_bound\_n)

# Рисуем графики, чтоб проверить устранение выбросов

plt.figure(figsize=(15, 10))

for i, col in enumerate(df\_d1.columns, 1):

plt.subplot(2, 4, i)

sns.boxplot(x=df\_d1[col])

plt.title(f'Ящик с усами для {col}')

plt.tight\_layout()

plt.show()

# Строим тепловую карту корреляций (для 1-ой модели)

plt.figure(figsize=(12, 8))

corr\_matrix1 = df\_d1.corr(numeric\_only=True) # Только числовые колонки

sns.heatmap(corr\_matrix1, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

plt.title("Матрица корреляции (без выбросов)")

plt.show()

# для модели с выбросами (df\_d2)

plt.figure(figsize=(12, 8))

corr\_matrix2 = df\_d2.corr(numeric\_only=True)

sns.heatmap(corr\_matrix2, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

plt.title("Матрица корреляции (с выбросами)")

plt.show()

# Функция для добавления лаговых признаков

def add\_lags(df, target\_column, n\_lags):

for i in range(1, n\_lags + 1):

df[f'{target\_column}\_lag{i}'] = df[target\_column].shift(i)

return df

# Добавляем лаговые признаки (за 7 дней)

n\_lags = 7

df\_d1 = add\_lags(df\_d1, 'curs usd', n\_lags)

df\_d2 = add\_lags(df\_d2, 'curs usd', n\_lags)

# Удаляем строки с NaN (на всякий случай, если они на этапе обработки данных вдруг не удалились)

df\_d1 = df\_d1.dropna()

df\_d2 = df\_d2.dropna()

# Разделение на обучающую и тестовую выборки (80% - обучение, 20% - тест)

def split\_data(df):

train\_size = int(len(df) \* 0.8)

train\_data = df.iloc[:train\_size]

test\_data = df.iloc[train\_size:]

X\_train = train\_data.drop('curs usd', axis=1)

y\_train = train\_data['curs usd']

X\_test = test\_data.drop('curs usd', axis=1)

y\_test = test\_data['curs usd']

return X\_train, X\_test, y\_train, y\_test

# Разделяем данные

X\_train\_d1, X\_test\_d1, y\_train\_d1, y\_test\_d1 = split\_data(df\_d1)

X\_train\_d2, X\_test\_d2, y\_train\_d2, y\_test\_d2 = split\_data(df\_d2)

from sklearn.linear\_model import LinearRegression # Импорт модели линейной регрессии для построения линейной зависимости между признаками и целевой переменной

from sklearn.ensemble import RandomForestRegressor # Импорт ансамблевой модели случайного леса для прогнозирования на основе множества деревьев решений

from sklearn.svm import SVR # Импорт метода опорных векторов для регрессии, эффективного при работе с нелинейными зависимостями

from tensorflow.keras.models import Sequential # Импорт последовательной модели нейронной сети from tensorflow.keras.layers import Dense # Импорт полносвязного слоя нейронной сети для построения многослойного перцептронаиз библиотеки Keras (входит в TensorFlow)

# Модели для df\_d1

# Линейная регрессия

lr\_model\_d1 = LinearRegression() # Создаём объект модели линейной регрессии

lr\_model\_d1.fit(X\_train\_d1, y\_train\_d1) # обучаем модель линейной регрессии на обучающих данных (y\_train\_d1 — целевая переменная (курс доллара curs usd))

y\_pred\_lr\_d1 = lr\_model\_d1.predict(X\_test\_d1) # Делаем предсказания на тестовой выборке X\_test\_d1 с помощью обученной модели

# Случайный лес

rf\_model\_d1 = RandomForestRegressor(n\_estimators=500, random\_state=42) # Создаём объект модели случайного леса rf\_model\_d1 с помощью класса RandomForestRegressor

# (n\_estimators=500 — число деревьев в "лесу" (чем больше, тем точнее, но дольше обучение), random\_state=42 — случайное начальное состояние)

rf\_model\_d1.fit(X\_train\_d1, y\_train\_d1) # обучаем модель

y\_pred\_rf\_d1 = rf\_model\_d1.predict(X\_test\_d1) # предсказания

# SVR (метод опорных векторов)

svr\_model\_d1 = SVR(kernel='rbf') # Создаем модель SVR (kernel='rbf' — используем радиально-базисное ядро (RBF), которое хорошо подходит для нелинейных зависимостей)

# SVR пытается найти гиперплоскость, которая наилучшим образом предсказывает значения, минимизируя ошибку

svr\_model\_d1.fit(X\_train\_d1, y\_train\_d1) # обучаем модель

y\_pred\_svr\_d1 = svr\_model\_d1.predict(X\_test\_d1) # предсказания

# Нейросеть

nn\_model\_d1 = Sequential([ # Создаём объект нейросети nn\_model\_d1 с помощью класса Sequential

# sequential означает что слои нейросети добавляются последовательно

Dense(64, activation='relu', input\_shape=(X\_train\_d1.shape[1],)), # первый слой

# activation='relu' — функция активации relu (Rectified Linear Unit), которая делает модель нелинейной

# То есть, если значение положительное — оно остаётся, если отрицательное — становится нулём

# Это помогает нейронной сети учиться нелинейным зависимостям в данных

# input\_shape=(X\_train\_d1.shape[1],) — размер входных данных (число признаков в X\_train\_d1)

# X\_train\_d1.shape[1] — это число признаков (колонок) в этих данных

# Например, если 10 признаков (столбцов), будет input\_shape=(10,)

# Круглая скобка с запятой (10,) — это кортеж с одним элементом, как требует Keras

Dense(32, activation='relu'), # второй слой

Dense(1) # слой с одним нейроном (без активации), который выдаёт предсказание (число — курс доллара)

])

'''

X\_train\_d1.shape[0] — это сколько строк (примеров/наблюдений)

X\_train\_d1.shape[1] — это сколько колонок (признаков)

'''

nn\_model\_d1.compile(optimizer='adam', loss='mse') # компиляция модели

# optimizer='adam' — используем оптимизатор Adam

# loss='mse' — функция потерь "среднеквадратичная ошибка"

nn\_model\_d1.fit(X\_train\_d1, y\_train\_d1, epochs=50, batch\_size=32, verbose=0) # обучение нейросетевой модели

'''

Обучаем нейронную сеть:

epochs=50 — сеть обучается в течение 50 проходов по всем данным

batch\_size=32 — данные разбиваются на батчи по 32 примера чтобы ускорить обучение

verbose=0 — отключаем вывод прогресса обучения (чтобы не загромождать вывод всяким хламом)

'''

y\_pred\_nn\_d1 = nn\_model\_d1.predict(X\_test\_d1, verbose=0).flatten() # предсказание

# .flatten() — преобразуем массив из формы (n\_samples, 1) в одномерный массив (n\_samples,) для удобства работы с другими моделями

# Аналогично для df\_d2

# Линейная регрессия

lr\_model\_d2 = LinearRegression()

lr\_model\_d2.fit(X\_train\_d2, y\_train\_d2)

y\_pred\_lr\_d2 = lr\_model\_d2.predict(X\_test\_d2)

# Случайный лес

rf\_model\_d2 = RandomForestRegressor(n\_estimators=500, random\_state=42)

rf\_model\_d2.fit(X\_train\_d2, y\_train\_d2)

y\_pred\_rf\_d2 = rf\_model\_d2.predict(X\_test\_d2)

# SVR

svr\_model\_d2 = SVR(kernel='rbf')

svr\_model\_d2.fit(X\_train\_d2, y\_train\_d2)

y\_pred\_svr\_d2 = svr\_model\_d2.predict(X\_test\_d2)

# Нейросеть

nn\_model\_d2 = Sequential([

Dense(64, activation='relu', input\_shape=(X\_train\_d2.shape[1],)),

Dense(32, activation='relu'),

Dense(1)

])

nn\_model\_d2.compile(optimizer='adam', loss='mse')

nn\_model\_d2.fit(X\_train\_d2, y\_train\_d2, epochs=50, batch\_size=32, verbose=0)

y\_pred\_nn\_d2 = nn\_model\_d2.predict(X\_test\_d2, verbose=0).flatten()

# Комбинированные предсказания для df\_d1

y\_pred\_combined\_d1 = (y\_pred\_lr\_d1 + y\_pred\_rf\_d1 + y\_pred\_svr\_d1 + y\_pred\_nn\_d1) / 4

print(y\_pred\_combined\_d1)

# Комбинированные предсказания для df\_d2

y\_pred\_combined\_d2 = (y\_pred\_lr\_d2 + y\_pred\_rf\_d2 + y\_pred\_svr\_d2 + y\_pred\_nn\_d2) / 4

print(y\_pred\_combined\_d2)

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_error # импорт метрик

def calculate\_metrics(y\_true, predictions, model\_names): # определение функции

for y\_pred, name in zip(predictions, model\_names): # цикл по прогнозам и именам моделей

r2 = r2\_score(y\_true, y\_pred) # расчет R^2

mae = mean\_absolute\_error(y\_true, y\_pred) # расчет MAE

mape = mean\_absolute\_percentage\_error(y\_true, y\_pred) \* 100 # расчет MAPE в процентах

mse = mean\_squared\_error(y\_true, y\_pred) # расчет MSE

rmse = sqrt(mse) # расчет RMSE

print(f"Метрики для {name}:") # вывод заголовка

print(f"R²: {r2:.4f}") # вывод R^2

print(f"MAE: {mae:.4f}") # вывод MAE

print(f"MAPE: {mape:.2f}%") # вывод MAPE

print(f"MSE: {mse:.4f}") # вывод MSE

print(f"RMSE: {rmse:.4f}") # вывод RMSE

print() # добавление пустой строки

# Список прогнозов и имен моделей для df\_d1

predictions\_d1 = [y\_pred\_lr\_d1, y\_pred\_rf\_d1, y\_pred\_svr\_d1, y\_pred\_nn\_d1, y\_pred\_combined\_d1] # список прогнозов для df\_d1

model\_names\_d1 = ["линейной регрессии", "случайного леса", "SVR (метод опорных векторов)", "нейронной сети", "комбинированной модели"] # список имен моделей

# Вычисление метрик для df\_d1

print('Расчет метрик для df\_d1:\n')

calculate\_metrics(y\_test\_d1, predictions\_d1, model\_names\_d1) # вызов функции для df\_d1

print('Расчет метрик для df\_d2:\n')

predictions\_d2 = [y\_pred\_lr\_d2, y\_pred\_rf\_d2, y\_pred\_svr\_d2, y\_pred\_nn\_d2, y\_pred\_combined\_d2] # список прогнозов для df\_d2

model\_names\_d2 = ["линейной регрессии", "случайного леса", "SVR (метод опорных векторов)", "нейронной сети", "комбинированной модели"] # список имен моделей

# Вычисление метрик для df\_d2

calculate\_metrics(y\_test\_d2, predictions\_d2, model\_names\_d2) # вызов функции для df\_d2