main.py)

# Импортируем необходимые библиотеки

import asyncio # Для асинхронного запуска и работы с событиями

import logging # Для логирования действий бота

from aiogram import Bot, Dispatcher # Основные компоненты aiogram для создания бота

from aiogram.types import BotCommand # Для настройки списка команд в интерфейсе Telegram

from dotenv import load\_dotenv # Для загрузки переменных окружения из .env файла

import os # Для работы с операционной системой (например, получение переменных окружения)

from handlers import commands # Подключаем обработчики команд из отдельного модуля

# Настройка логирования: запись в файл и вывод в консоль

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

handlers=[

logging.FileHandler('bot.log'), # Логи будут сохраняться в файл bot.log

logging.StreamHandler() # И одновременно выводиться в консоль

]

)

logger = logging.getLogger(\_\_name\_\_) # Создаем логгер для текущего модуля

# Загружаем переменные окружения из файла .env

load\_dotenv()

API\_TOKEN = os.getenv('BOT\_TOKEN') # Получаем токен бота

# Инициализируем бота и диспетчера

bot = Bot(token=API\_TOKEN) # Создаем объект бота с токеном

dp = Dispatcher() # Диспетчер будет обрабатывать входящие сообщения

# Регистрация обработчиков команд

dp.include\_router(commands.router) # Подключаем роутер с обработчиками команд

async def set\_commands(bot: Bot):

# Функция устанавливает список команд, отображаемых в интерфейсе Telegram

commands\_list = [

BotCommand(command="/start", description="Запустить/перезапустить бота"),

BotCommand(command="/predict", description="Прогноз курса доллара"),

BotCommand(command="/metrics", description="Метрики качества модели"),

BotCommand(command="/last", description="Последний известный курс"),

BotCommand(command="/help", description="Список команд"),

BotCommand(command="/correlation", description="Матрица корреляции признаков")

]

await bot.set\_my\_commands(commands\_list) # Устанавливаем команды

logger.info("Меню команд успешно настроено")

async def main():

# Главная функция для запуска бота

logger.info("Бот запущен") # Пишем в лог о начале работы

await set\_commands(bot) # Настраиваем меню команд

await dp.start\_polling(bot) # Запускаем процесс опроса сообщений от Telegram

# Точка входа в программу

if \_\_name\_\_ == '\_\_main\_\_':

asyncio.run(main()) # Запускаем асинхронную главную функцию

handlers/commands.py)

# Импорт необходимых библиотек

import pandas as pd # Для работы с табличными данными

import numpy as np # Для математических операций

import logging # Для логирования событий

import matplotlib.pyplot as plt # Для визуализации данных

import seaborn as sns # Для создания статистических графиков

import os # Для работы с файловой системой

from aiogram import Router, types # Фреймворк для создания Telegram бота

from aiogram.filters import Command # Для обработки команд бота

from aiogram.types import Message, ReplyKeyboardRemove, FSInputFile # Типы сообщений и вложений

from keyboards.reply import get\_days\_keyboard # Кастомная клавиатура для выбора дней

from models.models import predict\_combined, calculate\_metrics # Функции для прогнозирования и оценки модели

import joblib # Для загрузки сохраненных моделей

from datetime import datetime, timedelta # Для работы с датами

# Настройка логирования для отслеживания событий в боте

logger = logging.getLogger(\_\_name\_\_)

# Создаем роутер для обработчиков сообщений

router = Router()

# Загрузка данных и scaler из файлов

data = pd.read\_csv('latest\_data.csv', index\_col='data', parse\_dates=True) # Загрузка исторических данных

scaler = joblib.load('scaler.joblib') # Загрузка scaler для нормализации данных

# Колонки для предсказания (признаки модели)

FEATURES = ['gold', 'silver', 'palladium', 'key interest rate', 'inflation',

'curs eur', 'brent (in usd)', 'curs usd\_lag1', 'curs usd\_lag2',

'curs usd\_lag3', 'curs usd\_lag4', 'curs usd\_lag5', 'curs usd\_lag6',

'curs usd\_lag7']

# Основные признаки для корреляции (без лаговых признаков)

CORRELATION\_FEATURES = ['curs usd', 'gold', 'silver', 'palladium',

'key interest rate', 'inflation', 'curs eur', 'brent (in usd)']

# Функция проверки торгового дня

def is\_trading\_day(date):

# Получаем день недели (0 = понедельник, 6 = воскресенье)

weekday = date.weekday()

# Считаем воскресенье (6) и понедельник (0) неторговыми днями

if weekday in [0, 6]:

logger.info(f"Дата {date} неторговая (день недели: {weekday})")

return False

# Если дата в прошлом и присутствует в данных, проверяем её

trading\_dates = pd.to\_datetime(data.index).date

if date in trading\_dates:

logger.info(f"Дата {date} найдена в данных, считается торговой")

return True

# Для будущих дат предполагаем, что день торговый, если не выходной

logger.info(f"Дата {date} будущая, считается торговой")

return True

# Обработчик команды /start - приветственное сообщение

@router.message(Command('start'))

async def send\_welcome(message: Message):

logger.info(f"Получена команда /start от пользователя {message.from\_user.id}")

await message.reply(

"Привет! Я бот для прогнозирования курса доллара США к рублю. "

"Мои команды:\n"

"/predict - прогноз курса доллара на выбранное количество дней (или просто введите число от 1 до 7)\n"

"/metrics - посмотреть метрики качества\n"

"/last - последний курс\n"

"/correlation - матрица корреляции признаков\n"

"/help - список команд и информация о боте",

reply\_markup=ReplyKeyboardRemove()

)

# Обработчик команды /help - информация о командах

@router.message(Command('help'))

async def send\_help(message: Message):

logger.info(f"Получена команда /help от пользователя {message.from\_user.id}")

await message.reply(

"Доступные команды:\n"

"/start - запустить/перезапустить бота\n"

"/predict - прогноз курса доллара на выбранное количество дней (или просто введите число от 1 до 7)\n"

"/metrics - метрики качества модели\n"

"/last - последний известный курс\n"

"/correlation - матрица корреляции признаков\n"

"/help - этот список\n\n"

"Я бот, который строит прогнозы курса USD к RUB на 1–7 дней вперед...",

reply\_markup=ReplyKeyboardRemove()

)

# Обработчик команды /correlation - отправка матрицы корреляции

@router.message(Command('correlation'))

async def send\_correlation\_matrix(message: Message):

logger.info(f"Получена команда /correlation от пользователя {message.from\_user.id}")

try:

# Загружаем полные данные из файла

df = pd.read\_csv('df\_d2.csv', index\_col='data', parse\_dates=True)

# Вычисляем матрицу корреляции для выбранных признаков

corr\_matrix = df[CORRELATION\_FEATURES].corr()

# Создаем тепловую карту корреляции

plt.figure(figsize=(10, 8))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt='.2f',

square=True, cbar=True, annot\_kws={'size': 8})

plt.title('Correlation Matrix')

plt.tight\_layout()

# Сохраняем изображение во временный файл

image\_path = 'correlation\_matrix.png'

plt.savefig(image\_path)

plt.close()

# Проверяем существование файла перед отправкой

if not os.path.exists(image\_path):

logger.error(f"Файл {image\_path} не найден")

await message.reply("Ошибка: не удалось создать изображение...")

return

# Отправляем изображение пользователю

await message.reply\_photo(

photo=FSInputFile(image\_path),

caption="Матрица корреляции признаков",

reply\_markup=ReplyKeyboardRemove()

)

# Отправляем описание признаков

description = "Описание признаков:\n- curs usd: Курс доллара США к рублю\n..."

await message.answer(description, reply\_markup=ReplyKeyboardRemove())

# Удаляем временный файл после отправки

os.remove(image\_path)

logger.info(f"Матрица корреляции успешно отправлена")

except Exception as e:

logger.error(f"Ошибка при генерации матрицы корреляции: {str(e)}")

await message.reply("Произошла ошибка при генерации матрицы корреляции...")

# Обработчик команды /last - отправка последнего известного курса

@router.message(Command('last'))

async def send\_last(message: Message):

logger.info(f"Получена команда /last от пользователя {message.from\_user.id}")

last\_date = data.index[-1].strftime('%Y-%m-%d')

last\_curs = data['curs usd'].iloc[-1]

await message.reply(

f"Последний курс доллара на {last\_date}: {last\_curs:.2f} руб.",

reply\_markup=ReplyKeyboardRemove()

)

# Обработчик команды /metrics - отправка метрик модели

@router.message(Command('metrics'))

async def send\_metrics(message: Message):

logger.info(f"Получена команда /metrics от пользователя {message.from\_user.id}")

# Берем последние 100 записей для тестирования

X\_test = data.iloc[-100:][FEATURES]

y\_true = data.iloc[-100:]['curs usd']

y\_pred = predict\_combined(X\_test) # Получаем предсказания

metrics = calculate\_metrics(y\_true, y\_pred) # Вычисляем метрики

# Формируем ответ с метриками

response = (

"Метрики качества комбинированной модели:\n"

f"R²: {metrics['R²']:.4f}\n"

f"MAE: {metrics['MAE']:.4f}\n"

f"MAPE: {metrics['MAPE']:.2f}%\n"

f"MSE: {metrics['MSE']:.4f}\n"

f"RMSE: {metrics['RMSE']:.4f}"

)

await message.reply(response, reply\_markup=ReplyKeyboardRemove())

# Обработчик команды /predict - запуск процесса прогнозирования

@router.message(Command('predict'))

async def predict\_curs(message: Message):

logger.info(f"Получена команда /predict от пользователя {message.from\_user.id}")

await message.reply(

"Выберите день для прогноза (1-7 дней)...",

reply\_markup=get\_days\_keyboard() # Показываем клавиатуру с выбором дней

)

# Обработчик выбора количества дней для прогноза

@router.message(lambda message: message.text in [str(i) for i in range(1, 8)])

async def process\_days(message: Message):

logger.info(f"Пользователь {message.from\_user.id} выбрал прогноз на {message.text} дней")

days = int(message.text)

last\_data = data.iloc[-1][FEATURES].copy() # Берем последние доступные данные

predictions = [] # Список для хранения прогнозов

# Прогнозируем на каждый день отдельно

for day in range(1, days + 1):

pred\_date = data.index[-1] + timedelta(days=day)

pred\_date\_str = pred\_date.strftime('%Y-%m-%d')

# Пропускаем неторговые дни

if not is\_trading\_day(pred\_date.date()):

predictions.append(f"На {pred\_date\_str} не было торгов.")

continue

# Подготавливаем данные и делаем прогноз

data\_scaled = scaler.transform(last\_data.values.reshape(1, -1))

pred = predict\_combined(last\_data.values.reshape(1, -1))[0]

predictions.append(f"Прогноз на {pred\_date\_str}: {pred:.2f} руб.")

# Обновляем лаговые переменные для следующего дня

last\_data['curs usd\_lag7'] = last\_data['curs usd\_lag6']

last\_data['curs usd\_lag6'] = last\_data['curs usd\_lag5']

# ... (аналогично для других лагов)

last\_data['curs usd\_lag1'] = pred # Заменяем последний лаг на новый прогноз

# Отправляем все прогнозы пользователю

await message.reply("\n".join(predictions), reply\_markup=get\_days\_keyboard())

# Обработчик неизвестных команд/сообщений

@router.message()

async def handle\_unknown(message: Message):

logger.warning(f"Получено неизвестное сообщение '{message.text}' от пользователя {message.from\_user.id}")

await message.answer(

"Я знаю только команды и числа от 1 до 7 \nИспользуйте /help для открытия списка команд",

reply\_markup=ReplyKeyboardRemove()

)

keyboards/reply.py)

# Создает интерактивную клавиатуру для выбора количества дней прогноза (1-7 дней)

from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

def get\_days\_keyboard():

# Создаем список кнопок - каждая кнопка в отдельном ряду

buttons = [[KeyboardButton(text=str(i))] for i in range(1, 8)]

# Создаем клавиатуру с настройками:

# - keyboard: массив кнопок

# - resize\_keyboard: True - подстраивать размер клавиатуры под экран

keyboard = ReplyKeyboardMarkup(

keyboard=buttons,

resize\_keyboard=True,

one\_time\_keyboard=False # Клавиатура останется после выбора

)

return keyboard

models/models.py)

# Импорт библиотек для работы с данными и математических операций

import pandas as pd # Для работы с табличными данными (DataFrame)

import numpy as np # Для математических операций и работы с массивами

# Импорт моделей машинного обучения из scikit-learn

from sklearn.linear\_model import LinearRegression # Линейная регрессия

from sklearn.ensemble import RandomForestRegressor # Случайный лес для регрессии

from sklearn.svm import SVR # Метод опорных векторов для регрессии

from sklearn.model\_selection import train\_test\_split # Разделение данных на train/test

from sklearn.preprocessing import StandardScaler # Стандартизация данных

# Импорт компонентов для нейронной сети из TensorFlow/Keras

from tensorflow.keras.models import Sequential, load\_model # Последовательная модель НС

from tensorflow.keras.layers import Dense # Полносвязные слои для НС

# Импорт утилит

import joblib # Для сохранения и загрузки моделей

from utils.data\_preparation import prepare\_data # Кастомная функция для подготовки данных

# Подготовка данных

df\_d2 = prepare\_data() # Загружаем и предварительно обрабатываем данные

# Определение признаков (features) и целевой переменной (target)

features = [

'gold', 'silver', 'palladium', # Цены на драгметаллы

'key interest rate', 'inflation', # Макроэкономические показатели

'curs eur', 'brent (in usd)', # Курс евро и цена на нефть

'curs usd\_lag1', 'curs usd\_lag2', 'curs usd\_lag3', # Лаговые значения курса

'curs usd\_lag4', 'curs usd\_lag5', 'curs usd\_lag6', 'curs usd\_lag7'

]

X = df\_d2[features] # Матрица признаков

y = df\_d2['curs usd'] # Вектор целевой переменной (курс доллара)

# Разделение данных на обучающую и тестовую выборки (80%/20%)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y,

test\_size=0.2, # 20% данных в тест

random\_state=42 # Фиксируем random state для воспроизводимости

)

# Масштабирование данных (приведение к единому масштабу)

scaler = StandardScaler() # Инициализация стандартизатора

X\_train\_scaled = scaler.fit\_transform(X\_train) # Обучение scaler и трансформация train

X\_test\_scaled = scaler.transform(X\_test) # Трансформация test (без обучения!)

# Обучение модели линейной регрессии

lr\_model = LinearRegression() # Инициализация модели

lr\_model.fit(X\_train\_scaled, y\_train) # Обучение на масштабированных данных

# Обучение модели случайного леса

rf\_model = RandomForestRegressor(

n\_estimators=100, # Количество деревьев

random\_state=42 # Фиксируем random state

)

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

# Обучение модели SVR (Support Vector Regression)

svr\_model = SVR(kernel='rbf') # Используем radial basis function kernel

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

# Создание и обучение нейронной сети

nn\_model = Sequential([ # Последовательная модель

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

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

Dense(1) # Выходной слой (без активации для регрессии)

])

nn\_model.compile(

optimizer='adam', # Оптимизатор Adam

loss='mse' # Функция потерь - MSE (Mean Squared Error)

)

nn\_model.fit(

X\_train\_scaled, y\_train,

epochs=50, # Количество эпох обучения

batch\_size=32, # Размер батча

verbose=0 # Без вывода логов обучения

)

# Сохранение моделей и scaler'а

joblib.dump(lr\_model, 'lr\_model.joblib') # Линейная регрессия

joblib.dump(rf\_model, 'rf\_model.joblib') # Случайный лес

joblib.dump(svr\_model, 'svr\_model.joblib') # SVR

nn\_model.save('nn\_model.h5') # Нейронная сеть (сохраняется в HDF5)

joblib.dump(scaler, 'scaler.joblib') # Scaler для новых данных

# комбинированное предсказание на основе всех моделей (усреднение)

def predict\_combined(data):

data\_scaled = scaler.transform(data) # Масштабирование входных данных

# Получаем предсказания от всех моделей:

lr\_pred = lr\_model.predict(data\_scaled) # Линейная регрессия

rf\_pred = rf\_model.predict(data\_scaled) # Случайный лес

svr\_pred = svr\_model.predict(data\_scaled) # SVR

nn\_pred = nn\_model.predict(data\_scaled, verbose=0).flatten() # Нейросеть

# Усредняем предсказания:

combined\_pred = (lr\_pred + rf\_pred + svr\_pred + nn\_pred) / 4

return combined\_pred

# Расчет метрик качества модели

def calculate\_metrics(y\_true, y\_pred):

from sklearn.metrics import (

r2\_score, # Коэффициент детерминации

mean\_absolute\_error, # Средняя абсолютная ошибка

mean\_absolute\_percentage\_error, # Средняя абсолютная процентная ошибка

mean\_squared\_error # Среднеквадратичная ошибка

)

from math import sqrt # Для вычисления квадратного корня

# Вычисляем все метрики:

r2 = r2\_score(y\_true, y\_pred) # R² (чем ближе к 1, тем лучше)

mae = mean\_absolute\_error(y\_true, y\_pred) # MAE (в тех же единицах, что и target)

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

mse = mean\_squared\_error(y\_true, y\_pred) # MSE (штрафует большие ошибки)

rmse = sqrt(mse) # RMSE (интерпретируемая MSE)

return {

'R²': r2,

'MAE': mae,

'MAPE': mape,

'MSE': mse,

'RMSE': rmse

}

# Пример использования: расчет метрик на тестовых данных

y\_pred = predict\_combined(X\_test) # Получаем предсказания

metrics = calculate\_metrics(y\_test, y\_pred) # Вычисляем метрики

utils/data\_preparation.py)

import pandas as pd # Основная библиотека для работы с табличными данными

def prepare\_data(input\_file='Доллар табл.xlsx', output\_file='df\_d2.csv', latest\_file='latest\_data.csv'):

# загрузка данных

# Чтение данных из Excel-файла

df = pd.read\_excel(input\_file)

# предбразование дат

# Проверяем, нужно ли преобразовывать столбец с датами

if not pd.api.types.is\_datetime64\_any\_dtype(df['data']):

# Преобразование дат из Excel-формата (число дней с 1899-12-30)

df['data'] = pd.to\_datetime(df['data'], origin='1899-12-30', unit='D')

# Устанавливаем дату в качестве индекса и сортируем по возрастанию

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

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

# 3. обработка пропущенных значений

# Для ключевой ставки: сначала заполняем последующие значения, затем интерполируем

df['key interest rate'] = df['key interest rate'].bfill() # Заполнение последующими значениями

df['key interest rate'] = df['key interest rate'].interpolate(method='linear') # Линейная интерполяция

# Аналогично для инфляции

df['inflation'] = df['inflation'].bfill()

df['inflation'] = df['inflation'].interpolate(method='linear')

# удаление ненужных столбцов

# Удаляем столбцы, которые не используются в анализе

df.drop(columns=['nominal', 'name of cur'], inplace=True, errors='ignore')

# Добавление лаговых признаков

# Создаем 7 лаговых признаков для курса доллара

for i in range(1, 8):

df[f'curs usd\_lag{i}'] = df['curs usd'].shift(i) # Сдвиг на i дней назад

# Удаляем строки с пропусками (первые 7 дней, для которых нет всех лагов)

df.dropna(inplace=True)

# сохранение результатов

# Полные данные сохраняем в основной файл

df.to\_csv(output\_file)

# Последние 100 записей (актуальные данные) сохраняем в отдельный файл

df.tail(100).to\_csv(latest\_file)

return df

if \_\_name\_\_ == '\_\_main\_\_':

# Точка входа при запуске скрипта напрямую

prepare\_data()