Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамента цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №7
дисциплины «Основы нейронных сетей»

Выполнил:
Гайчук Дарья Дмитриевна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика
и вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А.-доцент департамента
цифровых, роботехнических систем и
электроники института перспективной
инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Исследование регрессии нейронных сетей

Цель: Научиться работать с временными рядами, устранением трендов, автокорреляцией и построением нейросетей для прогнозов

Ссылка на git: https://github.com/Ichizuchi/NN_LR7

Ход работы

В практических заданиях основной акцент был сделан на обработке и анализе временных рядов с применением различных нейросетевых архитектур. Начальные шаги включали создание синтетических рядов, расчет коэффициентов автокорреляции, визуализацию и анализ трендов. Ключевой статистический показатель — коэффициент автокорреляции — при сдвиге на один шаг оказался равен 0.2298. Это значение демонстрирует умеренную зависимость между текущим и предыдущим значением в ряду. Среднее значение ряда составило около 7.29, а стандартное отклонение — 1.71, что говорит о средней вариативности данных.

Далее проводилось сравнение нейросетевых архитектур. Простая полносвязная модель показала удовлетворительные результаты, однако была склонна к переобучению. Сеть с предварительным Flatten-слоем оказалась менее точной из-за потери временной структуры данных. Сверточные сети, особенно упрощённые, показали лучшую точность, благодаря способности извлекать локальные зависимости. LSTM, несмотря на теоретическую пригодность для временных рядов, в данной работе уступила по точности и стабильности.

Ниже представлены таблицы с ключевыми статистиками временного ряда и сравнением результатов архитектур нейросетей. Это позволяет сделать вывод о том, что даже простые архитектуры могут давать хорошее качество при корректной подготовке данных, а также о необходимости тщательно подбирать структуру сети под тип задачи.

Таблица 1. Статистика временного ряда

Метрика	Значение
Мат. ожидание (исходный)	7.29
Мат. ожидание (смещенный)	7.01
СКО (исходный)	1.71
СКО (смещенный)	1.61
Мат. ожидание произведения	51.74
Коэф. автокорреляции (1 шаг)	0.2298

Таблица 2. Сравнение архитектур нейросетей

Архитектура	Точность прогноза	Комментарий
Полносвязная	Средняя	Прогнозирует близко, но есть зависимость от автокорреляции
Полносвязная с Flatten	Низкая	Потеря структуры данных, что снижает точность
Сверточная (1D Conv)	Средняя	Более сложная архитектура, но не улучшила результат
Упрощенная сверточная	Хорошая	Простая, но эффективно извлекает закономерности
LSTM	Ниже среднего	Требует настройки, в данной работе уступила по результатам

Выполнение домашнего задания

Задание 1.

Условие: в данном задании используется база курса акций Лукойла.

Необходимо:

1. Обучить простую полносвязную сеть для прогнозирования временного ряда (только close) и визуализировать результат.

2. Обучить такую же архитектуру сети на прогнозирование на 10 шагов вперёд прямым способом и визуализировать результат.

3. Построить графики сравнения предсказания с оригинальным сигналом по всем 10 шагам предсказания (10 графиков на разных отдельных осях).

4. Сделать те же задания с другой сетью, которая будет использовать Conv1D или LSTM слои.

Начнем с загрузки необходимых для решения библиотек. Затем выполним назначение размера и стиля графиков.

```
[ ] # Работа с массивами
    import numpy as np

    # Работа с таблицами
    import pandas as pd

    # Классы-конструкторы моделей нейронных сетей
    from tensorflow.keras.models import Sequential, Model

    # Основные слои
    from tensorflow.keras.layers import concatenate, Input, Dense, Dropout, BatchNormalization, Flatten, Conv1D, LSTM

    # Оптимизаторы
    from tensorflow.keras.optimizers import Adam

    # Генератор выборки временных рядов
    from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator

    # Нормировщики
    from sklearn.preprocessing import StandardScaler, MinMaxScaler

    # Загрузка датасетов из облака google
    import gdown

    # Отрисовка графиков
    import matplotlib.pyplot as plt

    # Отрисовка графики в ячейке colab
    %matplotlib inline

    # Отключение предупреждений
    import warnings
    warnings.filterwarnings('ignore')
```

```
[ ] # Назначение размера и стиля графиков по умолчанию
    from pylab import rcParams
    plt.style.use('ggplot')
    rcParams['figure.figsize'] = (14, 7)
```

Рисунок 1. Импорт библиотек

Ввыполним загрузку данных, а также чтение данных в таблицу и выполним вывод первых строк таблицы.

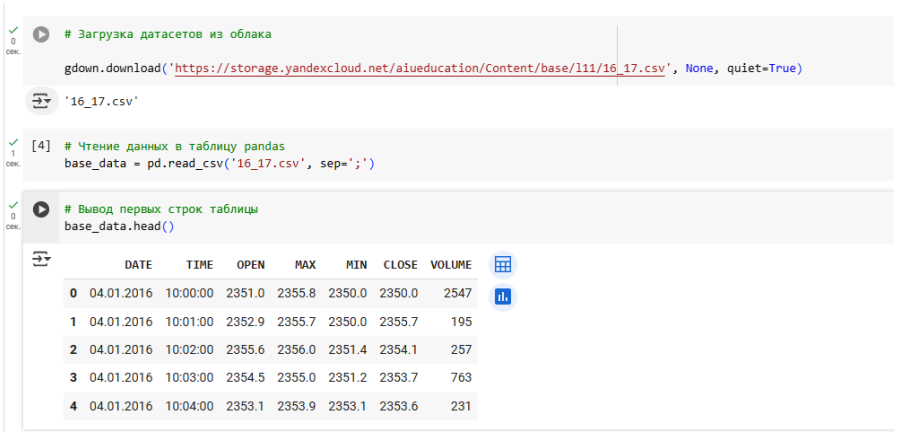


Рисунок 2. Вывод первых строк таблицы

Выполним загрузку датасетов с удалением ненужных столбцов по дате и времени, а также выполним вывод размерностей, получившихся таблицы и вывод примера данных одного датасета.

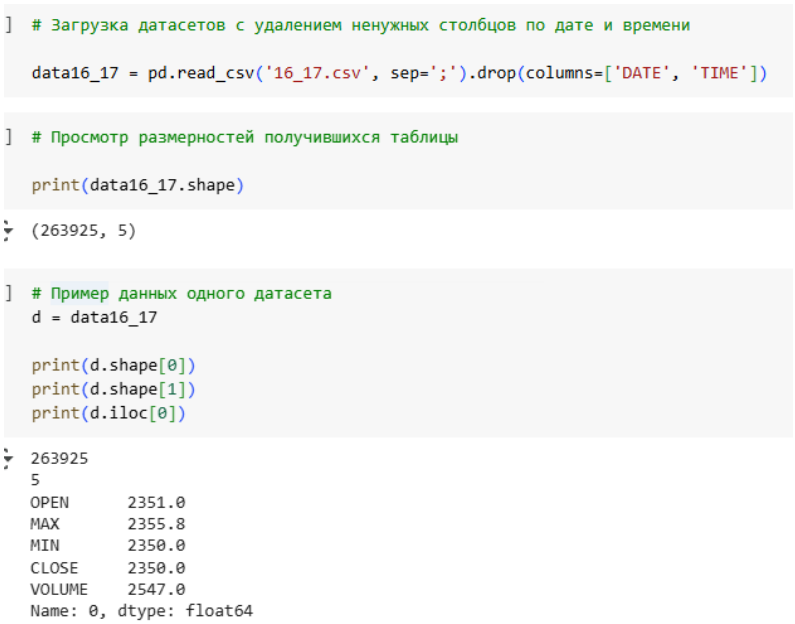


Рисунок 3. Просмотр размерностей получившихся таблицы

Далее выполним создание общего набора данных из двух датасетов, задание текстовых меток каналов данных и выполним просмотр размерности новой таблицы.

```
] # Создание общего набора данных из двух датасетов

  data = pd.concat([data16_17]).to_numpy()
```

```
] # Просмотр размерности новой таблицы

  print(data16_17.shape)
  print(data.shape)
```

```
(263925, 5)
(263925, 5)
```

```
] # Задание текстовых меток каналов данных (столбцов)

  channel_names = ['Open', 'Max', 'Min', 'Close', 'Volume']
```

Рисунок 4. Просмотр размерностей новой таблицы

Затем выполним отображение исходных данных от точки start и длинной length.

Иллюстрация данных в графическом виде

```
[ ] # Отображение исходных данных от точки start и длинной length
    start = 100
    length = 300

    # Задание полотна для графиков - два подграфика один под другим с общей осью x
    fig, ax = plt.subplots(figsize=(22, 13), sharex=True)

    # Четыре основных канала
    for chn in range(4):
        ax.plot(data[start:start + length, chn],
                label=channel_names[chn])
    ax.set_ylabel('Цена, руб')
    ax.legend()
    plt.xlabel('Время')

    # Регулировка пределов оси x
    plt.xlim(0, length)

    plt.tight_layout()

    # Фиксация графика
    plt.show()
```

Рисунок 5. Код для отображения исходных данных

Посмотрим на получившийся график.

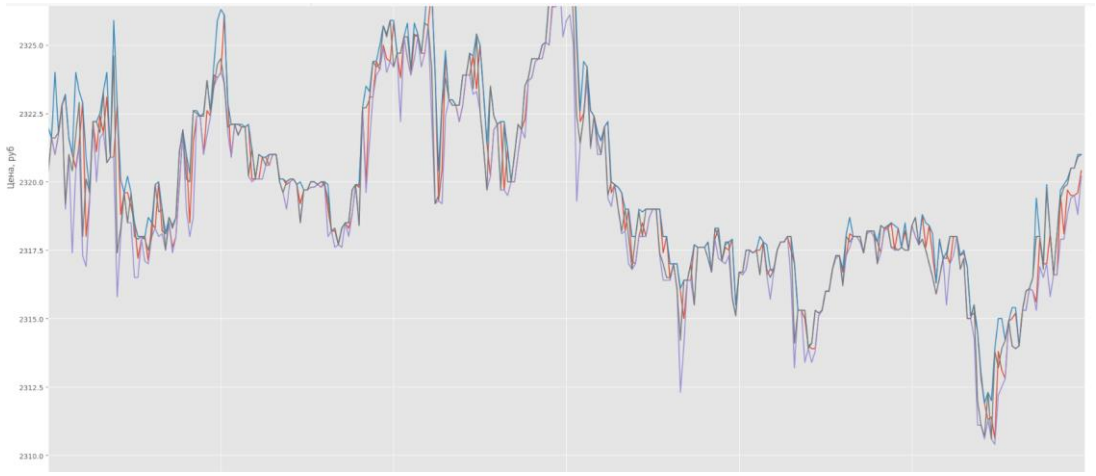


Рисунок 6. График отображения исходных данных

Далее зададим текстовые метки словарей каналов данных и также зададим гиперпараметры.

```
# Задание текстовых меток каналов данных (столбцов)
CHANNEL_NAMES = ['Open', 'Max', 'Min', 'Close', 'Volume']

# Получение словаря с именами и индексами каналов данных
CHANNEL_INDEX = {name: chan for chan, name in enumerate(CHANNEL_NAMES)}
print(CHANNEL_INDEX)
```

```
{'Open': 0, 'Max': 1, 'Min': 2, 'Close': 3, 'Volume': 4}
```

```
# Задание гиперпараметров
CHANNEL_X = CHANNEL_NAMES
CHANNEL_Y = ['Close']
SEQ_LEN = 300
BATCH_SIZE = 20
TEST_LEN = 30000
TRAIN_LEN = data.shape[0] - TEST_LEN
```

Рисунок 7. Задание гиперпараметров

Затем выполним формирование списков индексов каналов данных для входных и выходных выборок, так же выполним разделение данных на тренировочный и тестовый наборы, выполним масштабирование данных и создание генератора для обучения.

```
# Разделение данных на тренировочный и тестовый наборы
data_train, data_test = data[:TRAIN_LEN], data[TRAIN_LEN + 2*SEQ_LEN:]

# Отбор входных данных
x_data_train, x_data_test = data_train[:, chn_x], data_test[:, chn_x]

# Масштабирование данных
x_scaler = MinMaxScaler()
x_scaler.fit(x_data_train)
x_data_train = x_scaler.transform(x_data_train)
x_data_test = x_scaler.transform(x_data_test)

# Отбор выходных данных
y_data_train, y_data_test = data_train[:, chn_y], data_test[:, chn_y]

# Масштабирование данных
y_scaler = MinMaxScaler()
y_scaler.fit(y_data_train)
y_data_train = y_scaler.transform(y_data_train)
y_data_test = y_scaler.transform(y_data_test)

# Проверка формы данных
print(f'Train data: {x_data_train.shape}, {y_data_train.shape}')
print(f'Test  data: {x_data_test.shape}, {y_data_test.shape}')

# Создание генератора для обучения
train_datagen = TimeseriesGenerator(x_data_train,
                                    y_data_train,
                                    length=SEQ_LEN,
                                    stride=1,
                                    sampling_rate=1,
                                    batch_size=BATCH_SIZE)

# Аналогичный генератор для валидации при обучении
val_datagen = TimeseriesGenerator(x_data_test,
                                  y_data_test,
                                  length=SEQ_LEN,
                                  stride=1,
                                  sampling_rate=1,
                                  batch_size=BATCH_SIZE)

# Проверка формы выдаваемого генератором результата
print(f'Train batch x: {train_datagen[0][0].shape}, y: {train_datagen[0][1].shape}')
```

```
[0, 1, 2, 3, 4] [3]
Train data: (233925, 5), (233925, 1)
Test  data: (29400, 5), (29400, 1)
Train batch x: (20, 300, 5), y: (20, 1)
```

Рисунок 8. Формирование списков индексов

Далее напишем генератор для тестовой выборки и выполним проверку формы тестовой выборки.

```
# Генератор тестовой выборки, генерирует один батч на всю выборку
test_datagen = TimeseriesGenerator(x_data_test,
                                   y_data_test,
                                   length=SEQ_LEN,
                                   stride=1,
                                   sampling_rate=1,
                                   batch_size=x_data_test.shape[0])

# Формирование тестовой выборки из генератора
x_test, y_test = test_datagen[0]

# Проверка формы тестовой выборки
print(f'Test x: {x_test.shape}, y: {y_test.shape}')
```

```
Test x: (29100, 300, 5), y: (29100, 1)
```

Рисунок 9. Генератор тестовой выборки

```
] # Функция расчета результата прогнозирования сети

  def get_pred(model,
               x_test, y_test,
               y_scaler):

      # Вычисление и деномализация предсказания
      y_pred_unscaled = y_scaler.inverse_transform(model.predict(x_test))

      # Денормализация верных ответов
      y_test_unscaled = y_scaler.inverse_transform(y_test)

      # Возврат результата предсказания и верные ответы в исходном масштабе
      return y_pred_unscaled, y_test_unscaled
```

Рисунок 10. Функция расчета результата прогнозирования сети

Функция визуализации результата предсказания сети и верных ответов.

```
# Функция визуализации результата предсказания сети и верных ответов

def show_predict(y_pred, y_true,
                 start,
                 length,
                 chn_list=None,
                 chn_names=CHANNEL_Y,
                 title=''):

    if not chn_list:
        chn_list = list(range(y_true.shape[1]))

    # Построение графика по всем каналам данных
    plt.figure(figsize=(22, 6))

    for chn in chn_list:
        plt.plot(y_pred[start:start + length, chn],
                 label=f'{chn_names[chn]} Прогноз')
        plt.plot(y_true[start:start + length, chn],
                 label=f'{chn_names[chn]} Базовый')

    plt.title(title)
    plt.xlabel('Время')
    plt.ylabel('Данные')
    plt.legend()
    plt.show()
```

Рисунок 11. Функция визуализации результата предсказания сети

Функция рисования корреляций прогнозного ряда и исходного со смещением.

```python
# Функция рисования корреляций прогнозного ряда и исходного со смещением

def show_corr(y_pred, y_true,
              back_steps_max=30,
              chn_list=None,
              chn_names=CHANNEL_Y,
              title=''):

    # Если список каналов пуст - отображаются все каналы
    if not chn_list:
        chn_list = list(range(y_true.shape[1]))

    y_len = y_true.shape[0]
    steps = range(0, back_steps_max + 1)

    # Построение графика по всем каналам данных
    plt.figure(figsize=(14, 7))

    for chn in chn_list:
        # Вычисление коэффициентов корреляции базового ряда и предсказания с разным смещением
        cross_corr = [correlate(y_true[:y_len - step, chn], y_pred[step:, chn]) for step in steps]
        # Вычисление коэффициентов автокорреляции базового ряда с разным смещением
        auto_corr = [correlate(y_true[:y_len - step, chn], y_true[step:, chn]) for step in steps]

        plt.plot(cross_corr, label=f'{chn_names[chn]} Прогноз')
        plt.plot(auto_corr, label=f'{chn_names[chn]} Эталон')

    plt.title(title)

    plt.xticks(steps)
    plt.xlabel('Шаги смещения')
    plt.ylabel('Коэффициент корреляции')
    plt.legend()
    plt.show()
```

Рисунок 12. Функция рисования корреляций прогнозного ряда

```python
# Функция расчета корреляционного коэффициента Пирсона для двух рядов

def correlate(a, b):
    return np.corrcoef(a, b)[0, 1]
```

```python
# Функция визуализации результата работы сети

def eval_net(model,
             x_test, y_test,
             y_scaler,
             start=0, length=500, back_steps_max=30,
             title=''):

    # Получение денормализованного предсказания и данных базового ряда
    y_pred, y_true = get_pred(model, x_test, y_test, y_scaler)

    # Отрисовка графика сопоставления базового и прогнозного рядов
    show_predict(y_pred[1:], y_true[:-1], start, length,
                 title=f'{title}: Сопоставление базового и прогнозного рядов')

    show_corr(y_pred, y_true, back_steps_max=back_steps_max,
              title=f'{title}: Корреляционные коэффициенты по шагам смещения')
```

Рисунок 13. Функция расчета корреляционного коэффициента Пирсона

Последней сервисной функцией будет функция обучения модели и отрисовки прогресса и оценки результатов.

```python
# Функция обучения модели и отрисовки прогресса и оценки результатов

def train_eval_net(model, # модель
                   train_datagen, val_datagen, # генераторы обучающей и проверочной выборок
                   epoch_list, # список эпох в виде [(epochs1, opt1), (epochs2, opt2), ...]
                   x_test, y_test,
                   y_scaler,
                   start=0,
                   length=500,
                   back_steps_max=30,
                   title=''):

    # Отображение сводки модели
    model.summary()

    # Обучение модели в несколько фаз в соответствии со списком epoch_list
    for epochs, opt in epoch_list:
        # Компиляция модели
        model.compile(loss='mse', optimizer=opt)
        # Фаза обучения модели
        print(f'Обучение {epochs} эпох')
        history = model.fit(train_datagen,
                            epochs=epochs,
                            validation_data=val_datagen,
                            verbose=1)

        # Рисование графиков прошедшей фазы обучения
        fig = plt.figure(figsize=(14, 7))
        plt.plot(history.history['loss'], label='Ошибка на обучающем наборе')
        plt.plot(history.history['val_loss'], label='Ошибка на проверочном наборе')
        plt.title(f'{title}: График прогресса обучения')
        # Указание показывать только целые метки шкалы оси x
        fig.gca().xaxis.get_major_locator().set_params(integer=True)
        plt.xlabel('Эпоха обучения')
        plt.ylabel('Средняя ошибка')
        plt.legend()
        plt.show()

        # Рисование графиков оценки результата работы модели после фазы обучения
        eval_net(model, x_test, y_test, y_scaler, start=start,
                 length=length, back_steps_max=back_steps_max, title=title)
```

Рисунок 14. Функция обучения модели

Далее выполним построение и обучение простой полносвязной сети для прогнозирования временного ряда (только close) и визуализируем его результат.

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 300, 32)        │           192 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 9600)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 1)              │         9,601 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 9,793 (38.25 KB)
 Trainable params: 9,793 (38.25 KB)
 Non-trainable params: 0 (0.00 B)
Обучение 10 эпох
Epoch 1/10
11682/11682 ━━━━━━━━━━━━━━━━━━━━ 33s 3ms/step - loss: 0.0215 - val_loss: 1.1031e-04
Epoch 2/10
11682/11682 ━━━━━━━━━━━━━━━━━━━━ 32s 3ms/step - loss: 1.2159e-04 - val_loss: 9.2120e-05
Epoch 3/10
11682/11682 ━━━━━━━━━━━━━━━━━━━━ 30s 3ms/step - loss: 1.0791e-04 - val_loss: 7.6630e-05
Epoch 4/10
11682/11682 ━━━━━━━━━━━━━━━━━━━━ 31s 3ms/step - loss: 9.4204e-05 - val_loss: 8.2844e-05
Epoch 5/10
11682/11682 ━━━━━━━━━━━━━━━━━━━━ 33s 3ms/step - loss: 8.2885e-05 - val_loss: 6.3144e-05
Epoch 6/10
11682/11682 ━━━━━━━━━━━━━━━━━━━━ 31s 3ms/step - loss: 7.2456e-05 - val_loss: 5.8775e-05
Epoch 7/10
11682/11682 ━━━━━━━━━━━━━━━━━━━━ 31s 3ms/step - loss: 6.8018e-05 - val_loss: 5.1192e-05
Epoch 8/10
11682/11682 ━━━━━━━━━━━━━━━━━━━━ 31s 3ms/step - loss: 6.0629e-05 - val_loss: 8.1360e-05
Epoch 9/10
11682/11682 ━━━━━━━━━━━━━━━━━━━━ 32s 3ms/step - loss: 5.8278e-05 - val_loss: 5.2334e-05
Epoch 10/10
```

Рисунок 15. Архитектура модели

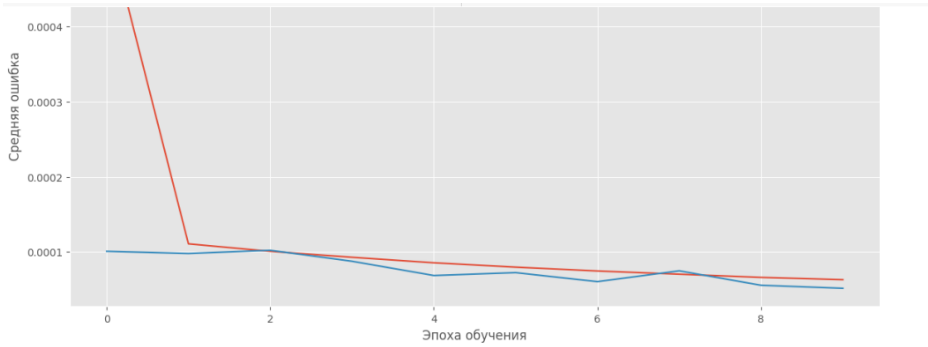


Рисунок 16. График процесса обучения

Рисунок 17. Сопоставление базового и прогнозного рядов

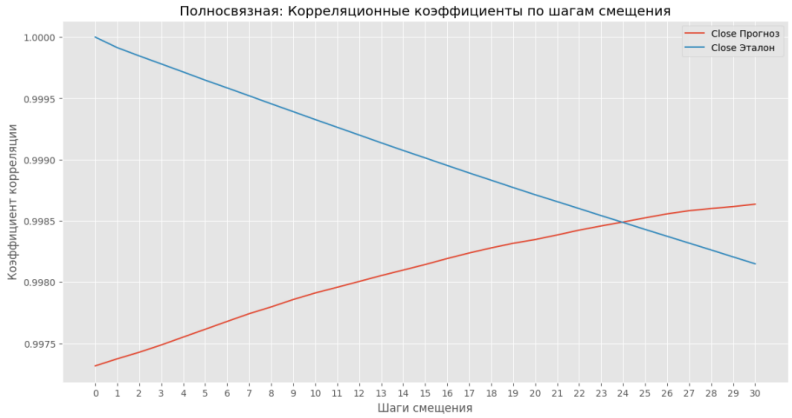


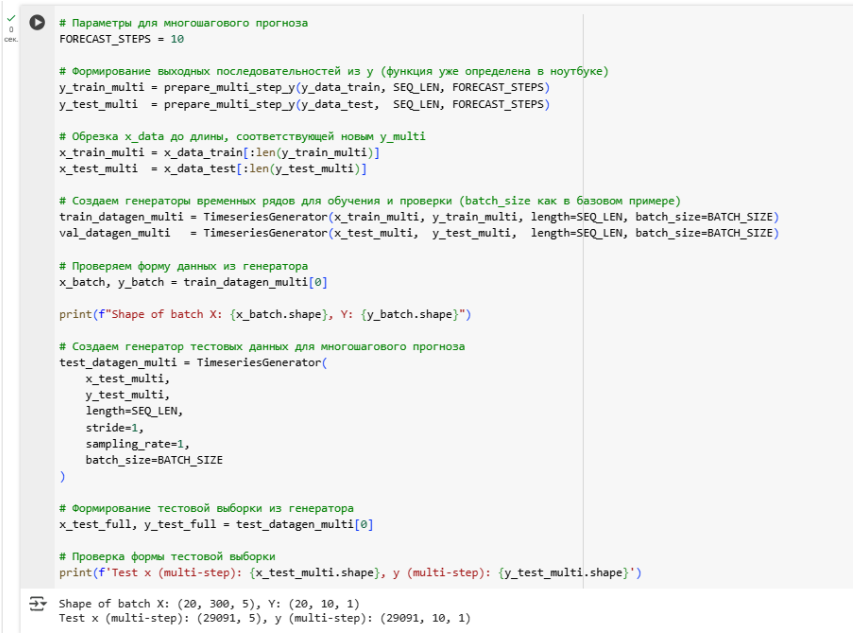
Рисунок 18. Корреляционные коэффициенты по шагам смещения

Далее необходимо обучить такую же архитектуру сети на прогнозирование на 10 шагов вперёд прямым способом и визуализировать результат.

```
[25] # Параметры
     FORECAST_STEPS = 10

     # Подготовка y_data для многошагового предсказания
     def prepare_multi_step_y(y_data, seq_len, forecast_steps):
         y_sequences = []
         for i in range(len(y_data) - seq_len - forecast_steps + 1):
             y_sequences.append(y_data[i+seq_len : i+seq_len+forecast_steps])
         return np.array(y_sequences)
```

Рисунок 19. Подготовка новых данных

Зададим параметры для многошагового прогноза.

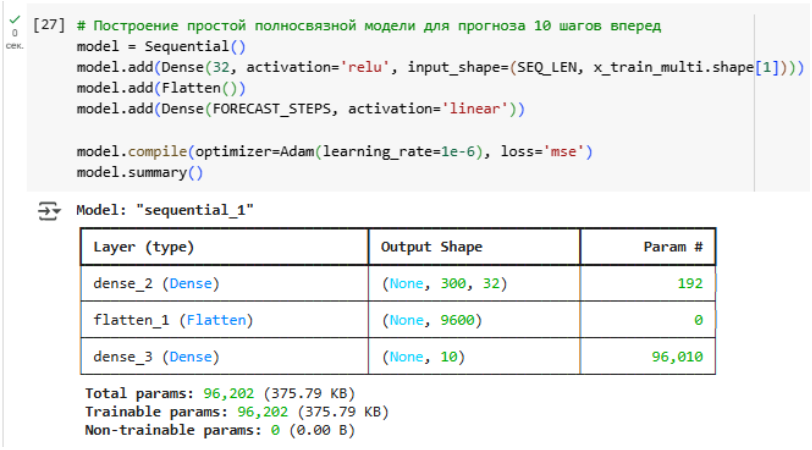


```
# Параметры для многошагового прогноза
FORECAST_STEPS = 10

# Формирование выходных последовательностей из y (функция уже определена в ноутбуке)
y_train_multi = prepare_multi_step_y(y_data_train, SEQ_LEN, FORECAST_STEPS)
y_test_multi  = prepare_multi_step_y(y_data_test,  SEQ_LEN, FORECAST_STEPS)

# Обрезка x_data до длины, соответствующей новым y_multi
x_train_multi = x_data_train[:len(y_train_multi)]
x_test_multi  = x_data_test[:len(y_test_multi)]

# Создаем генераторы временных рядов для обучения и проверки (batch_size как в базовом примере)
train_datagen_multi = TimeseriesGenerator(x_train_multi, y_train_multi, length=SEQ_LEN, batch_size=BATCH_SIZE)
val_datagen_multi   = TimeseriesGenerator(x_test_multi,  y_test_multi,  length=SEQ_LEN, batch_size=BATCH_SIZE)

# Проверяем форму данных из генератора
x_batch, y_batch = train_datagen_multi[0]

print(f"Shape of batch X: {x_batch.shape}, Y: {y_batch.shape}")

# Создаем генератор тестовых данных для многошагового прогноза
test_datagen_multi = TimeseriesGenerator(
    x_test_multi,
    y_test_multi,
    length=SEQ_LEN,
    stride=1,
    sampling_rate=1,
    batch_size=BATCH_SIZE
)

# Формирование тестовой выборки из генератора
x_test_full, y_test_full = test_datagen_multi[0]

# Проверка формы тестовой выборки
print(f'Test x (multi-step): {x_test_multi.shape}, y (multi-step): {y_test_multi.shape}')
```

```
Shape of batch X: (20, 300, 5), Y: (20, 10, 1)
Test x (multi-step): (29091, 5), y (multi-step): (29091, 10, 1)
```

Рисунок 20. Параметры для многошагового прогноза

Далее выполним создание простой полносвязной модели для прогноза 10 шагов вперед.



```
[27] # Построение простой полносвязной модели для прогноза 10 шагов вперед
     model = Sequential()
     model.add(Dense(32, activation='relu', input_shape=(SEQ_LEN, x_train_multi.shape[1])))
     model.add(Flatten())
     model.add(Dense(FORECAST_STEPS, activation='linear'))

     model.compile(optimizer=Adam(learning_rate=1e-6), loss='mse')
     model.summary()
```

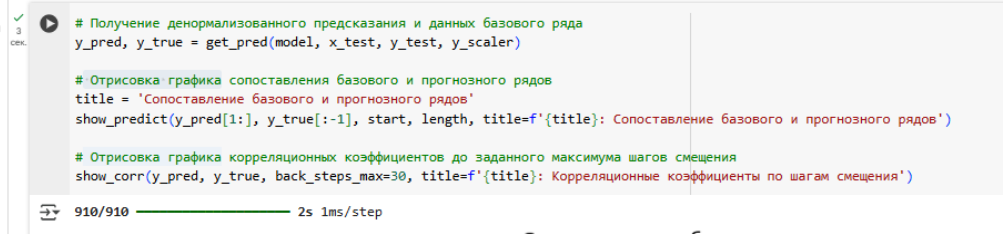
```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 300, 32)	192
flatten_1 (Flatten)	(None, 9600)	0
dense_3 (Dense)	(None, 10)	96,010

```
Total params: 96,202 (375.79 KB)
Trainable params: 96,202 (375.79 KB)
Non-trainable params: 0 (0.00 B)
```

Рисунок 21. Архитектура модели

Затем выполним обучение созданной модели для прогноза 10 шагов вперед.



```
# Получение денормализованного предсказания и данных базового ряда
y_pred, y_true = get_pred(model, x_test, y_test, y_scaler)

# Отрисовка графика сопоставления базового и прогнозного рядов
title = 'Сопоставление базового и прогнозного рядов'
show_predict(y_pred[1:], y_true[:-1], start, length, title=f'{title}: Сопоставление базового и прогнозного рядов')

# Отрисовка графика корреляционных коэффициентов до заданного максимума шагов смещения
show_corr(y_pred, y_true, back_steps_max=30, title=f'{title}: Корреляционные коэффициенты по шагам смещения')
```

```
910/910 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
```

Рисунок 22. Отрисовка графика

Далее посмотрим на получившиеся графики сопоставления базового и прогнозного рядов и график корреляционных коэффициентов до заданного максимума шагов смещения.

Рисунок 23. График сопоставления базового и прогнозного рядов

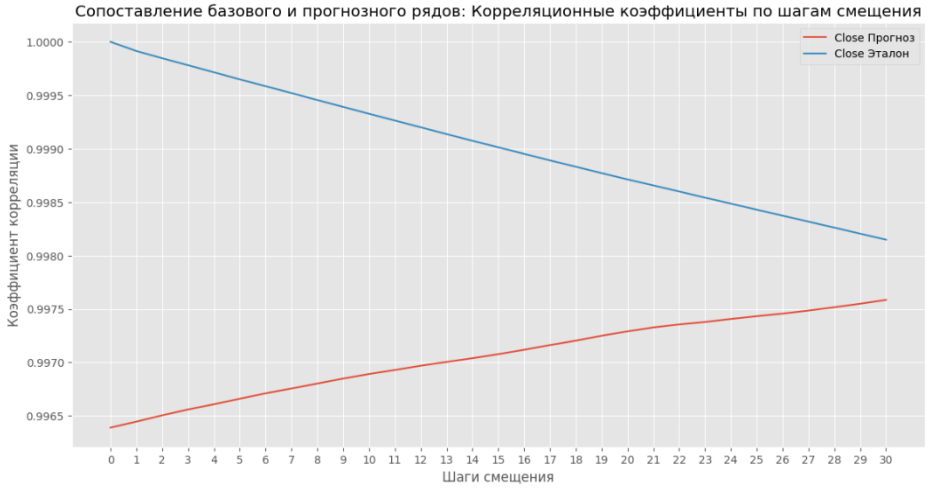


Рисунок 24. График корреляционных коэффициентов

Далее напишем код для предсказания на тестовой выборке и выполним преобразование предсказаний и правильных ответов обратно в исходный масштаб

```
[30] # Получение предсказаний на тестовой выборке (в масштабе [0,1])
     y_pred_scaled = model.predict(test_datagen_multi)  # форма (N, 10)

     # Преобразование предсказаний и правильных ответов обратно в исходный масштаб
     y_pred_reshaped = y_pred_scaled.reshape(-1, 1)
     y_test_reshaped = y_test_multi.reshape(-1, 1)

     y_pred_unscaled = y_scaler.inverse_transform(y_pred_reshaped).reshape(-1, FORECAST_STEPS)
     y_test_unscaled = y_scaler.inverse_transform(y_test_reshaped).reshape(-1, FORECAST_STEPS)

     1440/1440 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
```

Рисунок 25. Получение предсказаний на тестовой выборке

Далее выполним построение 10 графиков, которые отображают истинные и предсказанные значения для каждого горизонта прогноза.

Затем после выполнения кода на рисунке 76, посмотрим на построенные графики.

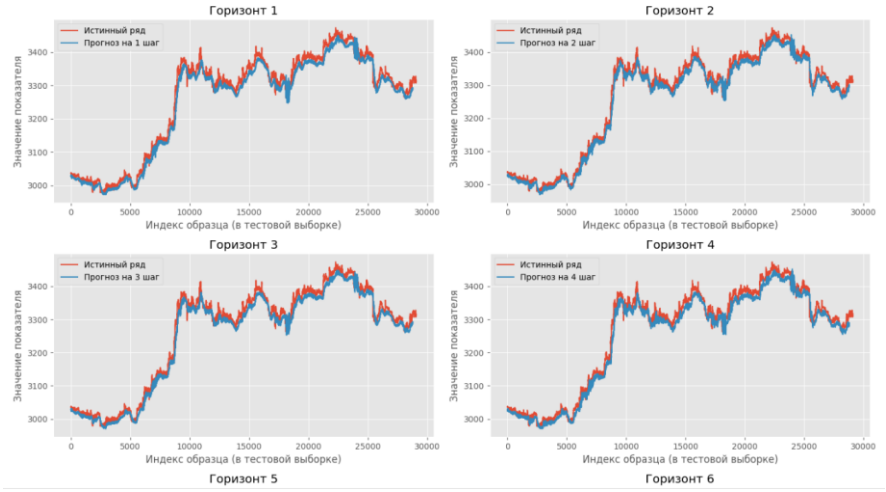


Рисунок 26. Отображение графиков для 1 – 4 горизонтов

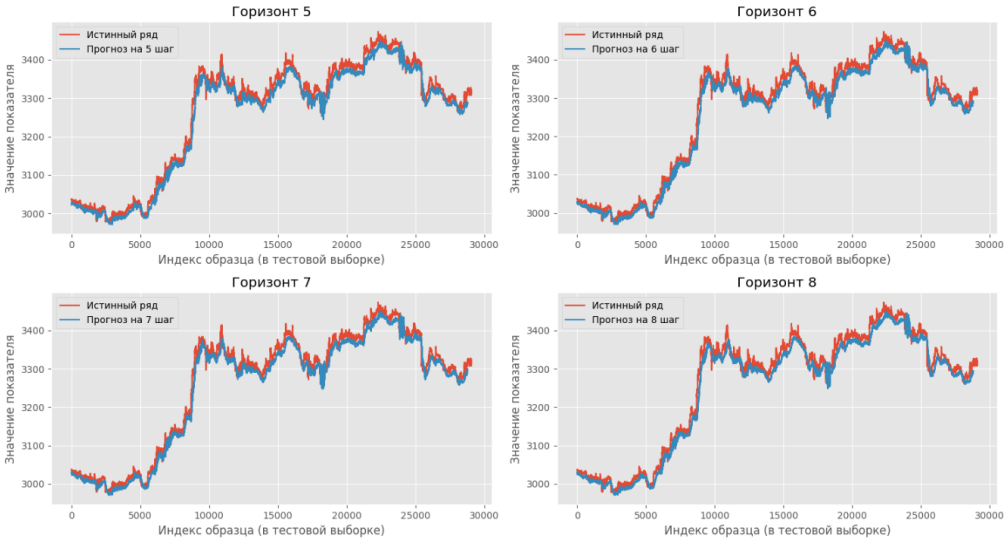


Рисунок 27. Отображение графиков для 5 – 8 горизонтов

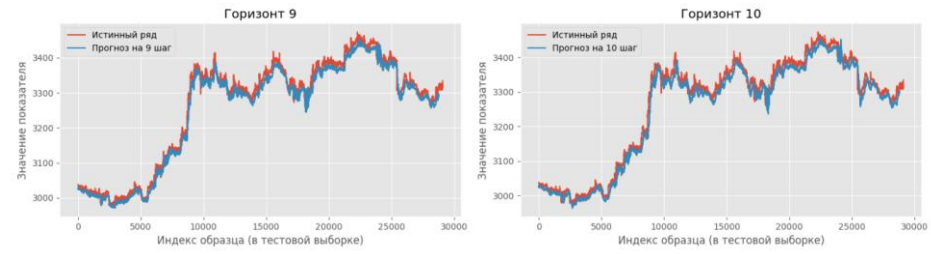


Рисунок 28. Отображение графиков для 9 и 10 горизонтов

Далее необходимо выполнить те же задания только с другой сетью, которая будет использовать Conv1D или LSTM слои. Для этого выполним построение модели сети с использованием Conv1D и выполним обучение .

```
[33] # Упрощенная модель с одномерной сверткой
     model_conv_2 = Sequential()
     model_conv_2.add(Conv1D(50, 5, input_shape=x_test.shape[1:], activation='linear'))
     model_conv_2.add(Flatten())
     model_conv_2.add(Dense(10, activation='linear'))
     model_conv_2.add(Dense(FORECAST_STEPS, activation='linear'))

     model_conv_2.compile(optimizer=Adam(learning_rate=1e-6), loss='mse')
     model_conv_2.summary()
```

Рисунок 29. Модель с использованием Conv1D

Затем посмотрим на результаты обучения данной модели и на получившиеся графики.

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 296, 50)	1,300
flatten_2 (Flatten)	(None, 14800)	0
dense_4 (Dense)	(None, 10)	148,010
dense_5 (Dense)	(None, 1)	11

Total params: 149,321 (583.29 KB)
Trainable params: 149,321 (583.29 KB)
Non-trainable params: 0 (0.00 B)
Обучение 10 эпох

Рисунок 30. Архитектура модели

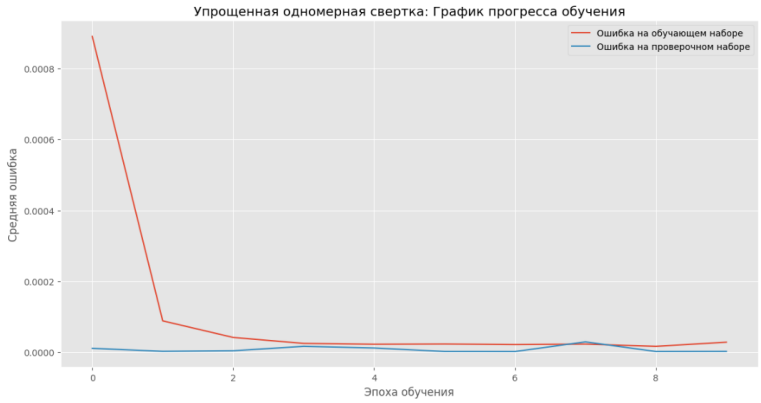


Рисунок 31. График процесса обучения

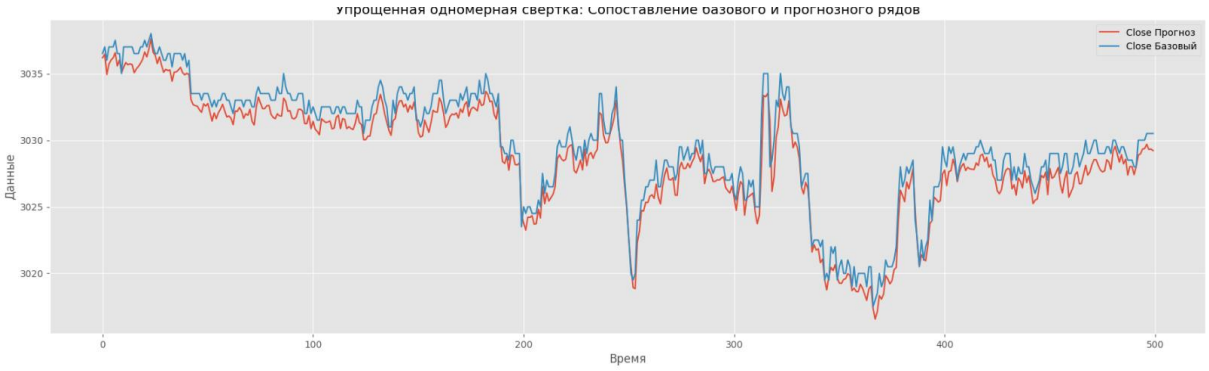


Рисунок 32. Сопоставление базового и прогнозного рядов

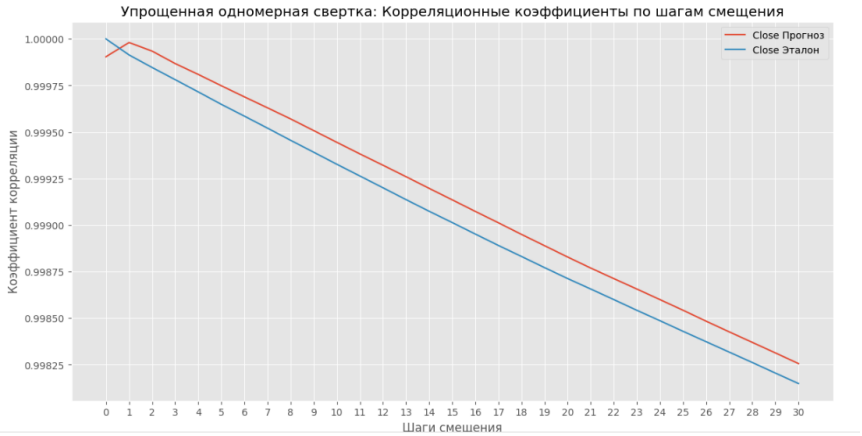


Рисунок 33. Корреляционные коэффициенты по шагам смещения

Далее напишем такую же сеть, но для предсказания 10 будущих значений.

```
# Упрощенная модель с одномерной сверткой
model_conv_2 = Sequential()
model_conv_2.add(Conv1D(50, 5, input_shape=x_test.shape[1:], activation='linear'))
model_conv_2.add(Flatten())
model_conv_2.add(Dense(10, activation='linear'))
model_conv_2.add(Dense(FORECAST_STEPS, activation='linear'))

model_conv_2.compile(optimizer=Adam(learning_rate=1e-6), loss='mse')
model_conv_2.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 296, 50)	1,300
flatten_3 (Flatten)	(None, 14800)	0
dense_6 (Dense)	(None, 10)	148,010
dense_7 (Dense)	(None, 10)	110

```
Total params: 149,420 (583.67 KB)
Trainable params: 149,420 (583.67 KB)
Non-trainable params: 0 (0.00 B)
```

Рисунок 34. Архитектура модели

Затем выполним обучение созданной модели.

```
[34] # Обучение модели в течение 10 эпох (как в исходном примере)
history = model_conv_2.fit(
    train_datagen_multi,
    validation_data=val_datagen_multi,
    epochs=10,
    verbose=1
)
```

```
Epoch 1/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 40s 3ms/step - loss: 0.0610 - val_loss: 4.6740e-04
Epoch 2/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 40s 3ms/step - loss: 5.4758e-04 - val_loss: 4.8578e-04
Epoch 3/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 38s 3ms/step - loss: 5.4076e-04 - val_loss: 4.1030e-04
Epoch 4/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 38s 3ms/step - loss: 5.2904e-04 - val_loss: 4.5494e-04
Epoch 5/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 38s 3ms/step - loss: 5.2476e-04 - val_loss: 4.8766e-04
Epoch 6/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 38s 3ms/step - loss: 5.3416e-04 - val_loss: 5.6734e-04
Epoch 7/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 38s 3ms/step - loss: 5.4028e-04 - val_loss: 4.1526e-04
Epoch 8/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 41s 3ms/step - loss: 5.2147e-04 - val_loss: 5.0214e-04
Epoch 9/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 38s 3ms/step - loss: 5.1271e-04 - val_loss: 4.1084e-04
Epoch 10/10
11666/11666 ━━━━━━━━━━━━━━━━━━━━ 38s 3ms/step - loss: 5.1899e-04 - val_loss: 4.0141e-04
```

Рисунок 35. Обучение модели

Далее посмотрим на получившиеся графики сопоставления базового и прогнозного рядов и график корреляционных коэффициентов до заданного максимума шагов смещения.

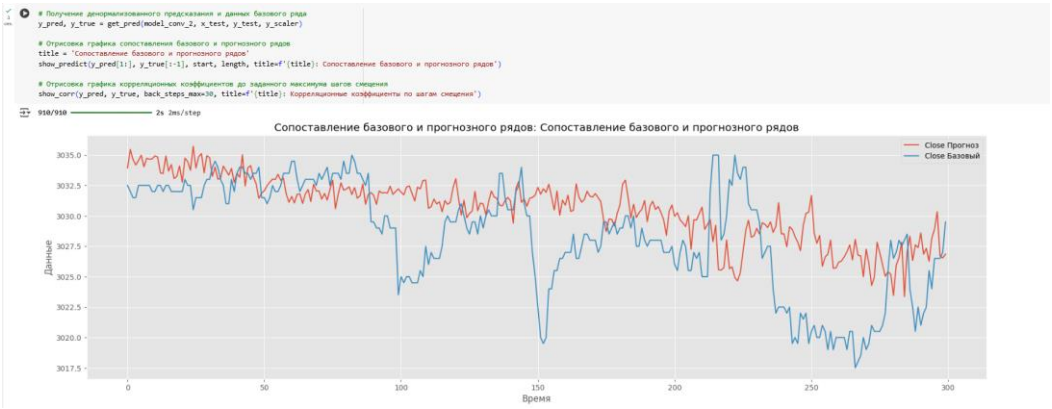


Рисунок 35. Сопоставление базового и прогнозного рядов

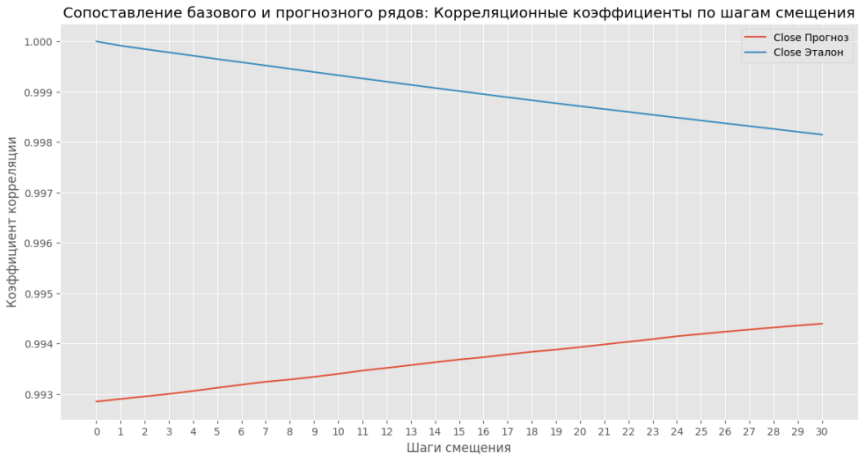


Рисунок 36. График корреляционных коэффициентов

Далее напишем код для предсказания на тестовой выборке и выполним преобразование предсказаний и правильных ответов обратно в исходный масштаб.

```
[36] # Получение предсказаний на тестовой выборке (в масштабе [0,1])
     y_pred_scaled = model_conv_2.predict(test_datagen_multi)

     # Преобразование предсказаний и правильных ответов обратно в исходный масштаб
     y_pred_reshaped = y_pred_scaled.reshape(-1, 1)
     y_test_reshaped = y_test_multi.reshape(-1, 1)

     y_pred_unscaled = y_scaler.inverse_transform(y_pred_reshaped).reshape(-1, FORECAST_STEPS)
     y_test_unscaled = y_scaler.inverse_transform(y_test_reshaped).reshape(-1, FORECAST_STEPS)

1440/1440 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step
```

Рисунок 37. Код для предсказания на тестовой выборке

Далее выполним построение 10 графиков, которые отображают истинные и предсказанные значения для каждого горизонта прогноза.

```python
# Построение 10 графиков: истинные и предсказанные значения для каждого горизонта прогноза
plt.figure(figsize=(15, 20))
for h in range(FORECAST_STEPS):
    plt.subplot(5, 2, h+1)
    true_series = y_test_unscaled[:, h]
    pred_series = y_pred_unscaled[:, h]
    plt.plot(true_series, label='Истинный ряд')
    plt.plot(pred_series, label='Прогноз на {} шаг'.format(h+1))
    plt.title(f'Горизонт {h+1}')
    plt.xlabel('Индекс образца (в тестовой выборке)')
    plt.ylabel('Значение показателя')
    plt.legend()
plt.tight_layout()
plt.show()
```

Рисунок 38. Построение 10 графиков

Затем после выполнения кода на рисунке 91, посмотрим на построенные графики.

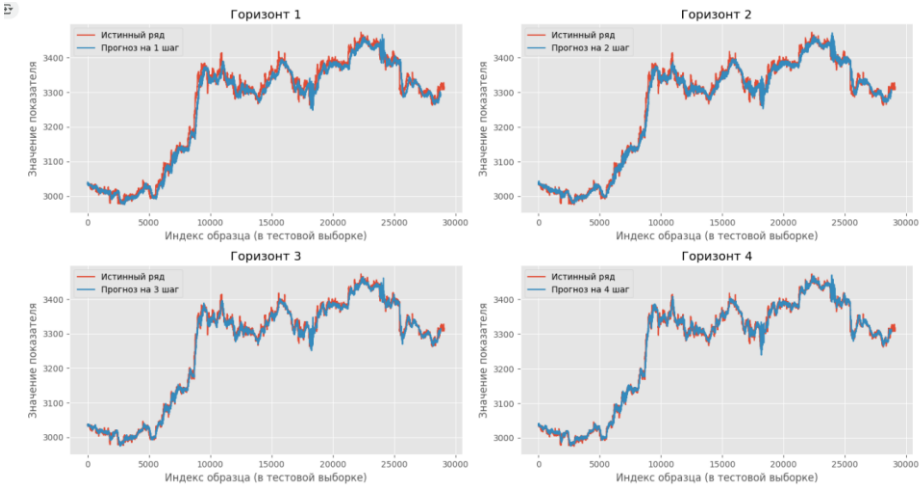


Рисунок 39. Отображение графиков для 1 – 4 горизонтов

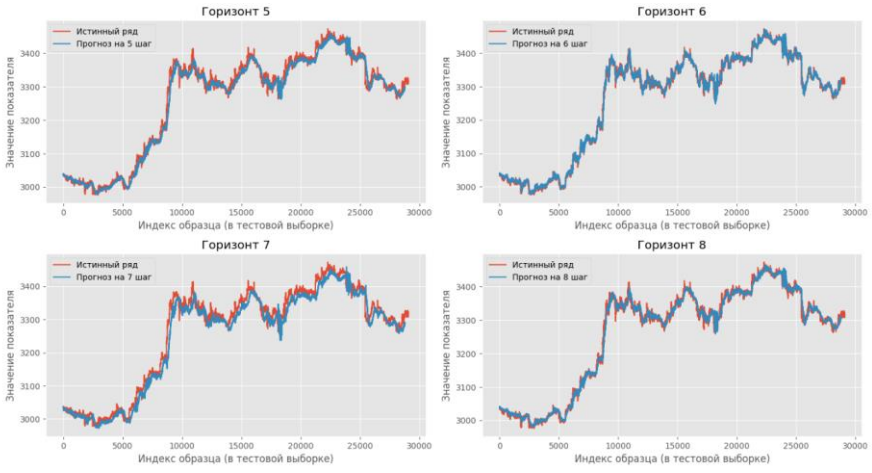


Рисунок 40. Отображение графиков для 5 – 8 горизонтов

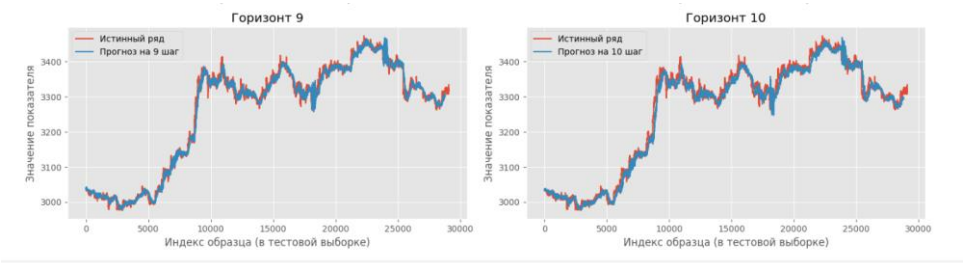


Рисунок 41. Отображение графиков для 9 и 10 горизонтов

Задание 2.

Условие: необходимо использовать базу акций Лукойла.

Сделать несколько усовершенствований в предсказании временного ряда. Добавить к исходному сигналу расширенные данные:

— попарные разности каналов

— модули попарных разностей каналов

— попарные произведения каналов

— обратное значение каналов x_new = 1/(x + 1e-3)

— первые производные каналов (x[n] - x[n-1])

— вторые производные каналов (x[n] - 2*x[n-1] + x[n-2])

Применить абсолютно новый подход. Сделать больший “просмотр сети в прошлое”, при формировании входного сигнала использовать:

— 100 точек с шагом назад по 1,

— 100 точек с шагом назад по 10 (или сами точки, или среднее по отрезку в 10 точек).

— Объединить эти точки

Для выполнения данного задания сначала выполним импорт необходимых библиотек.

Выполним загрузку датасета, а после выполним чтение данных в таблице с удалением ненужных столбцов и вывод размерностей получившихся таблиц.

Далее выполним создание общего набора данных из двух датасетов и выведем название столбцов.

```
# Создание общего набора данных из двух датасетов

data = pd.concat([data16_17,data18_19])
data = data.reset_index(drop = True)

# Проверка формы данных
print(data.shape)
```

```
(481872, 5)
```

```
# Получение названий столбцов

col = data.columns
print(col)
```

```
Index(['OPEN', 'MAX', 'MIN', 'CLOSE', 'VOLUME'], dtype='object')
```

Рисунок 42. Вывод названий столбцов

Далее зададим циклы для столбцов таким образом, чтобы происходил перебор всех возможных пар:

— попарные разности каналов

— модули попарных разностей каналов

— попарные произведения каналов

— обратное значение каналов x_new = 1/(x + 1e-3)

— первые производные каналов (x[n] - x[n-1])

— вторые производные каналов (x[n] - 2*x[n-1] + x[n-2])

```
# Задание циклов для столбцов таким образом, чтобы происходил перебор всех возможных пар:
for i in range(col.shape[0]): # Для всех пар
    for j in range(i + 1 , col.shape[0]): # Расчет
        data[col[i] + '-' + col[j]] = data[col[i]] - data[col[j]] # Разности
        data['|' + col[i] + '-' + col[j] + '|'] = abs(data[col[i]] - data[col[j]]) # Модулей разностей
        data[col[i] + '*' + col[j]] = data[col[i]] * data[col[j]] # Произведения

# Для каждого столбца 'OPEN', 'MAX', 'MIN', 'CLOSE', 'VOLUME' расчет:
for i in col:
    # Обратные значения. 1e-3 в формуле нужно, чтобы случайно не разделить на 0
    data['Обратный ' + i] = 1 / (data[i] + 1e-3)
    # Создание пустого столбца
    data['Производная от ' + i] = np.nan
    # При помощи срезов расчет первых производных, .reset_index(drop=True) нужен для корректных расчетов
    data['Производная от ' + i][1:] = data[i][1:].reset_index(drop=True) - data[i][:-1].reset_index(drop=True)
    # Создание пустого столбца
    data['Вторая производная от ' + i] = np.nan
    # При помощи срезов расчет вторых производных
    data['Вторая производная от ' + i][2:] = data[i][2:].reset_index(drop=True) - 2 * data[i][1:-1].reset_index(drop=True) + data[i][:-2].reset_index(drop=True
```

```
print(data.columns)
```

```
Index(['OPEN', 'MAX', 'MIN', 'CLOSE', 'VOLUME', 'OPEN-MAX', '|OPEN-MAX|',
       'OPEN*MAX', 'OPEN-MIN', '|OPEN-MIN|', 'OPEN*MIN', 'OPEN-CLOSE',
       '|OPEN-CLOSE|', 'OPEN*CLOSE', 'OPEN-VOLUME', '|OPEN-VOLUME|',
       'OPEN*VOLUME', 'MAX-MIN', '|MAX-MIN|', 'MAX*MIN', 'MAX-CLOSE',
       '|MAX-CLOSE|', 'MAX*CLOSE', 'MAX-VOLUME', '|MAX-VOLUME|', 'MAX*VOLUME',
       'MIN-CLOSE', '|MIN-CLOSE|', 'MIN*CLOSE', 'MIN-VOLUME', '|MIN-VOLUME|',
       'MIN*VOLUME', 'CLOSE-VOLUME', '|CLOSE-VOLUME|', 'CLOSE*VOLUME',
       'Обратный OPEN', 'Производная от OPEN', 'Вторая производная от OPEN',
       'Обратный MAX', 'Производная от MAX', 'Вторая производная от MAX',
       'Обратный MIN', 'Производная от MIN', 'Вторая производная от MIN',
       'Обратный CLOSE', 'Производная от CLOSE', 'Вторая производная от CLOSE',
       'Обратный VOLUME', 'Производная от VOLUME',
       'Вторая производная от VOLUME'],
      dtype='object')
```

Рисунок 43. Задание циклов для столбцов

Далее выполним просмотр результатов.

	OPEN	MAX	MIN	CLOSE	VOLUME	OPEN-MAX	|OPEN-MAX|	OPEN*MAX	OPEN-MIN	|OPEN-MIN|	...	вторая производная от MAX	Обратный MIN	Производная от MIN	вторая производная от MIN	Обратный CLOSE	Производная от CLOSE	вторая производная от CLOSE	Обратный VOLUME	Производная от VOLUME	вторая производная от VOLUME
0	2351.0	2355.8	2350.0	2350.0	2547	-4.8	4.8	5538485.80	1.0	1.0	...	NaN	0.000426	NaN	NaN	0.000426	NaN	NaN	0.000393	NaN	NaN
1	2352.9	2355.7	2350.0	2355.7	195	-2.8	2.8	5542726.53	2.9	2.9	...	NaN	0.000426	0.0	NaN	0.000425	5.7	NaN	0.005128	-2352.0	NaN
2	2355.6	2356.0	2351.4	2354.1	257	-0.4	0.4	5549793.60	4.2	4.2	...	0.4	0.000425	1.4	1.4	0.000425	-1.6	-7.3	0.003891	62.0	2414.0
3	2354.5	2355.0	2351.2	2353.7	763	-0.5	0.5	5544847.50	3.3	3.3	...	-1.3	0.000425	-0.2	-1.6	0.000425	-0.4	1.2	0.001311	506.0	444.0
4	2353.1	2353.9	2353.1	2353.6	231	-0.8	0.8	5538962.09	0.0	0.0	...	-0.1	0.000425	1.9	2.1	0.000425	-0.1	0.3	0.004329	-532.0	-1038.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
481867	5183.5	5183.5	5183.5	5183.5	31523	0.0	0.0	26868672.25	0.0	0.0	...	11.5	0.000193	16.0	18.0	0.000193	11.5	10.0	0.000032	29174.0	29480.0
481868	5183.5	5183.5	5183.5	5183.5	5090	0.0	0.0	26868672.25	0.0	0.0	...	-11.5	0.000193	0.0	-16.0	0.000193	0.0	-11.5	0.000196	-26433.0	-55607.0
481869	5183.5	5183.5	5183.5	5183.5	230	0.0	0.0	26868672.25	0.0	0.0	...	0.0	0.000193	0.0	0.0	0.000193	0.0	0.0	0.004348	-4860.0	21573.0
481870	5183.5	5183.5	5183.5	5183.5	5	0.0	0.0	26868672.25	0.0	0.0	...	0.0	0.000193	0.0	0.0	0.000193	0.0	0.0	0.199960	-225.0	4635.0
481871	5183.5	5183.5	5183.5	5183.5	994	0.0	0.0	26868672.25	0.0	0.0	...	0.0	0.000193	0.0	0.0	0.000193	0.0	0.0	0.001006	989.0	1214.0

481872 rows × 50 columns

Рисунок 44. Просмотр результатов

Затем выполним использование всех столбцов, кроме первых двух и зададим переменную, для использования одной и той же архитектуры под разные матрицы.

Затем удалим пропуски и выполним преобразование в массив. Далее выполним нормализацию и масштабирование.

```
[15] # Масштабирование признаков и целевой переменной с помощью MinMaxScaler (отдельно для X и y)
     X_scaler = MinMaxScaler()
     y_scaler = MinMaxScaler()

[16] # Fit на обучающей выборке и трансформация всех частей данных
     X_train_scaled = X_scaler.fit_transform(X_train)
     X_val_scaled   = X_scaler.transform(X_val)
     X_test_scaled  = X_scaler.transform(X_test)

     y_train_scaled = y_scaler.fit_transform(y_train)
     y_val_scaled   = y_scaler.transform(y_val)
     y_test_scaled  = y_scaler.transform(y_test)
```

Рисунок 45. Нормализация данных

Далее выполним задание, которое требует использовать:

— 100 точек с шагом назад по 1

— 100 точек с шагом назад по 10 (или усреднённые по 10 точкам)

Для этого зададим следующие параметры.

По напишем функцию, которая будет выполнять глубокий просмотр сети в прошлое, для этого используется 100 временных шагов назад с шагом 1, также реализуется просмотр назад с шагом 10 или усреднение по отрезкам из 10 точек, после реализуется объединение двух частей.

```
# Используем TimeseriesGenerator для создания батчей последовательностей длины SEQ_LEN
train_generator = TimeseriesGenerator(
    X_train_scaled, y_train_scaled,
    length=SEQ_LEN, batch_size=BATCH_SIZE, stride=1, sampling_rate=1
)
val_generator = TimeseriesGenerator(
    X_val_scaled, y_val_scaled,
    length=SEQ_LEN, batch_size=BATCH_SIZE, stride=1, sampling_rate=1
)
test_generator = TimeseriesGenerator(
    X_test_scaled, y_test_scaled,
    length=SEQ_LEN, batch_size=BATCH_SIZE, stride=1, sampling_rate=1
)
```

Рисунок 46. Функция для выполнения задания

Далее выполним создание нейронной модели.

```
# Определяем архитектуру нейронной сети
n_features = X_train_scaled.shape[1]
model = Sequential()
model.add(Conv1D(filters=50, kernel_size=5, activation='linear',
                 input_shape=(SEQ_LEN, n_features)))
model.add(Flatten())
model.add(Dense(10, activation='linear'))
model.add(Dense(FORECAST_STEPS, activation='linear'))
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 196, 50)	12,300
flatten (Flatten)	(None, 9800)	0
dense (Dense)	(None, 10)	98,010
dense_1 (Dense)	(None, 1)	11

```
Total params: 110,321 (430.94 KB)
Trainable params: 110,321 (430.94 KB)
Non-trainable params: 0 (0.00 B)
```

Рисунок 47. Создание модели

Затем выполним обучение созданной модели нейронной сети.

```
# Компиляция модели с оптимизатором Adam и функцией потерь MSE
model.compile(optimizer=Adam(), loss='mse')

# Обучение модели на генераторе обучающей выборки с проверкой на валидционной выборке
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    shuffle=False
)
```

```
Epoch 1/20
19265/19265 ━━━━━━━━━━━━━━━━━━━━ 77s 4ms/step - loss: 0.4157 - val_loss: 0.1917
Epoch 2/20
19265/19265 ━━━━━━━━━━━━━━━━━━━━ 71s 4ms/step - loss: 0.3690 - val_loss: 0.1902
Epoch 3/20
19265/19265 ━━━━━━━━━━━━━━━━━━━━ 72s 4ms/step - loss: 9.8067e-04 - val_loss: 0.0083
Epoch 4/20
19265/19265 ━━━━━━━━━━━━━━━━━━━━ 82s 4ms/step - loss: 8.6820e-04 - val_loss: 0.0261
Epoch 5/20
19265/19265 ━━━━━━━━━━━━━━━━━━━━ 66s 3ms/step - loss: 0.0029 - val_loss: 0.0377
Epoch 6/20
19265/19265 ━━━━━━━━━━━━━━━━━━━━ 71s 4ms/step - loss: 0.0031 - val_loss: 0.0521
Epoch 7/20
19265/19265 ━━━━━━━━━━━━━━━━━━━━ 69s 4ms/step - loss: 0.0014 - val_loss: 0.0113
Epoch 8/20
19265/19265 ━━━━━━━━━━━━━━━━━━━━ 68s 4ms/step - loss: 0.0014 - val_loss: 0.0175
Epoch 9/20
 8759/19265 ━━━━━━━━━━━━━━━━━━━━ 32s 3ms/step - loss: 0.0019
```

Рисунок 48. Обучение модели

Напишем функцию для построения графика процесса обучения данной сети.

```
[21] # Рисование графиков прошедшей фазы обучения
     title = 'График'
     fig = plt.figure(figsize=(14, 7))
     plt.plot(history.history['loss'], label='Ошибка на обучающем наборе')
     plt.plot(history.history['val_loss'], label='Ошибка на проверочном наборе')
     plt.title(f'{title}: График прогресса обучения')
     # Указание показывать только целые метки шкалы оси x
     fig.gca().xaxis.get_major_locator().set_params(integer=True)
     plt.xlabel('Эпоха обучения')
     plt.ylabel('Средняя ошибка')
     plt.legend()
     plt.show()
```

Рисунок 49. График процесса обучения

Далее выполним оценку данных, для этого построим график предсказания vs факт и график корреляции ошибок по лагам.

```
        x_batch, y_batch = generator[i]
        batch_pred = model.predict(x_batch, verbose=0)
        pred_scaled.append(batch_pred)
        y_true_scaled.append(y_batch)

    # Объединяем все батчи
    pred_scaled = np.concatenate(pred_scaled)
    y_true_scaled = np.concatenate(y_true_scaled)

    # Преобразуем обратно к исходному масштабу
    pred = y_scaler.inverse_transform(pred_scaled)
    y_true = y_scaler.inverse_transform(y_true_scaled)

    # Проверяем размерности
    if pred.ndim == 1:
        pred = pred.reshape(-1, 1)
    if y_true.ndim == 1:
        y_true = y_true.reshape(-1, 1)

    plt.figure(figsize=(14, 7))
    plt.plot(y_true[:, 0], label='Фактические значения', color='blue')
    plt.plot(pred[:, 0], label='Прогнозные значения', color='red', alpha=0.7)
    plt.title('Сопоставление фактических и прогнозных значений CLOSE')
    plt.xlabel('Временной шаг')
    plt.ylabel('Значение CLOSE')
    plt.legend()
    plt.grid(True)
    plt.show()

    def correlate(a, b):
        return np.corrcoef(a, b)[0, 1]

    y_len = len(y_true)
    steps = range(0, min(lags, y_len//2))

    cross_corr = [correlate(y_true[:y_len-step, 0], pred[step:, 0]) for step in steps]
    auto_corr = [correlate(y_true[:y_len-step, 0], y_true[step:, 0]) for step in steps]

    plt.figure(figsize=(14, 7))
    plt.plot(steps, cross_corr, label='Прогноз vs Факт')
    plt.plot(steps, auto_corr, label='Автокорреляция факта')
    plt.title('Корреляция с разными лагами')
    plt.xticks(steps[::5])
    plt.xlabel('Шаги смещения')
    plt.ylabel('Коэффициент корреляции')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Рисунок 50. Функция для построения графиков

Посмотрим на графики полученные в результате выполнения функции.

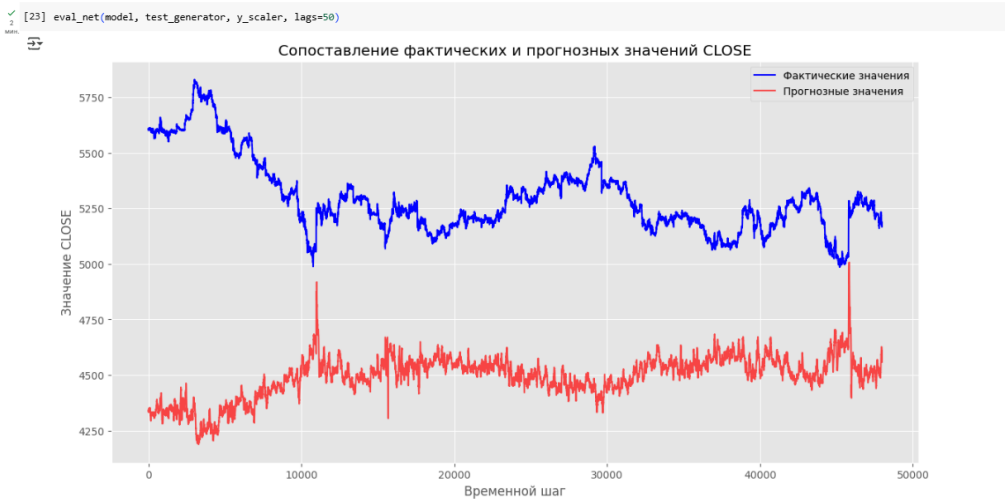


Рисунок 51. График фактических и прогнозных значений

Задание 3.

Условие: необходимо использовать базу трафика с сайта компании. Написать модель для прогнозирования трафика.

При параметре length = 60, добиться максимально точного соответствия между графиками корреляции исходного сигнала и прогноза.

Начнем с импорта необходимых библиотек для работы. Далее назначим размер и стиль для графиков. Затем выполним загрузку базы трафика с сайта компании.

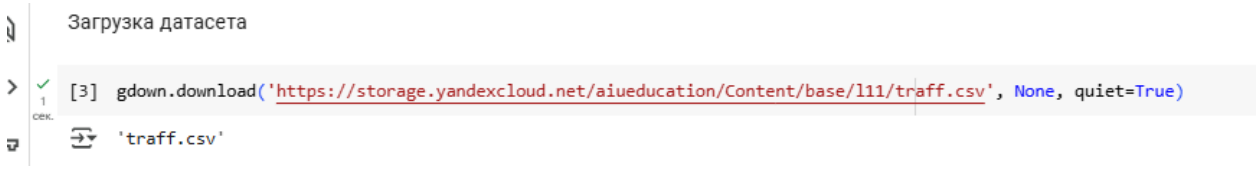


Рисунок 53. Загрузка базы

Далее выполним загрузку и вывод данных базы.

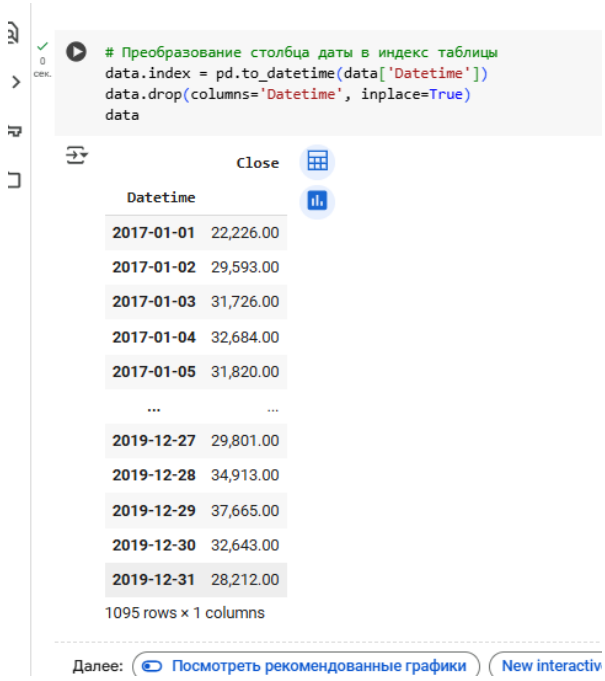
```
[4] # Загрузка данных
    data = pd.read_csv('traff.csv', header=None, names=['Datetime', 'Close'])
    data
```

	Datetime	Close
0	1/1/2017	22,226.00
1	1/2/2017	29,593.00
2	1/3/2017	31,726.00
3	1/4/2017	32,684.00
4	1/5/2017	31,820.00
...	...	...
1090	12/27/2019	29,801.00
1091	12/28/2019	34,913.00
1092	12/29/2019	37,665.00
1093	12/30/2019	32,643.00
1094	12/31/2019	28,212.00

1095 rows × 2 columns

Рисунок 54. Загрузка данных

Выполним преобразование столбца даты в индекс таблицы и удаление таблицы и посмотрим на результат.



```
# Преобразование столбца даты в индекс таблицы
data.index = pd.to_datetime(data['Datetime'])
data.drop(columns='Datetime', inplace=True)
data
```

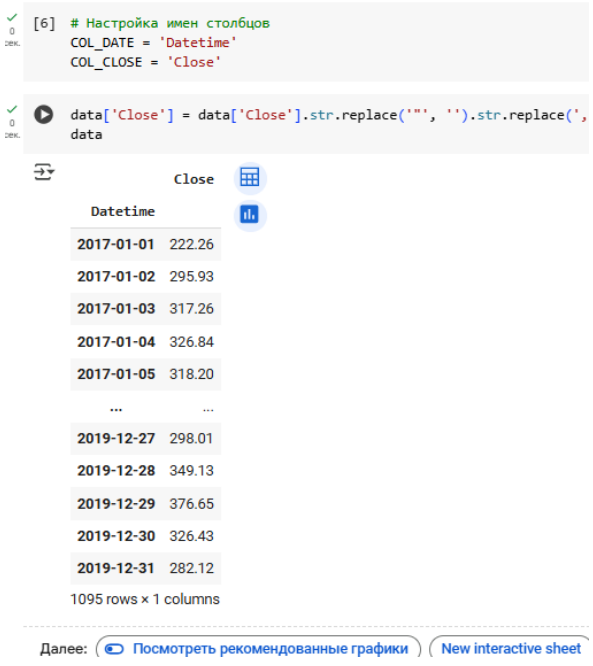
Datetime	Close
2017-01-01	22,226.00
2017-01-02	29,593.00
2017-01-03	31,726.00
2017-01-04	32,684.00
2017-01-05	31,820.00
...	...
2019-12-27	29,801.00
2019-12-28	34,913.00
2019-12-29	37,665.00
2019-12-30	32,643.00
2019-12-31	28,212.00

1095 rows × 1 columns

Далее: Посмотреть рекомендованные графики | New interactiv

Рисунок 55. Преобразование столбца даты

Далее выполним настройку имен столбцов.



```
[6] # Настройка имен столбцов
    COL_DATE = 'Datetime'
    COL_CLOSE = 'Close'
```

```
data['Close'] = data['Close'].str.replace('"', '').str.replace(',
data
```

Datetime	Close
2017-01-01	222.26
2017-01-02	295.93
2017-01-03	317.26
2017-01-04	326.84
2017-01-05	318.20
...	...
2019-12-27	298.01
2019-12-28	349.13
2019-12-29	376.65
2019-12-30	326.43
2019-12-31	282.12

1095 rows × 1 columns

Далее: Посмотреть рекомендованные графики | New interactive sheet

Рисунок 56. Вывод данных

Затем выполним отображение временного ряда в графическом виде.

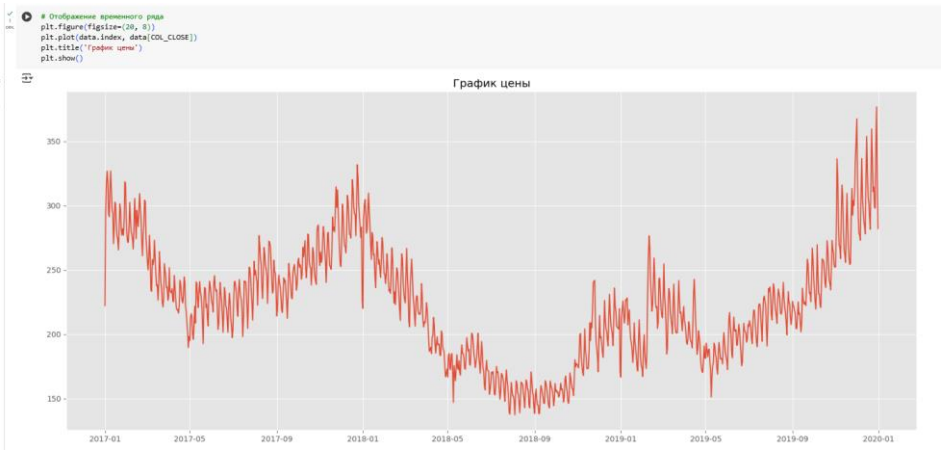


Рисунок 57. Отображение в графическом виде

Далее зададим гиперпараметры и выполним расчет индикаторов относительной доходности).

```
[9] # Задание гиперпараметров
    INDICATOR_DEPTH = 6                    # Глубина расчета
    PREDICT_LAG = 1                        # Количество шагов
```

```
# Расчет индикаторов относительной доходности
for i in range(1, INDICATOR_DEPTH + 1):
    indicator_change = f'Close_chng_{i}'
    # Относительная доходность в сотых долях за период i шагов назад
    data[indicator_change] = data[COL_CLOSE].pct_change(i)

# Проверка результата
print(data.shape)
data[:5]
```

```
(1095, 7)
```

Datetime	Close	Close_chng_1	Close_chng_2	Close_chng_3	Close_chng_4	Close_chng_5	Close_chng_6
2017-01-01	222.26	NaN	NaN	NaN	NaN	NaN	NaN
2017-01-02	295.93	0.331459	NaN	NaN	NaN	NaN	NaN
2017-01-03	317.26	0.072078	0.427427	NaN	NaN	NaN	NaN
2017-01-04	326.84	0.030196	0.104450	0.470530	NaN	NaN	NaN
2017-01-05	318.20	-0.026435	0.002963	0.075254	0.431657	NaN	NaN

Рисунок 58. Задание гиперпараметров

Затем выполним удаление строк с неполными данными и выведем результат.

```
[11] # Удаление строк с неполными данными
     data.dropna(inplace=True)

     # Проверка результата
     print(data.shape)
     data[:5]
```

```
(1089, 7)
```

Datetime	Close	Close_chng_1	Close_chng_2	Close_chng_3	Close_chng_4	Close_chng_5	Close_chng_6
2017-01-07	291.39	-0.007324	-0.084255	-0.108463	-0.081542	-0.015341	0.311032
2017-01-08	312.44	0.072240	0.064386	-0.018102	-0.044058	-0.015193	0.055790
2017-01-09	326.90	0.046281	0.121864	0.113647	0.027341	0.000184	0.030385
2017-01-10	312.99	-0.042551	0.001760	0.074127	0.066260	-0.016373	-0.042375
2017-01-11	297.38	-0.049874	-0.090303	-0.048201	0.020557	0.013082	-0.065431

Рисунок 59. Удаление строк с неполными данными

Далее получим общее количество строк в наборе данных, вычислим индекс конца обучающего и валидационного набора и выполним формирование обучающего, валидационного и тестового набора данных (80/10/10). Затем масштабируем признаки (все колонки, включая 'Close') для обучающего набора и применяем тот же масштабатор к валидационному и тестовому наборам. Масштабируем целевую переменную ('Close') отдельно.

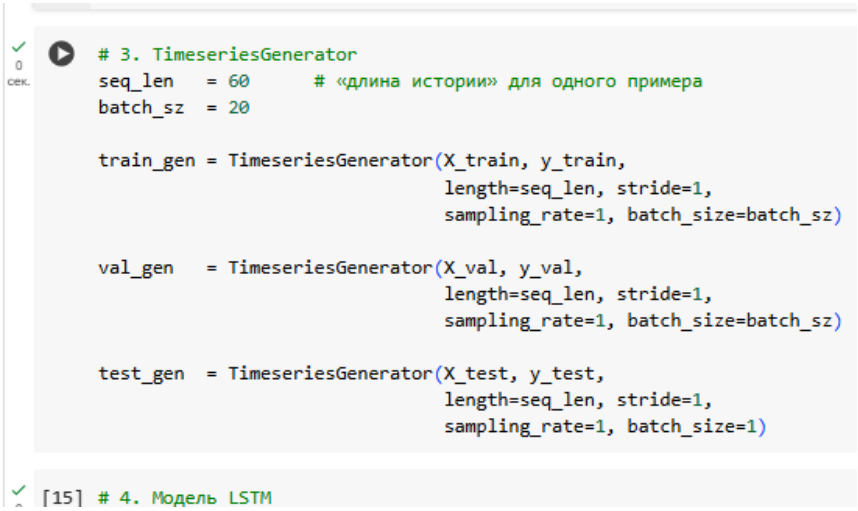
```
# 2. Масштабирование
feature_scaler = MinMaxScaler()
target_scaler  = MinMaxScaler()

X_train = feature_scaler.fit_transform(train_df)
X_val   = feature_scaler.transform(val_df)
X_test  = feature_scaler.transform(test_df)

y_train = target_scaler.fit_transform(train_df[['Close']])
y_val   = target_scaler.transform(val_df[['Close']])
y_test  = target_scaler.transform(test_df[['Close']])
```

```
[14] # 3. TimeseriesGenerator
     seq_len   = 60      # «длина истории» для одного примера
     batch_sz  = 20
```

Рисунок 60. Масштабирование признаков

Далее будем использовать TimeseriesGenerator, установим параметры.

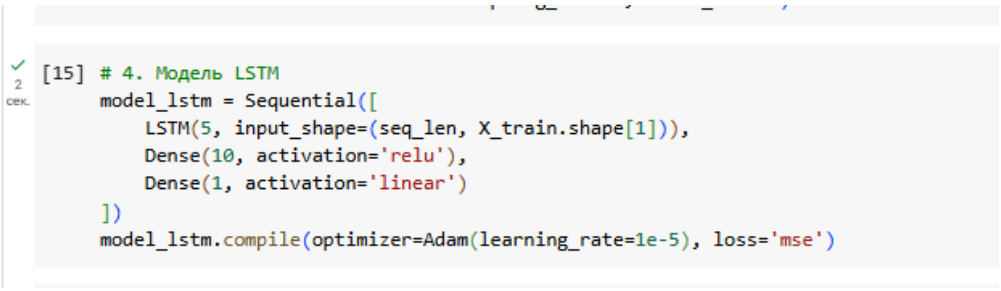
Создадим генераторы для обучающих, валидационных и тестовых данных.



```
# 3. TimeseriesGenerator
seq_len   = 60      # «длина истории» для одного примера
batch_sz  = 20

train_gen = TimeseriesGenerator(X_train, y_train,
                                length=seq_len, stride=1,
                                sampling_rate=1, batch_size=batch_sz)

val_gen   = TimeseriesGenerator(X_val, y_val,
                                length=seq_len, stride=1,
                                sampling_rate=1, batch_size=batch_sz)

test_gen  = TimeseriesGenerator(X_test, y_test,
                                length=seq_len, stride=1,
                                sampling_rate=1, batch_size=1)

[15] # 4. Модель LSTM
```

Рисунок 61. Использование TimeseriesGenerator

После выполним создание и обучение модели LSTM 126).



```
[15] # 4. Модель LSTM
model_lstm = Sequential([
    LSTM(5, input_shape=(seq_len, X_train.shape[1])),
    Dense(10, activation='relu'),
    Dense(1, activation='linear')
])
model_lstm.compile(optimizer=Adam(learning_rate=1e-5), loss='mse')
```

Рисунок 62. Создание модели

Далее простроим график процесса обучения.

Рисунок 63. График процесса обучения

Затем выполним визуализацию графиков сравнения прогноза и факта и крафик корреляции прогноза. Посмотрим на полученные графики.

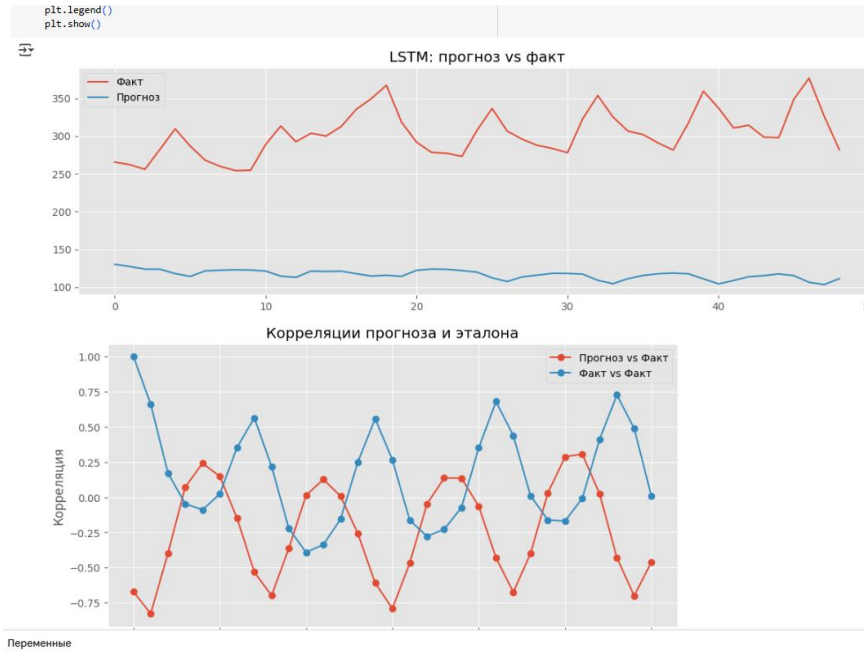


Рисунок 64. Полученные графики

Отсюда видно, что полученная модель LSTM при параметре length = 60 успешно обучена на данных трафика сайта и не обеспечивает высокое соответствие между фактическими и предсказанными значениями. График прогноза демонстрирует расхождение с реальными данными, а график корреляций показывает низкую степень совпадения между прогнозом и эталонным сигналом.

Вывод: в ходе выполнения лабораторной работы была изучена обработка новых типов данных – временные ряды. Также были рассмотрены особенности, сложности, которые возникают при работе и способы их устранения. Также были рассмотрены архитектуры, для дальнейшего использования.