МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования «Балтийский федеральный университет имени Иммануила Канта»

ОНК «Институт высоких технологий»

КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ «Интерпретируемое машинное обучение»

НА ТЕМУ «Прогнозирование стоимостного объема экспорта Москвы с использованием нейронных сетей и AR/MA моделей»

Выполнила Бабина П.В. _____
студент очной формы обучения 1-го курса
направления подготовки 02.04.03

«Математическое обеспечение и администрирование информационных систем»,
профиль обучения «Банковские информационные технологии»

Руководитель Ткаченко С.Н. _____

Калининград, 2023

Оглавление

Введение

В современной экономике прогнозирование стоимостного объема экспорта является важной задачей для управления экономическим развитием регионов и стран в целом. Одним из крупнейших экспортеров в России является город Москва, который имеет огромный экономический потенциал и множество возможностей для развития экспорта.

В данной курсовой работе будет рассмотрена задача прогнозирования стоимостного объема экспорта Москвы с использованием нейронных сетей и AR/MA моделей. На основе анализа экономических данных о ежемесячных изменениях будет построена модель, которая позволит прогнозировать стоимостной объем экспорта Москвы на будущие периоды.

Одним из основных преимуществ использования нейронных сетей и AR/MA моделей является их способность учитывать нелинейные зависимости между экономическими показателями и предсказывать сложные временные ряды. Кроме того, данные методы являются универсальными и могут быть использованы для прогнозирования различных экономических показателей.

Целью данной работы является разработка и тестирование модели прогнозирования стоимостного объема экспорта Москвы с использованием нейронных сетей и AR/MA моделей. Для достижения этой цели необходимо выполнить следующие задачи: провести анализ предметной области, собрать и подготовить данные для построения модели, разработать и реализовать модель, провести анализ результатов и оценить эффективность модели.

Глава 1. Теория

1.1. Нейронные сети

Нейронные сети — это класс алгоритмов машинного обучения, которые имитируют функционирование человеческого мозга. Они состоят из нейронов, которые соединены между собой и обрабатывают информацию с помощью математических операций.

Нейронные сети могут использоваться для решения широкого круга задач машинного обучения, таких как классификация, регрессия, кластеризация, сегментация и прогнозирование. Они могут обрабатывать сложные и нелинейные зависимости в данных, которые не могут быть решены традиционными методами машинного обучения.

Основным преимуществом нейронных сетей является их способность к автоматическому извлечению признаков из данных, что упрощает процесс обучения. Кроме того, они могут быть обучены на больших объемах данных, что позволяет получать более точные прогнозы и результаты.

Существует множество различных типов нейронных сетей, включая персептроны, рекуррентные нейронные сети, сверточные нейронные сети, глубокие нейронные сети и многие другие. Каждый тип нейронной сети имеет свои уникальные особенности и подходит для решения определенных задач машинного обучения.

1.1.1. RNN

Рекуррентные нейронные сети (RNN) — это класс нейронных сетей, которые используются для обработки последовательностей данных, таких как тексты, временные ряды и звуковые сигналы. RNN могут моделировать зависимости между элементами последовательности, что позволяет им извлекать контекстуальную информацию и сохранять ее в своем внутреннем состоянии.

RNN состоит из повторяющихся блоков нейронов, которые обрабатывают каждый элемент последовательности и передают информацию о своем состоянии следующему блоку. Каждый блок RNN имеет два типа входов: текущий элемент последовательности и информацию о предыдущем состоянии. Это позволяет модели учитывать контекст и историю последовательности.

Одна из основных проблем RNN заключается в том, что они могут страдать от затухающего градиента, когда информация о предыдущем состоянии затухает по мере продвижения через блоки RNN. Чтобы решить эту проблему, были разработаны различные типы RNN, такие как LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit), которые обеспечивают более эффективное сохранение информации о предыдущем состоянии.

RNN широко используются в задачах обработки естественного языка, например, в задачах машинного перевода, классификации текстов и генерации текста. Они также применяются для анализа временных рядов и прогнозирования, например, в задачах прогнозирования цен на акции или погоды. RNN являются мощным инструментом машинного обучения и позволяют эффективно работать с различными типами последовательностей данных.

1.1.2. LSTM

LSTM (Long Short-Term Memory) — это вид рекуррентных нейронных сетей, который был разработан для решения проблемы затухания градиентов в RNN. Он позволяет эффективно сохранять и использовать информацию внутри последовательностей.

Основная идея LSTM заключается в том, что каждый блок имеет внутреннюю память, которая может сохранять информацию на длительный период времени. Эта память может контролироваться с помощью специальных структур, называемых воротами.

Каждый блок LSTM состоит из четырех взаимодействующих между собой слоев:

1. Слой входного фильтра (Input Layer Gate) - принимает на вход последовательность данных.
2. Слой фильтра забывания (Forget Gate Layer) - контролирует поток информации и управляет тем, какая информация должна быть забыта, и какая сохранена в долговременной памяти.
3. Слой долговременной памяти (Long-Term Memory Layer) - хранит информацию, которую модель должна запомнить на длительный период времени.
4. Слой выходного фильтра (Output Gate Layer) - генерирует выходные данные на основе информации, хранящейся в долговременной памяти и текущего входа.

LSTM может использоваться для различных задач, включая обработку естественного языка, прогнозирование временных рядов и генерацию текста. Он является мощным инструментом машинного обучения, который позволяет эффективно обрабатывать и использовать информацию внутри последовательностей.

1.1.3. GRU

GRU (Gated Recurrent Unit) — это модификация рекуррентных нейронных сетей, которая также была разработана для решения проблемы затухания градиентов в RNN и может использоваться для обработки последовательностей.

В слоях GRU (Gated Recurrent Unit) обычно используются следующие типы слоев:

1. Слой входного фильтра (Input Layer Gate) - принимает на вход последовательность данных.

2. Слой сброса (Reset Gate Layer) - определяет, какую информацию необходимо забыть из предыдущего состояния.
3. Слой обновления (Update Gate Layer) - определяет, какую информацию необходимо добавить в новое состояние.
4. Слой выходного фильтра (Output Gate Layer) - генерирует выходные данные на основе информации, хранящейся в долговременной памяти и текущего входа.

GRU может использоваться для различных задач, таких как обработка естественного языка, музыкальный анализ и прогнозирование временных рядов. Он обычно используется в случаях, когда требуется меньшее количество вычислительных ресурсов, чем LSTM, но при этом сохраняется высокая точность предсказаний.

1.1.4. BiLSTM

BiLSTM (Bidirectional Long Short-Term Memory) — это модель нейронных сетей, которая объединяет две LSTM-сети, работающие в прямом и обратном направлении для обработки последовательностей.

Обычная LSTM-сеть работает только в одном направлении и не учитывает информацию из будущих элементов последовательности. В отличие от этого, BiLSTM использует две LSTM-сети, одну работающую в прямом направлении, а другую - в обратном направлении. Таким образом, модель может учитывать как прошлую, так и будущую информацию о последовательности.

Каждая из LSTM-сетей в BiLSTM имеет свою собственную внутреннюю память и выход. Когда последовательность обрабатывается в прямом направлении, выходы из слоев сохраняются в памяти и используются для обработки последовательности в обратном направлении. Таким образом, BiLSTM имеет более полное представление о контексте последовательности.

В слоях BiLSTM (Bidirectional Long Short-Term Memory) обычно используются следующие типы слоев:

1. Прямой слой LSTM (Forward LSTM Layer) - обрабатывает последовательность в прямом направлении (от начала до конца).
2. Обратный слой LSTM (Backward LSTM Layer) - обрабатывает последовательность в обратном направлении (от конца до начала).
3. Слой объединения (Merge Layer) - объединяет выходы прямого и обратного слоев LSTM.
4. Слой выходных данных (Output Layer) - генерирует выходные данные на основе информации, полученной от слоя объединения.

BiLSTM используется для множества задач, таких как обработка естественного языка, распознавание речи, классификация изображений и многих других. Он является одним из наиболее эффективных методов для обработки последовательностей и предсказания значений на основе контекста.

1.2. AR/MA модели

AR/MA модели являются одними из наиболее распространенных моделей временных рядов. AR (авторегрессия) модель описывает зависимость текущего значения временного ряда от предыдущих значений ряда, т.е. ряд рассматривается как комбинация его предыдущих значений. MA (скользящее среднее) модель описывает зависимость текущего значения ряда от случайных ошибок (шума), которые могут быть обусловлены какими-то внешними факторами.

AR модели представляют собой линейную комбинацию предыдущих значений временного ряда и оцениваются с помощью методов наименьших квадратов или максимального правдоподобия. MA модели представляют собой линейную комбинацию случайных ошибок и также оцениваются с помощью методов наименьших квадратов или максимального правдоподобия.

AR/MA модели широко используются для прогнозирования экономических показателей, таких как инфляция, производительность, валютные курсы, а также для прогнозирования погоды и других временных рядов.

1.2.1. ARMA

ARMA (Autoregressive Moving Average) — это модель, используемая для анализа временных рядов. Она сочетает в себе два подхода - авторегрессию (AR) и скользящее среднее (MA).

AR-модель представляет собой линейную регрессию, где зависимая переменная — это текущее значение временного ряда, а независимыми переменными являются его предыдущие значения. Модель MA представляет собой линейную комбинацию ошибок предыдущих значений временного ряда.

ARMA-модель объединяет эти два подхода и использует лаговые переменные, чтобы описать текущее значение временного ряда. Коэффициенты AR и MA определяются на основе данных временного ряда и выбираются таким образом, чтобы минимизировать ошибку модели. Они могут быть оценены с помощью метода максимального правдоподобия или метода наименьших квадратов.

ARMA-модель может использоваться для прогнозирования значений временных рядов на основе предыдущих значений и ошибок. Это может быть полезно для прогнозирования будущих цен, объемов продаж и других показателей. Однако, ARMA-модель имеет свои ограничения, такие как неспособность учитывать тренды и сезонность в данных. Для решения этих проблем используются более сложные модели временных рядов, такие как ARIMA и SARIMA.

1.2.2. ARIMA

ARIMA (Autoregressive Integrated Moving Average) — это расширенная версия модели ARMA, которая также учитывает стационарность и тренды в данных временных рядов. ARIMA включает в себя три компоненты: авторегрессионную (AR), компоненту интеграции (I) и скользящее среднее (MA).

AR-компонента в ARIMA модели отражает зависимость текущего значения временного ряда от его предыдущих значений. MA-компонента в ARIMA модели учитывает корреляцию между текущим значением временного ряда и ошибками модели в прошлом. Компонента интеграции I включает в себя различные методы для обеспечения стационарности временных рядов, например, дифференцирование.

ARIMA-модель может быть использована для прогнозирования значений временных рядов с учетом трендов, сезонности и других характеристик данных. Однако, ее использование может быть ограничено,

если данные не являются стационарными или содержат большое количество шума. В таких случаях могут быть применены более сложные модели, такие как SARIMA, которая учитывает сезонность в данных.

1.2.3. SARIMA

SARIMA (Seasonal Autoregressive Integrated Moving Average) — это модификация ARIMA модели, которая учитывает сезонность в данных временных рядов. SARIMA включает в себя четыре компоненты: сезонную авторегрессионную (SAR), сезонную компоненту интеграции (SI), сезонное скользящее среднее (SMA) и обычную ARIMA.

SAR-компонента в SARIMA модели отражает зависимость текущего значения временного ряда от его предыдущих значений в определенный момент времени в сезонном цикле. SMA-компонента учитывает корреляцию между текущим значением временного ряда и ошибками модели в прошлом в определенный момент времени в сезонном цикле. Компонента интеграции SI включает в себя различные методы для обеспечения стационарности временных рядов, но только в определенный момент времени в сезонном цикле.

SARIMA модель может быть использована для прогнозирования значений временных рядов с учетом сезонности и других характеристик данных. Однако, ее использование может быть ограничено, если данные не являются стационарными или содержат большое количество шума. SARIMA также требует большего количества данных для обучения, чем простые модели, такие как ARIMA, и может быть менее эффективна в предсказании на длинный срок.

Глава 2. Реализация

2.1. Обработка данных

Загружаем файл "*export1.csv*" в объект *pandas DataFrame* и выводит первые пять строк таблицы (Рис. 1). Для наглядности отображаем исходный временной ряд (Рис. 2).

```
data = pd.read_csv("export1.csv", delimiter=";", decimal=",")
data.head()
```

	Дата	За месяц	Нарастающий итог с начала года	Темп прироста к предыдущему месяцу	Темп прироста к аналогичному месяцу прошлого года	Темп прироста с начала года к аналогичному периоду предыдущего года
0	01.01.2012	17.8	17.8	NaN	NaN	NaN
1	01.02.2012	20.1	37.8	12,90%	NaN	NaN
2	01.03.2012	17.1	54.9	-14,90%	NaN	NaN
3	01.04.2012	18.2	73.1	6,90%	NaN	NaN
4	01.05.2012	17.2	90.3	-5,60%	NaN	NaN

Рисунок 1. Исходные данные

Рисунок 2. Исходный временной ряд

Автокорреляция — это мера того, насколько коррелированы значения временного ряда с его прошлыми значениями. График ACF (Рис. 3) показывает, насколько сильно каждое значение временного ряда коррелирует с предыдущими значениями на разных временных отрезках, отсчитываемых в единицах времени (лагах).

Если на графике ACF значения корреляции на некоторых лагах значительно отличаются от нуля, то это может означать, что значения временного ряда на этих лагах сильно коррелируют между собой. В случае, если значения корреляции на всех лагах близки к нулю, это может говорить о том, что временной ряд является стационарным и не имеет сезонности или трендов.

На графике можно увидеть, что автокорреляционная функция имеет значительное значение на лагах 1-2, что указывает на сильную корреляцию между значениями на этих лагах. Далее автокорреляционная функция постепенно убывает и в конце становится не значимой. Это может свидетельствовать о том, что этот временной ряд является стационарным и не имеет трендов или сезонности.

Зная график автокорреляции, можно определить оптимальные параметры модели ARIMA (авторегрессионная интегрированная скользящая средняя) для прогнозирования временного ряда.

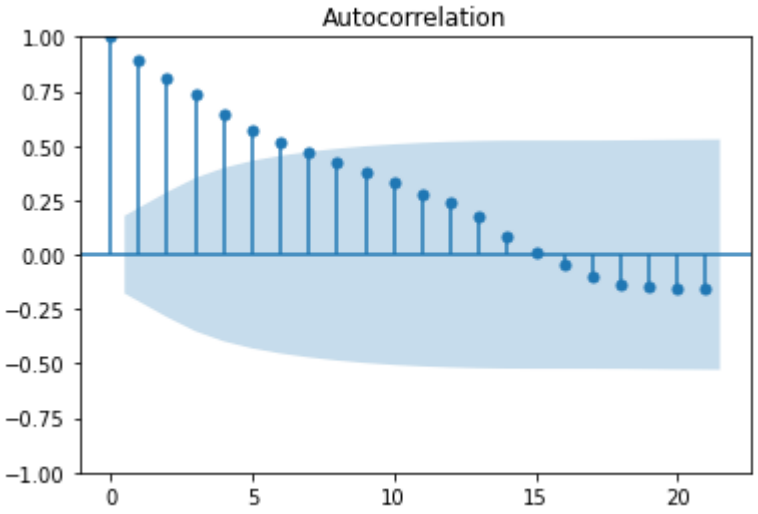


Рисунок 3. Автокорреляция

На графике (Рис. 4) показана функция частичной автокорреляции (PACF) для некоторых данных. Функция PACF — это мера корреляции между

значениями временного ряда и его отстающими значениями, учитывая все промежуточные значения (то есть исключая влияние более коротких лагов).

На графике можно увидеть, что функция PACF имеет значительное значение на лаге 1, а затем быстро убывает к нулю. Это может указывать на модель авторегрессии первого порядка (AR (1)), где каждое значение временного ряда зависит от его предыдущего значения.

Также на графике можно заметить, что значения на лагах 2-3 также превышают границы доверительного интервала, что может указывать на наличие некоторых дополнительных зависимостей между значениями временного ряда. Однако, при дальнейшем увеличении лагов значения функции PACF сильно колеблются и не превышают границы доверительного интервала, что указывает на отсутствие статистически значимых корреляций на более длинных лагах.

Используя этот график, можно также определить оптимальные параметры модели ARIMA для прогнозирования временного ряда.

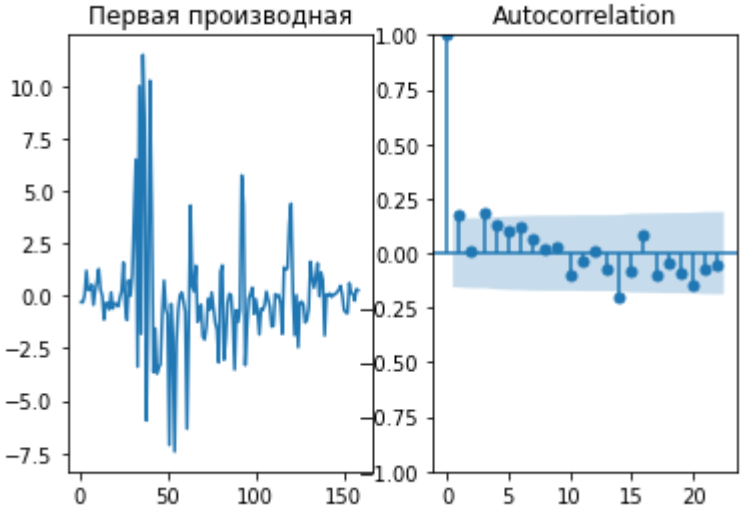


Рисунок 4. Первая производная и частичная автокорреляция

Преобразуем столбец "Дата" в формат datetime. Переменная **start_shift** устанавливает количество месяцев, на которые будет сдвигаться столбец "За месяц". Далее происходит сдвиг столбца "За месяц" на каждый месяц в

прошлом с помощью метода **shift()**, и создаются новые столбцы "За месяц 1", "За месяц 2" и т.д., в которых содержатся значения столбца "За месяц", сдвинутые на 1, 2 и т.д. месяца в прошлом. На выходе получается измененный DataFrame с новыми столбцами (Рис. 5). Подготавливаем данные для работы (Рис. 6).

	Дата	За месяц	За месяц 1	За месяц 2	За месяц 3	За месяц 4	За месяц 5
0	2012-01-01	17.8	NaN	NaN	NaN	NaN	NaN
1	2012-02-01	20.1	17.8	NaN	NaN	NaN	NaN
2	2012-03-01	17.1	20.1	17.8	NaN	NaN	NaN
3	2012-04-01	18.2	17.1	20.1	17.8	NaN	NaN
4	2012-05-01	17.2	18.2	17.1	20.1	17.8	NaN

Рисунок 5. Измененный DataFrame с новыми столбцами

Отнормируем данные

```
x_columns = ["За месяц 1", "За месяц 2", "За месяц 3", "За месяц 4", "За месяц 5"]
y_columns = ["За месяц"]
scaler_x = StandardScaler()
scaler_y = StandardScaler()
x_scaled = scaler_x.fit_transform(data[x_columns][start_shift:])
y_scaled = scaler_y.fit_transform(data[y_columns][start_shift:])
```

Подготовим двумерные массивы с временными окнами - для работы рекуррентных нейросетей

```
def getdata(data_x, data_y, lookback):
    X, Y = [], []
    for i in range(len(data) - lookback + 1 - start_shift):
        X.append(data_x[i:i+lookback])
        Y.append(data_y[i+lookback-1])
    return np.array(X), np.array(Y).reshape(-1, 1)
lookback = 10
x, y = getdata(x_scaled, y_scaled, lookback)
x = x.reshape(x.shape[0], x.shape[1], x.shape[2])
```

Рисунок 6. Подготовка данных

2.2. Обучение нейронных сетей

2.2.1. RNN

Создаем простую рекуррентную нейронную сеть с одним слоем SimpleRNN и одним выходным слоем Dense. Аргументы в функции SimpleRNN означают следующее:

- 5: количество нейронов в слое
- stateful=True: сохранять состояние нейронной сети между обучающими пакетами
- batch_input_shape=(batch_size, x.shape[1], x.shape[2]): размерность входных данных. Первый аргумент указывает размер мини-пакета, второй и третий аргументы задают размерность каждого входного образца. x.shape[1] означает количество временных шагов, x.shape[2] - количество признаков на каждом временном шаге.

Затем определяем функцию потерь как среднеквадратичное отклонение (MSE) и оптимизатор Adam.

После чего обучаем RNN модель, используя обучающий набор данных x и y, с 50 эпохами обучения и размером пакета (batch size) равным 1, исключая последние 10 элементов в обучающем наборе (Рис. 7).

```
batch_size=1
model = Sequential()
model.add(SimpleRNN(5, stateful=True, batch_input_shape=(batch_size, x.shape[1], x.shape[2])))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (1, 5)                    55

 dense_3 (Dense)             (1, 1)                    6

=================================================================
Total params: 61
Trainable params: 61
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x[:-10], y[:-10], epochs=50, batch_size=batch_size)
```

```
Epoch 1/50
97/97 [==============================] - 1s 1ms/step - loss: 3.2068
Epoch 2/50
97/97 [==============================] - 0s 974us/step - loss: 1.2667
Epoch 3/50
97/97 [==============================] - 0s 994us/step - loss: 0.5900
Epoch 4/50
97/97 [==============================] - 0s 987us/step - loss: 0.4405
Epoch 5/50
97/97 [==============================] - 0s 992us/step - loss: 0.3598
Epoch 6/50
97/97 [==============================] - 0s 1ms/step - loss: 0.3309
Epoch 7/50
97/97 [==============================] - 0s 1ms/step - loss: 0.3118
Epoch 8/50
97/97 [==============================] - 0s 994us/step - loss: 0.2802
Epoch 9/50
```

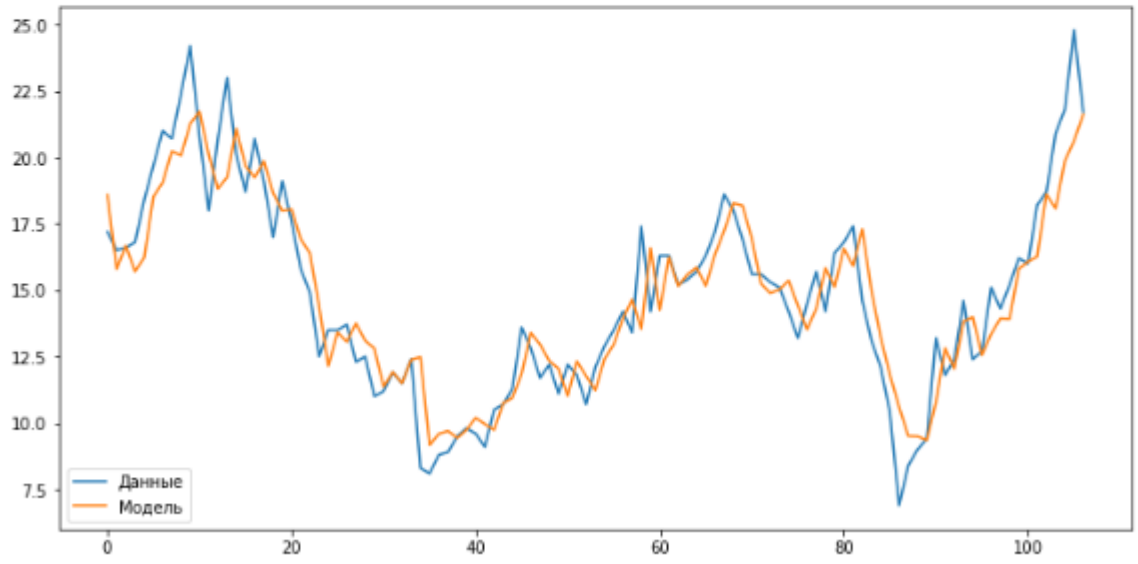
Рисунок 7. Обучение RNN

Визуализируем результат и выводим ошибку модели с помощью функции **smape()** (симметричное среднее абсолютное процентное отклонение) (Рис. 8).

```
y_test = scaler_y.inverse_transform(y[-112:])
forecast = scaler_y.inverse_transform(model.predict(x[-112:], batch_size=batch_size))

plt.figure(figsize=(12, 6))
plt.plot(y_test)
plt.plot(forecast)
plt.legend(labels=["Данные", "Модель"])
plt.show()
model_smape = smape(y_test[-10:], forecast[-10:])
print ("Ошибка модели:", model_smape)
```

```
107/107 [==============================] - 0s 662us/step
```




```
Ошибка модели: 68.75816314467059
```

Рисунок 8. Визуализация результата

2.2.2. LSTM

Определяем нейронную сеть с использованием модели **Sequential** из библиотеки Keras. Сеть содержит два слоя: слой LSTM (долгая краткосрочная память) с 10 скрытыми нейронами и функцией активации по умолчанию (гиперболический тангенс), и слой плотности с одним нейроном и линейной функцией активации.

- .stateful=True: сохранять состояние нейронной сети между обучающими пакетами
- batch_input_shape=(batch_size, x.shape[1], x.shape[2]): размерность входных данных. Первый аргумент указывает размер мини-пакета,

второй и третий аргументы задают размерность каждого входного образца. x.shape[1] означает количество временных шагов, x.shape[2] - количество признаков на каждом временном шаге.

Затем определяем функцию потерь как среднеквадратичное отклонение (MSE) и оптимизатор Adam.

После чего обучаем LSTM модель, используя обучающий набор данных x и y, с 50 эпохами обучения и размером пакета (batch size) равным 1, исключая последние 10 элементов в обучающем наборе (Рис. 9).

```
batch_size=1
model = Sequential()
model.add(LSTM(10, stateful=True, batch_input_shape=(batch_size, x.shape[1], x.shape[2])))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (1, 10)                   640

 dense_5 (Dense)             (1, 1)                    11

=================================================================
Total params: 651
Trainable params: 651
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x[:-10], y[:-10], epochs=50, batch_size=batch_size)
```

```
Epoch 1/50
97/97 [==============================] - 1s 2ms/step - loss: 0.6859
Epoch 2/50
97/97 [==============================] - 0s 2ms/step - loss: 0.4867
Epoch 3/50
97/97 [==============================] - 0s 2ms/step - loss: 0.4085
Epoch 4/50
97/97 [==============================] - 0s 2ms/step - loss: 0.3504
Epoch 5/50
97/97 [==============================] - 0s 2ms/step - loss: 0.3110
```

Рисунок 9. Обучение LSTM

Получаем прогнозные значения (forecast) с использованием модели LSTM на последних 112 точках временного ряда и обратного масштабирования прогноза (inverse_transform) для получения его в исходных единицах измерения. Визуализируем результат и выводим ошибку модели с

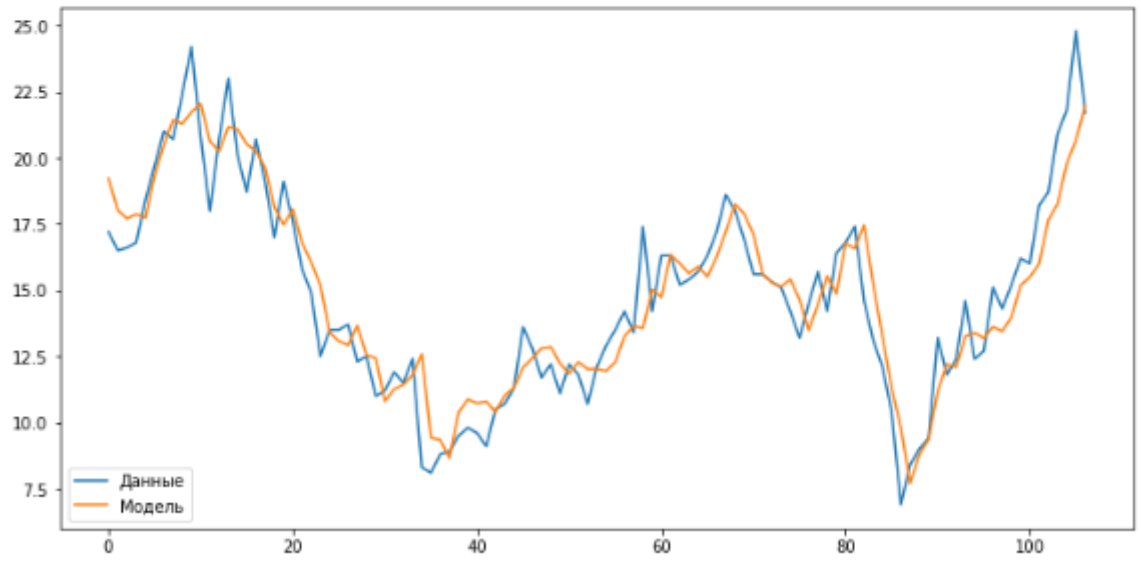
помощью функции **smape()** (симметричное среднее абсолютное процентное отклонение) (Рис. 10).

```
y_test = scaler_y.inverse_transform(y[-112:])
forecast = scaler_y.inverse_transform(model.predict(x[-112:], batch_size=batch_size))
```

```
107/107 [==============================] - 0s 791us/step
```

```
plt.figure(figsize=(12, 6))
plt.plot(y_test)
plt.plot(forecast)
plt.legend(labels=["Данные", "Модель"])
plt.show()

model_smape = smape(y_test[-10:], forecast[-10:])
print ("Ошибка модели:", model_smape)
```




```
Ошибка модели: 85.60304809175045
```

Рисунок 10. Визуализация результата

2.2.3. GRU

Создаем нейронную сеть, использующую слой GRU с 10 скрытыми нейронами. Модель также имеет полносвязный слой с одним выходом и использует функцию активации по умолчанию (линейную функцию).

- .stateful=True: сохранять состояние нейронной сети между обучающими пакетами
- batch_input_shape=(batch_size, x.shape[1], x.shape[2]): размерность входных данных. Первый аргумент указывает размер мини-пакета,

второй и третий аргументы задают размерность каждого входного образца. x.shape[1] означает количество временных шагов, x.shape[2] - количество признаков на каждом временном шаге.

Затем определяем функцию потерь как среднеквадратичное отклонение (MSE) и оптимизатор Adam.

После чего обучаем GRU модель, используя обучающий набор данных x и y, с 50 эпохами обучения и размером пакета (batch size) равным 1, исключая последние 10 элементов в обучающем наборе (Рис. 11).

```
batch_size=1
model = Sequential()
model.add(GRU(10, stateful=True, batch_input_shape=(batch_size, x.shape[1], x.shape[2])))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (1, 10)                   510

 dense_6 (Dense)             (1, 1)                    11

=================================================================
Total params: 521
Trainable params: 521
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x[:-10], y[:-10], epochs=50, batch_size=batch_size)
```

```
Epoch 1/50
97/97 [==============================] - 1s 2ms/step - loss: 0.4869
Epoch 2/50
97/97 [==============================] - 0s 2ms/step - loss: 0.2947
Epoch 3/50
97/97 [==============================] - 0s 2ms/step - loss: 0.2514
Epoch 4/50
97/97 [==============================] - 0s 2ms/step - loss: 0.2234
Epoch 5/50
```

Рисунок 11. Обучение GRU

Получаем прогнозные значения (forecast) с использованием модели GRU на последних 112 точках временного ряда и обратного масштабирования прогноза (inverse_transform) для получения его в исходных единицах измерения. Визуализируем результат и выводим ошибку модели с помощью

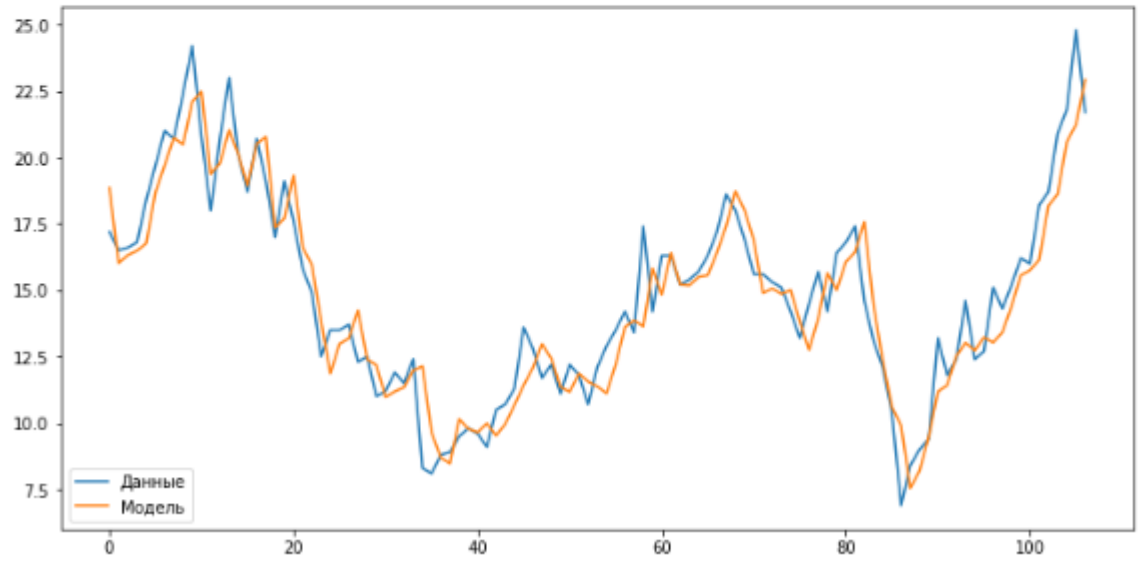
функции **smape()** (симметричное среднее абсолютное процентное отклонение) (Рис. 12).

```
y_test = scaler_y.inverse_transform(y[-112:])
forecast = scaler_y.inverse_transform(model.predict(x[-112:], batch_size=batch_size))
```

```
107/107 [==============================] - 0s 827us/step
```

```
plt.figure(figsize=(12, 6))
plt.plot(y_test)
plt.plot(forecast)
plt.legend(labels=["Данные", "Модель"])
plt.show()

model_smape = smape(y_test[-10:], forecast[-10:])
print ("Ошибка модели:", model_smape)
```




```
Ошибка модели: 70.59210758086184
```

Рисунок 12. Визуализация результата

2.2.4. BiLSTM

Создаем нейронную сеть с двунаправленной LSTM архитектурой. Благодаря двунаправленной LSTM сети, модель может использовать как предыдущие, так и последующие временные данные для прогнозирования значения в текущем временном шаге.

- stateful=True: сохранять состояние нейронной сети между обучающими пакетами
- batch_input_shape=(batch_size, x.shape[1], x.shape[2]): размерность входных данных. Первый аргумент указывает размер мини-пакета,

второй и третий аргументы задают размерность каждого входного образца. x.shape[1] означает количество временных шагов, x.shape[2] - количество признаков на каждом временном шаге.

Затем добавляется один слой Dense с одним выходом, который будет использоваться для прогнозирования одного значения.

Далее определяется функция потерь, используемая для обучения модели - mean_squared_error, и оптимизатор Adam.

- model.build(input_shape=(batch_size, x.shape[1], x.shape[2])) используется для явного построения модели, чтобы она могла быть использована для прогнозирования значений на тестовом наборе данных.
- model.summary() выводит сводку модели, в которой отображаются слои, их типы, размерности выходов, количество параметров, используемых для обучения и общее количество параметров.

После чего обучаем BiLSTM модель, используя обучающий набор данных x и y, с 100 эпохами обучения и размером пакета (batch size) равным 1, исключая последние 10 элементов в обучающем наборе (Рис. 13).

```
batch_size=1
model = Sequential()
model.add(Bidirectional(LSTM(10, stateful=True, batch_input_shape=(batch_size, x.shape[1], x.shape[2]))))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
model.build(input_shape=(batch_size, x.shape[1], x.shape[2]))
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona  (1, 20)                1280
 l)

 dense_7 (Dense)             (1, 1)                    21

=================================================================
Total params: 1,301
Trainable params: 1,301
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x[:-10], y[:-10], epochs=100, batch_size=batch_size)
```

```
Epoch 1/100
97/97 [==============================] - 2s 2ms/step - loss: 0.4836
Epoch 2/100
97/97 [==============================] - 0s 2ms/step - loss: 0.3637
Epoch 3/100
97/97 [==============================] - 0s 2ms/step - loss: 0.3236
```

Рисунок 13. Обучение BiLSTM

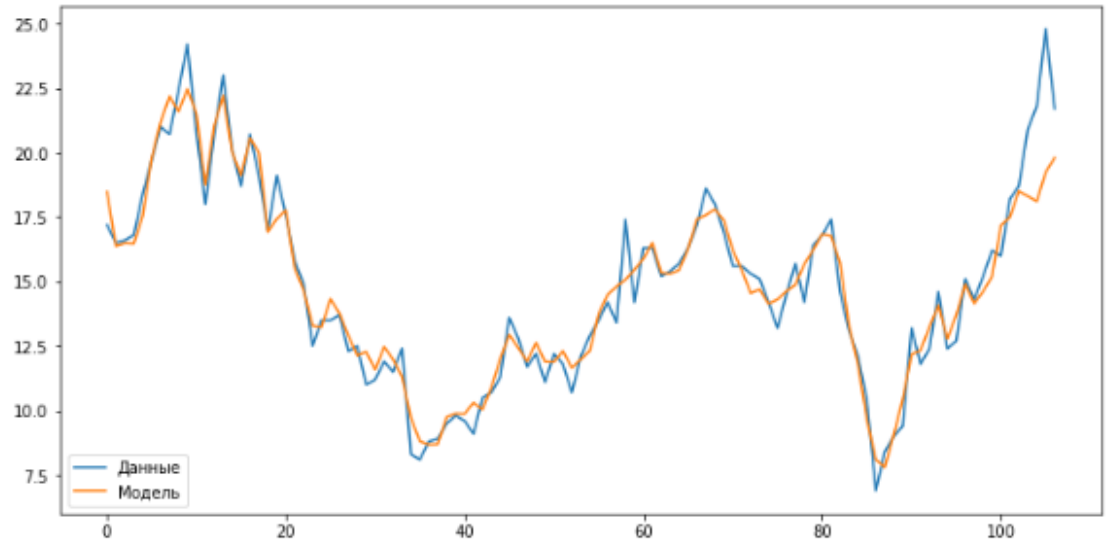
Получаем прогнозные значения (forecast) с использованием модели GRU на последних 112 точках временного ряда и обратного масштабирования прогноза (inverse_transform) для получения его в исходных единицах измерения. Визуализируем результат и выводим ошибку модели с помощью функции **smape()** (симметричное среднее абсолютное процентное отклонение) (Рис. 14).

```
y_test = scaler_y.inverse_transform(y[-112:])
forecast = scaler_y.inverse_transform(model.predict(x[-112:], batch_size=batch_size))
```

```
107/107 [==============================] - 0s 831us/step
```

```
plt.figure(figsize=(12, 6))
plt.plot(y_test)
plt.plot(forecast)
plt.legend(labels=["Данные", "Модель"])
plt.show()

model_smape = smape(y_test[-10:], forecast[-10:])
print ("Ошибка модели:", model_smape)
```




```
Ошибка модели: 89.67864815076207
```

Рисунок 14. Визуализация результата

2.2.5. Вывод

Согласно значению метрики SMAPE, лучшей моделью из представленных является RNN, которая имеет наименьшую ошибку SMAPE (68). Это может быть связано с тем, что RNN более подходит для данной задачи, поскольку сохраняет внутреннее состояние, учитывая предыдущие значения.

2.3. Обучение с использованием AR/MA моделей

2.3.1. ARMA

Подбираем коэффициенты (Рис. 15). Модель ARMA использует два целочисленных параметра: p и q.

- p – порядок авторегрессии (AR). Его можно интерпретировать как выражение «элемент ряда будет близок к X, если предыдущие p элементов были близки к X».
- q – порядок скользящего среднего (MA), который позволяет установить погрешность модели как линейную комбинацию наблюдавшихся ранее значений ошибок.

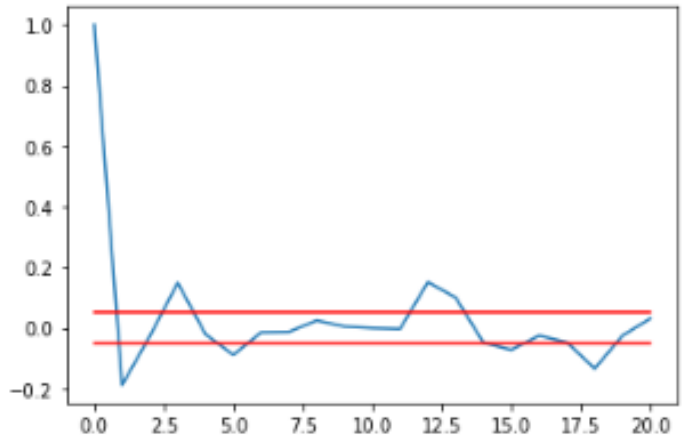
Найдем порядок автокорреляции (разностей исходного ряда)

$$Y_t=\alpha_0+ \alpha_1 Y_{t-1}+ \alpha_2 Y_{t-2}+ \alpha_3 Y_{t-3}$$

```
# ищем наибольший вклад компоненты в корреляцию по первым двум производным
data_ = data["За месяц"]
max_p = 2
pacf_coeff = np.zeros(max_p)
for p in range(1, max_p+1):
    data_ = data_.diff().dropna()
    pacf_vals = pacf(data_, nlags=p, method="ols")
    pacf_coeff[p-1] = np.abs(pacf_vals[-1])
p_best = np.argmax(pacf_coeff) + 1
print("Порядок авторегрессии (AR) равен", p_best)
```

```
Порядок авторегрессии (AR) равен 2
```

```
# ищем порядок скользящего среднего (по вкладу компонентов первой производной)
q_best = 0
result = acf(data["За месяц"].diff().dropna())
plt.plot(result)
plt.plot([0.05]*len(result), color="red")
plt.plot([-0.05]*len(result), color="red")
plt.show()
for i in range(len(result)):
    if abs(result[i]) > 0.05:
        q_best = i + 1
    else:
        break
print ("q - порядок скользящего среднего (MA) равен", q_best)
```




```
q - порядок скользящего среднего (MA) равен 2
```

Рисунок 15. Находим параметры p и q

Оцениваем модель ARMA на данных "За месяц" с параметрами (p_best, 0, q_best). Функция **summary()** отображает сводку статистических показателей и параметров модели (Рис. 16).

```
model = ARIMA(data["За месяц"], order=(p_best, 0, q_best)).fit()
print (model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:              За месяц   No. Observations:                  121
Model:                 ARIMA(2, 0, 2)   Log Likelihood                -224.957
Date:                Tue, 09 May 2023   AIC                            461.913
Time:                        14:47:37   BIC                            478.688
Sample:                             0   HQIC                           468.726
                                - 121
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         15.8642      1.625      9.764      0.000      12.680      19.049
ar.L1          1.4836      0.411      3.610      0.000       0.678       2.289
ar.L2         -0.5275      0.390     -1.351      0.177      -1.293       0.238
ma.L1         -0.7496      0.415     -1.807      0.071      -1.563       0.064
ma.L2          0.2533      0.098      2.575      0.010       0.061       0.446
sigma2         2.3731      0.319      7.440      0.000       1.748       2.998
===================================================================================
Ljung-Box (L1) (Q):                   0.10   Jarque-Bera (JB):                 0.21
Prob(Q):                              0.75   Prob(JB):                         0.90
Heteroskedasticity (H):               0.90   Skew:                            -0.09
Prob(H) (two-sided):                  0.74   Kurtosis:                         2.91
===================================================================================
```

Рисунок 16. Оцениваем модель

Переберем все коэффициенты "по сетке", выберем лучший вариант AIC (Рис. 17).

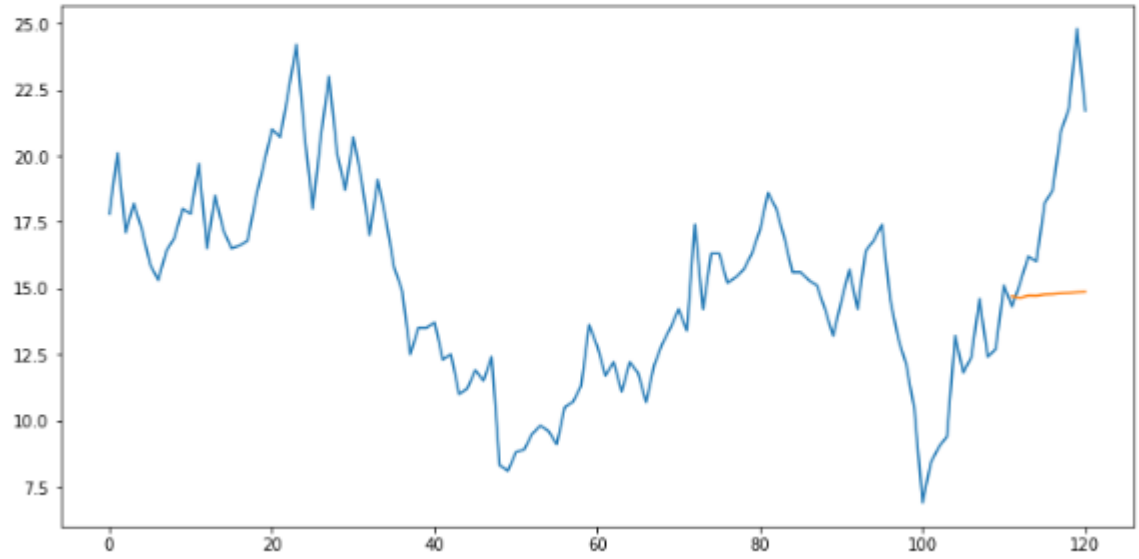
```
aic_best = model.aic
coeff_best = (p_best, 0, q_best)
for p in range(1,5):
    for q in range(1,5):
        model_ = ARIMA(data["За месяц"], order=(p, 0, q)).fit()
        aic = model_.aic
        if aic < aic_best:
            coeff_best = (p, 0, q)
            aic_best = aic
print ("Наилучшие коффициенты:", coeff_best)
```

```
Наилучшие коффициенты: (3, 0, 2)
```

Рисунок 17.Выбираем лучший вариант AIC

Визуализируем результат и выводим ошибку модели с помощью функции **smape()** (симметричное среднее абсолютное процентное отклонение) (Рис. 18).

```
model = ARIMA(data["За месяц"][:-10], order=(p_best, 0, q_best)).fit()
forecast = model.get_forecast(steps=10).predicted_mean
plt.figure(figsize=(12, 6))
plt.plot(data["За месяц"])
plt.plot(forecast)
plt.show()
def smape(A, F):
    return 100*np.sum(2 * np.abs(F - A) / (np.abs(A) + np.abs(F)))
print ("Ошибка модели:", smape(data["За месяц"][-10:], forecast))
```




```
Ошибка модели: 228.85491782543758
```

Рисунок 18.Визуализация результата

2.3.2. ARIMA

Подбираем коэффициенты. Модель ARIMA использует три целочисленных параметра: p, d и q.

- p – порядок авторегрессии (AR). Его можно интерпретировать как выражение «элемент ряда будет близок к X, если предыдущие p элементов были близки к X».
- d – порядок интегрирования (I) разностей исходного временного ряда. Можно понимать как «элемент будет близок по значению к предыдущим d элементам, если их разность минимальна».
- q – порядок скользящего среднего (MA), который позволяет установить погрешность модели как линейную комбинацию наблюдавшихся ранее значений ошибок.

Найдем порядок автокорреляции (разностей исходного ряда)

$$Y_t=\alpha_0+ \alpha_1 Y_{t-1}+ \alpha_2 Y_{t-2}+ \alpha_3 Y_{t-3}$$

Проведем тест Дики-Фуллера на стационарность данных производных временного ряда для нахождения порядка интегрирования (Рис. 19).

```
data_ = data["За месяц"]
d_best = 0
for d in range(1, 5):
    data_ = data_.diff().dropna()
    result = adfuller(data_)
    print ("p-значение для " + str(d) + "-производной ряда", result[1])
    if d_best == 0 and result[1] < 0.05:
        d_best = d
print ("d - порядок интегрирования (I) равен", d_best)
print ("Порядок авторегрессии (AR) равен", p_best)
print ("q - порядок скользящего среднего (MA) равен", q_best)
```

```
p-значение для 1-производной ряда 5.451861499085897e-07
p-значение для 2-производной ряда 5.524627713108054e-09
p-значение для 3-производной ряда 2.3288874449398707e-11
p-значение для 4-производной ряда 6.720765595947932e-10
d - порядок интегрирования (I) равен 1
Порядок авторегрессии (AR) равен 2
q - порядок скользящего среднего (MA) равен 2
```

Рисунок 19. Тест Дики-Фуллера

Оцениваем модель ARIMA на данных "За месяц" с параметрами (p_best, d_best, q_best). Функция **summary()** отображает сводку статистических показателей и параметров модели (Рис. 20).

```
model = ARIMA(data["За месяц"], order=(p_best, d_best, q_best)).fit()
print (model.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:              За месяц   No. Observations:                  121
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -219.638
Date:                Tue, 09 May 2023   AIC                            449.276
Time:                        14:47:40   BIC                            463.214
Sample:                             0   HQIC                           454.936
                                  - 121
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4384      0.080      5.466      0.000       0.281       0.596
ar.L2         -0.7539      0.064    -11.863      0.000      -0.878      -0.629
ma.L1         -0.6301      0.738     -0.854      0.393      -2.076       0.815
ma.L2          0.9992      2.333      0.428      0.668      -3.572       5.571
sigma2         2.1870      5.029      0.435      0.664      -7.670      12.044
===================================================================================
Ljung-Box (L1) (Q):                   0.25   Jarque-Bera (JB):                 0.47
Prob(Q):                              0.62   Prob(JB):                         0.79
Heteroskedasticity (H):               0.80   Skew:                            -0.09
Prob(H) (two-sided):                  0.48   Kurtosis:                         2.75
===================================================================================
```

Рисунок 20. Оцениваем модель

Переберем все коэффициенты "по сетке", выберем лучший вариант AIC (Рис. 21).

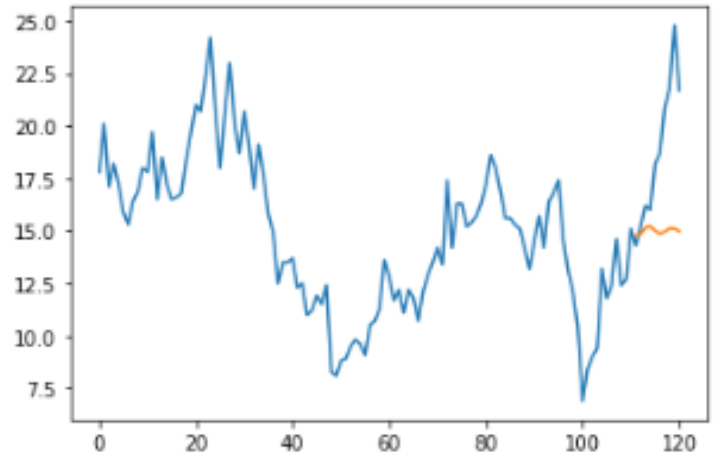
```
aic_best = model.aic
coeff_best = (p_best, d_best, q_best)
for p in range(1,5):
    for d in range(1,5):
        for q in range(1,5):
            model_ = ARIMA(data["За месяц"], order=(p, d, q)).fit()
            aic = model_.aic
            if aic < aic_best:
                coeff_best = (p, d, q)
                aic_best = aic
print ("Наилучшие коффициенты:", coeff_best)
```

```
Наилучшие коффициенты: (2, 1, 2)
```

Рисунок 21.Выбираем лучший вариант AIC

Визуализируем результат и выводим ошибку модели с помощью функции **smape()** (симметричное среднее абсолютное процентное отклонение) (Рис. 22).

```
model = ARIMA(data["За месяц"][:-10], order=coeff_best).fit()
forecast = model.get_forecast(steps=10).predicted_mean
plt.plot(data["За месяц"])
plt.plot(forecast)
plt.show()
print ("Ошибка модели:", smape(data["За месяц"][-10:], forecast))
```




```
Ошибка модели: 213.62896564611145
```

Рисунок 22. Визуализация результата

2.3.3. SARIMA

Подбираем коэффициенты. Модель SARIMA использует шесть целочисленных параметра: p, d, q, P, D, Q.

- p – порядок авторегрессии (AR). Его можно интерпретировать как выражение «элемент ряда будет близок к X, если предыдущие p элементов были близки к X».
- d – порядок интегрирования (I) разностей исходного временного ряда. Можно понимать как «элемент будет близок по значению к предыдущим d элементам, если их разность минимальна».
- q – порядок скользящего среднего (MA), который позволяет установить погрешность модели как линейную комбинацию наблюдавшихся ранее значений ошибок.
- P - порядок сезонной авторегрессии
- D - порядок сезонного интегрирования

- Q - порядок сезонного скользящего среднего
- T (период сезона) - 6

Оцениваем модель ARIMA на данных "За месяц". SARIMAX имеет порядок (2, 1, 2) для компонентов ARIMA и порядок (1, 1, 1, 6) для компонентов сезонности. Компоненты ARIMA отвечают за моделирование зависимости от предыдущих значений временного ряда, в то время как компоненты сезонности отвечают за моделирование повторяющихся паттернов в данных. Функция **summary()** отображает сводку статистических показателей и параметров модели (Рис. 23).

```
model = SARIMAX(data["За месяц"], order=(2, 1, 2), seasonal_order=(1,1,1,6)).fit()
print (model.summary())
```

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                          За месяц   No. Observations:                  121
Model:             SARIMAX(2, 1, 2)x(1, 1, [1], 6)   Log Likelihood                -216.136
Date:                             Tue, 09 May 2023   AIC                            446.271
Time:                                     14:47:54   BIC                            465.424
Sample:                                          0   HQIC                           454.044
                                               - 121
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4076      0.090      4.533      0.000       0.231       0.584
ar.L2         -0.7485      0.064    -11.628      0.000      -0.875      -0.622
ma.L1         -0.6229      0.042    -14.843      0.000      -0.705      -0.541
ma.L2          0.9700      0.081     11.989      0.000       0.811       1.129
ar.S.L6        0.0407      0.125      0.325      0.745      -0.205       0.286
ma.S.L6       -0.9915      1.288     -0.770      0.441      -3.516       1.533
sigma2         2.1898      2.706      0.809      0.418      -3.113       7.493
===================================================================================
Ljung-Box (L1) (Q):                   0.12   Jarque-Bera (JB):                 0.80
Prob(Q):                              0.73   Prob(JB):                         0.67
Heteroskedasticity (H):               0.80   Skew:                            -0.09
Prob(H) (two-sided):                  0.48   Kurtosis:                         3.37
===================================================================================
```

Рисунок 23. Оцениваем модель

Переберем все коэффициенты "по сетке", выберем лучший вариант AIC (Рис. 24).

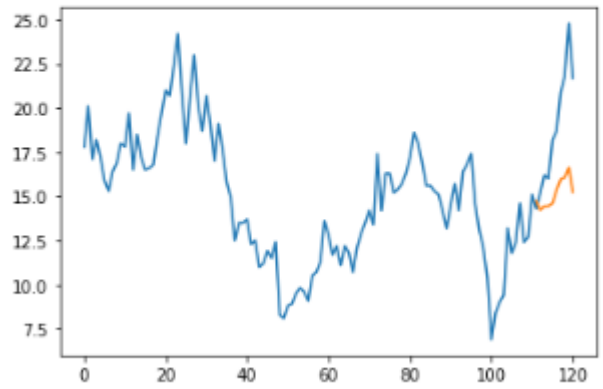
```
aic_best = model.aic
seasonal_coeff_best = (1, 1, 1, 6)
for P in range(1,3):
    for D in range(1,3):
        for Q in range(1,3):
            model_ = SARIMAX(data["За месяц"], order=(2, 1, 2), seasonal_order=(P, D, Q, 6)).fit()
            aic = model_.aic
            if aic < aic_best:
                seasonal_coeff_best = (P, D, Q, 6)
                aic_best = aic
print ("Наилучшие коффициенты:", seasonal_coeff_best)
```

```
Наилучшие коффициенты: (2, 1, 2, 6)
```

Рисунок 24. Выбираем лучший вариант AIC

Визуализируем результат и выводим ошибку модели с помощью функции **smape()** (симметричное среднее абсолютное процентное отклонение) (Рис. 25).

```
model = SARIMAX(data["За месяц"][:-10], order=(2, 1, 2), seasonal_order=seasonal_coeff_best).fit()
forecast = model.get_forecast(steps=10).predicted_mean
plt.plot(data["За месяц"])
plt.plot(forecast)
plt.show()
print ("Ошибка модели:", smape(data["За месяц"][-10:], forecast))
```




```
Ошибка модели: 204.22010260157907
```

Рисунок 25. Визуализация результата

2.3.4. Вывод

Исходя из значений метрики SMAPE, SARIMA показала наилучший результат с ошибкой прогнозирования на уровне 204, ARIMA также демонстрирует хороший результат с ошибкой на уровне 213. ARMA, в свою очередь, показала наихудший результат, ошибку на уровне 228.

Вывод

Исходя из результатов SMAPE, нейронные сети (RNN, LSTM, GRU, BiLSTM) показали более высокую точность прогнозирования, чем модели AR/MA (ARMA, ARIMA, SARIMA).

В частности, лучшей моделью оказалась нейронная сеть RNN с ошибкой SMAPE 68, что значительно лучше, чем у лучшей модели AR/MA (SARIMA) с ошибкой SMAPE 204.

Таким образом, для данной временной рядовой задачи, нейронные сети оказались более эффективным инструментом для прогнозирования, чем классические AR/MA модели.

Список литературы

1. Что такое нейронная сеть? // aws.amazon URL: https://aws.amazon.com/ru/what-is/neural-network/ (дата обращения: 20.04.2023).

2. Модели ARMA (p, q) // univer-nn URL: https://univer-nn.ru/modeli-arma-p-q/ (дата обращения: 20.04.2023).

3. Как рассчитать SMAPE в Python // codecamp URL: https://www.codecamp.ru/blog/smape-python/ (дата обращения: 20.04.2023).

4. Салихов Марсель Линейные модели финансовых серий (AR) “Количественные финансы” // quantviews.github URL: https://quantviews.github.io/Financial_Markets4-old/lectures/lecture-5.html# (дата обращения: 20.04.2023).

5. Салихов Марсель Линейные модели финансовых серий (MA, ARMA и ARIMA) “Количественные финансы” // quantviews.github URL: https://quantviews.github.io/Financial_Markets4-old/lectures/lecture-5.html# (дата обращения: 20.04.2023).

6. Рекуррентные нейронные сети (RNN) с Keras // habr URL: https://habr.com/ru/articles/487808/ (дата обращения: 25.04.2023).

7. LSTM – сети долгой краткосрочной памяти // habr URL: https://habr.com/ru/companies/wunderfund/articles/331310/ (дата обращения: 25.04.2023).

8. RNN, LSTM, GRU и другие рекуррентные нейронные сети. // vbystricky URL: http://vbystricky.ru/2021/05/rnn_lstm_gru_etc.html (дата обращения: 25.04.2023).

9. bilstmLayer - Документация // docs.exponenta URL: https://docs.exponenta.ru/deeplearning/ref/nnet.cnn.layer.bilstmlayer.html (дата обращения: 26.04.2023).