Дипломная работа на тему "Анализ и прогнозирование временного ряда на основе данных о продажах акций компании "Coca-cola" с 19 января 1962 года по 19 декабря 2021 года Введение. Постановка целей и задач.

Часть1.

Импорт библиотек, ознакомление с данными. Импорт Датафрейма, библиотек, написание функций.

```
## Загружаем pmdarima
!pip install pmdarima
```

```
Requirement already satisfied: pmdarima in /usr/local/lib/python3.7/dist-packages (1
Requirement already satisfied: numpy>=1.19.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: statsmodels!=0.12.0,>=0.11 in /usr/local/lib/python3.7
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3
Requirement already satisfied: Cython!=0.29.18,>=0.29 in /usr/local/lib/python3.7/dis
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/d
```

```
## Загружаем fbprophet
!pip install fbprophet
```

```
Requirement already satisfied: fbprophet in /usr/local/lib/python3.7/dist-packages (0
Requirement already satisfied: Cython>=0.22 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: cmdstanpy==0.9.5 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pystan>=2.14 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: LunarCalendar>=0.0.9 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: convertdate>=2.1.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: holidays>=0.10.2 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: setuptools-git>=1.2 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: hijri-converter in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from ho
Requirement already satisfied: korean-lunar-calendar in /usr/local/lib/python3.7/dist
Requirement already satisfied: ephem>=3.7.5.3 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pytz in /usr/local/lib/python3.7/dist-packages (from l
```

```
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-pac
```

```
## Импортируем библиотеки и функции
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline


## Импорт моделей
from statsmodels.tsa.statespace.sarimax import SARIMAX      # для модели SARIMAX
from statsmodels.tsa.seasonal import seasonal_decompose      # для ETS графиков
from fbprophet import Prophet                                # для модели Профет
from statsmodels.tsa.holtwinters import ExponentialSmoothing # для модели Экспоненцальное


##from pmdarima import auto_arima                              # для поиска ARIMA моделей


## Метрики
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from statsmodels.tools.eval_measures import rmse


## Здесь импортируем данные с компьютера для работы в колаб
from google.colab import files
uploaded = files.upload()
```

Выбрать файлы Файл не выбран Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving COCO COLA.csv to COCO COLA.csv

```
df = pd.read_csv('COCO COLA.csv', index_col='Date',parse_dates=True)
##data ={'Column 1'   :[1., 2., 3., 4., 5., 6.],
##       'Index Title':["Open", "High", "Low", "Close", "Adj Close", "Volume"]}
##df = pd.DataFrame (data)
## Проверим загруженные данные
##df.head


df.head
```

```
<bound method NDFrame.head of                 Open       High        Low      Close
Date
1962-01-02   0.263021   0.270182   0.263021   0.263021   0.051133    806400
1962-01-03   0.259115   0.259115   0.253255   0.257161   0.049994   1574400
1962-01-04   0.257813   0.261068   0.257813   0.259115   0.050374    844800
1962-01-05   0.259115   0.262370   0.252604   0.253255   0.049234   1420800
1962-01-08   0.251302   0.251302   0.245768   0.250651   0.048728   2035200
...               ...        ...        ...        ...        ...       ...
```

```
2021-12-13  56.980000  57.930000  56.959999  57.759998  57.759998  31362800
2021-12-14  57.400002  58.169998  57.400002  57.799999  57.799999  24806600
2021-12-15  57.930000  58.250000  57.650002  58.060001  58.060001  24923800
2021-12-16  57.980000  58.880001  57.900002  58.650002  58.650002  24696900
2021-12-17  58.490002  58.919998  57.700001  57.730000  57.730000  51874400

[15096 rows x 6 columns]>
```

```
## Количество строк в нашем датасете
print(len(df))
```

```
15096
```

```
## Проверим общую информацию по датасету
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 15096 entries, 1962-01-02 to 2021-12-17
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       15096 non-null  float64
 1   High       15096 non-null  float64
 2   Low        15096 non-null  float64
 3   Close      15096 non-null  float64
 4   Adj Close  15096 non-null  float64
 5   Volume     15096 non-null  int64  
dtypes: float64(5), int64(1)
memory usage: 825.6 KB
```

```
## Проверим датафрейм на наличие пропусков
print(df.isna().any(axis=None))
```

```
False
```

```
## Проверим, везде ли отсутствуют пропуски
df_check = df.isna()

for i in df_check.columns:
    print(f'Для признака {i} пропуски: ', df_check[i].unique())
```

```
Для признака Open пропуски:  [False]
Для признака High пропуски:  [False]
Для признака Low пропуски:  [False]
Для признака Close пропуски:  [False]
Для признака Adj Close пропуски:  [False]
Для признака Volume пропуски:  [False]
```

```
## Можем заменить на 0, чтобы не прерывать временной ряд
df_new = df.fillna(0)
```

```
## Проверим тип данных
df_new.dtypes
```

```
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

Вывод по части 1: Были загружены необходимые библиотеки, функции, модели, метрики. Был загружен датасет. Датасет был рассмотрен на предмет того, что с ним можно работать: что отсутствуют пропуски, проверен формат данных и дата находится в порядке возрастания.

Часть 2.

Знакомство с данными.

Этот блок посвящен первоначальному знакомству с данными

Наша задача - посмотреть на данные методами .info(), а также изучить их визуально.

Возможные метрики для прогнозирования:

1) Open - это цена, по которой финансовая ценная бумага открывается на рынке, когда начинается торговля

2) High - максимум-это самая высокая цена, по которой акции торгуются в течение периода.

3) Low - минимальная цена акции за период.

4) Adj Close - скорректированная цена закрытия изменяет цену закрытия акции, чтобы отразить стоимость этой акции

5) Close - цена закрытия обычно относится к последней цене, по которой акции торгуются во время обычной торговой сессии

```
## Смотрим общую информацию о датасете
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 15096 entries, 1962-01-02 to 2021-12-17
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       15096 non-null  float64
 1   High       15096 non-null  float64
 2   Low        15096 non-null  float64
 3   Close      15096 non-null  float64
 4   Adj Close  15096 non-null  float64
 5   Volume     15096 non-null  int64
```

```
dtypes: float64(5), int64(1)
memory usage: 825.6 KB
```

```
## Смотрим список аналитической информации и выбираем себе необходимую для изучения
df_new.head
```

```
<bound method NDFrame.head of                  Open       High        Low      Close
Date
1962-01-02   0.263021   0.270182   0.263021   0.263021   0.051133    806400
1962-01-03   0.259115   0.259115   0.253255   0.257161   0.049994   1574400
1962-01-04   0.257813   0.261068   0.257813   0.259115   0.050374    844800
1962-01-05   0.259115   0.262370   0.252604   0.253255   0.049234   1420800
1962-01-08   0.251302   0.251302   0.245768   0.250651   0.048728   2035200
...               ...        ...        ...        ...        ...       ...
2021-12-13  56.980000  57.930000  56.959999  57.759998  57.759998  31362800
2021-12-14  57.400002  58.169998  57.400002  57.799999  57.799999  24806600
2021-12-15  57.930000  58.250000  57.650002  58.060001  58.060001  24923800
2021-12-16  57.980000  58.880001  57.900002  58.650002  58.650002  24696900
2021-12-17  58.490002  58.919998  57.700001  57.730000  57.730000  51874400

[15096 rows x 6 columns]>
```

Вывод по части 2: Выбираем необходимые для анализа столбцы: Date, Volume

Часть 3.

Предобработка данных.Фильтрация данных.

Фильтрую по условию, выбирая признаки по порядку: дата, объем акций.

```
df_new = df[['Volume']]
```

```
## Проверяем
df_new.head()
```

	Volume
Date	
1962-01-02	806400
1962-01-03	1574400
1962-01-04	844800
1962-01-05	1420800
1962-01-08	2035200

Необходимо для Volume и Date поменять тип данных для корректного считывания и

```
df_new['Volume'] = df_new['Volume'].astype('string')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarnir
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  """Entry point for launching an IPython kernel.
```

```
## Проверим тип данных
df_new.dtypes
```

```
Volume    string
dtype: object
```

```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 15096 entries, 1962-01-02 to 2021-12-17
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Volume  15096 non-null  string
dtypes: string(1)
memory usage: 235.9 KB
```

Часть 4

EDA (exploratory data analysis) или разведочный анализ данных.

Цель данного блока: познакомиться "поближе" с данными, которыми мы располагаем.

Задачи данного блока:

Вывести статистику по нужным столбцам;

Построить графическое отображение столбцов.

```
## Входная выборка для анализа
df_new.head()
```

	Volume
Date	

```
## Проведем расчет основных статистических метрик
df_new.describe()
```

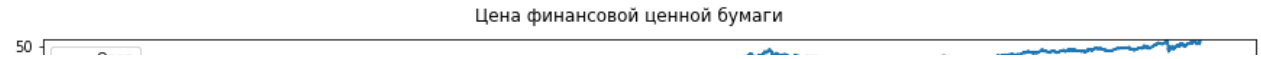
	Volume
count	15096
unique	10188
top	537600
freq	65

```
## Замена значений N/A значением 0
df_new = df.fillna(0)
```

```
## Проведем расчет основных статистических метрик
df_new.describe()
```

	Open	High	Low	Close	Adj Close	V
count	15096.000000	15096.000000	15096.000000	15096.000000	15096.000000	1.50960
mean	16.056202	16.188428	15.921876	16.060734	11.367487	9.03290
std	16.939301	17.064336	16.811753	16.941712	14.253637	7.93574
min	0.192708	0.193359	0.182292	0.192057	0.037855	7.68000
25%	0.860677	0.869792	0.854167	0.860352	0.242312	2.81280
50%	9.328125	9.398438	9.218750	9.351562	4.642848	7.58520
75%	28.875000	29.167500	28.563125	28.901562	17.251080	1.29127
max	59.810001	60.130001	59.619999	60.130001	58.650002	1.24169

```
## Построим общий график для метрик
df_new[['Open','High','Low','Close','Adj Close']].plot(subplots=True, figsize=(15, 3), tit
```

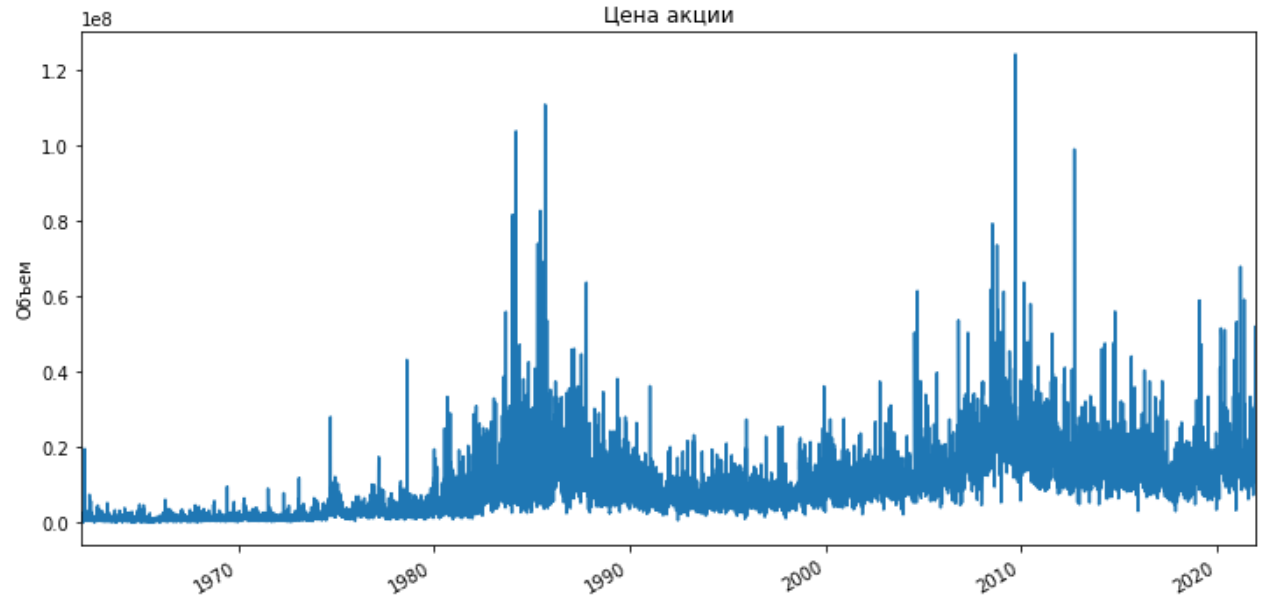


Промежуточный вывод:

Средняя цена акции практически одинакова, заметное повышение цены наблюдается по данным из графика 'Open', т.е. на время открытия биржи, значительное снижение цены наблюдается по данным из графика 'Adj Close'.

```
## Строим временной ряд по колонке 'Объем' из загруженного датасета

title = 'Цена акции' # название графика
ylabel = 'Объем' # название оси Y
xlabel = '' # по сравнению с ожидаемым результатом не указываем название оси X

df1 = df_new.resample('W').mean()
ax = df_new['Volume'].plot(figsize=(12,6),title=title) # добавлем заголовок
ax.autoscale(axis='x', tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel)
```

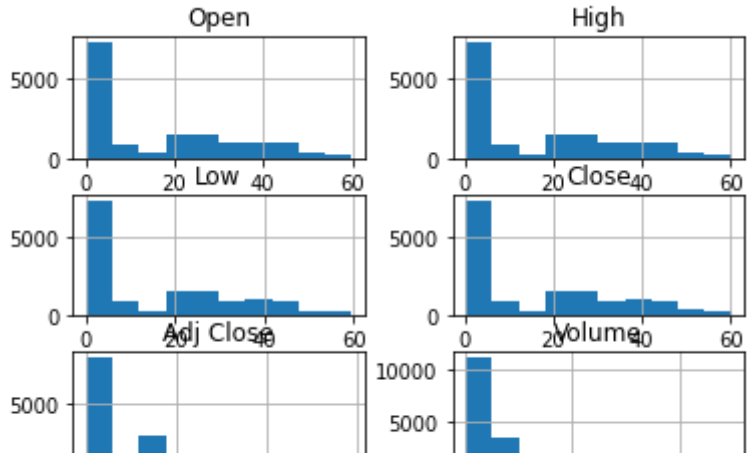
```
[Text(0, 0.5, 'Объем'), Text(0.5, 0, '')]
```



Промежуточный вывод: Видим уровень корреляции, между ценой продажи акции в течении одного дня. Также выявлен положительный тренд потребления электроэнергии. Строим гипотезу: будет ли в будущем расти цена акции с учетом появления новых продукции Coca-cola и работы на дому с учетом сезонности.

```
## Построим гистограмму для определения распределения данных
df_new.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f5bc3d13850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5bc10953d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5bc32fef10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5bc0fe8710>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5bc0fe2ed0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5bbf18d650>]],
      dtype=object)
```



Промежуточный вывод:

Анализ представленного распределения позволяет сделать вывод о пиковых временных отрезках при продаже акций. Это видно на общем графике: 4 пиковых точки продажи. В остальное время продажи осуществляются с более сглаженными всплесками роста.

```
## Построим матрицу корреляции признаков
df_new.corr()
```

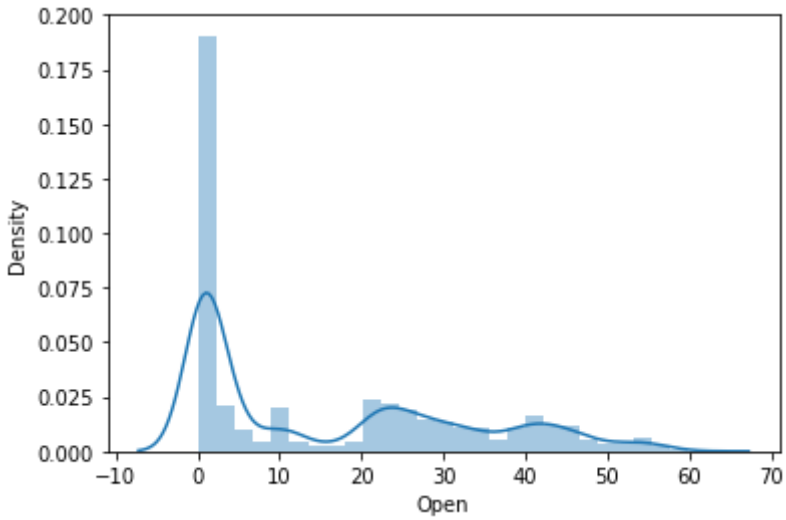
	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999938	0.999929	0.999883	0.971996	0.472328
High	0.999938	1.000000	0.999899	0.999942	0.971578	0.473719
Low	0.999929	0.999899	1.000000	0.999939	0.972530	0.470407
Close	0.999883	0.999942	0.999939	1.000000	0.972004	0.471971
Adj Close	0.971996	0.971578	0.972530	0.972004	1.000000	0.452271
Volume	0.472328	0.473719	0.470407	0.471971	0.452271	1.000000

```
## Построим график зависимости Open от Volume
sns.lineplot(data = df_new, x = 'Open', y = 'Volume')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bc11cf5d0>
```

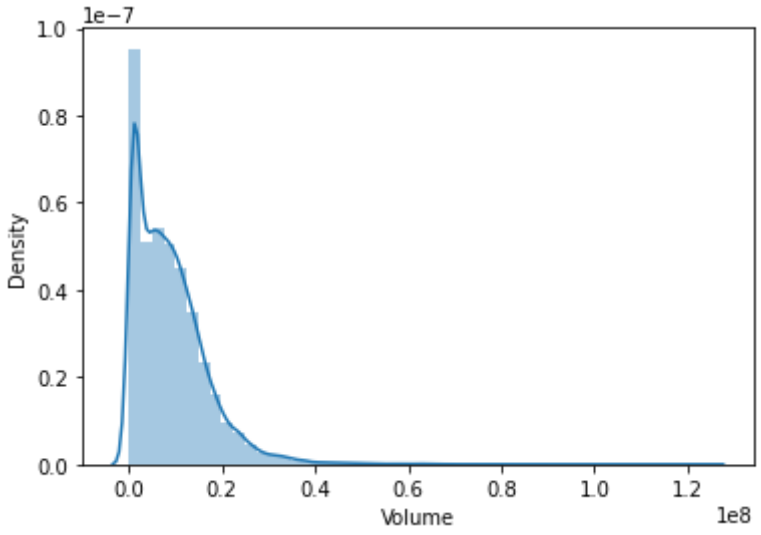

```
## График плотности распределения Open
sns.distplot(df_new.Open)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarnin
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bbeb5ac10>
```




```
# график плотности распределения Volume
sns.distplot(df_new.Volume)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarnin
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bbeb64bd0>
```



Промежуточный вывод:

Видим, что величины коэффициентов корреляции у цены акции на время открытия биржи и объемом продажи равны 0, что свидетельствует о том, что переменные не связаны между собой.

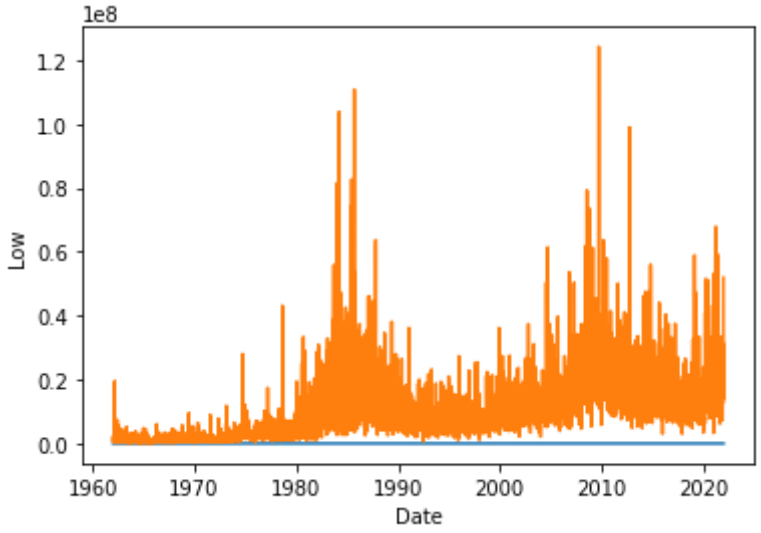
Также на графиках KDE видно, что графики распределения отличаются, благодаря этому наблюдению можно сформулировать гипотезу исследования:

Если цена акции снижается, то общий объем продаж остается прежним.

Для этого построим прогнозную модель расчёта.

```
## При снижении цены акции при открытии биржи объем продаж не увеличивается и остается пре
sns.lineplot(data = df_new, x = 'Date', y = 'Low')
sns.lineplot(data = df_new, x = 'Date', y = 'Volume')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bbeb63690>
```



Часть 5

Построение моделей, анализ результатов.

```
## Выделяем выборки, где тестовая размером 1 месяц (20 рабочих дней, время работы биржи в
train = df.iloc[:len(df)-20]
test = df.iloc[len(df)-20:]
```

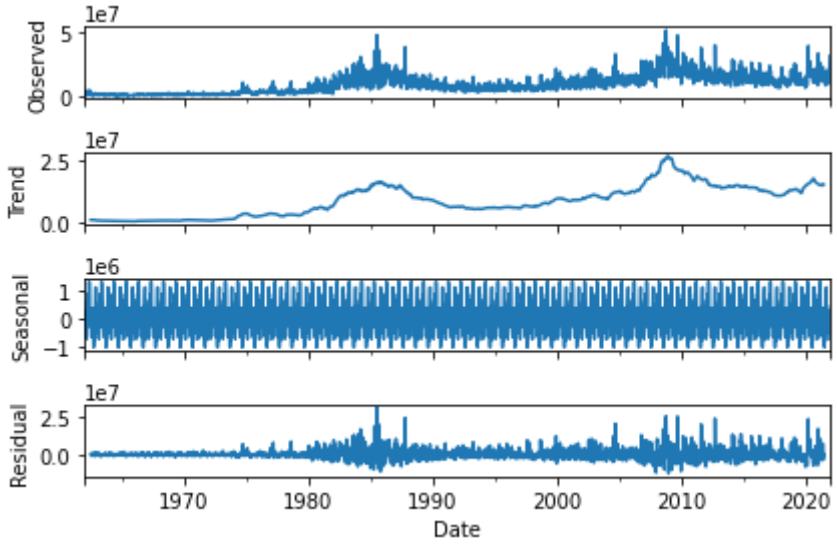
```
## Посмотрим на обучающую выборку
train.head()
```

Date	Open	High	Low	Close	Adj Close	Volume

```
## Посмотрим на тестовую выборку
test.head()
```

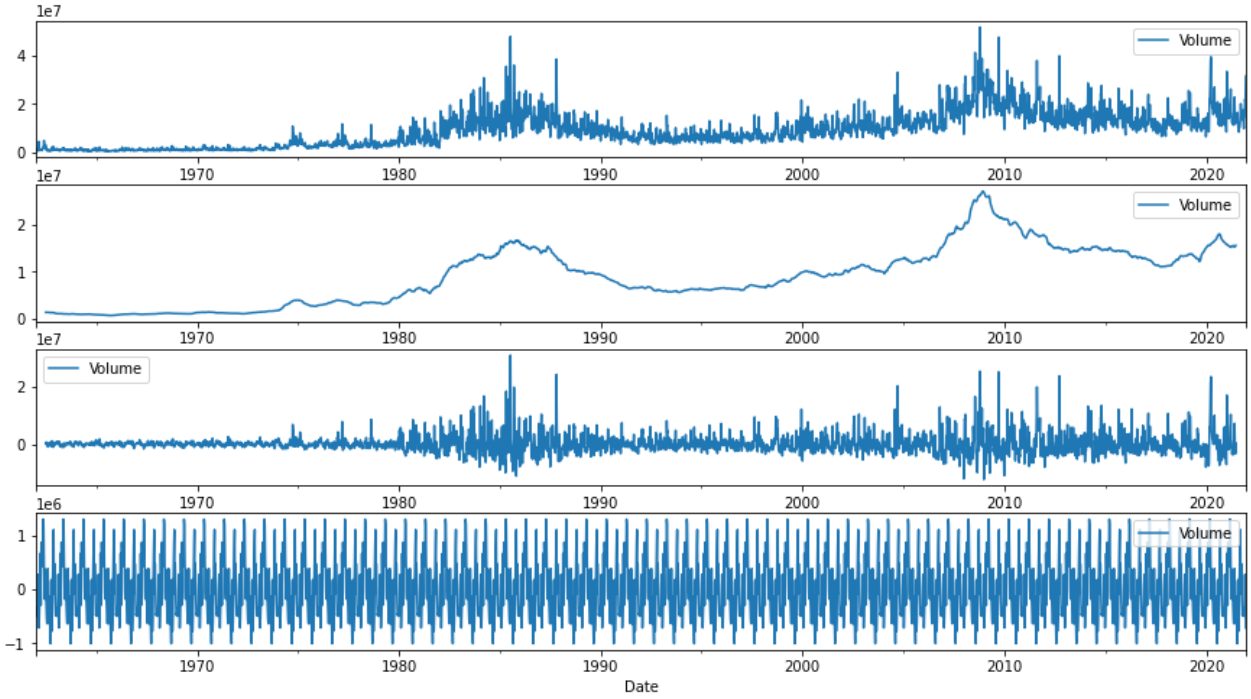
Date	Open	High	Low	Close	Adj Close	Volume
2021-11-19	55.439999	55.490002	54.900002	55.130001	54.705769	15813700
2021-11-22	55.099998	56.020000	55.080002	55.470001	55.043152	16905600
2021-11-23	55.660000	56.110001	55.500000	55.880001	55.449997	13835900
2021-11-24	55.709999	55.840000	55.029999	55.430000	55.003460	12598900
2021-11-26	54.590000	54.750000	53.580002	53.730000	53.316540	14754300

```
## Строим декомпозицию временного ряда (ETS декомпозиция)
r = seasonal_decompose(df1['Volume'], model='additive')
r.plot();
```




```
## seasonal_decompose в увеличенном виде
fig, (ax1,ax2,ax3,ax4) = plt.subplots(4,1, figsize=(15,8))
r.observed.plot(ax=ax1, legend=True)
r.trend.plot(ax=ax2, legend=True)
r.resid.plot(ax=ax3, legend=True)
r.seasonal.plot(ax=ax4, legend=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5bbe90a810>
```



Промежуточный вывод:

На последнем графике наблюдаем цикличные сезонные колебания на протяжении всего периода анализа объема продаж.

Построение моделей. Задачи:

описать модель

подобрать оптимальные параметры

создать модель

обучить модель

сделать прогноз на период тестовой выборки

сравнить прогноз с тестовой выборкой (построить график)

оценить качество прогноза

сделать прогноз на 10-15 дней на будущее

сделать выводы о работе данного метода прогнозирования

2й метод прогнозирования - PROPHET

```
# подготовим данные для модели
train_prophet = train[['Volume']]
test_prophet = test[['Volume']]


#посмотрим на обущающую выборку
```

```
train_prophet.head()
```

Date	Volume
1962-01-02	806400
1962-01-03	1574400
1962-01-04	844800
1962-01-05	1420800
1962-01-08	2035200

```
#посмотрим на тестовую выборку
test_prophet.head()
```

Date	Volume
2021-11-19	15813700
2021-11-22	16905600
2021-11-23	13835900
2021-11-24	12598900
2021-11-26	14754300

```
train_prophet = train.reset_index() #Индекс сбросим, чтобы работать только с колонками.
test_prophet = test.reset_index() #Индекс сбросим, чтобы работать только с колонками.
```

```
train_prophet.head() #посмотрим преобразование после сброса индексов
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	1962-01-02	0.263021	0.270182	0.263021	0.263021	0.051133	806400
1	1962-01-03	0.259115	0.259115	0.253255	0.257161	0.049994	1574400
2	1962-01-04	0.257813	0.261068	0.257813	0.259115	0.050374	844800
3	1962-01-05	0.259115	0.262370	0.252604	0.253255	0.049234	1420800
4	1962-01-08	0.251302	0.251302	0.245768	0.250651	0.048728	2035200

```
test_prophet.head() #посмотрим преобразование после сброса индексов
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2021-11-19	55.439999	55.490002	54.900002	55.130001	54.705769	15813700
1	2021-11-22	55.099998	56.020000	55.080002	55.470001	55.043152	16905600
2	2021-11-23	55.660000	56.110001	55.500000	55.880001	55.449997	13835900

```
train_prophet_0_new = train_prophet[['Date','Volume']]
test_prophet_0_new = test_prophet[['Date','Volume']]
```

```
train_prophet_0_new.head()
```

	Date	Volume
0	1962-01-02	806400
1	1962-01-03	1574400
2	1962-01-04	844800
3	1962-01-05	1420800
4	1962-01-08	2035200

```
test_prophet_0_new.head()
```

	Date	Volume
0	2021-11-19	15813700
1	2021-11-22	16905600
2	2021-11-23	13835900
3	2021-11-24	12598900
4	2021-11-26	14754300

```
# Переименуем столбцы в обучающем и тестовом датасетах, чтобы они подходили для использова
train_prophet_0_new.columns = ['ds', 'y'] # переименовали столбцы
test_prophet_0_new.columns = ['ds', 'y'] # переименовали столбцы
```

```
train_prophet_0_new.head()
```

	ds	y
0	1962-01-02	806400
1	1962-01-03	1574400
2	1962-01-04	844800
3	1962-01-05	1420800
4	1962-01-08	2035200

```
test_prophet_0_new.head()
```

	ds	y
0	2021-11-19	15813700
1	2021-11-22	16905600
2	2021-11-23	13835900
3	2021-11-24	12598900
4	2021-11-26	14754300

```
model = Prophet()
model.fit(train_prophet_0_new) # подогнали модель под наши данные
```

```
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True t
<fbprophet.forecaster.Prophet at 0x7f5bc3d7ad90>
```

Промежуточные выводы:

Алгоритм проигнорировал, выбирая параметры

Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.

годовую сезонность (для годовой сезонности нам нужно иметь данные минимум за 2 года, чтобы суметь использовать ее в модели) INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.

дневную сезонность (дневная сезонность может использоваться в случае, если данные собираются по часам/минутам, в нашем случае данные представлены по месяцам). Зато он обнаружил недельную сезонность и использовал его при настройке модели.

```
# говорим профету сделать дата-фрейм на 15 дней вперед
future = model.make_future_dataframe(periods=15, freq='W')
future.tail() # выводим 5 строк с конца
```

	ds
15086	2022-01-30
15087	2022-02-06
15088	2022-02-13
15089	2022-02-20
15090	2022-02-27

```
# предсказываем значения по модели, доверительный интервал по умолчанию 95%
```

```
forecast = model.predict(future)
forecast.head() # возвращает много колонок
```

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper
0	1962-01-02	972065.160830	-6.276535e+06	6.567038e+06	972065.160830	972065.160830
1	1962-01-03	972052.898636	-6.620535e+06	6.808263e+06	972052.898636	972052.898636
2	1962-01-04	972040.636443	-7.051706e+06	6.258448e+06	972040.636443	972040.636443
3	1962-01-05	972028.374249	-5.918402e+06	6.878686e+06	972028.374249	972028.374249
4	1962-01-08	971991.587667	-6.807430e+06	6.455872e+06	971991.587667	971991.587667

Основные поля в прогнозе следующие:

ds — дата прогноза

yhat — спрогнозированное значение

yhat_lower — нижняя граница доверительного интервала для прогноза

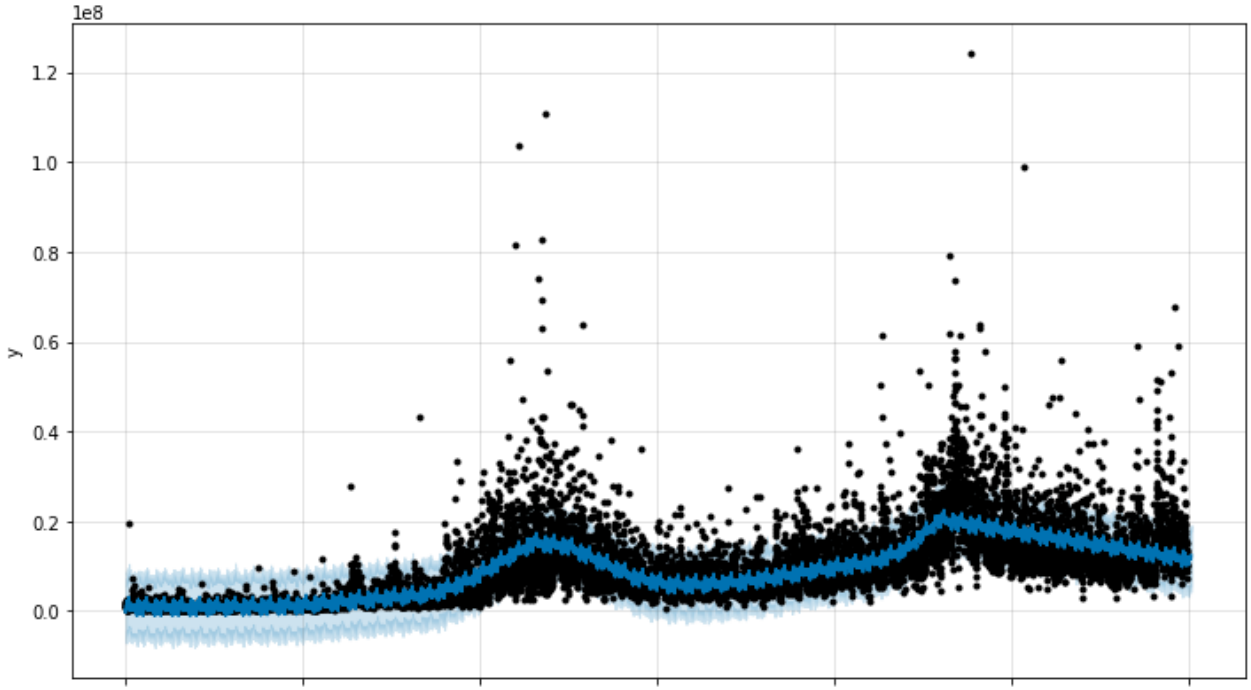
yhat_upper — верхняя граница доверительного интервала для прогноза

```
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail() # оставим только нужные
```

	ds	yhat	yhat_lower	yhat_upper
15086	2022-01-30	1.221374e+07	5.450933e+06	1.875159e+07
15087	2022-02-06	1.177093e+07	5.052111e+06	1.818367e+07
15088	2022-02-13	1.148307e+07	5.457614e+06	1.791894e+07
15089	2022-02-20	1.171377e+07	4.942068e+06	1.807594e+07
15090	2022-02-27	1.239107e+07	5.786342e+06	1.912836e+07

сравниваем прогноз и тестовую выборку, где черные точки - выборка

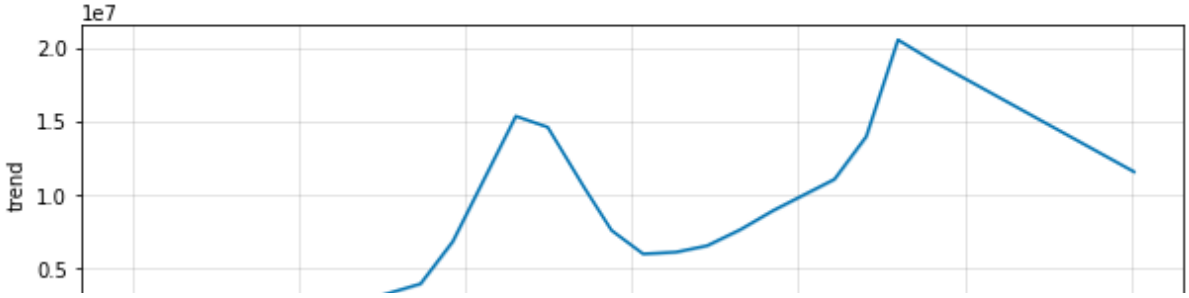
```
model.plot(forecast);
```



Кроме того, Prophet позволяет также наглядно разложить ряд на основные компоненты — тренд и сезонность:

plot_components() — возвращает несколько графиков, среди которых тренд и столько сезонностей, сколько он найдет.

```
model.plot_components(forecast);
```



Промежуточные выводы: Видим, наличие возрастающего тренда стоимости акций и годовую сезонность. Видим каким образом изменяются цены по месяцам.

```
#оцениваем качество модели методом MSE, RMSE, MAE, MAPE

mae_error = mean_absolute_error(test_prophet_0_new['y'], forecast['yhat'].tail(15))
mse_error = mean_squared_error(test_prophet_0_new['y'], forecast['yhat'].tail(52))
rmse_error = rmse(test_prophet_0_new['y'], forecast['yhat'].tail(52))
mape_error = np.mean(np.abs(forecast['yhat'] - test_prophet_0_new['y'])/test_prophet_0_new

print(f'Prophet MAE Error: {mae_error:11.10}')
print(f'Prophet MSE Error: {mse_error:11.10}')
print(f'Prophet RMSE Error: {rmse_error:11.10}')
print(f'Prophet MAPE Error: {mape_error:11.10}')
```

```
# делаем прогноз на будущее - 1 год вперед

# обучаем модель на всем датасете
fut_model_prophet = Prophet(seasonality_mode='multiplicative')
fut_model_prophet.fit(df1)

# говорим профету сделать дата-фрейм на 1 год вперед
fut_future_prophet = fut_model_prophet.make_future_dataframe(periods=52)
fut_fcast_prophet = fut_model_prophet.predict(fut_future_prophet)


# устанавливаем индекс
fut_fcast_prophet.index = fut_fcast_prophet.ds


# убедимся в изменениях
fut_fcast_prophet.head()


#строим график на будущее

ax = df1['Open'].plot(legend=True, figsize=(10,3),title='Open price')
fut_fcast_prophet['yhat'].tail(12).plot(legend=True)
ax.autoscale(axis='x',tight=True)
ax.legend(["Open", "PROPHET"]);C
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-201-0e6a387a16ba> in <module>()
      2
      3 ax = df1['Open'].plot(legend=True, figsize=(10,3),title='Open price')
----> 4 fut_fcast_prophet['yhat'].tail(12).plot(legend=True)
      5 ax.autoscale(axis='x',tight=True)
      6 ax.legend(["Open", "PROPHET"]);C

NameError: name 'fut_fcast_prophet' is not defined
```

SEARCH STACK OVERFLOW

3й метод прогнозирования "Экспоненциальное сглаживание" (Exponential smoothing)

Date

Описание модели Exponential smoothing: to do

Экспоненциальное сглаживание — метод математического преобразования, используемый при прогнозировании временных рядов.

Метод также известен как метод простого экспоненциального сглаживания, или метод Брауна

```
# создаем модель с подобранными параметрами
model_exps = ExponentialSmoothing (train['Volume'], seasonal_periods=7, trend = 'add')# ro
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:219: ValueWa
  the underlying index is a RangeIndex or an integral index.
```

```
#обучаем модель на обучающей выборке данных
model_exps.fit()
```

```
<statsmodels.tsa.holtwinters.HoltWintersResultsWrapper at 0x7f5bbed0cb10>
```

```
#посмотрим на подобранные параметры модели
model_exps.params
```

```
{'damping_slope': nan,
 'initial_level': 806400.0,
 'initial_seasons': array([], dtype=float64),
 'initial_slope': 767999.9999999998,
 'lamda': None,
```

```
 'remove_bias': False,
 'smoothing_level': 0.2918169471796316,
 'smoothing_seasonal': nan,
 'smoothing_slope': 0.003272306024537332,
 'use_boxcox': False}
```

```
import pandas as pd
```

```
#предсказываем значения,передав модели results точку начала и окончания
prediction_exps = model_exps.predict(model_exps.params, start=test.index[0], end=test.inde
```

```
test.index[0]
```

```
Timestamp('2021-11-19 00:00:00')
```

```
test.index[-1]
```

```
Timestamp('2021-12-17 00:00:00')
```

```
prediction_exps
```

```
#преобразуем в датафрейм с индексами
prediction_exps = pd.DataFrame(prediction_exps)
prediction_exps.index = pd.date_range("2021-12-10 00:00:00", periods=10, freq="D")

prediction_exps.columns = ['prediction_exps']
```

```
#проверим
prediction_exps.head()
```

```
#сравниваем прогноз и тестовую выборку
ax = test['Volume'].plot(legend=True, figsize=(12,6),title='Объем продаж')
prediction_exps['prediction_exps'].plot(legend=True)
ax.autoscale(axis='x',tight=True)
```

```
#проверим
prediction_exps.head()
```

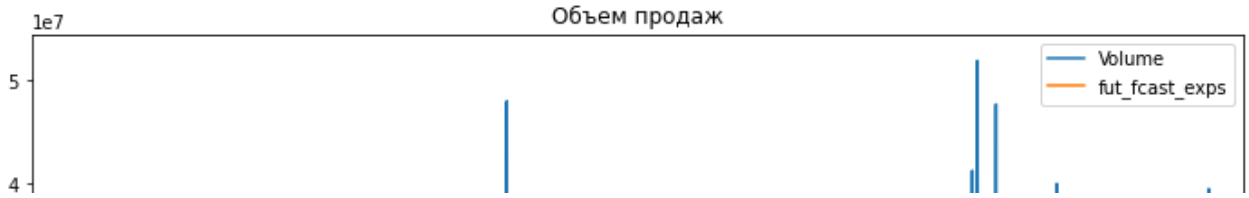
```
#сравниваем прогноз и тестовую выборку
ax = test.plot(legend=True, figsize=(12,6),title='Объем продаж')
prediction_exps['prediction_exps'].plot(legend=True)
ax.autoscale(axis='x',tight=True)
```

```
#оцениваем качество модели методом MSE, RMSE, MAE, MAPE
mae_error = mean_absolute_error(test, prediction_exps['prediction_exps'])
mse_error = mean_squared_error(test, prediction_exps['prediction_exps'])
rmse_error = rmse(test, prediction_exps['prediction_exps'])
mape_error = np.mean(np.abs(prediction_exps['prediction_exps'] - test)/test)*100
```

```
print(f'Exponential smoothing MAE Error: {mae_error:11.10}')
print(f'Exponential smoothing MSE Error: {mse_error:11.10}')
print(f'Exponential smoothing RMSE Error: {rmse_error:11.10}')
print(f'Exponential smoothing MAPE Error: {mape_error:11.10}')


#делаем прогноз на будущее

# обучаем модель на всем датасете
fut_model_exps = ExponentialSmoothing(df1['Open'], seasonal_periods=12, trend = 'add')
fut_model_exps.fit()

#задаем точки будущего
fut_fcast_exps = fut_model_exps.predict(fut_model_exps.params, start=len(df1)-1, end=len(d

#преобразуем в датафрейм с индексами
fut_fcast_exps = pd.DataFrame(fut_fcast_exps)
fut_fcast_exps.index = pd.date_range('2020-02-09 00:00:00', periods=12, freq="W-SUN")
fut_fcast_exps.columns = ['fut_fcast_exps']
#проверим
fut_fcast_exps.head()
```

	fut_fcast_exps
2020-02-09	54.908381
2020-02-16	57.774382
2020-02-23	57.792763
2020-03-01	57.811144
2020-03-08	57.829525

```
#строим график на будущее
ax = df1['Volume'].plot(legend=True, figsize=(12,6),title='Объем продаж')
fut_fcast_exps['fut_fcast_exps'].plot(legend=True)
ax.autoscale(axis='x',tight=True)
```



Выводы Проведен анализ данных с использованием современных методов обработки статистической информации.

Рассчитаны основные статистические метрики, позволяющие судить о характере исследуемого явления.

Результаты анализа представленных данных помогли выявить зависимость цены акции на мрмент открытия от объема продаж.

Прогнозная модель позволила зафиксировать что снижение цены имеет слабое влияние на объем продажи акций.