[1]:
```python
# Loading Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as pt
```

[2]:
```python
# Loading Bitcoin dataset in a dataframe named df

from google.colab import files
uploaded = files.upload()
import io
df = pd.read_csv(io.BytesIO(uploaded['BitcoinDataset3.csv']))
print(df)
```

```
<IPython.core.display.HTML object>

Saving BitcoinDataset3.csv to BitcoinDataset3.csv
              Date      Price       Open  …        Low     Vol. Change %
0     Aug 02, 2020  11,105.80  11,802.60  …  10,730.70  698.62K   -5.91%
1     Aug 01, 2020  11,803.10  11,333.20  …  11,226.10  611.47K    4.14%
2     Jul 31, 2020  11,333.40  11,096.50  …  10,964.60  530.95K    2.14%
3     Jul 30, 2020  11,096.20  11,105.80  …  10,861.60  501.14K   -0.09%
4     Jul 29, 2020  11,105.90  10,908.40  …  10,771.80  576.83K    1.81%
…              …          …          …  …          …        …        …
1185  May 05, 2017   1,507.80   1,516.80  …   1,485.00  120.38K   -0.59%
1186  May 04, 2017   1,516.80   1,485.60  …   1,437.10  136.71K    2.10%
1187  May 03, 2017   1,485.60   1,445.90  …   1,424.10   81.72K    2.74%
1188  May 02, 2017   1,445.90   1,415.80  …   1,394.80   70.01K    2.13%
1189  May 01, 2017   1,415.80   1,351.90  …   1,342.80  100.44K    0.00%

[1190 rows x 7 columns]
```

[3]:
```python
#Visualization of data

df
```

[3]:
```
              Date      Price       Open  …        Low     Vol. Change %
0     Aug 02, 2020  11,105.80  11,802.60  …  10,730.70  698.62K   -5.91%
1     Aug 01, 2020  11,803.10  11,333.20  …  11,226.10  611.47K    4.14%
2     Jul 31, 2020  11,333.40  11,096.50  …  10,964.60  530.95K    2.14%
3     Jul 30, 2020  11,096.20  11,105.80  …  10,861.60  501.14K   -0.09%
4     Jul 29, 2020  11,105.90  10,908.40  …  10,771.80  576.83K    1.81%
…              …          …          …  …          …        …        …
1185  May 05, 2017   1,507.80   1,516.80  …   1,485.00  120.38K   -0.59%
1186  May 04, 2017   1,516.80   1,485.60  …   1,437.10  136.71K    2.10%
1187  May 03, 2017   1,485.60   1,445.90  …   1,424.10   81.72K    2.74%
1188  May 02, 2017   1,445.90   1,415.80  …   1,394.80   70.01K    2.13%
1189  May 01, 2017   1,415.80   1,351.90  …   1,342.80  100.44K    0.00%
```

```
[1190 rows x 7 columns]
```

[4]:
```python
# Displaying the information of dataframe

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1190 entries, 0 to 1189
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      1190 non-null   object
 1   Price     1190 non-null   object
 2   Open      1190 non-null   object
 3   High      1190 non-null   object
 4   Low       1190 non-null   object
 5   Vol.      1187 non-null   object
 6   Change %  1190 non-null   object
dtypes: object(7)
memory usage: 65.2+ KB
None
```

[5]:
```python
# Changing the name of Volume Column to represent volumn in thousands

df.rename(columns = {'Vol.': 'Vol(in K)'}, inplace=True)
```

Changing the format of attribute values

[6]:
```python
# Changing the format of attribute values

df["Price"] = df["Price"].replace(",", "", regex=True)
df["Open"] = df["Open"].replace(",", "", regex=True)
df["High"] = df["High"].replace(",", "", regex=True)
df["Low"] = df["Low"].replace(",", "", regex=True)
df["Vol(in K)"] = df["Vol(in K)"].replace("K", "", regex=True)
df["Vol(in K)"] = df["Vol(in K)"].replace("M", "", regex=True)
df["Change %"] = df["Change %"].replace("%", "", regex=True)
```

[7]:
```python
#Displaying final Dataframe

df.head()
```

[7]:

	Date	Price	Open	High	Low	Vol(in K)	Change %
0	Aug 02, 2020	11105.80	11802.60	12061.10	10730.70	698.62	-5.91
1	Aug 01, 2020	11803.10	11333.20	11847.70	11226.10	611.47	4.14
2	Jul 31, 2020	11333.40	11096.50	11434.80	10964.60	530.95	2.14

```
3  Jul 30, 2020  11096.20  11105.80  11164.40  10861.60    501.14    -0.09
4  Jul 29, 2020  11105.90  10908.40  11336.50  10771.80    576.83     1.81
```

[8]:
```
# Converting Object Column to float

df["Price"] = df["Price"].astype('float64')
df["Open"] = df["Open"].astype('float64')
df["High"] = df["High"].astype('float64')
df["Low"] = df["Low"].astype('float64')
df['Vol(in K)'] = pd.to_numeric(df['Vol(in K)'],errors='coerce')
df["Change %"] = df["Change %"].astype('float64')
```

[9]:
```
# Displaying the Information of Dataframe

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1190 entries, 0 to 1189
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1190 non-null   object
 1   Price      1190 non-null   float64
 2   Open       1190 non-null   float64
 3   High       1190 non-null   float64
 4   Low        1190 non-null   float64
 5   Vol(in K)  1187 non-null   float64
 6   Change %   1190 non-null   float64
dtypes: float64(6), object(1)
memory usage: 65.2+ KB
None
```

[10]:
```
# Question: Are there any "Missing" values present in the Dataset?

df.isna().sum()
```

[10]:
```
Date         0
Price        0
Open         0
High         0
Low          0
Vol(in K)    3
Change %     0
dtype: int64
```

How many unique values each column has ?

[11]:
```
# How many unique values each column has?

df.nunique()
```

[11]:
```
Date          1190
Price         1184
Open          1178
High          1180
Low           1183
Vol(in K)     1122
Change %       776
dtype: int64
```

Take a look at the Summary of the Dataset

[12]:
```
# Take a look at the summary of the Dataset

df.describe().transpose()
```

[12]:
```
            count         mean          std  ...       50%        75%       max
Price      1190.0  7252.800504  2992.103176  ...  7292.850  9232.6250  19345.50
Open       1190.0  7244.515798  2994.837939  ...  7289.400  9230.5250  19346.60
High       1190.0  7452.483109  3110.469739  ...  7455.150  9392.3500  19870.60
Low        1190.0  7012.438824  2837.488382  ...  7121.900  9048.2000  18750.90
Vol(in K)  1187.0   320.976773   283.020520  ...   217.960   535.6700    999.53
Change %   1190.0     0.272403     4.422303  ...     0.155     2.1375     25.56

[6 rows x 8 columns]
```

[13]:
```
# Converting date Column to standard format i.e. in numeric format so that we
#can do some month level analysis as well

df['Date']=pd.to_datetime(df['Date'])
```

[14]:
```
# Again Displaying dataframe

print(df)
```

```
           Date    Price     Open     High      Low  Vol(in K)  Change %
0    2020-08-02  11105.8  11802.6  12061.1  10730.7     698.62     -5.91
1    2020-08-01  11803.1  11333.2  11847.7  11226.1     611.47      4.14
2    2020-07-31  11333.4  11096.5  11434.8  10964.6     530.95      2.14
3    2020-07-30  11096.2  11105.8  11164.4  10861.6     501.14     -0.09
4    2020-07-29  11105.9  10908.4  11336.5  10771.8     576.83      1.81
...         ...      ...      ...      ...      ...        ...
1185 2017-05-05   1507.8   1516.8   1588.1   1485.0     120.38     -0.59
1186 2017-05-04   1516.8   1485.6   1609.8   1437.1     136.71      2.10
1187 2017-05-03   1485.6   1445.9   1496.4   1424.1      81.72      2.74
```

```
1188 2017-05-02   1445.9   1415.8   1471.1   1394.8      70.01       2.13
1189 2017-05-01   1415.8   1351.9   1448.7   1342.8     100.44       0.00

[1190 rows x 7 columns]
```

[15]:
```python
# Setting date as Index

df_non_indexed=df.copy()
df=df.set_index('Date')
```

[16]:
```python
# Displaying top 5 entries of our Dataframe to verify the Date index column

df.head()
```

[16]:
```
              Price     Open     High      Low  Vol(in K)  Change %
Date
2020-08-02  11105.8  11802.6  12061.1  10730.7     698.62     -5.91
2020-08-01  11803.1  11333.2  11847.7  11226.1     611.47      4.14
2020-07-31  11333.4  11096.5  11434.8  10964.6     530.95      2.14
2020-07-30  11096.2  11105.8  11164.4  10861.6     501.14     -0.09
2020-07-29  11105.9  10908.4  11336.5  10771.8     576.83      1.81
```

[17]:
```python
# Getting Data between two dates

df.loc['2020-08-02':'2020-07-29']
```

[17]:
```
              Price     Open     High      Low  Vol(in K)  Change %
Date
2020-08-02  11105.8  11802.6  12061.1  10730.7     698.62     -5.91
2020-08-01  11803.1  11333.2  11847.7  11226.1     611.47      4.14
2020-07-31  11333.4  11096.5  11434.8  10964.6     530.95      2.14
2020-07-30  11096.2  11105.8  11164.4  10861.6     501.14     -0.09
2020-07-29  11105.9  10908.4  11336.5  10771.8     576.83      1.81
```

[18]:
```python
# Getting data between two years

df.loc['2019':'2018']
```

[18]:
```
              Price     Open     High      Low  Vol(in K)  Change %
Date
2019-12-31   7196.4   7261.5   7331.0   7167.4     586.60     -0.90
2019-12-30   7261.8   7397.5   7420.9   7244.1     606.11     -1.84
2019-12-29   7397.5   7321.6   7518.9   7303.0     611.69      1.04
2019-12-28   7321.5   7261.9   7375.9   7256.5     610.96      0.82
2019-12-27   7261.7   7210.8   7293.8   7128.5     718.07      0.70
…                …        …        …        …          …        …
2018-01-05  16954.8  15180.1  17126.9  14832.4     141.96     11.69
```

```
2018-01-04  15180.1  15156.5  15408.7  14244.7     110.97      0.15
2018-01-03  15156.6  14754.1  15435.0  14579.7     106.54      2.73
2018-01-02  14754.1  13444.9  15306.1  12934.2     137.73      9.74
2018-01-01  13444.9  13850.5  13921.5  12877.7      78.43     -2.93

[730 rows x 6 columns]
```

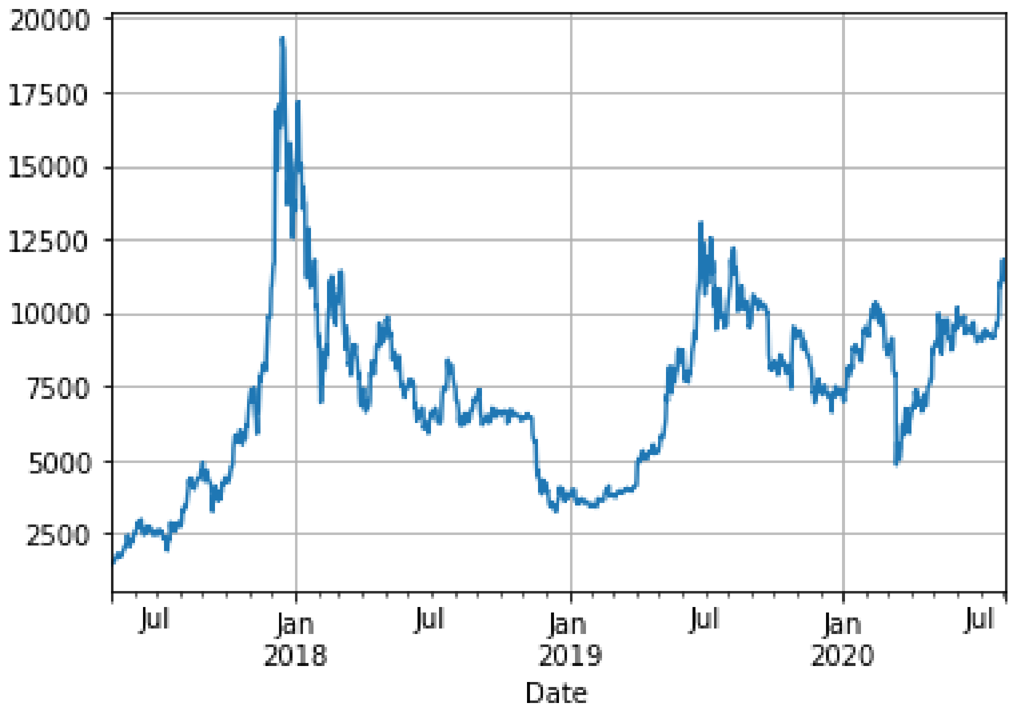
[19]:
```python
# Setting Price column as our target column

print('Please identify the Target column')
print('\n')
target=df['Price']
target.head()
```

```
Please identify the Target column
```

[19]:
```
Date
2020-08-02    11105.8
2020-08-01    11803.1
2020-07-31    11333.4
2020-07-30    11096.2
2020-07-29    11105.9
Name: Price, dtype: float64
```

[20]:
```python
# Plotting of Target Data

target.plot(grid=True)
```

[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabdc2c3f50>
```



Conclusion from above plot:

[21]:
```
# To Visualize Plotting of a particular year 2018

df_2018=df.loc['2018']
target_2018=df_2018['Price']
target_2018.plot(grid=True)
```

[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabdbf22a50>

Conclusion from above plot:

[22]:

```
# To Visualize Plotting of a particular year 2019

df_2019=df.loc['2019']
target_2019=df_2019['Price']
target_2019.plot(grid=True)
```

[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabdb9b9e10>

Coclusion from above plot:

[23]:
```
# To Visualize Plotting of a particular year

import plotly.express as px
fig=px.line(df_non_indexed, x='Date',y='Price', title='Price with slider')
fig.update_xaxes(rangeslider_visible=True)
fig.show()
```

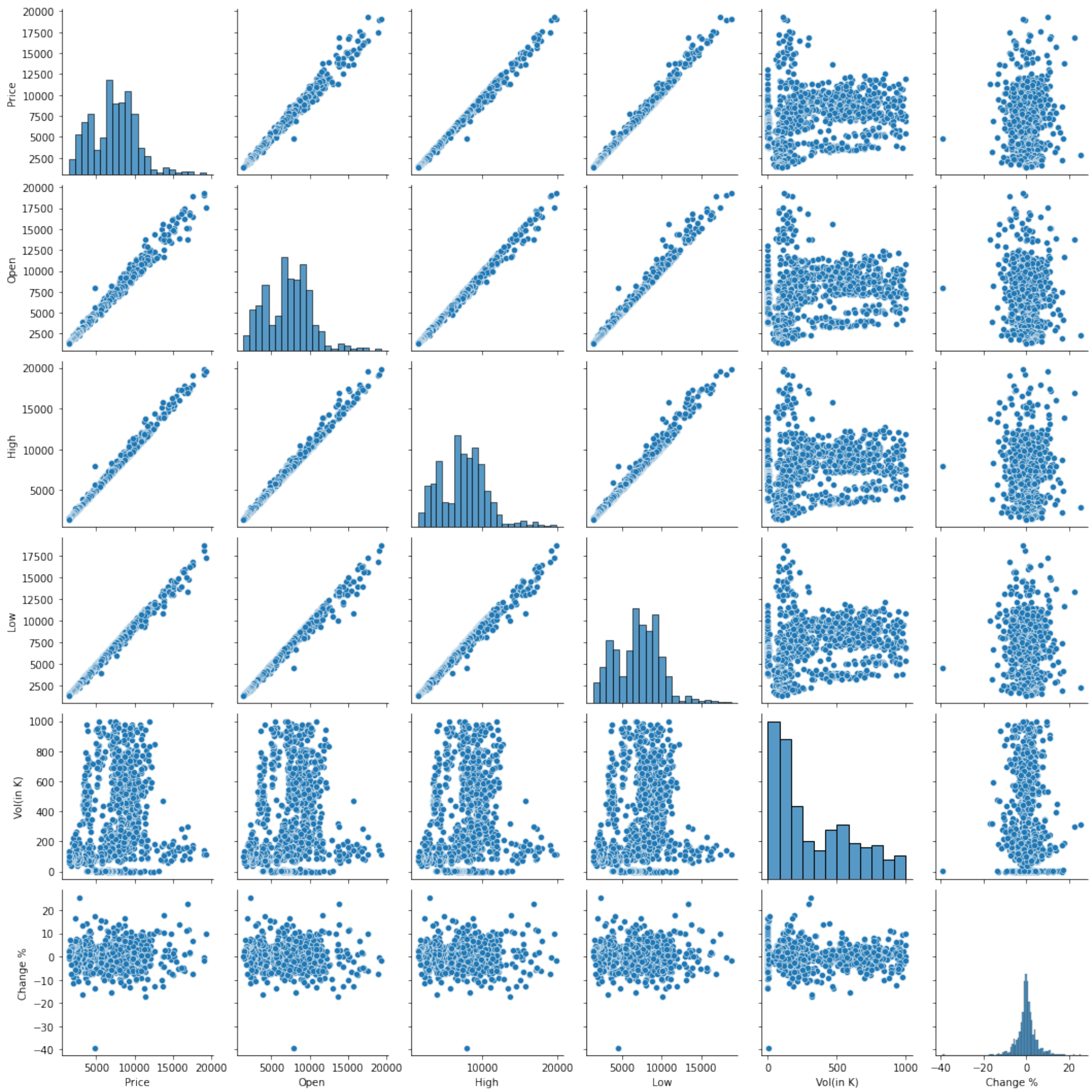
Putting Button to directly switch between years in graph

[24]:
```
fig=px.line(df_non_indexed, x='Date',y='Price', title='Price with slider')
fig.update_xaxes(
    rangeslider_visible=True, rangeselector=dict(
        buttons=list([
              dict(count=1, label="1y", step="year", stepmode="backward"),
              dict(count=2, label="2y", step="year", stepmode="backward"),
              dict(count=3, label="3y", step="year", stepmode="backward"),
              dict(count=4, label="4y", step="year", stepmode="backward"),
              dict(step="all")
        ])
    ))
```

[25]:
```
# Plotting all variables

all_var_plot=df[['Price','Open','High','Low','Vol(in K)','Change %']]
all_var_plot.plot(subplots=True)
```

[25]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fabd8be5210>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fabd8c87750>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fabd8c6b350>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fabd93d9690>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fabd8faf9d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fabd9362390>],
      dtype=object)
```

[26]:
```
# Checking correlation between two different variables

import seaborn as sns
g=sns.pairplot(df[['Price','Open','High','Low','Vol(in K)','Change %']])
```



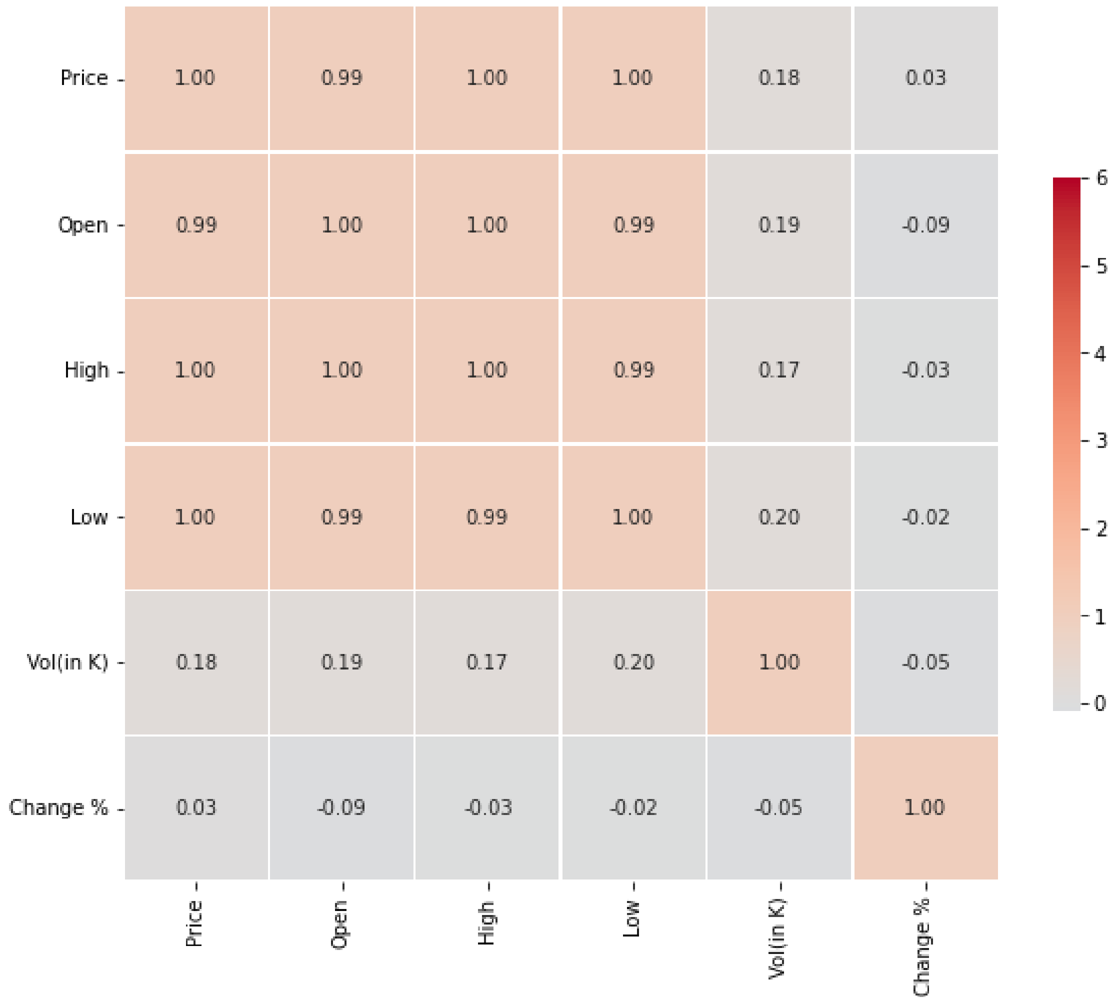

```
[27]: # Checking correlation in form of matrix

      corr=df[['Price','Open','High','Low','Vol(in K)','Change %']].
       ↪corr(method='pearson')
      corr
```

[27]:

	Price	Open	High	Low	Vol(in K)	Change %
Price	1.000000	0.992000	0.996439	0.995404	0.182075	0.026430
Open	0.992000	1.000000	0.996026	0.992945	0.185995	-0.087664
High	0.996439	0.996026	1.000000	0.991235	0.173983	-0.026556
Low	0.995404	0.992945	0.991235	1.000000	0.197464	-0.018682
Vol(in K)	0.182075	0.185995	0.173983	0.197464	1.000000	-0.054359
Change %	0.026430	-0.087664	-0.026556	-0.018682	-0.054359	1.000000

[28]:

```
# Geberating Heatmap correlation matrix to visualize correlation in better way

h=sns.heatmap(corr, vmax=6, center=0, square=True, linewidths=.5,␣
 ↪cbar_kws={"shrink":.5}, annot=True, fmt='.2f', cmap='coolwarm')
h.figure.set_size_inches(10,10)
pt.show()
```



[29]:

```
# Dealing with missing values

df.isna().sum()
```

[29]:

```
Price        0
Open         0
High         0
Low          0
Vol(in K)    3
```

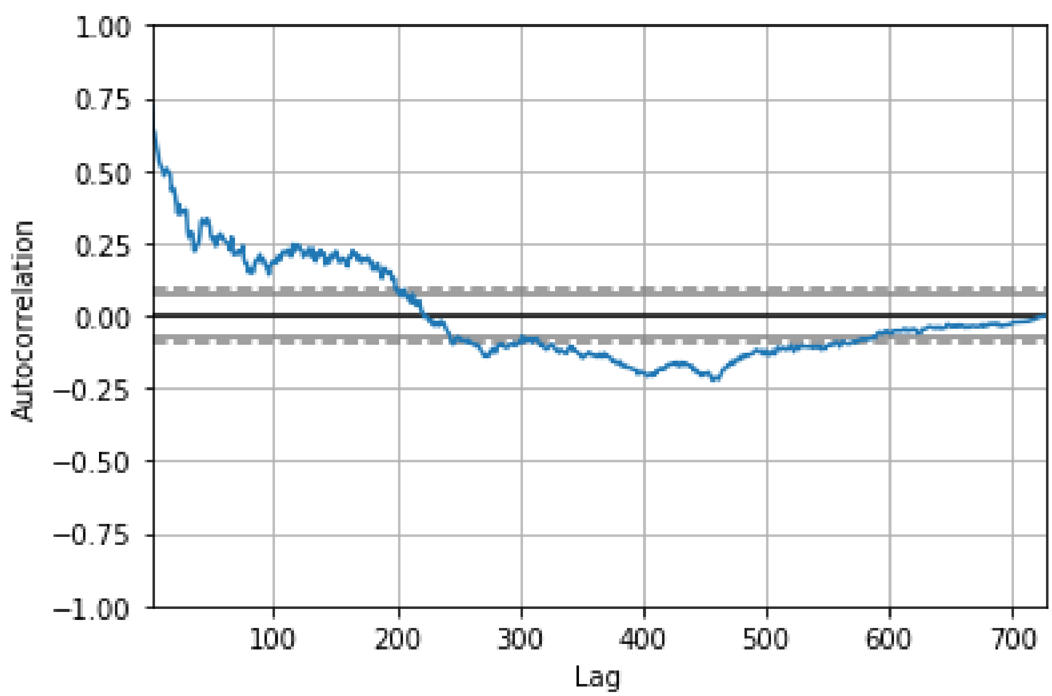
```
Change %      0
dtype: int64
```

[30]:
```
# Visualization of missing values

fig=px.line(df_non_indexed, x='Date',y='Vol(in K)', title='Volume with slider')
fig.update_xaxes(
    rangeslider_visible=True, rangeselector=dict(
        buttons=list([
              dict(count=1, label="1y", step="year", stepmode="backward"),
              dict(count=2, label="2y", step="year", stepmode="backward"),
              dict(count=3, label="3y", step="year", stepmode="backward"),
              dict(count=4, label="4y", step="year", stepmode="backward"),
              dict(step="all")
        ])
    ))
```

[31]:
```
# Auto correlation graph

df_na=df.copy()
df_na=df_na.dropna()
```

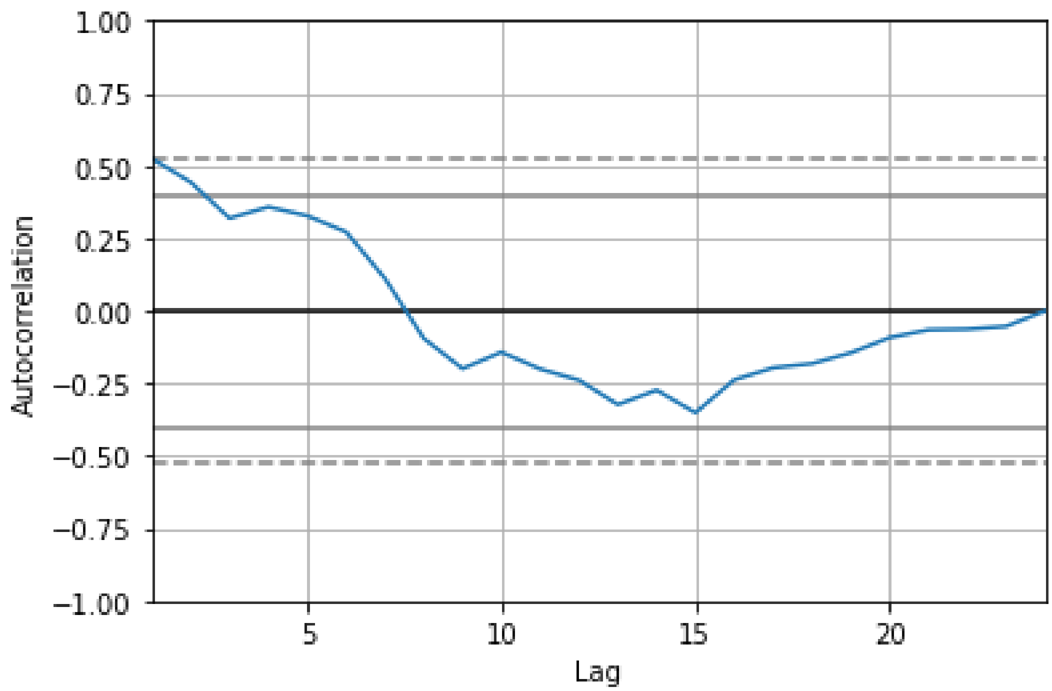
[32]:
```
# Visualizing Data in days form from 2018 to 2019

pd.plotting.autocorrelation_plot(df_na['2019':'2018']['Vol(in K)'])
```

[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabbeebb1d0>



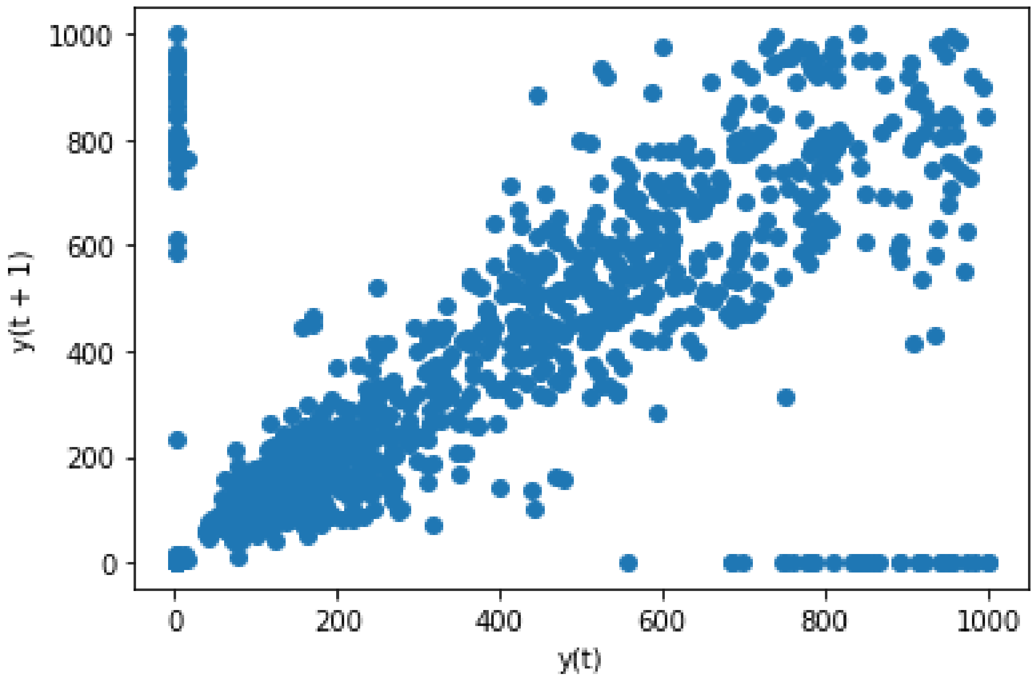
[33]:
```
# Visualizing Data in months form from 2018 to 2019

pd.plotting.autocorrelation_plot(df_na['2019':'2018']['Vol(in K)'].
 ↪resample("1m").mean())
```

[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabbef07ad0>



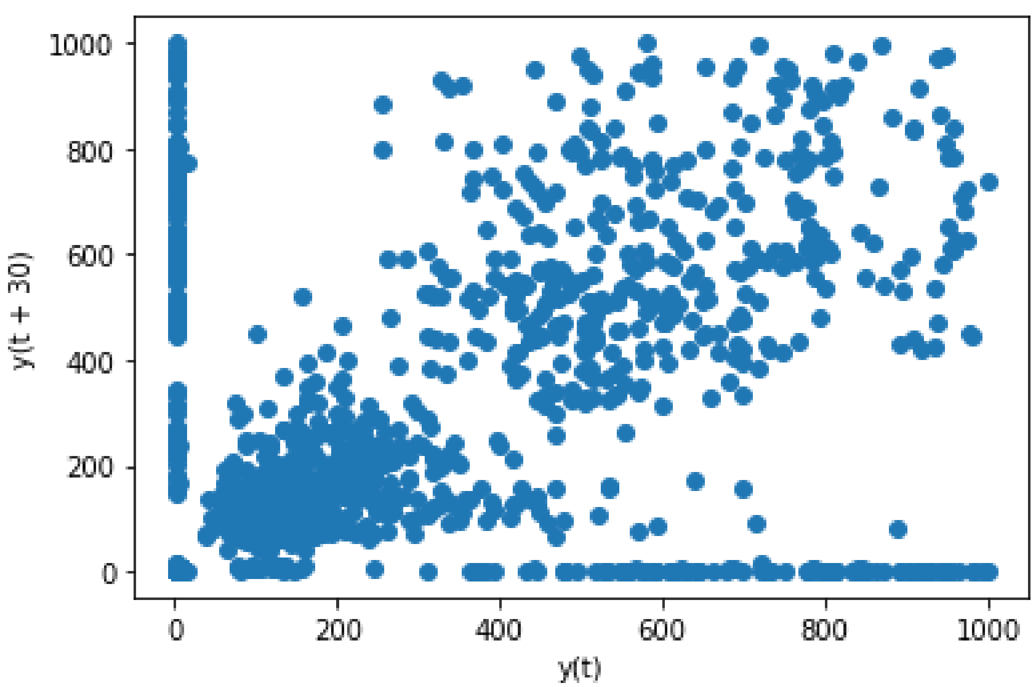
[34]:
```
pd.plotting.lag_plot(df['Vol(in K)'],lag=1)
```

[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabbe6a2950>



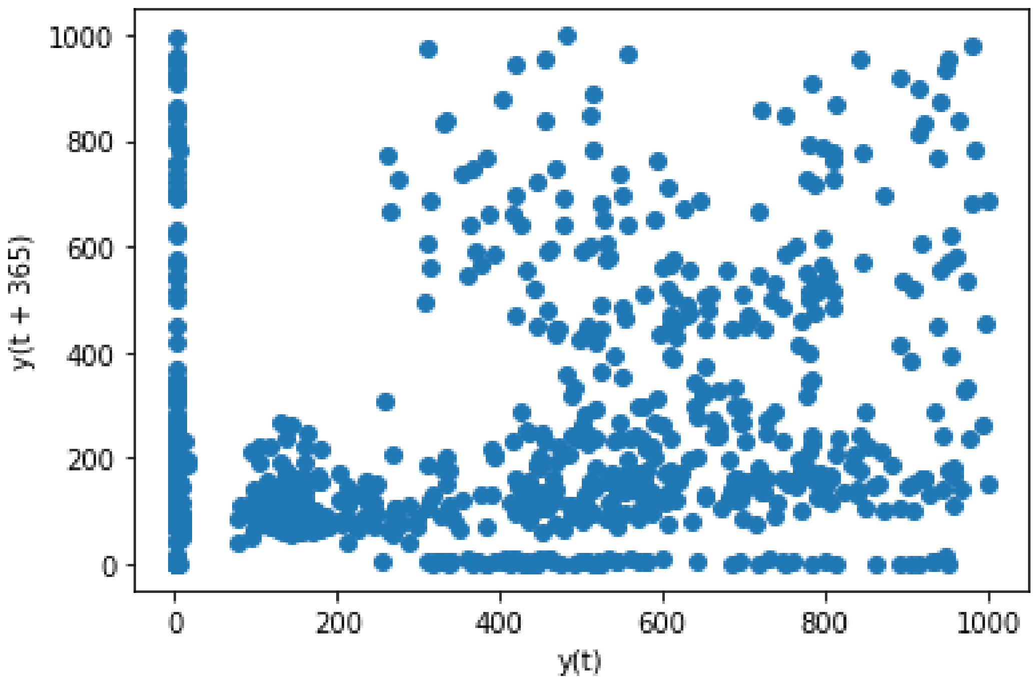

```
[35]: pd.plotting.lag_plot(df['Vol(in K)'],lag=30)
```

```
[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabbe668710>
```



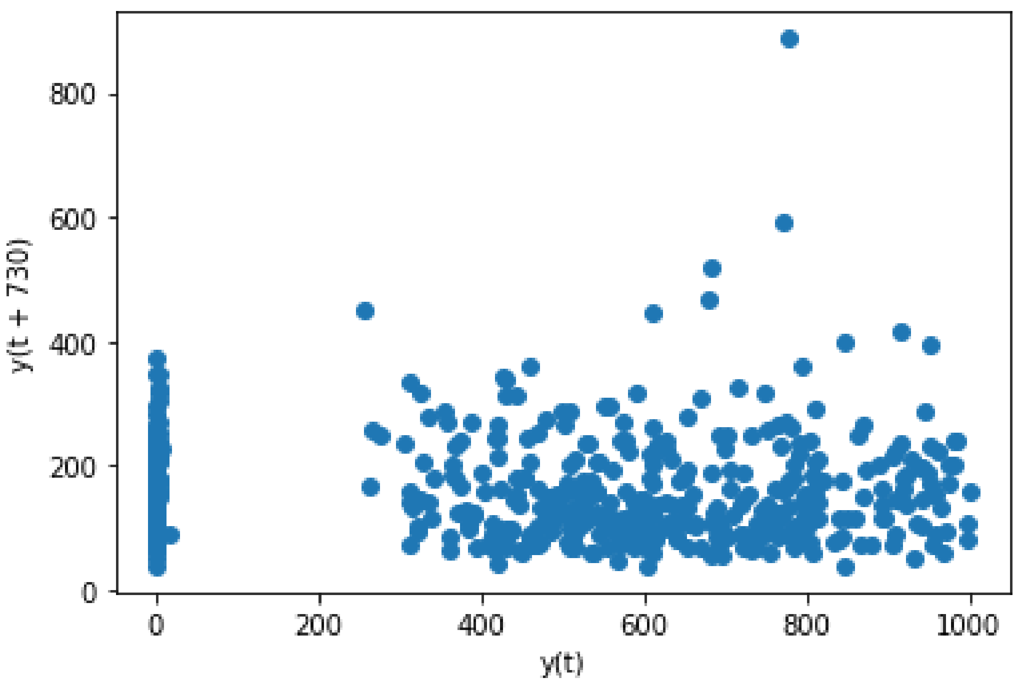
[36]:
```
pd.plotting.lag_plot(df['Vol(in K)'],lag=365)
```

[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabbe5de410>



[37]:
```
pd.plotting.lag_plot(df['Vol(in K)'],lag=730)
```

[37]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabbe5c8d50>



6 Missing Values

we have already seen various methods to deal with missing values like: global mean, global median, drop values etc. But in time series we can not deal with missing data using drop values. The reason is: in time series we care about the order of events. In addition to this we can not even use global median or global mean because our data might have some seasonality or trend and if we do global median we may messup the current information. therefore, we will use different methods here to fill our missing data like: forward fill or backward fill methods.

[38]:
```
df_imp=df['2018-12-31':'2018-11-25'][['Vol(in K)']]
```

[39]:
```
df_imp
```

[39]:
```
            Vol(in K)
Date
2018-12-31        NaN
2018-12-30     519.17
2018-12-29     505.41
2018-12-28     565.24
2018-12-27     543.44
2018-12-26     567.11
2018-12-25     670.94
2018-12-24     716.29
2018-12-23     572.21
```

```
2018-12-22     554.59
2018-12-21     748.76
2018-12-20        NaN
2018-12-19     792.40
2018-12-18     698.53
2018-12-17     534.78
2018-12-16     335.69
2018-12-15     332.95
2018-12-14     445.56
2018-12-13     435.89
2018-12-12     391.17
2018-12-11     418.53
2018-12-10     394.49
2018-12-09     327.93
2018-12-08     424.40
2018-12-07     639.33
2018-12-06     467.54
2018-12-05     341.60
2018-12-04     326.71
2018-12-03     355.10
2018-12-02     298.63
2018-12-01     316.30
2018-11-30     444.52
2018-11-29     414.46
2018-11-28     533.04
2018-11-27     457.27
2018-11-26     571.04
2018-11-25        NaN
```

[40]:
```
df_imp['Vol_Ffill']=df_imp['Vol(in K)'].fillna(method='ffill')
```

[41]:
```
df_imp
```

[41]:
```
            Vol(in K)  Vol_Ffill
Date
2018-12-31        NaN        NaN
2018-12-30     519.17     519.17
2018-12-29     505.41     505.41
2018-12-28     565.24     565.24
2018-12-27     543.44     543.44
2018-12-26     567.11     567.11
2018-12-25     670.94     670.94
2018-12-24     716.29     716.29
2018-12-23     572.21     572.21
2018-12-22     554.59     554.59
2018-12-21     748.76     748.76
2018-12-20        NaN     748.76
```

```
2018-12-19      792.40      792.40
2018-12-18      698.53      698.53
2018-12-17      534.78      534.78
2018-12-16      335.69      335.69
2018-12-15      332.95      332.95
2018-12-14      445.56      445.56
2018-12-13      435.89      435.89
2018-12-12      391.17      391.17
2018-12-11      418.53      418.53
2018-12-10      394.49      394.49
2018-12-09      327.93      327.93
2018-12-08      424.40      424.40
2018-12-07      639.33      639.33
2018-12-06      467.54      467.54
2018-12-05      341.60      341.60
2018-12-04      326.71      326.71
2018-12-03      355.10      355.10
2018-12-02      298.63      298.63
2018-12-01      316.30      316.30
2018-11-30      444.52      444.52
2018-11-29      414.46      414.46
2018-11-28      533.04      533.04
2018-11-27      457.27      457.27
2018-11-26      571.04      571.04
2018-11-25         NaN      571.04
```

[42]:
```
df_imp['Vol_Bfill']=df_imp['Vol(in K)'].fillna(method='bfill')
```

[43]:
```
df_imp
```

[43]:
```
            Vol(in K)  Vol_Ffill  Vol_Bfill
Date
2018-12-31         NaN        NaN     519.17
2018-12-30      519.17     519.17     519.17
2018-12-29      505.41     505.41     505.41
2018-12-28      565.24     565.24     565.24
2018-12-27      543.44     543.44     543.44
2018-12-26      567.11     567.11     567.11
2018-12-25      670.94     670.94     670.94
2018-12-24      716.29     716.29     716.29
2018-12-23      572.21     572.21     572.21
2018-12-22      554.59     554.59     554.59
2018-12-21      748.76     748.76     748.76
2018-12-20         NaN     748.76     792.40
2018-12-19      792.40     792.40     792.40
2018-12-18      698.53     698.53     698.53
2018-12-17      534.78     534.78     534.78
```

```
2018-12-16     335.69     335.69     335.69
2018-12-15     332.95     332.95     332.95
2018-12-14     445.56     445.56     445.56
2018-12-13     435.89     435.89     435.89
2018-12-12     391.17     391.17     391.17
2018-12-11     418.53     418.53     418.53
2018-12-10     394.49     394.49     394.49
2018-12-09     327.93     327.93     327.93
2018-12-08     424.40     424.40     424.40
2018-12-07     639.33     639.33     639.33
2018-12-06     467.54     467.54     467.54
2018-12-05     341.60     341.60     341.60
2018-12-04     326.71     326.71     326.71
2018-12-03     355.10     355.10     355.10
2018-12-02     298.63     298.63     298.63
2018-12-01     316.30     316.30     316.30
2018-11-30     444.52     444.52     444.52
2018-11-29     414.46     414.46     414.46
2018-11-28     533.04     533.04     533.04
2018-11-27     457.27     457.27     457.27
2018-11-26     571.04     571.04     571.04
2018-11-25        NaN     571.04        NaN
```

[44]:
```
new_df=df.fillna(method='ffill')
```

[45]:
```
new_df.isna().sum()
```

[45]:
```
Price         0
Open          0
High          0
Low           0
Vol(in K)     0
Change %      0
dtype: int64
```

[46]:
```
new_df
```

[46]:
```
              Price     Open     High      Low  Vol(in K)  Change %
Date
2020-08-02  11105.8  11802.6  12061.1  10730.7     698.62     -5.91
2020-08-01  11803.1  11333.2  11847.7  11226.1     611.47      4.14
2020-07-31  11333.4  11096.5  11434.8  10964.6     530.95      2.14
2020-07-30  11096.2  11105.8  11164.4  10861.6     501.14     -0.09
2020-07-29  11105.9  10908.4  11336.5  10771.8     576.83      1.81
…                 …      …        …        …        …
2017-05-05   1507.8   1516.8   1588.1   1485.0     120.38     -0.59
2017-05-04   1516.8   1485.6   1609.8   1437.1     136.71      2.10
```

```
2017-05-03   1485.6   1445.9   1496.4   1424.1     81.72      2.74
2017-05-02   1445.9   1415.8   1471.1   1394.8     70.01      2.13
2017-05-01   1415.8   1351.9   1448.7   1342.8    100.44      0.00

[1190 rows x 6 columns]
```

#Decomposition of Time series

Time series decomposition is a statistical technique that deconstruct a time into several componets like trend, seasonality, clyclic and stationarity.

Need of Decomposition: Some models work better if data is stationary and others work better if data is not stationary. So decomposition of data helps us to find out which time series model we should use.

[47]:
```
fig=px.line(df_non_indexed, x='Date',y='Price', title='Price with slider')
fig.update_xaxes(
    rangeslider_visible=True, rangeselector=dict(
        buttons=list([
              dict(count=1, label="1y", step="year", stepmode="backward"),
              dict(count=2, label="2y", step="year", stepmode="backward"),
              dict(count=3, label="3y", step="year", stepmode="backward"),
              dict(count=4, label="4y", step="year", stepmode="backward"),
              dict(step="all")
        ])
    ))
```

Conclusion: There is no seasonality or trend in price from 2017 to 2020.

[48]:
```
# kpss test to check stationarity. It can be stationary around the mean, means␣
 ↪there is no trend in data or it can be stationary around the trend line



from statsmodels.tsa.stattools import kpss
tstest=kpss(new_df['Price'],'c')
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning:

pandas.util.testing is deprecated. Use the functions in the public API at
pandas.testing instead.

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/stattools.py:1685:
FutureWarning:

The behavior of using lags=None will change in the next release. Currently
lags=None is the same as lags='legacy', and so a sample-size lag length is used.
After the next release, the default will change to be the same as lags='auto'
which uses an automatic lag length selection method. To silence this warning,
```

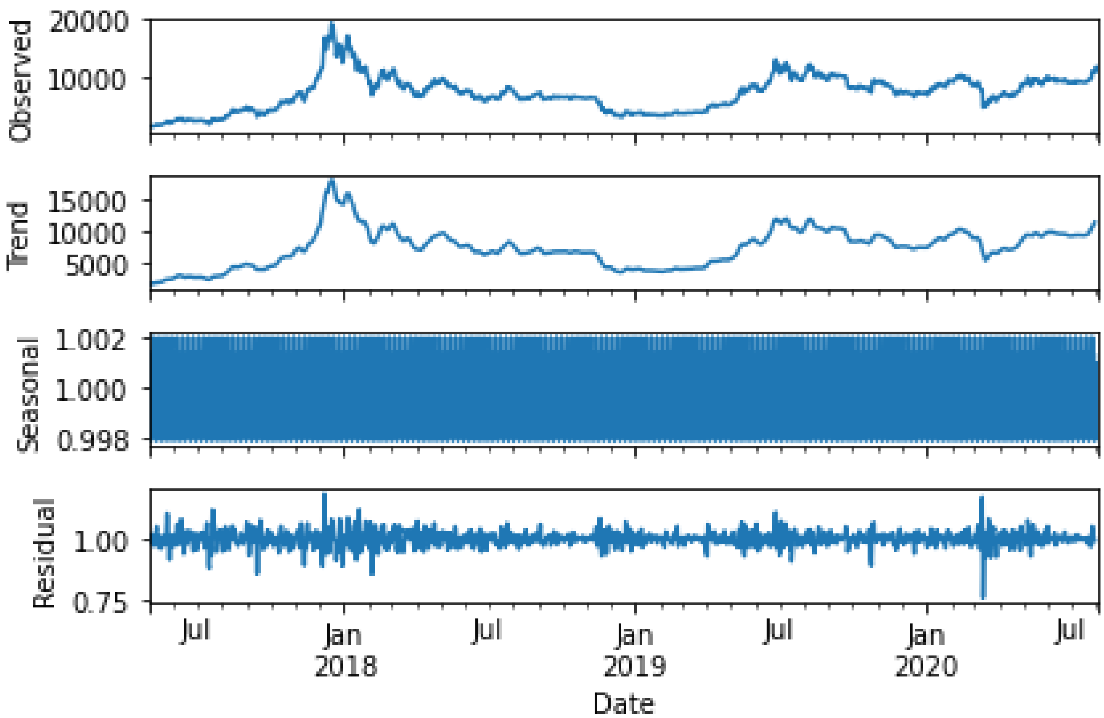
either use 'auto' or 'legacy'

[49]:
```
tstest
```

[49]:
```
(0.6829481326311675,
 0.015095624306257499,
 23,
 {'1%': 0.739, '10%': 0.347, '2.5%': 0.574, '5%': 0.463})
```

Observation: The second column is having p value 0.01 which is less than 0.05. Hence our data is not stationary

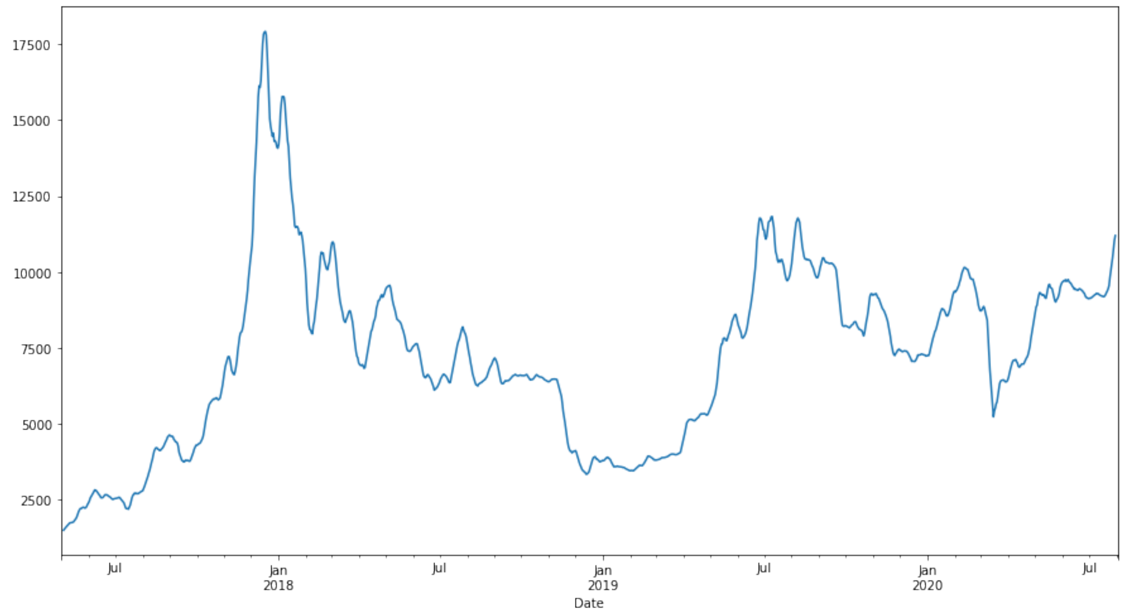
[50]:
```
# Above we have seen that data is not stationary. Hence while decomposition we␣
 ↪will use multiplicative model.

import statsmodels.api as sm
res=sm.tsa.seasonal_decompose(new_df['Price'],model='multiplicative')
resplot=res.plot()
```



Observation: ??

[51]:
```
pt.figure(figsize=(15,8))
res.trend.plot()
```

[51]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabae298590>



Observation: ??

[52]:
```
res.observed
```

[52]:
```
Date
2020-08-02    11105.8
2020-08-01    11803.1
2020-07-31    11333.4
2020-07-30    11096.2
2020-07-29    11105.9
               …
2017-05-05     1507.8
2017-05-04     1516.8
2017-05-03     1485.6
2017-05-02     1445.9
2017-05-01     1415.8
Name: Price, Length: 1190, dtype: float64
```

[53]:
```
print(res.trend)
```

```
Date
2020-08-02             NaN
2020-08-01             NaN
2020-07-31             NaN
2020-07-30    11196.528571
2020-07-29    11028.914286
```

```
                     …
2017-05-05     1531.471429
2017-05-04     1495.942857
2017-05-03             NaN
2017-05-02             NaN
2017-05-01             NaN
Name: Price, Length: 1190, dtype: float64
```

Observation: The initial values are null because internally it does (t-1), (t-2) .. analysis

[54]:
```
print(res.seasonal)
```

```
Date
2020-08-02    0.998044
2020-08-01    1.000973
2020-07-31    0.999294
2020-07-30    0.997878
2020-07-29    1.001348
                 …
2017-05-05    0.999294
2017-05-04    0.997878
2017-05-03    1.001348
2017-05-02    1.000459
2017-05-01    1.002004
Name: Price, Length: 1190, dtype: float64
```

[55]:
```
res.resid
```

[55]:
```
Date
2020-08-02         NaN
2020-08-01         NaN
2020-07-31         NaN
2020-07-30    0.993147
2020-07-29    1.005624
                 …
2017-05-05    0.985239
2017-05-04    1.016098
2017-05-03         NaN
2017-05-02         NaN
2017-05-01         NaN
Name: Price, Length: 1190, dtype: float64
```

[56]:
```
res.observed[4]
```

[56]:
```
11105.9
```

[57]:
```
# Costructing observed value by multiplying trend, seasonal and residula values.
```

```
res.trend[4]*res.seasonal[4]*res.resid[4]
```

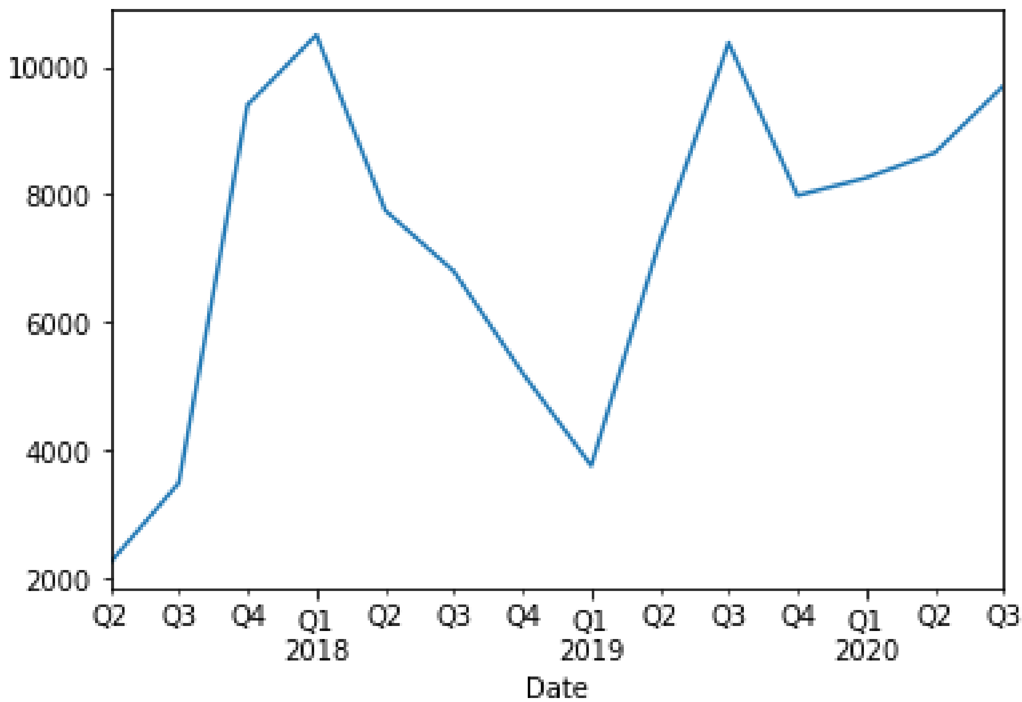
[57]: 11105.9

[58]:
```
# To reduce the complexity of plots, we can resample data on quarterly or␣
↪monthly basis

new_df['Price'].resample('Q').mean()
```

[58]:
```
Date
2017-06-30     2248.745902
2017-09-30     3493.634783
2017-12-31     9403.500000
2018-03-31    10496.793333
2018-06-30     7758.698901
2018-09-30     6803.539130
2018-12-31     5215.860870
2019-03-31     3764.360000
2019-06-30     7283.050549
2019-09-30    10374.306522
2019-12-31     7986.182609
2020-03-31     8263.290110
2020-06-30     8655.456044
2020-09-30     9704.969697
Freq: Q-DEC, Name: Price, dtype: float64
```

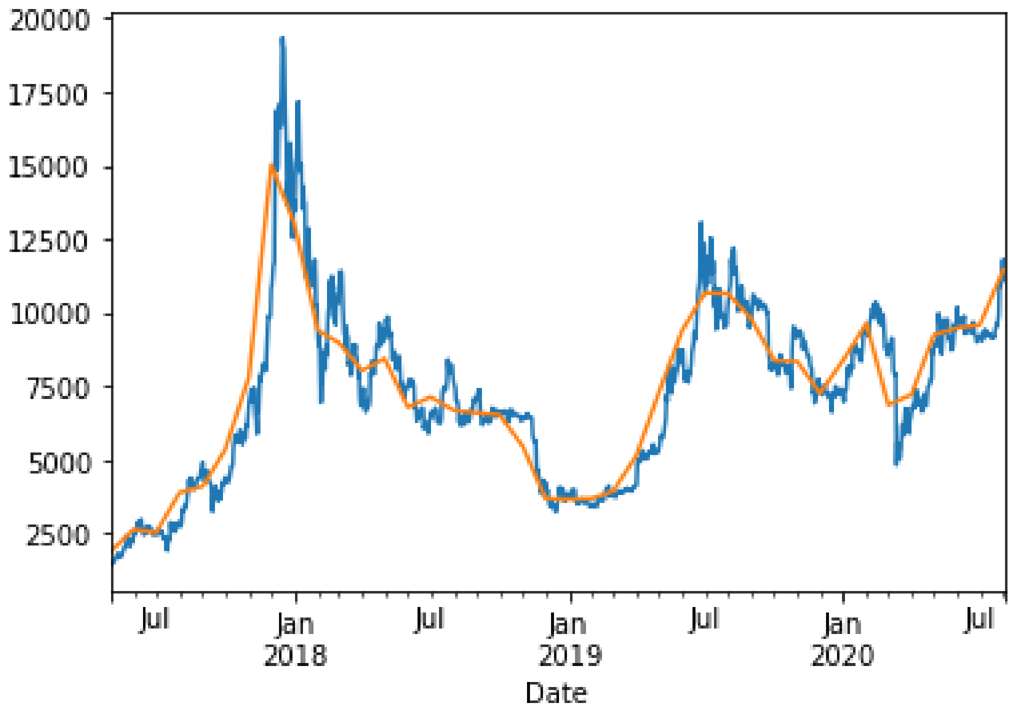
[59]:
```
new_df['Price'].resample('Q').mean().plot()
```

[59]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabae1b2c10>



[60]:
```
new_df['Price'].plot()
new_df['Price'].resample('M').mean().plot()
```

[60]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabae115150>



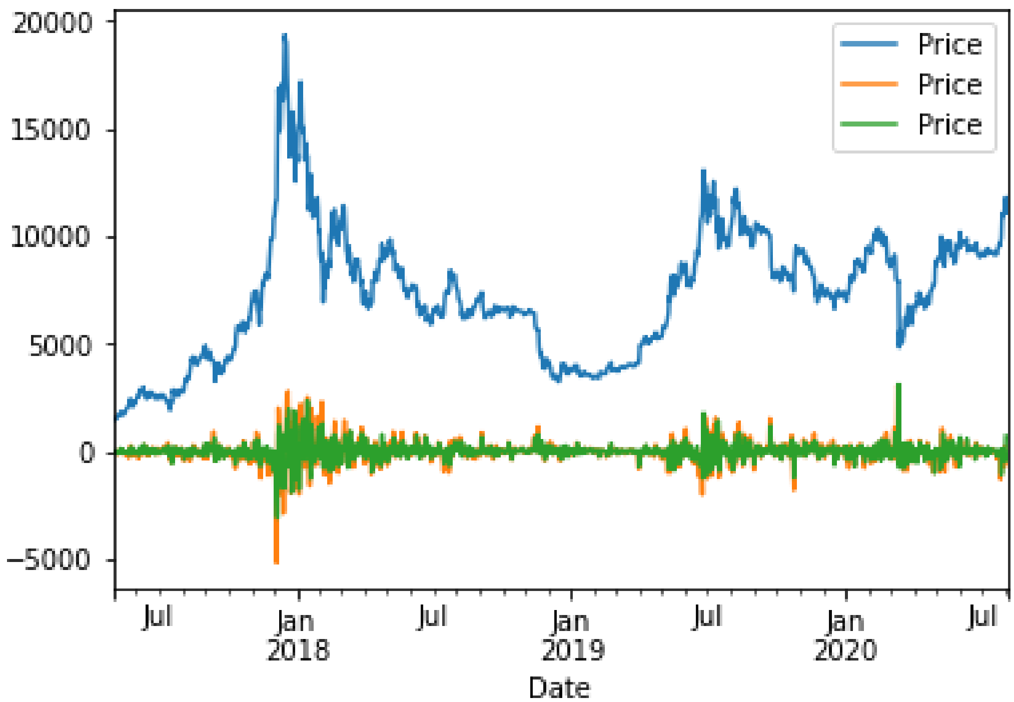
[61]:
```
sta_df=df['Price'].diff()
sta_df.head()
```

[61]:
```
Date
2020-08-02      NaN
2020-08-01    697.3
2020-07-31   -469.7
2020-07-30   -237.2
2020-07-29      9.7
Name: Price, dtype: float64
```

[62]:
```
pt.figure(figsize=(20,10))
pd.concat([new_df['Price'],new_df['Price'].diff(2),new_df['Price'].
 ↪diff()],axis=1).plot()
```

[62]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabae025890>
```

```
<Figure size 1440x720 with 0 Axes>
```



#MODELS

7 1. MOVING AVERAGES

if Data is not stationary then MA might not be the right choice to do forecasting. But if data is stationary then we can use MA. This is not the perfect model but its always good to start with a basic model.

[63]:
```
df_ma = df['2020':'2017'][['Price']]
```

[64]:
```
df_ma
```

[64]:

	Price
Date	
2020-08-02	11105.8
2020-08-01	11803.1
2020-07-31	11333.4
2020-07-30	11096.2
2020-07-29	11105.9
…	…
2017-05-05	1507.8
2017-05-04	1516.8
2017-05-03	1485.6
2017-05-02	1445.9

```
2017-05-01   1415.8

[1190 rows x 1 columns]
```

8 1.1 Simple Moving Average

Simple Moving average calculates an average of the last n data points. Where n represents number of periods for which we want to average.

Simple Moving Average=(t+(t-1)+(t-2)+…+(t-n))/n

[65]:
```
df_ma['Price'].rolling(window=3).mean()
```

[65]:
```
Date
2020-08-02             NaN
2020-08-01             NaN
2020-07-31    11414.100000
2020-07-30    11410.900000
2020-07-29    11178.500000
                  …
2017-05-05     1535.833333
2017-05-04     1523.300000
2017-05-03     1503.400000
2017-05-02     1482.766667
2017-05-01     1449.100000
Name: Price, Length: 1190, dtype: float64
```

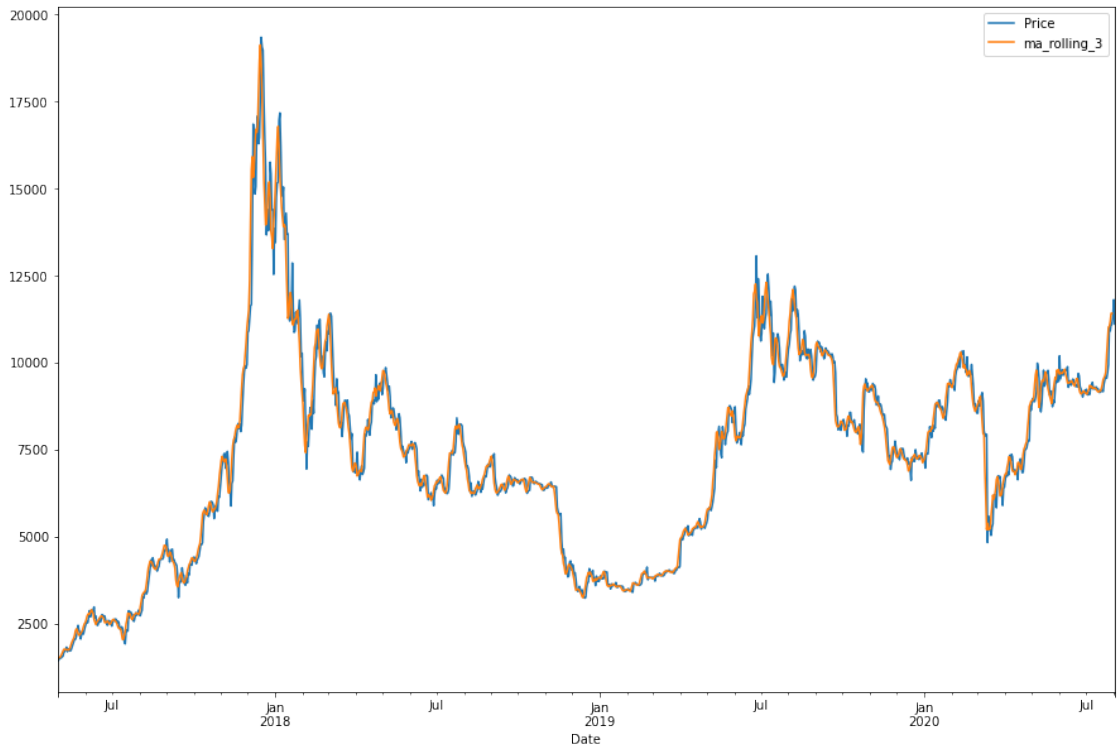
[66]:
```
df_ma['ma_rolling_3']=df_ma['Price'].rolling(window=3).mean().shift(1)
```

[67]:
```
df_ma
```

[67]:
```
              Price  ma_rolling_3
Date
2020-08-02  11105.8           NaN
2020-08-01  11803.1           NaN
2020-07-31  11333.4           NaN
2020-07-30  11096.2  11414.100000
2020-07-29  11105.9  11410.900000
…              …            …
2017-05-05   1507.8   1588.066667
2017-05-04   1516.8   1535.833333
2017-05-03   1485.6   1523.300000
2017-05-02   1445.9   1503.400000
2017-05-01   1415.8   1482.766667

[1190 rows x 2 columns]
```

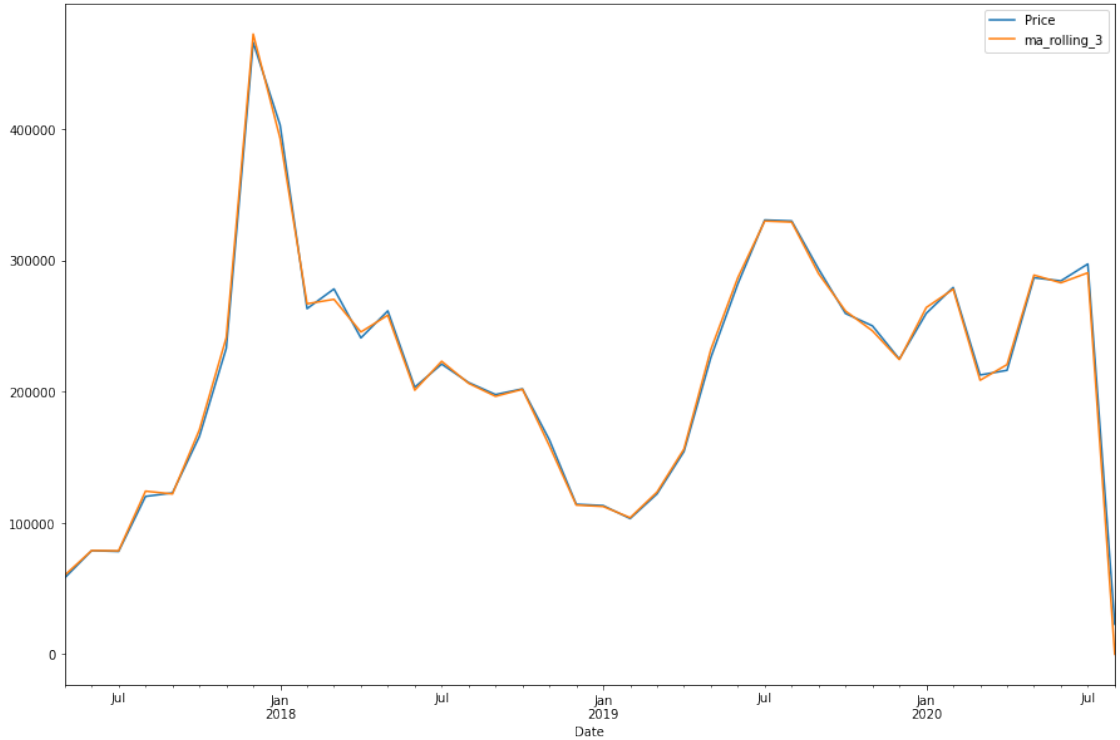
[68]:
```
import matplotlib as mpl
mpl.rcParams['figure.figsize']=(15,10)
mpl.rcParams['axes.grid']=False
df_ma.plot()
```

[68]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabade88e10>



[69]:
```
# Visualize data Monthwise

df_ma.resample('M').sum().plot()
```

[69]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabaddbb290>



9 1.2 Weighted Moving Average

Weighted Moving average gives us weighted average of last n data points where weights are assigned by us. Generally for latest data points we put higher weightage and lower weightage to previous data points. This model is generally more sensitive to data points and it can find trend sooner than Simple Moving Average. The only disadvantage of this model over Simple Moving Average is that we need to assign weights manually.

Weighted Moving Average=(t * (weighted factor))+((t-1) * weighted factor-1)+((t-n) * weighted factor-n)/n

[69]:

```

```

[70]:

```
def wma(weights):
  def result(x):
    return (weights*x).mean()
  return result
```

[71]:

```
df_ma['Price'].rolling(window=3).apply(wma(np.array([0.5,1,1.5])))
```

[71]:

```
Date
2020-08-02          NaN
2020-08-01          NaN
```

```
2020-07-31    11452.033333
2020-07-30    11293.083333
2020-07-29    11140.583333
                  …
2017-05-05     1528.066667
2017-05-04     1518.550000
2017-05-03     1499.700000
2017-05-02     1470.950000
2017-05-01     1437.466667
Name: Price, Length: 1190, dtype: float64
```

[72]:

```
df_ma['wma_rolling_3']=df_ma['Price'].rolling(window=3).apply(wma(np.array([0.
↪5,1,1.5]))).shift(1)
```

[73]:

```
df_ma
```

[73]:

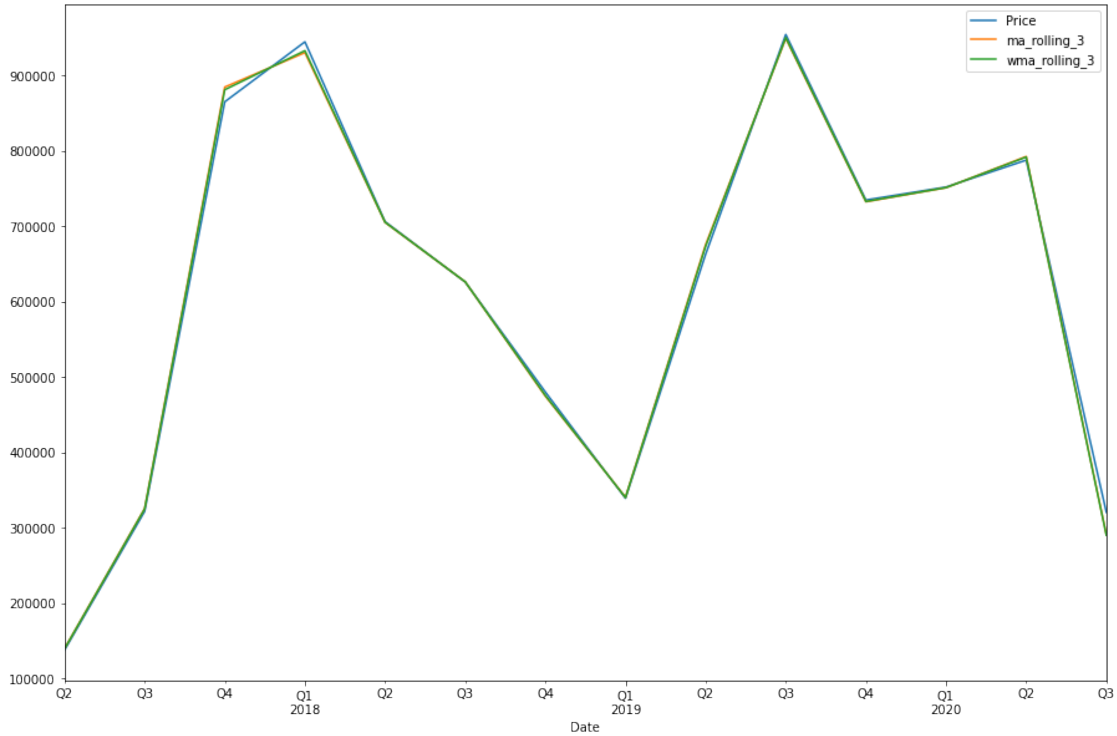
```
              Price  ma_rolling_3  wma_rolling_3
Date
2020-08-02  11105.8           NaN            NaN
2020-08-01  11803.1           NaN            NaN
2020-07-31  11333.4           NaN            NaN
2020-07-30  11096.2  11414.100000   11452.033333
2020-07-29  11105.9  11410.900000   11293.083333
…              …          …              …
2017-05-05   1507.8   1588.066667    1568.200000
2017-05-04   1516.8   1535.833333    1528.066667
2017-05-03   1485.6   1523.300000    1518.550000
2017-05-02   1445.9   1503.400000    1499.700000
2017-05-01   1415.8   1482.766667    1470.950000

[1190 rows x 3 columns]
```

[74]:

```
df_ma.resample('Q').sum().plot()
```

[74]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabadcda150>
```



10 1.3 Exponential Moving Average

Exponential Moving Average is similar to Weighed Average but we dont assign weights here. It takes the previous time period and calculate the exponential moving average and then again take the exponential moving average as the next input instead of time (t-1) and (t-2). The advantage of Exponential Moving Average is it adopts more quickly to data point changes than Simple Moving Average and also it does not need to assign weights manually.

[75]:
```
df_ma['Price'].ewm(span=3,adjust=False,min_periods=0).mean()
```

[75]:
```
Date
2020-08-02    11105.800000
2020-08-01    11454.450000
2020-07-31    11393.925000
2020-07-30    11245.062500
2020-07-29    11175.481250
                  …
2017-05-05     1546.674203
2017-05-04     1531.737102
2017-05-03     1508.668551
2017-05-02     1477.284275
2017-05-01     1446.542138
Name: Price, Length: 1190, dtype: float64
```

[76]:
```
df_ma['ewm_win_3']=df_ma['Price'].ewm(span=3,adjust=False,min_periods=0).mean().
 ↪shift(1)
```

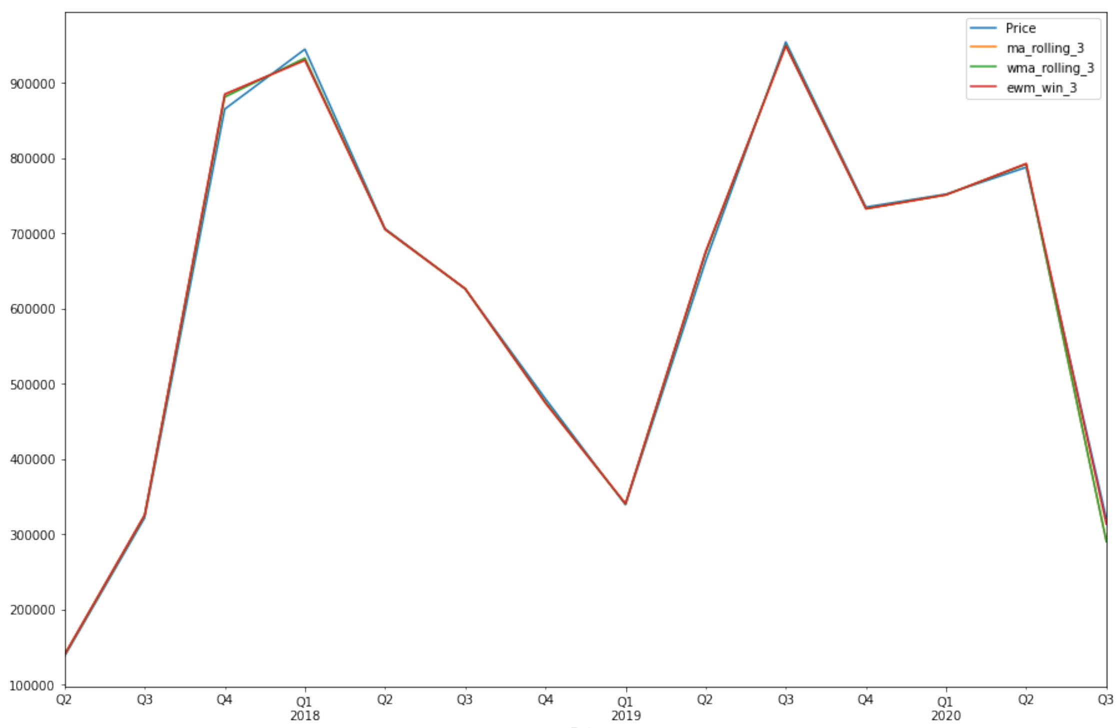
[77]:
```
df_ma
```

[77]:
```
              Price  ma_rolling_3  wma_rolling_3     ewm_win_3
Date
2020-08-02  11105.8           NaN            NaN           NaN
2020-08-01  11803.1           NaN            NaN  11105.800000
2020-07-31  11333.4           NaN            NaN  11454.450000
2020-07-30  11096.2  11414.100000   11452.033333  11393.925000
2020-07-29  11105.9  11410.900000   11293.083333  11245.062500
…               …          …              …           …
2017-05-05   1507.8   1588.066667    1568.200000   1585.548407
2017-05-04   1516.8   1535.833333    1528.066667   1546.674203
2017-05-03   1485.6   1523.300000    1518.550000   1531.737102
2017-05-02   1445.9   1503.400000    1499.700000   1508.668551
2017-05-01   1415.8   1482.766667    1470.950000   1477.284275

[1190 rows x 4 columns]
```

Observation: ??

[78]:
```
df_ma.resample('Q').sum().plot()
```

[78]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabadc15d90>
```



Find RMSE for all above three models:

[79]:
```
# RMSE for Simple Moving Average

((df_ma['Price']-df_ma['ma_rolling_3'])**2).mean()**0.5
```

[79]: 472.45724896106003

[80]:
```
# RMSE % for Simple Moving Average

rmspe = (np.sqrt(np.mean(np.square((df_ma['Price'] - df_ma['ma_rolling_3']) /␣
↪df_ma['Price'])))) * 100
print(rmspe)
```

5.483002449119356

[81]:
```
# RMSE for Weighted Moving Average

((df_ma['Price']-df_ma['wma_rolling_3'])**2).mean()**0.5
```

[81]: 427.9055377789721

[82]:
```
# RMSE % for Weighted Moving Average

rmspe = (np.sqrt(np.mean(np.square((df_ma['Price'] - df_ma['wma_rolling_3']) /␣
↪df_ma['Price'])))) * 100
print(rmspe)
```

4.9635184032905695

[83]:
```
# RMSE for Exponential Moving Average

((df_ma['Price']-df_ma['ewm_win_3'])**2).mean()**0.5
```

[83]: 438.02080086620896

[84]:
```
# RMSE % for Exponential Moving Average

rmspe = (np.sqrt(np.mean(np.square((df_ma['Price'] - df_ma['ewm_win_3']) /␣
↪df_ma['Price'])))) * 100
print(rmspe)
```

5.1209377690520235

#2. LSTM MODEL

LSTM is a Long Short Term Memory Model. It is a class of Recurrent Neural Network. LSTM models remember information for long periods of time and this is practically their behaviour not something they struggle to learn.

[84]:

```

```

[85]:

```
from keras.preprocessing.sequence import TimeseriesGenerator
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import tensorflow as tf
```

[86]:

```
lstm_df_input=new_df[['Price','Open','High']]
```

[87]:

```
lstm_df_input
```

[87]:

```
               Price     Open     High
Date
2020-08-02  11105.8  11802.6  12061.1
2020-08-01  11803.1  11333.2  11847.7
2020-07-31  11333.4  11096.5  11434.8
2020-07-30  11096.2  11105.8  11164.4
2020-07-29  11105.9  10908.4  11336.5
…              …        …        …
2017-05-05   1507.8   1516.8   1588.1
2017-05-04   1516.8   1485.6   1609.8
2017-05-03   1485.6   1445.9   1496.4
2017-05-02   1445.9   1415.8   1471.1
2017-05-01   1415.8   1351.9   1448.7

[1190 rows x 3 columns]
```

[88]:

```
lstm_df_input.describe()
```

[88]:

```
              Price          Open          High
count   1190.000000   1190.000000   1190.000000
mean    7252.800504   7244.515798   7452.483109
std     2992.103176   2994.837939   3110.469739
min     1415.800000   1351.900000   1448.700000
25%     4865.425000   4832.975000   5017.400000
50%     7292.850000   7289.400000   7455.150000
75%     9232.625000   9230.525000   9392.350000
max    19345.500000  19346.600000  19870.600000
```

Here we have large variation in some values. So we have to do some scaling.

[89]:

```
# We need to scale our data before feeding into neural network, we want␣
 ↪gradients to converge faster.
# If we dont scale our data it may take long time to converge.
scaler=MinMaxScaler()
```

```
lstm_data_scaled=scaler.fit_transform(lstm_df_input)
```

[90]:
```
lstm_data_scaled
```

[90]:
```
array([[0.54044407, 0.58076545, 0.57607521],
       [0.57933485, 0.55467999, 0.56449118],
       [0.55313809, 0.54152612, 0.54207764],
       …,
       [0.00389298, 0.00522376, 0.00258931],
       [0.00167878, 0.00355105, 0.00121594],
       [0.        , 0.        , 0.        ]])
```

[91]:
```
lstm_features=lstm_data_scaled
lstm_target=lstm_data_scaled[:,0]
```

[92]:
```
TimeseriesGenerator(lstm_features,lstm_target,length=2,sampling_rate=1,batch_size=1)[0]
```

[92]:
```
(array([[[0.54044407, 0.58076545, 0.57607521],
         [0.57933485, 0.55467999, 0.56449118]]]), array([0.55313809]))
```

[93]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(lstm_features,lstm_target,test_size=0.
 ↪20,random_state=123,shuffle=False)
# x_train,x_val,y_train,y_val=train_test_split(x_train,y_train,test_size=0.15,␣
 ↪random_state=1)
```

[94]:
```
x_train.shape
```

[94]:
```
(952, 3)
```

[95]:
```
x_test.shape
```

[95]:
```
(238, 3)
```

[96]:
```
#Defining actual Timeseries generator that we are going to feed into the model
win_length=10
# win_length=10
batch_size=30
# batch_size=30
num_features=3
train_generator=TimeseriesGenerator(x_train,y_train,length=win_length,sampling_rate=1,batch_si
test_generator=TimeseriesGenerator(x_test,y_test,length=win_length,sampling_rate=1,batch_size=
#␣
 ↪val_generator=TimeseriesGenerator(x_val,y_val,length=win_length,sampling_rate=1,batch_size=
```

[97]:
```
train_generator[0]
```

```
[97]: (array([[[0.54044407, 0.58076545, 0.57607521],
         [0.57933485, 0.55467999, 0.56449118],
         [0.55313809, 0.54152612, 0.54207764],
         [0.53990864, 0.54204293, 0.52739945],
         [0.54044965, 0.53107304, 0.5367416 ],
         [0.52943998, 0.53400168, 0.52097775],
         [0.53581488, 0.47685152, 0.53839723],
         [0.47500516, 0.46336421, 0.468893  ],
         [0.46226652, 0.45540076, 0.44949761],
         [0.45347106, 0.45831828, 0.44390101]],

        [[0.57933485, 0.55467999, 0.56449118],
         [0.55313809, 0.54152612, 0.54207764],
         [0.53990864, 0.54204293, 0.52739945],
         [0.54044965, 0.53107304, 0.5367416 ],
         [0.52943998, 0.53400168, 0.52097775],
         [0.53581488, 0.47685152, 0.53839723],
         [0.47500516, 0.46336421, 0.468893  ],
         [0.46226652, 0.45540076, 0.44949761],
         [0.45347106, 0.45831828, 0.44390101],
         [0.4564382 , 0.45356133, 0.44500296]],

        [[0.55313809, 0.54152612, 0.54207764],
         [0.53990864, 0.54204293, 0.52739945],
         [0.54044965, 0.53107304, 0.5367416 ],
         [0.52943998, 0.53400168, 0.52097775],
         [0.53581488, 0.47685152, 0.53839723],
         [0.47500516, 0.46336421, 0.468893  ],
         [0.46226652, 0.45540076, 0.44949761],
         [0.45347106, 0.45831828, 0.44390101],
         [0.4564382 , 0.45356133, 0.44500296],
         [0.45164727, 0.44654815, 0.43830441]],

        [[0.53990864, 0.54204293, 0.52739945],
         [0.54044965, 0.53107304, 0.5367416 ],
         [0.52943998, 0.53400168, 0.52097775],
         [0.53581488, 0.47685152, 0.53839723],
         [0.47500516, 0.46336421, 0.468893  ],
         [0.46226652, 0.45540076, 0.44949761],
         [0.45347106, 0.45831828, 0.44390101],
         [0.4564382 , 0.45356133, 0.44500296],
         [0.45164727, 0.44654815, 0.43830441],
         [0.44459751, 0.43403891, 0.43308236]],

        [[0.54044965, 0.53107304, 0.5367416 ],
         [0.52943998, 0.53400168, 0.52097775],
         [0.53581488, 0.47685152, 0.53839723],
```

```
 [0.47500516, 0.46336421, 0.468893  ],
 [0.46226652, 0.45540076, 0.44949761],
 [0.45347106, 0.45831828, 0.44390101],
 [0.4564382 , 0.45356133, 0.44500296],
 [0.45164727, 0.44654815, 0.43830441],
 [0.44459751, 0.43403891, 0.43308236],
 [0.43205408, 0.43658411, 0.42176431]],

[[0.52943998, 0.53400168, 0.52097775],
 [0.53581488, 0.47685152, 0.53839723],
 [0.47500516, 0.46336421, 0.468893  ],
 [0.46226652, 0.45540076, 0.44949761],
 [0.45347106, 0.45831828, 0.44390101],
 [0.4564382 , 0.45356133, 0.44500296],
 [0.45164727, 0.44654815, 0.43830441],
 [0.44459751, 0.43403891, 0.43308236],
 [0.43205408, 0.43658411, 0.42176431],
 [0.43459734, 0.43447237, 0.42211173]],

[[0.53581488, 0.47685152, 0.53839723],
 [0.47500516, 0.46336421, 0.468893  ],
 [0.46226652, 0.45540076, 0.44949761],
 [0.45347106, 0.45831828, 0.44390101],
 [0.4564382 , 0.45356133, 0.44500296],
 [0.45164727, 0.44654815, 0.43830441],
 [0.44459751, 0.43403891, 0.43308236],
 [0.43205408, 0.43658411, 0.42176431],
 [0.43459734, 0.43447237, 0.42211173],
 [0.43248911, 0.43367769, 0.42070036]],

[[0.47500516, 0.46336421, 0.468893  ],
 [0.46226652, 0.45540076, 0.44949761],
 [0.45347106, 0.45831828, 0.44390101],
 [0.4564382 , 0.45356133, 0.44500296],
 [0.45164727, 0.44654815, 0.43830441],
 [0.44459751, 0.43403891, 0.43308236],
 [0.43205408, 0.43658411, 0.42176431],
 [0.43459734, 0.43447237, 0.42211173],
 [0.43248911, 0.43367769, 0.42070036],
 [0.43168597, 0.43254403, 0.41978298]],

[[0.46226652, 0.45540076, 0.44949761],
 [0.45347106, 0.45831828, 0.44390101],
 [0.4564382 , 0.45356133, 0.44500296],
 [0.45164727, 0.44654815, 0.43830441],
 [0.44459751, 0.43403891, 0.43308236],
 [0.43205408, 0.43658411, 0.42176431],
```

```
 [0.43459734, 0.43447237, 0.42211173],
 [0.43248911, 0.43367769, 0.42070036],
 [0.43168597, 0.43254403, 0.41978298],
 [0.43054262, 0.43606173, 0.42207916]],

[[0.45347106, 0.45831828, 0.44390101],
 [0.4564382 , 0.45356133, 0.44500296],
 [0.45164727, 0.44654815, 0.43830441],
 [0.44459751, 0.43403891, 0.43308236],
 [0.43205408, 0.43658411, 0.42176431],
 [0.43459734, 0.43447237, 0.42211173],
 [0.43248911, 0.43367769, 0.42070036],
 [0.43168597, 0.43254403, 0.41978298],
 [0.43054262, 0.43606173, 0.42207916],
 [0.43407865, 0.43911263, 0.42457618]],

[[0.4564382 , 0.45356133, 0.44500296],
 [0.45164727, 0.44654815, 0.43830441],
 [0.44459751, 0.43403891, 0.43308236],
 [0.43205408, 0.43658411, 0.42176431],
 [0.43459734, 0.43447237, 0.42211173],
 [0.43248911, 0.43367769, 0.42070036],
 [0.43168597, 0.43254403, 0.41978298],
 [0.43054262, 0.43606173, 0.42207916],
 [0.43407865, 0.43911263, 0.42457618],
 [0.43712946, 0.43852912, 0.42482589]],

[[0.45164727, 0.44654815, 0.43830441],
 [0.44459751, 0.43403891, 0.43308236],
 [0.43205408, 0.43658411, 0.42176431],
 [0.43459734, 0.43447237, 0.42211173],
 [0.43248911, 0.43367769, 0.42070036],
 [0.43168597, 0.43254403, 0.41978298],
 [0.43054262, 0.43606173, 0.42207916],
 [0.43407865, 0.43911263, 0.42457618],
 [0.43712946, 0.43852912, 0.42482589],
 [0.43658288, 0.44171895, 0.42780061]],

[[0.44459751, 0.43403891, 0.43308236],
 [0.43205408, 0.43658411, 0.42176431],
 [0.43459734, 0.43447237, 0.42211173],
 [0.43248911, 0.43367769, 0.42070036],
 [0.43168597, 0.43254403, 0.41978298],
 [0.43054262, 0.43606173, 0.42207916],
 [0.43407865, 0.43911263, 0.42457618],
 [0.43712946, 0.43852912, 0.42482589],
 [0.43658288, 0.44171895, 0.42780061],
```

```
 [0.43977311, 0.43803453, 0.42824573]],

[[0.43205408, 0.43658411, 0.42176431],
 [0.43459734, 0.43447237, 0.42211173],
 [0.43248911, 0.43367769, 0.42070036],
 [0.43168597, 0.43254403, 0.41978298],
 [0.43054262, 0.43606173, 0.42207916],
 [0.43407865, 0.43911263, 0.42457618],
 [0.43712946, 0.43852912, 0.42482589],
 [0.43658288, 0.44171895, 0.42780061],
 [0.43977311, 0.43803453, 0.42824573],
 [0.43600841, 0.44090204, 0.42594955]],

[[0.43459734, 0.43447237, 0.42211173],
 [0.43248911, 0.43367769, 0.42070036],
 [0.43168597, 0.43254403, 0.41978298],
 [0.43054262, 0.43606173, 0.42207916],
 [0.43407865, 0.43911263, 0.42457618],
 [0.43712946, 0.43852912, 0.42482589],
 [0.43658288, 0.44171895, 0.42780061],
 [0.43977311, 0.43803453, 0.42824573],
 [0.43600841, 0.44090204, 0.42594955],
 [0.43889747, 0.43830128, 0.4267258 ]],

[[0.43248911, 0.43367769, 0.42070036],
 [0.43168597, 0.43254403, 0.41978298],
 [0.43054262, 0.43606173, 0.42207916],
 [0.43407865, 0.43911263, 0.42457618],
 [0.43712946, 0.43852912, 0.42482589],
 [0.43658288, 0.44171895, 0.42780061],
 [0.43977311, 0.43803453, 0.42824573],
 [0.43600841, 0.44090204, 0.42594955],
 [0.43889747, 0.43830128, 0.4267258 ],
 [0.43614227, 0.44892107, 0.43349492]],

[[0.43168597, 0.43254403, 0.41978298],
 [0.43054262, 0.43606173, 0.42207916],
 [0.43407865, 0.43911263, 0.42457618],
 [0.43712946, 0.43852912, 0.42482589],
 [0.43658288, 0.44171895, 0.42780061],
 [0.43977311, 0.43803453, 0.42824573],
 [0.43600841, 0.44090204, 0.42594955],
 [0.43889747, 0.43830128, 0.4267258 ],
 [0.43614227, 0.44892107, 0.43349492],
 [0.44697346, 0.43926267, 0.43478686]],

[[0.43054262, 0.43606173, 0.42207916],
```

```
 [0.43407865, 0.43911263, 0.42457618],
 [0.43712946, 0.43852912, 0.42482589],
 [0.43658288, 0.44171895, 0.42780061],
 [0.43977311, 0.43803453, 0.42824573],
 [0.43600841, 0.44090204, 0.42594955],
 [0.43889747, 0.43830128, 0.4267258 ],
 [0.43614227, 0.44892107, 0.43349492],
 [0.44697346, 0.43926267, 0.43478686],
 [0.43727447, 0.44384736, 0.43005879]],

[[0.43407865, 0.43911263, 0.42457618],
 [0.43712946, 0.43852912, 0.42482589],
 [0.43658288, 0.44171895, 0.42780061],
 [0.43977311, 0.43803453, 0.42824573],
 [0.43600841, 0.44090204, 0.42594955],
 [0.43889747, 0.43830128, 0.4267258 ],
 [0.43614227, 0.44892107, 0.43349492],
 [0.44697346, 0.43926267, 0.43478686],
 [0.43727447, 0.44384736, 0.43005879],
 [0.44190366, 0.42948757, 0.42961367]],

[[0.43712946, 0.43852912, 0.42482589],
 [0.43658288, 0.44171895, 0.42780061],
 [0.43977311, 0.43803453, 0.42824573],
 [0.43600841, 0.44090204, 0.42594955],
 [0.43889747, 0.43830128, 0.4267258 ],
 [0.43614227, 0.44892107, 0.43349492],
 [0.44697346, 0.43926267, 0.43478686],
 [0.43727447, 0.44384736, 0.43005879],
 [0.44190366, 0.42948757, 0.42961367],
 [0.42751412, 0.4324829 , 0.41802963]],

[[0.43658288, 0.44171895, 0.42780061],
 [0.43977311, 0.43803453, 0.42824573],
 [0.43600841, 0.44090204, 0.42594955],
 [0.43889747, 0.43830128, 0.4267258 ],
 [0.43614227, 0.44892107, 0.43349492],
 [0.44697346, 0.43926267, 0.43478686],
 [0.43727447, 0.44384736, 0.43005879],
 [0.44190366, 0.42948757, 0.42961367],
 [0.42751412, 0.4324829 , 0.41802963],
 [0.43049242, 0.42874846, 0.41980469]],

[[0.43977311, 0.43803453, 0.42824573],
 [0.43600841, 0.44090204, 0.42594955],
 [0.43889747, 0.43830128, 0.4267258 ],
 [0.43614227, 0.44892107, 0.43349492],
```

```
 [0.44697346, 0.43926267, 0.43478686],
 [0.43727447, 0.44384736, 0.43005879],
 [0.44190366, 0.42948757, 0.42961367],
 [0.42751412, 0.4324829 , 0.41802963],
 [0.43049242, 0.42874846, 0.41980469],
 [0.42673887, 0.4297432 , 0.41644456]],

[[0.43600841, 0.44090204, 0.42594955],
 [0.43889747, 0.43830128, 0.4267258 ],
 [0.43614227, 0.44892107, 0.43349492],
 [0.44697346, 0.43926267, 0.43478686],
 [0.43727447, 0.44384736, 0.43005879],
 [0.44190366, 0.42948757, 0.42961367],
 [0.42751412, 0.4324829 , 0.41802963],
 [0.43049242, 0.42874846, 0.41980469],
 [0.42673887, 0.4297432 , 0.41644456],
 [0.4277428 , 0.43779557, 0.42400621]],

[[0.43889747, 0.43830128, 0.4267258 ],
 [0.43614227, 0.44892107, 0.43349492],
 [0.44697346, 0.43926267, 0.43478686],
 [0.43727447, 0.44384736, 0.43005879],
 [0.44190366, 0.42948757, 0.42961367],
 [0.42751412, 0.4324829 , 0.41802963],
 [0.43049242, 0.42874846, 0.41980469],
 [0.42673887, 0.4297432 , 0.41644456],
 [0.4277428 , 0.43779557, 0.42400621],
 [0.43581878, 0.43257181, 0.42559671]],

[[0.43614227, 0.44892107, 0.43349492],
 [0.44697346, 0.43926267, 0.43478686],
 [0.43727447, 0.44384736, 0.43005879],
 [0.44190366, 0.42948757, 0.42961367],
 [0.42751412, 0.4324829 , 0.41802963],
 [0.43049242, 0.42874846, 0.41980469],
 [0.42673887, 0.4297432 , 0.41644456],
 [0.4277428 , 0.43779557, 0.42400621],
 [0.43581878, 0.43257181, 0.42559671],
 [0.4305482 , 0.43535597, 0.42075465]],

[[0.44697346, 0.43926267, 0.43478686],
 [0.43727447, 0.44384736, 0.43005879],
 [0.44190366, 0.42948757, 0.42961367],
 [0.42751412, 0.4324829 , 0.41802963],
 [0.43049242, 0.42874846, 0.41980469],
 [0.42673887, 0.4297432 , 0.41644456],
 [0.4277428 , 0.43779557, 0.42400621],
```

```
 [0.43581878, 0.43257181, 0.42559671],
 [0.4305482 , 0.43535597, 0.42075465],
 [0.43333687, 0.43184382, 0.42234514]],

[[0.43727447, 0.44384736, 0.43005879],
 [0.44190366, 0.42948757, 0.42961367],
 [0.42751412, 0.4324829 , 0.41802963],
 [0.43049242, 0.42874846, 0.41980469],
 [0.42673887, 0.4297432 , 0.41644456],
 [0.4277428 , 0.43779557, 0.42400621],
 [0.43581878, 0.43257181, 0.42559671],
 [0.4305482 , 0.43535597, 0.42075465],
 [0.43333687, 0.43184382, 0.42234514],
 [0.42991238, 0.42548639, 0.4199784 ]],

[[0.44190366, 0.42948757, 0.42961367],
 [0.42751412, 0.4324829 , 0.41802963],
 [0.43049242, 0.42874846, 0.41980469],
 [0.42673887, 0.4297432 , 0.41644456],
 [0.4277428 , 0.43779557, 0.42400621],
 [0.43581878, 0.43257181, 0.42559671],
 [0.4305482 , 0.43535597, 0.42075465],
 [0.43333687, 0.43184382, 0.42234514],
 [0.42991238, 0.42548639, 0.4199784 ],
 [0.4234594 , 0.43393888, 0.42024981]],

[[0.42751412, 0.4324829 , 0.41802963],
 [0.43049242, 0.42874846, 0.41980469],
 [0.42673887, 0.4297432 , 0.41644456],
 [0.4277428 , 0.43779557, 0.42400621],
 [0.43581878, 0.43257181, 0.42559671],
 [0.4305482 , 0.43535597, 0.42075465],
 [0.43333687, 0.43184382, 0.42234514],
 [0.42991238, 0.42548639, 0.4199784 ],
 [0.4234594 , 0.43393888, 0.42024981],
 [0.43192022, 0.43877364, 0.42600383]],

[[0.43049242, 0.42874846, 0.41980469],
 [0.42673887, 0.4297432 , 0.41644456],
 [0.4277428 , 0.43779557, 0.42400621],
 [0.43581878, 0.43257181, 0.42559671],
 [0.4305482 , 0.43535597, 0.42075465],
 [0.43333687, 0.43184382, 0.42234514],
 [0.42991238, 0.42548639, 0.4199784 ],
 [0.4234594 , 0.43393888, 0.42024981],
 [0.43192022, 0.43877364, 0.42600383],
 [0.43680039, 0.44173562, 0.42767576]]]),
```

```
array([0.4564382 , 0.45164727, 0.44459751, 0.43205408, 0.43459734,
       0.43248911, 0.43168597, 0.43054262, 0.43407865, 0.43712946,
       0.43658288, 0.43977311, 0.43600841, 0.43889747, 0.43614227,
       0.44697346, 0.43727447, 0.44190366, 0.42751412, 0.43049242,
       0.42673887, 0.4277428 , 0.43581878, 0.4305482 , 0.43333687,
       0.42991238, 0.4234594 , 0.43192022, 0.43680039, 0.43984004]))
```

[98]:

```
#Define LSTM Model

model=tf.keras.Sequential()
# model.add(tf.keras.layers.
 ↪LSTM(128,input_shape=(win_length,num_features),return_sequences=True))
# model.add(tf.keras.layers.
 ↪LSTM(256,input_shape=(win_length,num_features),return_sequences=True))
model.add(tf.keras.layers.
 ↪LSTM(64,input_shape=(win_length,num_features),return_sequences=True))

model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
# model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
# model.add(tf.keras.layers.LeakyReLU(alpha=0.5))

# model.add(tf.keras.layers.LSTM(128, return_sequences=True))
# model.add(tf.keras.layers.LSTM(256, return_sequences=True))
model.add(tf.keras.layers.LSTM(64, return_sequences=True))

model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
# model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
# model.add(tf.keras.layers.LeakyReLU(alpha=0.5))

model.add(tf.keras.layers.Dropout(0.3))
# model.add(tf.keras.layers.Dropout(0.3))
# model.add(tf.keras.layers.Dropout(0.3))

#model.add(tf.keras.layers.LSTM(128, return_sequences=True))
# model.add(tf.keras.layers.LSTM(128, return_sequences=True))
# model.add(tf.keras.layers.LSTM(128, return_sequences=True))

#model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
# model.add(tf.keras.layers.LeakyReLU(alpha=0.5))
# model.add(tf.keras.layers.LeakyReLU(alpha=0.5))

#model.add(tf.keras.layers.Dropout(0.3))
# model.add(tf.keras.layers.Dropout(0.3))
# model.add(tf.keras.layers.Dropout(0.3))

#model.add(tf.keras.layers.LSTM(64,return_sequences=False))
model.add(tf.keras.layers.LSTM(32,return_sequences=False))
```

```
model.add(tf.keras.layers.Dropout(0.3))
# model.add(tf.keras.layers.Dropout(0.3))
# model.add(tf.keras.layers.Dropout(0.3))

model.add(tf.keras.layers.Dense(1))
```

[99]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 10, 64)            17408

 leaky_re_lu (LeakyReLU)     (None, 10, 64)            0

 lstm_1 (LSTM)               (None, 10, 64)            33024

 leaky_re_lu_1 (LeakyReLU)   (None, 10, 64)            0

 dropout (Dropout)           (None, 10, 64)            0

 lstm_2 (LSTM)               (None, 32)                12416

 dropout_1 (Dropout)         (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 62,881
Trainable params: 62,881
Non-trainable params: 0
_________________________________________________________________
```

[100]:
```
#!pip install keras==2.6.0
```

[101]:
```
#!pip install tensorflow==2.6.0
```

[102]:
```
#pip install git+git://github.com/keras-team/keras.git --upgrade --no-deps
```

[103]:
```
# Defining Early stopping because we dont want to run forever. So if validation␣
↪los doesn't improve we can stop the training.

early_stopping=tf.keras.callbacks.
↪EarlyStopping(monitor='val_loss',patience=2,mode='min')
```

```
# early_stopping=tf.keras.callbacks.
↪EarlyStopping(monitor='val_loss',patience=2,mode='min')
model.compile(loss=tf.losses.MeanSquaredError(),optimizer=tf.optimizers.
↪Adam(),metrics=[tf.metrics.MeanAbsoluteError()])
# model.compile(loss=tf.losses.MeanSquaredError(),optimizer=tf.optimizers.
↪Adam(),metrics=[tf.metrics.MeanAbsoluteError()])
# history=model.fit_generator(train_generator,␣
↪epochs=20,validation_data=test_generator,shuffle=False,callbacks=[early_stopping])
# history=model.fit_generator(train_generator,␣
↪epochs=20,validation_data=test_generator,shuffle=False,callbacks=[early_stopping])
# history=model.fit_generator(train_generator,␣
↪epochs=20,validation_data=val_generator,shuffle=False,callbacks=[early_stopping])
# history=model.fit(x_train, y_train,␣
↪epochs=20,validation_data=valid_data,shuffle=False,callbacks=[early_stopping])
history=model.fit_generator(train_generator,␣
↪epochs=20,validation_data=test_generator,shuffle=False,callbacks=[early_stopping])
```

```
Epoch 1/20

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: UserWarning:

`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.


32/32 [==============================] - 7s 62ms/step - loss: 0.0375 -
mean_absolute_error: 0.1507 - val_loss: 0.0098 - val_mean_absolute_error: 0.0843
Epoch 2/20
32/32 [==============================] - 1s 21ms/step - loss: 0.0131 -
mean_absolute_error: 0.0909 - val_loss: 0.0090 - val_mean_absolute_error: 0.0825
Epoch 3/20
32/32 [==============================] - 1s 21ms/step - loss: 0.0112 -
mean_absolute_error: 0.0841 - val_loss: 0.0076 - val_mean_absolute_error: 0.0778
Epoch 4/20
32/32 [==============================] - 1s 20ms/step - loss: 0.0105 -
mean_absolute_error: 0.0804 - val_loss: 0.0068 - val_mean_absolute_error: 0.0752
Epoch 5/20
32/32 [==============================] - 1s 20ms/step - loss: 0.0094 -
mean_absolute_error: 0.0750 - val_loss: 0.0066 - val_mean_absolute_error: 0.0750
Epoch 6/20
32/32 [==============================] - 1s 20ms/step - loss: 0.0085 -
mean_absolute_error: 0.0714 - val_loss: 0.0057 - val_mean_absolute_error: 0.0696
Epoch 7/20
32/32 [==============================] - 1s 21ms/step - loss: 0.0071 -
mean_absolute_error: 0.0656 - val_loss: 0.0059 - val_mean_absolute_error: 0.0717
Epoch 8/20
32/32 [==============================] - 1s 21ms/step - loss: 0.0065 -
mean_absolute_error: 0.0627 - val_loss: 0.0051 - val_mean_absolute_error: 0.0651
```

```
Epoch 9/20
32/32 [==============================] - 1s 20ms/step - loss: 0.0058 -
mean_absolute_error: 0.0596 - val_loss: 0.0054 - val_mean_absolute_error: 0.0666
Epoch 10/20
32/32 [==============================] - 1s 23ms/step - loss: 0.0057 -
mean_absolute_error: 0.0586 - val_loss: 0.0056 - val_mean_absolute_error: 0.0672
```

[104]:
```
# model.evaluate_generator(val_generator,verbose=0)
model.evaluate_generator(test_generator,verbose=0)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: UserWarning:

`Model.evaluate_generator` is deprecated and will be removed in a future
version. Please use `Model.evaluate`, which supports generators.
```

[104]:
```
[0.005559174809604883, 0.0672387182712555]
```

[105]:
```
predictions=model.predict_generator(test_generator)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning:

`Model.predict_generator` is deprecated and will be removed in a future version.
Please use `Model.predict`, which supports generators.
```

[106]:
```
# predictions.shape[0]
```

[107]:
```
# predictions
```

[108]:
```
# y_test
```

[109]:
```
# x_test
```

[110]:
```
# x_test[:,1:][win_length:]
```

[111]:
```
df_pred=pd.concat([pd.DataFrame(predictions), pd.DataFrame(x_test[:,1:
 ↪][win_length:])],axis=1)
```

[112]:
```
rev_trans=scaler.inverse_transform(df_pred)
```

[113]:
```
rev_trans
```

[113]:
```
array([[15653.10125206, 16286.8       , 16941.1       ],
       [16028.59095272, 17083.9       , 17268.        ],
       [16317.76869968, 16733.3       , 17560.7       ],
       [16499.27131186, 15060.5       , 17399.2       ],
       [16518.95130081, 14840.       , 15783.2       ],
```

```
[16411.19387665, 16048.2       , 16313.2       ],
[16167.80109232, 16868.        , 17294.8       ],
[15882.20560564, 13750.1       , 16879.3       ],
[15573.26023688, 11667.1       , 13843.2       ],
[15269.43177212, 11624.4       , 11901.9       ],
[14988.36327068, 11244.2       , 11624.6       ],
[14661.94155876, 10912.7       , 11851.1       ],
[14213.78925648, 10861.5       , 11175.2       ],
[13674.920915  ,  9947.1       , 10942.8       ],
[13138.71575746,  9848.        , 10689.1       ],
[12636.83703191,  9906.        , 11417.8       ],
[12060.42121653,  9732.6       ,  9969.6       ],
[11401.51402634,  9318.4       ,  9733.6       ],
[10846.92199886,  8754.6       ,  9474.6       ],
[10517.32519885,  8203.5       ,  8762.        ],
[10267.37918659,  8013.4       ,  8332.9       ],
[10000.32606474,  8234.5       ,  8266.5       ],
[ 9716.53399818,  8099.9       ,  8304.4       ],
[ 9441.32941743,  8245.9       ,  8368.4       ],
[ 9192.68079895,  8042.6       ,  8294.1       ],
[ 8974.94467141,  7781.        ,  8100.9       ],
[ 8740.69936823,  7700.        ,  7857.5       ],
[ 8551.29804313,  7853.7       ,  8000.2       ],
[ 8389.94991976,  7283.        ,  7964.6       ],
[ 8265.14950761,  6597.1       ,  7330.1       ],
[ 8175.40330766,  6522.5       ,  6728.6       ],
[ 8086.12893585,  5878.1       ,  6760.1       ],
[ 7959.78907086,  6339.9       ,  6490.5       ],
[ 7796.60653528,  6565.8       ,  6821.5       ],
[ 7612.73195809,  7129.6       ,  7319.        ],
[ 7436.73418561,  7444.4       ,  7457.1       ],
[ 7301.21958779,  7102.2       ,  7869.1       ],
[ 7211.78598065,  6959.3       ,  7222.4       ],
[ 7142.3839626 ,  7389.5       ,  7429.7       ],
[ 7113.05580964,  7363.8       ,  7599.4       ],
[ 7155.85751463,  7147.        ,  7492.2       ],
[ 7241.88573025,  7024.8       ,  7445.6       ],
[ 7344.76670642,  6737.8       ,  7339.9       ],
[ 7441.8361279 ,  6449.1       ,  6738.7       ],
[ 7486.71483851,  6124.3       ,  6467.2       ],
[ 7466.6474482 ,  6147.7       ,  6226.2       ],
[ 7391.19609116,  5726.6       ,  6295.4       ],
[ 7282.87012212,  5764.6       ,  5871.        ],
[ 7178.52557033,  5887.6       ,  5997.8       ],
[ 7040.36435331,  5734.        ,  5970.4       ],
[ 6878.92752839,  5513.1       ,  5748.        ],
[ 6722.9661541 ,  5903.6       ,  5904.6       ],
```

```
[ 6578.91376471,  5982.9        ,  6049.         ],
[ 6459.58185669,  6006.6        ,  6070.6        ],
[ 6384.45751983,  5993.1        ,  6187.2        ],
[ 6345.53730927,  5698.6        ,  6075.3        ],
[ 6327.72432768,  5576.7        ,  5737.4        ],
[ 6320.35034323,  5598.6        ,  5601.3        ],
[ 6325.68045156,  5759.3        ,  5773.3        ],
[ 6316.01999757,  5677.4        ,  5795.3        ],
[ 6304.0987227 ,  5824.7        ,  5862.7        ],
[ 6305.50298583,  5637.3        ,  5839.6        ],
[ 6294.46819213,  5432.6        ,  5852.8        ],
[ 6265.46438762,  4824.2        ,  5432.6        ],
[ 6224.64350584,  4763.4        ,  4874.9        ],
[ 6162.84898175,  4777.5        ,  4930.         ],
[ 6093.0429976 ,  4611.7        ,  4875.4        ],
[ 6022.56427094,  4435.8        ,  4622.9        ],
[ 5936.74124446,  4371.9        ,  4472.9        ],
[ 5822.70791832,  4321.4        ,  4422.1        ],
[ 5693.56647358,  4218.7        ,  4365.8        ],
[ 5552.224825  ,  4314.2        ,  4355.3        ],
[ 5413.0718605 ,  4401.3        ,  4436.         ],
[ 5289.33960741,  4403.1        ,  4477.4        ],
[ 5207.06464301,  4360.6        ,  4412.6        ],
[ 5160.34083324,  4172.8        ,  4383.3        ],
[ 5115.9355538 ,  4195.6        ,  4237.5        ],
[ 5078.30365313,  4212.2        ,  4274.6        ],
[ 5055.65563544,  3892.7        ,  4232.4        ],
[ 5038.79138642,  3932.8        ,  3982.3        ],
[ 5018.01945999,  3667.5        ,  3971.5        ],
[ 4992.2615445 ,  3788.         ,  3790.9        ],
[ 4954.58101828,  3600.8        ,  3813.4        ],
[ 4899.9373889 ,  3617.3        ,  3753.5        ],
[ 4834.50049053,  3882.2        ,  3912.8        ],
[ 4769.91293624,  3908.         ,  4053.         ],
[ 4717.2509316 ,  4100.3        ,  4120.7        ],
[ 4680.05666087,  3689.6        ,  4122.8        ],
[ 4649.81343919,  3698.9        ,  3796.6        ],
[ 4633.84689263,  3713.8        ,  3893.7        ],
[ 4630.67447627,  3243.1        ,  3824.4        ],
[ 4629.59001964,  3870.3        ,  3930.7        ],
[ 4629.14490884,  4158.9        ,  4174.6        ],
[ 4631.80862714,  4217.9        ,  4387.8        ],
[ 4647.54593897,  4245.9        ,  4364.4        ],
[ 4662.60543201,  4335.1        ,  4338.1        ],
[ 4680.63108357,  4326.5        ,  4402.1        ],
[ 4704.47363332,  4635.6        ,  4699.6        ],
[ 4750.60111217,  4618.7        ,  4690.5        ],
```

```
[ 4826.21010149,  4409.1       ,  4660.       ],
[ 4905.42272494,  4267.5       ,  4496.7      ],
[ 4997.84546751,  4612.9       ,  4621.       ],
[ 5074.5124633 ,  4573.8       ,  4719.8      ],
[ 5118.71950011,  4921.9       ,  4976.5      ],
[ 5152.80120131,  4735.1       ,  4925.2      ],
[ 5191.7422514 ,  4583.       ,  4765.1      ],
[ 5230.92055142,  4597.3       ,  4644.1      ],
[ 5265.31537979,  4390.3       ,  4647.8      ],
[ 5279.25595084,  4345.8       ,  4403.1      ],
[ 5279.36949951,  4352.3       ,  4408.2      ],
[ 5278.36519489,  4364.4       ,  4379.3      ],
[ 5281.05162292,  4318.4       ,  4461.7      ],
[ 5265.04446601,  4141.1       ,  4364.1      ],
[ 5232.44504256,  4089.7       ,  4255.6      ],
[ 5178.35472918,  4005.1       ,  4142.7      ],
[ 5118.33583918,  4066.6       ,  4097.3      ],
[ 5067.49328505,  4150.5       ,  4182.3      ],
[ 5020.66688071,  4105.4       ,  4189.7      ],
[ 4979.60126815,  4278.9       ,  4362.7      ],
[ 4953.68010974,  4387.4       ,  4487.5      ],
[ 4934.90797583,  4161.7       ,  4398.1      ],
[ 4923.29582053,  4327.9       ,  4436.5      ],
[ 4917.15136783,  4062.6       ,  4336.7      ],
[ 4923.36394973,  3871.6       ,  4189.4      ],
[ 4934.77225177,  3654.4       ,  3967.3      ],
[ 4942.110435  ,  3425.7       ,  3706.5      ],
[ 4933.46630844,  3348.8       ,  3453.8      ],
[ 4898.01187057,  3429.4       ,  3437.1      ],
[ 4842.85286379,  3401.9       ,  3494.9      ],
[ 4766.2371653 ,  3232.       ,  3425.1      ],
[ 4671.61665475,  3262.8       ,  3295.1      ],
[ 4579.0694094 ,  2878.5       ,  3344.       ],
[ 4483.46302919,  2810.       ,  2892.7      ],
[ 4390.80784581,  2720.5       ,  2822.9      ],
[ 4308.875932  ,  2747.       ,  2773.8      ],
[ 4236.65790684,  2883.3       ,  2946.       ],
[ 4176.3937517 ,  2766.5       ,  2916.3      ],
[ 4125.0435682 ,  2733.5       ,  2773.1      ],
[ 4071.23164946,  2806.8       ,  2812.1      ],
[ 4015.92703321,  2691.9       ,  2843.8      ],
[ 3970.21180284,  2559.2       ,  2712.9      ],
[ 3931.86013716,  2582.6       ,  2631.7      ],
[ 3902.0176755 ,  2763.4       ,  2779.1      ],
[ 3890.28796392,  2756.6       ,  2798.9      ],
[ 3883.49588302,  2836.5       ,  2856.7      ],
[ 3879.95263005,  2675.1       ,  2876.7      ],
```

```
[ 3871.13671103,  2866.        ,  2874.        ],
[ 3866.22526334,  2282.6       ,  2932.8       ],
[ 3867.2431938 ,  2320.2       ,  2412.4       ],
[ 3861.07122225,  2233.4       ,  2400.7       ],
[ 3848.11037587,  1914.1       ,  2233.8       ],
[ 3832.78771708,  1975.1       ,  2044.4       ],
[ 3809.61550528,  2234.2       ,  2237.1       ],
[ 3772.19627459,  2362.4       ,  2370.5       ],
[ 3731.13600549,  2403.1       ,  2436.7       ],
[ 3691.13881915,  2324.3       ,  2424.8       ],
[ 3659.02110924,  2344.       ,  2412.8       ],
[ 3631.68246263,  2511.4       ,  2530.3       ],
[ 3618.9317481 ,  2564.9       ,  2576.7       ],
[ 3625.48631189,  2513.9       ,  2568.7       ],
[ 3637.06747509,  2614.2       ,  2617.5       ],
[ 3660.23915255,  2627.9       ,  2634.8       ],
[ 3690.70092203,  2617.3       ,  2642.7       ],
[ 3714.23355069,  2572.5       ,  2658.7       ],
[ 3731.82210665,  2536.5       ,  2617.5       ],
[ 3746.054431  ,  2424.6       ,  2555.3       ],
[ 3760.45774629,  2480.6       ,  2529.6       ],
[ 3772.87409338,  2558.4       ,  2576.3       ],
[ 3776.40933114,  2577.7       ,  2605.9       ],
[ 3774.518011  ,  2583.8       ,  2616.9       ],
[ 3773.43221851,  2446.1       ,  2585.1       ],
[ 3770.10671182,  2541.6       ,  2584.8       ],
[ 3764.84499987,  2590.1       ,  2660.7       ],
[ 3760.49087579,  2710.4       ,  2741.6       ],
[ 3759.38023617,  2722.8       ,  2759.7       ],
[ 3764.0095488 ,  2677.6       ,  2757.3       ],
[ 3775.81513761,  2754.4       ,  2804.4       ],
[ 3790.69829624,  2616.8       ,  2800.5       ],
[ 3803.36525193,  2539.6       ,  2617.8       ],
[ 3812.98830165,  2655.1       ,  2676.        ],
[ 3820.30243927,  2508.6       ,  2690.7       ],
[ 3828.86881839,  2442.5       ,  2536.4       ],
[ 3833.40622339,  2467.3       ,  2521.6       ],
[ 3829.5103016 ,  2713.        ,  2803.7       ],
[ 3819.41969852,  2656.8       ,  2784.8       ],
[ 3808.64860492,  2973.4       ,  2985.1       ],
[ 3804.07433   ,  2900.3       ,  2977.9       ],
[ 3804.68615697,  2811.4       ,  2914.2       ],
[ 3813.66318153,  2798.8       ,  2852.1       ],
[ 3833.79976973,  2691.5       ,  2808.4       ],
[ 3851.89087877,  2870.5       ,  2880.9       ],
[ 3870.57377678,  2705.        ,  2931.2       ],
[ 3895.09254227,  2524.1       ,  2705.4       ],
```

```
       [ 3916.37370111,  2545.4       ,  2559.8       ],
       [ 3918.56612562,  2492.6       ,  2582.8       ],
       [ 3912.36236043,  2412.6       ,  2494.       ],
       [ 3889.22381247,  2303.3       ,  2460.8       ],
       [ 3857.85418794,  2192.6       ,  2330.6       ],
       [ 3826.31998463,  2278.2       ,  2329.3       ],
       [ 3793.96903239,  2189.       ,  2337.4       ],
       [ 3763.17409774,  2052.4       ,  2300.5       ],
       [ 3727.08592468,  2244.9       ,  2322.4       ],
       [ 3691.39477122,  2307.       ,  2616.5       ],
       [ 3668.82476814,  2445.3       ,  2781.8       ],
       [ 3654.91171595,  2272.6       ,  2497.       ],
       [ 3645.20664402,  2124.4       ,  2286.3       ],
       [ 3640.54553775,  2044.2       ,  2264.8       ],
       [ 3638.17277123,  2040.2       ,  2094.9       ],
       [ 3637.79846137,  1962.       ,  2048.4       ],
       [ 3632.78709085,  1881.       ,  1969.7       ],
       [ 3623.57121332,  1801.3       ,  1980.5       ],
       [ 3613.25297855,  1729.3       ,  1842.8       ],
       [ 3595.21650646,  1708.9       ,  1752.6       ],
       [ 3565.47009362,  1772.6       ,  1776.7       ],
       [ 3528.31576531,  1763.7       ,  1802.8       ],
       [ 3497.17390606,  1686.4       ,  1770.5       ],
       [ 3473.7075364 ,  1819.3       ,  1822.5       ],
       [ 3453.89957141,  1752.3       ,  1864.8       ],
       [ 3438.85143324,  1697.5       ,  1766.2       ],
       [ 3426.91826916,  1664.5       ,  1757.4       ],
       [ 3418.65112419,  1554.4       ,  1667.7       ],
       [ 3411.49515319,  1545.3       ,  1572.9       ],
       [ 3406.24199078,  1507.8       ,  1560.4       ],
       [ 3400.48654248,  1516.8       ,  1588.1       ],
       [ 3390.76544015,  1485.6       ,  1609.8       ],
       [ 3378.39184079,  1445.9       ,  1496.4       ],
       [ 3366.34232272,  1415.8       ,  1471.1       ],
       [ 3350.61035436,  1351.9       ,  1448.7       ]])
```

[114]:
```
lstm_df_final=lstm_df_input[predictions.shape[0]*-1:]
```

[115]:
```
# lstm_df_final.count()
```

[116]:
```
lstm_df_final['Price_pred']=rev_trans[:,0]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:


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/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[117]:
```
# lstm_df_final
```

[118]:
```
lstm_df_final[['Price','Price_pred']].plot()
```

[118]: <matplotlib.axes._subplots.AxesSubplot at 0x7fab44dbc450>

[119]:
```
# RMSE for LSTM Model

((lstm_df_final['Price']-lstm_df_final['Price_pred'])**2).mean()**0.5
```

[119]: 1336.836148623697

[120]:
```
# RMSE % for LSTM Model

rmspe = (np.sqrt(np.mean(np.
 ↪square((lstm_df_final['Price']-lstm_df_final['Price_pred']) /
 ↪lstm_df_final['Price'])))) * 100
```

```python
print(rmspe)
```

```
47.82700161162234
```

11 3. XGBOOST MODEL

XGBoost stands for eXtreme Gradient Boosting. XGBoost is very efficient implementation of gradient boosting for classification as well as regression problems. While applying XGBoost on timeseries data first we need to transform our data in supervised learning. XGBoost is very good at identifying patterns in data, if we have enough temporal features describing our dataset, it provides very decent predictions.

[121]:
```python
xg_df=new_df[['Price']].copy()
xg_df
```

[121]:
```
                Price
Date
2020-08-02   11105.8
2020-08-01   11803.1
2020-07-31   11333.4
2020-07-30   11096.2
2020-07-29   11105.9
…              …
2017-05-05    1507.8
2017-05-04    1516.8
2017-05-03    1485.6
2017-05-02    1445.9
2017-05-01    1415.8

[1190 rows x 1 columns]
```

Transform this to a supervised learning problem

[122]:
```python
xg_df["Target"]=xg_df.Price.shift(-1)
xg_df.dropna(inplace=True)
```

[123]:
```python
xg_df
```

[123]:
```
                Price    Target
Date
2020-08-02   11105.8   11803.1
2020-08-01   11803.1   11333.4
2020-07-31   11333.4   11096.2
2020-07-30   11096.2   11105.9
2020-07-29   11105.9   10908.5
…              …         …
2017-05-06    1545.3    1507.8
```

```
2017-05-05   1507.8   1516.8
2017-05-04   1516.8   1485.6
2017-05-03   1485.6   1445.9
2017-05-02   1445.9   1415.8

[1189 rows x 2 columns]
```

[124]:
```
# Train Test Split

def train_test_split(data, per):
  data=data.values
  n=int(len(data) * (1-per))
  return data[:n], data[n:]
```

[125]:
```
train, test = train_test_split(xg_df, 0.2)
```

[126]:
```
print(len(df))
print(len(train))
print(len(test))
```

```
1190
951
238
```

[127]:
```
X=train[:, :-1]
y=train[:,-1]
```

[128]:
```
# Prediction

def xgb_predict(train, val):
  train = np.array(train)
  X,y = train[:, :-1], train[:, -1]
  model = XGBRegressor(objective="reg:squarederror", n_estimators=1000)
  # model = XGBRegressor(objective="reg:squarederror", n_estimators=1000,␣
↪learning_rate=0.1, booster='gbtree', max_depth=6, gamma=0 )
  # model = XGBRegressor(objective="reg:squarederror", n_estimators=2000,␣
↪learning_rate=0.1, booster='gbtree', max_depth=6, gamma=0 )
  # model = XGBRegressor(objective="reg:squarederror", n_estimators=1000,␣
↪learning_rate=0.1, booster='gbtree', max_depth=10, gamma=0)
  model.fit(X,y)

  val = np.array(val).reshape(1,-1)
  pred=model.predict(val)
  return pred[0]
```

[129]:
```
# Walk Forward Validation
```

```python
from sklearn.metrics import mean_squared_error

def validate(data,per):
  predictions = []

  train, test = train_test_split(data,per)
  history = [x for x in train]

  for i in range(len(test)):
    test_X, test_y = test[i, :-1], test[i, -1]
    pred = xgb_predict(history, test_X[0])
    predictions.append(pred)

    history.append(test[i])

  error = mean_squared_error(test[:,-1], predictions, squared=False)

  return error, test[:,-1], predictions
```

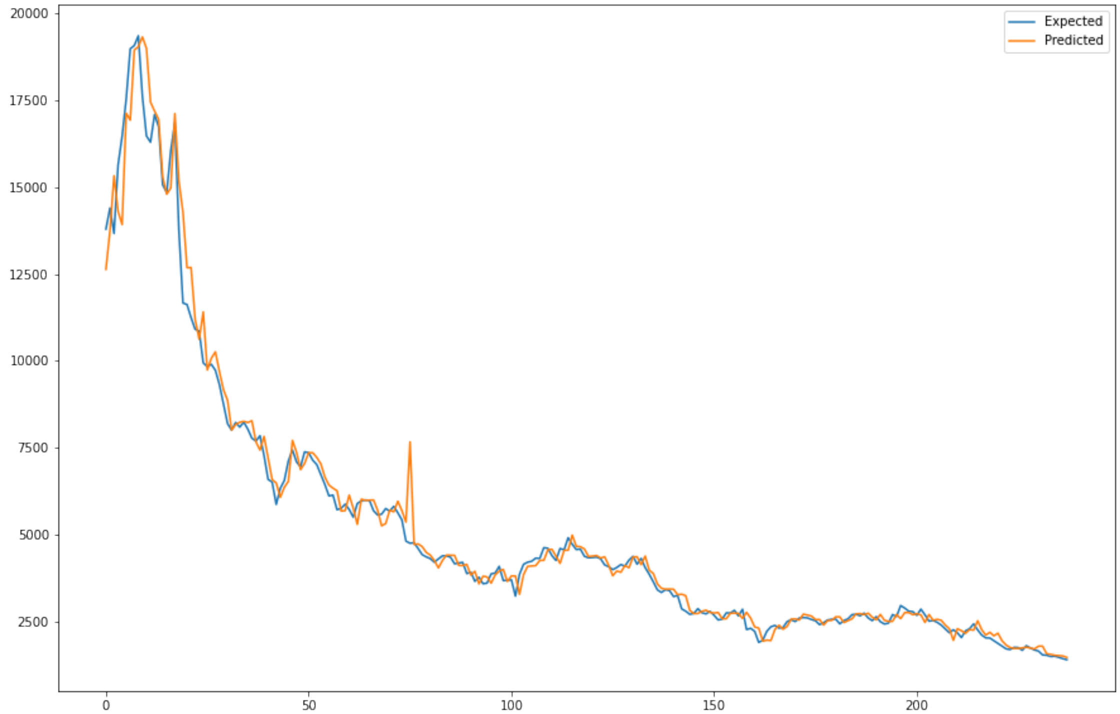
[130]:
```python
from xgboost import XGBRegressor
#%%time
rmse,y,pred = validate(xg_df, 0.2)
print(rmse)
```

```
/usr/local/lib/python3.7/dist-packages/xgboost/core.py:613: UserWarning:

Use subset (sliced data) of np.ndarray is not recommended because it will
generate extra copies and increase memory consumption


513.6327119929689
```

[131]:
```python
# RMSE % for XGBoost Model

rmspe = (np.sqrt(np.mean(np.square((y-pred) / y)))) * 100
print(rmspe)
```

```
7.591406062882989
```

[132]:
```python
pt.plot(y, label='Expected')
pt.plot(pred, label='Predicted')
pt.legend()
pt.show()
```



ARIMA

[133]:
```
import statsmodels.api as sm
train=new_df.head(n=800)
model=sm.tsa.statespace.SARIMAX(train['Price'],order=(5, 3,␣
↪1),seasonal_order=(5,1,1,12))
results=model.fit()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165:
ValueWarning:

No frequency information was provided, so inferred frequency -1D will be used.

/usr/local/lib/python3.7/dist-
packages/statsmodels/tsa/statespace/sarimax.py:949: UserWarning:

Non-stationary starting autoregressive parameters found. Using zeros as starting
parameters.

/usr/local/lib/python3.7/dist-
packages/statsmodels/tsa/statespace/sarimax.py:961: UserWarning:

Non-invertible starting MA parameters found. Using zeros as starting parameters.
```

[135]:
```
model2=sm.tsa.statespace.SARIMAX(train['Price'],order=(3, 2, 1))
results2=model2.fit()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165:
ValueWarning:

No frequency information was provided, so inferred frequency -1D will be used.
```

[137]:
```
import matplotlib.pyplot as plt
forecatedvalue=results.forecast(steps=200)
train['Predicted']=results.predict()
plt.plot(train['Predicted'],label="train predicted")
plt.plot(new_df['Price'],label="Actual price")
plt.plot(forecatedvalue,label="forecasted")
plt.title("Actual vs forecasted vs training prediction")
plt.legend()
plt.rcParams['figure.figsize'] = [4,4]
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3:
SettingWithCopyWarning:


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/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[138]:
```
forecatedvalue2=results2.forecast(steps=200)
train['Predicted2']=results2.predict()
plt.plot(train['Predicted2'],label="train predicted")
plt.plot(new_df['Price'],label="Actual price")
plt.plot(forecatedvalue2,label="forecasted")
plt.title("Actual vs forecasted vs training prediction")
plt.legend()
plt.rcParams['figure.figsize'] = [12,6]
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:


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/user_guide/indexing.html#returning-a-view-versus-a-copy
```


```
[139]: mse=mean_squared_error(train["Price"], train['Predicted'], squared=False)
rmspe = (np.sqrt(np.mean(np.square((train["Price"]-train['Predicted']) /
    train['Price'])))) * 100
print(rmspe)
print(mse)
```

```
7.2002248498345205
730.3503950024801
```

```
[140]: mse=mean_squared_error(train["Price"], train['Predicted2'], squared=False)
rmspe = (np.sqrt(np.mean(np.square((train["Price"]-train['Predicted2']) /
    train['Price'])))) * 100
print(rmspe)
print(mse)
```

```
5.42347347555938
528.2954383455034
```

Comparision on a simpler dataset for understanding models

The additive model is Y[t] = T[t] + S[t] + e[t]

The multiplicative model is Y[t] = T[t] * S[t] * e[t]

The results are obtained by first estimating the trend by applying a convolution filter to the data. The trend is then removed from the series and the average of this de-trended series for each period is the returned seasonal component.