TimeSeries

May 23, 2022

1 Time Series Forecasting

Dataset Introduction: This dataset used here isTime Series Forecasting with Yahoo Stock Price from kaggle .This dataset is historical stocks of yahoo finance corp . The dataset consists of following columns

1. Date - Trading Date
2. High - the high refers to the maximum prices in a given time period.
3. Low - the low refers to the minimum prices in a given time period.
4. Open - prices at which a stock began trading in the same period.
5. close - the prices at which a stock ended trading in the same period.
6. Volume - Volume is the total amount of trading activity
7. Adj close - Adjusted values factor in corporate actions such as dividends, stock splits, and new share issuance.

Problem statement - Forecast the close series using deep learning

1.1 Exploratory data analysis

[60]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore')
import pandas as pd
from datetime import datetime
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, LSTM, Activation, Dropout
import math
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.optimizers import Adam
```

[2]:
```
data = pd.read_csv('/content/drive/MyDrive/ibm/Projects/timeseriesanalysis/
↪yahoo_stock(Time series).csv',sep=",")
```

```
data.head()
```

[2]:
```
         Date         High          Low         Open        Close  \
0  2015-11-23  2095.610107  2081.389893  2089.409912  2086.590088
1  2015-11-24  2094.120117  2070.290039  2084.419922  2089.139893
2  2015-11-25  2093.000000  2086.300049  2089.300049  2088.870117
3  2015-11-26  2093.000000  2086.300049  2089.300049  2088.870117
4  2015-11-27  2093.290039  2084.129883  2088.820068  2090.110107

         Volume    Adj Close
0  3.587980e+09  2086.590088
1  3.884930e+09  2089.139893
2  2.852940e+09  2088.870117
3  2.852940e+09  2088.870117
4  1.466840e+09  2090.110107
```

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1825 entries, 0 to 1824
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1825 non-null   object
 1   High       1825 non-null   float64
 2   Low        1825 non-null   float64
 3   Open       1825 non-null   float64
 4   Close      1825 non-null   float64
 5   Volume     1825 non-null   float64
 6   Adj Close  1825 non-null   float64
dtypes: float64(6), object(1)
memory usage: 99.9+ KB
```

[5]:
```
data.dtypes
```

[5]:
```
Date          object
High         float64
Low          float64
Open         float64
Close        float64
Volume       float64
Adj Close    float64
dtype: object
```

[6]:
```
data.isnull().sum()
```

[6]:
```
Date         0
High         0
Low          0
Open         0
Close        0
Volume       0
Adj Close    0
dtype: int64
```

[7]:
```
data.Date=pd.to_datetime(data['Date'])
```

[8]:
```
data.set_index('Date',inplace=True)
```

[9]:
```
data.head()
```

[9]:
```
                   High          Low         Open        Close        Volume  \
Date
2015-11-23  2095.610107  2081.389893  2089.409912  2086.590088  3.587980e+09
2015-11-24  2094.120117  2070.290039  2084.419922  2089.139893  3.884930e+09
2015-11-25  2093.000000  2086.300049  2089.300049  2088.870117  2.852940e+09
2015-11-26  2093.000000  2086.300049  2089.300049  2088.870117  2.852940e+09
2015-11-27  2093.290039  2084.129883  2088.820068  2090.110107  1.466840e+09

              Adj Close
Date
2015-11-23  2086.590088
2015-11-24  2089.139893
2015-11-25  2088.870117
2015-11-26  2088.870117
2015-11-27  2090.110107
```

[10]:
```
data.describe()
```

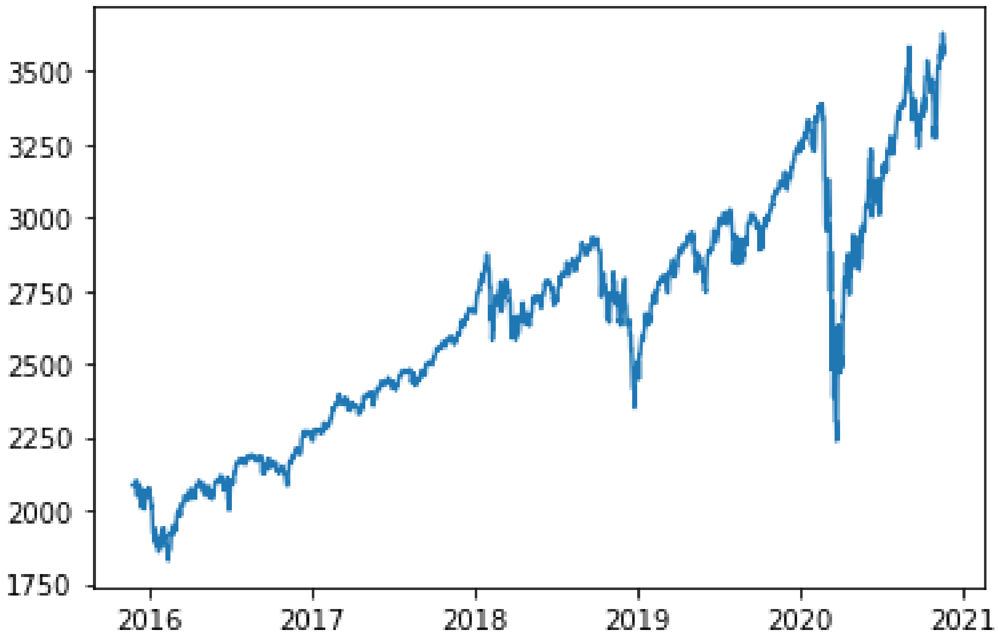
[10]:
```
              High          Low         Open        Close        Volume  \
count  1825.000000  1825.000000  1825.000000  1825.000000  1.825000e+03
mean   2660.718673  2632.817580  2647.704751  2647.856284  3.869627e+09
std     409.680853   404.310068   407.169994   407.301177  1.087593e+09
min    1847.000000  1810.099976  1833.400024  1829.079956  1.296540e+09
25%    2348.350098  2322.250000  2341.979980  2328.949951  3.257950e+09
50%    2696.250000  2667.840088  2685.489990  2683.340088  3.609740e+09
75%    2930.790039  2900.709961  2913.860107  2917.520020  4.142850e+09
max    3645.989990  3600.159912  3612.090088  3626.909912  9.044690e+09

         Adj Close
count  1825.000000
mean   2647.856284
std     407.301177
```

```
min     1829.079956
25%     2328.949951
50%     2683.340088
75%     2917.520020
max     3626.909912
```

[13]:
```
plt.plot(data.Close)
```

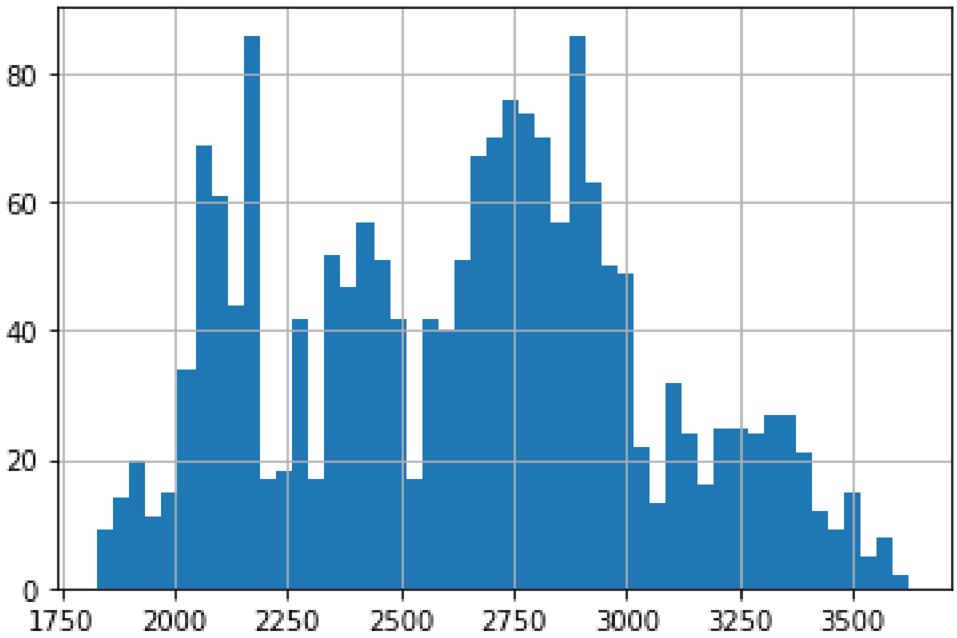
[13]:
```
[<matplotlib.lines.Line2D at 0x7fb3fda02510>]
```



1.2 Stationary Check

[14]:
```
data.Close.hist(bins=50)
```

[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3fd956590>
```



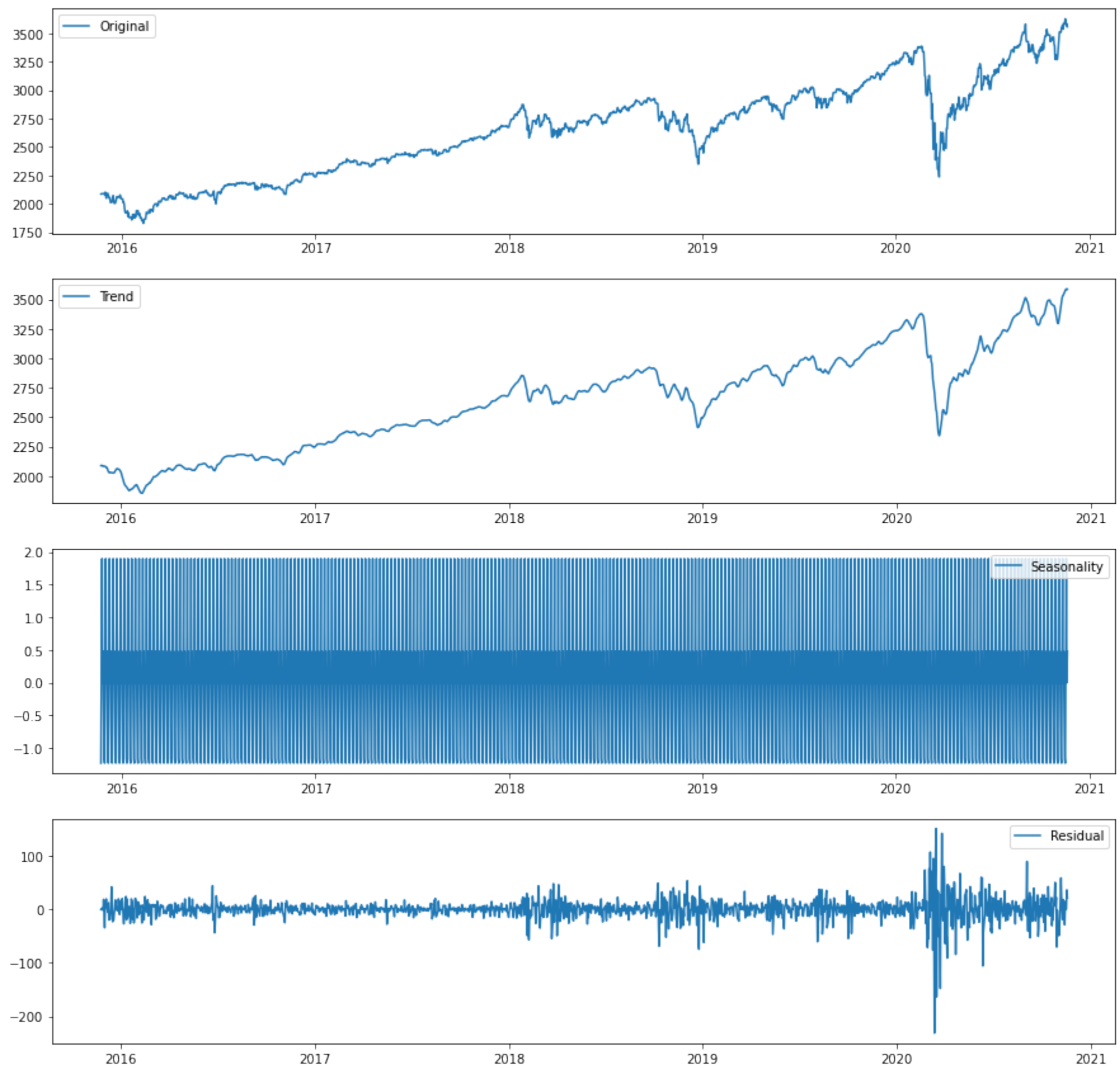
It seems that Close series is non stationary. Lets check this further

Lets decompose this series using additive and multiplicative decomposiion

1.2.1 Decomposition

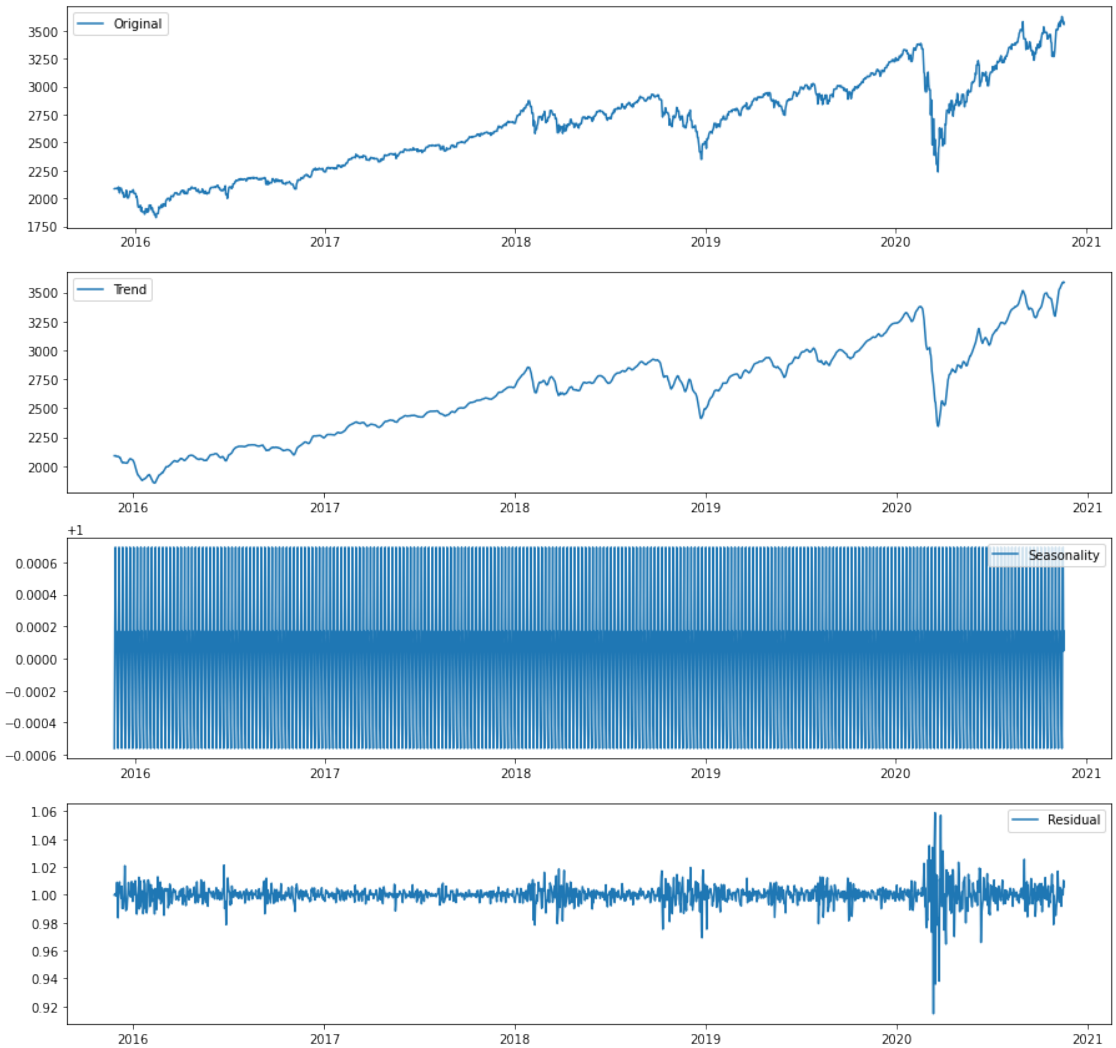
[25]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
decompose_add=seasonal_decompose(data['Close'], model='additive')
plt.figure(figsize=(15,15))
plt.subplot(411)
plt.plot(data['Close'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(decompose_add.trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(decompose_add.seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(decompose_add.resid, label='Residual')
plt.legend(loc='best')
```

[25]:
```
<matplotlib.legend.Legend at 0x7fb3f8badb50>
```



[24]:
```
decompose_add=seasonal_decompose(data['Close'], model='multiplicative')
plt.figure(figsize=(15,15))
plt.subplot(411)
plt.plot(data['Close'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(decompose_add.trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(decompose_add.seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(decompose_add.resid, label='Residual')
plt.legend(loc='best')
```

[24]: <matplotlib.legend.Legend at 0x7fb3f8cc82d0>

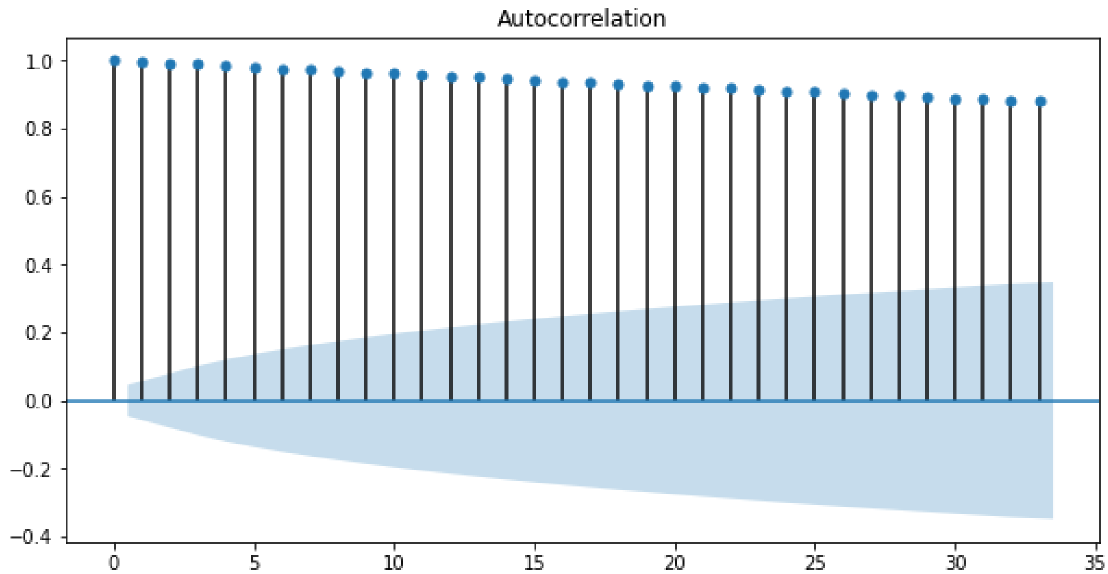


We can conclude that additive decomposition explains the series better than multiplicative decompositions

1.2.2 Plots

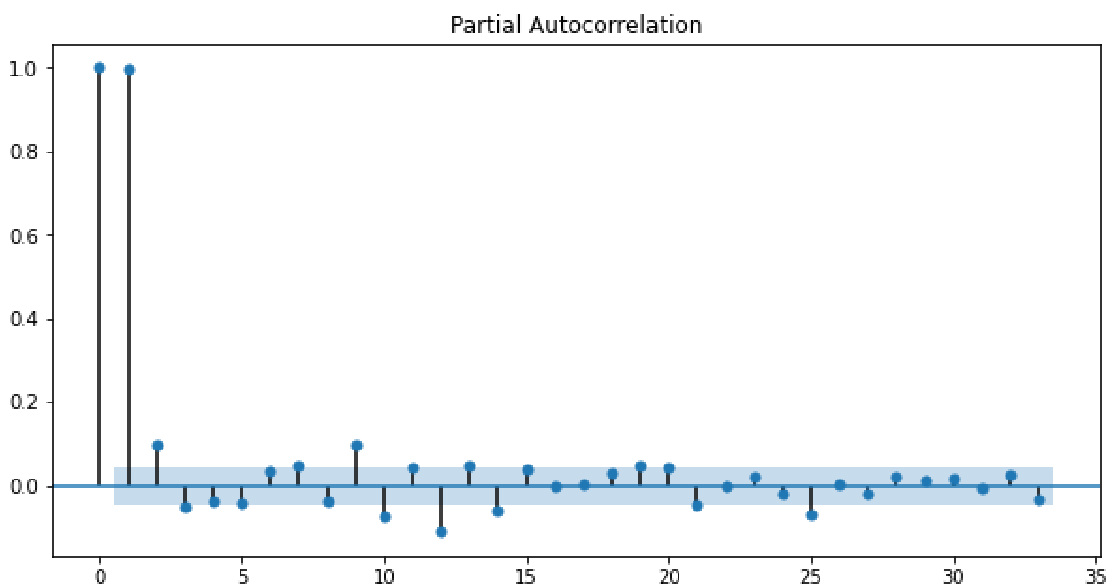
AutoCorrelation

[21]:
```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
plt.rc("figure", figsize=(10,5))
plot_acf(data.Close)
print()
```



Partial Autocorrelation

[22]:
```
plt.rc("figure", figsize=(10,5))
plot_pacf(data.Close)
print()
```



Hypothesis Testing 1. H0 = non-stationary type. 2. H1 = stationary series

[23]:
```
from statsmodels.tsa.stattools import adfuller
```

[26]:
```
result = adfuller(data['Close'])
print('p-value:' +str(result[1]))
```

```
p-value:0.7975646340657458
```

Thus we can conclude Close series is a non stationary series

1.3 Train and test split

[65]:
```
data2 = data[['Close']]
timesteps = 50
train = data2[:len(data)-timesteps]['Close'].values
test = data2[len(train):]['Close'].values
train=train.reshape(train.shape[0],1)
test=test.reshape(test.shape[0],1)
sc = MinMaxScaler(feature_range= (0,1))
train = sc.fit_transform(train)
X_train = []
y_train = []

for i in range(timesteps, train.shape[0]):
    X_train.append(train[i-timesteps:i,0])
    y_train.append(train[i,0])

X_train = np.array(X_train)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
y_train = np.array(y_train)
print('Training input shape: {}'.format(X_train.shape))
print('Training output shape: {}'.format(y_train.shape))
```

```
Training input shape: (1725, 50, 1)
Training output shape: (1725,)
```

[66]:
```
inputs = data2[len(data) - len(test) - timesteps:]

inputs = sc.transform(inputs)

X_test = []

for i in range(timesteps, 100):
    X_test.append(inputs[i-timesteps:i,0])

X_test = np.array(X_test)
```

```
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
X_test.shape
```

[66]: (50, 50, 1)

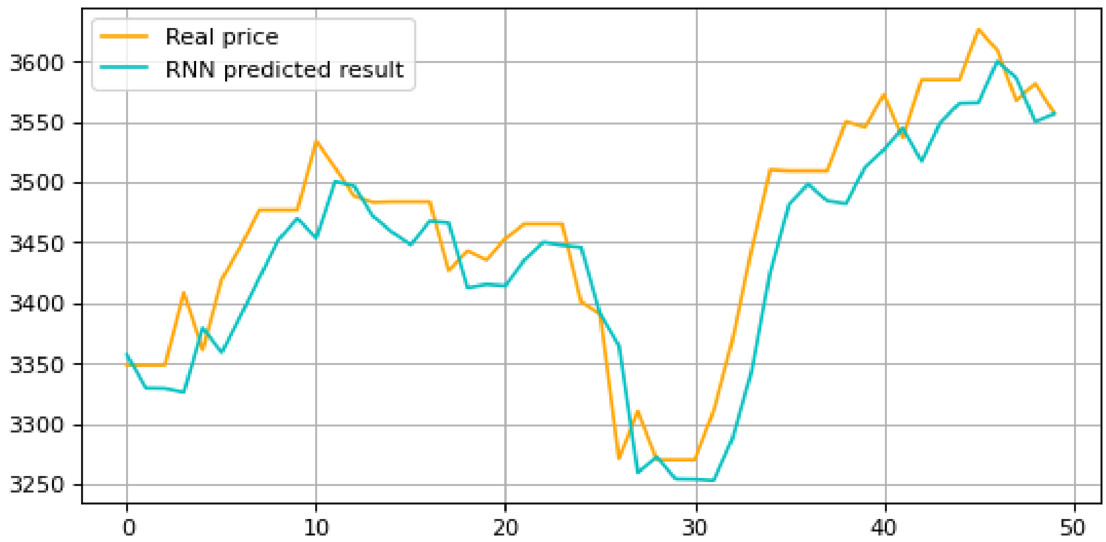
2 Models

2.1 RNN

[ ]:
```
model=Sequential()
model.add(SimpleRNN(50,return_sequences=True, activation='relu',␣
↪input_shape=(X_train.shape[1],1)))
model.add(SimpleRNN(50,return_sequences=False,activation='relu'))
model.add(Dense(100))
model.add(Dense(25))
model.add(Dense(1))
opt1=Adam(learning_rate=1e-4,beta_1=0.9,beta_2=0.7)
model.compile(loss='mean_squared_error', optimizer=opt1)
model.fit(X_train, y_train, epochs=100, batch_size=32)
```

[68]:
```
predicted = model.predict(X_test)
predicted = sc.inverse_transform(predicted)

plt.figure(figsize=(8,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(test,color="orange",label="Real price")
plt.plot(predicted,color="c",label="RNN predicted result")
plt.legend()
plt.grid(True)
plt.show()
```



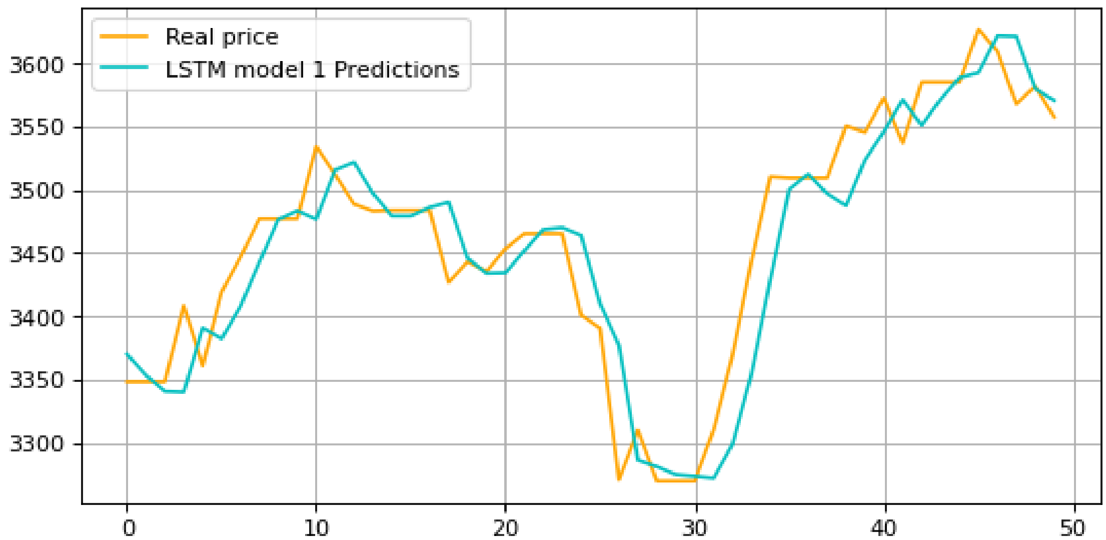
3 LSTM

[ ]:
```
model2 = Sequential()
model2.add(LSTM(70, input_shape=(X_train.shape[1],1)))
model2.add(Dense(1))

model2.compile(loss='mean_squared_error', optimizer='adam')
model2.fit(X_train, y_train, epochs=100, batch_size=32)
```

[74]:
```
predicted2 = model2.predict(X_test)
predicted2 = sc.inverse_transform(predicted2)

plt.figure(figsize=(8,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(test,color="orange",label="Real price")
plt.plot(predicted2,color="c",label="LSTM model 1 Predictions")
plt.legend()
plt.grid(True)
plt.show()
```

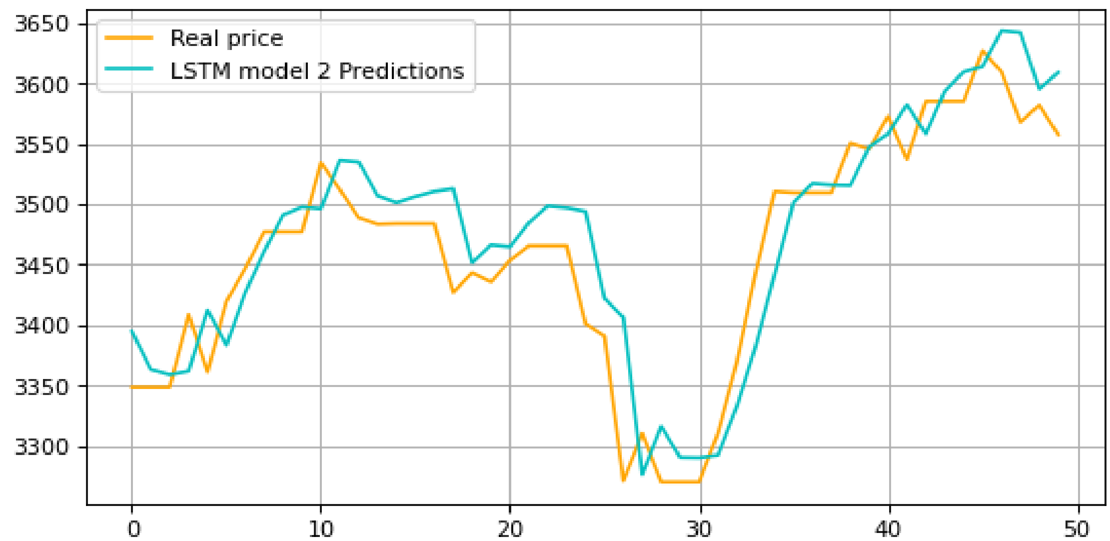


[ ]:
```
model3=Sequential()
model3.add(LSTM(50,return_sequences=True, activation='relu',␣
↪input_shape=(X_train.shape[1],1)))
model3.add(LSTM(50,return_sequences=False,activation='relu'))
model3.add(Dense(100))
```

```
model3.add(Dense(25))
model3.add(Dense(1))
opt1=Adam(learning_rate=0.001,beta_1=0.9,beta_2=0.999)
model3.compile(loss='mean_squared_error', optimizer=opt1)
model3.fit(X_train, y_train, epochs=100, batch_size=10)
```

[75]:
```
predicted3 = model3.predict(X_test)
predicted3 = sc.inverse_transform(predicted3)

plt.figure(figsize=(8,4), dpi=80, facecolor='w', edgecolor='k')
plt.plot(test,color="orange",label="Real price")
plt.plot(predicted3,color="c",label="LSTM model 2 Predictions")
plt.legend()
plt.grid(True)
plt.show()
```



[80]:
```
print("rnn :"+str((mean_squared_error(y_train, model.predict(X_train)))))
print("lstm model 1 :"+str((mean_squared_error(y_train, model2.
 ↪predict(X_train)))))
print("lstm model 2 :"+str((mean_squared_error(y_train, model3.
 ↪predict(X_train)))))
```

```
rnn :0.00022443603655724226
lstm model 1 :0.00023869740249628116
lstm model 2 :0.0002348539642872087
```

4 Results

Among the 3 models all models performed moderatively with rnn showing the least mse even though the lstm were trained longer

5 Next Steps

We can use hyperparameters tuning to improve accuracy of lstm and we need more data for lstm , We can also use hyperparameter tuning on rnn