Stock_market_analysis

September 10, 2022

Analyze ups and downs in the market and predict future stock price returns based on Indian Market data from 2000 to 2020:

Installing all the required libraries

[ ]:
```
%pip install yfinance
%pip install tensorflow
%pip install pandas
%pip install numpy
%pip install plotly
%pip install nbformat
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting yfinance
  Downloading yfinance-0.1.74-py2.py3-none-any.whl (27 kB)
Collecting requests>=2.26
  Downloading requests-2.28.1-py3-none-any.whl (62 kB)
     |                                | 62 kB 1.6 MB/s
Requirement already satisfied: lxml>=4.5.1 in /usr/local/lib/python3.7/dist-packages (from yfinance) (4.9.1)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.21.6)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packages (from yfinance) (1.3.5)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2022.2.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.0->yfinance) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.24.0->yfinance) (1.15.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.10)
Requirement already satisfied: charset-normalizer<3,>=2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance) (2.1.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.26->yfinance)
```

Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.7/dist-packages (from nbformat) (5.1.1)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.7/dist-packages (from nbformat) (2.16.1)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.7/dist-packages (from nbformat) (4.11.1)
Requirement already satisfied: importlib-resources>=1.4.0 in /usr/local/lib/python3.7/dist-packages (from jsonschema>=2.6->nbformat) (5.9.0)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.7/dist-packages (from jsonschema>=2.6->nbformat) (22.1.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/local/lib/python3.7/dist-packages (from jsonschema>=2.6->nbformat) (0.18.1)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from jsonschema>=2.6->nbformat) (4.12.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from jsonschema>=2.6->nbformat) (4.1.1)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.7/dist-packages (from importlib-resources>=1.4.0->jsonschema>=2.6->nbformat) (3.8.1)

Importing

[ ]:
```
import yfinance as yf     # open source api to fetch historical stock market data
import numpy as np        # used for fast numerical operations
import pandas as pd       # used for row/column wise operations on dataframe
import tensorflow as tf # used for creating deep learning pipeline
import plotly.graph_objects as go    # used for ploting data in to graph
```

Feching Stock Market Data

[ ]:
```
# Downloads the day-by-day(as interval is set to '1d') stock market data of␣
↪Bajaj Auto from 2000 to 2020
data = yf.download("BAJAJ-AUTO.NS" , start = "2000-01-01", end = "2020-12-31" ,␣
↪interval = '1d')        # stock symbol = "BAJAJ-AUTO.NS" for Bajaj Auto in NSE␣
↪| "MOSFT" for micerosoft | "AAPL" for apple | "GOOGL" for google
```

[**********100%**********] 1 of 1 completed

[ ]:
```
# Gives the size of dataset
print(data.shape)
data.head()
```

(4601, 6)

[ ]:

Date	Open	High	Low	Close	Adj Close	Volume
2002-07-01	126.5000	130.000000	125.062500	129.250000	86.242096	114874
2002-07-02	131.1875	131.187500	126.512497	127.262497	84.915939	70272
2002-07-03	130.0000	133.300003	127.500000	132.074997	88.127075	158600
2002-07-04	127.2500	138.125000	127.250000	136.937500	91.371574	389836

```
2002-07-05   138.7500  138.750000  133.399994  134.187500  89.536621  162464
```

[ ]:
```
# Sort the data points based on indexes just for confirmation
data.sort_index(inplace = True)

# Remove any duplicate index
data = data.loc[~data.index.duplicated(keep='first')]

# Check for missing values
data.isnull().sum()
```

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

[ ]:
```
data.tail()
```

[ ]:
```
                 Open         High          Low        Close    Adj Close   \
Date
2020-12-23  3281.000000  3318.949951  3261.000000  3309.649902  3083.218750
2020-12-24  3318.000000  3423.550049  3316.050049  3374.750000  3143.864990
2020-12-28  3388.000000  3422.000000  3374.000000  3414.699951  3181.081787
2020-12-29  3433.000000  3459.899902  3420.050049  3431.550049  3196.779053
2020-12-30  3437.199951  3472.850098  3405.250000  3448.149902  3212.243164

              Volume
Date
2020-12-23    816586
2020-12-24   1567636
2020-12-28    536954
2020-12-29    682613
2020-12-30    639180
```

[ ]:
```
# Get the statistics of the data
data.describe()
```

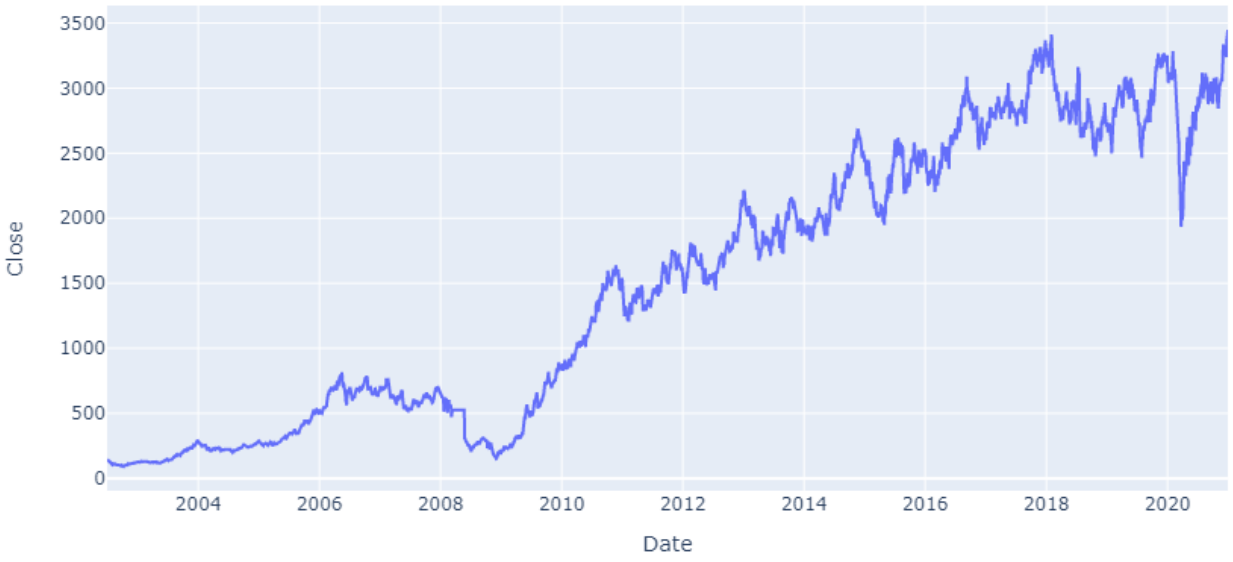
[ ]:
```
              Open         High          Low        Close    Adj Close   \
count  4601.000000  4601.000000  4601.000000  4601.000000  4601.000000
mean   1501.160174  1520.378396  1480.887455  1500.148467  1216.772515
std    1053.954358  1064.503109  1042.549098  1052.976073   936.606604
min      91.250000    93.125000    90.199997    91.525002    61.070076
25%     503.600006   512.000000   495.100006   503.812500   341.879333
50%    1521.000000  1543.000000  1492.000000  1517.449951  1122.432495
```

```
75%     2530.949951  2561.000000  2496.449951  2530.050049  2104.513184
max     3437.199951  3472.850098  3420.050049  3448.149902  3212.243164
```

```
              Volume
count 4.601000e+03
mean  4.101147e+05
std   3.647727e+05
min   0.000000e+00
25%   2.135750e+05
50%   3.219780e+05
75%   4.967100e+05
max   6.107930e+06
```

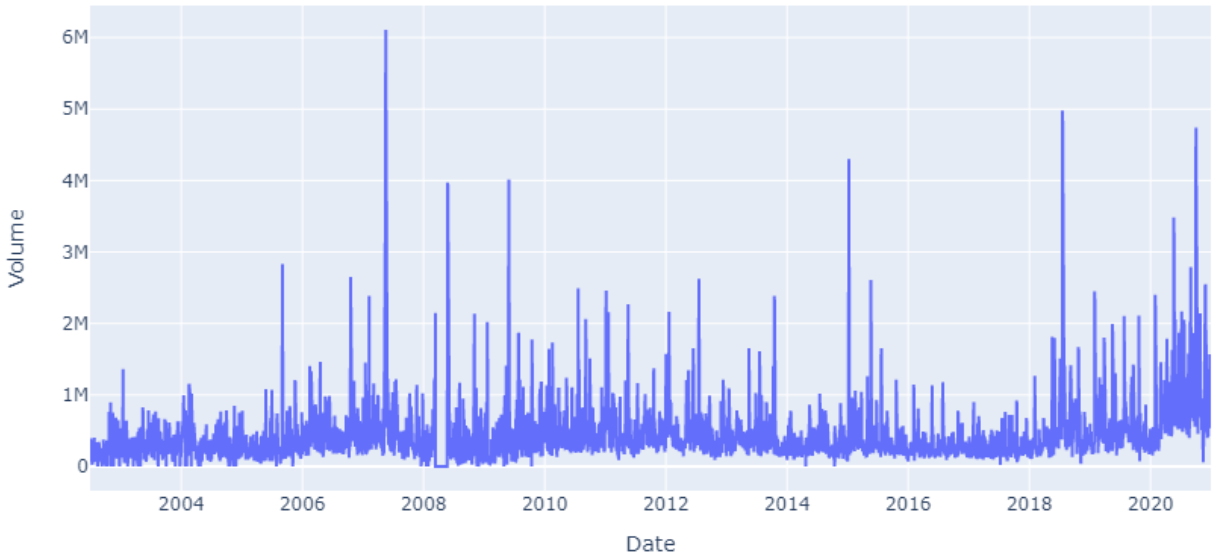
[ ]:
```
# Check the trend in Closing Values
fig = go.Figure()

fig.add_trace(go.Scatter(x = data.index , y = data['Close'] , mode = 'lines'))
fig.update_layout(height = 500 , width = 900,
                  xaxis_title='Date' , yaxis_title='Close')
fig.show()
```



[ ]:
```
# Check the trend in Volume Traded
fig = go.Figure()

fig.add_trace(go.Scatter(x = data.index , y = data['Volume'] , mode = 'lines'))
fig.update_layout(height = 500 , width = 900,
                  xaxis_title='Date' , yaxis_title='Volume')
fig.show()
```



Creating DataSet

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
import pickle                              # used to save and retrive any python object
from tqdm.notebook import tnrange          # graphical progress bar, to track the␣
↪progress of our preprocessing
```

[ ]:
```
# Filter only required data
data = data[['Close' , 'Volume']]          # volume plays crucial role in increase/
↪decrease of stock price
data.head()
```

[ ]:

Date	Close	Volume
2002-07-01	129.250000	114874
2002-07-02	127.262497	70272
2002-07-03	132.074997	158600
2002-07-04	136.937500	389836

```
2002-07-05   134.187500  162464
```

Creating Training Dataset

[ ]:
```
# Confirm the Testing Set length
test_length = data[(data.index >= "2018-01-01")].shape[0]
print(test_length)
```

```
739
```

[ ]:
```
def CreateFeatures_and_Targets(data, feature_length):
    X = []
    Y = []

    for i in tnrange(len(data) - feature_length):
        X.append(data.iloc[i : i + feature_length,:].values)
        Y.append(data["Close"].values[i+feature_length])

    X = np.array(X)
    Y = np.array(Y)

    return X , Y
```

[ ]:
```
X , Y = CreateFeatures_and_Targets(data , 32)
```

```
  0%|          | 0/4569 [00:00<?, ?it/s]
```

[ ]:
```
# Check the shapes
X.shape , Y.shape
```

[ ]:
```
((4569, 32, 2), (4569,))
```

[ ]:
```
Xtrain , Xtest , Ytrain , Ytest = X[:-test_length] , X[-test_length:] , Y[:
→-test_length] , Y[-test_length:]
```

[ ]:
```
# Check Training Dataset Shape
Xtrain.shape , Ytrain.shape
```

[ ]:
```
((3830, 32, 2), (3830,))
```

[ ]:
```
# Check Testing Dataset Shape
Xtest.shape , Ytest.shape
```

[ ]:
```
((739, 32, 2), (739,))
```

[ ]:
```
# Create a Scaler to Scale Vectors with Multiple Dimensions
class MultiDimensionScaler():
```

```
    def _init_(self):
        self.scalers = []

    def fit_transform(self , X):
        total_dims = X.shape[2]
        for i in range(total_dims):
            Scaler = MinMaxScaler()
            X[:, :, i] = Scaler.fit_transform(X[:,:,i])
            self.scalers.append(Scaler)
        return X

    def transform(self , X):
        for i in range(X.shape[2]):
            X[:, :, i] = self.scalers[i].transform(X[:,:,i])
        return X
```

[ ]:
```
Feature_Scaler = MultiDimensionScaler()
Xtrain = Feature_Scaler.fit_transform(Xtrain)
Xtest = Feature_Scaler.transform(Xtest)
```

[ ]:
```
Target_Scaler = MinMaxScaler()
Ytrain = Target_Scaler.fit_transform(Ytrain.reshape(-1,1))
Ytest = Target_Scaler.transform(Ytest.reshape(-1,1))
```

[ ]:
```
def save_object(obj , name : str):
    pickle_out = open(f"{name}.pck","wb")
    pickle.dump(obj, pickle_out)
    pickle_out.close()

def load_object(name : str):
    pickle_in = open(f"{name}.pck","rb")
    data = pickle.load(pickle_in)
    return data
```

[ ]:
```
# Save your objects for future purposes
save_object(Feature_Scaler , "Feature_Scaler")
save_object(Target_Scaler , "Target_Scaler")
```

[ ]:
```
from tensorflow.keras.callbacks import ModelCheckpoint , ReduceLROnPlateau

save_best = ModelCheckpoint("best_weights.h5", monitor="val_loss",␣
↪save_best_only=True, save_weights_only=True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.25,patience=5,␣
↪min_lr=0.00001,verbose = 1)
```

```
[ ]: from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense , Dropout , LSTM , Bidirectional ,␣
      ↪BatchNormalization

     model = Sequential()

     model.add(Bidirectional(LSTM(512 ,return_sequences=True , recurrent_dropout=0.
      ↪1, input_shape=(32, 3))))
     model.add(LSTM(256 ,recurrent_dropout=0.1))
     model.add(Dropout(0.3))
     model.add(Dense(64 , activation="elu"))
     model.add(Dropout(0.3))
     model.add(Dense(32 , activation="elu"))
     model.add(Dense(1 , activation="linear"))
```

```
[ ]: optimizer = tf.keras.optimizers.SGD(learning_rate = 0.002)
     model.compile(loss="mse",  optimizer=optimizer)
```

```
[ ]: history = model.fit(Xtrain, Ytrain,
                        epochs=10,
                        batch_size = 1,
                        verbose=1,
                        shuffle=False ,
                        validation_data=(Xtest , Ytest),
                        callbacks=[reduce_lr , save_best])
```

```
Epoch 1/10
3830/3830 [==============================] - 546s 141ms/step - loss: 0.0019 -
val_loss: 0.0138 - lr: 0.0020
Epoch 2/10
3830/3830 [==============================] - 532s 139ms/step - loss: 0.0037 -
val_loss: 0.0108 - lr: 0.0020
Epoch 3/10
3830/3830 [==============================] - 528s 138ms/step - loss: 0.0028 -
val_loss: 0.0092 - lr: 0.0020
Epoch 4/10
3830/3830 [==============================] - 528s 138ms/step - loss: 0.0024 -
val_loss: 0.0079 - lr: 0.0020
Epoch 5/10
3830/3830 [==============================] - 527s 138ms/step - loss: 0.0020 -
val_loss: 0.0051 - lr: 0.0020
Epoch 6/10
3830/3830 [==============================] - 535s 140ms/step - loss: 0.0018 -
val_loss: 0.0042 - lr: 0.0020
Epoch 7/10
3830/3830 [==============================] - 530s 138ms/step - loss: 0.0017 -
val_loss: 0.0043 - lr: 0.0020
```

```
Epoch 8/10
3830/3830 [==============================] - 527s 137ms/step - loss: 0.0015 -
val_loss: 0.0023 - lr: 0.0020
Epoch 9/10
3830/3830 [==============================] - 534s 140ms/step - loss: 0.0014 -
val_loss: 0.0030 - lr: 0.0020
Epoch 10/10
3830/3830 [==============================] - 528s 138ms/step - loss: 0.0014 -
val_loss: 0.0023 - lr: 0.0020
```

[ ]:
```
# Checking the model Structure
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona (1, 32, 1024)             2109440
 l)

 lstm_1 (LSTM)               (1, 256)                  1311744

 dropout (Dropout)           (1, 256)                  0

 dense (Dense)               (1, 64)                   16448

 dropout_1 (Dropout)         (1, 64)                   0

 dense_1 (Dense)             (1, 32)                   2080

 dense_2 (Dense)             (1, 1)                    33

=================================================================
Total params: 3,439,745
Trainable params: 3,439,745
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
# Load the best weights
model.load_weights("best_weights.h5")
```

Visualize prediction on Test Set

[ ]:
```
Predictions = model.predict(Xtest)

Predictions = Target_Scaler.inverse_transform(Predictions)
Actual = Target_Scaler.inverse_transform(Ytest)
```

```
Predictions = np.squeeze(Predictions , axis = 1)
Actual = np.squeeze(Actual , axis = 1)
```

```
# Creating Sample Test Dataframe
test_dataframe_dict = {'Actual' : list(Actual) , 'Predicted' :
↪list(Predictions)}
test_df = pd.DataFrame.from_dict(test_dataframe_dict)

test_df.index = data.index[-test_length:]
```

```
# Check the trend in Volume Traded
fig = go.Figure()

fig.add_trace(go.Scatter(x = test_df.index , y = Actual , mode = 'lines' ,
↪name='Actual'))
fig.add_trace(go.Scatter(x = test_df.index , y = Predictions , mode = 'lines' ,
↪name='Predicted'))
fig.show()
```

Visualize Prediction on whole data

```
Total_features = np.concatenate((Xtrain , Xtest) , axis = 0)
Total_Targets = np.concatenate((Ytrain , Ytest) , axis = 0)
Predictions = model.predict(Total_features)
```

```
Predictions = Target_Scaler.inverse_transform(Predictions)
Actual = Target_Scaler.inverse_transform(Total_Targets)
```

```
Predictions = np.squeeze(Predictions , axis = 1)
Actual = np.squeeze(Actual , axis = 1)
```

[ ]:
```
# Check the trend in Volume Traded
fig = go.Figure()

fig.add_trace(go.Scatter(x = data.index , y = Actual , mode = 'lines' ,
↪name="Actual"))
fig.add_trace(go.Scatter(x = data.index , y = Predictions , mode = 'lines' ,
↪name="Predicted"))
fig.show()
```