Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
2nd Semester	AY 2023-2024

ACTIVITY	**Hands-on 11.1**
Name	Cuevas, Christian Jay L.
Section	CPE32S3
Date Performed:	5/7/2024
Date Submitted:	5/11/2024
Instructor:	Engr. Roman M. Richard

Resources

- Jupyter Notebook/Google Colab
- IBM Dataset (https://raw.githubusercontent.com/ChristianJayCuevas/CPE-019---Emerging-Technologies-2/main/Hands-on%2011.1/IBM_2006-01-01_to_2018-01-01.csv)

Instructions:

- Given an IBM stocks dataset between 2006 to 2018. You are task to do the following:
  - Load the dataset and examine it.
  - Check for missing values.
  - Satisfy the following condition: Training set (before 2017) and Test set (2017 and beyond)
  - Scale the training set from 0 to 1. Use MinMaxScaler and fit_transform function to do this.
  - LSTM stores long-term memory states. To do this, create a data structure with 60 timesteps and 1 output. Thus, for each element of the training set, we shall have 60 previous training set elements.
  - Reshape the X_train for efficient modeling
- Submit the accomplished Jupyter notebook (PDF).

Importing Libraries and Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from pandas import read_csv
from pandas import DataFrame
from statsmodels.tsa.stattools import adfuller
from sklearn.preprocessing import MinMaxScaler
from numpy import array
import tensorflow as tf
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
from tensorflow.keras.losses import MAPE
```

```
ibm_dataset = read_csv('https://raw.githubusercontent.com/ChristianJayCuevas/CPE-019---Emerging-Technologies-2/main/Hands-on%2011.1/IBM_2006-01-01
```

Observation:

- We can utilize the read_csv() function within the pandas library to bring in the dataset. In addition, we'll apply parameters such as header, parse_dates, and index_col. The header parameter identifies where the header begins, parse_dates designates the date column, and index_col specifies the index column [1][2].

Exploratory Data Analysis

```python
ibm_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3020 entries, 2006-01-03 to 2017-12-29
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Open    3019 non-null   float64
 1   High    3020 non-null   float64
 2   Low     3019 non-null   float64
 3   Close   3020 non-null   float64
 4   Volume  3020 non-null   int64  
 5   Name    3020 non-null   object 
dtypes: float64(4), int64(1), object(1)
memory usage: 165.2+ KB
```

Observation:

- We can observe from this dataset that there is a DatetimeIndex with 3020 entries ranging from 2006-01-03 to 2017-12-29. This shows that the index of the our dataset comprises of dates. We can also observe that there are 3020 entries for other columns but for Open and Low there are only 3019 so this indicates there are missing data. For the data type, 4 are float64, the volume is int64 and the name is an object.

```python
ibm_dataset.isna().sum()
```

```
Open      1
High      0
Low       1
Close     0
Volume    0
Name      0
dtype: int64
```

Observation:

- To confirm if there are missing data, we used .isna() function to check. We can see that there is indeed missing data in the columns Open and Low.

```python
ibm_dataset.dropna(inplace = True)
```

```python
ibm_dataset.isna().sum()
```

```
Open      0
High      0
Low       0
Close     0
Volume    0
Name      0
dtype: int64
```

Observation:

- By using the .dropna() function, all the Nan values will be dropped automatically. We can check the updated dataset and we can see that the values are now 0.

```python
ibm_dataset.head()
```

	Open	High	Low	Close	Volume	Name
Date						
2006-01-03	82.45	82.55	80.81	82.06	11715200	IBM
2006-01-04	82.20	82.50	81.33	81.95	9840600	IBM
2006-01-05	81.40	82.90	81.00	82.50	7213500	IBM
2006-01-06	83.95	85.03	83.41	84.95	8197400	IBM
2006-01-09	84.10	84.25	83.38	83.73	6858200	IBM

Observation: We can look at the .head() of the dataset which shows the first 5 rows. We can observe that the values for Open, High, Close and Low have decimal values. Also the Values in Open, High, Low, CLose are close to each other. For the volume, it is a large number but with no decimal value. Lastly, for the name, it shows the name of the stock.

```
ibm_dataset.drop(["Name"], axis = 1,inplace = True)
```

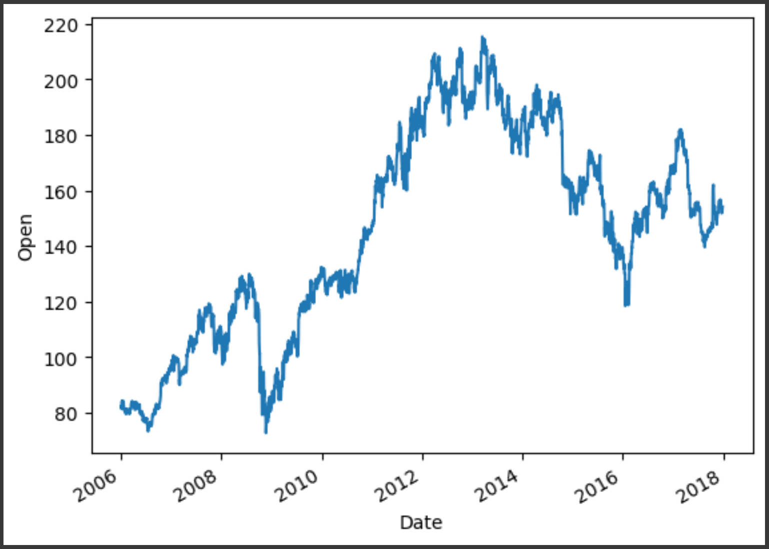
```
ibm_dataset.describe()
```

	Open	High	Low	Close	Volume
count	3019.000000	3019.000000	3019.000000	3019.000000	3.019000e+03
mean	145.515545	146.682319	144.471597	145.617592	5.773770e+06
std	37.554946	37.619664	37.477641	37.535600	3.193255e+06
min	72.740000	73.940000	69.500000	71.740000	2.542560e+05
25%	116.405000	117.750000	115.490000	116.520000	3.622178e+06
50%	149.610000	150.410000	148.470000	149.330000	4.931203e+06
75%	178.445000	179.775000	177.330000	178.690000	6.966642e+06
max	215.380000	215.900000	214.300000	215.800000	3.077428e+07

Observation: The provided data showcases statistics for the columns Open, High, Low, Close, and Volume. Looking at the descriptive statistics, we notice that the values for Open, High, Low, and Close exhibit decimal values, suggesting precision in the data. Additionally, these values appear to be relatively close to each other, indicating similarity in their distributions.

Visualizing the dataset

```
ibm_dataset["Open"].plot()
plt.xlabel("Date")
plt.ylabel("Open")
plt.show()
```



Observation:

- By observing the graph above, we can oserve that there are really no obvious trend with this dataset, even if we look at the seasonality of the dataset, it exhibits no seasonality. This shows that stocks are really hard to predict and it may depend on several factors.

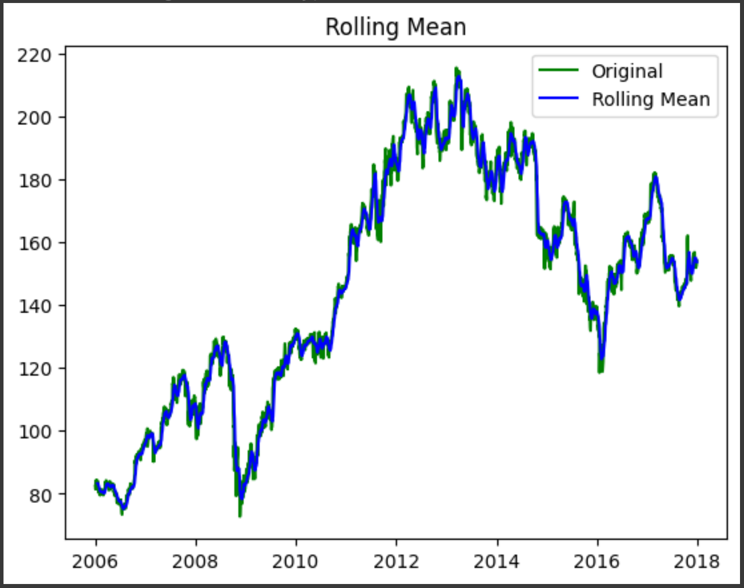
Checking stationarity

```
#H0 = The dataset is non-stationary
#H1 = The dataset is stationary

def stationarity_check(ts):
    roll_mean = ts.rolling(12).mean()
    plt.plot(ts, color='green',label='Original')
    plt.plot(roll_mean, color='blue', label='Rolling Mean')
    plt.legend(loc='best')
    plt.title('Rolling Mean')
    # Perform Augmented Dickey-Fuller test:
    print('Augmented Dickey-Fuller test:')
    df_test = adfuller(ts)
    df_output = pd.Series(df_test[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in df_test[4].items():
        df_output['Critical Value (%s)'%key] = value
    print(df_output)
    if df_test[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary ")
```

```
stationarity_check(ibm_dataset["Open"])
```

```
Augmented Dickey-Fuller test:
Test Statistic                   -1.749373
p-value                           0.405870
#Lags Used                        7.000000
Number of Observations Used    3011.000000
Critical Value (1%)              -3.432524
Critical Value (5%)              -2.862500
Critical Value (10%)             -2.567281
dtype: float64
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```



Observation:

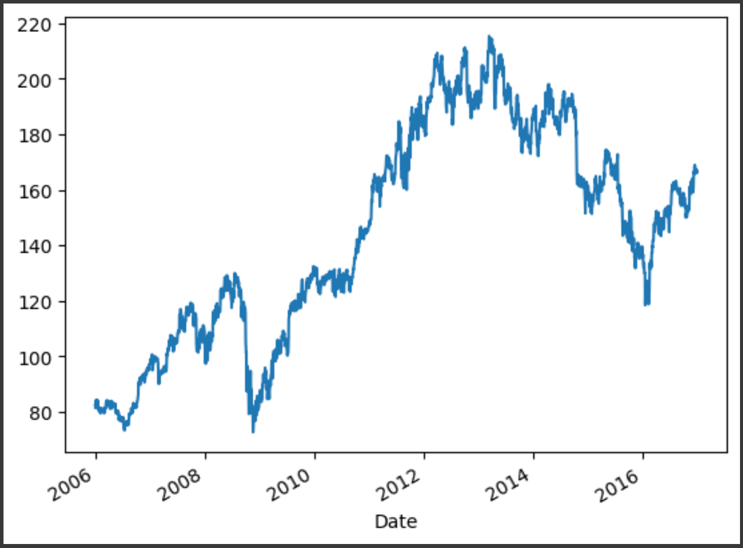
- By checking the stationarity of the dataset, we can determine if this dataset is stationary or non-stationary. By looking at the result, we can see that the p-value is less than 0.5 so this means that the data is stationary. When a data is stationary, this means that there is little to no trent or seasonality to the dataset [3].

Split dataset into train and test set

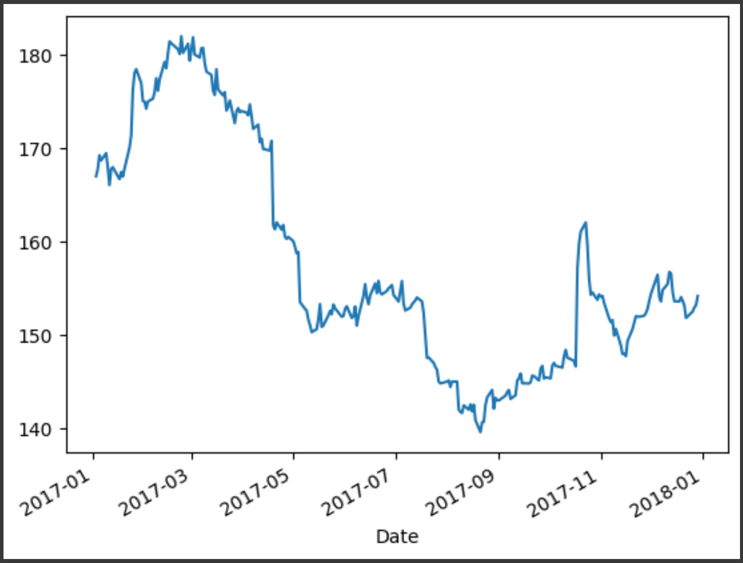
```
train_set = ibm_dataset[ibm_dataset.index.year < 2017]
```

```
test_set = ibm_dataset[ibm_dataset.index.year >= 2017]
```

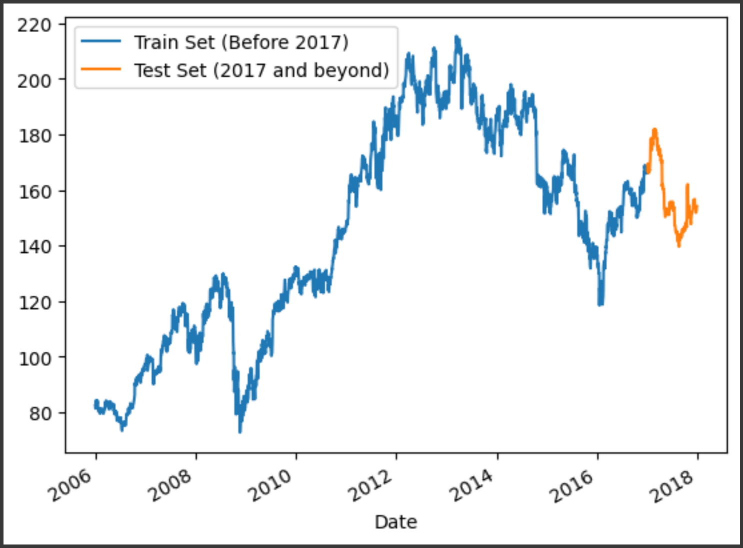
```
train_set["Open"].plot()
plt.show()
```



```
test_set["Open"].plot()
plt.show()
```



```
train_set["Open"].plot()
test_set["Open"].plot()
plt.legend(["Train Set (Before 2017)", "Test Set (2017 and beyond)"])
plt.show()
```



Observation: Here, we can observe the process of splitting the dataset by basing it on the dates. We are tasked to split the dataset to training and test set. The training set comprises of data before the year 2017 while the test set comprises of the data after 2017.

Data Preprocessing

```
scaler = MinMaxScaler()
scaled_train = scaler.fit_transform(train_set)
print(scaled_train)
```

```
[[0.06807347 0.06065089 0.07810773 0.07163682 0.35122061]
 [0.06632081 0.06029868 0.0816989  0.07087325 0.2874084 ]
 [0.06071228 0.06311637 0.07941989 0.0746911  0.19798075]
 ...
 [0.66285754 0.66074951 0.66643646 0.6556296  0.01225609]
 [0.65395401 0.65546633 0.66643646 0.65847564 0.00905772]
 [0.65689849 0.6534235  0.66298343 0.65424129 0.05294548]]
```

```
X_train = []
y_train = []
for i in range(60,2769):
    X_train.append(scaled_train[i-60:i,0])
    y_train.append(scaled_train[i,0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

```
X_train.shape
```

```
(2709, 60)
```

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

```
scaled_test = scaler.transform(test_set)
print(scaled_test)
```

```
[[0.66082445 0.66166526 0.66650552 0.66257115 0.05231485]
 [0.66622266 0.67575373 0.67582873 0.67694016 0.06753546]
 [0.67659843 0.6723725  0.67513812 0.67305289 0.04373673]
 ...
 [0.56232473 0.5581854  0.57396409 0.56497293 0.02559167]
 [0.5640774  0.56480699 0.57803867 0.57128974 0.04391792]
 [0.57087773 0.56903353 0.57955801 0.56698598 0.06568553]]
```

```
scaled_test.shape
```

```
(250, 5)
```

```
X_test = []
y_test = []
for i in range(60,250):
    X_test.append(scaled_test[i-60:i,0])
    y_test.append(scaled_test[i,0])
X_test, y_test = np.array(X_test), np.array(y_test)
```

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

Observation:

- Here, we were tasked to use MinMaxScaler() to standardize the dataset, then create a training and testing set with 60 timesteps. To do that, we can use a for loop which appends data to the predefined arrays. For every 1 y, there will be 60 x, this means that for every output there are 60 previous inputs. I also reshaped the training and test set to have a better model.

Model Training (Additional Task)

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout

model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(X_train.shape[1],1)))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')
model.fit(X_train,y_train,epochs=50,batch_size=32)
```

```
Epoch 18/50
85/85 [==============================] - 2s 26ms/step - loss: 0.0028
Epoch 19/50
85/85 [==============================] - 3s 39ms/step - loss: 0.0028
Epoch 20/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0029
Epoch 21/50
85/85 [==============================] - 2s 24ms/step - loss: 0.0026
Epoch 22/50
85/85 [==============================] - 2s 29ms/step - loss: 0.0025
Epoch 23/50
85/85 [==============================] - 3s 33ms/step - loss: 0.0025
Epoch 24/50
85/85 [==============================] - 4s 45ms/step - loss: 0.0026
Epoch 25/50
85/85 [==============================] - 4s 46ms/step - loss: 0.0024
Epoch 26/50
85/85 [==============================] - 5s 54ms/step - loss: 0.0024
Epoch 27/50
85/85 [==============================] - 3s 40ms/step - loss: 0.0022
Epoch 28/50
85/85 [==============================] - 2s 24ms/step - loss: 0.0023
Epoch 29/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0022
Epoch 30/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0021
Epoch 31/50
85/85 [==============================] - 3s 31ms/step - loss: 0.0022
Epoch 32/50
85/85 [==============================] - 2s 28ms/step - loss: 0.0021
Epoch 33/50
85/85 [==============================] - 2s 24ms/step - loss: 0.0021
Epoch 34/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0022
Epoch 35/50
85/85 [==============================] - 2s 24ms/step - loss: 0.0019
Epoch 36/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0019
Epoch 37/50
85/85 [==============================] - 3s 32ms/step - loss: 0.0019
Epoch 38/50
85/85 [==============================] - 2s 28ms/step - loss: 0.0020
Epoch 39/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0019
Epoch 40/50
85/85 [==============================] - 2s 24ms/step - loss: 0.0017
Epoch 41/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0019
Epoch 42/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0019
Epoch 43/50
85/85 [==============================] - 3s 33ms/step - loss: 0.0018
Epoch 44/50
85/85 [==============================] - 2s 26ms/step - loss: 0.0018
Epoch 45/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0019
Epoch 46/50
85/85 [==============================] - 2s 23ms/step - loss: 0.0019
Epoch 47/50
```

Observation:

- The provided code snippet demonstrates the construction and training of a Long Short-Term Memory (LSTM) neural network using the Keras library. Initially, a Sequential model is created, indicating a linear stack of layers. Within this model, an LSTM layer with 50 units and a Rectified Linear Unit (ReLU) activation function is added. The input shape is determined by the shape of the training data, likely formatted as (number of time steps, number of features). To prevent overfitting, a Dropout layer with a dropout rate of 0.2 is incorporated after the LSTM layer. Following this, a Dense layer with one unit is added, implying a single output neuron.
- The model is then compiled with the Adam optimizer and Mean Squared Error (MSE) loss function. Subsequently, it undergoes training using the fit() function with the training data X_train and y_train, with 50 epochs and a batch size of 32. This configuration suggests a relatively simple LSTM model trained for regression tasks, where it learns to predict a continuous value based on input sequences [3].

Model Evaluation

```
y_pred = model.predict(X_test)
```

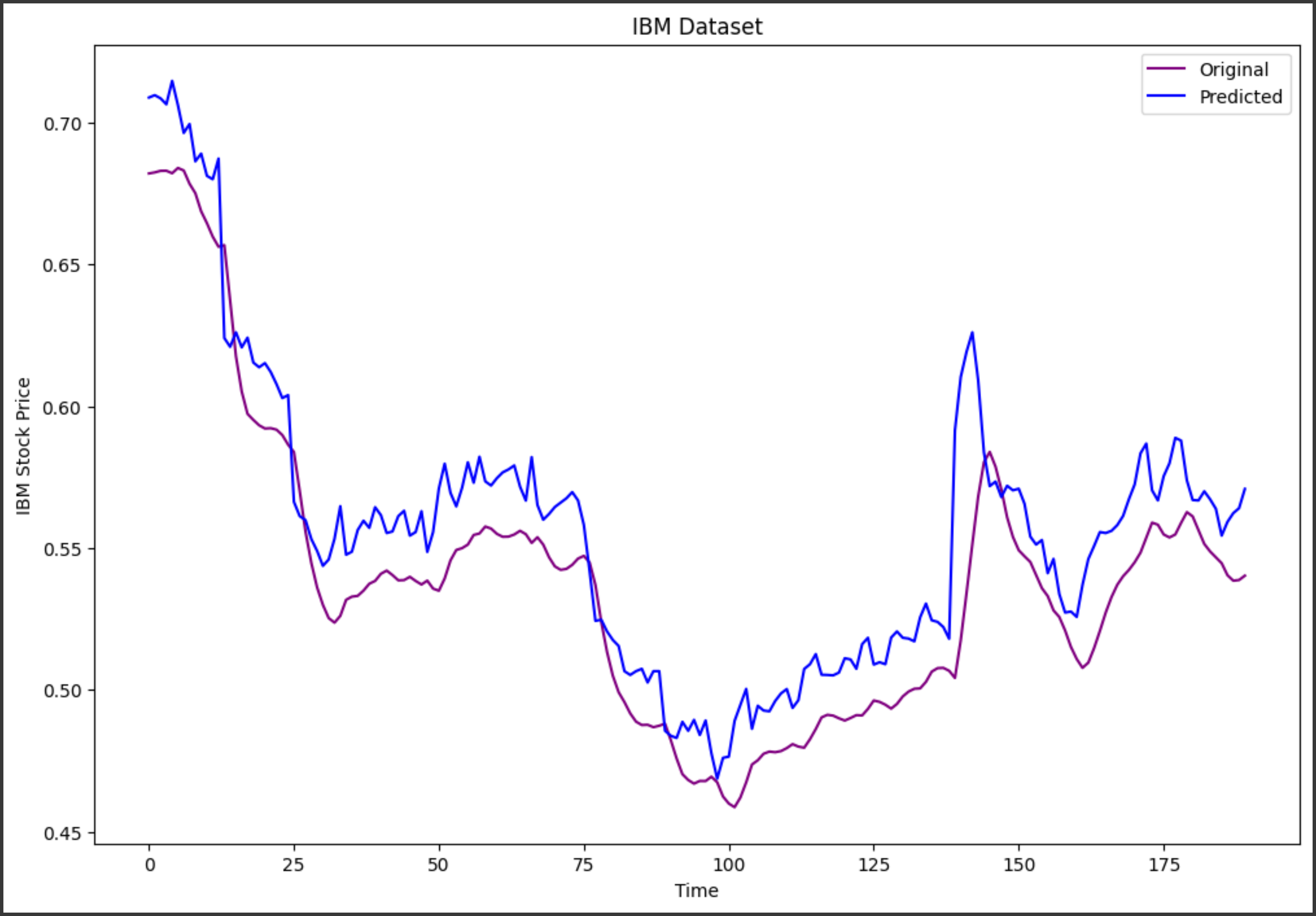
```
6/6 [==============================] - 0s 8ms/step
```

```
mean_mape = tf.reduce_mean(tf.abs(y_test - y_pred) / tf.abs(y_test)) * 100

print("Mean MAPE:", mean_mape.numpy())
```

```
Mean MAPE: 10.687273258015844
```

```
plt.figure(figsize=(12,8))
plt.plot(y_pred,color='purple',label='Original')
plt.plot(y_test,color='blue',label='Predicted')
plt.title('IBM Dataset')
plt.xlabel('Time')
plt.ylabel('IBM Stock Price')
plt.legend();
```



Observation:

- The code above calculates the Mean Absolute Percentage Error (MAPE) for a machine learning model's predictions. MAPE is a metric commonly used to evaluate the accuracy of regression models. In this case, the mean MAPE is computed as the average of the absolute percentage differences between the true values (y_test) and the predicted values (y_pred), expressed as a percentage [5].
- The calculated mean MAPE value is approximately 10.69%, indicating the average percentage deviation between the model's predictions and the actual values. This suggests that, on average, the model's predictions deviate from the true values by around 10.69%. A lower MAPE value generally indicates better accuracy, with 0% representing perfect predictions. Therefore, in this context, a mean MAPE of approximately 10.69% suggests that the model performs reasonably well in terms of accuracy, but there is still room for improvement.

Summary

- In this activity, we were tasked to prepare a dataset for LSTM and CNN training. To start off with that, I imported the dataset from my github and indicated in the read_csv that the index will be the date time. After that, I performed Exploratory Data Analysis to check the dataset for missing values, there are indeed missing values so I removed them.
- Then I checked the descriptive statistics using the .describe() function and found out that the data is very clustered which is indicated by the std.
- After that, I checked the dataset by visualizing it and observed that the there are no obvious trend or seasonality. I tested it using the dicky fuller test to further confirm if the dataset is stationary or not, and I found out that this dataset is indeed non-stationary which means that it does not show any trend or seasonality.
- Then I proceed to data splitting and data preprocessing. For the data splitting, I splitted the dataset into training and testing sets. The training set contains all the data before 2017 while the testing sets contains all the data after 2017. The testing set contains all the data

after 2017. After splitting the dataset, I preprocessed them using MInMaxScaler() and by assigning 60 timesteps for every 1 output y using for loop. After that I reshaped the training and tests to the appropriate shape for better modeling.
- Lastly, I trained an LSTM model with the baseline model setting, and got a very low MSE and MAPE score. This indicates that the baseline model that I created is a well-fitted model.

Conclusion

- I really enjoyed this activity and I hope that there will be more activities like this. I learned a lot about LSTM and how to prepare dataset for that specific model. I also learned new knowledge regarding the modelling around a non-stationary dataset. I was amazed that a model can learn the pattern of a dataset even if there are no visible trends or seasonality. Overall, I hope that there will be more activities like this in the future and I will certainly recommend this to future learners.

References

- [1]“pandas.read_csv — pandas 0.25.3+0.g43013d49f.dirty documentation,” Pydata.org, 2019. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html
- [2]Brownlee, J. (2020, April 30). How to load and explore Time Series data in Python. MachineLearningMastery.com.
- [3]J. Brownlee, “How to Check if Time Series Data is Stationary with Python,” Machine Learning Mastery, Dec. 29, 2016. https://machinelearningmastery.com/time-series-data-stationary-python/
- [4]J. Brownlee, “How to Develop LSTM Models for Time Series Forecasting,” Machine Learning Mastery, Nov. 13, 2018. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
- [5]“Mean Absolute Percentage Error,” Arize AI. https://arize.com/glossary/mean-absolute-percentage-error/#:~:text=Mean%20Absolute%20Percentage%20Error%20is (accessed May 11, 2024).