Stock Price Prediction Using LSTM

STAT 6289 Statistical Deep Learning

Presenter: Xinyi Li

Step1: Upload the Data

```python
# Load the data and inspect them
data = pd.read_csv('TSLA.csv')
data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-07-10	52.444000	52.599998	51.563999	51.830002	51.830002	13054500
1	2015-07-13	52.450001	52.509998	51.209999	52.431999	52.431999	14801500
2	2015-07-14	52.419998	53.198002	52.102001	53.130001	53.130001	9538000
3	2015-07-15	53.348000	53.498001	52.416000	52.627998	52.627998	10108000
4	2015-07-16	52.844002	53.439999	52.632000	53.335999	53.335999	8080000

Choosing to predict the **Closing Price** of each trading day

Other potential options: open price, or adjusted closing price

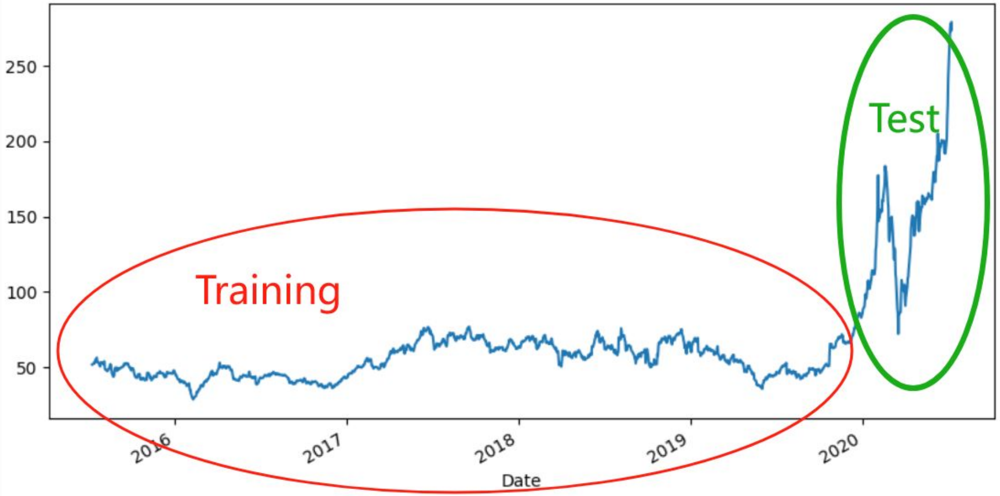
Step2: Split the Data

Why can't we randomly separate the data? Sequences matter; Unlike classification

```
#split the data into training and test sets
#set that training data is about 80%, test is about 20% (as normal setting)
training_size = int(len(data_normalized) * 0.8)
test_size= len(data_normalized)-training_size

train = data_normalized[0:training_size, :]
test = data_normalized[training_size:len(data_normalized), :]
```

```
print(training_size)
print(test_size)
train.shape, test.shape
```

```
1007
252
((1007, 1), (252, 1))
```



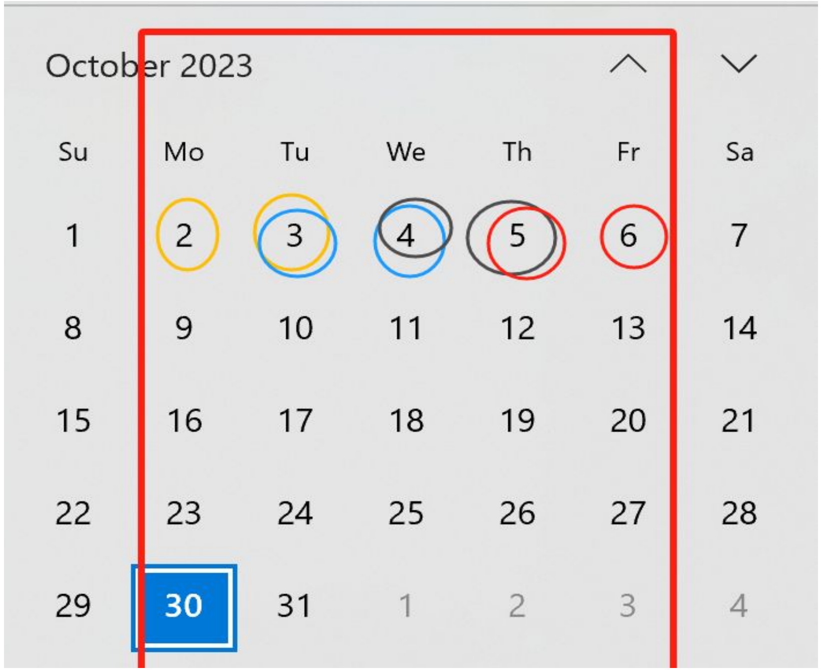
Step3: Input the Time Lags

Date	Open	High	Low	Close	Adj Close	Volume
7/10/2015	52.444	52.6	51.564	51.83	51.83	13054500
7/13/2015	52.45	52.51	51.21	52.432	52.432	14801500
7/14/2015	52.42	53.198	52.102	53.13	53.13	9538000
7/15/2015	53.348	53.498	52.416	52.628	52.628	10108000
7/16/2015	52.844	53.44	52.632	53.336	53.336	8080000
7/17/2015	54.5	55.108	53.65	54.932	54.932	25020500
7/20/2015	55	57.33	54.508	56.452	56.452	24892500
7/21/2015	54.01	54.7	53.31	53.354	53.354	30543500
7/22/2015	52.254	53.888	52.172	53.574	53.574	15525000
7/23/2015	53.93	53.98	53.054	53.44	53.44	11136000
7/24/2015	53.476	54.218	52.784	53.082	53.082	14182500

Not always constant;

Only trading days (weekdays) recorded

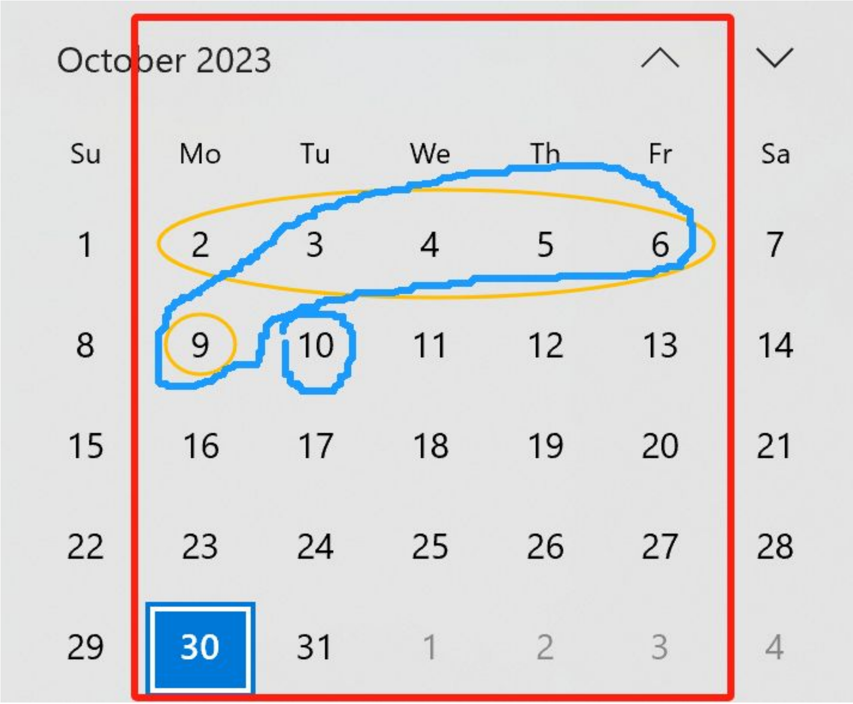
Step3: Input the Time Lags



Use the price of yesterday to predict that of today; use the price of today to predict that of tomorrow

Futurn One Day (Time lag=1)

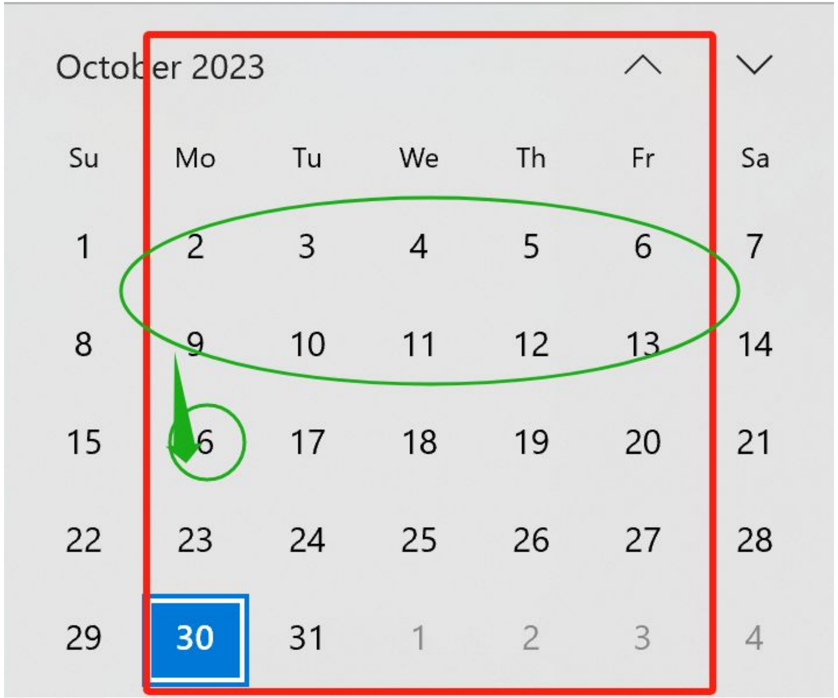
Step3: Input the Time Lags



Use the prices of the previous five days to predict that of today; use the price of previous four days and today to predict that of tomorrow

Futurn One Week (Time lag=5)

Step3: Input the Time Lags



Futurn Two Weeks (Time lag=10)

Step3: Input the Time Lags

```python
#build the input features with different time lags
def input_features(data, lag):
    X, Y = [], []
    for i in range(len(data) -lag):
        X.append(data[i:(i + lag), 0])
        Y.append(data[i + lag, 0])
    return np.array(X), np.array(Y)
```

lag = [1, 5, 10]

Step4: Build the LSTM Model

Choosing the comparisions of **the number of hidden layers** and **drop out rate**

```
#create the LSTM model
def lstm(lstm_layer=4):
  model = Sequential()

  model.add(LSTM(units = 50, activation = 'relu', return_sequences=True,
                 input_shape = (X_train.shape[1], X_train.shape[2])))
  model.add(Dropout(0.2))

  for layer in range(lstm_layer-1):
    model.add(LSTM(50, return_sequences= False))
    model.add(Dropout(0.2))

    model.add(Dense(1))

    return model

for lstm_layer in [4,5,6]:
  model=lstm(lstm_layer)
```

```
#change another drop rate
def lstm2(lstm_layer=4):
  model = Sequential()

  model.add(LSTM(units = 50, activation = 'relu', return_sequences=True,
                 input_shape = (X_train.shape[1], X_train.shape[2])))
  model.add(Dropout(0.3))

  for layer in range(lstm_layer-1):
    model.add(LSTM(50, return_sequences= False))
    model.add(Dropout(0.3))

    model.add(Dense(1))

    return model

for lstm_layer in [4,5,6]:
  model=lstm2(lstm_layer)
```

```
models={'drop_0.2_4_h_layer': lstm(4),'drop_0.2_5_h_layer': lstm(5), 'drop_0.2_6_h_layer': lstm(6),
        'drop_0.3_4_h_layer': lstm2(4),'drop_0.3_5_h_layer': lstm2(5), 'drop_0.3_6_h_layer': lstm2(6)}
```

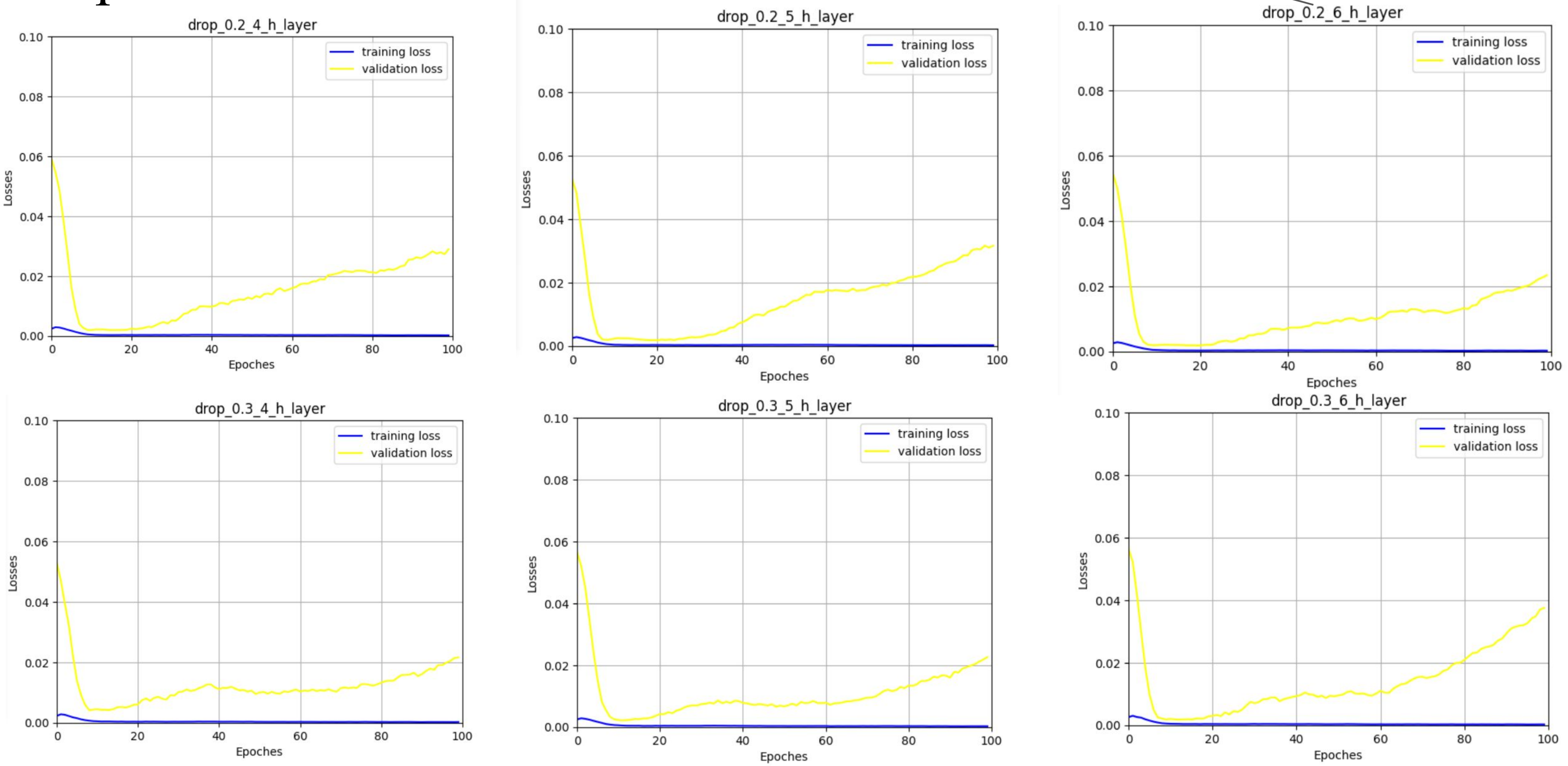
Step4: Build the LSTM Model

Design optimizer and loss function; Set the number of epoches and batch sizes

```
for name, model in models.items():
  model.compile(optimizer = 'adam', loss = 'mse' , metrics="mean_absolute_error")

  history = model.fit(X_train, Y_train, epochs = 100, batch_size = 20,
                      validation_data = (X_test, Y_test), verbose = 1, shuffle = False)
```

Best model

Step4: Build the LSTM Model



Step5: Predict the stock

```
#prediction
Y_prediction = model.predict(X_test)
Y_prediction = scaler.inverse_transform(Y_prediction)
```

```
train_df = data.iloc[:training_size , :]
train_df.Date = pd.to_datetime(train_df.Date, format = '%Y/%m/%d')
test_df = data.iloc[training_size: , :]
test_df.Date = pd.to_datetime(test_df.Date, format = '%Y/%m/%d')
```

test_df.head()

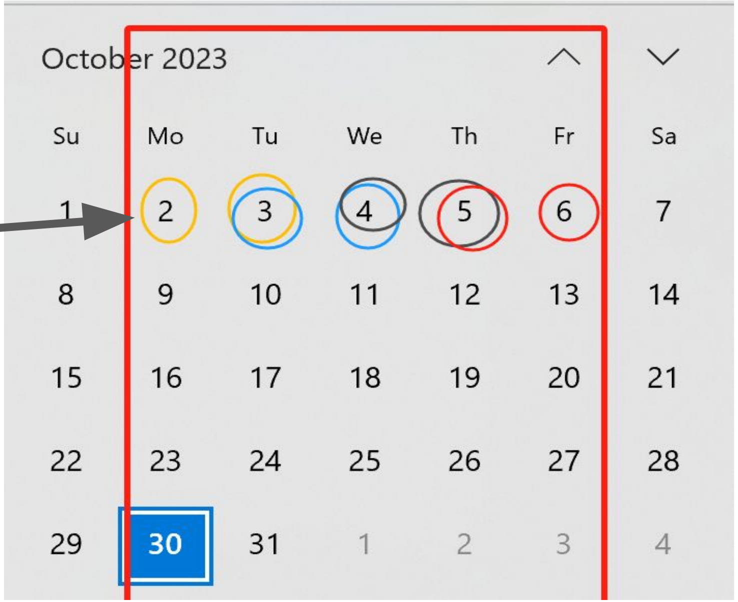
	Date	Close
1007	2019-07-11	47.720001
1008	2019-07-12	49.015999
1009	2019-07-15	50.700001
1010	2019-07-16	50.476002
1011	2019-07-17	50.972000

Step5: Predict the stock

```python
test_df['Prediction'] = np.nan # Initialize new column 'Prediction' with NaN values

test_df.iloc[lag:, 2] = Y_prediction
```

```python
test_df.head()
```

	Date	Close	Prediction
1007	2019-07-11	47.720001	NaN
1008	2019-07-12	49.015999	48.985081
1009	2019-07-15	50.700001	50.010769
1010	2019-07-16	50.476002	51.342285
1011	2019-07-17	50.972000	51.165260

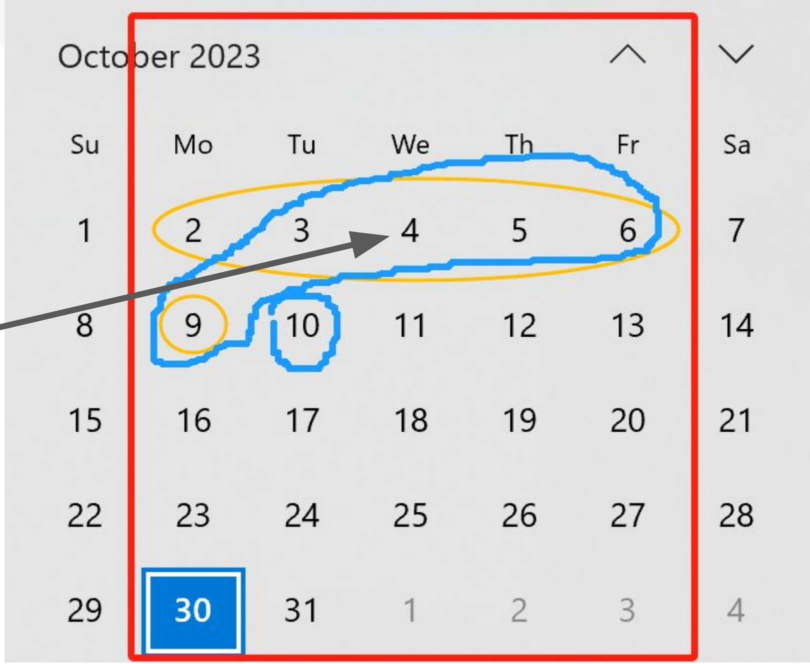


Step5: Predict the stock

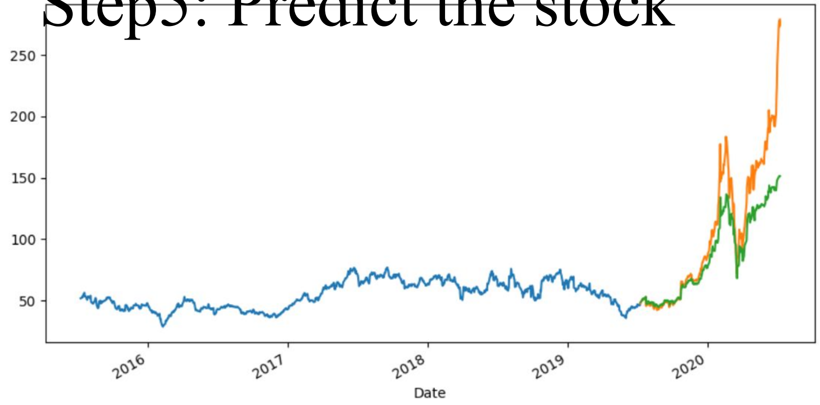
```
test_df['Prediction'] = np.nan # Initialize new column 'Prediction' with NaN values

test_df.iloc[lag:, 2] = Y_prediction
```

```
test_df.head(7)
```

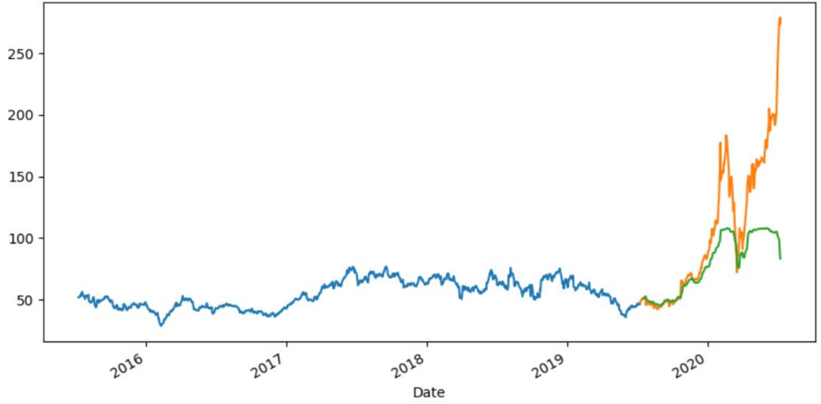
	Date	Close	Prediction
1007	2019-07-11	47.720001	NaN
1008	2019-07-12	49.015999	NaN
1009	2019-07-15	50.700001	NaN
1010	2019-07-16	50.476002	NaN
1011	2019-07-17	50.972000	NaN
1012	2019-07-18	50.708000	50.954666
1013	2019-07-19	51.636002	51.234512



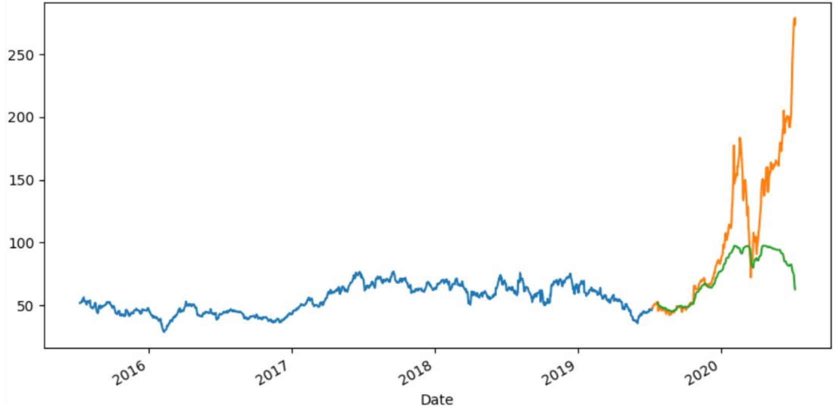
Step5: Predict the stock



Futurn One Day Prediction (Time lag=1)



Future One Week Prediction (Time lag=5)



Future Two Weeks Prediction (Time lag=10)