STOCK PRICE PREDICTION

FEATURE ENGINEERING, MODEL TRAINING AND EVALUATION

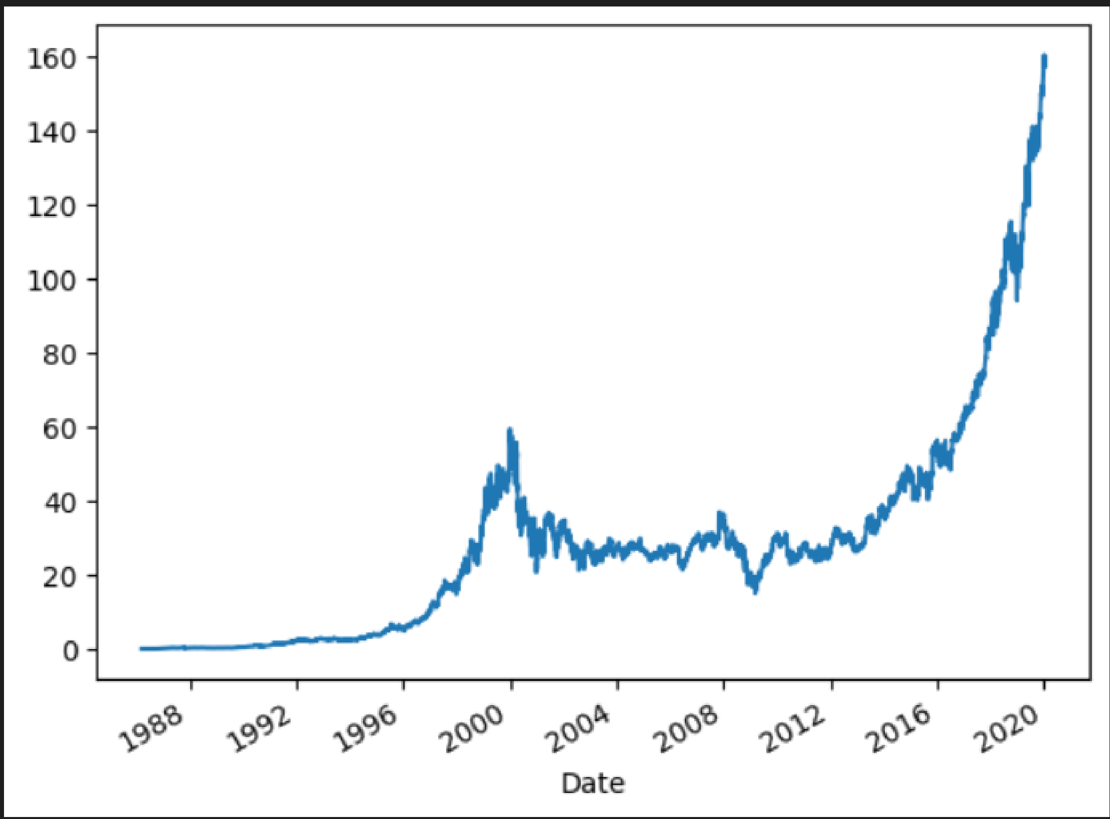
PHASE 4: DEVELOPMENT PART 2

Step 1: Plotting the Close value

```
df['Close'].plot()
```

✓ 0.2s

```
<AxesSubplot: xlabel='Date'>
```



Explanation:

To plot the value of the close price of total time period of stock for acknowledgement

Step 2: select the feature and variables

```python
var = pd.DataFrame(df['Close'])
```

[69] ✓ 0.0s

```python
features = ['Open', 'High', 'Low', 'Volume']
```

[68] ✓ 0.0s

Explanation:

The column for output is assigned to target variables .The feature is being serve as the independent variable to the dependent variable.

On those Open, High, Low, Volume as features

Step 3: Data normalization

Explanation: normalizing the data for better efficient to model. By scaling the data. The values of each feature are scaled to a specific range, typically between 0 and 1

```python
scaler = MinMaxScaler()
feature_transform = scaler.fit_transform(df[features])
feature_transform= pd.DataFrame(columns=features, data=feature_transform, index=df.index)
feature_transform.head()
```

[308] 0.0s

	Open	High	Low	Volume
Date				
1986-03-13	0.000000	0.000059	0.000000	1.000000
1986-03-14	0.000054	0.000065	0.000055	0.297096
1986-03-17	0.000076	0.000070	0.000077	0.127119
1986-03-18	0.000087	0.000070	0.000066	0.063588
1986-03-19	0.000071	0.000054	0.000055	0.044285

Step 4: Creating a test set, training set and processing the data for model

```python
timesplit= TimeSeriesSplit(n_splits=10)
for train_index, test_index in timesplit.split(feature_transform):
        X_train, X_test = feature_transform[:len(train_index)], feature_transform[len(train_index): (len(train_index)+len(test_index))]
        y_train, y_test = var[:len(train_index)].values.ravel(), var[len(train_index): (len(train_index)+len(test_index))].values.ravel()
```

[73] Python

+ Code + Markdown

Explanation: splitting the data into training set and test set for the model and split the 10% of the data for the test sets. And 90% of data for training the model

```python
trainX =np.array(X_train)
testX =np.array(X_test)
X_train = trainX.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = testX.reshape(X_test.shape[0], 1, X_test.shape[1])
```

Explanation: model can interact with data in its format.

The data is being reshaped as arrays by numpy modules.

Step 5: Building the model and Training the model

```
lstm = Sequential()
lstm.add(LSTM(25, input_shape=(1, trainX.shape[1]), activation='relu', return_sequences=False))
lstm.add(Dense(1))
```

57]

Explanation: for a model we use an sequential keras with LSTM. The LSTM has 25 units and Dense layer of one neuron.

```
lstm.compile(loss='mean_squared_error', optimizer='adam')
```

[283] ✓ 0.0s

Explanation: then compiling the model by adam optimizer and mean squared error.

Step 6: Training the LSTM model

```
history=lstm.fit(X_train, y_train, epochs=25, batch_size=8, verbose=1, shuffle=False)
```

```
[572] ✓ 56.9s

Epoch 1/25
969/969 [==============================] - 4s 3ms/step - loss: 99.7876
Epoch 2/25
969/969 [==============================] - 2s 2ms/step - loss: 69.0783
Epoch 3/25
969/969 [==============================] - 2s 2ms/step - loss: 46.0799
Epoch 4/25
969/969 [==============================] - 2s 3ms/step - loss: 22.5158
Epoch 5/25
969/969 [==============================] - 2s 2ms/step - loss: 8.1545
Epoch 6/25
969/969 [==============================] - 2s 2ms/step - loss: 2.2912
Epoch 7/25
969/969 [==============================] - 2s 2ms/step - loss: 0.7796
Epoch 8/25
969/969 [==============================] - 2s 3ms/step - loss: 0.4146
Epoch 9/25
969/969 [==============================] - 2s 2ms/step - loss: 0.2907
Epoch 10/25
969/969 [==============================] - 2s 2ms/step - loss: 0.2338
Epoch 11/25
969/969 [==============================] - 2s 2ms/step - loss: 0.2036
Epoch 12/25
969/969 [==============================] - 2s 2ms/step - loss: 0.1864
Epoch 13/25
...
Epoch 24/25
969/969 [==============================] - 2s 2ms/step - loss: 0.1239
Epoch 25/25
969/969 [==============================] - 2s 2ms/step - loss: 0.1215
```

Explanation: using the fit() function to train the LSTM model on the data for epochs with batch size of 8

Step 7: LSTM prediction for data

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

```
20] ✓ 0.3s

25/25 [==============================] - 0s 2ms/step
```

Explanation: Using the LSTM model on the test data set for the data prediction.

Step 8: Evaluations of data

evaluation metrics to measure the accuracy and reliability of regression model's prediction.

MEAN ABSOLUTE ERROR (MAE): Measures the average absolute difference between the predicted and actual values. Smaller MAE values indicate better accuracy

```
mae = np.mean(np.abs(np.array(y_test) - np.array(y_pred)))
print("Mean Absolute Error:", mae)
```

[616] ✓ 0.0s

```
Mean Absolute Error: 28.62626750551688
```

MEAN SQUARED ERROR (MSE): Measures the average squared difference between the predicted and actual values. It penalizes larger errors more heavily and provides insight into the model's ability to capture variations in the data

```
mse = np.mean((np.array(y_test) - np.array(y_pred))**2)
print("Mean Squared Error:", mse)
```

[617] ✓ 0.0s

```
Mean Squared Error: 1250.4298832807494
```

ROOT MEAN ERROR (RMSE): The square root of MSE, which provides an interpretable metric in the same units as the target variable.

```
mse = np.mean((np.array(y_test) - np.array(y_pred_np))**2)

rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
[618]  ✓ 0.0s

... Root Mean Squared Error (RMSE): 35.361418004383665
```

R-SQUARED (R^2): Also known as the coefficient of determination, R^2 quantifies the proportion of variance in the target variable explained by the model. A higher R^2 indicates a better fit to the data

```
r_squared = r2_score(y_test,y_pred)
print("R-squared (R²):", r_squared)
[619]  ✓ 0.0s

... R-squared (R²): 0.9385675860791486
```

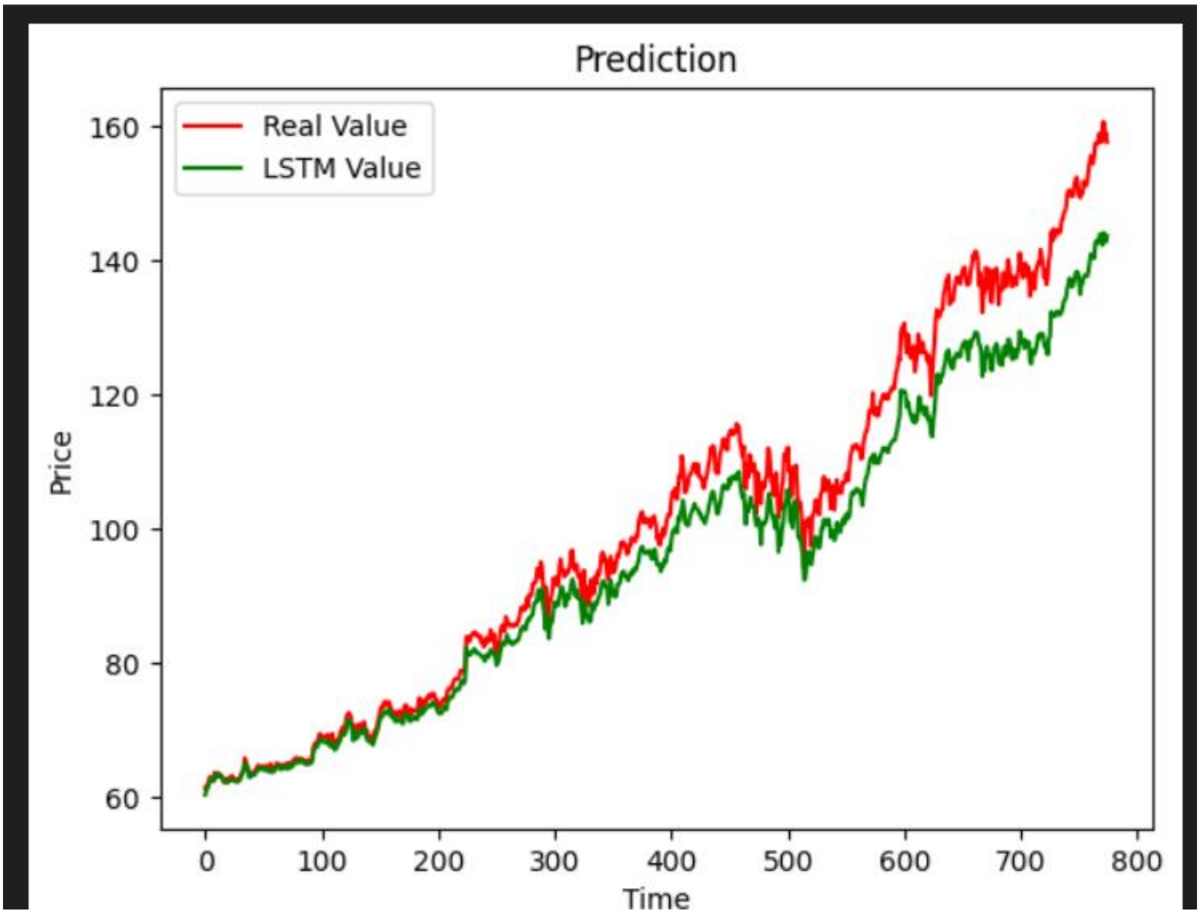
Step 9: Comparing the data by visually

```
plt.plot(y_test,color='red', label='Real Value')
plt.plot(y_pred,color='green', label='LSTM Value')
plt.title("Prediction")
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()

[74]  ✓ 0.2s
```

Explanation: plotting the data from the prediction.

FINAL COMPARISON



CONCLUSION:

Prediction of stock price by the data science and machine learning algorithm. for analyzing the stock value. provided data of stocks daily value of open, close, high, low , volume of market. By data science and machine learning that we can efficiently and easily analysis the data in various model for prediction of the market stocks by fact and statistically. and evaluate the variations.