**Phase 4: Development Part 2**

**Dataset Link:**

<https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset>

**Feature Engineering:**

Feature engineering is a critical part of building accurate and effective LSTM models for various tasks, including stock price prediction. While LSTM networks can capture temporal patterns from raw data, feature engineering can help enhance the model's ability to learn and extract meaningful information. Here are some feature engineering techniques you can apply when working with LSTM models:

* **Lag Features:**

Create lag features by shifting your target variable (e.g., stock price) to previous time steps. These lag features can help the model learn from historical data and capture autocorrelations. For instance, you can use the closing price from the past 'n' days as features for predicting the future price.

* **Moving Averages:**
* Compute and include various moving averages (e.g., simple moving average, exponential moving average) over different time windows. These moving averages can capture trends and smooth out noise in the data.
* **Technical Indicators:**

**-** Integrate popular technical indicators such as Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Bollinger Bands, Stochastic Oscillator, and others. These indicators provide additional information about market sentiment and momentum

* **Volatility Indicators:**
* Include indicators that measure market volatility, like Average True Range (ATR) or historical volatility, to help the model adapt to changing market conditions.
* **Price Rate of Change:**
* Calculate the rate of change of prices over a given time period. This can help the model understand the velocity of price movements.
* **Feature Scaling:**
* Ensure that all engineered features are scaled or normalized to a similar range to prevent any one feature from dominating the model's learning process.
* **Model Training:**
* - Start with a basic LSTM model and gradually increase complexity if needed.
* - Consider stacking multiple LSTM layers or using Bidirectional LSTMs for capturing more complex patterns.
* - Experiment with other recurrent layers like GRU or combinations of LSTM and GRU.
* **Hyperparameter Tuning:**

- Adjust hyperparameters like the number of hidden units, dropout rates, and learning rate. Grid search or random search can be helpful.

* **Sequence Length:**

- Determine the optimal sequence length for input data. Longer sequences may capture more context but can lead to vanishing gradient problems

**Batch Size:**

- Experiment with batch sizes; smaller batches can lead to faster convergence, while larger batches can provide better generalization.

* **Learning Rate:**
* - Fine-tune the learning rate. A smaller learning rate can help in fine-grained convergence, while a larger one can lead to faster initial training.
* **Regularization:**
* - Implement regularization techniques like L1 or L2 regularization to prevent overfitting.
* **Initializations:**
* - Try different weight initialization methods, such as Xavier or He initialization, to ensure a balanced learning process.

**PROGRAM:**

*# Make a new tech returns DataFrame*

tech\_rets **=** closing\_df**.**pct\_change()

tech\_rets**.**head()

*# Comparing Google to itself should show a perfectly linear relationship*

sns**.**jointplot(x**=**'GOOG', y**=**'GOOG', data**=**tech\_rets, kind**=**'scatter', color**=**'seagreen')

*# We can simply call pairplot on our DataFrame for an automatic visual analysis*

*# of all the comparisons*

sns**.**pairplot(tech\_rets, kind**=**'reg')

*# Set up our figure by naming it returns\_fig, call PairPLot on the DataFrame*

return\_fig **=** sns**.**PairGrid(tech\_rets**.**dropna())

*# Using map\_upper we can specify what the upper triangle will look like.*

return\_fig**.**map\_upper(plt**.**scatter, color**=**'purple')

*# We can also define the lower triangle in the figure, inclufing the plot type (kde)*

*# or the color map (BluePurple)*

return\_fig**.**map\_lower(sns**.**kdeplot, cmap**=**'cool\_d')

*# Finally we'll define the diagonal as a series of histogram plots of the daily return*

return\_fig**.**map\_diag(plt**.**hist, bins**=**30)

*# Set up our figure by naming it returns\_fig, call PairPLot on the DataFrame*

returns\_fig **=** sns**.**PairGrid(closing\_df)

*# Using map\_upper we can specify what the upper triangle will look like.*

returns\_fig**.**map\_upper(plt**.**scatter,color**=**'purple')

*# We can also define the lower triangle in the figure, inclufing the plot type (kde) or the color map (BluePurple)*

returns\_fig**.**map\_lower(sns**.**kdeplot,cmap**=**'cool\_d')

*# Finally we'll define the diagonal as a series of histogram plots of the daily return*

returns\_fig**.**map\_diag(plt**.**hist,bins**=**30)

plt**.**figure(figsize**=**(12, 10))

plt**.**subplot(2, 2, 1)

sns**.**heatmap(tech\_rets**.**corr(), annot**=True**, cmap**=**'summer')

plt**.**title('Correlation of stock return')

plt**.**subplot(2, 2, 2)

sns**.**heatmap(closing\_df**.**corr(), annot**=True**, cmap**=**'summer')

plt**.**title('Correlation of stock closing price')

rets **=** tech\_rets**.**dropna()

area **=** np**.**pi **\*** 20

plt**.**figure(figsize**=**(10, 8))

plt**.**scatter(rets**.**mean(), rets**.**std(), s**=**area)

plt**.**xlabel('Expected return')

plt**.**ylabel('Risk')

**for** label, x, y **in** zip(rets**.**columns, rets**.**mean(), rets**.**std()):

plt**.**annotate(label, xy**=**(x, y), xytext**=**(50, 50), textcoords**=**'offset points', ha**=**'right', va**=**'bottom',

arrowprops**=**dict(arrowstyle**=**'-', color**=**'blue', connectionstyle**=**'arc3,rad=-0.3'))

*# Get the stock quote*

df **=** pdr**.**get\_data\_yahoo('AAPL', start**=**'2012-01-01', end**=**datetime**.**now())

*# Show teh data*

Df

plt**.**figure(figsize**=**(16,6))

plt**.**title('Close Price History')

plt**.**plot(df['Close'])

plt**.**xlabel('Date', fontsize**=**18)

plt**.**ylabel('Close Price USD ($)', fontsize**=**18)

plt**.**show()

*# Create a new dataframe with only the 'Close column*

data **=** df**.**filter(['Close'])

*# Convert the dataframe to a numpy array*

dataset **=** data**.**values

*# Get the number of rows to train the model on*

training\_data\_len **=** int(np**.**ceil( len(dataset) **\*** .95 ))

training\_data\_len

*# Scale the data*

**from** sklearn.preprocessing **import** MinMaxScaler

scaler **=** MinMaxScaler(feature\_range**=**(0,1))

scaled\_data **=** scaler**.**fit\_transform(dataset)

scaled\_data

*# Create the training data set*

*# Create the scaled training data set*

train\_data **=** scaled\_data[0:int(training\_data\_len), :]

*# Split the data into x\_train and y\_train data sets*

x\_train **=** []

y\_train **=** []

**for** i **in** range(60, len(train\_data)):

x\_train**.**append(train\_data[i**-**60:i, 0])

y\_train**.**append(train\_data[i, 0])

**if** i**<=** 61:

print(x\_train)

print(y\_train)

print()

*# Convert the x\_train and y\_train to numpy arrays*

x\_train, y\_train **=** np**.**array(x\_train), np**.**array(y\_train)

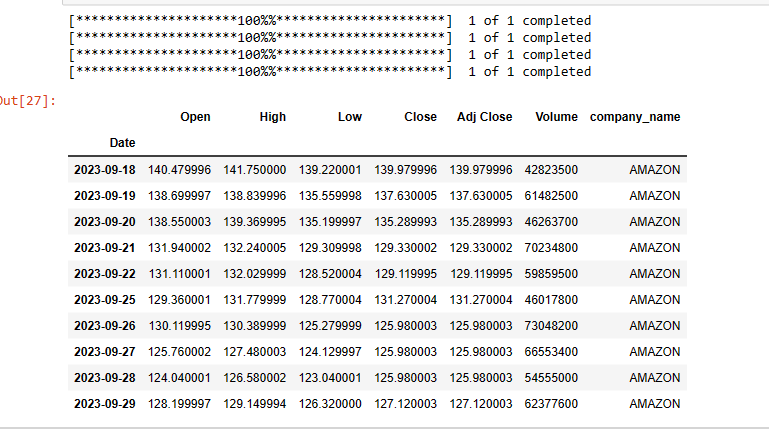
*# Reshape the data*

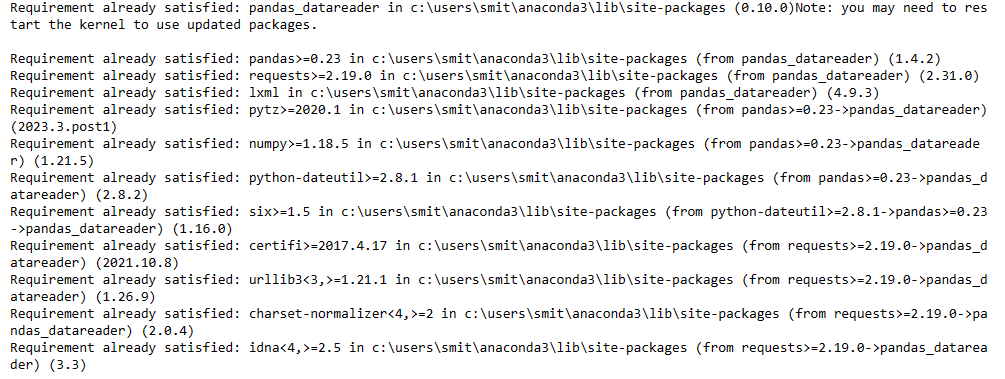
x\_train **=** np**.**reshape(x\_train, (x\_train**.**shape[0], x\_train**.**shape[1], 1))

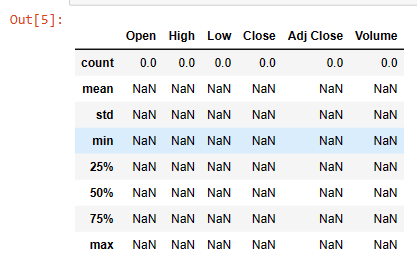
*# x\_train.shape*

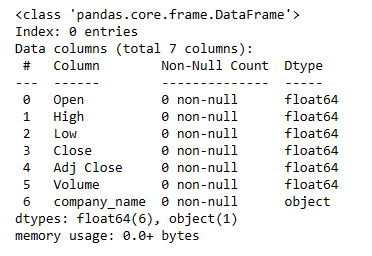
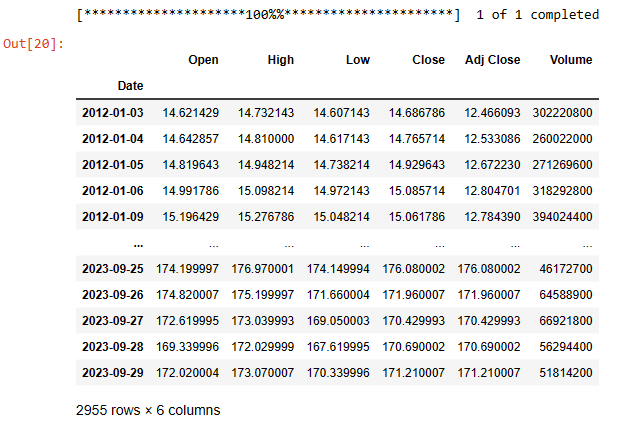
**Output:**

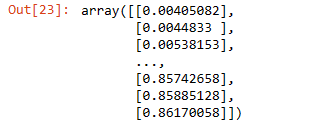


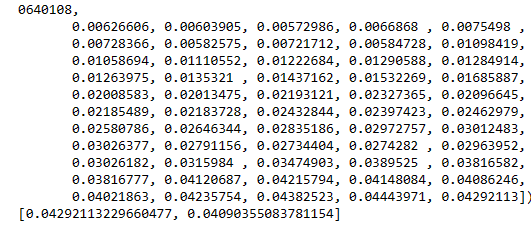


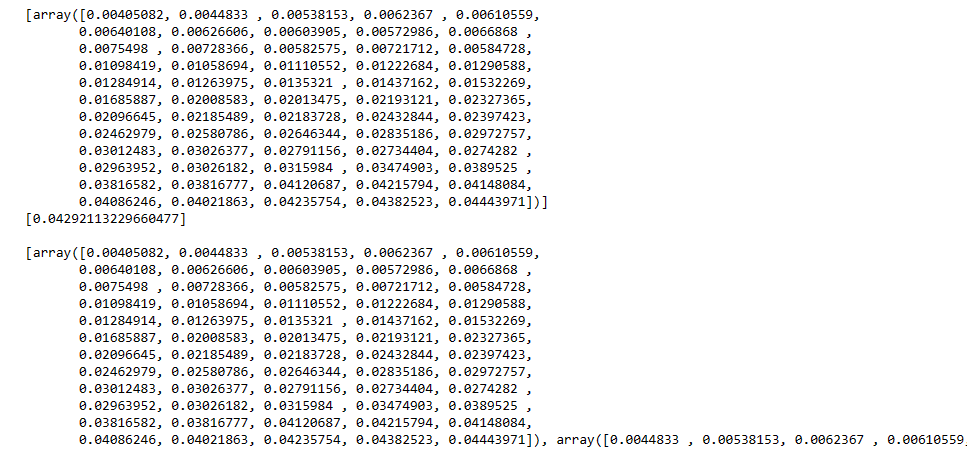












1. **Evaluation:**

Evaluate the model's performance using appropriate metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and visualizations like price predictions vs. actual prices.\