APPLIED DATA SCIENCE
PHASE-4 SUBMISSION
STOCK PRICE PREDICTION

FEATURE ENGINEERING:

Feature engineering is the process of selecting, creating, or transforming variables in a dataset to improve the performance of machine learning models. It involves tasks like selecting relevant features, handling missing data, and creating new variables. Effective feature engineering can significantly impact model accuracy and predictive power, making it a crucial step in data analysis and machine learning.

We are planning to create window statistics of 1 week, 2 week, 1 month and 1 year and so on. Use common and simple statistics like mean, median, max, min and exponentially weighted mean.

```
BASE_FEATURES = ['returnsOpenPrevMktres10',
'returnsOpenPrevRaw10',
'open',
'close']
```

To generate the ratio between these features and market mean.

```
def add_market_mean_col(market_df):
daily_market_mean_df = market_df.groupby('time').mean()
daily_market_mean_df = daily_market_mean_df[['volume', 'close']]
merged_df = market_df.merge(daily_market_mean_df, left_on='time',
right_index=True, suffixes=("",'_market_mean'))
merged_df['volume/volume_market_mean'] = merged_df['volume'] /
merged_df['volume_market_mean']
merged_df['close/close_market_mean'] = merged_df['close'] /
merged_df['close_market_mean']
return merged_df.reset_index(drop = True)
```

```
BASE_FEATURES = BASE_FEATURES + ['volume',
'volume/volume_market_mean',
'close/close_market_mean']
```

In a similar context, the ratio of opening price and closing price should tell us more than just raw closing/opening price

```
def
generate_open_close_ratio(df):
df['open/close']     =     df['open']     /
df['close']

BASE_FEATURES = BASE_FEATURES + ['open/close']
```

Likewise, we can generate the ratio of raw return values to its current opening/closing prices. Note that this is not a duplicate of residual return columns which are the returns after movement of the market as a whole has been accounted for, leaving only movements inherent to the instrument. The generated ratio is not adjusted by market movements it is just a ratio of its price delta versus its price.

```
open_raw_cols            =            ['returnsOpenPrevRaw1',
'returnsOpenPrevRaw10']
close_raw_cols            =            ['returnsClosePrevRaw1',
'returnsClosePrevRaw10']
 def
raw_features_to_ratio_features(df):
for col in open_raw_cols:
df[col + '/open' ] = df[col] / df['open']
for col in close_raw_cols:
df[col + '/close'] = df[col] / df['close']


BASE_FEATURES          =          BASE_FEATURES          +
['returnsClosePrevRaw1/close',
'returnsClosePrevRaw10/close',
'returnsOpenPrevRaw1/open'
'returnsOpenPrevRaw10/open']
```

The previously mentioned window statistics feature is generated based on the BASE_FEATURES we gathered. And merged on time and assetCode(which is like the id column of the market data)

```
new_df = generate_features(market_train_df)
market_train_df = pd.merge(market_train_df, new_df, how =
'left', on = ['time', 'assetCode'])
```

‘Microsoft Corp’ has related ‘assetCodes’ of ‘{‘MSFT.O’, ‘MSFT.F’, ‘MSFT.DE’, ‘MSFT.OQ’}’. These codes are just Microsoft stocks on different stock exchanges. So now it is clear that if one news is related to an asset name all related asset codes will be affected. Also merging table on ‘assetName’ is practically much easier because it is single name of an asset in both market and news data (On the contrary market data has ‘assetCode’ which has a single asset code and news data has ‘assetCodes’ which has more than one asset codes). So now let’s transform sentiment column accordingly and merge market and news data.

```
def merge_with_news_data(market_df, news_df):
news_df['firstCreated'] = news_df.firstCreated.dt.hour
news_df['assetCodesLen'] = news_df['assetCodes'].map(lambda
x: len(eval(x))) news_df['asset_sentiment_count'] =
news_df.groupby(['assetName',
'sentimentClass'])['firstCreated'].transform('count') kcol
= ['time', 'assetName'] news_df = news_df.groupby(kcol,
as_index=False).mean() market_df = pd.merge(market_df,
news_df, how='left', on=kcol, suffixes=("", "_news"))
return market_df
 market_train_df = merge_with_news_data(market_train_df,
news_train_df)
```

MODEL TRAINING:

Model training in stock price prediction involves using historical stock market data to train a machine learning model that can make predictions about future stock prices. Here is a step-by-step explanation of the process:

Data collection:

Gather historical stock market data, including variables such as stock prices, trading volume, technical indicators, news sentiment, and macroeconomic factors for a given time period.

Data preprocessing:

Clean the collected data, handle missing values, and remove outliers. Normalize or scale the data as needed.

Feature engineering:

Select or create relevant features that may have an impact on stock prices. This can involve transforming or combining existing variables to extract meaningful information. For example, using moving averages or creating lagged variables.

Train-test split:

Split the dataset into two parts: a training set and a test set. The training set will be used to train the model, while the test set will be used to evaluate its performance.

Model selection:

Choose an appropriate machine learning algorithm or model for stock price prediction. Some commonly used models include linear regression, support vector machines (SVM), random forests, or deep neural networks.

Hyperparameter tuning:

Fine-tune the model by selecting the best hyperparameters. Hyperparameters are settings that control the learning process of the model, such as learning rate, regularization strength, or the number of hidden layers in a neural network.

Model training:

Fit the chosen model to the training data using the selected features. The model will learn patterns and relationships present in the training data.

Model evaluation:

Assess the performance of the trained model by evaluating its predictions on the test set. Metrics such as mean squared error (MSE), mean absolute error (MAE), or root mean squared error (RMSE) can be used to measure prediction accuracy.

Model refinement:

If the model's performance is not satisfactory, additional steps can be taken to improve it. This may include experimenting with different algorithms, adjusting hyperparameters, adding new features, or gathering more data.

Model deployment:

Once the model is trained and evaluated, it can be deployed to make predictions on new, unseen data. This can involve automating the prediction process and updating the model periodically with new data to enhance its accuracy.

CODE:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

```
[10] data=pd.read_csv('/content/MSFT.csv')
```

```
[11] training_set=data.iloc[:,1:2].values
```

```
[12] from sklearn.preprocessing import MinMaxScaler
     scaler=MinMaxScaler(feature_range=(0,1))
     scaled_training_set=scaler.fit_transform(training_set)
     scaled_training_set
```

```
array([[0.00000000e+00],
       [5.44673742e-05],
       [7.62543239e-05],
       ...,
       [9.92909264e-01],
       [9.85128179e-01],
       [9.99184307e-01]])
```

```
X_train=[]
y_train=[]
for i in range(60,1258):
 X_train.append(scaled_training_set[i-60:i,0])
 y_train.append(scaled_training_set[i,0])
X_train=np.array(X_train)
y_train=np.array(y_train)
```

```
[14] print(X_train.shape)
```

```
(1198, 60)
```

```
[15] from keras.models import Sequential
     from keras.layers import LSTM
     from keras.layers import Dense
     from keras.layers import Dropout
```

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

```
16] regressor=Sequential()
    regressor.add(LSTM(units=50,return_sequences=True,input_shape=(X_train.shape[1],1)))
    regressor.add(Dropout(0.2))
    regressor.add(LSTM(units=50,return_sequences=True))
    regressor.add(Dropout(0.2))
    regressor.add(LSTM(units=50,return_sequences=True))
    regressor.add(Dropout(0.2))
    regressor.add(LSTM(units=50))
    regressor.add(Dropout(0.2))
    regressor.add(Dense(units=1))
```

```
[17] regressor.compile(optimizer='adam',loss='mean_squared_error')
     regressor.fit(X_train,y_train,epochs=10,batch_size=32)

     Epoch 1/10
     38/38 [==============================] - 10s 84ms/step - loss: 3.2573e-05
     Epoch 2/10
     38/38 [==============================] - 3s 80ms/step - loss: 3.7207e-06
     Epoch 3/10
     38/38 [==============================] - 3s 85ms/step - loss: 3.4982e-06
     Epoch 4/10
     38/38 [==============================] - 4s 96ms/step - loss: 3.4549e-06
     Epoch 5/10
     38/38 [==============================] - 3s 80ms/step - loss: 2.9047e-06
     Epoch 6/10
     38/38 [==============================] - 3s 74ms/step - loss: 1.9501e-06
     Epoch 7/10
     38/38 [==============================] - 3s 78ms/step - loss: 1.1707e-06
     Epoch 8/10
     38/38 [==============================] - 3s 91ms/step - loss: 6.0697e-07
     Epoch 9/10
     38/38 [==============================] - 3s 82ms/step - loss: 4.9459e-07
     Epoch 10/10
     38/38 [==============================] - 3s 78ms/step - loss: 6.1307e-07
     <keras.src.callbacks.History at 0x785b64137850>
```

```
[18] import pandas as pd
     data_test=pd.read_csv('/content/MSFT.csv')
     actual_stock_price=data_test.iloc[:,1:2].values
```

EVALUTION:

Stock price prediction is an important aspect of financial analysis. It involves using various techniques and models to forecast the future price movements of a stock or a group of stocks. There are several methods and metrics that can be used to evaluate the accuracy and effectiveness of stock price prediction models. Here are some key evaluation measures:

Mean Squared Error (MSE):

MSE provides a measure of the average squared difference between the predicted and actual stock prices. Lower MSE values indicate better accuracy in predicting stock prices.

Root Mean Squared Error (RMSE):

RMSE is the square root of the MSE and provides a measure of the average difference between the predicted and actual stock prices. It is useful for comparing models across different datasets.

It's important to note that stock price prediction is a challenging task, as it depends on various factors such as market conditions, investor behaviour, and unexpected events. Therefore, it's crucial to apply critical thinking and use multiple evaluation measures to assess the performance of a prediction model. Additionally, backtesting and out-of-sample testing can provide insights into the model's robustness and generalizability.

CODE:

```
[26] def calculate_rmse(y_true,y_pred):
         rmse=np.sqrt(np.mean((y_true-y_pred)**2))
         return rmse
```