APPLIED DATA SCIENCE

PHASE 4

STOCK PRICE PREDICTION

Problem Definition:

The problem is to build a predictive model that forecasts stock prices based on historical market data. The goal is to create a tool that assists investors in making well-informed decisions and optimizing their investment strategies. This project involves data collection, data preprocessing, feature engineering, model selection, training, and evaluation.

As the problem definition of the project tells us that , the main motive of our model is to predict prices of a given stock and then predict its price for the next few days. As this will help the investors in understanding the market using appropriate data points and thus giving them a clear vision in making the right and well informed decisions.

As moving on with our project we will be doing feature engineering model training and model evaluation using CNN algorithm

Feature Engineering:

Feature engineering in data science refers to the process of selecting, creating, or transforming input variables (features) from raw data to improve the performance of machine learning models. Effective feature engineering can have a significant impact on the quality and accuracy of predictive models.

Feature engineering in stock price prediction involves creating and selecting relevant input variables (features) for machine learning models to make more accurate predictions of stock prices. The goal is to transform raw data, such as historical stock prices, into meaningful features that capture relevant patterns and relationships in the data. Here are some common feature engineering techniques used in stock price prediction:

1. Lag Features: One of the most straightforward techniques is to use lag features, which involve using past stock prices as features. For example, you can create features for the previous day's closing price, the closing price from one week ago, or the closing price from one month ago. These lag features can help capture trends and autocorrelation in the data.

2. Moving Averages: Moving averages, such as the simple moving average (SMA) and exponential moving average (EMA), are commonly used features. They help smooth out price data and highlight trends over a specific time period.

3. Volatility Measures: Features that capture volatility, such as standard deviation or average true range, can provide insights into market uncertainty and risk. Volatility features can help predict stock price movements, especially in more volatile markets.

4. Volume-related Features: Trading volume can be crucial in stock price prediction. Features based on trading volume, such as average trading volume over a specific period or volume ratios, can provide valuable information.

5. Technical Indicators: Various technical indicators, such as Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and Bollinger Bands, are often used as features. These indicators can provide signals for potential price movements.

6. Market Sentiment: Sentiment analysis of news articles, social media, or financial reports can be used to create sentiment-related features. These features can capture the market sentiment and its potential impact on stock prices.

7. Economic Indicators: Features related to macroeconomic indicators like GDP growth, inflation rates, interest rates, and unemployment rates can be useful for understanding the broader economic context affecting stock prices.

8. Seasonal and Calendar Features: Features related to specific calendar events or seasonal trends, like earnings reports, holidays, or market seasons, can be relevant for certain stocks.

9. Fundamental Data: Features based on fundamental financial data, such as earnings per share (EPS), price-to-earnings (P/E) ratios, and other financial metrics, can be valuable for fundamental analysis.

10. Market Sentiment and News: Incorporating sentiment analysis of news and social media can help capture public perception and market sentiment about a particular stock or sector.

11. Custom Features: Domain-specific features can be engineered based on your understanding of the stock market and specific trading strategies.

It's important to note that not all features are equally relevant for every stock or market condition. Feature engineering requires domain knowledge and experimentation to determine which features are most useful for a particular stock price prediction task. Additionally, feature engineering should be followed by appropriate model selection and evaluation to ensure the best predictive performance.

Model Training And Evaluation:

Model training and evaluation are crucial steps in the data science workflow, especially when working with machine learning models. These steps involve training a model on a dataset and then assessing its performance to determine how well it can make predictions. Here's an overview of each step:

1. Model Training:

- Data Preparation: Before training a model, you need to prepare your data. This involves cleaning, preprocessing, and transforming the data so that it can be used as input for the machine learning model. This may include tasks like handling missing values, encoding categorical variables, and splitting the data into training and testing sets.

- Model Selection: Choose the appropriate machine learning algorithm or model for your problem. This selection is based on the nature of the problem (classification, regression, clustering, etc.) and the characteristics of the data.

- Feature Engineering: Create relevant features or input variables for the model. This may involve selecting, transforming, or generating features that are most informative for the task.

- Model Training: The model is trained on the training dataset, which involves exposing it to the historical data to learn patterns and relationships between the input features and the target variable. During training, the model adjusts its parameters to minimize a chosen objective or loss function.

2. Model Evaluation:

- Testing and Validation: Once the model is trained, it is tested and validated on a separate dataset called the testing or validation set. This set wasn't used during the training process, and it helps assess how well the model generalizes to unseen data.

- Performance Metrics: Various metrics are used to evaluate the model's performance. The choice of metrics depends on the nature of the problem. For instance, in classification tasks, metrics like accuracy, precision, recall, F1 score, and ROC-AUC are commonly used. In regression tasks, metrics like mean squared error (MSE), root mean squared error (RMSE), and R-squared are often employed.

- Cross-Validation: To ensure robustness of the model, cross-validation techniques can be used. K-fold cross-validation, for example, involves splitting the data into K subsets (folds) and training and evaluating the model K times, each time using a different fold as the validation set.

- Hyperparameter Tuning: Many machine learning models have hyperparameters that need to be tuned for optimal performance. Techniques like grid search or random search can be employed to find the best combination of hyperparameters.

- Overfitting and Underfitting: Evaluate the model for signs of overfitting (when the model performs well on the training data but poorly on new data) and underfitting (when the model fails to capture the underlying patterns). Adjustments may be needed to address these issues.

Model training and evaluation are iterative processes, and the goal is to build a model that not only performs well on the evaluation metrics but also meets the practical needs of the problem at hand.

Model Selection:

LINEAR REGRESSION AND RNN

Linear regression and recurrent neural network (RNN) are two different approaches used in machine learning for stock price prediction, each with its strengths and applications:

LINEAR REGRESSION

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. The key assumption of linear regression is that there exists a linear relationship between the independent variables (predictors) and the dependent variable (the outcome being predicted).

Linear Regression for Stock Price Prediction:

- Linear regression is a statistical method used to model the relationship between a dependent variable (target) and one or more independent variables (features).
- In the context of stock price prediction, it can be used to analyze how the stock price is affected by various factors such as opening price, closing price, trading volume, etc.
- The linear regression model assumes a linear relationship between the input features and the output variable, which might be suitable for simpler, well-behaved data.

- It is particularly useful for providing insights into how one or more independent variables contribute to the overall prediction of the target variable (stock price in this case).

BENEFITS OF LINEAR REGRESSION:

It provides a straightforward way to understand how different features contribute to the prediction. It can serve as a good starting point for analysis and is computationally efficient. However, it might not capture complex patterns in data and can be limited when dealing with non-linear relationships.

RNN ALGORITHM:

RNN, or Recurrent Neural Network, is a type of artificial neural network designed to work with sequence data, making it well-suited for tasks such as natural language processing, time series analysis, and sequential data modeling. Unlike feedforward neural networks, RNNs have a memory component that enables them to exhibit temporal dynamic behavior.

Key characteristics of RNNs include:

1. **Sequential Memory**: RNNs have a form of memory that allows them to process sequences of data, making them effective for tasks that involve sequential or time-series data. This memory is what enables RNNs to remember patterns in the data they have seen previously.

2. **Feedback Loops**: RNNs have a feedback mechanism that allows information to persist, which makes them capable of handling input sequences of varying lengths.

3. **Shared Weights**: RNNs use the same weights across different time steps, allowing them to learn from sequential data and capture temporal dependencies.

   BENEFITS:

   RNNs are well-suited for capturing sequential dependencies and patterns in time-series data. They can handle non-linear relationships and are effective in capturing complex relationships between data points. However, they are computationally more intensive than linear regression and might require more data for effective training.

CODE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```python
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, LSTM

# Load the dataset
df = pd.read_csv('MSFT.csv')
print(df.describe())
# Preprocess the data
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df.dropna(inplace=True)

# Visualize the closing prices
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Close'])
plt.title('Historical Stock Prices of MSFT)
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.show()

# Create a new column for the target variable (e.g., next day's closing price)
df['target'] = df['Close'].shift(-1)

# Drop any remaining rows with missing values
df.dropna(inplace=True)

df['20MA'] = df['Close'].rolling(window=20).mean()
df['50MA'] = df['Close'].rolling(window=50).mean()

# Plotting the moving averages
plt.figure(figsize=(12,6))
plt.plot(df.index, df['Close'], label='Closing Price')
plt.plot(df.index, df['20MA'], label='20-day Moving Average')
plt.plot(df.index, df['50MA'], label='50-day Moving Average')
plt.legend()
plt.title('Moving Averages for Stock Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.show()

# Split the data into features and target
X = df[['Open', 'Low', 'High', 'Volume']]
y = df['target']
```

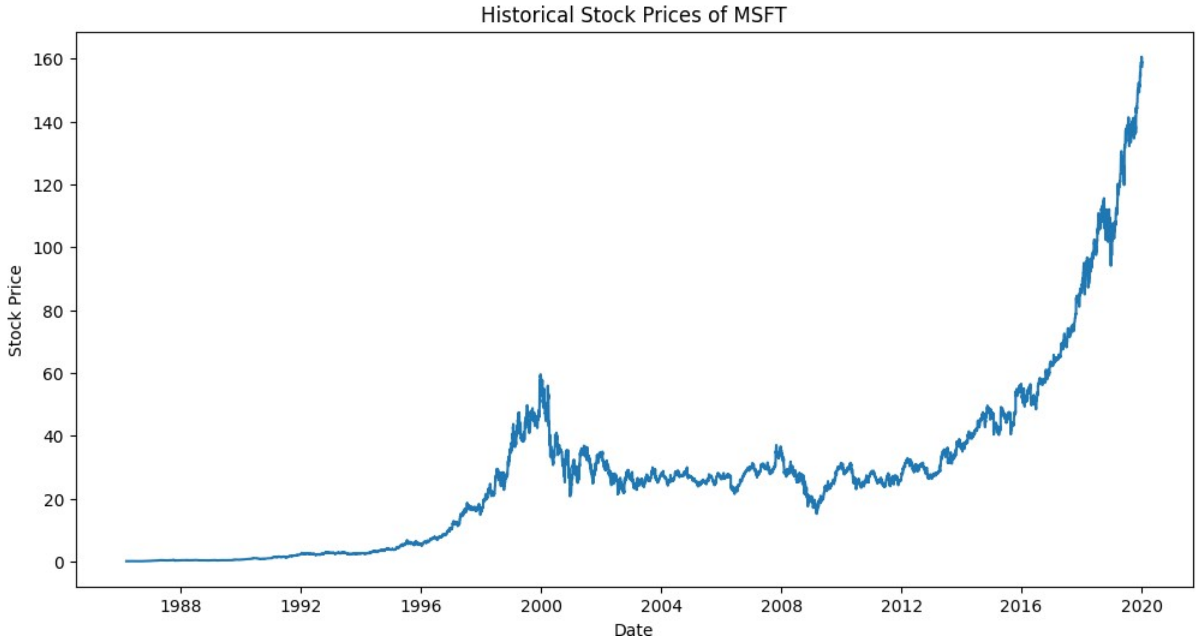
```python
# Feature scaling
scaler  =  MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=0)

# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")

# Visualize the predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test.index, y_test.values, label='Actual',color='red')
plt.scatter(y_test.index, y_pred, label='Predicted',alpha=0.5,color='black')
plt.title('Actual vs Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()


# Reshape the data for RNN
X_rnn = X_scaled.reshape(X_scaled.shape[0], X_scaled.shape[1], 1)

# Split the data into training and testing sets for RNN
X_train_rnn, X_test_rnn, y_train_rnn, y_test_rnn = train_test_split(X_rnn, y, test_size=0.2,
random_state=0)

# Create an RNN model
rnn_model = Sequential()
rnn_model.add(SimpleRNN(50, input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2]),
activation='relu'))
rnn_model.add(Dense(1))
rnn_model.compile(optimizer='adam', loss='mean_squared_error')
rnn_model.fit(X_train_rnn, y_train_rnn, epochs=50, batch_size=32, verbose=2)

# Make predictions using RNN
y_pred_rnn = rnn_model.predict(X_test_rnn)
```

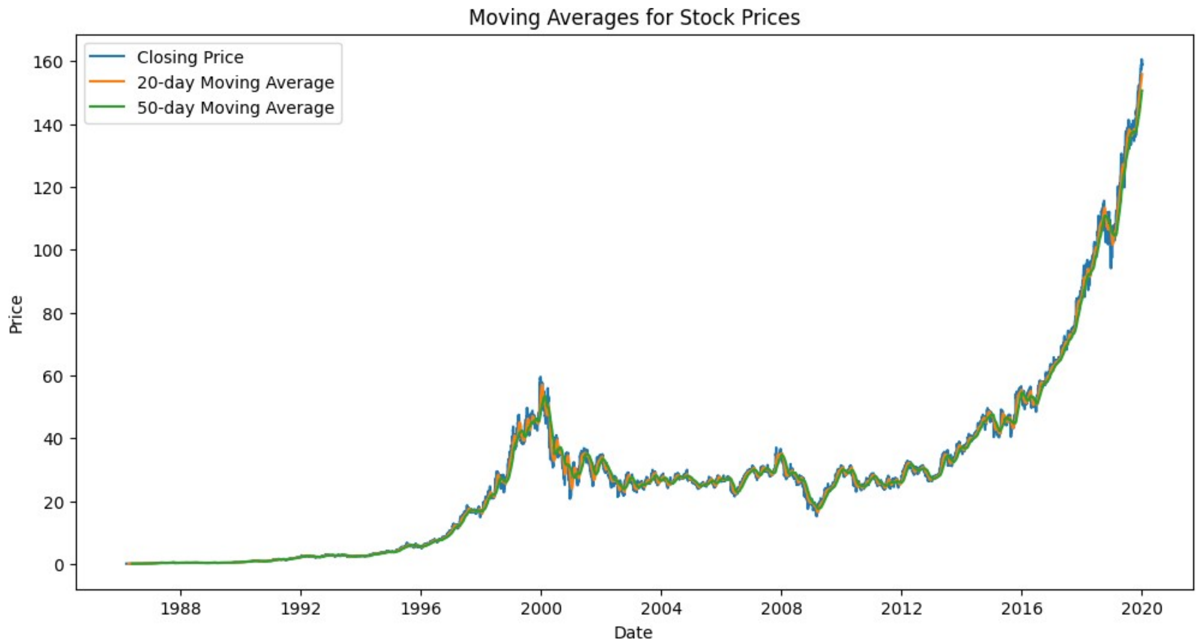
```python
# Model evaluation for RNN
rnn_mse = mean_squared_error(y_test_rnn, y_pred_rnn)
rnn_mae = mean_absolute_error(y_test_rnn, y_pred_rnn)
print(f"RNN Mean Squared Error: {rnn_mse}")
print(f"RNN Mean Absolute Error: {rnn_mae}")

# Visualize the predicted values from RNN
plt.figure(figsize=(12, 6))
plt.scatter(y_test.index, y_test.values, label='Actual', color='red')
plt.scatter(y_test.index, y_pred_rnn, label='RNN Predicted', alpha=0.5, color='blue')
plt.title('Actual vs RNN Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
# Visualize the linear regression and RNN predicted values
plt.figure(figsize=(12, 6))

# Plotting linear regression predictions
plt.scatter(y_test.index, y_test.values, label='Actual', color='red')
plt.scatter(y_test.index, y_pred, label='Linear Regression Predicted', alpha=0.5, color='green')

# Plotting RNN predictions
plt.scatter(y_test.index, y_pred_rnn, label='RNN Predicted', alpha=0.5, color='blue')

plt.title('Actual vs Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

OUTPUT:

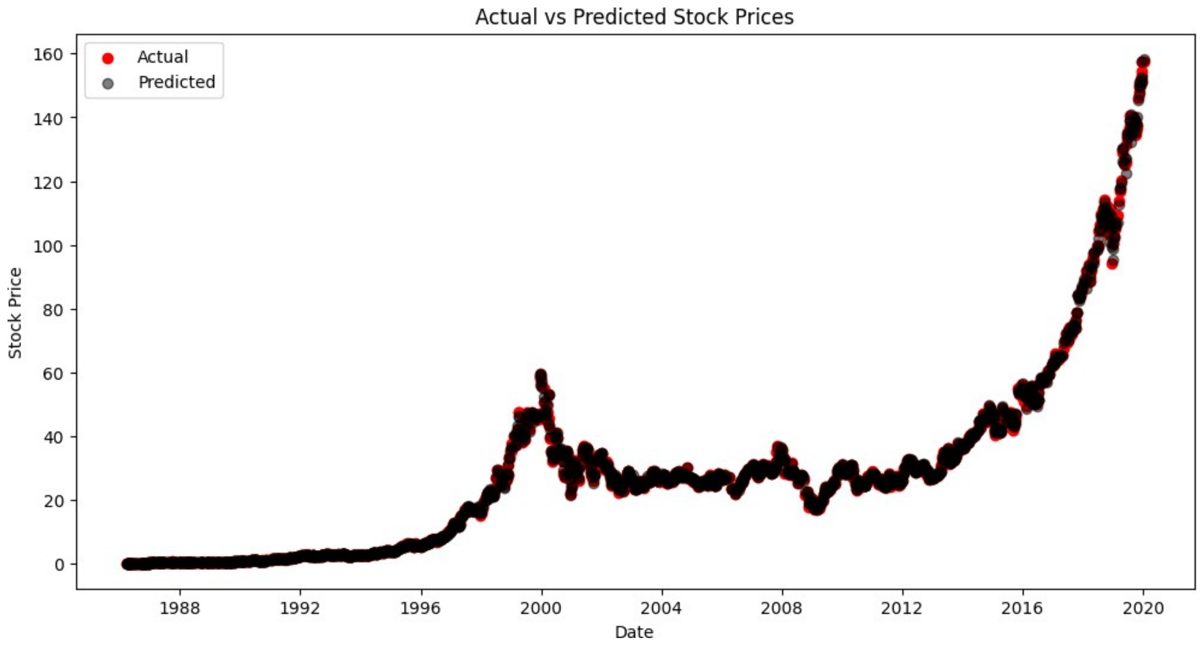
```
             Open         High          Low        Close    Adj Close  \
count  8525.000000  8525.000000  8525.000000  8525.000000  8525.000000
mean     28.220247    28.514473    27.918967    28.224480    23.417934
std      28.626752    28.848988    28.370344    28.626571    28.195330
min       0.088542     0.092014     0.088542     0.090278     0.058081
25%       3.414063     3.460938     3.382813     3.414063     2.196463
50%      26.174999    26.500000    25.889999    26.160000    18.441576
75%      34.230000    34.669998    33.750000    34.230000    25.392508
max     159.449997   160.729996   158.330002   160.619995   160.619995

             Volume
count  8.525000e+03
mean   6.045692e+07
std    3.891225e+07
min    2.304000e+06
```

25%   3.667960e+07
50%   5.370240e+07
75%   7.412350e+07
max   1.031789e+09





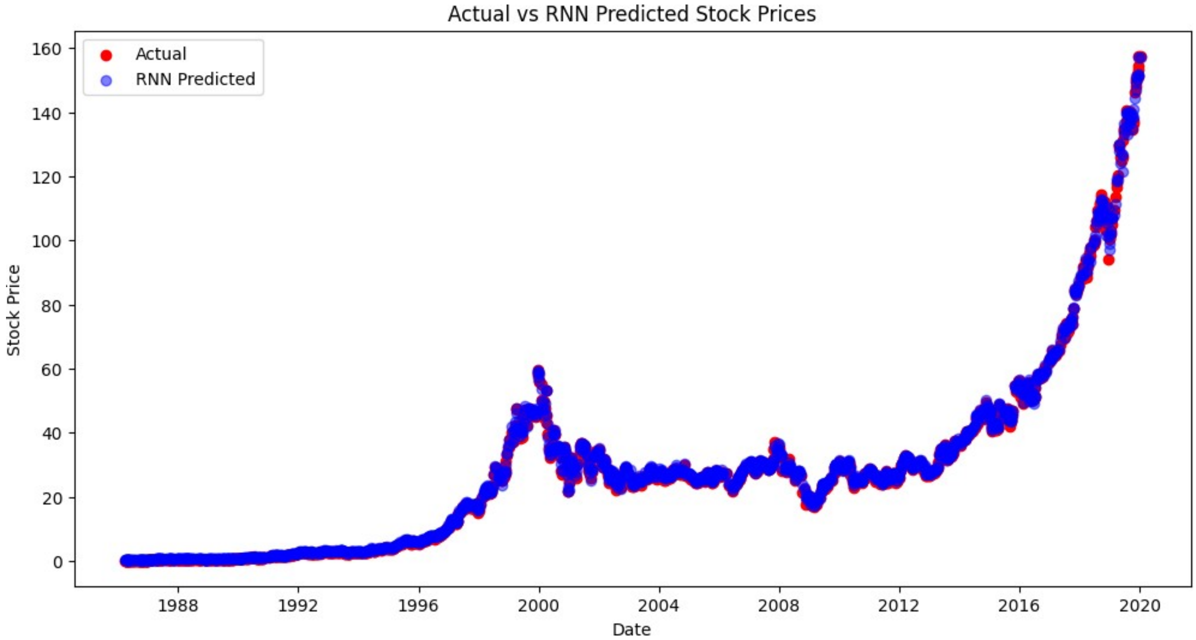
Mean Squared Error: 0.45474141895263165
Mean Absolute Error: 0.35983741296866834

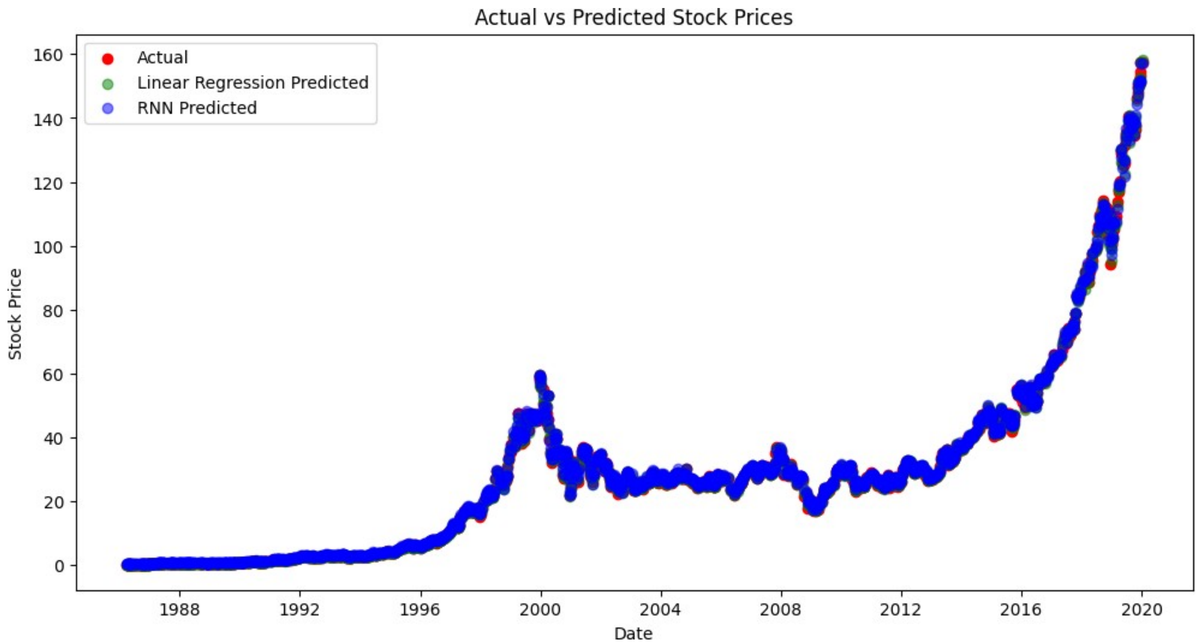


Epoch 1/50
214/214 - 5s - loss: 568.4958 - 5s/epoch - 22ms/step
Epoch 2/50
214/214 - 1s - loss: 0.8088 - 760ms/epoch - 4ms/step
Epoch 3/50
214/214 - 1s - loss: 0.6690 - 803ms/epoch - 4ms/step
Epoch 4/50
214/214 - 1s - loss: 0.6638 - 837ms/epoch - 4ms/step
Epoch 5/50
214/214 - 1s - loss: 0.6470 - 832ms/epoch - 4ms/step
Epoch 6/50
214/214 - 1s - loss: 0.6531 - 848ms/epoch - 4ms/step
Epoch 7/50
214/214 - 1s - loss: 0.6423 - 760ms/epoch - 4ms/step
Epoch 8/50
214/214 - 1s - loss: 0.6668 - 762ms/epoch - 4ms/step
Epoch 9/50
214/214 - 1s - loss: 0.6560 - 733ms/epoch - 3ms/step
Epoch 10/50
214/214 - 1s - loss: 0.6894 - 790ms/epoch - 4ms/step
Epoch 11/50
214/214 - 1s - loss: 0.6499 - 859ms/epoch - 4ms/step
Epoch 12/50
214/214 - 1s - loss: 0.6724 - 1s/epoch - 5ms/step
Epoch 13/50
214/214 - 2s - loss: 0.6945 - 2s/epoch - 7ms/step
Epoch 14/50
214/214 - 1s - loss: 0.6957 - 1s/epoch - 5ms/step
Epoch 15/50
214/214 - 1s - loss: 0.6632 - 856ms/epoch - 4ms/step
Epoch 16/50
214/214 - 1s - loss: 0.6972 - 1s/epoch - 6ms/step
Epoch 17/50
214/214 - 1s - loss: 0.6945 - 898ms/epoch - 4ms/step
Epoch 18/50
214/214 - 1s - loss: 0.6916 - 845ms/epoch - 4ms/step
Epoch 19/50
214/214 - 1s - loss: 0.7161 - 760ms/epoch - 4ms/step

Epoch 20/50
214/214 - 1s - loss: 0.7949 - 832ms/epoch - 4ms/step
Epoch 21/50
214/214 - 1s - loss: 0.7825 - 934ms/epoch - 4ms/step
Epoch 22/50
214/214 - 1s - loss: 0.6881 - 784ms/epoch - 4ms/step
Epoch 23/50
214/214 - 1s - loss: 0.6659 - 799ms/epoch - 4ms/step
Epoch 24/50
214/214 - 1s - loss: 0.7201 - 945ms/epoch - 4ms/step
Epoch 25/50
214/214 - 1s - loss: 0.7051 - 1s/epoch - 6ms/step
Epoch 26/50
214/214 - 1s - loss: 0.7130 - 1s/epoch - 6ms/step
Epoch 27/50
214/214 - 1s - loss: 0.7499 - 1s/epoch - 6ms/step
Epoch 28/50
214/214 - 1s - loss: 0.7532 - 908ms/epoch - 4ms/step
Epoch 29/50
214/214 - 0s - loss: 0.6760 - 466ms/epoch - 2ms/step
Epoch 30/50
214/214 - 0s - loss: 0.7531 - 464ms/epoch - 2ms/step
Epoch 31/50
214/214 - 0s - loss: 0.7102 - 462ms/epoch - 2ms/step
Epoch 32/50
214/214 - 0s - loss: 0.7620 - 458ms/epoch - 2ms/step
Epoch 33/50
214/214 - 0s - loss: 0.7607 - 455ms/epoch - 2ms/step
Epoch 34/50
214/214 - 0s - loss: 0.7370 - 464ms/epoch - 2ms/step
Epoch 35/50
214/214 - 0s - loss: 0.7361 - 449ms/epoch - 2ms/step
Epoch 36/50
214/214 - 0s - loss: 0.6654 - 481ms/epoch - 2ms/step
Epoch 37/50
214/214 - 0s - loss: 0.8175 - 451ms/epoch - 2ms/step
Epoch 38/50
214/214 - 0s - loss: 0.7421 - 457ms/epoch - 2ms/step
Epoch 39/50
214/214 - 0s - loss: 0.7381 - 465ms/epoch - 2ms/step
Epoch 40/50
214/214 - 0s - loss: 0.7497 - 440ms/epoch - 2ms/step
Epoch 41/50
214/214 - 0s - loss: 0.7005 - 476ms/epoch - 2ms/step
Epoch 42/50
214/214 - 0s - loss: 0.7046 - 452ms/epoch - 2ms/step
Epoch 43/50
214/214 - 0s - loss: 0.7002 - 471ms/epoch - 2ms/step
Epoch 44/50
214/214 - 0s - loss: 0.7235 - 455ms/epoch - 2ms/step
Epoch 45/50
214/214 - 0s - loss: 0.7230 - 463ms/epoch - 2ms/step
Epoch 46/50
214/214 - 0s - loss: 0.7502 - 450ms/epoch - 2ms/step
Epoch 47/50
214/214 - 0s - loss: 0.7051 - 460ms/epoch - 2ms/step
Epoch 48/50
214/214 - 1s - loss: 0.7524 - 690ms/epoch - 3ms/step
Epoch 49/50
214/214 - 1s - loss: 0.7146 - 712ms/epoch - 3ms/step

Epoch 50/50
214/214 - 1s - loss: 0.7100 - 771ms/epoch - 4ms/step
54/54 [==============================] - 0s 2ms/step
RNN Mean Squared Error: 0.6643683191384256
RNN Mean Absolute Error: 0.5405774408745059



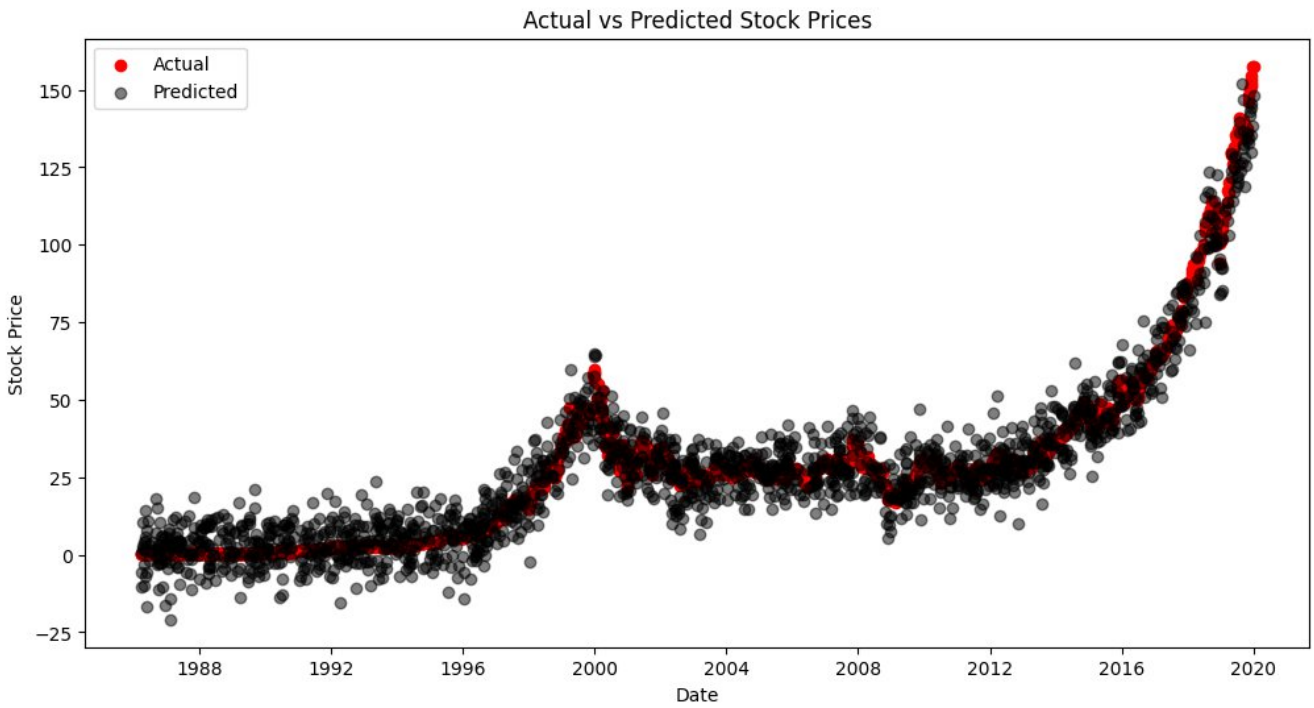


As shown in the above graph of the model trained on the given set there is an Overfitting of the model for the dataset thus we can conclude that there is a pattern formation in the model for the given dataset.
Thus,in regards to the overfitting we will introduce randomness to the dataset as such patterns are improbable in the real world application.

The randomness is introduced with the below lines of code:

```
X_scaled = X_scaled + np.random.normal(0, 0.075, X_scaled.shape)
y = y + np.random.normal(0, 0.01, y.shape)
```

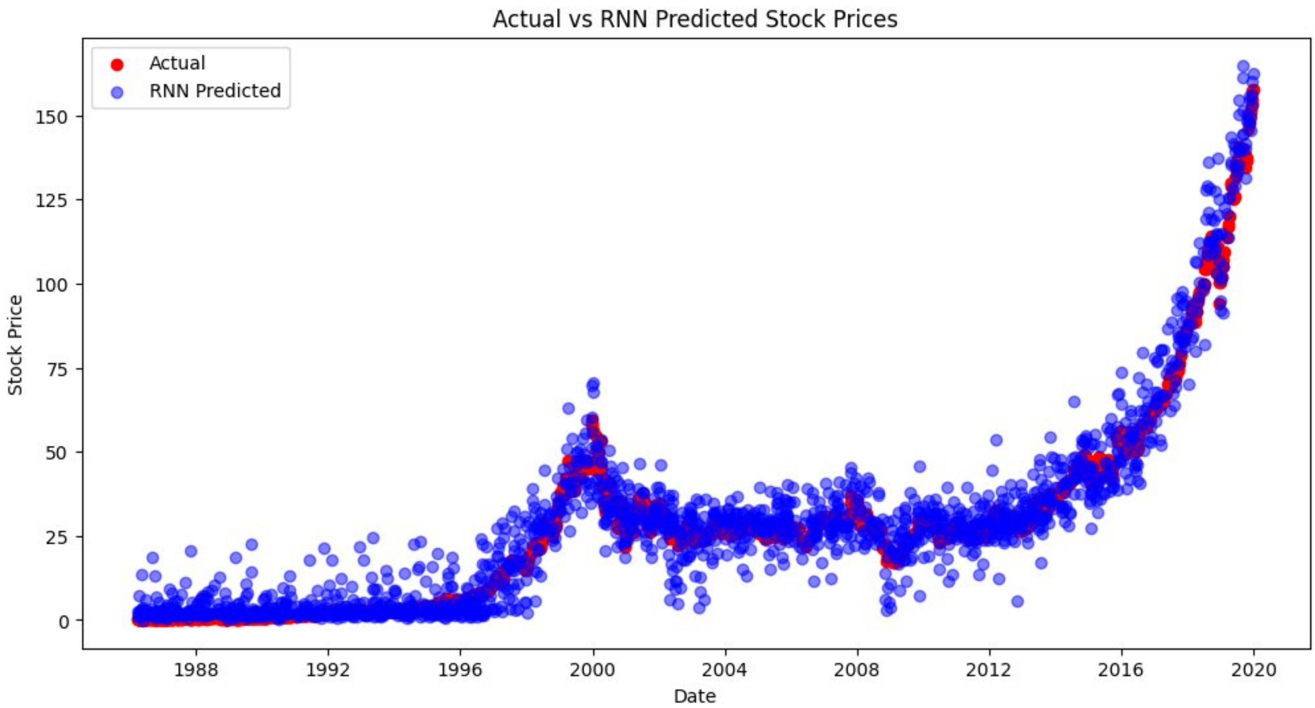
OUTPUT:
Mean Squared Error: 48.84306692940564
Mean Absolute Error: 5.5535263329560625

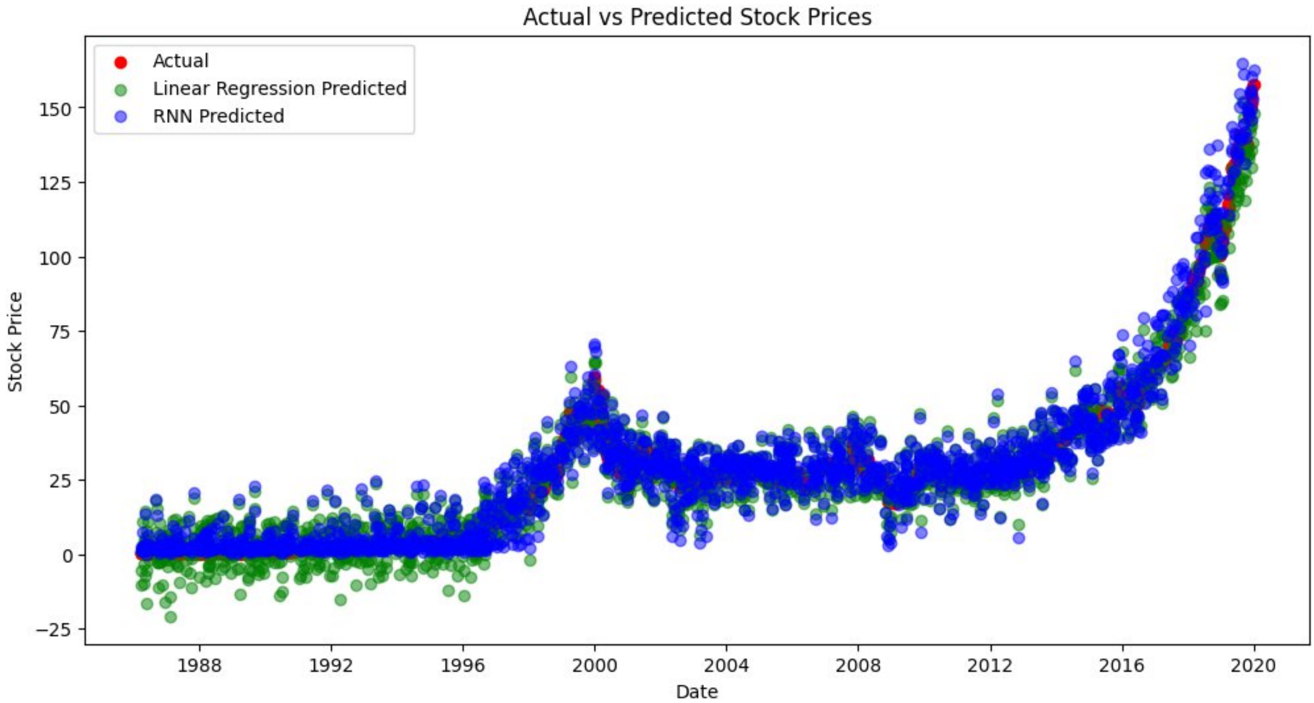


Epoch 1/50
214/214 - 1s - loss: 634.9327 - 1s/epoch - 7ms/step
Epoch 2/50
214/214 - 0s - loss: 63.4872 - 444ms/epoch - 2ms/step
Epoch 3/50
214/214 - 0s - loss: 56.9085 - 419ms/epoch - 2ms/step
Epoch 4/50
214/214 - 0s - loss: 52.5969 - 451ms/epoch - 2ms/step
Epoch 5/50
214/214 - 0s - loss: 49.5959 - 450ms/epoch - 2ms/step
Epoch 6/50
214/214 - 0s - loss: 48.2406 - 469ms/epoch - 2ms/step
Epoch 7/50
214/214 - 0s - loss: 48.5187 - 442ms/epoch - 2ms/step
Epoch 8/50
214/214 - 0s - loss: 47.2123 - 456ms/epoch - 2ms/step
Epoch 9/50
214/214 - 0s - loss: 45.0150 - 432ms/epoch - 2ms/step
Epoch 10/50
214/214 - 0s - loss: 44.4014 - 434ms/epoch - 2ms/step
Epoch 11/50
214/214 - 0s - loss: 43.3951 - 457ms/epoch - 2ms/step
Epoch 12/50
214/214 - 0s - loss: 44.9555 - 434ms/epoch - 2ms/step
Epoch 13/50

```
214/214 - 0s - loss: 42.6302 - 461ms/epoch - 2ms/step
Epoch 14/50
214/214 - 0s - loss: 40.9970 - 439ms/epoch - 2ms/step
Epoch 15/50
214/214 - 0s - loss: 40.5456 - 476ms/epoch - 2ms/step
Epoch 16/50
214/214 - 0s - loss: 40.2339 - 434ms/epoch - 2ms/step
Epoch 17/50
214/214 - 0s - loss: 40.4421 - 452ms/epoch - 2ms/step
Epoch 18/50
214/214 - 0s - loss: 39.7864 - 463ms/epoch - 2ms/step
Epoch 19/50
214/214 - 0s - loss: 38.9945 - 458ms/epoch - 2ms/step
Epoch 20/50
214/214 - 1s - loss: 39.1094 - 569ms/epoch - 3ms/step
Epoch 21/50
214/214 - 1s - loss: 38.7289 - 750ms/epoch - 4ms/step
Epoch 22/50
214/214 - 1s - loss: 38.9546 - 703ms/epoch - 3ms/step
Epoch 23/50
214/214 - 1s - loss: 39.0682 - 789ms/epoch - 4ms/step
Epoch 24/50
214/214 - 1s - loss: 38.9879 - 558ms/epoch - 3ms/step
Epoch 25/50
214/214 - 0s - loss: 38.4918 - 455ms/epoch - 2ms/step
Epoch 26/50
214/214 - 0s - loss: 38.4851 - 450ms/epoch - 2ms/step
Epoch 27/50
214/214 - 0s - loss: 38.9154 - 433ms/epoch - 2ms/step
Epoch 28/50
214/214 - 0s - loss: 37.9100 - 436ms/epoch - 2ms/step
Epoch 29/50
214/214 - 0s - loss: 38.9169 - 448ms/epoch - 2ms/step
Epoch 30/50
214/214 - 0s - loss: 37.9605 - 462ms/epoch - 2ms/step
Epoch 31/50
214/214 - 0s - loss: 38.3789 - 477ms/epoch - 2ms/step
Epoch 32/50
214/214 - 0s - loss: 38.4470 - 449ms/epoch - 2ms/step
Epoch 33/50
214/214 - 0s - loss: 38.1417 - 482ms/epoch - 2ms/step
Epoch 34/50
214/214 - 0s - loss: 38.7707 - 446ms/epoch - 2ms/step
Epoch 35/50
214/214 - 0s - loss: 38.1561 - 465ms/epoch - 2ms/step
Epoch 36/50
214/214 - 0s - loss: 38.4503 - 466ms/epoch - 2ms/step
Epoch 37/50
214/214 - 0s - loss: 38.1264 - 431ms/epoch - 2ms/step
Epoch 38/50
214/214 - 0s - loss: 37.7799 - 447ms/epoch - 2ms/step
Epoch 39/50
214/214 - 0s - loss: 38.4195 - 460ms/epoch - 2ms/step
Epoch 40/50
214/214 - 0s - loss: 38.9777 - 448ms/epoch - 2ms/step
Epoch 41/50
214/214 - 0s - loss: 37.6866 - 453ms/epoch - 2ms/step
Epoch 42/50
214/214 - 0s - loss: 37.6476 - 445ms/epoch - 2ms/step
Epoch 43/50
```

```
214/214 - 0s - loss: 37.7552 - 443ms/epoch - 2ms/step
Epoch 44/50
214/214 - 0s - loss: 37.5282 - 454ms/epoch - 2ms/step
Epoch 45/50
214/214 - 0s - loss: 38.0236 - 453ms/epoch - 2ms/step
Epoch 46/50
214/214 - 1s - loss: 37.8537 - 675ms/epoch - 3ms/step
Epoch 47/50
214/214 - 1s - loss: 38.0985 - 721ms/epoch - 3ms/step
Epoch 48/50
214/214 - 1s - loss: 37.5385 - 748ms/epoch - 3ms/step
Epoch 49/50
214/214 - 1s - loss: 37.3092 - 643ms/epoch - 3ms/step
Epoch 50/50
214/214 - 0s - loss: 37.6031 - 450ms/epoch - 2ms/step
54/54 [==============================] - 0s 2ms/step
RNN Mean Squared Error: 43.92814931300681
RNN Mean Absolute Error: 4.80198262565908
```





As we can infer from the above graph, the RNN model is giving more accurate results when compared to Linear Regression.

MODEL EVALUATION:

Model evaluation is the process of assessing how well a machine learning model performs on the test dataset. It involves the use of various metrics to quantify the performance of the model in terms of its ability to make accurate predictions. In the context of stock price prediction, it is essential to evaluate theperformance of both the linear regression and RNN models to understand how well they predict stock prices.

In the provided code, model evaluation is done using two common metrics:

1. Mean Squared Error (MSE): It measures the average of the squares of the errors, which are the differences between actual and predicted values. A lower MSE indicates better performance.

2. Mean Absolute Error (MAE): It measures the average of the absolute differences between actual and predicted values. MAE provides a more intuitive understanding of the model's performance in terms of absolute errors.

Both the linear regression and RNN models are evaluated using these metrics. The code calculates and prints the MSE and MAE for both models, providing insights into their predictive accuracy. Additionally, the code visualizes the predicted values against the actual stock prices for both the linear regression and RNN models, allowing for a graphical comparison of their performance.

By evaluating the models using these metrics, one can make informed decisions about which model performs better for the task of stock price prediction and can assess whether one model outperforms the other in terms of accuracy and reliability.