import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from sklearn.svm import SVR

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

from sklearn.model\_selection import train\_test\_split

# 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']

# Feature scaling

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

X\_scaled = X\_scaled + np.random.normal(0, 0.075, X\_scaled.shape)

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

# 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)

# 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()

# Train an SVM model

svm\_model = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)

svm\_model.fit(X\_train, y\_train)

# Make predictions using the SVM model

y\_pred\_svm = svm\_model.predict(X\_test)

# Model evaluation for SVM

svm\_mse = mean\_squared\_error(y\_test, y\_pred\_svm)

svm\_mae = mean\_absolute\_error(y\_test, y\_pred\_svm)

print(f"SVM Mean Squared Error: {svm\_mse}")

print(f"SVM Mean Absolute Error: {svm\_mae}")

# Visualize the predicted values from SVM

plt.figure(figsize=(12, 6))

plt.scatter(y\_test.index, y\_test.values, label='Actual', color='red')

plt.scatter(y\_test.index, y\_pred\_svm, label='SVM Predicted', alpha=0.5, color='purple')

plt.title('Actual vs SVM Predicted Stock Prices')

plt.xlabel('Date')

plt.ylabel('Stock Price')

plt.legend()

plt.show()

# Visualize all the predictions (Linear Regression, RNN, SVM)

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='Linear Regression Predicted', alpha=0.5, color='green')

plt.scatter(y\_test.index, y\_pred\_rnn, label='RNN Predicted', alpha=0.5, color='blue')

plt.scatter(y\_test.index, y\_pred\_svm, label='SVM Predicted', alpha=0.5, color='purple')

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

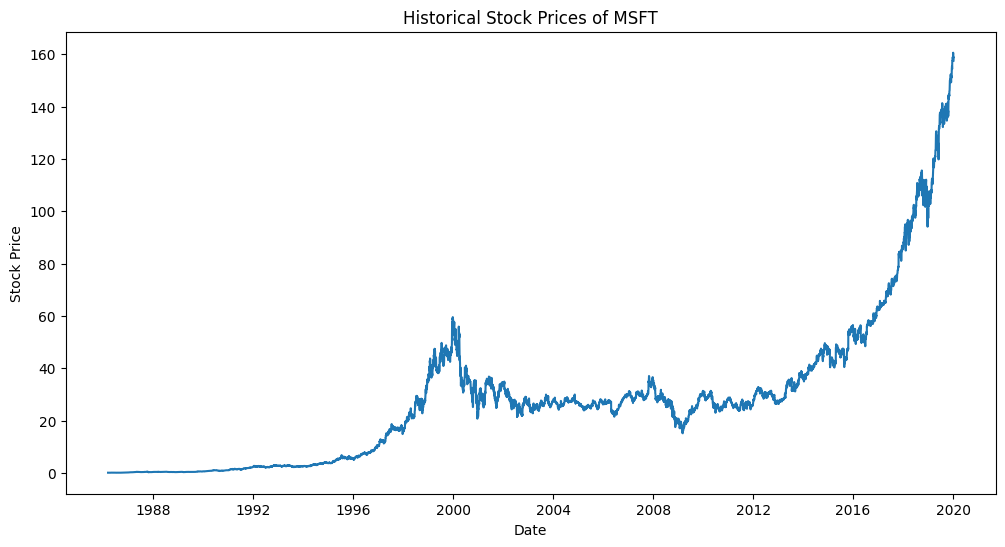
min 2.304000e+06

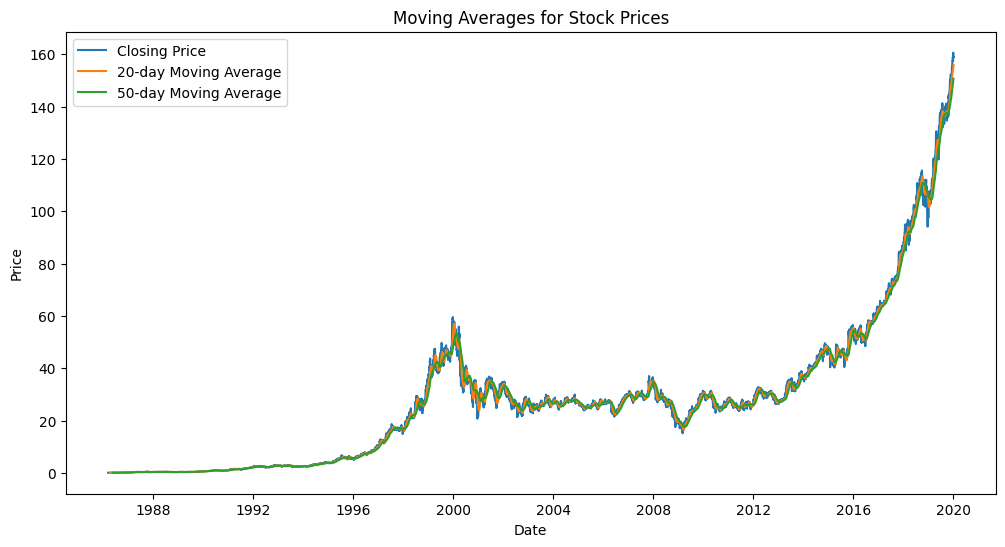
25% 3.667960e+07

50% 5.370240e+07

75% 7.412350e+07

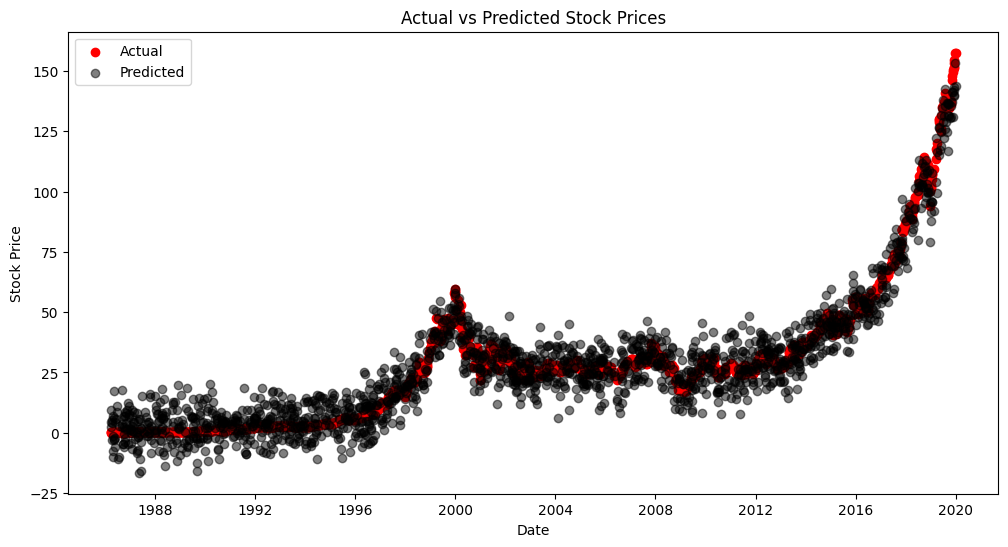
max 1.031789e+09





Mean Squared Error: 47.70428113506811

Mean Absolute Error: 5.540635678906947



Epoch 1/50

214/214 - 3s - loss: 609.0205 - 3s/epoch - 16ms/step

Epoch 2/50

214/214 - 1s - loss: 66.3182 - 857ms/epoch - 4ms/step

Epoch 3/50

214/214 - 1s - loss: 57.5875 - 1s/epoch - 5ms/step

Epoch 4/50

214/214 - 1s - loss: 52.0961 - 762ms/epoch - 4ms/step

Epoch 5/50

214/214 - 1s - loss: 50.4893 - 1s/epoch - 5ms/step

Epoch 6/50

214/214 - 1s - loss: 50.9139 - 1s/epoch - 6ms/step

Epoch 7/50

214/214 - 2s - loss: 52.0067 - 2s/epoch - 8ms/step

Epoch 8/50

214/214 - 1s - loss: 49.9739 - 1s/epoch - 6ms/step

Epoch 9/50

214/214 - 0s - loss: 49.0960 - 489ms/epoch - 2ms/step

Epoch 10/50

214/214 - 1s - loss: 48.5520 - 511ms/epoch - 2ms/step

Epoch 11/50

214/214 - 1s - loss: 47.4742 - 531ms/epoch - 2ms/step

Epoch 12/50

214/214 - 0s - loss: 46.6914 - 460ms/epoch - 2ms/step

Epoch 13/50

214/214 - 0s - loss: 47.8069 - 480ms/epoch - 2ms/step

Epoch 14/50

214/214 - 0s - loss: 45.1142 - 459ms/epoch - 2ms/step

Epoch 15/50

214/214 - 0s - loss: 45.1304 - 497ms/epoch - 2ms/step

Epoch 16/50

214/214 - 0s - loss: 44.2735 - 458ms/epoch - 2ms/step

Epoch 17/50

214/214 - 0s - loss: 44.2807 - 484ms/epoch - 2ms/step

Epoch 18/50

214/214 - 0s - loss: 43.7559 - 466ms/epoch - 2ms/step

Epoch 19/50

214/214 - 1s - loss: 43.0986 - 506ms/epoch - 2ms/step

Epoch 20/50

214/214 - 0s - loss: 43.2854 - 472ms/epoch - 2ms/step

Epoch 21/50

214/214 - 0s - loss: 42.2965 - 486ms/epoch - 2ms/step

Epoch 22/50

214/214 - 0s - loss: 42.3305 - 462ms/epoch - 2ms/step

Epoch 23/50

214/214 - 0s - loss: 43.2547 - 490ms/epoch - 2ms/step

Epoch 24/50

214/214 - 0s - loss: 42.1658 - 469ms/epoch - 2ms/step

Epoch 25/50

214/214 - 0s - loss: 41.0218 - 477ms/epoch - 2ms/step

Epoch 26/50

214/214 - 0s - loss: 40.9037 - 474ms/epoch - 2ms/step

Epoch 27/50

214/214 - 0s - loss: 40.6307 - 475ms/epoch - 2ms/step

Epoch 28/50

214/214 - 1s - loss: 41.6998 - 696ms/epoch - 3ms/step

Epoch 29/50

214/214 - 1s - loss: 42.0923 - 770ms/epoch - 4ms/step

Epoch 30/50

214/214 - 1s - loss: 41.0411 - 725ms/epoch - 3ms/step

Epoch 31/50

214/214 - 1s - loss: 41.4113 - 709ms/epoch - 3ms/step

Epoch 32/50

214/214 - 1s - loss: 40.6816 - 501ms/epoch - 2ms/step

Epoch 33/50

214/214 - 0s - loss: 40.9345 - 464ms/epoch - 2ms/step

Epoch 34/50

214/214 - 0s - loss: 40.1759 - 488ms/epoch - 2ms/step

Epoch 35/50

214/214 - 0s - loss: 40.1066 - 488ms/epoch - 2ms/step

Epoch 36/50

214/214 - 0s - loss: 40.6465 - 499ms/epoch - 2ms/step

Epoch 37/50

214/214 - 0s - loss: 41.1360 - 474ms/epoch - 2ms/step

Epoch 38/50

214/214 - 0s - loss: 40.6142 - 475ms/epoch - 2ms/step

Epoch 39/50

214/214 - 0s - loss: 40.5984 - 473ms/epoch - 2ms/step

Epoch 40/50

214/214 - 0s - loss: 40.0818 - 468ms/epoch - 2ms/step

Epoch 41/50

214/214 - 0s - loss: 40.2311 - 477ms/epoch - 2ms/step

Epoch 42/50

214/214 - 0s - loss: 40.2067 - 471ms/epoch - 2ms/step

Epoch 43/50

214/214 - 0s - loss: 40.4598 - 468ms/epoch - 2ms/step

Epoch 44/50

214/214 - 0s - loss: 40.1106 - 482ms/epoch - 2ms/step

Epoch 45/50

214/214 - 0s - loss: 40.0732 - 496ms/epoch - 2ms/step

Epoch 46/50

214/214 - 0s - loss: 40.3023 - 490ms/epoch - 2ms/step

Epoch 47/50

214/214 - 0s - loss: 39.6917 - 480ms/epoch - 2ms/step

Epoch 48/50

214/214 - 0s - loss: 39.9114 - 481ms/epoch - 2ms/step

Epoch 49/50

214/214 - 0s - loss: 39.4110 - 482ms/epoch - 2ms/step

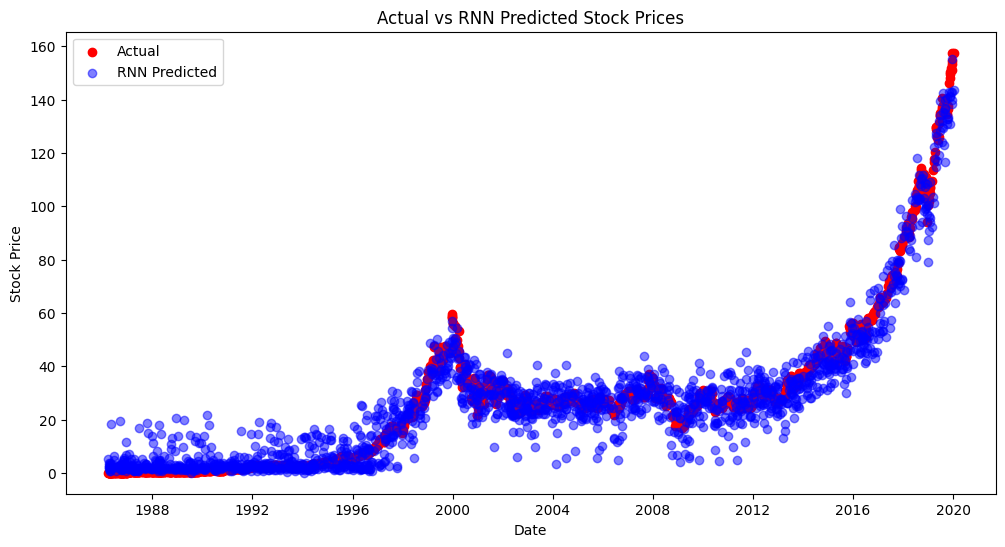
Epoch 50/50

214/214 - 0s - loss: 40.4630 - 479ms/epoch - 2ms/step

54/54 [==============================] - 0s 2ms/step

RNN Mean Squared Error: 39.74273884332308

RNN Mean Absolute Error: 4.690089046279755



SVM Mean Squared Error: 43.267522205818885

SVM Mean Absolute Error: 5.298840015338619

