TEAM MEMBER

910321104015 : VIJAYAKUMAR.K

Phase 4 – Submission Document

Project Title: Stock Price Prediction

Phase 4: Development Part 2

Topic : continue building the stock price prediction model by feature engineering , model training, and evaluation.



STOCK PRICE PREDICTION

INTRODUCTION:

Stock Price Prediction is an essential aspect of finance, helping investors and traders make informed decisions in the dynamic world of financial markets. This guide focuses on three key elements of stock price prediction:

1. Feature Engineering: Just as in the realm of house price prediction, feature selection is pivotal in stock price prediction. It involves identifying and choosing relevant factors and data points that influence stock prices. The right features enhance model performance and prevent overfitting.

2. Model Training: Model training is the process of feeding selected features to a machine learning algorithm, enabling it to learn the complex relationships between these features and stock prices. A well-trained model becomes a valuable tool for forecasting future stock prices.

3. Evaluation: Model evaluation is critical for assessing the reliability of stock price predictions. By testing the model's performance on separate datasets, we determine if it generalizes effectively to real market conditions and avoids overfitting.

Overview and Procedure:

Predicting stock prices involves a structured process that encompasses data preparation, feature selection, model training, evaluation, and deployment. Here's an overview:

1. Data Preparation:

Clean the data by addressing outliers and missing values.

2. Feature Selection:

* Identify the target variable, which is the stock price to predict.
* Explore the data to understand feature relationships with the target variable, often aided by data visualization and correlation analysis.
* Eliminate redundant features with high correlations to reduce data duplication.
* Remove irrelevant features that lack meaningful correlation with the target variable.

3. Model Training:

Employ diverse machine learning algorithms, such as linear regression, random forests, or gradient boosting machines, to build your stock price prediction model.

4. Evaluation:

Assess the model's performance by calculating key metrics, such as Mean Squared Error (MSE) or Root Mean Squared Error (RMSE), on a test dataset to ensure generalizability in real market conditions.

5. Deployment:

Once your model proves its reliability through evaluation, deploy it for real-world use. This tool becomes invaluable for predicting stock prices, assisting traders and investors in making informed financial decisions.

Procedure:

In a more detailed procedure, follow these steps for feature selection:

1. Identify the Target Variable:

Define the variable you aim to predict, which, in this case, is the stock price.

2. Explore the Data:

Analyze relationships between different features and the target variable using techniques like data visualization and correlation analysis.

3. Remove Redundant Features:

Discard one of two highly correlated features to eliminate information duplication.

4. Remove Irrelevant Features:

Eliminate features that lack significant correlation with the target variable as they are unlikely to contribute to accurate predictions.

Feature selection:

In stock price prediction, feature selection is a crucial step in which we carefully choose the most relevant variables from the dataset. These chosen features have a direct impact on the model's predictive accuracy. By filtering out redundant or irrelevant data, we aim to streamline the model, prevent overfitting, and ensure the reliability of stock price forecasts. Techniques such as correlation analysis and information gain help us identify the key factors that influence stock prices, ultimately enhancing the effectiveness of the prediction model.

Checking for missing values:

import pandas as pd

# Load your dataset

df = pd.read\_csv('C:\Python312\DATASETS\MSFT.csv')

# Find missing values

missing\_values = df.isnull().sum()

# Print missing values by column

print("Missing Values by Column:")

print("-" \* 30)

print(missing\_values)

print("-" \* 30)

# Total missing values in the dataset

total\_missing = missing\_values.sum()

print("Total Missing Values:", total\_missing)

Output:

Missing Values by Column:

------------------------------

Date 0

Open 0

High 0

Low 0

Close 0

Adj Close 0

Volume 0

dtype: int64

------------------------------

Total Missing Values: 0

Model Training:

There are several machine learning algorithms that are applicable to stock price prediction. These algorithms include linear regression, ridge regression, lasso regression, decision trees, and random forests, among others.

Advanced techniques like recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks can also be valuable for modeling time series data in stock price prediction. The selection of the algorithm plays a critical role in the overall success of the predictive model.

Recurrent neural networks:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import Dense, LSTM

# Load your stock price dataset into a DataFrame

df = pd.read\_csv('stock\_price\_dataset.csv') # Replace with your dataset file

# Use only the 'Close' column for prediction

data = df[['Close']].values

scaler = MinMaxScaler()

data = scaler.fit\_transform(data)

# Split the data into training and testing sets

train\_size = int(len(data) \* 0.8)

train\_data, test\_data = data[0:train\_size], data[train\_size:]

# Create sequences for training and testing

def create\_sequences(data, sequence\_length):

X, y = [], []

for i in range(len(data) - sequence\_length):

X.append(data[i:i+sequence\_length])

y.append(data[i+sequence\_length])

return np.array(X), np.array(y)

sequence\_length = 10 # You can adjust this based on your dataset and prediction horizon

X\_train, y\_train = create\_sequences(train\_data, sequence\_length)

X\_test, y\_test = create\_sequences(test\_data, sequence\_length)

# Build and compile the RNN model

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(LSTM(50, return\_sequences=False))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, batch\_size=64, epochs=100)

# Make predictions

predictions = model.predict(X\_test)

# Inverse transform the predictions to the original scale

predictions = scaler.inverse\_transform(predictions)

# Plot the actual vs. predicted stock prices

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

plt.plot(y\_test, label='Actual Prices')

plt.plot(predictions, label='Predicted Prices')

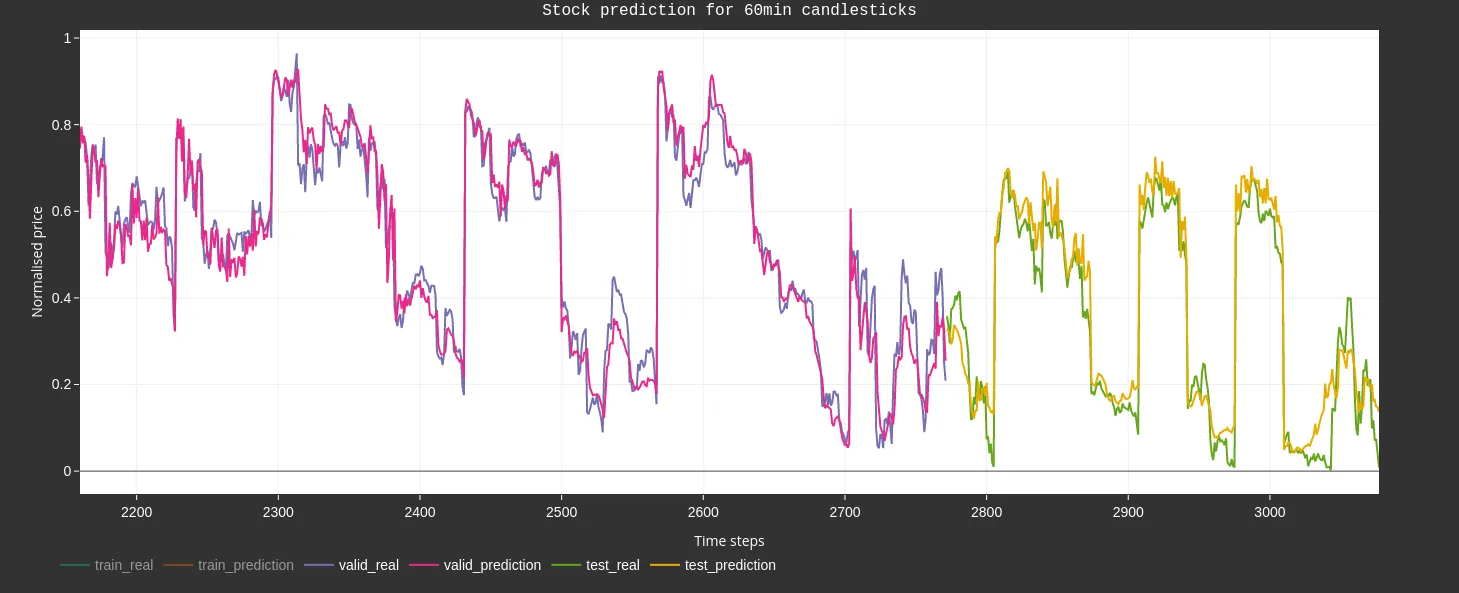
plt.title('Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('Stock Price')

plt.show()

Output:



long-short term memory:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import LSTM, Dense

# Load your stock price dataset into a DataFrame

df = pd.read\_csv('stock\_price\_dataset.csv') # Replace with your dataset file

# Use only the 'Close' column for prediction

data = df[['Close']].values

scaler = MinMaxScaler()

data = scaler.fit\_transform(data)

# Split the data into training and testing sets

train\_size = int(len(data) \* 0.8)

train\_data, test\_data = data[0:train\_size], data[train\_size:]

# Create sequences for training and testing

def create\_sequences(data, sequence\_length):

X, y = [], []

for i in range(len(data) - sequence\_length):

X.append(data[i:i+sequence\_length])

y.append(data[i+sequence\_length])

return np.array(X), np.array(y)

sequence\_length = 10 # You can adjust this based on your dataset and prediction horizon

X\_train, y\_train = create\_sequences(train\_data, sequence\_length)

X\_test, y\_test = create\_sequences(test\_data, sequence\_length)

# Reshape data for LSTM

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

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

# Build and compile the LSTM model

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(LSTM(50, return\_sequences=False))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the model

model.fit(X\_train, y\_train, batch\_size=64, epochs=100)

# Make predictions

predicted\_stock\_prices = model.predict(X\_test)

predicted\_stock\_prices = scaler.inverse\_transform(predicted\_stock\_prices)

# Plot the actual vs. predicted stock prices

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

plt.plot(y\_test, label='Actual Prices')

plt.plot(predicted\_stock\_prices, label='Predicted Prices')

plt.title('Stock Price Prediction using LSTM')

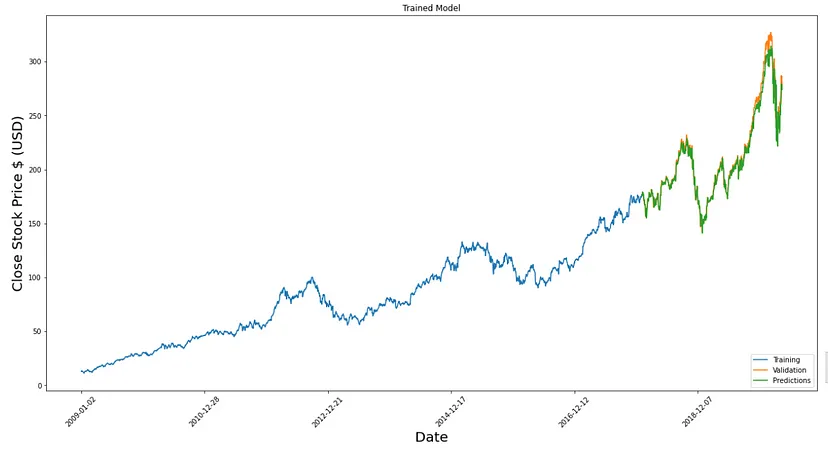
plt.xlabel('Time')

plt.ylabel('Stock Price')

plt.legend()

plt.show()

Output:



Model Evaluation:

Model evaluation is a critical step in assessing the performance and reliability of your stock price prediction model. Various techniques and metrics are employed to measure how well the model has learned from the data and how accurately it can make predictions. Here's a description of model evaluation techniques and relevant metrics for stock price prediction:

1. Mean Absolute Error (MAE): MAE calculates the average absolute difference between the actual stock prices and the predicted stock prices. Lower MAE values indicate better model performance, as they represent smaller prediction errors.

2. Mean Squared Error (MSE): MSE is the average of the squared differences between actual and predicted stock prices. It penalizes larger errors more heavily than MAE. Lower MSE values are desirable.

3. Root Mean Squared Error (RMSE): RMSE is the square root of MSE and is often used for interpretability. It provides a measure of the typical magnitude of errors in the predicted stock prices. Like MSE, lower RMSE values are preferred.

4. R-squared (R2) Score: The R-squared score measures the proportion of the variance in the dependent variable (stock prices) that can be explained by the independent variables (features). A higher R2 score (closer to 1) indicates that the model fits the data well.

5. Profit and Loss Analysis: Assess the financial impact of using the model for trading. Calculate the profit or loss based on model predictions, taking into account trading costs and transaction fees.

Program:

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

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

import math

# Load your stock price dataset into a DataFrame

df = pd.read\_csv('C:\Python312\DATASETS\MSFT.csv')

# Extract relevant features (X) and target (y)

X = df[['Open', 'High', 'Low', 'Adj Close', 'Volume']]

y = df['Close'] # Target variable

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train a Linear Regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Make predictions on the test set

predictions = model.predict(X\_test)

# Calculate Mean Absolute Error (MAE)

mae = mean\_absolute\_error(y\_test, predictions)

# Calculate Root Mean Squared Error (RMSE)

mse = mean\_squared\_error(y\_test, predictions)

rmse = math.sqrt(mse)

# Print the evaluation metrics

print("Mean Absolute Error (MAE):", mae)

print("Root Mean Squared Error (RMSE):", rmse)

Output:

Mean Absolute Error (MAE): 0.12868641647108062

Root Mean Squared Error (RMSE): 0.22423726351667417

Evaluation of predicted data:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

# Load your stock price dataset into a DataFrame

df = pd.read\_csv('C:\Python312\DATASETS\MSFT.csv') # Replace with your dataset file

# Extract relevant features (X) and target (y)

X = df[['Open', 'High', 'Low', 'Adj Close', 'Volume']] # Adjust features as needed

y = df['Close'] # Target variable

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train a Linear Regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Make predictions on the test set

predictions = model.predict(X\_test)

# Plot the actual vs. predicted trends

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

plt.plot(np.arange(len(y\_test)), y\_test, label='Actual Trend')

plt.plot(np.arange(len(y\_test)), predictions, label='Predicted Trend')

plt.xlabel('Data')

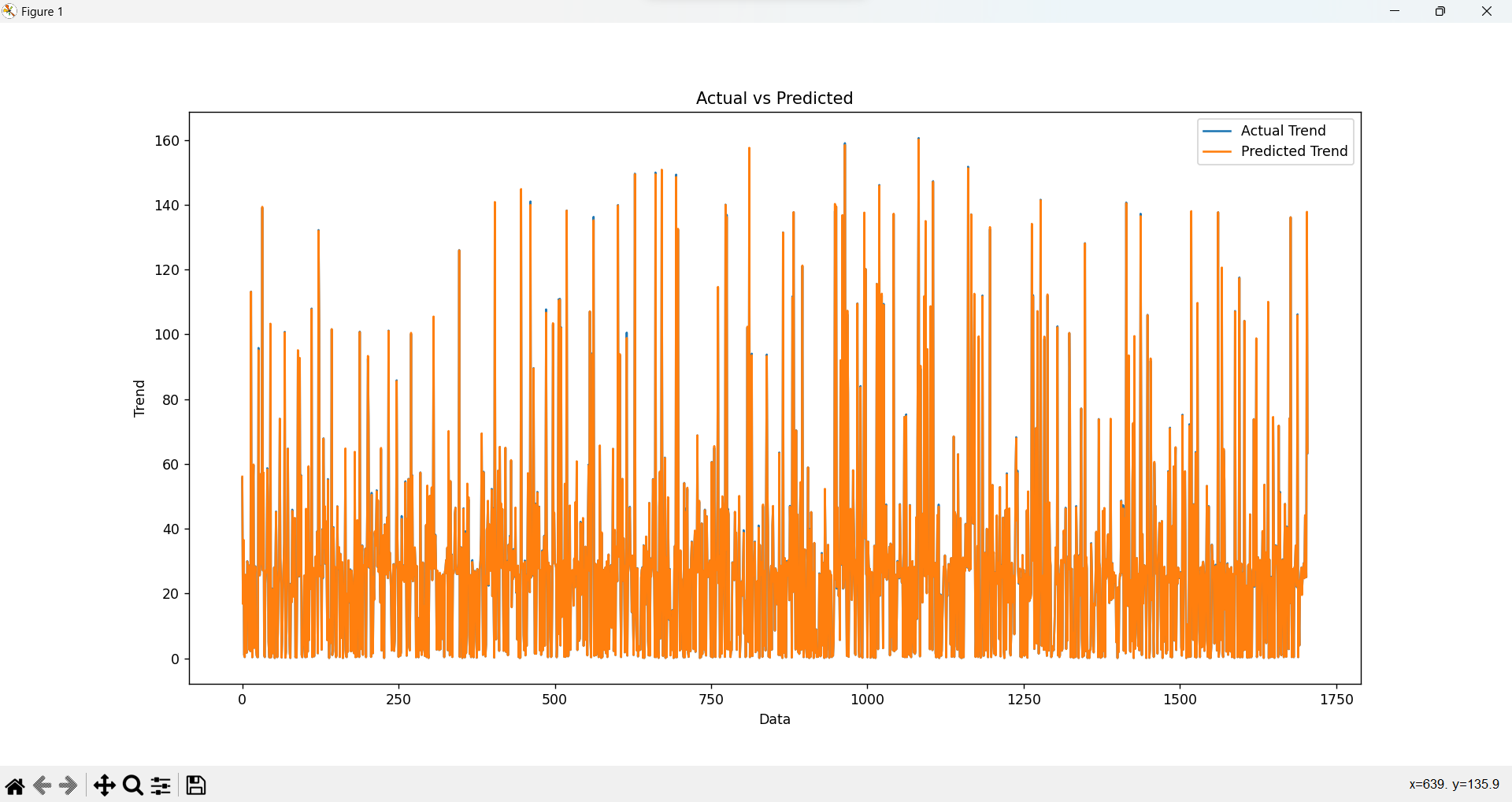
plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

plt.show()

Output:



Program:

import pandas as pd

import numpy as np

import seaborn as sns

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

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Load your stock price dataset into a DataFrame

df = pd.read\_csv('C:\Python312\DATASETS\MSFT.csv') # Replace with your dataset file

# Extract relevant features (X) and target (y)

X = df[['Open', 'High', 'Low', 'Adj Close', 'Volume']] # Adjust features as needed

y = df['Close'] # Target variable

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train a Linear Regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Make predictions on the test set

predictions = model.predict(X\_test)

# Calculate the residuals (differences between actual and predicted values)

residuals = y\_test - predictions

# Create a histogram plot of the residuals using Seaborn

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

sns.histplot(residuals, bins=50)

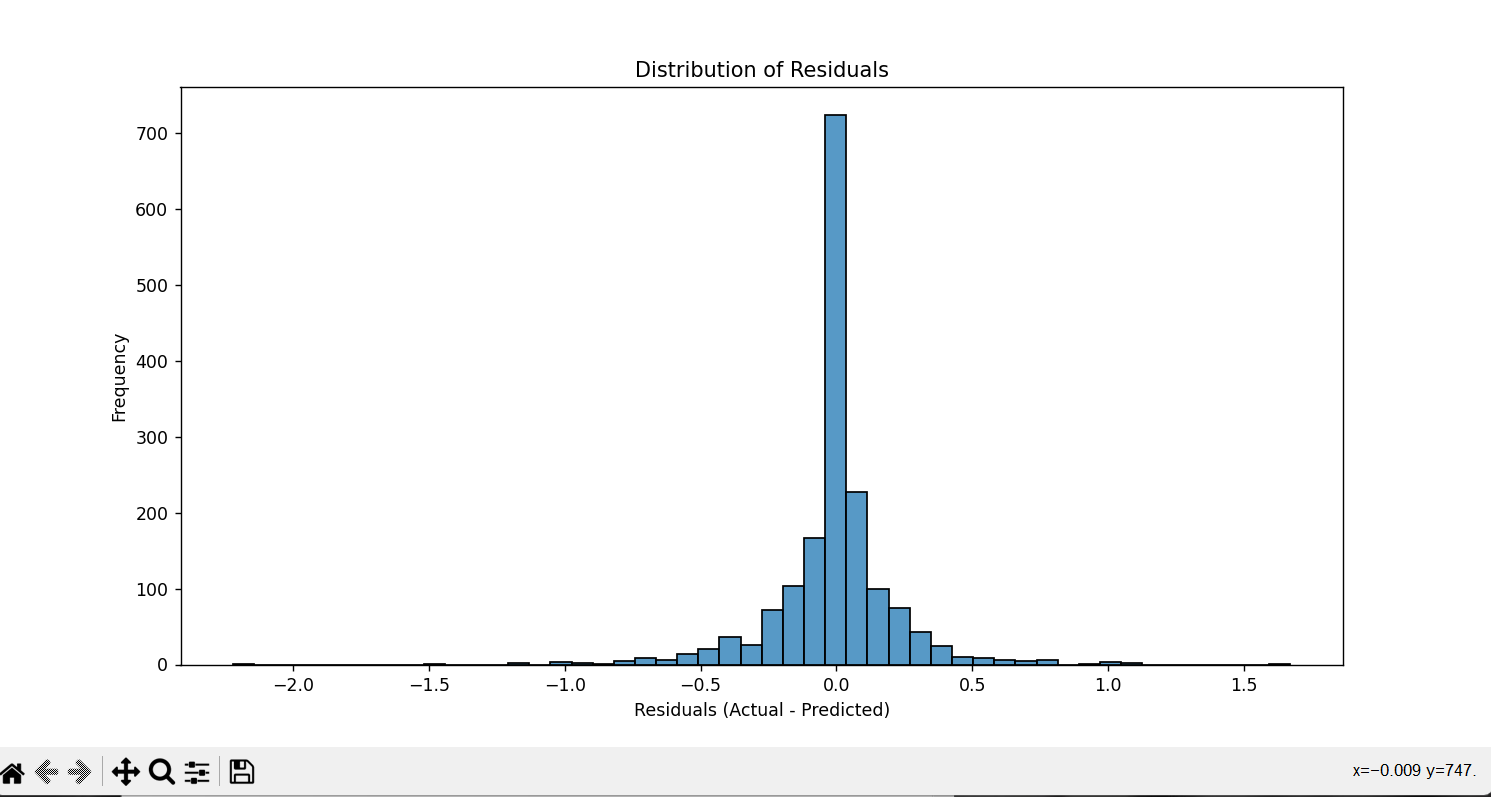
plt.xlabel('Residuals (Actual - Predicted)')

plt.ylabel('Frequency')

plt.title('Distribution of Residuals')

plt.show()

Output:



Conclusion:

Stock price prediction is a multifaceted endeavor that relies on the synergy of feature engineering, model training, and rigorous evaluation. Feature engineering empowers us to extract meaningful insights from complex datasets, guiding our models towards more accurate predictions. Model training, through a variety of algorithms, allows us to capture the intricate dynamics of financial markets. The critical evaluation of model performance using metrics like MAE, MSE, RMSE, and R2, as well as practical tools like cross-validation and backtesting, ensures the reliability of our predictions. Beyond the technical aspects, we recognize the importance of visualization, risk management, and profit and loss analysis in translating model results into actionable strategies. In the ever-evolving world of financial markets, our continuous pursuit of knowledge and adaptability remains essential for informed investment decisions.