Project Title: Stock Price Prediction Using Historical Market Data

Phase 1: Problem Definition and Design Thinking

Problem Definition:

The problem at hand is to develop a predictive model that can forecast stock prices based on historical market data. The primary objective is to create a valuable tool that assists investors in making informed decisions and optimizing their investment strategies. The project encompasses various stages, including data collection, data preprocessing, feature engineering, model selection, model training, and model evaluation.

provided dataset columns: Date, Open, High, Low, Close, Adj Close, and Volume.

1. Data Collection:

Collect historical stock market data, including the following columns:

- Date: The date of the trading day.
- Open: The opening price of the stock on that day.
- High: The highest price reached during the trading day.
- Low: The lowest price reached during the trading day.
- Close: The closing price of the stock on that day.
- Adj Close: The adjusted closing price, which accounts for events such as dividends and stock splits.
- Volume: The trading volume, which represents the number of shares traded on that day.

2. Data Preprocessing:
Clean and preprocess the data:

- Remove duplicate rows to maintain data integrity.
- Handle missing values in any of the columns, possibly by imputing or removing rows with missing data.
- Ensure that all columns are of the appropriate data types, especially the 'Date' column, which should be in a datetime format.
- Normalize or scale the numerical columns if needed.

3. Feature Engineering:
Create additional features that can potentially improve the predictive power of the model:

- Compute various technical indicators like moving averages (e.g., 10-day moving average, 50-day moving average), Relative Strength Index (RSI), and Moving Average Convergence Divergence (MACD).
- Calculate daily price changes or percentage changes.
- Introduce lagged variables, where you use previous day's data to predict the next day's closing price.

4. Model Selection:
Choose a suitable time series forecasting model based on the nature of the data and problem:

- You can explore models such as ARIMA (AutoRegressive Integrated Moving Average), GARCH (Generalized Autoregressive Conditional Heteroskedasticity), or machine learning models like LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit) for more complex patterns.

5. Model Training:
Train the selected model using historical data. The model should take into account the features generated in the previous step, such as technical indicators and lagged variables.
6. Evaluation:
Evaluate the model's performance using appropriate time series forecasting metrics:

- Calculate Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), or any other relevant metrics.
- Perform a visual inspection of the predicted vs. actual prices using plots and graphs to assess how well the model captures trends and patterns.

This structured approach will help you build a predictive model for stock price forecasting based on the provided dataset. The choice of specific techniques and models will depend on the characteristics of the data and your project's objectives

Project code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

.These lines import essential libraries:

- numpy (as np): A library for numerical computations.
- pandas (as pd): A library for data manipulation and analysis.
- matplotlib.pyplot (as plt): A library for creating visualizations and plots.

```
data = pd.read_csv("MSFT.csv")
print("\n")
```

- This code reads a CSV file named "MSFT.csv" using pandas' read_csv function and stores the data in the data DataFrame.
- It also prints a newline character to add some space before the next output.

```
data_cleaned = data.drop_duplicates()  # Remove duplicates
data_cleaned = data_cleaned.dropna()   # Remove rows with missing values
print(data_cleaned)
print(data_cleaned.head())
```

These lines perform data preprocessing steps:

- data.drop_duplicates(): Removes duplicate rows from the DataFrame data and assigns the result to data_cleaned.
- data_cleaned.dropna(): Removes rows with missing values (NaN) from data_cleaned.
- Finally, it prints the cleaned DataFrame and its first five rows.

```
closing_price = data['Close']
print(closing_price.head())
```

- This code creates a new Series named closing_price containing the 'Close' column from the original DataFrame data.
- It then prints the first five entries of this Series, showing the closing prices of the stock.

```
data['EMA_10'] = data['Close'].ewm(span=10, adjust=False).mean()
data['Rolling_20D_Std'] = data['Close'].rolling(window=20).std()
print(data[['EMA_10', 'Rolling_20D_Std']])
```

These lines perform feature engineering:

- data['EMA_10'] = data['Close'].ewm(span=10, adjust=False).mean(): Calculates the 10-day Exponential Moving Average (EMA) of the 'Close' column and stores it in a new 'EMA_10' column in the DataFrame.
- data['Rolling_20D_Std'] = data['Close'].rolling(window=20).std(): Computes the 20-day rolling standard deviation of the 'Close' column and stores it in a new 'Rolling_20D_Std' column in the DataFrame.
- Finally, it prints the 'EMA_10' and 'Rolling_20D_Std' columns of the DataFrame.

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(train_data['Close'], order=(1, 1, 1))
model_fit = model.fit()
print(model_fit.summary())
```

These lines involve model selection and training using ARIMA (AutoRegressive Integrated Moving Average):

- from statsmodels.tsa.arima.model import ARIMA: Imports the ARIMA model from the statsmodels library.
- model = ARIMA(train_data['Close'], order=(1, 1, 1)): Initializes an ARIMA model with the specified order (1, 1, 1) and uses the 'Close' column of train_data as input data.
- model_fit = model.fit(): Fits the ARIMA model to the training data.

- print(model_fit.summary()): Prints a summary of the model's parameters and statistical information.

```
from sklearn.metrics import mean_absolute_error
predictions = model_fit.forecast(steps=len(test_data))
mae = mean_absolute_error(test_data['Close'], predictions)
print("Mean Absolute Error (MAE):", mae)
```

- These lines evaluate the ARIMA model's performance:
  - from sklearn.metrics import mean_absolute_error: Imports the mean_absolute_error function from scikit-learn's metrics module.
  - predictions = model_fit.forecast(steps=len(test_data)): Generates predictions using the trained model for a number of steps equal to the length of test_data.
  - mae = mean_absolute_error(test_data['Close'], predictions): Computes the Mean Absolute Error (MAE) between the actual 'Close' prices in test_data and the predicted prices.
  - Finally, it prints the calculated MAE.

This code demonstrates the steps of data preprocessing, feature engineering, model selection, and evaluation for stock price prediction using the ARIMA model. It's essential to note that the code assumes the existence of train_data and test_data, which are not shown in the provided script. These datasets are typically split from the original data for training and testing purposes in time series forecasting.