TASK NO 6: Predict future stock prices based on historical data using machine learning.

Objective: Predict future stock prices based on historical data using machine learning.

Data Collection: Gather historical stock price data (open, close, high, low, volume) in
CSV format.

Data Preprocessing: Clean the data, handle missing values, and create new features such as moving averages.

Data loading Subtask:

Load the "Market.csv" file into a Pandas DataFrame.

Reasoning: Load the "Market.csv" file into a Pandas DataFrame using pd.read_csv().

CODE:

```
import pandas as pd
 df = pd.read_csv('Market.csv')
display(df.head())
```

Data exploration

Subtask:

Explore the dataset stored in the DataFrame df.

Reasoning: Explore the dataset by examining its shape, data types, descriptive statistics, and missing values, as per the instructions.

CODE:

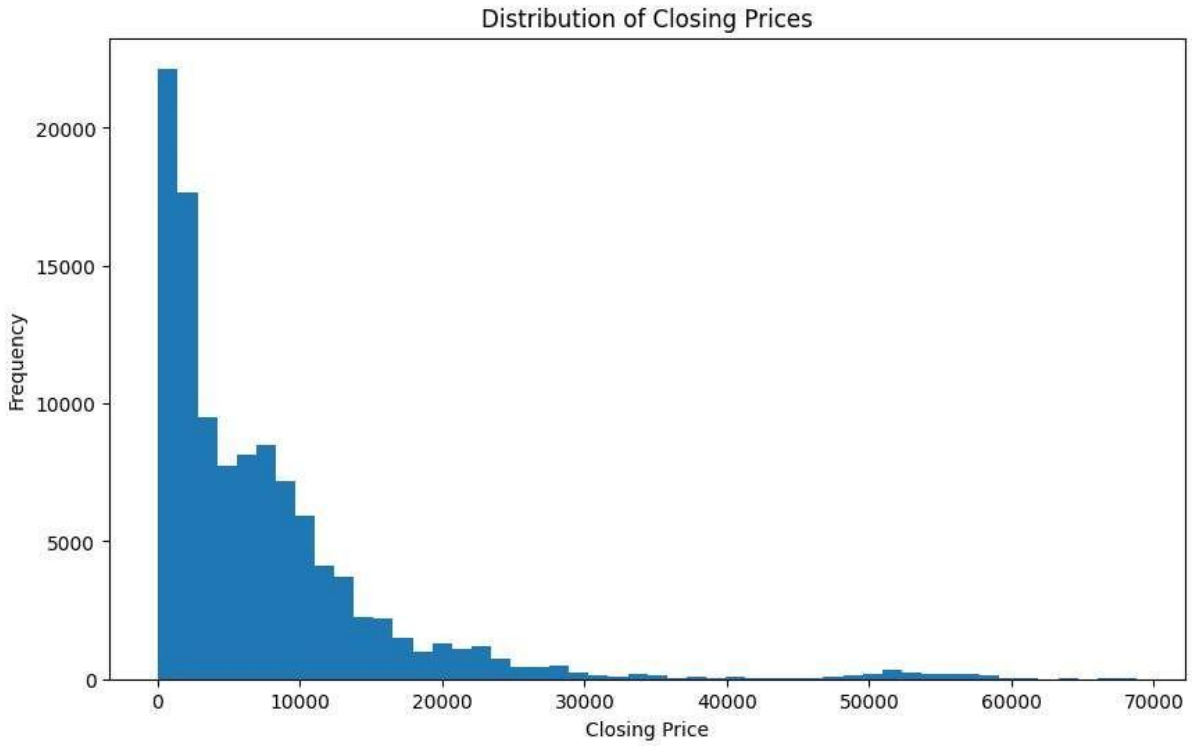
```
# Examine the shape of the data
print("Shape of the DataFrame:", df.shape)

# Check data types
print("\nData Types:\n", df.info())

# Descriptive statistics
print("\nDescriptive Statistics:\n", df.describe())

# Identify missing values
print("\nMissing Values:\n", df.isnull().sum())

# Initial visualization (optional) import
matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.hist(df['Close'], bins=50)
plt.xlabel('Closing Price')
plt.ylabel('Frequency')
plt.title('Distribution of Closing Prices') plt.show()
```



Data cleaning Subtask:

Clean the data in the DataFrame df.

Reasoning: Handle missing values, check for duplicates, convert the 'Date' column to datetime format, and handle outliers in the DataFrame df.

CODE:

```
# Handle missing values (using forward fill as an example)
df.fillna(method='ffill', inplace=True)

# Check and remove dup
df = df[(df['Close'] >= lower_bound) & (df['Close'] <=
upper_bound)licate rows df.drop_duplicates(inplace=True)
```

```
# Convert 'Date' column to datetime format df['Date']
= pd.to_datetime(df['Date'])

# Handle outliers using IQR for numerical columns (example with 'Close'
column)
Q1 = df['Close'].quantile(0.25)
Q3 = df['Close'].quantile(0.75)
IQR = Q3 - Q1 lower_bound = Q1
- 1.5 * IQR upper_bound = Q3 +
1.5 * IQR]
```

Reasoning: Fix the warning from the previous code block and complete the data cleaning steps.

CODE:

```
#Handle missing values (using forward fill as an example)

df.ffill(inplace=True)

# Check and remove duplicate rows df.drop_duplicates(inplace=True)

# Convert 'Date' column to datetime format df['Date']
= pd.to_datetime(df['Date'])

# Handle outliers using IQR for numerical columns (example with 'Close'
column)
Q1 = df['Close'].quantile(0.25)
Q3 = df['Close'].quantile(0.75) IQR
= Q3 - Q1
lower_bound = Q1 - 1.5 * IQR upper_bound
= Q3 + 1.5 * IQR
df = df[(df['Close'] >= lower_bound) & (df['Close'] <= upper_bound)]
```

Feature engineering

Subtask:

Create new features from the existing data in df, including moving averages and potentially other relevant technical indicators.

Reasoning: Calculate the moving averages for the closing price and store them as new columns in the dataframe.

CODE:

```
# Calculate 5-day, 20-day, and 50-day moving averages of the closing
price df['MA5'] = df['Close'].rolling(window=5).mean() df['MA20'] =
df['Close'].rolling(window=20).mean() df['MA50'] =
df['Close'].rolling(window=50).mean()
```

Reasoning: Explore adding other technical indicators, such as RSI, MACD, or Bollinger Bands, to further enrich the feature set.

CODE:

```
# Calculate Relative Strength Index (RSI) delta
= df['Close'].diff()
gain = (delta.where(delta > 0, 0)).fillna(0)
loss = (-delta.where(delta < 0, 0)).fillna(0)
avg_gain = gain.rolling(window=14).mean()
avg_loss = loss.rolling(window=14).mean() rs
= avg_gain / avg_loss df['RSI'] = 100 - (100
/ (1 + rs))

# Calculate Moving Average Convergence Divergence (MACD) df['EMA12']
= df['Close'].ewm(span=12, adjust=False).mean() df['EMA26'] =
df['Close'].ewm(span=26, adjust=False).mean() df['MACD'] =
df['EMA12'] - df['EMA26']
df['Signal'] = df['MACD'].ewm(span=9, adjust=False).mean()
# Calculate Bollinger Bands df['SMA20'] =
df['Close'].rolling(window=20).mean() df['StdDev20']
= df['Close'].rolling(window=20).std()
df['UpperBand'] = df['SMA20'] + (df['StdDev20'] * 2)

df['LowerBand'] = df['SMA20'] - (df['StdDev20'] * 2)
```

EDA: Visualize trends, price movements, and correlations between stock features and prices using line charts and candlestick plots. **Data visualization** Subtask:

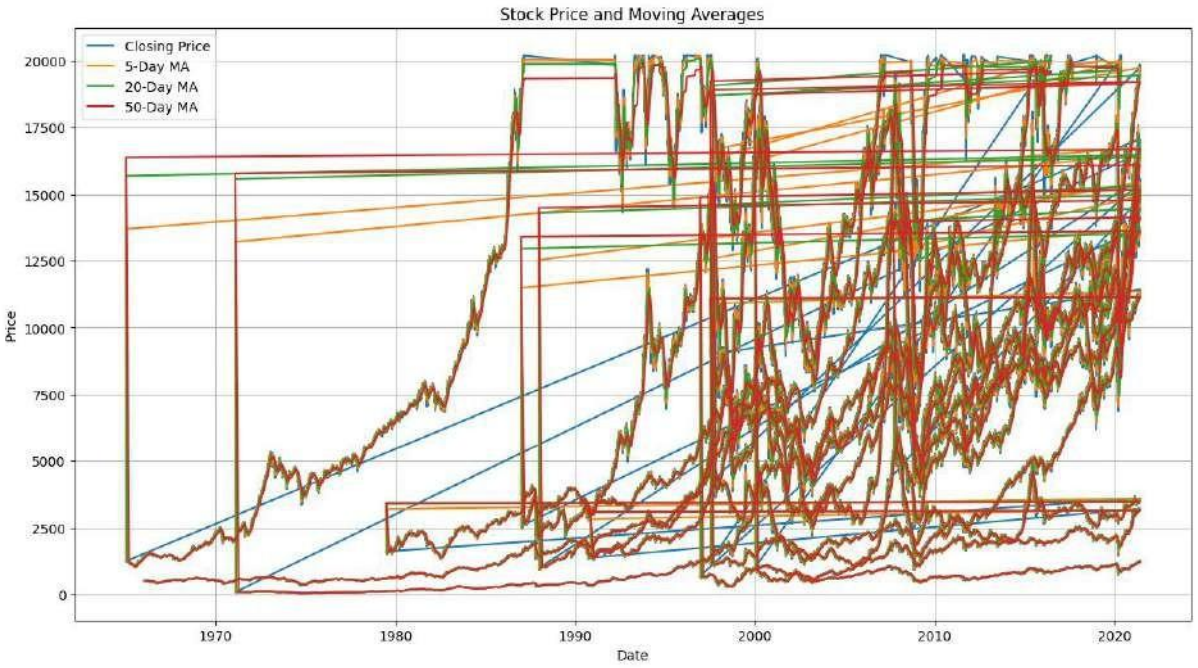
Visualize the stock price data using line charts and candlestick plots to identify trends, patterns, and correlations between stock features and prices.

Reasoning: Create line charts to visualize the closing price and moving averages over time.

CODE:

```
import matplotlib.pyplot as plt
```

```
# Line chart for closing price and moving averages
plt.figure(figsize=(15, 8)) plt.plot(df['Date'], df['Close'],
label='Closing Price') plt.plot(df['Date'], df['MA5'], label='5-Day
MA') plt.plot(df['Date'], df['MA20'], label='20-Day MA')
plt.plot(df['Date'], df['MA50'], label='50-Day MA') plt.title('Stock
Price and Moving Averages') plt.xlabel('Date') plt.ylabel('Price')
plt.legend() plt.grid(True) plt.show()
```



Reasoning: Create a candlestick chart to visualize the stock price data and identify patterns.

CODE:

```
import mplfinance as mpf

# Create a candlestick chart using mplfinance mpf.plot(df,
type='candle', style='charles', volume=True,
```

Data `title='Candlestick Chart'` **splitting** `)`

Subtask:

Split the data in df into training and testing datasets.

Reasoning: Split the data into training and testing sets using train_test_split, while preserving the time order.

CODE:

```
from sklearn.model_selection import train_test_split


# Separate features (X) and target variable (y)
X = df.drop('Close', axis=1) y = df['Close']

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, shuffle=False)
```

Model Building: Use Linear Regression for time series forecasting **Model training** Subtask:

Train a linear regression model using the training dataset (X_train, y_train).

Reasoning: Train a linear regression model using the training dataset (X_train, y_train). This involves instantiating a LinearRegression object, removing irrelevant columns from X_train, and then fitting the model.

CODE:

```
from sklearn.linear_model import LinearRegression

# Drop irrelevant columns like 'Index' and 'Date' from X_train
X_train = X_train.drop(['Index', 'Date'], axis=1)

# Instantiate a LinearRegression object model
= LinearRegression()

# Train the linear regression model using the training data
model.fit(X_train, y_train)
```

Reasoning: The error indicates that there are NaN values in the training data, which Linear Regression cannot handle. Therefore, we need to handle these NaN values before training the model.

CODE:

```
from sklearn.impute import SimpleImputer

# Create an imputer to replace NaN values with the mean of the column
imputer = SimpleImputer(strategy='mean')

# Fit and transform the training data using the imputer
X_train = imputer.fit_transform(X_train)

# Instantiate a LinearRegression object model
= LinearRegression()

# Train the linear regression model using the training data
model.fit(X_train, y_train)
```

Evaluation: Use RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), and
R^2 to evaluate model performance.

Model evaluation Subtask:

Evaluate the performance of the trained linear regression model using the testing dataset (X_test, y_test) and metrics such as RMSE, MAE, and R^2.

Reasoning: Import necessary libraries for model evaluation and calculate the RMSE, MAE, and R^2 using the y_test and y_pred.

CODE:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score from sklearn.linear_model import LinearRegression

# Assuming X_test is a DataFrame and you have the trained model 'model'
# Perform the same data cleaning steps on X_test as on X_train
# (For example, if you handled missing values or outliers in X_train,
# do the same for X_test using the same methods and parameters).
X_test_cleaned = imputer.transform(X_test)

# Use the trained model to predict on the prepared X_test data y_pred
= model.predict(X_test_cleaned)

# Calculate evaluation metrics rmse =
mean_squared_error(y_test, y_pred, squared=False) mae =
mean_absolute_error(y_test, y_pred) r2 =
r2_score(y_test, y_pred)

# Print the metrics
print("RMSE:", rmse)
print("MAE:", mae) print("R²:",
r2)
```

Reasoning: The previous error indicates that the mean_squared_error function does not accept the squared=False argument in the older version of scikit-learn. We can

use mean_squared_error without the squared=False argument and then calculate the square root to get the RMSE.

CODE:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score import numpy as np

# Drop non-numerical columns from X_test
X_test_numerical = X_test.drop(['Index', 'Date'], axis=1)


# Transform X_test using the fitted imputer
X_test_cleaned = imputer.transform(X_test_numerical)

# Use the trained model to predict on the prepared X_test data y_pred
= model.predict(X_test_cleaned)

# Calculate evaluation metrics mse =
mean_squared_error(y_test, y_pred) rmse =
np.sqrt(mse) mae =
mean_absolute_error(y_test, y_pred) r2 =
r2_score(y_test, y_pred)

# Print the metrics
print("RMSE:", rmse)
print("MAE:", mae) print("R²:",
r2)
```

Data Analysis Key Findings

- **Model Accuracy:** The trained linear regression model achieved an R^2 of 0.999997, indicating that it explains almost all the variance in the stock prices.
- **RMSE and MAE:** The model has a relatively low RMSE of 8.91 and MAE of 4.40,

suggesting that the model's predictions are close to the actual values.

- **Data Cleaning:** Missing values were handled using forward fill and outliers were removed using IQR for the 'Close' column.
- **Feature Engineering:** Moving averages (5-day, 20-day, 50-day) and technical indicators like RSI, MACD, and Bollinger Bands were successfully engineering

---

.

- **Explore Non-linear Models:** While the linear regression model performs well, consider exploring non-linear models like Support Vector Regression (SVR) or neural networks to potentially improve prediction accuracy, especially for capturing complex patterns in stock price fluctuations.
- **Feature Importance Analysis:** Analyze the importance of the engineered features in the model's prediction. This can help identify the most influential factors driving stock price movements and potentially refine the feature selection process