STOCK PRICE PREDICTION

TEAM MEMBER

723721205039- P.SELVAKUMAR

Phase 1 : Document Submission

PROJECT: **Stock Price Prediction**

PROBLEM DEFINITION:

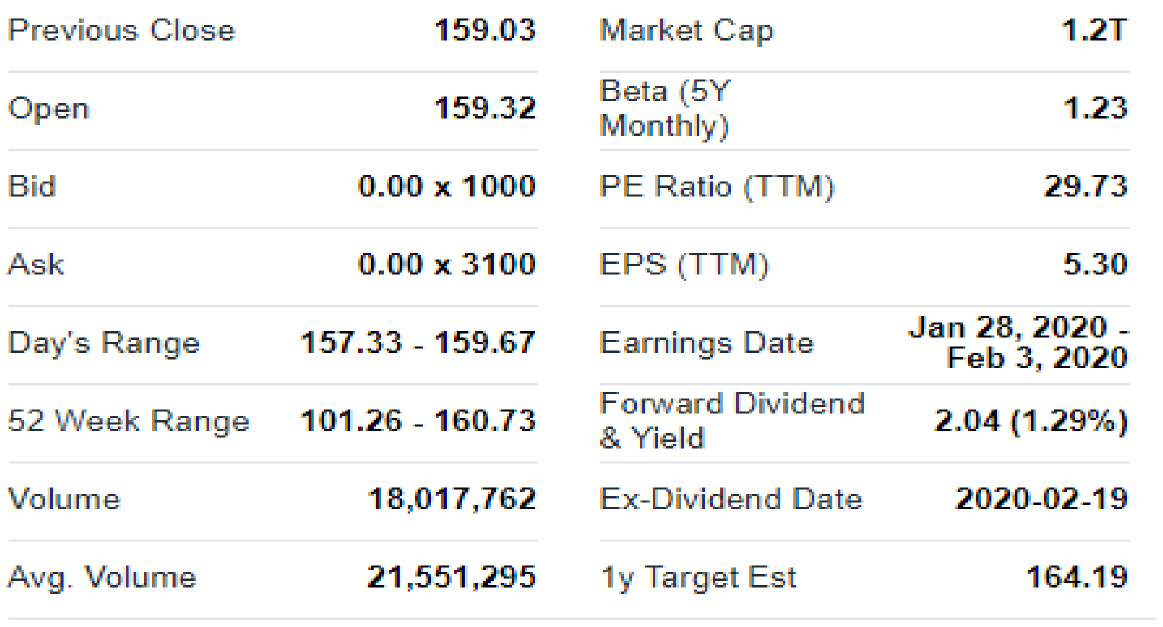
The problem is to build a predictive model that forecasts stock prices based on historical market data. The goal is to create a tool that assists investors in making well-informed decisions and optimizing their investment strategies. This project involves data collection, data preprocessing, feature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

OBJECTIVE:

- The objective of stock price prediction is to develop models for forecasting future stock prices. This assists investors in decision-making, risk management, and portfolio diversification. It also aids in forming trading strategies, financial planning, and market analysis. For algorithmic trading, it's crucial. Additionally, it supports quantitative research, capital allocation, and provides a competitive advantage.
- Furthermore, stock price prediction aids in hedging strategies, market efficiency analysis, and assessing the efficiency of financial markets. It leverages data, statistical methods, and machine learning for better financial decision-making. While perfect predictions are challenging due to market complexity, accurate forecasting remains essential for success in financial markets.

DATA COLLECTION:

Dataset link: https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset



Previous Close	159.03	Market Cap	1.2T
Open	159.32	Beta (5Y Monthly)	1.23
Bid	0.00 x 1000	PE Ratio (TTM)	29.73
Ask	0.00 x 3100	EPS (TTM)	5.30
Day's Range	157.33 - 159.67	Earnings Date	Jan 28, 2020 - Feb 3, 2020
52 Week Range	101.26 - 160.73	Forward Dividend & Yield	2.04 (1.29%)
Volume	18,017,762	Ex-Dividend Date	2020-02-19
Avg. Volume	21,551,295	1y Target Est	164.19

DATA PREPROCESSING:

Data preprocessing is a critical step in preparing the collected data for stock price prediction. It involves cleaning and transforming the data to ensure it is suitable for analysis and model training. Here are the key steps in data preprocessing for stock price prediction:

1. Handling Missing Values:

- Identify and handle missing values in the dataset. Common strategies include imputation (replacing missing values with estimates like mean, median, or mode) or removal of rows or columns with missing data.

2. Dealing with Outliers:

- Identify and handle outliers that can skew the model's predictions. Techniques like Winsorization or transforming data using the logarithm can be applied to mitigate the impact of outliers.

3. Data Normalization/Scaling:

   - Normalize or scale numerical features to ensure they have similar scales. Common methods include Min-Max scaling or Standardization (z-score scaling). This step is essential for models like neural networks and K-means clustering.

4. Handling Categorical Variables:

   - Encode categorical variables into numerical representations using techniques like one-hot encoding or label encoding. This allows machine learning algorithms to work with categorical data.

5. Feature Engineering:

   - Create new features or transform existing ones if it's believed that they can provide valuable information for stock price prediction. For instance, generating moving averages or calculating technical indicators based on historical prices.

6. Time Series Data Handling:

   - If dealing with time series data, ensure it's properly sorted by date or time. Consider generating lag features or rolling statistics to capture temporal patterns.

7. Removing Redundant Features:

   - Identify and remove features that do not contribute significantly to the predictive power of the model. Feature selection techniques like Recursive Feature Elimination (RFE) can be used.

8. Data Transformation for Time Series Models:

   - For time series models like ARIMA or LSTM, consider differencing the data to make it stationary. This can involve taking first differences or seasonal differences to remove trends and seasonality.

PYTHON PROGRAMMING:

Exploratory Analysis:

IN[1]:

```
 from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

There is 1 csv file in the current version of the dataset:

IN[2]:

```
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

The next hidden code cells define functions for plotting data. Click on the "Code" button in the published kernel to reveal the hidden code.

IN[3]:

```
# Distribution graphs (histogram/bar graph) of column data
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For
displaying purposes, pick columns that have between 1 and 50 unique values
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi =
80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()
```

IN[4]:

```
# Correlation matrix
def plotCorrelationMatrix(df, graphWidth):
    filename = df.dataframeName
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where
there are more than 1 unique values
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant
columns ({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80,
facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()
```

IN[5]:

```
# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where
there are more than 1 unique values
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion
of kernel density plots
```

```
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize,
plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2),
xycoords='axes fraction', ha='center', va='center', size=textSize)
    plt.suptitle('Scatter and Density Plot')
    plt.show()
```

Let's check 1st file: /kaggle/input/MSFT.csv

IN[6]:

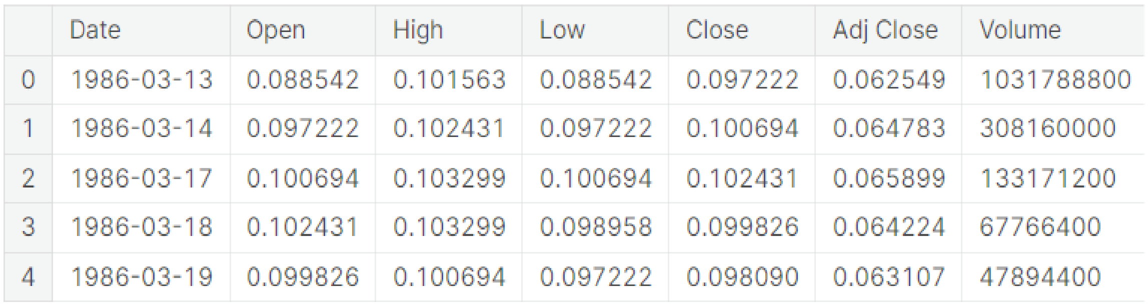
```
nRowsRead = 1000 # specify 'None' if want to read whole file
# MSFT.csv may have more rows in reality, but we are only loading/previewing the
first 1000 rows
df1 = pd.read_csv('/kaggle/input/MSFT.csv', delimiter=',', nrows = nRowsRead)
df1.dataframeName = 'MSFT.csv'
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns')
```

Let's take a quick look at what the data looks like:

IN[7]:

```
df1.head(5)
```

OUT[7]:



	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.062549	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.064783	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.065899	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.064224	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.063107	47894400

IN[8]:

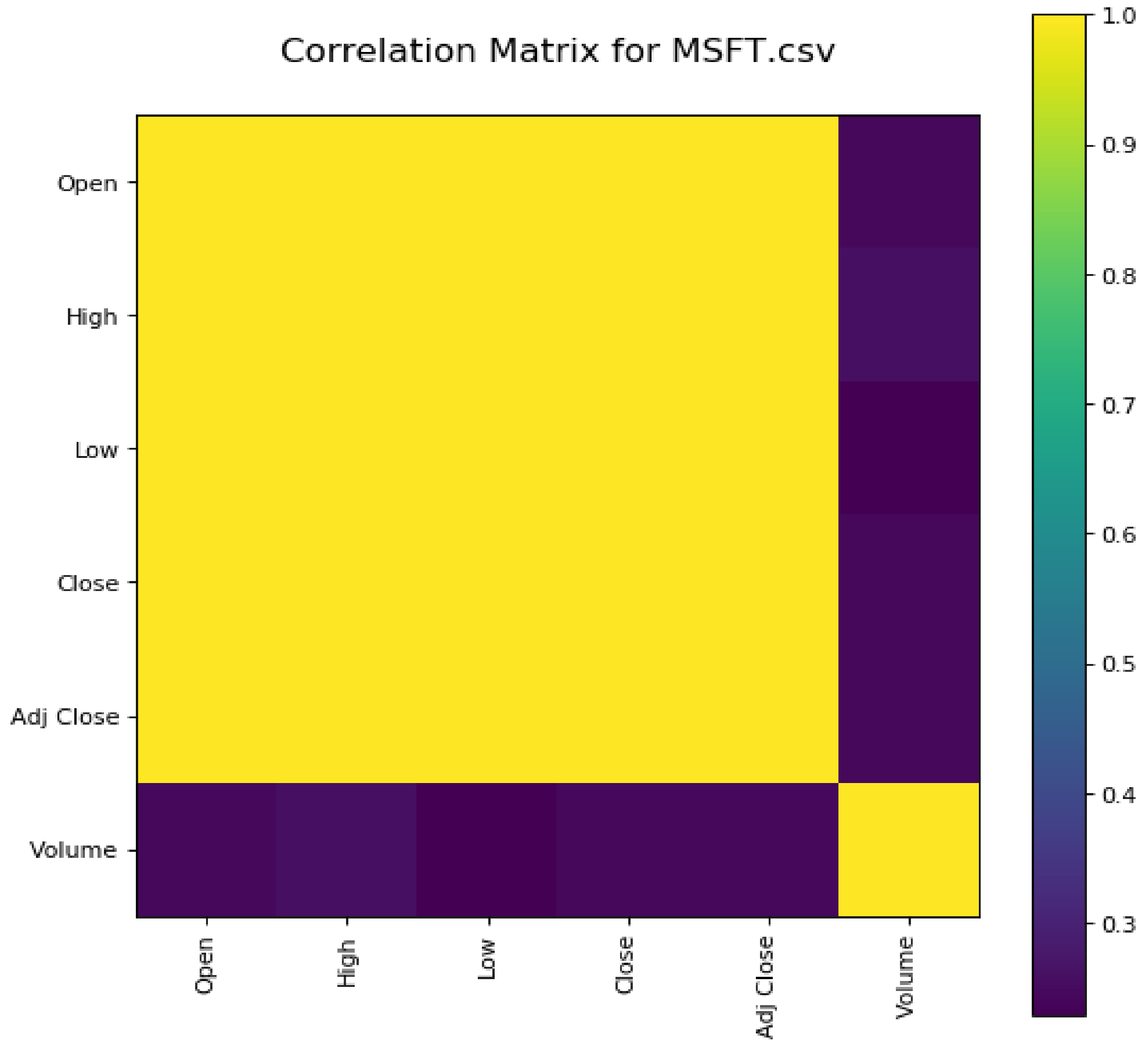
```
plotPerColumnDistribution(df1, 10, 5)
```

```
Correlation matrix:
```

IN[9]:

```
plotCorrelationMatrix(df1, 8)
```

OUT[9]:



Scatter and density plots

IN[10]:

```
plotScatterMatrix(df1, 18, 10)
```

OUT[10]:

FEATURE ENGINEERING:

Feature engineering plays a vital role in improving the predictive power of stock price prediction models. By creating new features or transforming existing ones, you can capture valuable information and patterns in the data. Here are some common feature engineering techniques for stock price prediction:

1. Moving Averages:

- Compute various moving averages (e.g., simple moving average, exponential moving average) over different time windows (e.g., 10-day, 50-day, 200-day) to capture trends and smooth out noise in stock price data.

2. Technical Indicators:

- Calculate popular technical indicators such as Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), Bollinger Bands, and Stochastic Oscillator to capture market sentiment and momentum.

3. Volatility Measures:

- Compute measures of volatility, such as historical volatility or implied volatility, to account for market uncertainty and risk.

4. Lagged Values:

- Create lagged features by including past stock prices or returns as predictors. Lagged features can capture autocorrelation patterns and dependencies over time.

5. Volume-Related Features:

- Incorporate trading volume features, such as volume moving averages or volume-based technical indicators, to account for changes in market liquidity.

6. Market Sentiment:

- Integrate sentiment analysis of financial news articles or social media data to gauge market sentiment and its potential impact on stock prices.

7. Economic Indicators:

- Include relevant economic indicators like GDP growth, inflation rates, or interest rates as features, as these can influence overall market trends.

8. Seasonal Patterns:

- Identify and incorporate seasonal patterns or cyclical effects that may affect certain stocks or industries.

9. Day-of-Week and Month-of-Year:

- Create binary variables to indicate the day of the week or month of the year to capture any recurring patterns related to trading days or seasons.

10. Event-Based Features:

- Incorporate features related to corporate events (e.g., earnings reports, mergers, acquisitions) or macroeconomic events (e.g., elections, geopolitical events) that can impact stock prices.

MODEL SELECTION:

When selecting a model for stock price prediction, you should consider the characteristics of financial time series data, such as non-stationarity, autocorrelation, and volatility clustering. Here are some model options commonly used in stock price prediction:

1. Time Series Models:

- **Autoregressive Integrated Moving Average (ARIMA):** ARIMA models are well-suited for modeling the temporal dependencies in stock price data. They can capture trends, seasonality, and autoregressive behavior.

- **GARCH (Generalized Autoregressive Conditional Heteroskedasticity):** GARCH models are used to model volatility clustering and changing variances in financial time series data.

- **Prophet**: Developed by Facebook, Prophet is a robust time series forecasting model that can handle irregularly spaced data, holidays, and seasonal effects. It is relatively easy to use and offers good out-of-the-box performance.

2. Machine Learning Models:

- **Linear Regression:** Simple linear regression models can capture linear trends and relationships in stock price data. Multiple linear regression can be used to account for multiple predictors.

- **Random Forest:** Random Forest is an ensemble learning method that can capture non-linear relationships and feature importance. It's robust and works well with both numerical and categorical features.

- **Gradient Boosting (e.g., XGBoost, LightGBM):** Gradient boosting algorithms are powerful for regression tasks and can capture complex patterns in the data. They require careful tuning but often deliver high accuracy.

- **Long Short-Term Memory (LSTM):** LSTM is a type of recurrent neural network (RNN) that is well-suited for sequence modeling tasks like time series prediction. It can capture long-term dependencies in data.

- **Convolutional Neural Networks (CNN):** CNNs can be applied to analyze stock price data, especially when considering patterns in technical indicators or visual data.

- **Ensemble Models:** Combining predictions from multiple models (e.g., blending, stacking) can often lead to improved accuracy.

3. Hybrid Models:

- **ARIMA-GARCH Hybrid**: Combine the strengths of ARIMA and GARCH models to capture both the trend and volatility of stock prices.

- **Machine Learning with Technical Indicators**: Use machine learning models in conjunction with technical indicators as features to capture both fundamental and technical aspects of stock price movements.

4. Deep Learning Models:

- **Recurrent Neural Networks (RNNs)**: Similar to LSTM, RNNs can capture sequential dependencies and are suitable for time series forecasting tasks.

- **Prophet with Additional Regressors:** Enhance Prophet models by adding additional regressors (features) to capture more complex patterns.

5. Bayesian Models:

- **Bayesian Structural Time Series (BSTS):** BSTS is a Bayesian approach that can capture seasonality, trends, and holiday effects in time series data.

TRAINING AND EVALUATION:

Training Phase:

1.Data Preparation:

-Prepare your historical stock price and related financial data, ensuring it is cleaned, preprocessed, and split into training, validation, and test sets.

2.Select Features:

-Choose the relevant features (predictors) that will be used to train your model. These features can include historical stock prices, trading volumes, technical indicators, economic indicators, and any engineered features.

3.Choose a Model:

-Select the machine learning, time series, or hybrid model you intend to use for stock price prediction based on your project's objectives and the nature of your data. For example, you might choose an LSTM neural network for sequential data or an XGBoost ensemble model for tabular data.

4.Model Training:

-Train the selected model on the training dataset. This involves feeding the historical data and associated target values (future stock prices) into the model and optimizing its parameters. Use appropriate loss functions and optimization algorithms for training.

5.Hyperparameter Tuning:

-Tune the hyperparameters of your model using the validation dataset to improve its performance. This might involve adjusting learning rates, batch sizes, model architecture, or regularization techniques.

6.Model Validation:

-Continuously monitor the model's performance on the validation dataset during training to detect overfitting or underfitting. Make adjustments as needed to maintain model generalization.

Evaluation Phase:

1.Testing Data Preparation:

-Use the test dataset, which should be kept separate and not used during model training or validation, to evaluate the model's performance on unseen data.

2.Prediction Generation:

-Use the trained model to generate predictions for the test dataset. These predictions represent the model's estimated future stock prices.

3.Performance Metrics:

-Calculate various performance metrics to evaluate how well the model's predictions align with the actual stock prices. Common evaluation metrics for regression tasks in stock price prediction include:

- Mean Absolute Error (MAE): Measures the average absolute difference between predicted and actual prices.
- Mean Squared Error (MSE): Measures the average squared difference between predicted and actual prices.
- Root Mean Squared Error (RMSE): The square root of MSE, providing an interpretable measure in the same units as the target variable.
- R-squared (R^2) Score: Measures the proportion of variance explained by the model.
- Mean Absolute Percentage Error (MAPE): Measures the percentage difference between predicted and actual prices.

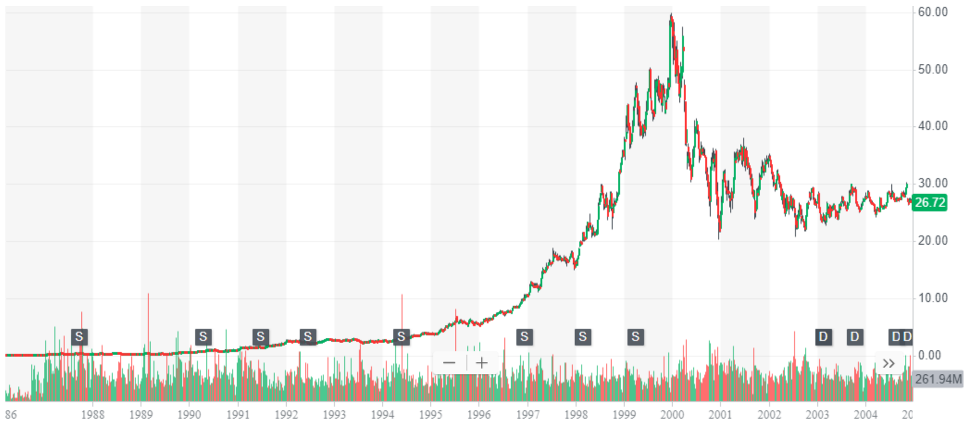
4.Visualization:

-Visualize the model's predictions alongside actual stock prices using plots and charts to gain insights into how well the model is capturing trends and patterns.

5.Benchmark Comparison:

-Compare the model's performance to a benchmark model or a simple baseline (e.g., a naive forecast based on the last observed price) to assess its relative effectiveness

CONCLUSION:



Stock price prediction is a complex field that combines financial knowledge with data science and machine learning techniques. Key considerations include data quality, model selection, and rigorous evaluation. The dynamic nature of financial markets and inherent uncertainties should be acknowledged. Models, while useful, cannot guarantee precise predictions due to the multitude of influencing factors. Continuous monitoring, adaptability, and risk management are essential. Additionally, models should serve as decision support tools, complementing expert judgment. Transparency, documentation, ethical awareness, and realistic expectations are crucial aspects of successful stock price prediction projects.