STOCK PRICE PREDICTION

PROBLEM STATEMENT

The problem at hand is related to stock price prediction for a particular stock (Microsoft, in this case) using historical stock data. The goal is to create a predictive model that can forecast future stock prices based on historical data and evaluate the performance of the model in terms of accuracy and error metrics.

DESIGN THINKING PROCESS

1. **Empathize:** The need for accurate stock price predictions for investors, traders, or financial professionals is analyzed, and user pain points related to stock price forecasting, such as the challenge of making informed investment decisions, are identified. The preferences and requirements of users who rely on stock price predictions for financial decision-making are investigated.
2. **Define:** The problem is defined as follows: "A stock price prediction model that can provide reliable forecasts based on historical data to support investment strategies is required by investors and financial professionals." User needs, including the demand for accurate and interpretable stock price predictions, and the ability to customize model parameters for different financial assets and markets, are specified.
3. **Ideate:** Innovative features and concepts for stock price prediction are brainstormed, such as the development of models that offer transparency by explaining the factors influencing stock price predictions (interpretability). The implementation of a system for continuously updating predictions as new market data becomes available to facilitate timely decision-making (real-time predictions) is considered. Allowing users to tailor model settings, time horizons, and asset classes to align with their investment strategies (customization) is explored. Additionally, the incorporation of portfolio optimization techniques to suggest diversified investment strategies based on predicted stock prices (portfolio optimization) is proposed.
4. **Prototype:** Prototypes for the stock price prediction system are built, including interactive dashboards displaying real-time stock price predictions and underlying factors. User-friendly interfaces for customizing model parameters, selecting assets, and specifying timeframes are designed. Explanatory visualizations demonstrating how different features influence stock price predictions are created. Integration with financial data APIs to provide real-time market data is established. Options for backtesting various investment strategies based on historical predictions are implemented.
5. **Test:** Usability testing and validation are conducted with users from the financial domain, including investors, traders, or professionals interested in stock price predictions.

How users interact with the prototype is observed, and feedback on usability, interpretability, and the overall user experience is gathered. The prototype is iteratively refined based on user feedback, addressing any usability issues and customization preferences tailored to the financial industry.

PHASES OF DEVELOPMENT

1. **Data Collection**: In this case, the data is loaded from a CSV file ('MSFT.csv'). It's essential to gather historical stock price and volume data for Microsoft to build the predictive model.
2. **Data Preprocessing**:
   - **Handling Missing Values**: Missing values in the dataset are handled using the mean imputation strategy. Any columns with missing values are filled with the mean values of the respective columns.
   - **Feature Selection**: Relevant columns are selected ('Date', 'Close', 'Volume') for the analysis.
   - **Data Transformation**: The 'Date' column is converted into a datetime format and set as the index. Lag features are created by shifting the 'Close' prices by one day. A rolling mean feature is also calculated. Finally, the data is split into training and testing sets.
3. **Feature Scaling**: MinMaxScaler is used to scale the features, ensuring that they fall within a specific range (typically between 0 and 1). This step is necessary for linear regression and some other machine learning algorithms.
4. **Model Training**:
   - A linear regression model is chosen for stock price prediction. The model is created and trained on the training dataset (X_train, y_train).
5. **Model Evaluation**:
   - The model is evaluated using Mean Absolute Error (MAE) and R-squared (R2) score. These metrics help assess the accuracy and predictive power of the model.
   - The code also calculates accuracy based on a price change threshold (a prediction is considered correct if it's within a certain percentage of the actual price change).
   - An example prediction for a new data point is demonstrated using the trained model

DATASET DESCRIPTION

- The dataset appears to be historical stock price and volume data for Microsoft (MSFT).
- Columns in the dataset include 'Date', 'Close' (closing stock price), and 'Volume' (trading volume).
- The data appears to be time-series data as it's indexed by date.

DATA PREPROCESSING STEPS

1. **Loading the Dataset:**
    - The code begins by importing the necessary libraries, such as pandas for data manipulation and scikit-learn for preprocessing and modeling.
    - It loads the dataset from a CSV file named 'MSFT.csv' using the **pd.read_csv()** function.
2. **Handling Missing Values:**
    - The code checks for missing values in the dataset using the **df.isnull().sum()** method to identify columns with missing data.
    - It employs the **SimpleImputer** from scikit-learn with the 'mean' strategy to fill missing values with the mean values of the respective columns. This step ensures that the dataset is complete and ready for analysis.
3. **Selecting Relevant Columns:**
    - The code selects only the relevant columns from the dataset, which include 'Date', 'Close' (closing stock price), and 'Volume' (trading volume).
4. **Data Transformation:**
    - It ensures that the 'Date' column is in datetime format using the **pd.to_datetime()** method.
    - The 'Date' column is set as the index of the DataFrame to make it the time series index.
5. **Feature Engineering (Optional):**
    - The code creates lag features by shifting the 'Close' prices by one day, resulting in a new column named 'Close_Lag1'. This can capture temporal dependencies in the data.
    - A rolling mean of 'Close' prices is calculated with a specified window size, creating a new column called 'Close_Rolling_Mean'. Rolling means are used to smooth out noise in the data and highlight trends.

MODEL TRAINING PROCESS

1. **Data Splitting:**
    - The dataset is split into training and testing sets using **train_test_split** from scikit-learn. The features (X) consist of 'Close_Lag1' and 'Volume', while the target variable (y) is 'Close'. The split ratio is set to 80% for training and 20% for testing.
2. **Feature Scaling:**
    - Feature scaling is applied to the training and testing sets using **MinMaxScaler** from scikit-learn. Min-max scaling transforms the features to fall within a specific range, typically between 0 and 1. This is important for linear regression and other algorithms that are sensitive to feature scales.
3. **Model Selection:**
    - A linear regression model is selected and created using **LinearRegression** from scikit-learn.
4. **Model Training:**
    - The model is trained on the training dataset using the **model.fit()** method with the training features (X_train) and target variable (y_train).
5. **Model Evaluation:**
    - Predictions are made on the test dataset using the trained model with the **model.predict()** method.
    - Mean Absolute Error (MAE) and R-squared (R2) score are calculated using the **mean_absolute_error** and **r2_score** functions from scikit-learn, respectively. These metrics help evaluate the model's performance in terms of accuracy and goodness of fit.
    - The code also assesses accuracy based on a defined price change threshold to determine if predictions are within an acceptable range.
6. **Example Prediction:**
    - An example prediction is demonstrated for a new data point. The code constructs a new DataFrame with feature values and uses the model to predict the 'Close' price for the given data point.

KEY FINDINGS

1. **Data Preprocessing:** The code effectively handles missing values in the dataset by filling them with the mean values. Additionally, it converts the 'Date' column to a

datetime format and creates lag features and a rolling mean to capture temporal patterns in the data.

2. **Feature Scaling:** Feature scaling using MinMaxScaler is applied to ensure that the features fall within a specific range, which is essential for some machine learning algorithms, including linear regression.
3. **Linear Regression Model:** A linear regression model is used for stock price prediction, which is a straightforward choice for a baseline model. The code successfully trains the model using the preprocessed data.
4. **Model Evaluation:** The code calculates Mean Absolute Error (MAE), R-squared (R2) score, and accuracy based on a price change threshold to evaluate the model's performance. These metrics provide insight into the accuracy and reliability of the predictions.

INSIGHTS

1. **Feature Engineering:** The code introduces feature engineering by creating lag features and a rolling mean. This can potentially help capture short-term and long-term trends in stock prices, providing more information for the model.
2. **Visualization:** The code includes visualizations to show the scaling of the 'Close' prices, which is crucial for understanding how the data is transformed for modeling. Additionally, it visualizes the rolling mean feature, which helps users interpret the impact of this feature on predictions.

RECOMMENDATIONS

1. **Advanced Models:** While linear regression is a good starting point, it's recommended to explore more advanced models, especially for time series data. Consider models like ARIMA, GARCH, or machine learning algorithms tailored for time series forecasting, as they may capture complex patterns better.
2. **Hyperparameter Tuning:** Experiment with hyperparameter tuning for the chosen model to optimize its performance. Grid search or random search can be employed to find the best combination of hyperparameters.
3. **Feature Selection:** Evaluate the importance of features in the model and consider removing less relevant features. Feature selection techniques like Recursive Feature Elimination (RFE) or feature importance from tree-based models can be used.
4. **Ensemble Methods:** Explore ensemble methods such as Random Forest or Gradient Boosting to combine multiple models for improved predictive accuracy.

5. **Cross-Validation:** Implement cross-validation techniques to assess the model's performance more robustly. This ensures that the model's performance is reliable and not overfitting to the training data.
6. **Risk Assessment:** For financial applications, it's crucial to assess the model's risk and uncertainty. Consider adding confidence intervals or volatility measures to provide users with insights into the stability of predictions.
7. **Real-World Data:** Use real-world financial data and continuously update the model to reflect changing market conditions. This ensures the model remains relevant and accurate over time.
8. **User Interface:** If this model is intended for use by investors or financial professionals, consider developing a user-friendly interface that allows users to input their preferences, select assets, and receive predictions and insights in a clear and interpretable format.