**Stock Price Prediction using Applied datascience**

**Phase – 05**

**Problem Statement:**

The problem statement for the provided code is to build a predictive model for determining whether to buy or not buy Microsoft stock based on historical stock price data. The dataset contains features like 'Open,' 'High,' 'Low,' 'Close,' and 'Volume' for each trading day, and the target variable is 'target,' which indicates whether it's a good decision to buy (1) or not buy (0) the stock based on the next day's closing price compared to the current day's closing price.

**Design Thinking Process:**

The code appears to follow a basic design thinking process to solve this problem, which can be broken down into the following phases:

**Data Understanding and Preparation:**

Import necessary libraries for data analysis, visualization, and machine learning.

Load historical stock price data from a CSV file into a Pandas DataFrame.

Check the data's basic information, such as data types and missing values.Visualize the data by plotting the stock price over time and examining distribution plots and boxplots to identify outliers.

Perform feature engineering by splitting the 'Date' column, adding features like 'is\_quarter\_end,' and creating new features like 'open-close' and 'low-high.'

Define the target variable 'target' based on the problem statement.

**Data Exploration:**

Visualize the data by creating bar plots to understand trends over the years.

Group the data by 'is\_quarter\_end' to analyze differences between quarter-end and non-quarter-end days.

**Data Preprocessing:**

Standardize and normalize the feature data using StandardScaler.

Split the dataset into training and validation sets.

**Model Selection and Training:**

Choose three different machine learning models: Logistic Regression, Support Vector Classifier (SVC) with a polynomial kernel, and XGBoost Classifier.

Train these models on the training data to learn the patterns in the historical stock price data.

**Model Evaluation:**

Evaluate the trained models using ROC AUC score to measure the accuracy of predictions with soft probabilities.

Visualize the performance of the Logistic Regression model with a confusion matrix.

**Prediction:**

Use the trained XGBoost Classifier model to predict the target variable based on a sample input feature.

**Phases of Development:**

The development of the code follows several distinct phases:

**Data Collection:**

Historical stock price data for Microsoft is loaded from a CSV file.

**Data Preprocessing:**

Data is cleaned and transformed, including handling missing values, creating new features, and splitting the dataset.

**Model Development:**

Three machine learning models are selected, trained, and evaluated.

**Model Evaluation:**

Model performance is evaluated using ROC AUC score and visualized using a confusion matrix.

**Prediction:**

The trained model is used for making predictions on new data.

**Deployment:**

**Dataset Description:**

The dataset is loaded from a CSV file named "MSFT.csv."

The dataset contains the following columns:

**'Date':** The date of the trading day.

**'Open':** The opening price of Microsoft stock on that day.

**'High':** The highest price of the stock during the trading day.

**'Low':** The lowest price of the stock during the trading day.

**'Close':** The closing price of the stock on that day.

**'Volume':** The trading volume of Microsoft stock on that day.

Additionally, the dataset may contain the 'Adj Close' column, which is dropped in the code due to redundancy with the 'Close' column.

**Data Preprocessing Steps:**

The code performs several data preprocessing steps to prepare the dataset for modeling:

**Loading Data:** The historical stock price data is loaded from the CSV file into a Pandas DataFrame.

**Handling Missing Values:** The code does not explicitly address missing values, but it's important to ensure that the dataset is complete and does not contain missing data.

**Feature Engineering:** The code creates new features and manipulates existing ones as follows:

**Splitting 'Date':** The 'Date' column is split into 'day,' 'month,' and 'year' columns, allowing the model to capture temporal patterns.

**'is\_quarter\_end':** A binary feature is added, indicating whether the month is a quarter-end month (e.g., March, June, September, December).

**'open-close' and 'low-high':** New features are created by subtracting 'Close' from 'Open' and 'High' from 'Low,' respectively.

**'target':** The target feature is added based on whether the next day's closing price is higher (1) or lower (0) than the current day's closing price.

**Data Splitting:** The dataset is split into training and validation sets using scikit-learn's train\_test\_split function. The training set is used to train the models, and the validation set is used to evaluate their performance.

**Feature Scaling:** The 'open-close,' 'low-high,' and 'is\_quarter\_end' features are standardized using scikit-learn's StandardScaler to ensure that all features have similar scales. This is important for many machine learning algorithms.

**Feature Extraction Techniques:**

The code mainly focuses on feature engineering, where new features are created or existing features are transformed to improve the model's ability to capture patterns in the data. Here are the key feature extraction techniques applied in the code:

**Temporal Features:** 'day,' 'month,' and 'year' are extracted from the 'Date' column. These features allow the model to capture temporal patterns that may affect stock prices differently over time.

**Binary Feature:** The 'is\_quarter\_end' feature is binary, indicating whether a trading day falls at the end of a fiscal quarter. This can capture quarterly financial reporting patterns, which may impact stock prices.

**Arithmetic Operations:** 'open-close' and 'low-high' are derived from existing columns. These features may provide insights into price volatility and trading range for each day.

**Target Feature:** The 'target' feature is created as the target variable for classification, which is used to determine whether it's a good decision to buy or not buy the stock.

**Choice of Machine Learning Algorithms:**

The code uses three different machine learning algorithms:

a. **Logistic Regression:** Logistic regression is a commonly used algorithm for binary classification tasks. It is chosen as one of the models due to its simplicity, interpretability, and efficiency. Logistic regression models the probability of the binary target variable, making it suitable for this problem where the goal is to predict whether to buy (1) or not buy (0).

b. **Support Vector Classifier :** Support Vector Classifier (SVC) with a Polynomial Kernel: SVC with a polynomial kernel is a powerful classification algorithm. The choice of a polynomial kernel allows the model to capture non-linear relationships between features and the target variable. SVC is a good choice when there might be complex decision boundaries in the data.

c. **XGBoost Classifier:** XGBoost is a popular ensemble learning algorithm that often performs well in classification tasks. It is known for its ability to handle complex relationships in the data, feature importance ranking, and the ability to handle missing values. XGBoost is a strong choice for improving prediction accuracy.

**Model Training:**

The code trains each of the three selected machine learning models (Logistic Regression, SVC, and XGBoost Classifier) on the training data. Model training is a critical step in which the models learn the underlying patterns in the historical stock price data. The code uses the fit method of each model to train them.

**Evaluation Metrics:**

The code uses the ROC AUC (Receiver Operating Characteristic Area Under the Curve) metric for model evaluation. Here's why ROC AUC is chosen:

**ROC AUC:** ROC AUC is a suitable metric for binary classification tasks. It measures the ability of a model to distinguish between the two classes, in this case, whether to buy (1) or not buy (0). A higher ROC AUC score indicates a better-performing model. The choice of ROC AUC over accuracy is appropriate, especially if there is class imbalance in the dataset, as it provides a more comprehensive evaluation.

**Confusion Matrix:** While not explicitly explained in the code comments, a confusion matrix is also used to visualize the model's performance. The confusion matrix helps understand the true positive, true negative, false positive, and false negative predictions made by the models, providing insights into the model's accuracy and potential misclassifications.

**Innovative technique:**

**Feature Engineering:** While the code performs standard feature engineering tasks such as splitting the date and creating binary and derived features, it's important to emphasize the importance of feature engineering in this context. Financial time series data analysis often benefits from domain-specific feature engineering. Advanced techniques such as calculating technical indicators (e.g., moving averages, relative strength index) or sentiment analysis based on news or social media data could provide more informative features for predicting stock prices.

**Ensemble Models:** The code uses three different machine learning algorithms to train individual models. An innovative approach could involve creating an ensemble of these models to combine their predictions, potentially improving overall predictive accuracy. Techniques like bagging, boosting, or stacking can be explored to harness the strengths of each model.

**Hyperparameter Optimization:** The code doesn't explicitly perform hyperparameter optimization for the selected models. Applying techniques like grid search or random search to fine-tune the model hyperparameters can lead to improved model performance. Automated hyperparameter tuning libraries such as Scikit-learn's GridSearchCV or RandomizedSearchCV could be incorporated.

**Time Series Analysis:** Financial data, particularly stock prices, often exhibit temporal dependencies. Innovative approaches might involve time series analysis techniques such as autoregressive integrated moving average (ARIMA), GARCH, or Prophet for modeling stock price movements. Additionally, deep learning models like recurrent neural networks (RNNs) or long short-term memory networks (LSTMs) can be applied to capture temporal patterns more effectively.

**Feature Selection and Importance:** The code doesn't explicitly explore feature selection or feature importance techniques. Utilizing techniques like recursive feature elimination (RFE), feature importance ranking, or permutation importance could help identify the most relevant features for prediction, potentially simplifying the model and improving interpretability.

**Handling Class Imbalance:** The code includes a pie chart to visualize the balance of the target variable but doesn't explicitly address class imbalance. Innovative techniques for handling class imbalance, such as oversampling, undersampling, or using specialized algorithms like SMOTE, could be considered to improve model performance.

**Model Explainability:** While the code doesn't include model interpretability techniques, it's important to note that financial decision-making often requires model transparency. Using innovative tools like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) can help explain model predictions and build trust in the model's recommendations.

**Conclusion:**

The provided code initiates the development of a predictive model for Microsoft stock buying decisions based on historical data. It conducts essential data preprocessing, feature engineering, and model training, employing three machine learning algorithms. However, the code is a starting point and requires further refinement. Model performance evaluation should be extended beyond ROC AUC to include more metrics. Addressing class imbalance, exploring advanced techniques, and ensuring model explainability are recommended. Additionally, real-world deployment considerations and backtesting should be incorporated to assess practical viability. Continuous model monitoring is essential to adapt to dynamic financial markets. The code's potential can be realized with these enhancements.