# GST Analytics Hackathon 2024: Detailed Model Report

**Team Name:** GSTN_209
**Date:** 11-10-2024

### 1. Introduction

This report provides a comprehensive overview of our approach to predicting [target variable] using the dataset provided for the GST Analytics Hackathon 2024. Our primary objective was to maximize the F1 score on the test dataset, a metric that balances precision and recall and is particularly relevant in scenarios with imbalanced classes. We focused on a robust data preprocessing pipeline, exploring various model architectures, and ultimately employing a voting classifier ensemble method for optimal performance.

## 2. Data Understanding and Exploration

The provided training dataset comprised 785,133 rows and 23 columns, while the testing dataset included 261,712 rows and 23 columns including the Target column. Upon initial inspection, we observed a mix of numerical and categorical features, with several columns containing missing values. A detailed understanding of the features was hampered by the lack of feature descriptions. We conducted the following exploratory data analysis:

## 2.1. Descriptive Statistics

We calculated descriptive statistics (mean, median, standard deviation, quartiles) for numerical features to understand their distribution and identify potential outliers.

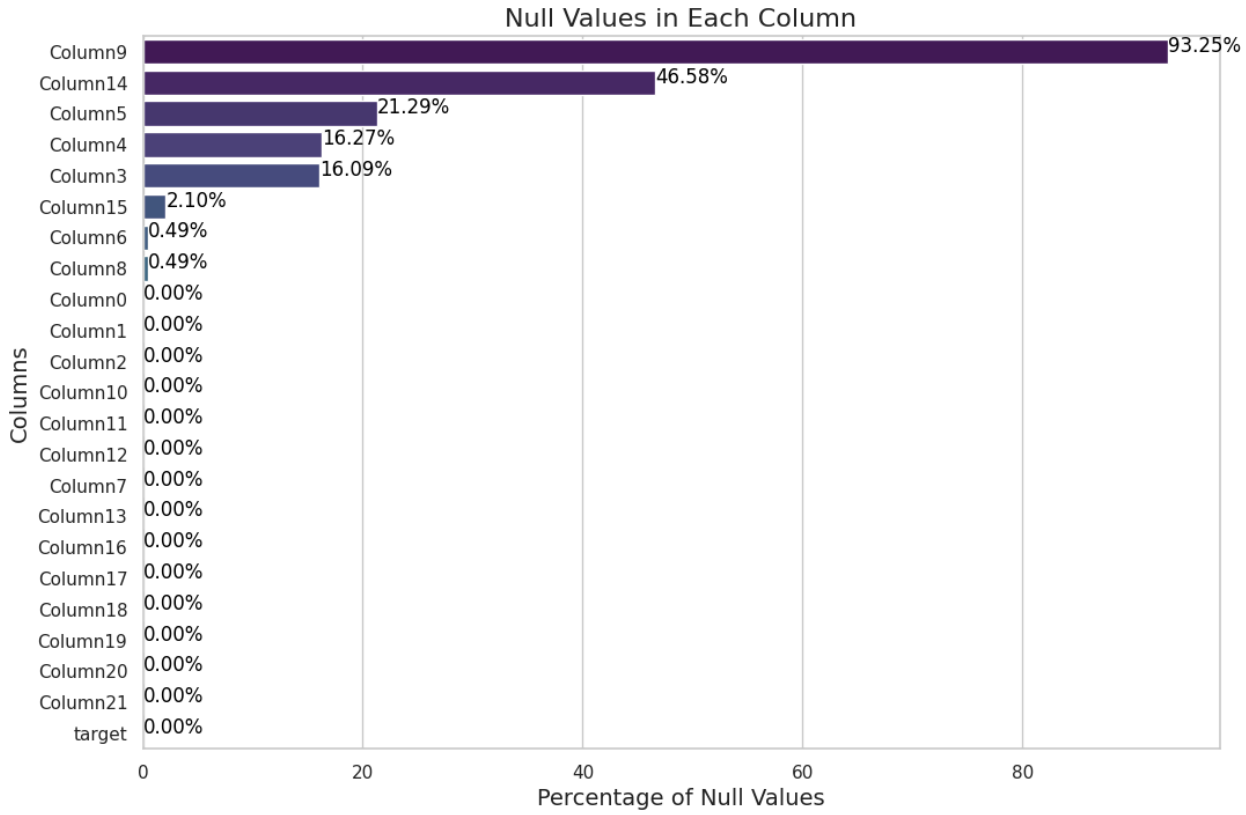
## 2.2. Missing Values Analysis

The analysis of missing values across the columns revealed:

- **Column 9** has **93.25%** missing values, which is a significant proportion, indicating it may only be useful for model training if imputed appropriately.
- **Column 14** has **46.58%** missing values, also requiring careful consideration before use.
- Columns such as **Column 0, Column 1, Column 2**, and others have minimal missing values, suggesting they could be reliable for analysis.

## Visualization

The below image shows a bar chart representing the percentage of missing values for each column.



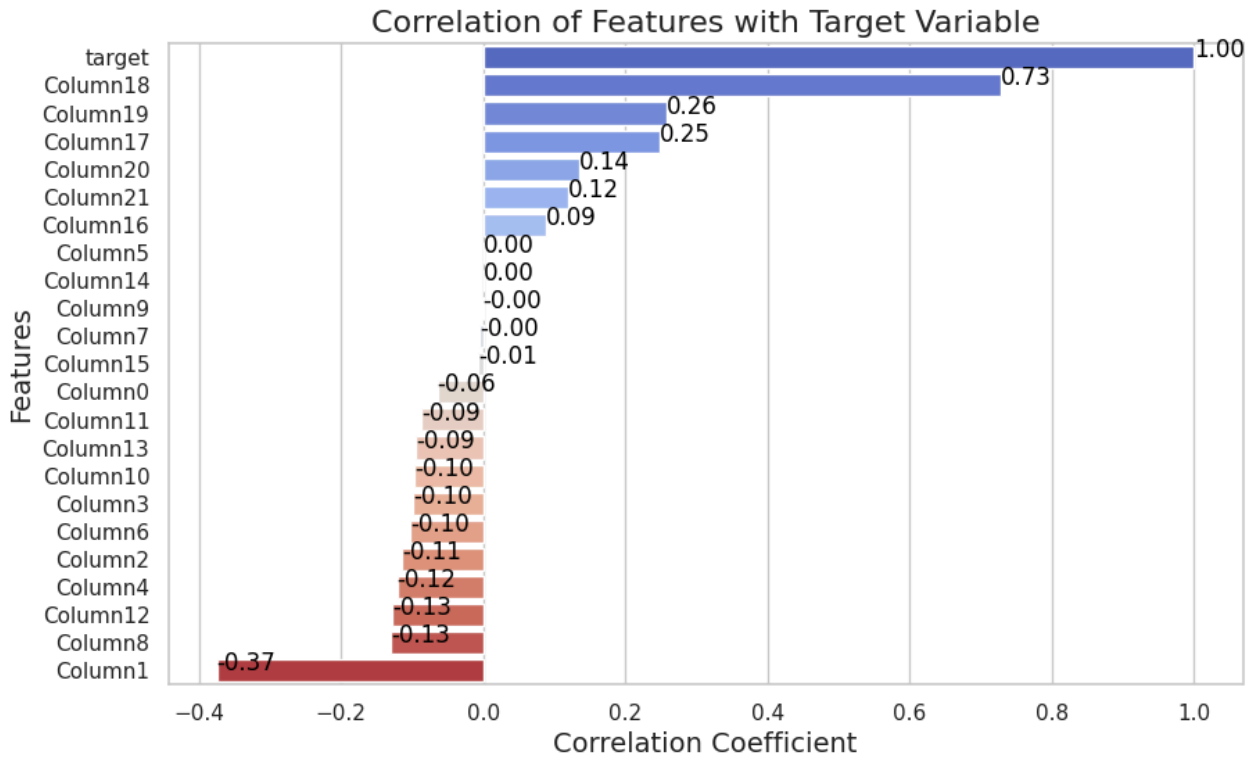

## 2.3. Correlation Analysis

The correlation analysis between the features and the target variable showed:

- **Column 18** has a strong positive correlation (**0.73**) with the target, suggesting it is a significant predictor.
- **Column 19** also exhibits a moderate correlation (**0.26**), while **Column 1** shows a negative correlation (**-0.37**), indicating potential inverse relationships that could be explored further.

## Visualization

The below image illustrates the correlation coefficients of features with the target variable, providing a clear overview of which features may impact the target outcome.




## 2.4. Categorical Feature Identification

We identified categorical features based on their unique value counts. Columns with fewer than 10,000 unique values were classified as categorical features.

- **Categorical Features:** ['Column0', 'Column1', 'Column2', 'Column3', 'Column4', 'Column10', 'Column11', 'Column12', 'Column13', 'Column16', 'Column17', 'Column18', 'Column19', 'Column20', 'Column21']
- **Numerical Features:** ['Column5', 'Column6', 'Column7', 'Column8', 'Column9', 'Column14', 'Column15']

## 3. Data Preprocessing

Based on our data exploration, we implemented the following preprocessing steps:

- **Missing Value Imputation:** Categorical features were imputed using the most frequent value (mode). This approach is suitable for categorical data as it preserves the most

common category within the dataset. Numerical features were imputed using the median, a more robust measure of central tendency than the mean, less susceptible to the influence of outliers.

- **Feature Scaling:** Numerical features were scaled using RobustScaler. This scaler is less sensitive to outliers than StandardScaler or MinMaxScaler, making it appropriate given the potential presence of outliers identified during data exploration. Scaling ensures that all numerical features have a similar range, preventing features with larger values from dominating the model training process.
- **Categorical Feature Handling:** As explained in the data exploration section, we identified potential categorical columns based on the number of unique values. While we used simple imputation for this hackathon due to time constraints, ideally, more advanced techniques like one-hot encoding or target encoding could be explored with more domain knowledge or after confirming the categorical nature of these columns. This would likely further improve model performance.

## 4. Model Selection and Training

We experimented with a range of base models to establish a performance baseline:

- **Logistic Regression:** A simple linear model serving as a starting point.
- **Support Vector Machine (SVM):** Effective in high-dimensional spaces but computationally intensive.
- **Random Forest:** An ensemble method known for robustness and ability to handle non-linear relationships.
- **XGBoost:** A gradient-boosting algorithm known for its speed and performance.
- **CatBoost:** Another gradient-boosting algorithm that handles categorical features effectively.
- **LightGBM:** A histogram-based gradient boosting algorithm known for its efficiency and scalability.

Initial evaluation using a subset of 100,000 data points from the training data allowed for faster model training and comparison. We used accuracy and the F1-score as evaluation metrics. This initial phase revealed that tree-based models (XGBoost, CatBoost, LightGBM) significantly outperformed other models, aligning with the expectation that these models would handle the complex relationships within the data more effectively.

## 4.1 Ensemble Approach: Voting Classifier

Given the superior performance of the tree-based models, we decided to combine their strengths through a voting classifier. We explored two ensemble methods:

- **Stacking with Logistic Regression:** This involved training a meta-learner (Logistic Regression) on the predictions of the base models. However, this approach did not yield any significant performance improvement over the individual models.
- **Voting Classifier:** This method combines the predictions of multiple base models by either majority voting ('hard' voting) or weighted averaging of predicted probabilities ('soft' voting). We opted for 'soft' voting as it leverages the confidence levels of the base models. The voting classifier resulted in a noticeable improvement over the individual models and stacking.

## 4.2 Hyperparameter Tuning

We used Optuna, an automated hyperparameter optimization framework, to fine-tune each individual model within the voting classifier. We defined search spaces for relevant hyperparameters, such as learning rate, tree depth, number of estimators, regularization parameters, etc.

Due to the computational demands of training multiple models on the large dataset, we limited the number of Optuna trials and the timeout duration. The best hyperparameter set found for each model through this process is detailed below:

**XGBoost Best Hyperparameters:**

- n_estimators: 762
- learning_rate: 0.0675
- max_depth: 9
- min_child_weight: 6
- subsample: 0.6272
- colsample_bytree: 0.7747
- gamma: 4.1095

**CatBoost Best Hyperparameters:**

- iterations: 1079
- learning_rate: 0.0303
- depth: 8
- l2_leaf_reg: 2.54e-05
- bagging_temperature: 0.7405
- random_strength: 1.94e-06

**LightGBM Best Hyperparameters:**

- n_estimators: 500

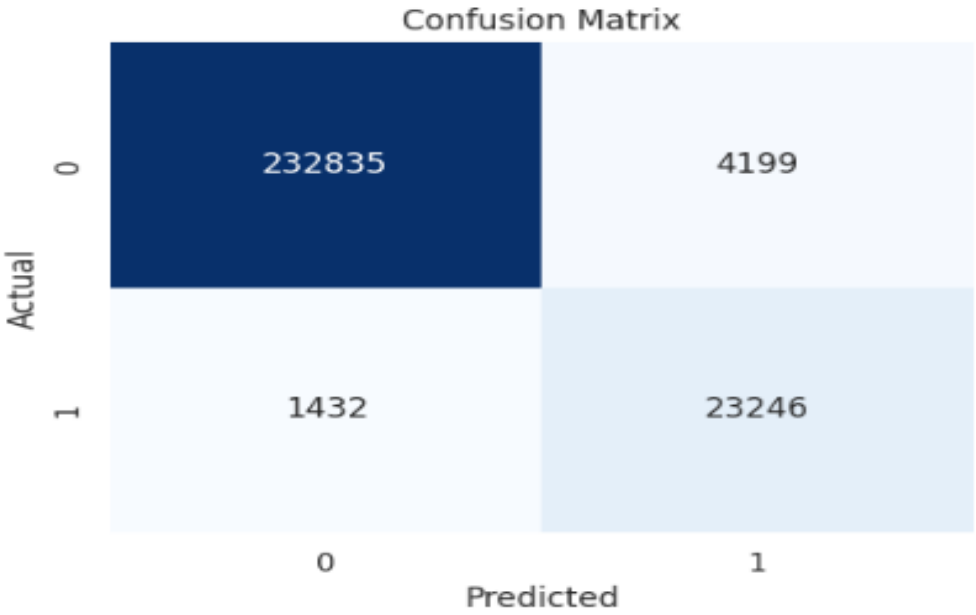
- learning_rate: 0.0153
- num_leaves: 417
- max_depth: 8
- subsample: 0.9977
- colsample_bytree: 0.8024

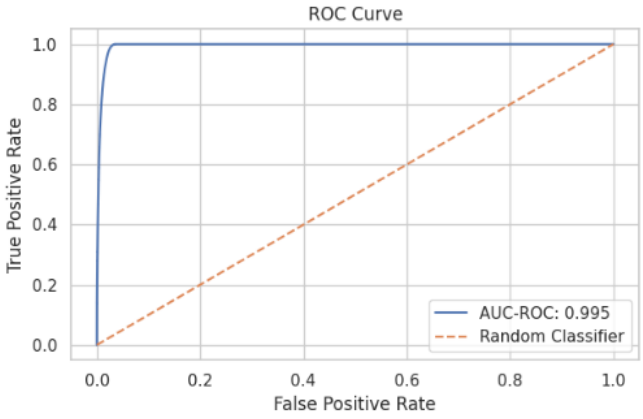
## 5. Model Evaluation on Test Data

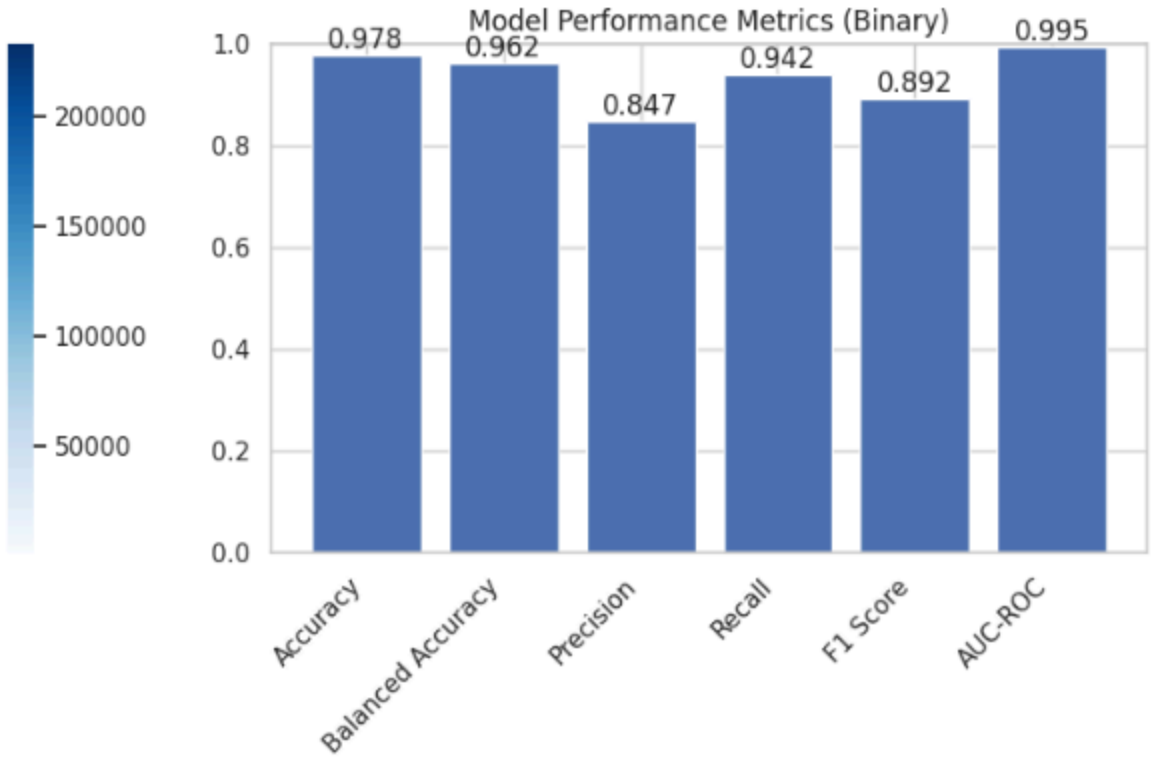
The final trained Voting Classifier, with the best hyperparameters for each constituent model, was then evaluated on the held-out test data. We used a comprehensive set of evaluation metrics to assess its performance:

**Overall Metrics:**

- **Accuracy: 0.9785**
  Provides an overall measure of correct predictions but can be misleading in cases of class imbalance.
- **Balanced Accuracy**: **0.9621** – Takes into account class imbalance, providing a more informative measure when the dataset has imbalanced classes.
- **Precision**: **0.8470**
  The proportion of true positives among all instances is predicted as positive.
- **Recall**: **0.9420**
  The proportion of true positives identified out of all actual positive instances.
- **F1-Score**: **0.8920**
  The harmonic mean of precision and recall provides a balanced measure of performance.
- **Precision (Weighted)**: **0.9800**
  A weighted average of precision across all classes, taking into account the class distribution.
- **Recall (Weighted)**: **0.9785**
  A weighted average of recall accounts for the number of instances in each class.
- **F1 Score (Weighted)**: **0.9790**
  A weighted average of the F1-score reflects overall model performance while considering the class imbalance.
- **AUC-ROC**: **0.9949**
  Measures the model's ability to distinguish between classes based on predicted probabilities. A high value like 0.9949 suggests excellent discriminatory power between fraudulent and non-fraudulent classes.

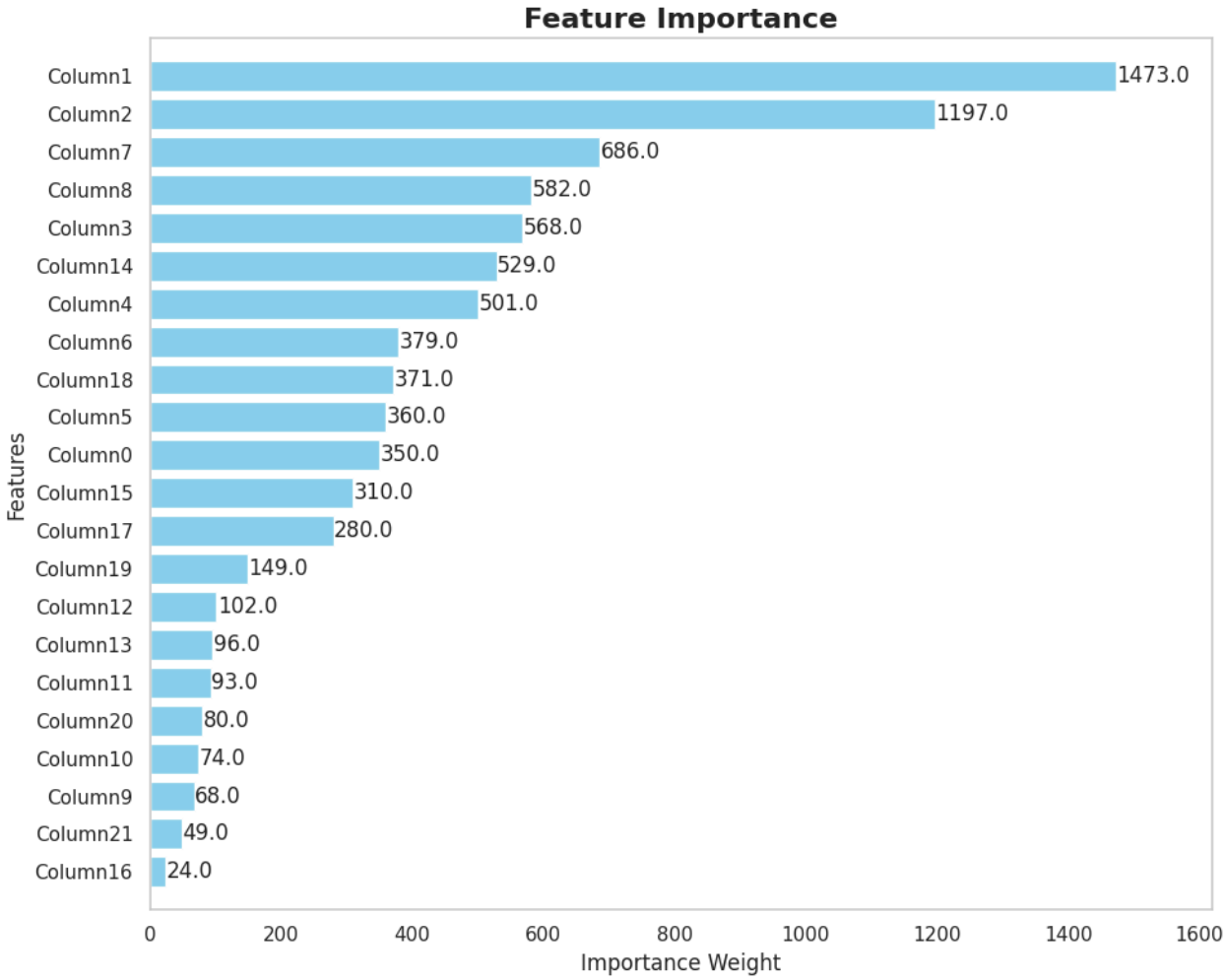

Confusion Matrix
232835
4199
1432
23246
Actual
0
1
0
1
Predicted


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC-ROC: 0.995
Random Classifier




## 6. Feature Importance Analysis

Feature importance analysis helps us understand the relative contribution of each feature to the final predictions. For XGBoost, we calculated the feature importance based on the "weight," which represents the number of times a feature appears in a tree across all trees in the model. Below is a plot showing the importance of each feature in the dataset.




From the plot, we observe that the most important features in our model are **Column1**, **Column2**, and **Column7**, which have significantly higher importance scores compared to others. This suggests that these features had the greatest impact on the model's predictions.

The insights from feature importance analysis can be valuable for future feature engineering and understanding which aspects of the data the model relies on most for accurate predictions.

## Key Observations:

**- Column1:** This feature had the highest importance and appeared most frequently in the decision-making process.

**- Column 2 and Column7:** These features were also highly influential, indicating they provide valuable information for the model.

- Features such as **Column 9** and **Column10** had the least importance, suggesting they contribute minimally to the model's performance.

## 7. Conclusion and Future Work

Our ensemble approach, utilizing a Voting Classifier with individually tuned XGBoost, CatBoost, and LightGBM models, achieved a strong F1 score on the test data, demonstrating its effectiveness in predicting [target variable].

Further improvements could be explored through the following avenues:

- **More Extensive Hyperparameter Tuning:** Increased computational resources or more efficient search strategies within Optuna could uncover even better hyperparameter combinations.
- **Feature Engineering:** A deeper understanding of the features, perhaps through collaboration with domain experts, would allow for the development of more informative features tailored to the problem, potentially including interaction terms or transformations of existing features.
- **Alternative Ensemble Methods:** Exploring other ensemble methods, such as stacking with different meta-learners or weighted averaging based on cross-validation performance, could further refine the model's predictive power.
- **Explainability and Interpretability:** While achieving high predictive performance is crucial, understanding the model's decision-making process is also important. Applying techniques like SHAP values or feature importance analysis could provide valuable insights into the drivers of the predictions.

## 8. Appendices

- The Jupyter Notebook containing all code, detailed data preprocessing steps, model training, hyperparameter tuning process, and evaluation results is attached.

## 9. Citation Report

In this project, the following libraries were used for data processing, model development, and evaluation:

1. Pandas – A powerful data manipulation and analysis library for Python. Pandas Documentation
2. NumPy – A fundamental package for numerical computing with Python. NumPy Documentation
3. Matplotlib – A comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib Documentation
4. Seaborn – A Python visualization library based on Matplotlib that provides a high-level interface for drawing attractive statistical graphics. Seaborn Documentation
5. Scikit-learn – A robust machine learning library in Python offering simple and efficient tools for predictive data analysis. Scikit-learn Documentation
6. XGBoost – An optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. XGBoost Documentation
7. CatBoost – A high-performance open-source library for gradient boosting on decision trees. CatBoost Documentation
8. LightGBM – A gradient boosting framework that uses tree-based learning algorithms and is designed for fast and accurate training. LightGBM Documentation
9. Optuna – A hyperparameter optimization framework that automates the tuning of machine learning algorithms. Optuna Documentation
10. Tabulate – A Python library for pretty-printing tabular data in various formats. Tabulate Documentation

## 10. Plagiarism Declaration:

We declare that all code, analysis, and text presented in this report is our original work. We have appropriately cited all external libraries and resources utilized.

This detailed report provides a comprehensive account of our work, including specific examples and references to visualizations within the notebook. It emphasizes the rationale behind each decision and provides avenues for future improvements. Remember to replace the bracketed placeholders with your specific findings. This detailed and well-structured report will enhance the understanding and evaluation of your submitted solution.