GSTN Analytics Hackathon Project Report

Team ID: GSTN_651

Team Members:

Jeomon George

PUDUPPULY RAJESH SURENDRAN

OMKAR SHENDE

AATHI MADHAV G

RAMSHAD ABDUL RAHEEM

GitHub Repository URL:

<github repository url>

1. Introduction

The aim of this project is to develop a machine learning model capable of solving the binary classification problem presented in the competition. The challenge involves accurately classifying instances into one of two categories based on the provided dataset. Given the real-world importance of such classification tasks, an effective solution can have significant impact.

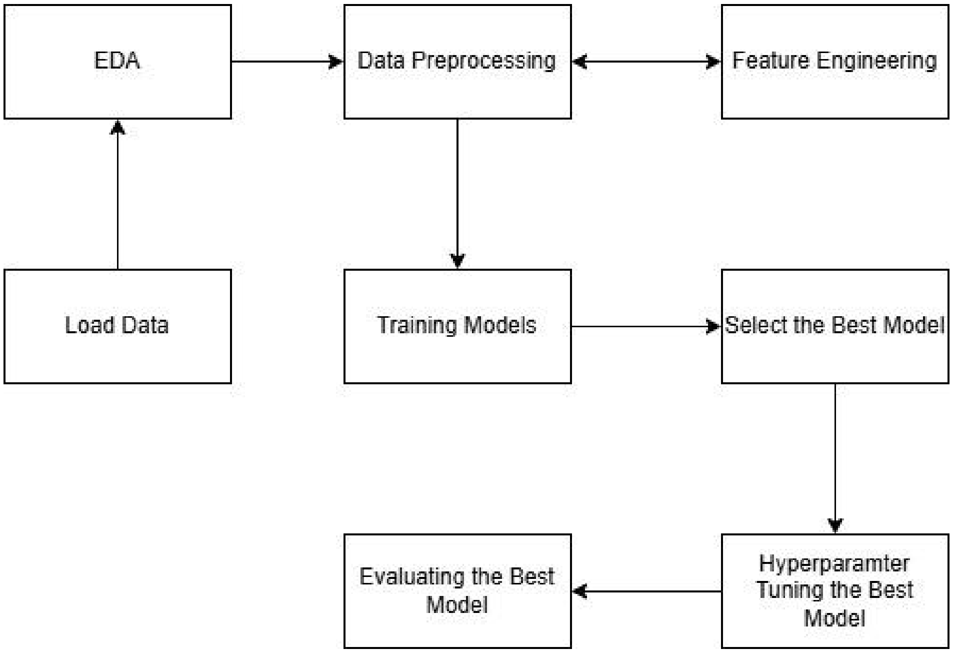
The focus of this project is to explore algorithms, fine-tune them, and apply the most suitable model to the problem. In addition, the evaluation of the model's performance will be based on metrics such as accuracy, precision, recall, F1 score, and AUC-ROC, ensuring a well-rounded analysis of its effectiveness.

In this report, we will describe the steps taken during data preprocessing, feature engineering, model selection, and performance evaluation, alongside the challenges faced and the insights gained throughout the process.

2. Problem Statement

Given a dataset D, which consists of: D_{train} A matrix of dimension $R(m \times n)$ representing the training data. D_{test} A matrix of dimension $R(m_1 \times n)$ representing the test data. We have also provided corresponding target variable Y_{train} matrix dimension of $R(m \times 1)$ and Y_{test} with matrix dimension of $R(m_1 \times 1)$. The objective is to construct a predictive model $F_\theta(X) \rightarrow Y_{pred}$ that accurately estimates the target variable $Y_{\{i\}}$ for new, unseen inputs $X_{\{i\}}$.

3. Project Methodology

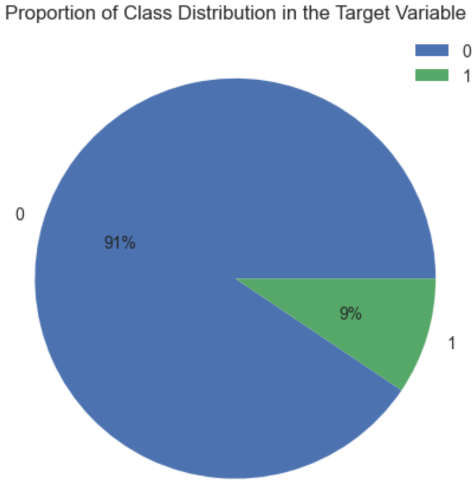


3.5 Load Data

We load the training and testing datasets in pandas it's a python library for data manipulation. The given dataset is in csv format. Training set has 60% of data and testing set has 20% of data. Here we treated ID as index since we found # of unique id's and # of instances are the same.

3.6 EDA (Exploratory Data Analysis)

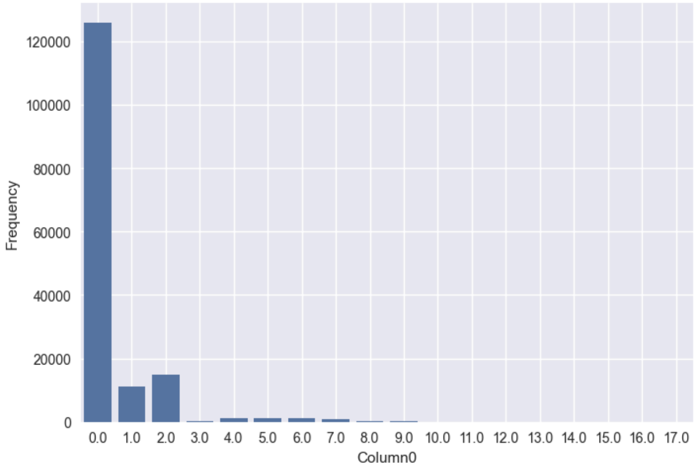
First we checked whether the dataset is class balanced or class imbalanced and found that the dataset is class imbalanced. Here 0's and 1's are the labels of the classification problem.



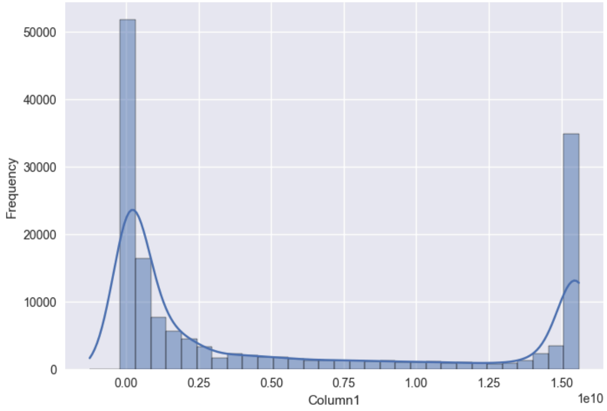
Since the training set consist of 7,85,133 instances, it will be bit hard to EDA. So we took 20% data of the training set for EDA and called this set as **Exploratory Dataset** while preserving the class distribution since dataset is class imbalanced.

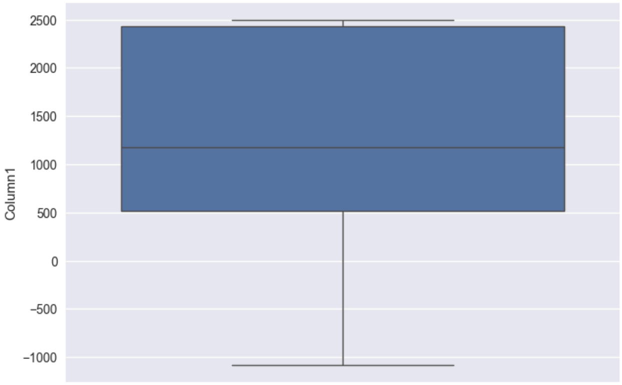
3.7 Univariate Analysis

From the first look the dataset consist of only numerical features. From there we found column10-13 and column19-21 are binary features and column0 has 20 unique labels, column16 has 3 unique labels and much more. Highest number of unique values found in column5-8.

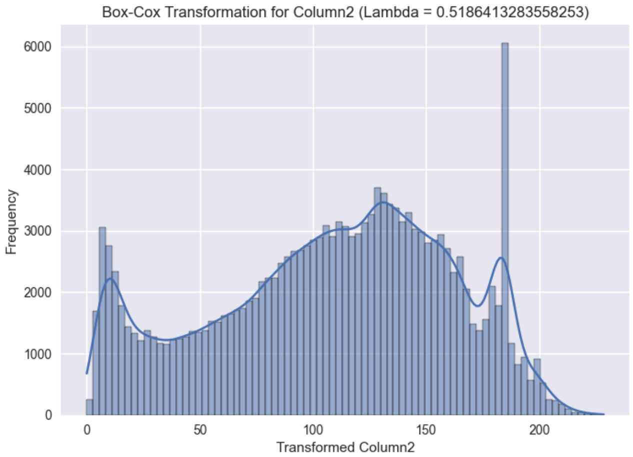


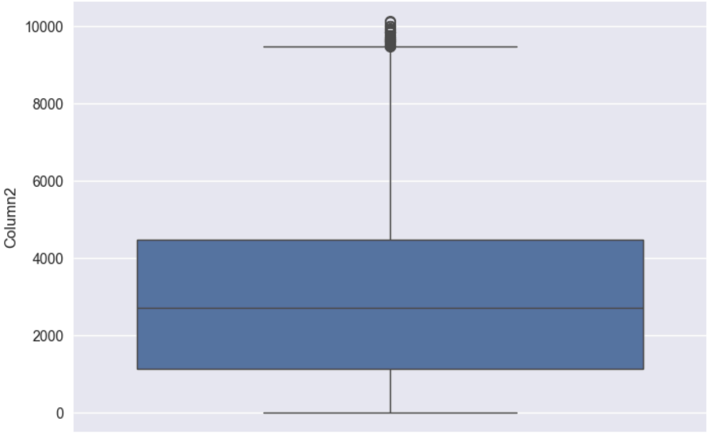
In column0 we have the value 0 of this column taking the lead and all other values aren't coming close to that and values are between 0-17 and they are integer, we are assuming that column0 as categorical and ordinal.



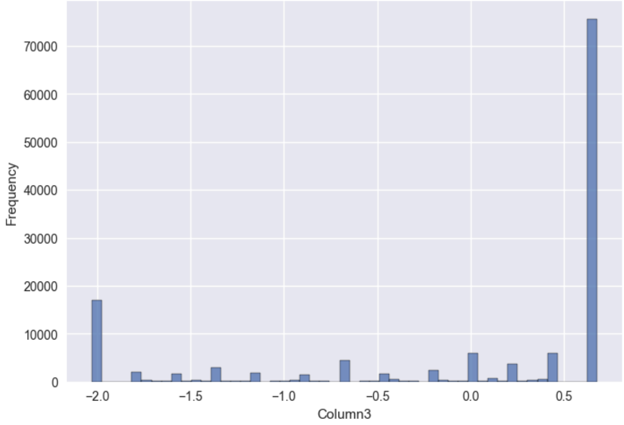


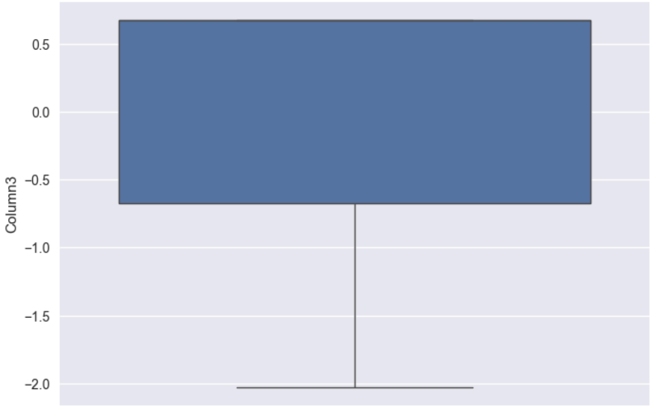
Here in column1 we assume that it is a continuous feature because of too much unique values and no outliers found in the box plot also it's showing some kind of skewed distribution when power transformation of 3 applied.



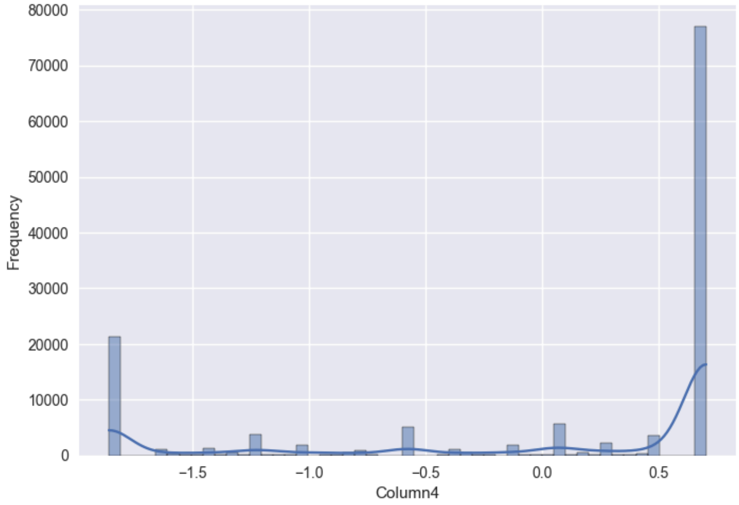


For column2 also just like colum1 it is also assuming continous feature but this has some outliers and when box-cox transformation applied it becomes much like normally dist.





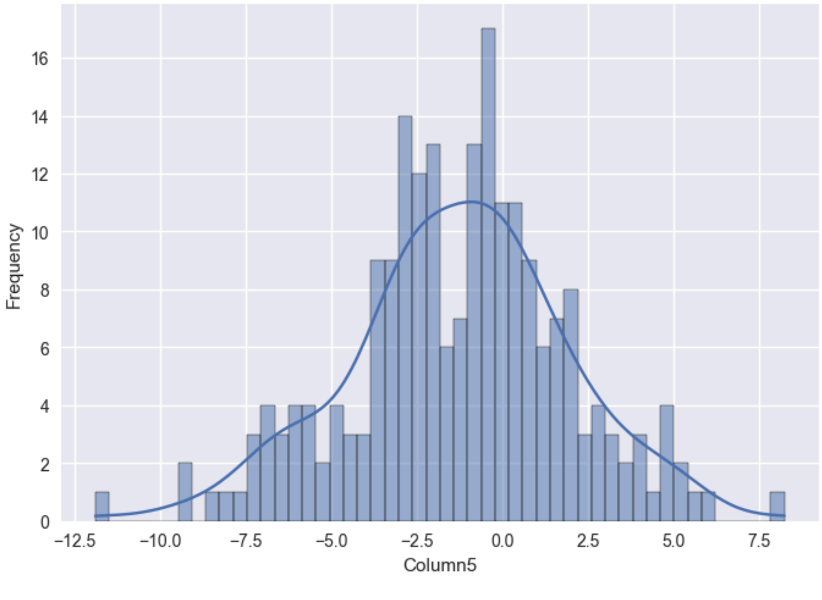
In column3 the values are having decimals and up on checking only 48 unique values found and no outliers, for now assume as continuous numerical feature.



Column4 has some asymmetry at both tails.

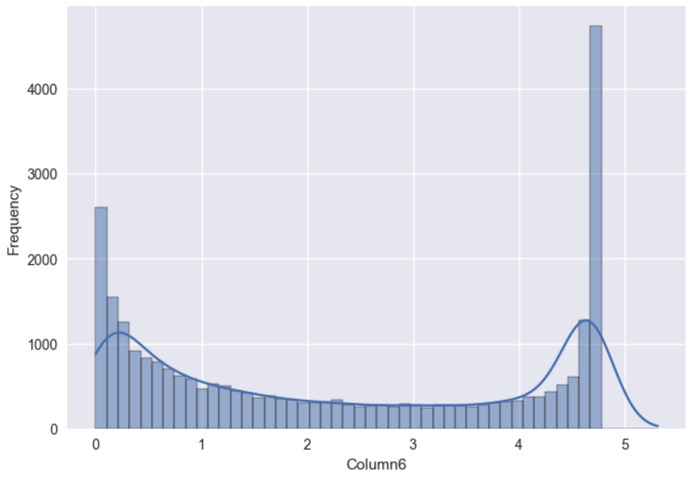
Column4 and column3 are some what similar when histogram is checked also no outliers. So we need to check for any correlation column3 and column4.

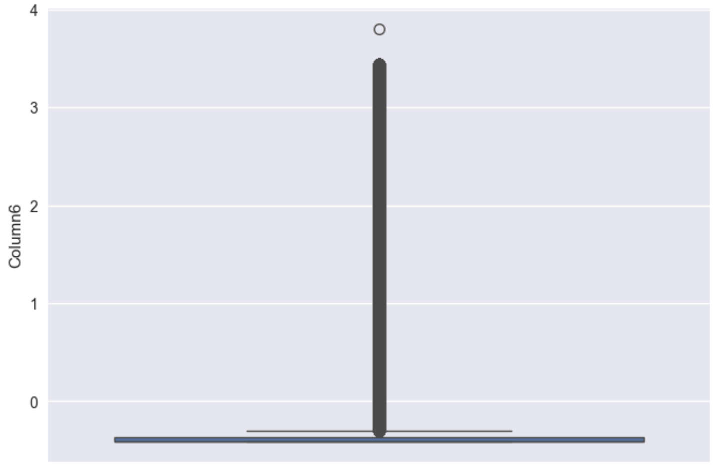
So upon conducting pearson's correlation we found a correlation of 0.881 which indicates either column3 or column4 can be dropped.

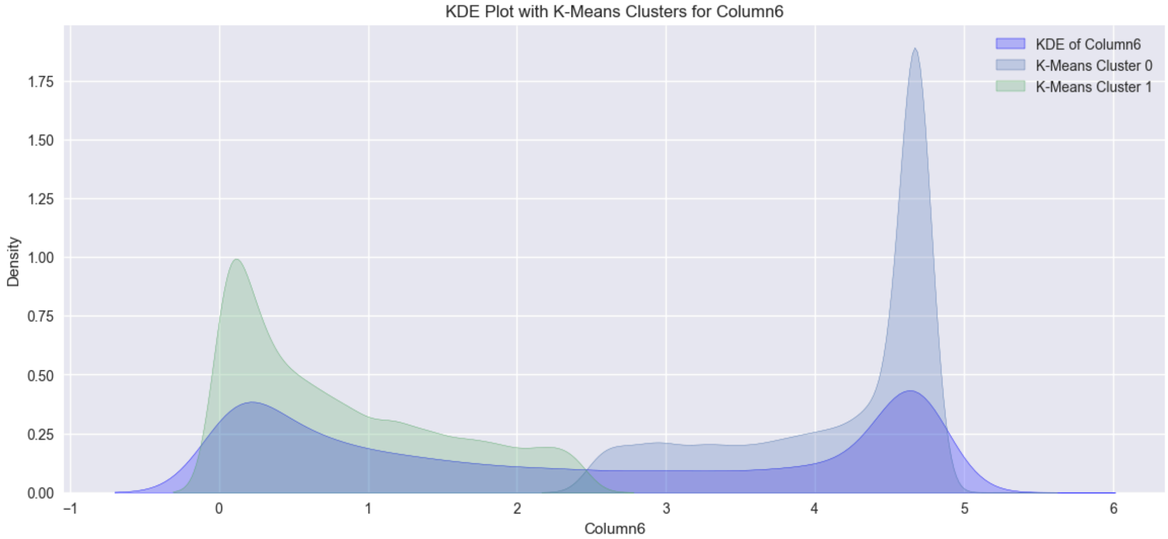


In Column5, after applying $\log_{10}$ transformation we got almost a normal distribution.

Column6 majority of the values are outliers and when plotted the KDE we found two humps towards the ends of the distribution, this suggest that there might be 2 clusters within this feature.







After doing K-means clustering we found 2 clusters in this column and these two clusters overlap on a small area which is clear from the graph.

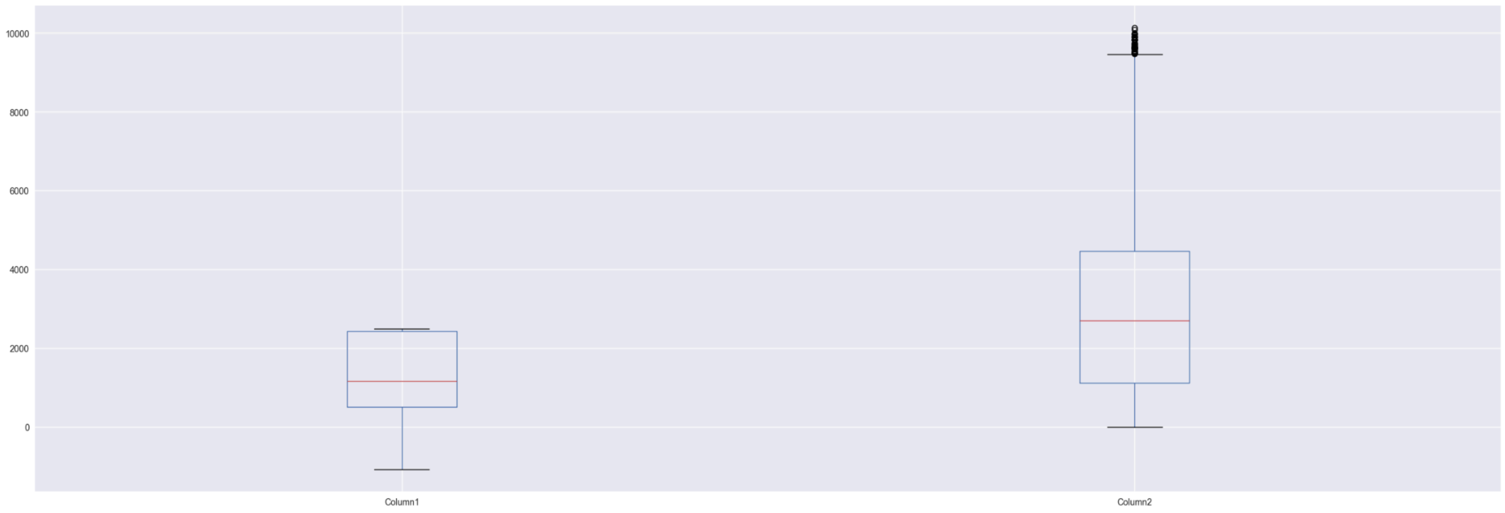
In column5 and column6, these two columns poses some sort of non-linear relationship as pearson's correlation was about 0.0015 and in spearman shows 0.294.

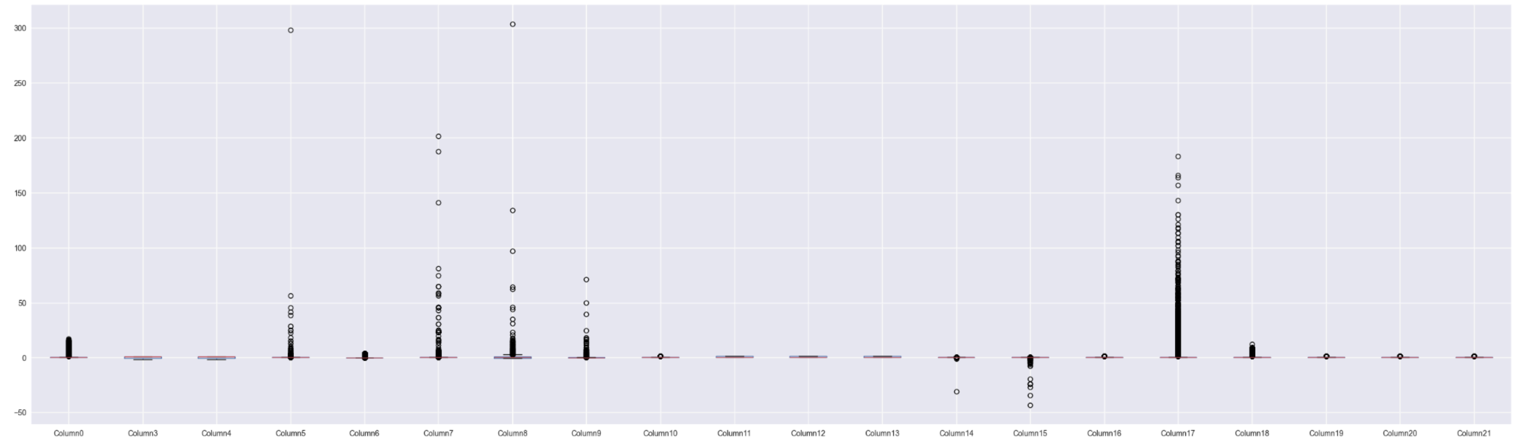
Column7-9 we are considering as continuous feature because the number of unique instances are quite high (37-43K range)

Column10-13, Column16 and Column19-21 are binary features consisting of 0’s and 1’s.

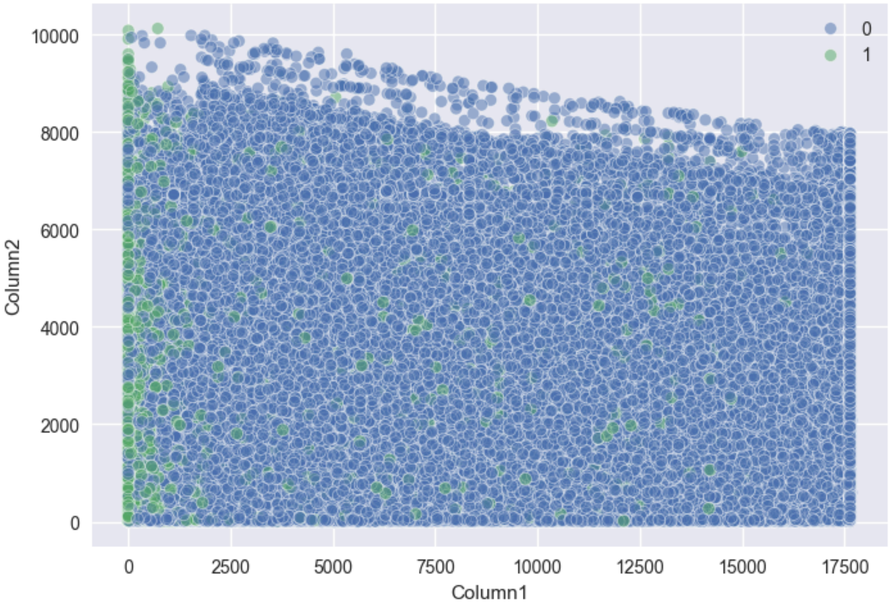
3.8 <u>Outliers Analysis</u>

Here the majority of the features have outliers but Column1, Column3-4, Column11-13 have zero outliers. Column17 has more dense outliers. When we tried to remove outliers from Column17 only one particular value is retaining so we are keeing the outliers. Similarly when we removed outliers from all columns the training set have only label 0’s and no 1’s due to this we decided to keep the outliers in the dataset during model building.

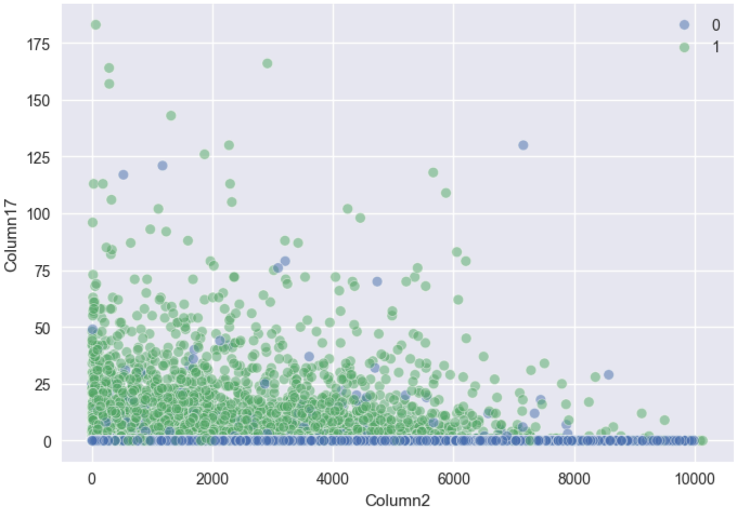




3.9 **Multivariate Analysis**



There is a certain extend of linear correlation between Column1 and Column2 as the pearson's correlation is 0.229. This is got by applying power transformation 1.25 to the Column1.



This the scatter plot between Column2 and Column17. It is observed that the datapoints coming under label 1 is more spread than compared to the label 0. The datapoints coming under label 0 is not possessing any relationship between these columns. While label 1 is more concentrated on towards the origin and decreases as we go away from origin.

4. Preprocessing

4.1 Missing values

The following shows the details about the NaN values in the training set.

Column0: 0.001% missing values
Column3: 16.087% missing values
Column4: 16.266% missing values
Column5: 21.293% missing values
Column6: 0.49% missing values
Column8: 0.49% missing values
Column9: 93.25% missing values
Column14: 46.578% missing values
Column15: 2.096% missing values

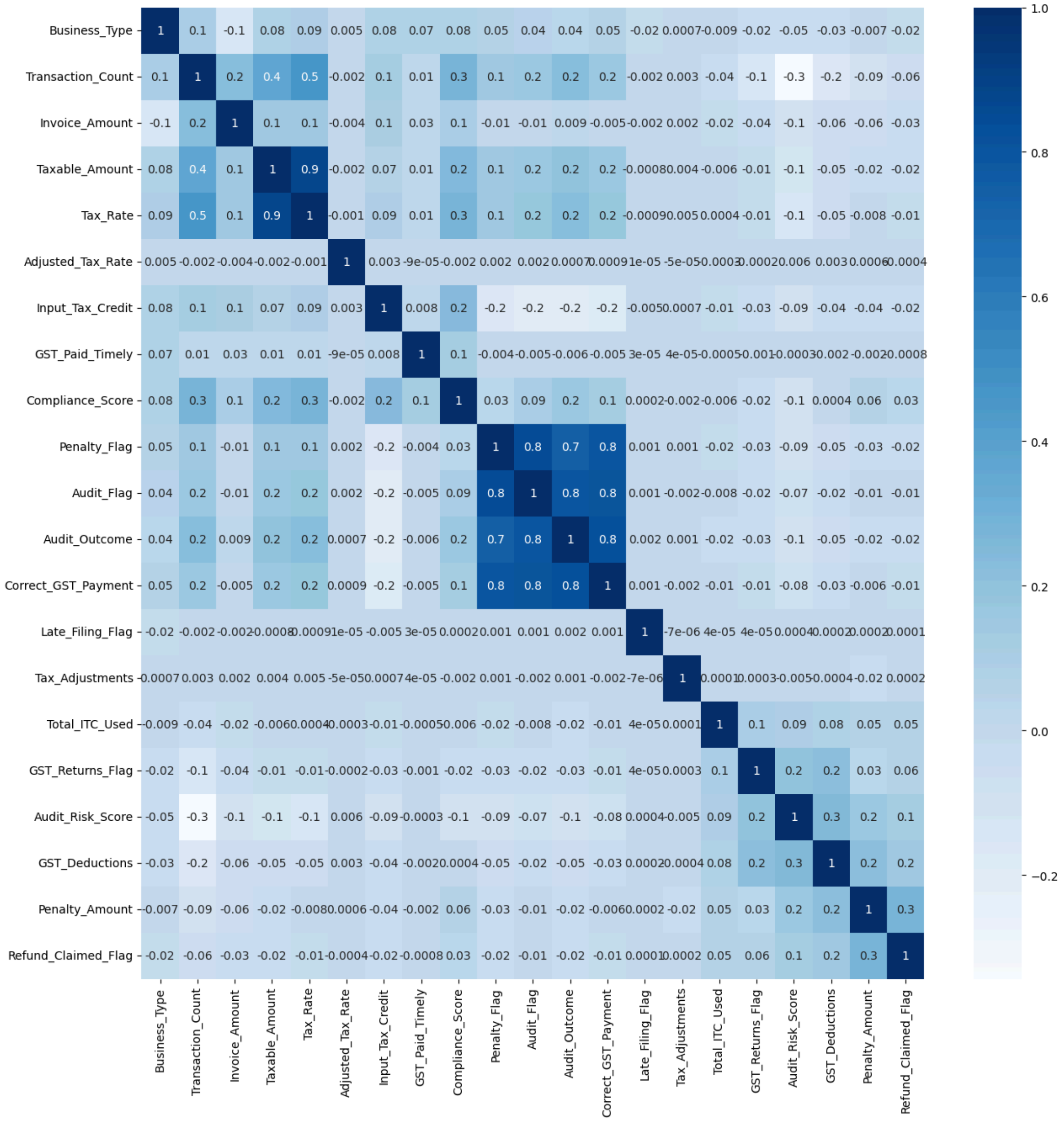
All other columns have zero missing values. From here it is clear that the column with maximum missing values is Column9, almost 93.21% of data is missing hence it better to delete that column. On rest of the columns having missing values we will do imputation techniques to fill the missing values. Also, column16, column2, column0 and column18 are treated with wrong datatype initially it was float but they were just numbers without decimal so converted them to integer type.

Imputation Techniques

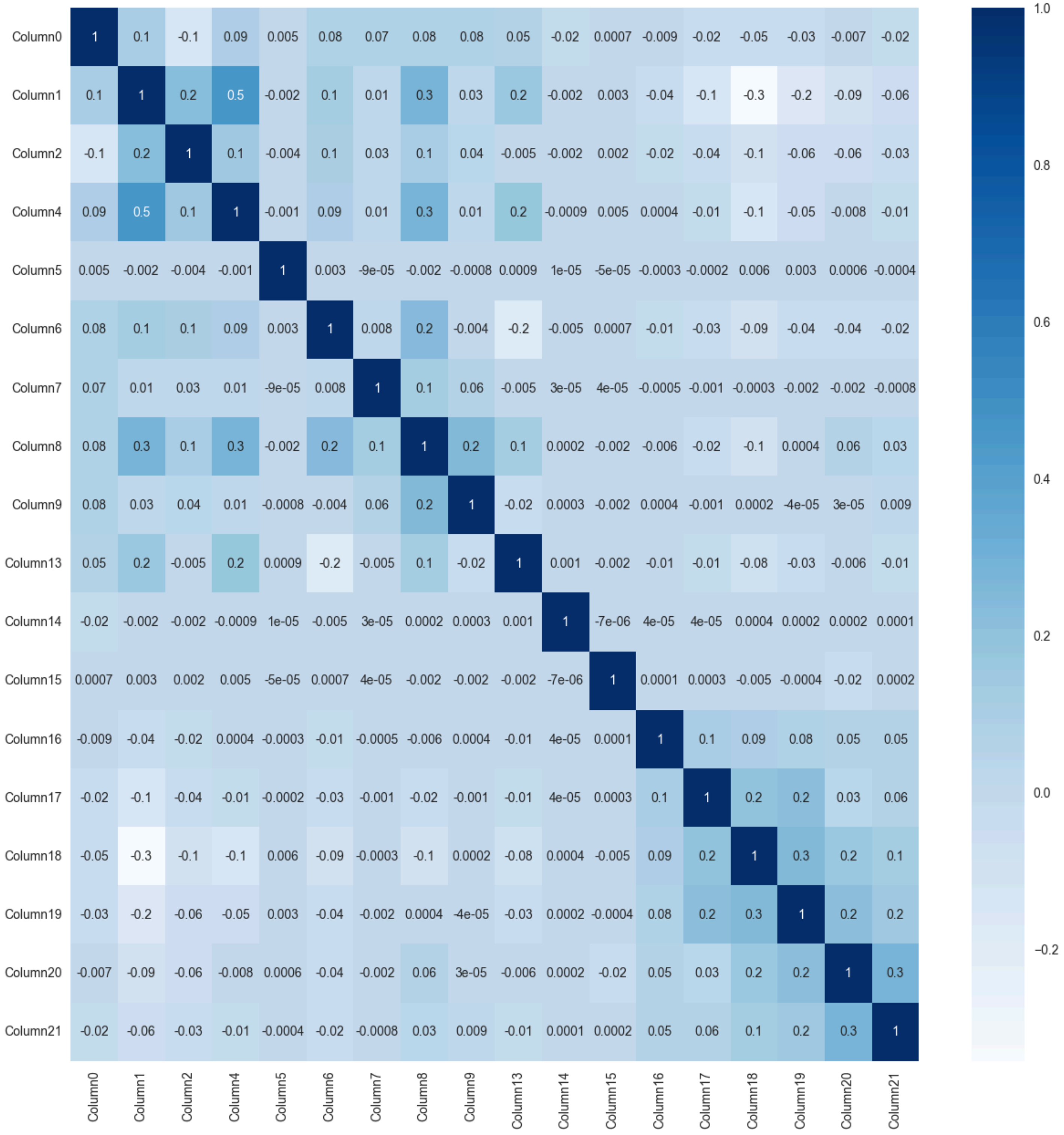
Column0 : Impute with most frequent values because they are categorical.
Column4, Column5, Column6, Column8, Column9, Column14, Column15 : Impute with mean because these columns have continuous values.

5. Feature Selection

5.1 Correlation Matrix



From the above correlation matrix we decided to remove all the columns that are having correlation beyond 0.8. So we remove Column3, Column10, Column11, Column12. This is done so as to remove multi colinearity as it will degrade the performance of the model.



	Column0	Column1	Column2	Column4	Column5	Column6	Column7	Column8	Column9	Column13	Column14	Column15	Column16	Column17	Column18	Column19	Column20	Column21
Column0	1	0.1	-0.1	0.09	0.005	0.08	0.07	0.08	0.08	0.05	-0.02	0.0007	-0.009	-0.02	-0.05	-0.03	-0.007	-0.02
Column1	0.1	1	0.2	0.5	-0.002	0.1	0.01	0.3	0.03	0.2	-0.002	0.003	-0.04	-0.1	-0.3	-0.2	-0.09	-0.06
Column2	-0.1	0.2	1	0.1	-0.004	0.1	0.03	0.1	0.04	-0.005	-0.002	0.002	-0.02	-0.04	-0.1	-0.06	-0.06	-0.03
Column4	0.09	0.5	0.1	1	-0.001	0.09	0.01	0.3	0.01	0.2	-0.0009	0.005	0.0004	-0.01	-0.1	-0.05	-0.008	-0.01
Column5	0.005	-0.002	-0.004	-0.001	1	0.003	-9e-05	-0.002	-0.0008	0.0009	1e-05	-5e-05	-0.0003	-0.0002	0.006	0.003	0.0006	-0.0004
Column6	0.08	0.1	0.1	0.09	0.003	1	0.008	0.2	-0.004	-0.2	-0.005	0.0007	-0.01	-0.03	-0.09	-0.04	-0.04	-0.02
Column7	0.07	0.01	0.03	0.01	-9e-05	0.008	1	0.1	0.06	-0.005	3e-05	4e-05	-0.0005	-0.001	-0.0003	-0.002	-0.002	-0.0008
Column8	0.08	0.3	0.1	0.3	-0.002	0.2	0.1	1	0.2	0.1	0.0002	-0.002	-0.006	-0.02	-0.1	0.0004	0.06	0.03
Column9	0.08	0.03	0.04	0.01	-0.0008	-0.004	0.06	0.2	1	-0.02	0.0003	-0.002	0.0004	-0.001	0.0002	-4e-05	3e-05	0.009
Column13	0.05	0.2	-0.005	0.2	0.0009	-0.2	-0.005	0.1	-0.02	1	0.001	-0.002	-0.01	-0.01	-0.08	-0.03	-0.006	-0.01
Column14	-0.02	-0.002	-0.002	-0.0009	1e-05	-0.005	3e-05	0.0002	0.0003	0.001	1	-7e-06	4e-05	4e-05	0.0004	0.0002	0.0002	0.0001
Column15	0.0007	0.003	0.002	0.005	-5e-05	0.0007	4e-05	-0.002	-0.002	-0.002	-7e-06	1	0.0001	0.0003	-0.005	-0.0004	-0.02	0.0002
Column16	-0.009	-0.04	-0.02	0.0004	-0.0003	-0.01	-0.0005	-0.006	0.0004	-0.01	4e-05	0.0001	1	0.1	0.09	0.08	0.05	0.05
Column17	-0.02	-0.1	-0.04	-0.01	-0.0002	-0.03	-0.001	-0.02	-0.001	-0.01	4e-05	0.0003	0.1	1	0.2	0.2	0.03	0.06
Column18	-0.05	-0.3	-0.1	-0.1	0.006	-0.09	-0.0003	-0.1	0.0002	-0.08	0.0004	-0.005	0.09	0.2	1	0.3	0.2	0.1
Column19	-0.03	-0.2	-0.06	-0.05	0.003	-0.04	-0.002	0.0004	-4e-05	-0.03	0.0002	-0.0004	0.08	0.2	0.3	1	0.2	0.2
Column20	-0.007	-0.09	-0.06	-0.008	0.0006	-0.04	-0.002	0.06	3e-05	-0.006	0.0002	-0.02	0.05	0.03	0.2	0.2	1	0.3
Column21	-0.02	-0.06	-0.03	-0.01	-0.0004	-0.02	-0.0008	0.03	0.009	-0.01	0.0001	0.0002	0.05	0.06	0.1	0.2	0.3	1

After removing those columns that poses multicolinearity from training set the correlation matrix have only those columns whose correlation with other columns less than 0.8. Thus this is the feature selection that we have done for this dataset. This is done because multicolinearity degrade the performance of the model and increase computation cost by performing reductant computations. So removing such columns is the best solution to avoid mutlicolinearity.

5.2 Feature Scaling

Here for column1, column4, column5, column6, column7, column8, column14, column15 we applied Standard Scalar because those are continous features and ordinal encoding to column0 because it is a categorical feature. The remaining features are binary in nature so we kept those features as it were without applying any feature scaling. We used sklearn library to do the feature scaling.

We didn't applied the $\log_{10}$ transformation on column5, similarly box-cox transformation in column2 because they didn't significantly improve the performance of the selected model.

6. Model Training and Selection

Here we used both parametric and non parametric models for training and for the time being we set the parameters in default settings with random state=42 and subjected to 5 fold cross validation to see the performance in training and validation set. For this purpose we are using sklearn and we will focus more on the mean and standard deviation of each metric for each model.

Since this is a binary classification problem we will use the Accuracy, Recall, Precision, F1 to evaluate each model, so as to find the best model.

6.1 Model Performance (Cross Validation)

Cross validation settings:

- 5 Fold cross validation
- Scoring metric: accuracy, precision, recall, f1
- Error score: 0

Models used for cross validation:

- Logistic Regression
- SGD Classifier
- Ridge Classifier
- Decision Tree Classifier
- Random Forest Classifier
- Adaboost Classifier
- Gradient Boosting Classifier
- MLP Classifier

NOTE: *M: mean, S: standard deviation*

Model Name	Dataset	Accuracy (M,S)	Recall (M,S)	Precision (M,S)	F1 (M,S)
Logistic Regression	Train	0.968, 0.000	0.882, 0.05	0.800, 0.003	0.839, 0.001
	Validation	0.968, 0.000	0.882, 0.07	0.801, 0.003	0.839, 0.002
SGD Classifier	Train	0.954, 0.014	0.665, 0.209	0.820, 0.013	0.712, 0.145
	Validation	0.954, 0.014	0.666, 0.208	0.820, 0.013	0.713, 0.144
Ridge Classifier	Train	0.966, 0.001	0.793, 0.016	0.833, 0.003	0.812, 0.007
	Validation	0.965, 0.001	0.793, 0.020	0.833, 0.003	0.812, 0.009
Decision Tree Classifier	Train	1.000, 0.000	0.997, 0.000	0.999, 0.000	0.998, 0.000
	Validation	0.968, 0.000	0.824, 0.003	0.831, 0.003	0.828, 0.002
Random Forest Classifier	Train	0.998, 0.000	0.997, 0.000	0.979, 0.001	0.988, 0.000
	Validation	0.976, 0.000	0.913, 0.002	0.843, 0.002	0.877, 0.001
Adaboost Classifier	Train	0.975, 0.000	0.930, 0.002	0.823, 0.001	0.873, 0.001
	Validation	0.974, 0.000	0.930, 0.003	0.822, 0.002	0.873, 0.002
Gradient Boosting Classifier	Train	0.976, 0.000	0.947, 0.003	0.821, 0.002	0.880, 0.000
	Validation	0.975, 0.000	0.947, 0.004	0.820, 0.002	0.879, 0.001
MLP Classifier	Train	0.972, 0.000	0.924, 0.017	0.806, 0.007	0.861, 0.003
	Validation	0.971, 0.001	0.922, 0.018	0.805, 0.006	0.859, 0.004

Top Contenders:

- Random Forest Classifier: Offers the best balance across accuracy, recall, precision, and F1 score with low standard deviation. It generalizes well without signs of overfitting.
- Gradient Boosting Classifier: Slightly better recall but lower precision. It’s consistent and competitive with Random Forest.
- AdaBoost Classifier: Slightly behind Random Forest but still solid and consistent in recall and precision.

Runners-Up:

- Logistic Regression: A strong baseline but may not match the top contenders in recall and precision for complex tasks.

- MLP Classifier: Good overall, but others like Random Forest and Gradient Boosting slightly outperform it.

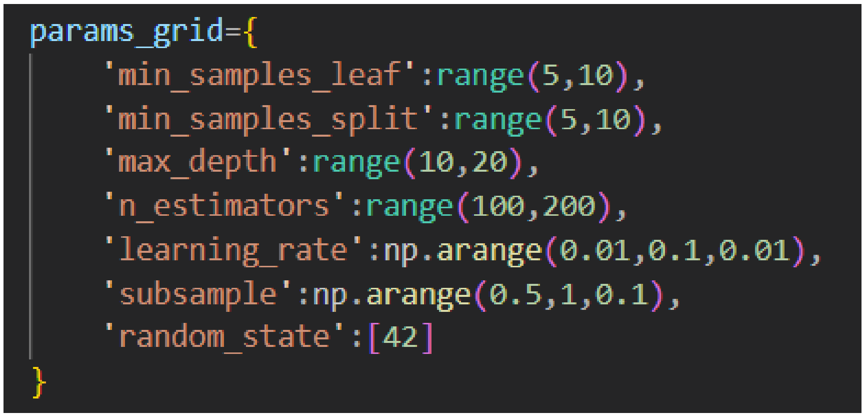
Underperformers:

- SGD Classifier: It has poor recall, which disqualifies it from being suitable for handling imbalanced classes.
- Ridge Classifier: It is decent overall but lacks the precision and recall balance that the top ensemble methods offer.
- Decision Tree Classifier: It suffers from overfitting, making it unreliable for generalization despite high training performance.

7. Hyperparameter Tuning

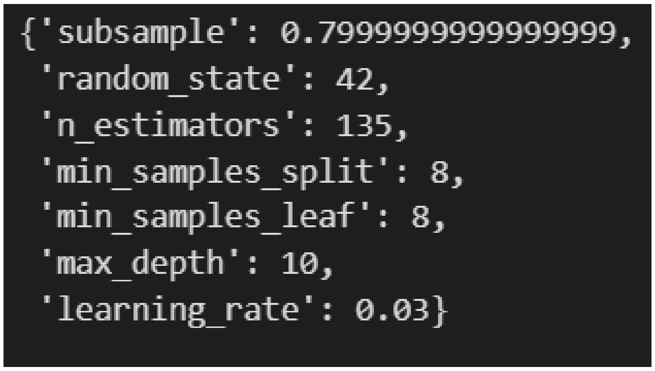
Given the performance of each model across each metric, the Gradient Boosting Classifier seems to be the best candidate for hyperparameter tuning due to its strong performance on both the train and validation sets, balancing recall and precision.

We used the following parameters for Randomized Search CV.



```
params_grid={
    'min_samples_leaf':range(5,10),
    'min_samples_split':range(5,10),
    'max_depth':range(10,20),
    'n_estimators':range(100,200),
    'learning_rate':np.arange(0.01,0.1,0.01),
    'subsample':np.arange(0.5,1,0.1),
    'random_state':[42]
}
```

The hyperparameter-tuned (HPT) model has slightly better overall performance compared to the default model, especially in terms of Precision and F1 Score on both the training and testing sets.



```
{'subsample': 0.7999999999999999,
 'random_state': 42,
 'n_estimators': 135,
 'min_samples_split': 8,
 'min_samples_leaf': 8,
 'max_depth': 10,
 'learning_rate': 0.03}
```

Model Performance before HPT

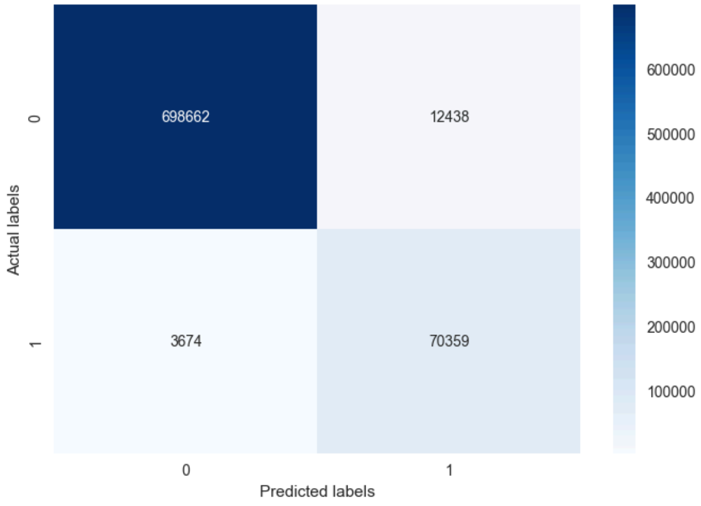
Model Name	Dataset	Accuracy	Recall	Precision	F1
Gradient Boosting Classifier	Training	0.975	0.947	0.820	0.879
	Testing	0.975	0.950	0.818	0.879
ROC AUC Score (Testing data)	0.99371				

Model Performance after HPT

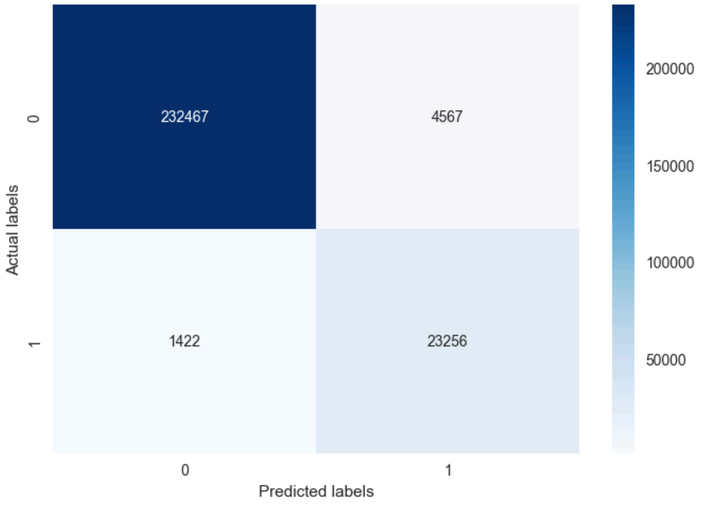
Model Name	Dataset	Accuracy	Recall	Precision	F1
Gradient Boosting Classifier	Training	0.979	0.950	0.850	0.897
	Testing	0.977	0.942	0.836	0.886
ROC AUC Score (Testing data)	0.99436				

Given the improvements in **accuracy, precision**, **F1 score**, and **ROC AUC** after hyperparameter tuning, we decided to select the **HPT model**. The improvements may seem small but it's still compelling.

Confusion Matrix for HPT Gradient Boosting Classifier

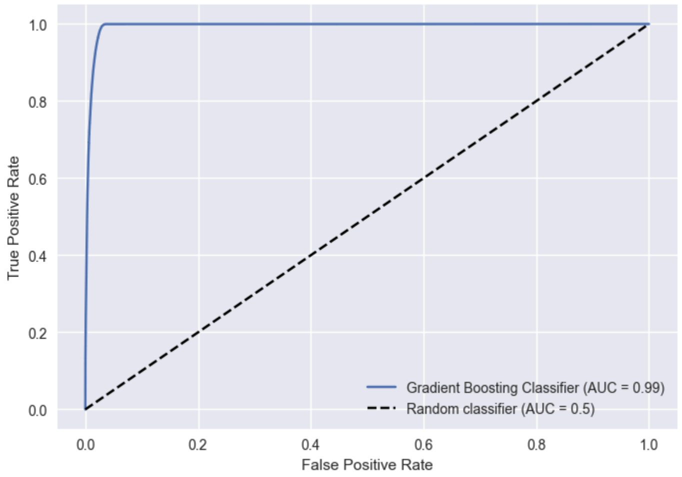


Training set



Testing set

ROC curve for HPT Gradient Boosting Classifier



The ROC curve for the HPT Gradient Boosting Classifier model on the testing set.

Save the ML Model

The trained model is saved in a .pkl file using joblib python library to used for saving and loading Python objects, for our purpose it’s the trained model. We named the file as model.pkl and it can be found in the model directory of this project or run the notebook.ipynb to create the .pkl file which is the model.

We also include python files to easily load the model from outside the notebook and see it’s prediction and metrics on the unseen data.

References

- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2

- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95.

- Joblib Development Team. (2020). *Joblib: Running Python functions as pipeline jobs*. Joblib Documentation. https://joblib.readthedocs.io/

- Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., … Willing, C. (2016). Jupyter notebooks – A publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), *Positioning and power in academic publishing: Players, agents and agendas* (pp. 87–90).

- McKinney, W., & Others. (2010). Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51–56).

- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct), 2825–2830.

- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17, 261–272. https://doi.org/10.1038/s41592-019-0686-2

- Waskom, M., Botvinnik, O., O'Kane, D., Hobson, P., Lukauskas, S., Gemperline, D. C., … Qalieh, A. (2017). *mwaskom/seaborn: v0.8.1 (September 2017)*. Zenodo. https://doi.org/10.5281/zenodo.883859

- Van Rossum, G., & Drake, F. L., Jr. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica.