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:

https://github.com/Jeomon/GST

Plagiarism Declaration

We, the undersigned, hereby declare that the work presented in this report is our own and has been developed as part of the **Analytics Hackathon on Developing a Predictive Model** organized by **GSTN**. We affirm that this report is free from plagiarism and all external sources, if any, have been properly cited and referenced according to the guidelines provided.

Declaration of Originality

1. Original Work: The content of this report, including the model code, documentation, and all associated deliverables, is our original work and has not been copied from any other source.
2. Citations: All work, ideas, and contributions of other authors, researchers, or institutions that have been used or referred to in this report are cited appropriately. The sources of such work are acknowledged in the reference section of the report.
3. External Resources: Any external resources or pre-trained models used in the development of this project have been clearly identified, and proper permissions have been obtained where necessary.
4. Plagiarism: We understand that plagiarism is a serious academic offense and any instance of plagiarism or failure to adhere to ethical guidelines will result in disqualification from the Hackathon. We assure that our work complies with the highest standards of academic and professional integrity.
5. Affirmation: By submitting this report, we affirm that the information provided is accurate and that we have adhered to all ethical and plagiarism guidelines set forth by the GSTN.

Team Members:

Jeomon George

Puduppuly Rajesh Surendran

Omkar Shende

Aathi Madhav G

Ramshad Abdul Raheem

Team ID: GSTN_651

Date: 26/09/2024

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

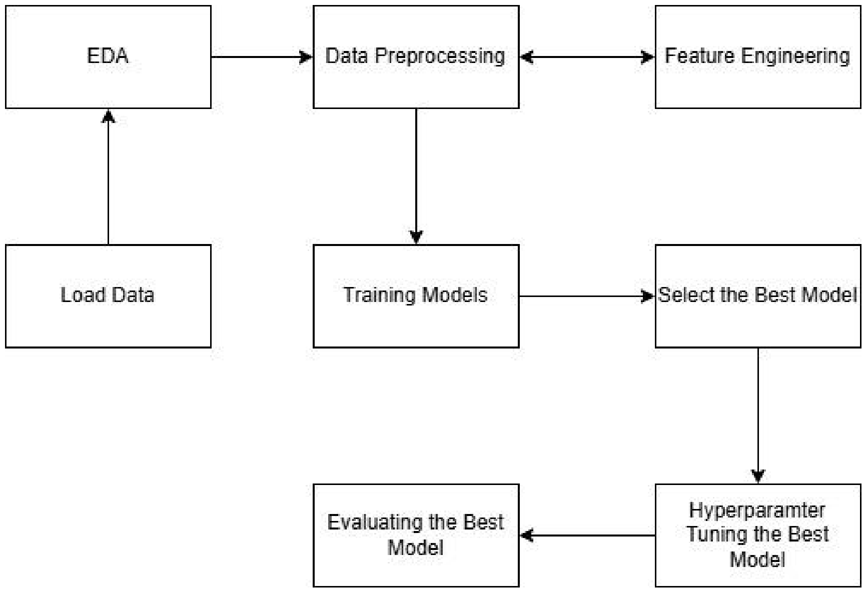


Fig. 1.0 shows the methodology followed for this project

3.1 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.

3.2 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.

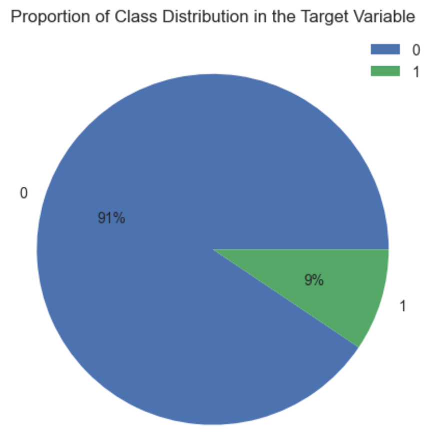


Fig 2.0 shows the class imbalance in the dataset

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.

3.3 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 hence we considered those as continuous features.

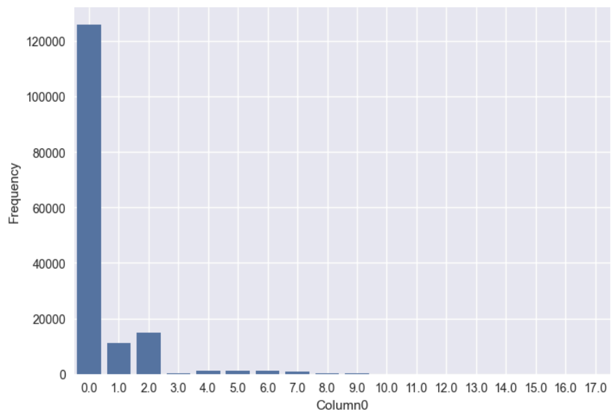
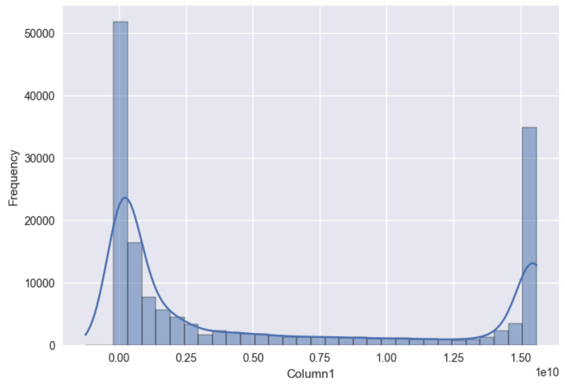


Fig 2.1 shows the histogram of column0

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.



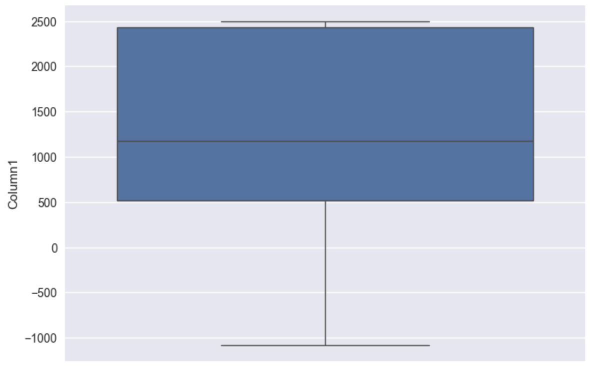
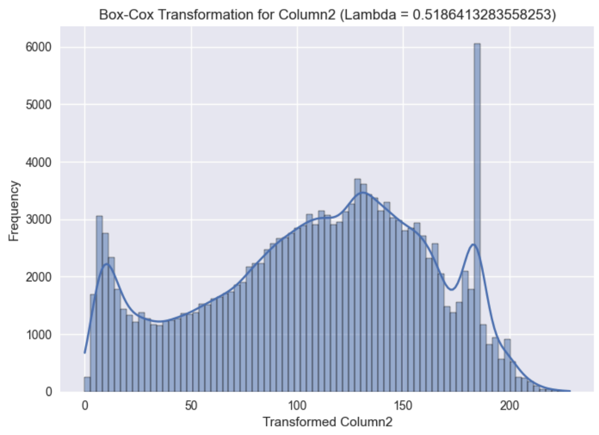


Fig 2.2 shows the histogram of column1 *Fig 2.3 shows the boxplot of column1*

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.



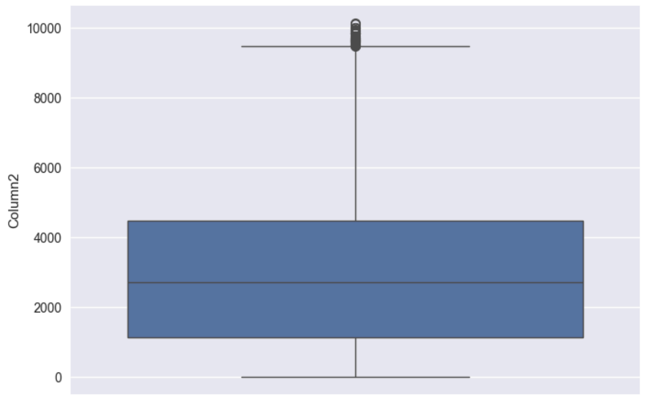
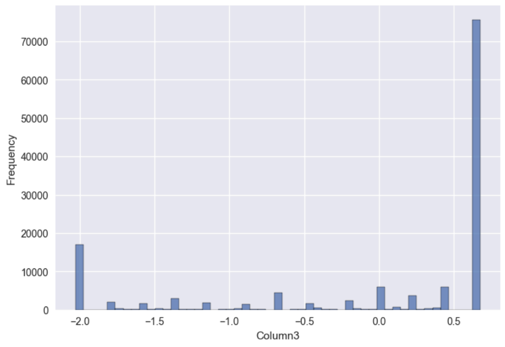


Fig 2.4 shows the histogram of column2 *Fig 2.5 shows the boxplot of column2*

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.



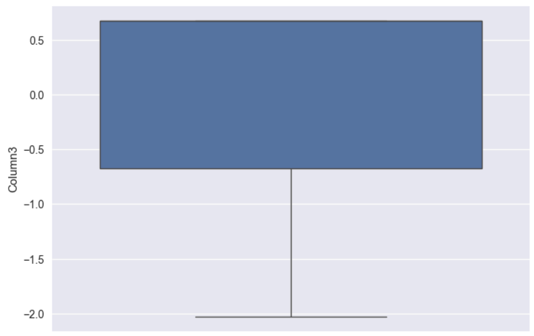


Fig 2.6 shows the histogram of column3 *Fig 2.7 shows the boxplot of column3*

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.

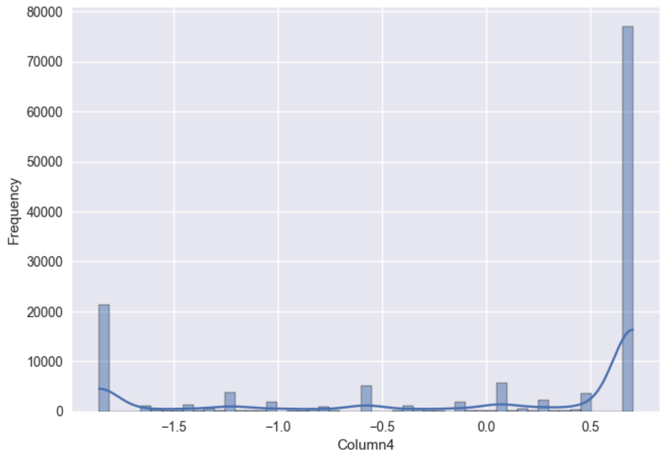


Fig 2.8 shows the histogram of column4

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.

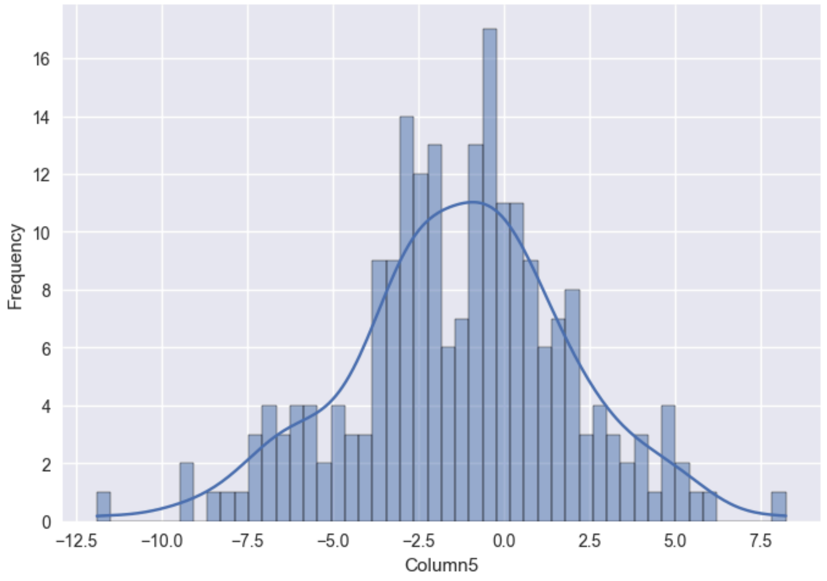


Fig 2.9 shows the histogram of column2

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 in this feature.

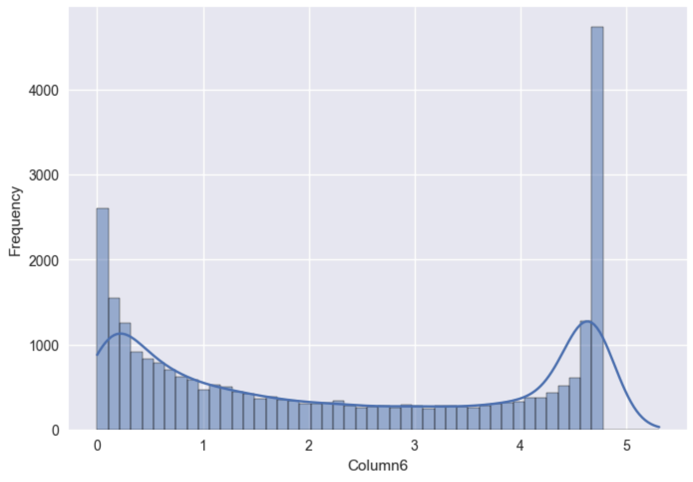


Fig 2.10 shows the histogram of column6

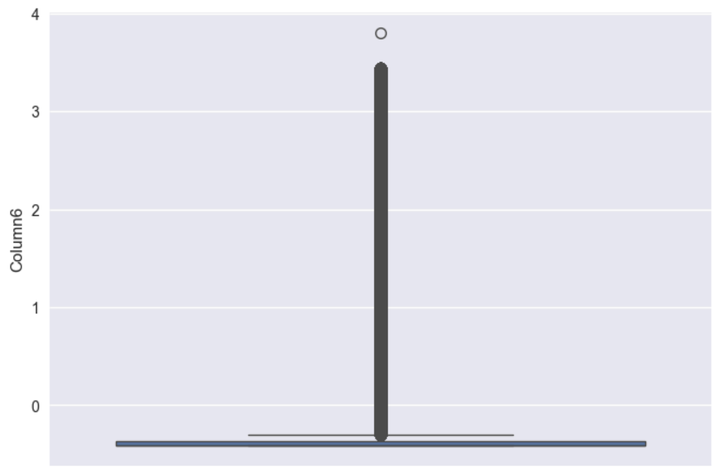


Fig 2.11 shows the boxplot of column6

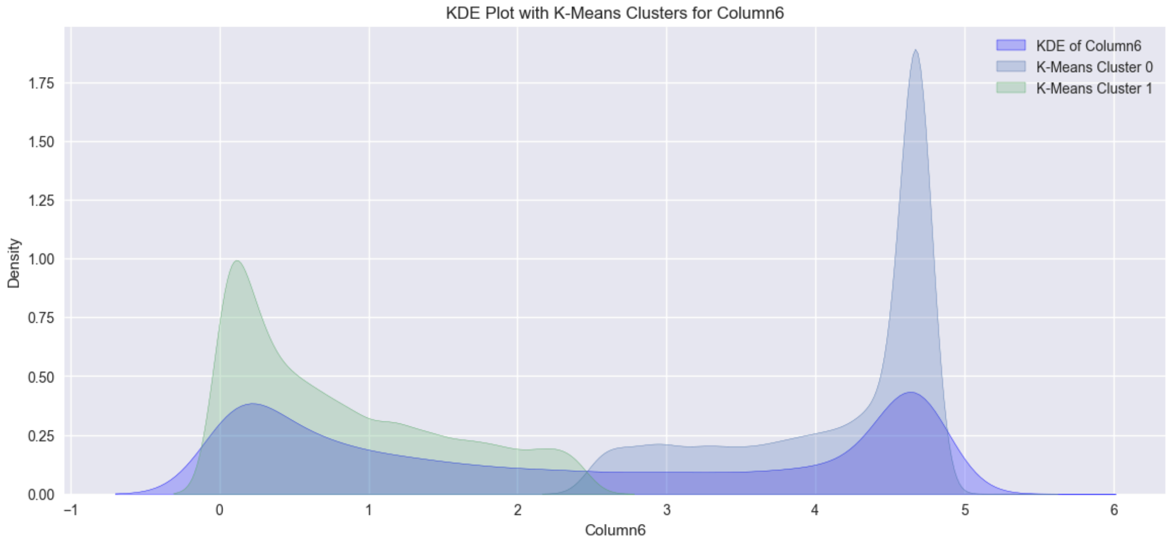


Fig 2.12 shows the KDE plot for Column6 to show the clustering

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.

3.4 Outliers Analysis

- Majority of the features have outliers but Column1, Column3, Column4, Column11-13 have zero outliers.
- 68.7% of the Train Data is consisting of outliers so we can't remove outliers for this case

rather keep them. The columns having less than 5% outliers we can cap them but when done that, it degrade the performance of the models.

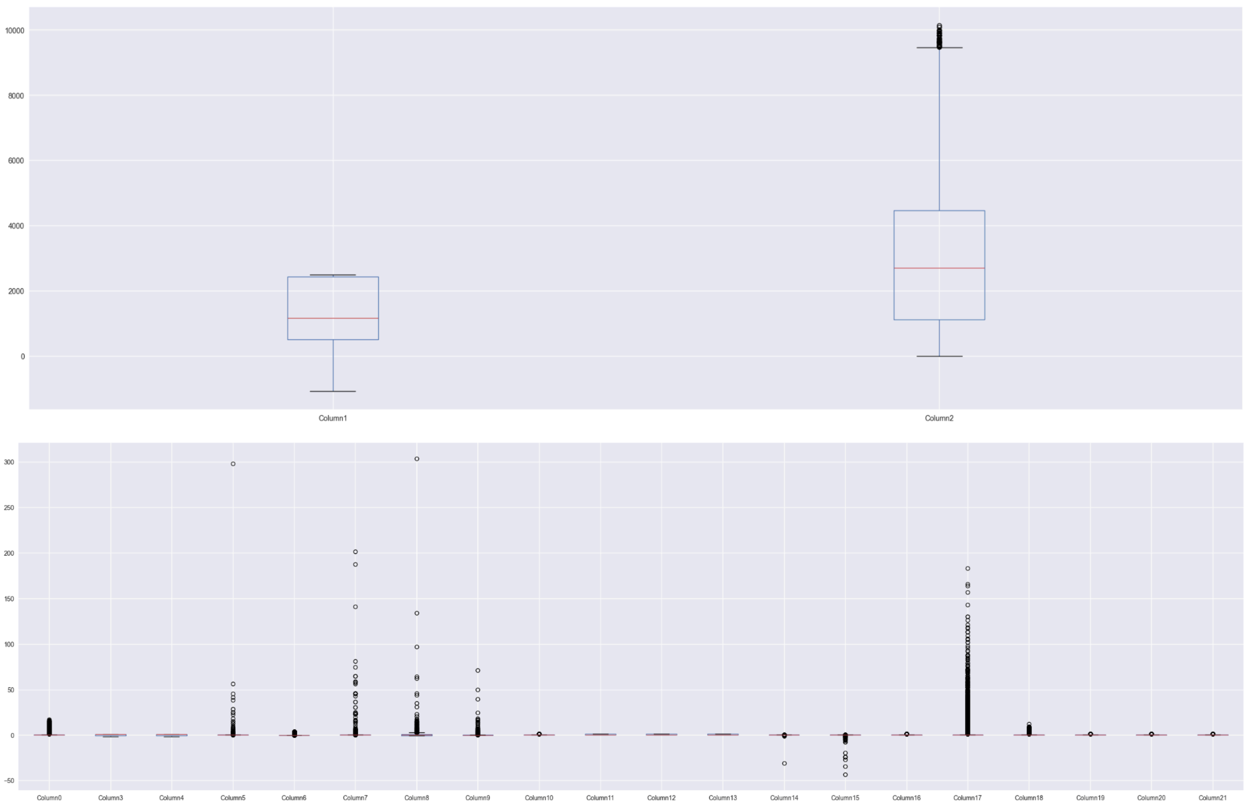


Fig 2.13 shows the boxplot in each column of the dataset

3.5 <u>Multivariate Analysis</u>

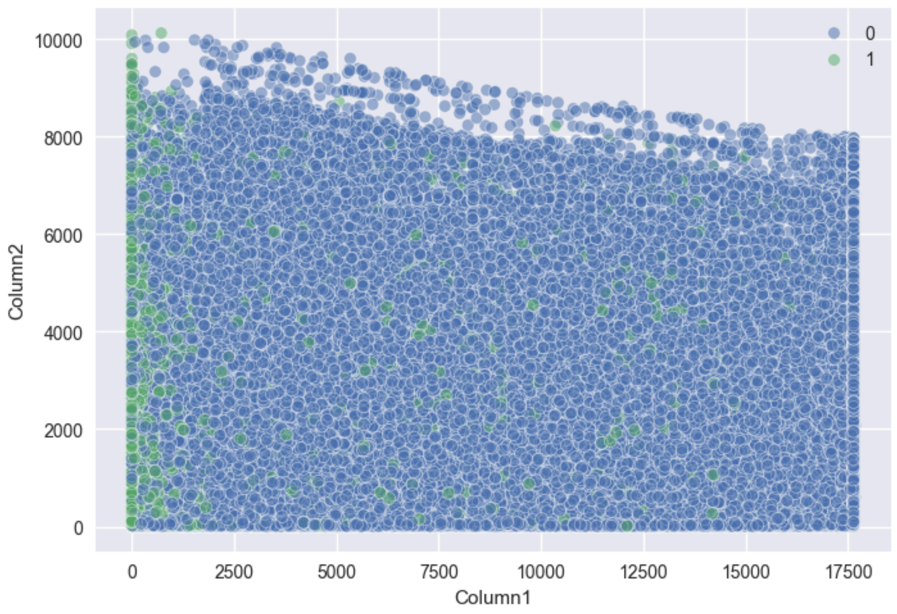


Fig 2.14 shows the scatterplot between Column2 and Column1

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.

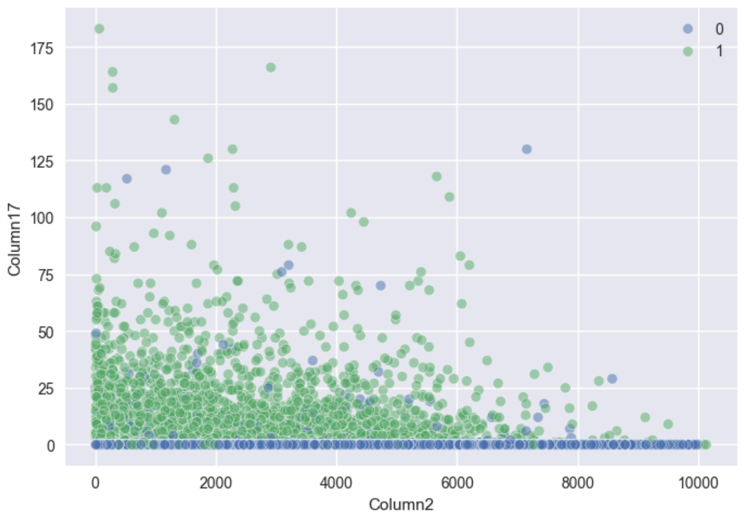


Fig 2.15 shows the scatterplot between Column17 and Column2

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**

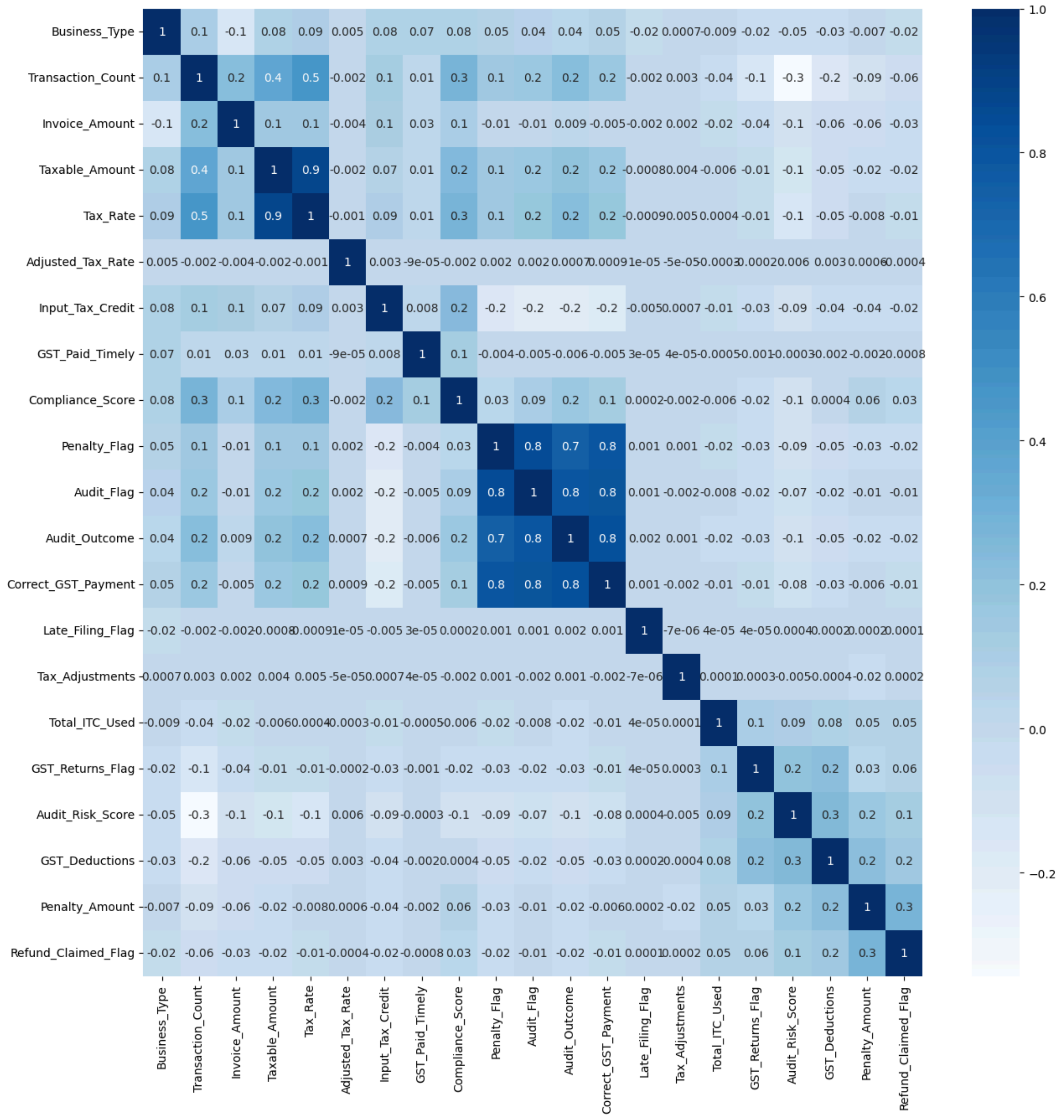


Fig 3.0 shows the correlation matrix of the dataset before removing the multicolinearity

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.

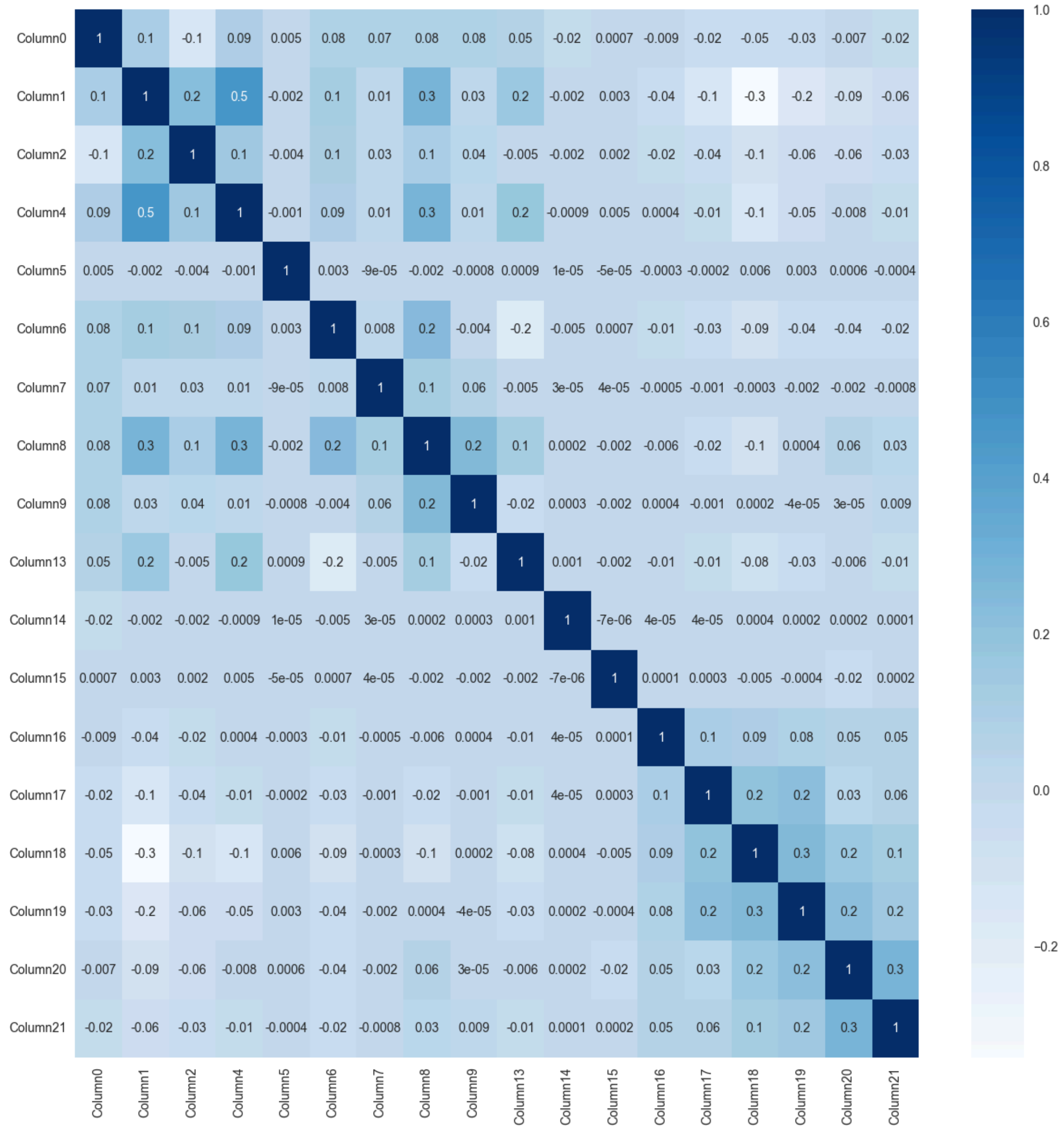


Fig 3.1 shows the correlation matrix of the dataset after removing the multicolinearity

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.

6. <u>Feature Scaling</u>

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.

7. <u>Model Training and Selection</u>

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.

7.1 <u>Model Performance (Cross Validation)</u>

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
- XGBoost 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
XGBoost Classifier	Train	0.981, 0.000	0.948, 0.001	0.860, 0.001	0.902, 0.000
	Validation	0.978, 0.001	0.933, 0.003	0.846, 0.001	0.888, 0.001

Table 1.0 shows the performance of each model against each metrics

Top Contenders:

- **XGBoost Classifier:** Stands out with the highest recall and precision, offering an impressive balance between these metrics. It achieves the highest F1 score and generalizes well on both training and testing data. Its performance reflects strong handling of complex data, including imbalanced classes and potential outliers, making it a top choice for applications requiring high accuracy and robustness.

- **Random Forest Classifier:** Delivers exceptional performance across accuracy, recall, precision, and F1 score with very low standard deviation. Its ability to generalize well and maintain a good balance between metrics without overfitting makes it a highly reliable model. It’s a solid choice for scenarios requiring robust performance with less sensitivity to hyperparameters.

- **Gradient Boosting Classifier:** Shows slightly better recall than Random Forest but has marginally lower precision. It is consistent and competitive, providing strong overall performance with high F1 scores. It’s well-suited for tasks where capturing the majority of positive cases is critical, though it may need careful tuning to balance precision and recall effectively.

Runners-Up:

- **AdaBoost Classifier:** Performs well but slightly lags behind the top contenders in terms of precision and overall metrics. It maintains consistent performance across recall and F1 score, making it a good choice for scenarios where a balanced approach to recall and precision is needed, but with a bit less emphasis on absolute performance compared to Random Forest and XGBoost.
- **Logistic Regression:** Serves as a strong baseline with reliable performance, but does not match the top contenders in recall and precision for more complex tasks. It’s suitable for simpler tasks where interpretability and straightforward modeling are prioritized over achieving the highest possible performance metrics.

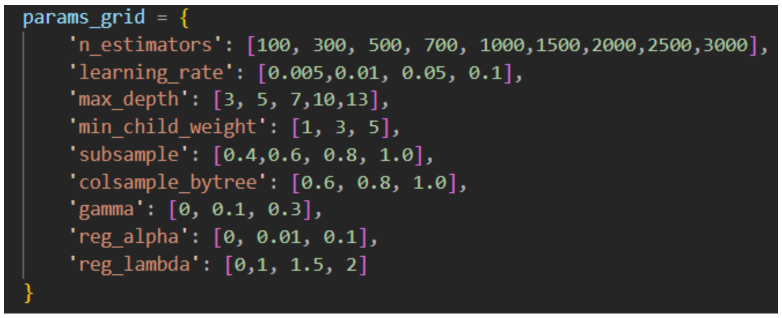
Underperformers:

- **SGD Classifier:** Displays notably lower recall compared to other models, which undermines its suitability for handling imbalanced classes. Its lower performance metrics overall make it less effective for scenarios where high recall is essential.
- **Ridge Classifier:** Provides decent performance but lacks the balance of precision and recall seen in the top ensemble methods. It is less capable in scenarios where nuanced trade-offs between recall and precision are required.
- **Decision Tree Classifier:** Shows perfect training performance but suffers from significant overfitting, leading to lower performance on testing data. Its high variance indicates that it’s less reliable for generalization and more prone to overfitting issues.

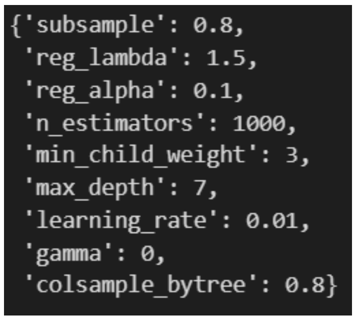
XGBoost Classifier is the best candidate for further tuning due to its superior balance of recall and precision, coupled with strong generalization. **Random Forest Classifier** is a close second for its robustness and consistency, while **Gradient Boosting Classifier** offers competitive performance with a focus on high recall.

8. <u>Hyperparameter Tuning</u>

Given the performance of each model across each metric, the **XGBoost 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.



```python
params_grid = {
    'n_estimators': [100, 300, 500, 700, 1000,1500,2000,2500,3000],
    'learning_rate': [0.005,0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7,10,13],
    'min_child_weight': [1, 3, 5],
    'subsample': [0.4,0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.3],
    'reg_alpha': [0, 0.01, 0.1],
    'reg_lambda': [0,1, 1.5, 2]
}
```



```
{'subsample': 0.8,
 'reg_lambda': 1.5,
 'reg_alpha': 0.1,
 'n_estimators': 1000,
 'min_child_weight': 3,
 'max_depth': 7,
 'learning_rate': 0.01,
 'gamma': 0,
 'colsample_bytree': 0.8}
```

Fig 4.0 shows the parameter grid for HPT and the selected parameters from the HPT

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.

Model Name	Dataset	Accuracy	Recall	Precision	F1
XGBoost Classifier	Training	0.980	0.946	0.857	0.899
	Testing	0.978	0.937	0.845	0.889
ROC AUC Score	0.99466				

Table 2.0 shows performance of the XGBoost Classifier model before HPT on testing set

Model Name	Dataset	Accuracy	Recall	Precision	F1
XGBoost Classifier	Training	0.979	0.945	0.848	0.894
	Testing	0.978	0.944	0.841	0.890
ROC AUC Score	0.99477				

Table 2.1 shows performance of the XGBoost Classifier model after HPT on testing set

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

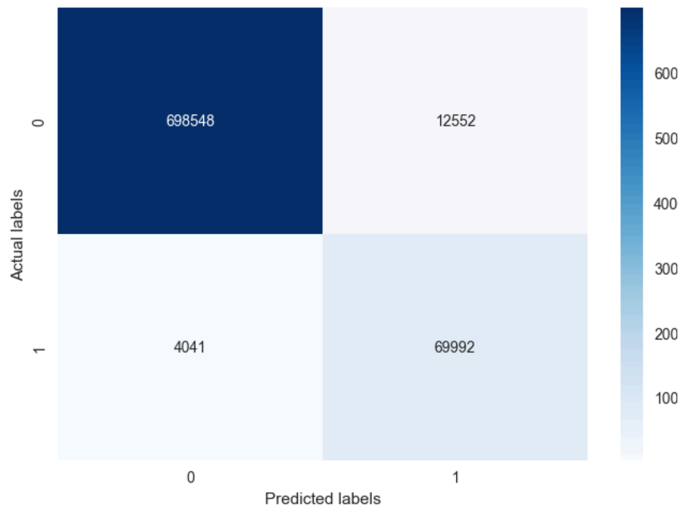


Fig 5.0 shows the performance of XGBoost Classifier on training set

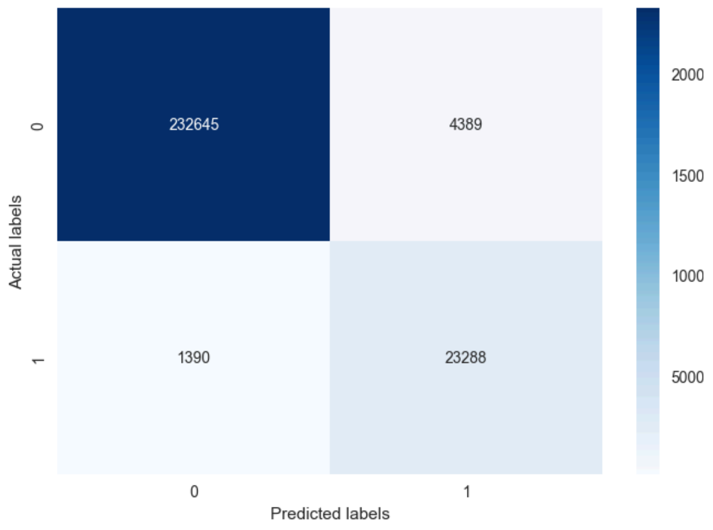


Fig 5.1 shows the performance of XGBoost Classifier on testing set

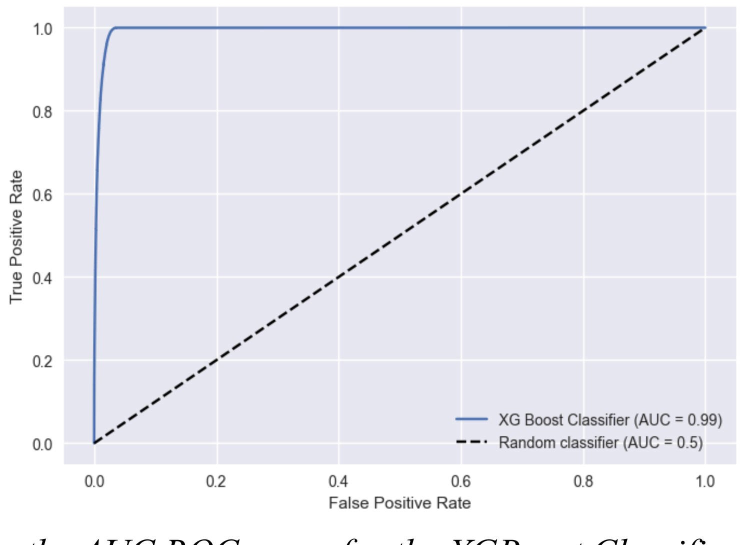


Fig 5.2 shows the AUC ROC curve for the XGBoost Classifier on testing set

9. <u>Save the HPT Model</u>

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.

- Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). New York, NY, USA: ACM. https://doi.org/10.1145/2939672.2939785