Team Name: Machine

Team Members:

1. Subodh Khanduri (Indian Institute of Technology Kharagpur)
2. Brijesh Dangwal (National Institute of Technology Uttarakhand)
3. Shivam Dangwal (G.B.P.I.E.T.)

Section:

XGBoost Model for Feature Importance Extraction

Objective:

The goal of this process is to prepare the dataset, build an Lightgbm and XGBoost model, and extract important features based on their contribution to the prediction of the target variable (`bad_flag`). This includes handling class imbalance, applying stratified sampling, scaling the data, and extracting and sorting feature importance.

Steps Involved:

1. Data Loading and Preprocessing

We start by reading the dataset `Dev_data_to_be_shared.csv` using `pandas`. The dataset contains 1216 columns, with the target variable `bad_flag` being used to predict the likelihood of a class imbalance.

data = pd.read_csv('Dev_data_to_be_shared.csv')

- **Null Value Handling:**
  - Any null values in the dataset are filled with `0`.
  - This is done to maintain a clean dataset without affecting the model's performance due to missing values: data.fillna(0)
- **Class Distribution:** The target variable `bad_flag` has two classes, and their distribution is printed as follows:

0	95434
1	1372

2. Data Splitting

- Class Imbalance: Class `0` being heavily overrepresented compared to class `1`. Therefore, we apply a **stratified sampling** technique to create a balanced dataset for training.

This helps the model see similar class distributions in both the training and testing datasets, which is important for accurate evaluation.

X_train = train_data.drop(columns=['bad_flag', 'account_number'])

y_train = train_data['bad_flag']

- `X_train` consists of all the features, while `y_train` is the target variable (`bad_flag`).
- The test set is also prepared similarly, using the test data (`test_data`), and the same columns (`bad_flag` and `account_number`) are excluded.

3. Feature Scaling

Since machine learning models like XGBoost are sensitive to the scale of the features, it is crucial to standardize the data. The **StandardScaler** is used to scale the training and test data, ensuring that all features are centered around 0 with a standard deviation of 1.

4. Training the XGBoost Model

The next step involves training the XGBoost model. After training the model, the **feature importances** are extracted, which give us an indication of how much each feature contributes to the model's predictions. These importances are crucial for feature selection and understanding which variables are most impactful.

feature_importance = best_model.feature_importances_

- The importance values are stored and exported to a CSV file, `feature_importance.csv`, which lists the features and their importance scores.

5. Sorting and Saving Feature Importance

We filter out features with zero importance and sort them in descending order of importance. This sorted list helps in identifying the most influential features that should be prioritized in the next stages of analysis or model refinement.

filtered_df = new[new['Value'] != 0]

sorted_df = filtered_df.sort_values(by='Value', ascending=False)

- The sorted feature importance values are saved in `sorted.csv` to allow easy access and further analysis.

sorted_df.to_csv('sorted.csv', index=False)

Conclusion:

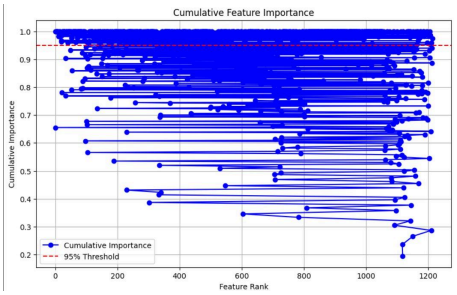
- **Balanced Sampling:** The stratified sampling method ensured that the model trained on a balanced dataset, helping prevent bias towards the majority class.
- **Data Scaling:** Feature scaling was performed to standardize the data and prevent any feature from dominating the model due to varying scales.
- **XGBoost Model:** The XGBoost model was trained, and the most important features were extracted to help understand which attributes have the highest impact on the target variable.
- **Feature Importance:** The feature importance values were sorted and saved for easy access, helping guide future steps in feature selection or model improvement.

Data Visualization

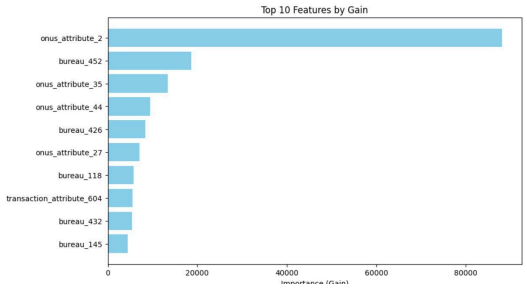
- **Class Imbalance**:

Value	Count	Frequency (%)
0	95434	98.6%
1	1372	1.4%

- **Lightgbm model:** Selected 351 features contributing to 95% of importance. These were stored in a "`374_attributes.csv`" file.



- **Top 10 Featured**



- **XGBoost model:** After training, the **feature importances** are extracted and stored in **sorted.csv**
- Top 10 values of **sorted.csv**:

	Value	Attributes
0	0.019186	onus_attribute_17
1	0.018101	bureau_452
2	0.010174	onus_attribute_2
3	0.007787	onus_attribute_26
4	0.007395	transaction_attribute_302
5	0.006655	transaction_attribute_660
6	0.006553	transaction_attribute_424
7	0.006037	onus_attribute_35
8	0.006018	transaction_attribute_141
9	0.005955	transaction_attribute_126
10	0.005602	bureau_426

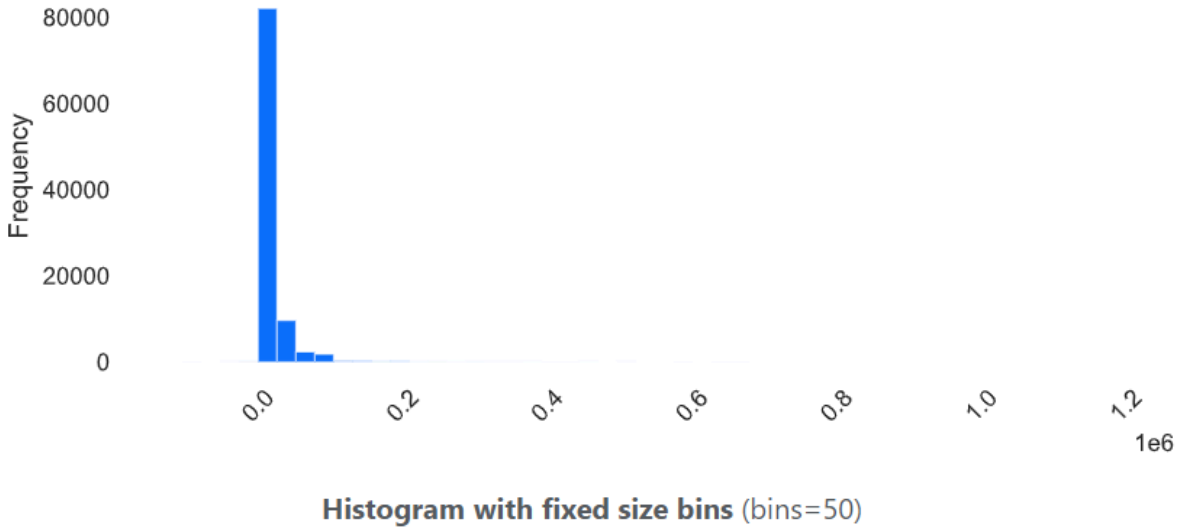
- In **sorted.csv, 488** Attributes were correlated.
- From these 488, the top 50 features were selected, and every feature was explored and understood. Eg:

transaction_attribute_105

Real number (ℝ)

High correlation **Zeros**

Distinct	27199	**Minimum**	-106045
Distinct (%)	28.1%	**Maximum**	1200000
Missing	0	**Zeros**	25231
Missing (%)	0.0%	**Zeros (%)**	26.1%
Infinite	0	**Negative**	178
Infinite (%)	0.0%	**Negative (%)**	0.2%
Mean	12274.778	**Memory size**	756.4 KiB



Histogram with fixed size bins (bins=50)

Statistics | Histogram | Common values | Extreme values

Quantile statistics

Minimum	-106045
5-th percentile	0
Q1	0
median	2596.855
Q3	12130
95-th percentile	51500
Maximum	1200000
Range	1306045
Interquartile range (IQR)	12130

Descriptive statistics

Standard deviation	26513.901
Coefficient of variation (CV)	2.1600309
Kurtosis	138.1569
Mean	12274.778
Median Absolute Deviation (MAD)	2596.855
Skewness	7.4084816
Sum	1.1882722×10^{9}
Variance	7.0298693×10^{8}
Monotonicity	Not monotonic

bureau_112 is highly overall correlated with bureau_404

bureau_195 is highly overall correlated with bureau_277 and 1 other fields

bureau_277 is highly overall correlated with bureau_195 and 4 other fields

bureau_278 is highly overall correlated with bureau_277 and 3 other fields

bureau_315 is highly overall correlated with bureau_195 and 4 other fields

bureau_317 is highly overall correlated with bureau_277 and 3 other fields

bureau_357 is highly overall correlated with bureau_277 and 3 other fields

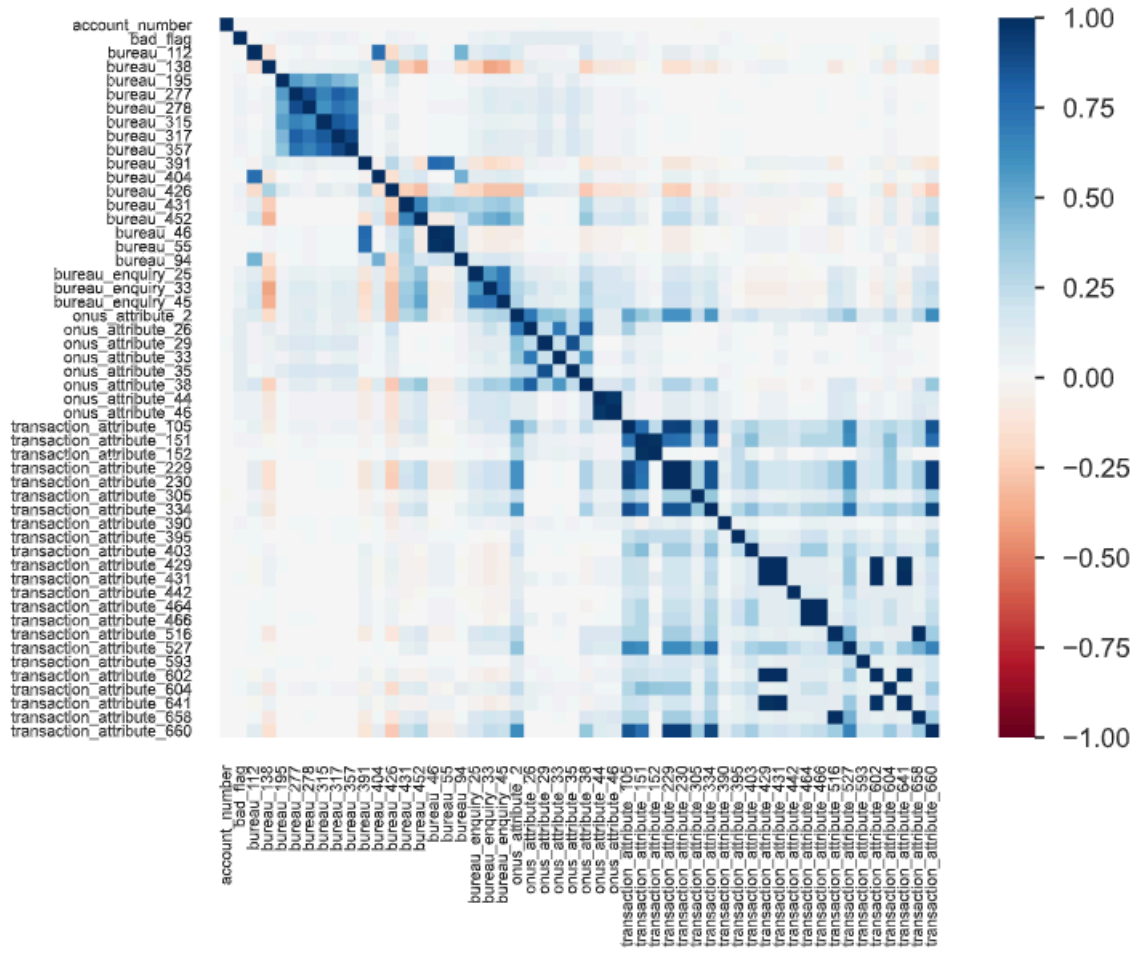
bureau_391 is highly overall correlated with bureau_46 and 1 other fields

bureau_404 is highly overall correlated with bureau_112

bureau_431 is highly overall correlated with bureau_452

- Then **Correlation matrix** between 50 features were calculated:

	account_number	bad_flag	bureau_112	bureau_138	bureau_195	bureau_277	bureau_278	bureau_315	bureau_317	bureau_357
account_number	1.000	0.000	-0.004	0.004	0.004	0.003	0.000	0.002	0.005	0.000
bad_flag	0.000	1.000	0.000	0.016	0.039	0.049	0.047	0.034	0.044	0.039
bureau_112	-0.004	0.000	1.000	-0.128	0.000	0.009	0.028	0.004	0.013	0.013
bureau_138	0.004	0.016	-0.128	1.000	0.015	0.059	0.060	0.054	0.066	0.064
bureau_195	0.004	0.039	0.000	0.015	1.000	0.552	0.484	0.532	0.447	0.405
bureau_277	0.003	0.049	0.009	0.059	0.552	1.000	0.876	0.667	0.810	0.734
bureau_278	0.000	0.047	0.028	0.060	0.484	0.876	1.000	0.594	0.724	0.662
bureau_315	0.002	0.034	0.004	0.054	0.532	0.667	0.594	1.000	0.840	0.761
bureau_317	0.005	0.044	0.013	0.066	0.447	0.810	0.724	0.840	1.000	0.906
bureau_357	0.000	0.039	0.013	0.064	0.405	0.734	0.662	0.761	0.906	1.000
bureau_391	-0.003	0.014	-0.020	0.118	0.071	0.052	0.079	0.058	0.050	0.055
bureau_404	-0.001	0.000	0.751	-0.101	0.006	0.009	0.034	0.000	0.009	0.009
bureau_426	-0.004	0.049	-0.169	0.316	0.126	0.046	0.036	0.106	0.056	0.081
bureau_431	-0.001	0.026	0.108	-0.237	0.008	0.005	0.003	0.000	0.000	0.000



- **50 columns** contributing **0** correlation

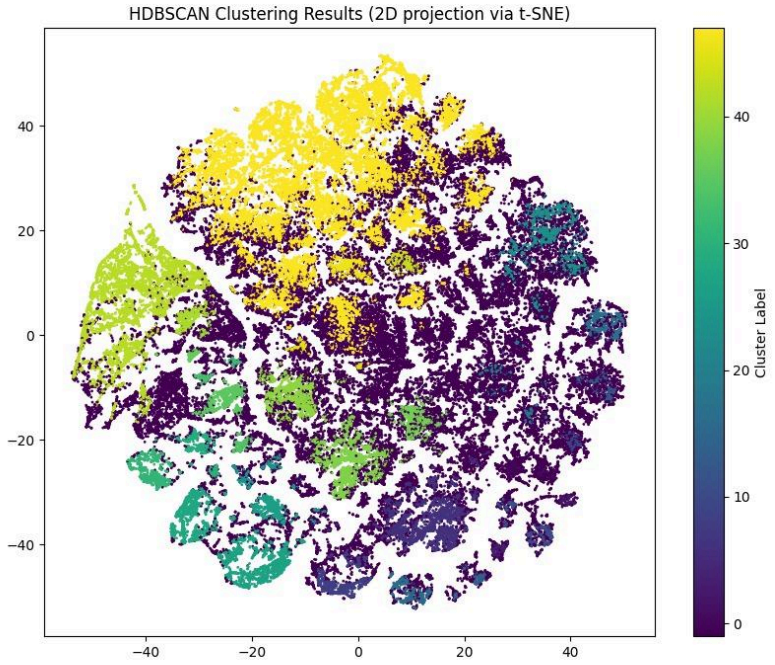
```
columns_with_all_nan = data.columns[(data.isna() | (data == 0.0)).all()].tolist()

print(columns_with_all_nan)
```

['bureau_4', 'bureau_16', 'bureau_26', 'bureau_38', 'bureau_47', 'bureau_56', 'bureau_70', 'bureau_80', 'bureau_90', 'bureau_100', 'bureau_110', 'bureau_120', 'bureau_131', 'bureau_142', 'bureau_152', 'bureau_162', 'bureau_172', 'bureau_182', 'bureau_192', 'bureau_202', 'bureau_212', 'bureau_222', 'bureau_232', 'bureau_242', 'bureau_252', 'bureau_262', 'bureau_272', 'bureau_282', 'bureau_292', 'bureau_302', 'bureau_312', 'bureau_322', 'bureau_332', 'bureau_342', 'bureau_352', 'bureau_362', 'bureau_372', 'bureau_382', 'bureau_392', 'bureau_402', 'bureau_412', 'bureau_423', 'bureau_436', 'bureau_447', 'onus_attribute_28', 'bureau_enquiry_7', 'bureau_enquiry_17', 'bureau_enquiry_27', 'bureau_enquiry_37', 'bureau_enquiry_47']

HDBSCAN Clustering Analysis

Introduction

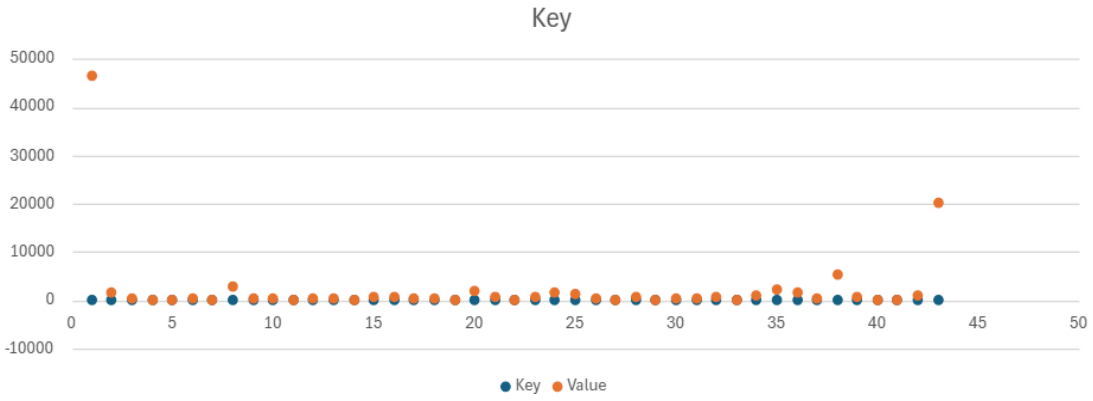
The application of the HDBSCAN clustering algorithm to a dataset with 96,000 samples and 50 features. The primary objective was to identify distinct clusters and analyze their characteristics. Visualization of the clustering results was performed using t-SNE for dimensionality reduction.



Data Preprocessing

1. **Standardization**:
    - The features were scaled using the `StandardScaler` to normalize the data.

Clustering with HDBSCAN

2. **HDBSCAN Parameters**:
    - `min_cluster_size`: 80
    - `min_samples`: 10
    - `cluster_selection_epsilon`: 0.5
    - `metric`: Euclidean
3. **Results**:
    - **Number of Clusters Found**: 42
    - 

4. **Noise**:
    - Points labeled as `-1` are considered noise. In this dataset, **46,578** points (48.5%) were classified as noise.

Cluster Visualization

5. **Dimensionality Reduction**:
    - t-SNE was applied to reduce the dataset to two dimensions for visualization.
    - Parameters:
        - `perplexity`: 30
        - `learning_rate`: 200
6. **Visualization Output**:
    - A 2D scatter plot showing clusters and noise points was generated.
    - The image file provided (`my_plot.png`) displays the clustering results.

Cluster Characteristics

The following is a summary of distinct clusters:

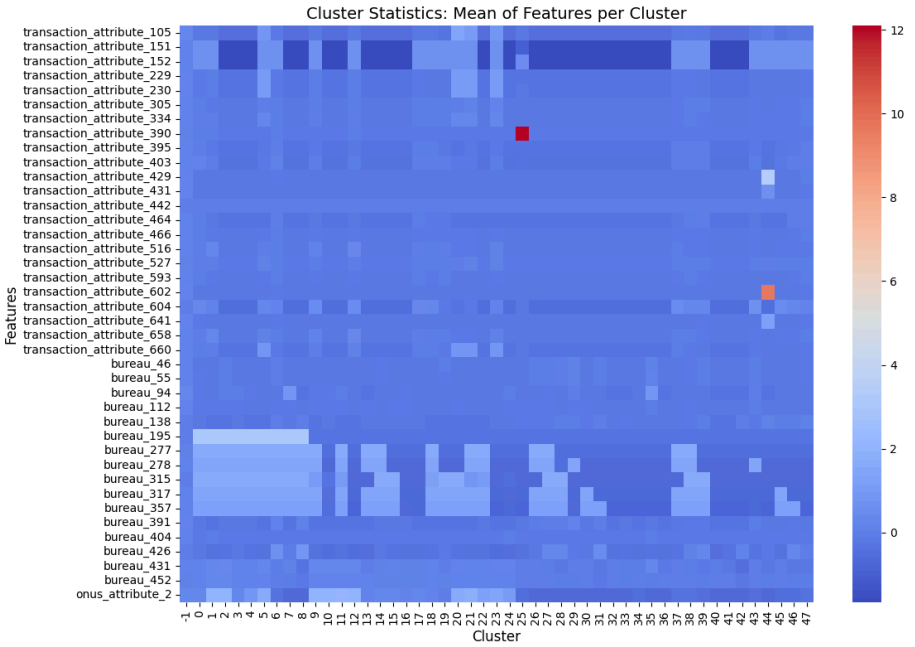
Cluster	Size	Characteristics
-1	46578	Noise, scattered data points
0	1644	Compact cluster with moderately dense points
6	2724	Large and well-defined cluster
18	2009	Moderate size, tightly grouped
33	2233	Medium density cluster
36	5340	Large, highly populated cluster
41	20333	Largest cluster with high density
Others	Varies	Smaller, sparsely populated clusters

Key Observations:

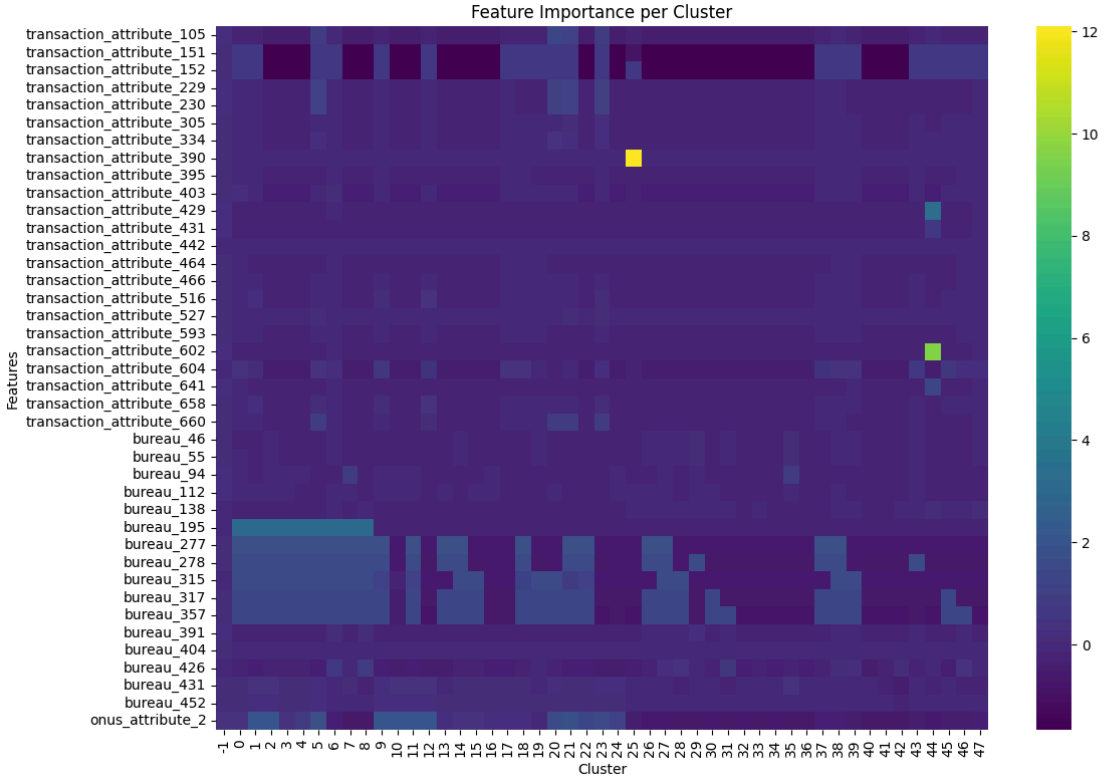
- Cluster **41** and **36** are the largest, representing the dominant groups in the data.
- Smaller clusters such as **0**, **6**, and **18** indicate localized groupings.
- High noise (~48.5%) suggests the presence of many outliers or a need for parameter adjustment.

Conclusions and Recommendations

1. **Clustering Results**:
    - Clusters **41**, **36**, and **6** dominate in terms of size and density.
    - Cluster Stats



- Feature Importance



Code:

1. Data Preprocessing
Standardization:

- Features were scaled using StandardScaler to normalize the data.
- Training and testing datasets were processed separately:
- Training data: X_train = scaler.fit_transform(X_train)
- Testing data: X_test = scaler.transform(X_test)

2. Model Development
a. Libraries Used:

- Core libraries: numpy, pandas, and matplotlib.
- Machine learning and preprocessing:
- TensorFlow: Used for advanced ML workflows.
- scikit-learn: Tools for splitting data, scaling, and evaluation metrics.
- **LightGBM**: A gradient boosting framework for fast and efficient modeling.

b. Understanding data:

- XGBClassifier and Lightgbm were chosen for their efficiency and support for GPU optimization:
- XGBClassifier and Lightgbm were used for important feature extractions and understanding the distribution of data.
- HDBSCAN Visualization of the clustering results was performed using t-SNE for dimensionality reduction.
- **Exploratory Data Analysis (EDA):** was done with the help of ydata_profiling

c. Model Selection:

- Final choice was made and **LightGbm** was chosen from XGboost, randomforest, deep neural network and Lightgbm.
- **Why Lightgbm:** LightGBM is faster and more efficient for large datasets due to its histogram-based algorithm and leaf-wise tree growth. It performs well with high-dimensional feature spaces because of its efficient feature bundling technique,lightGBM handles sparse data efficiently.

d. Hyperparameter Tuning:

- **Tuning Approach**:GridSearchCV and Cross-Validation(StratifiedKFold)
- **n_estimators**: Number of boosting iterations (tested: 60).
- **learning_rate**: Learning rate for boosting (tested: 0.05, 0.1).
- **max_depth**: Maximum tree depth to control model complexity (tested: 4, 6).
- **num_leaves**: Number of leaves in each decision tree (tested: 31, 50).
- **min_child_samples**: Minimum number of data points required in a leaf (tested: 2, 4, 8).
- **min_child_weight**: Minimum weight for each leaf node to control overfitting (tested: 0.5, 1, 10).
- **subsample**: Fraction of data to randomly sample to avoid overfitting (tested: 1.0).

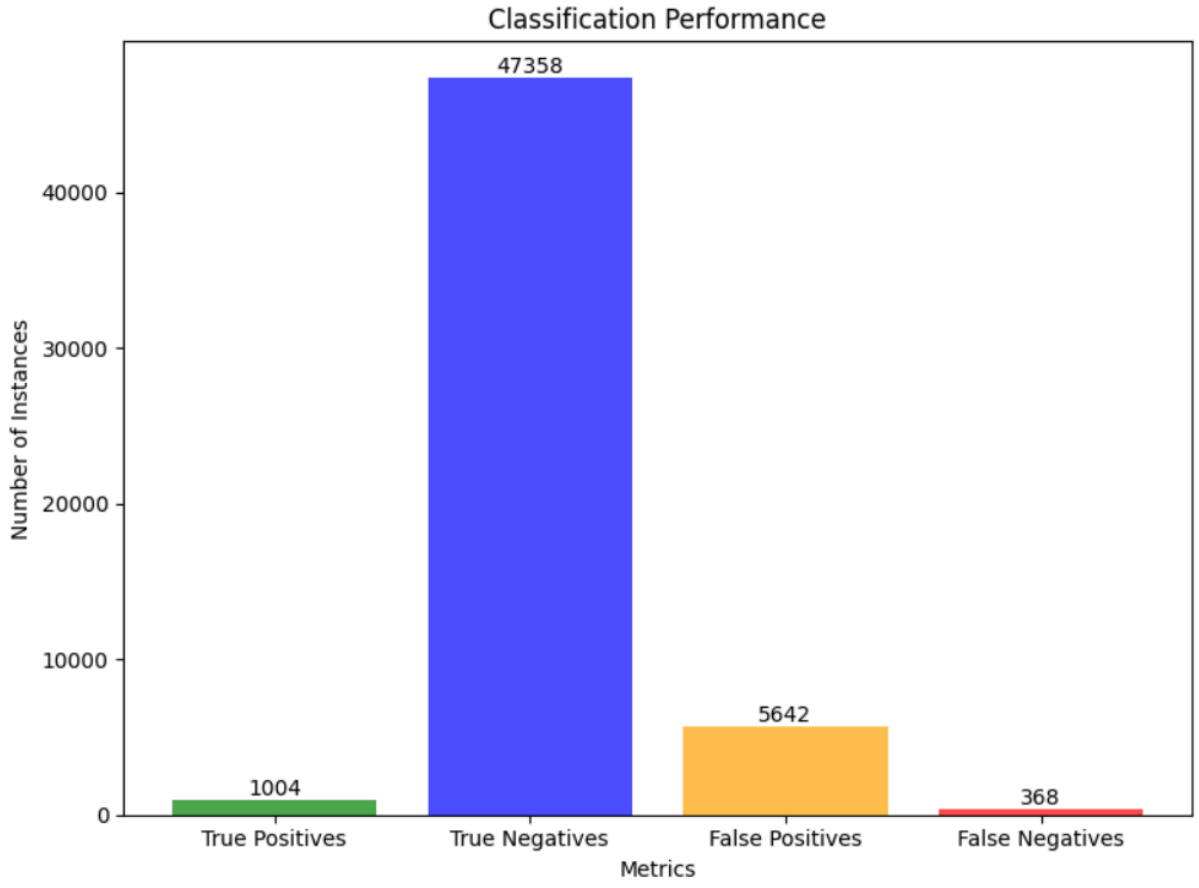
- **colsample_bytree**: Fraction of features to randomly sample at each split (tested: 0.8).

e. Best Parameter and Score:

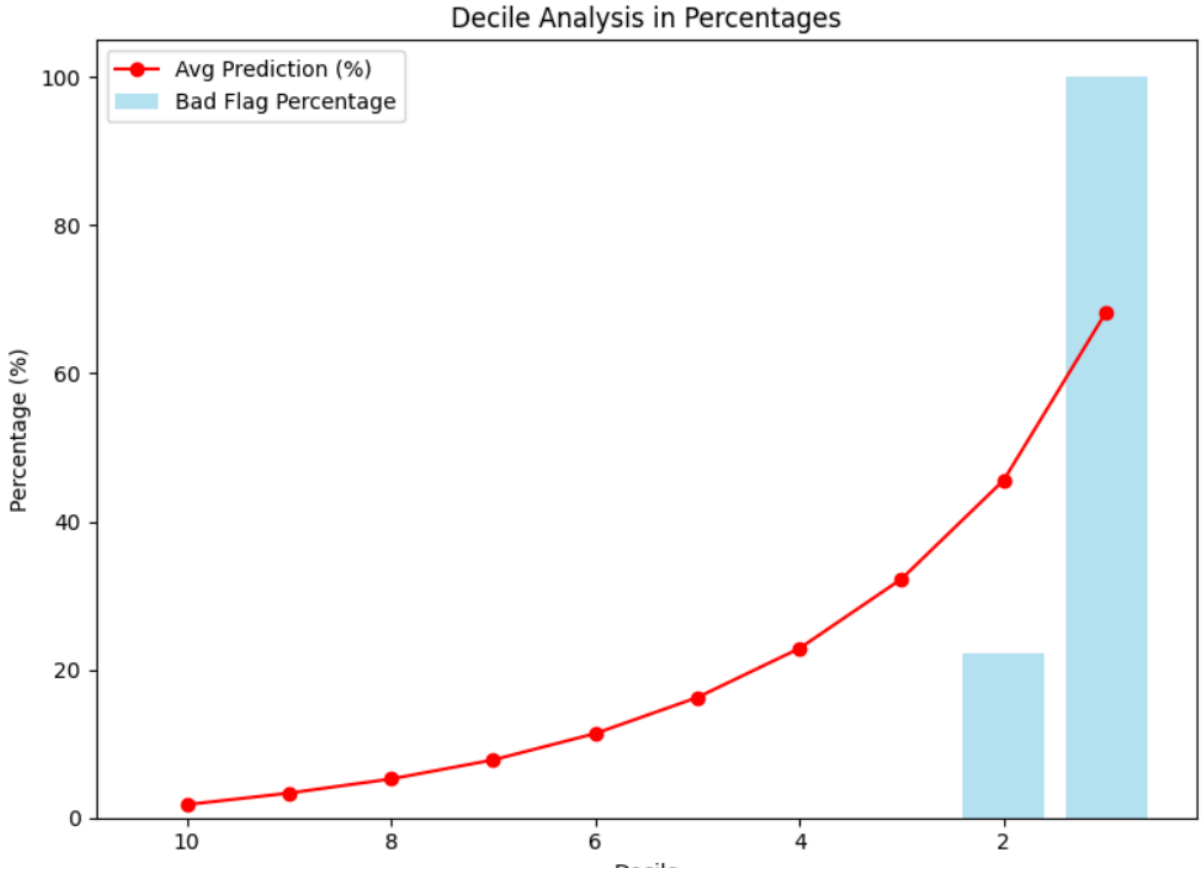
- Best Parameters:
  {'colsample_bytree': 0.8,
  'learning_rate': 0.1,
  'max_depth': 6,
  'min_child_samples': 4,
  'min_child_weight': 10,
  'n_estimators': 60,
  'num_leaves': 50,
  'subsample': 1.0}
- Best ROC AUC Score: 0.8286171383647799

3. Model Evaluation

a. Classification Performance:



b. Decile Chart::

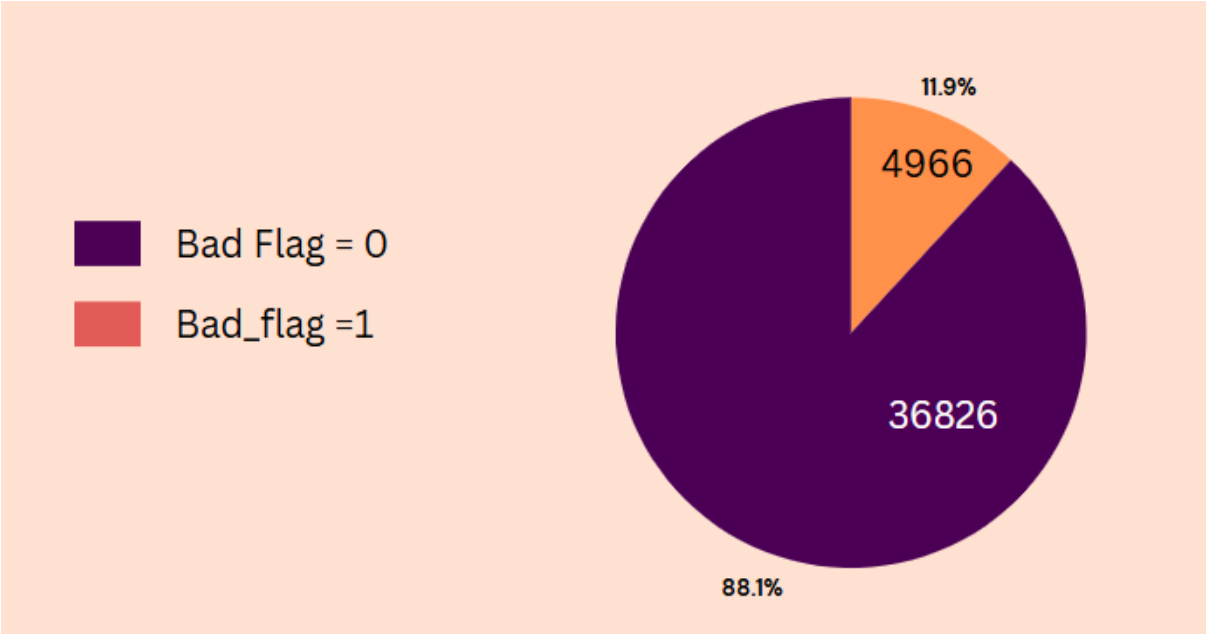


---

4. Results

- For Test file:

.



5. Conclusion

The LightGBM model, with its optimal hyperparameters, emerged as the best choice for this task, offering a robust solution with high efficiency and good predictive performance.