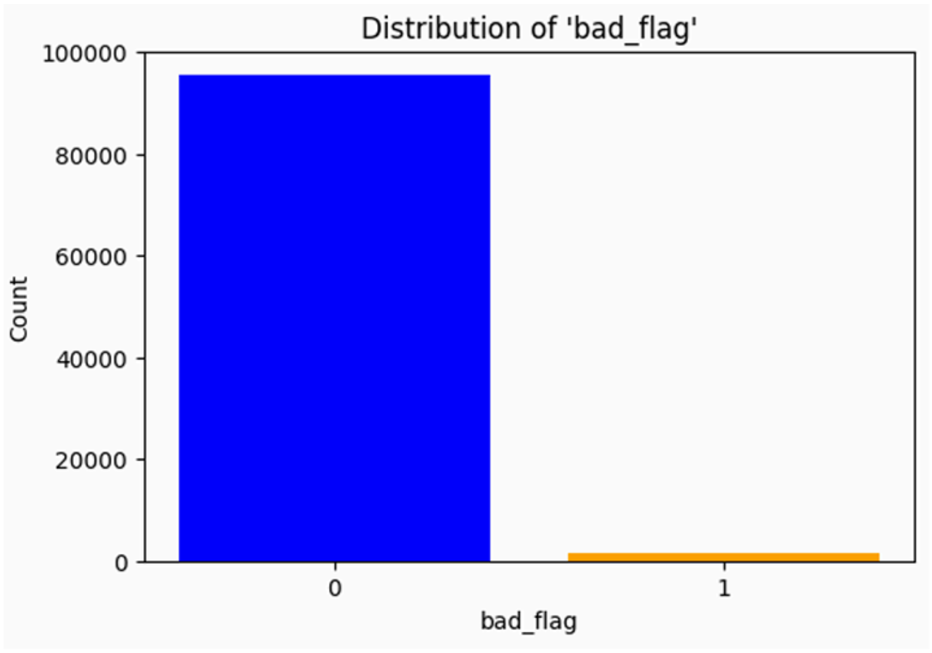
DOCUMENTATION OF CONVOLVE 3.0

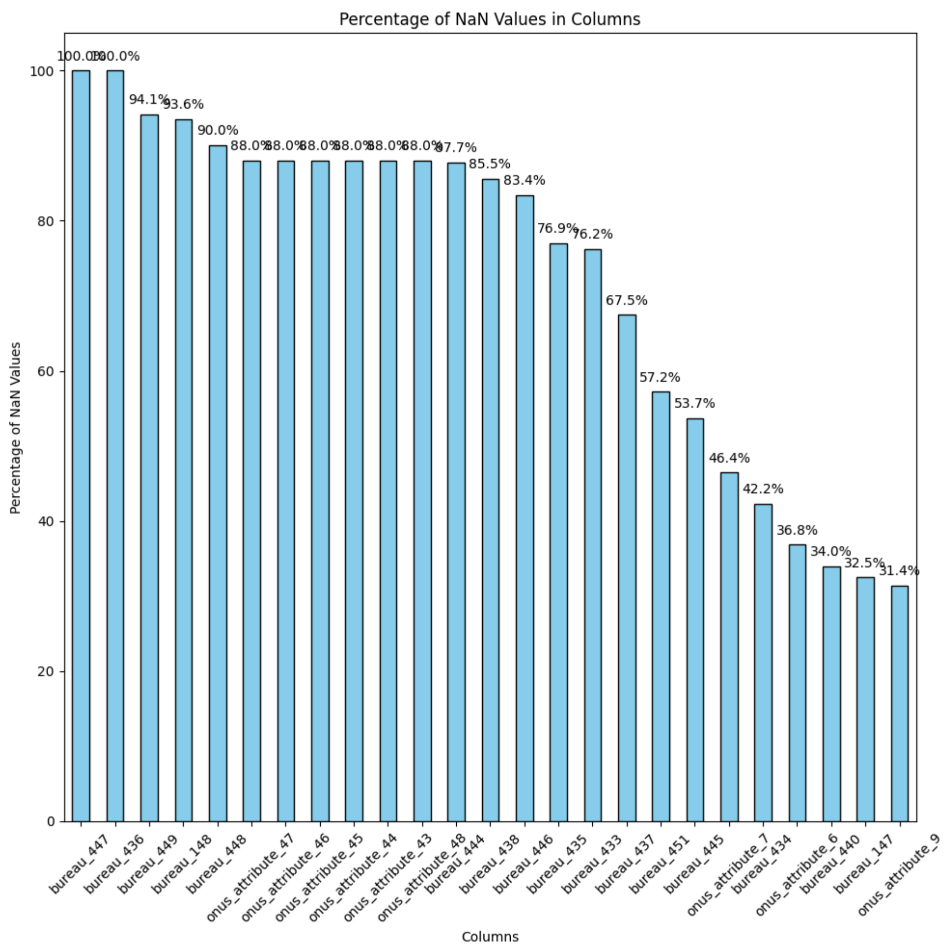
INITIAL INSIGHTS AND ANALYSIS:

- A total of 1214 columns were divided into separate categories called transaction_columns, bureau_columns, enquiry_columns, and onus_columns.

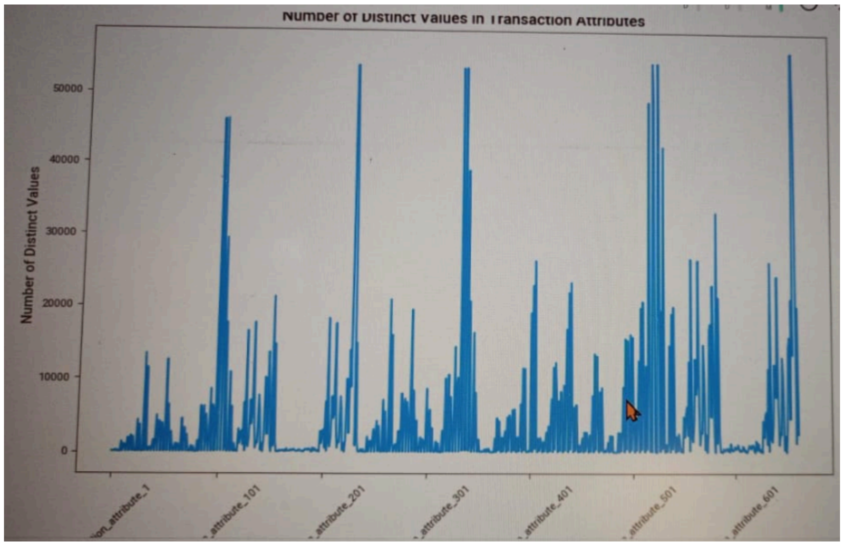


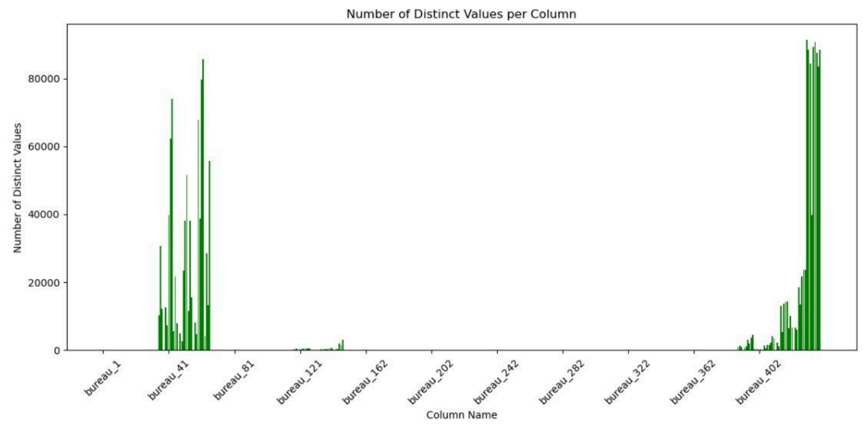
That's a **severe class imbalance**!

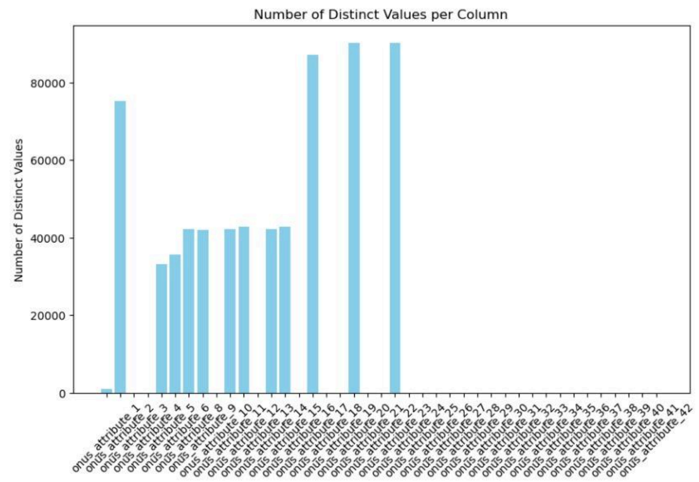
- 2 columns that had all NaN values were dropped. Percent NaN values for each column was plotted against the column name.

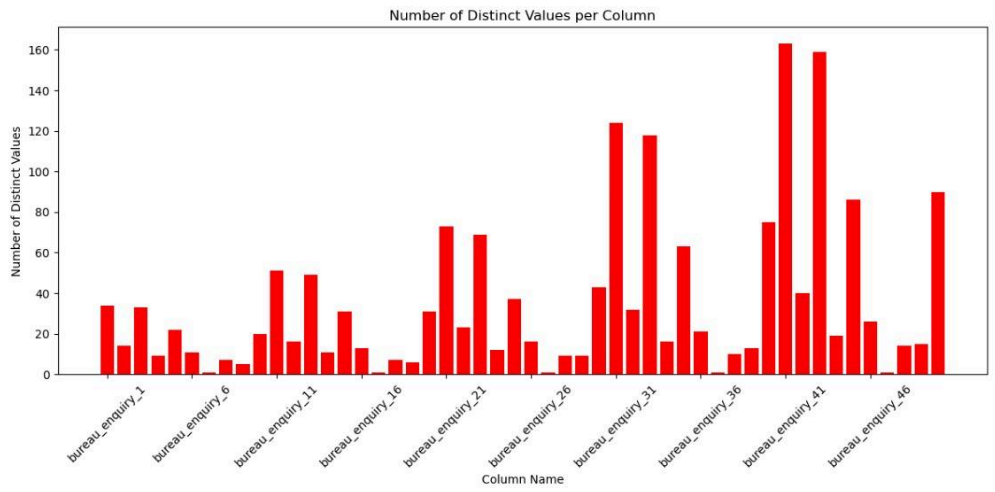


- Each set of columns were divided into sets of categorical and regressive columns for imputational purposes.For this purpose we plotted the number of distinct values in each column vs. the column name.





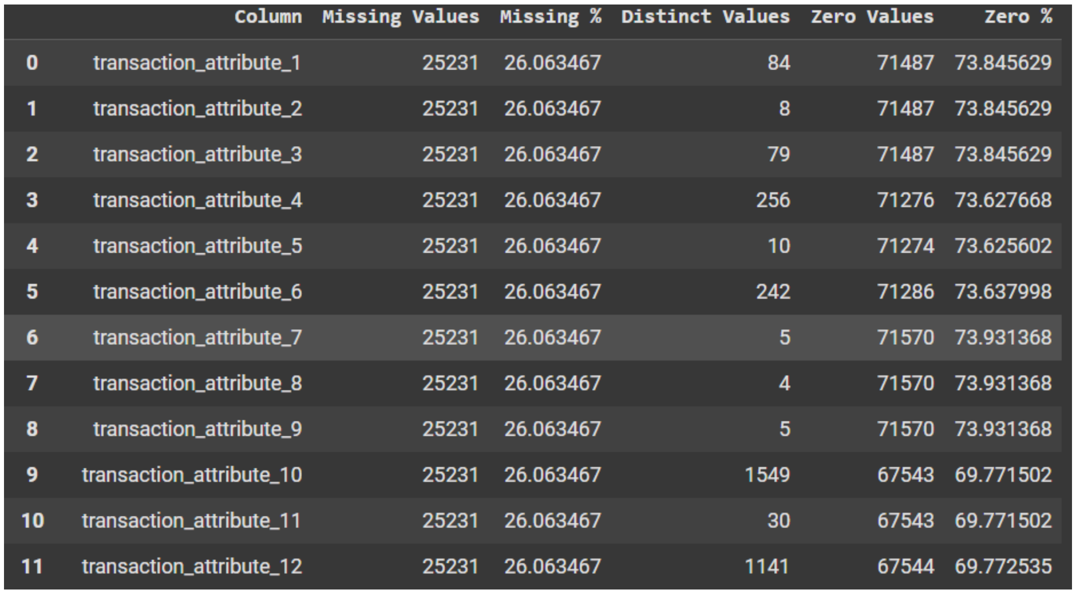




`

- **Interesting observation in the transaction_columns:**
  There are triads of consecutive columns which have very similar numbers of zeroes and the middle row of the triad has a very small number of distinct values in comparison to the extreme columns of the triad . On inspection, we find that these 3 columns of the triad take non-zero values simultaneously, the middle column taking a small integral

value while the two extreme columns take on large and distinct values the difference between which is observed to increase with increase in the middle column value. **We hypothesize that each triad represents transactions within a particular time frame, the middle one capturing the number of transactions while the extreme columns capturing the maximum and minimum value of the transactions within that time frame.**



	Column	Missing Values	Missing %	Distinct Values	Zero Values	Zero %
0	transaction_attribute_1	25231	26.063467	84	71487	73.845629
1	transaction_attribute_2	25231	26.063467	8	71487	73.845629
2	transaction_attribute_3	25231	26.063467	79	71487	73.845629
3	transaction_attribute_4	25231	26.063467	256	71276	73.627668
4	transaction_attribute_5	25231	26.063467	10	71274	73.625602
5	transaction_attribute_6	25231	26.063467	242	71286	73.637998
6	transaction_attribute_7	25231	26.063467	5	71570	73.931368
7	transaction_attribute_8	25231	26.063467	4	71570	73.931368
8	transaction_attribute_9	25231	26.063467	5	71570	73.931368
9	transaction_attribute_10	25231	26.063467	1549	67543	69.771502
10	transaction_attribute_11	25231	26.063467	30	67543	69.771502
11	transaction_attribute_12	25231	26.063467	1141	67544	69.772535

Summary of Our Exploratory Data Analysis(EDA) and Feature Engineering:

Our project involved enhancing a dataset by analyzing and handling missing values (`NaN`) in specific columns and creating new features for modeling. Here's a breakdown of the key tasks and insights:

1. **Identifying and Analyzing `NaN` Groups**:
    - We first identified which columns in our dataset (`x_train_dropped`) had `NaN` values. This led to discovering groups of columns where `NaN` values often appeared together.
    - Using a count-based approach, we grouped these columns by their `NaN` occurrences and tracked how many rows had missing values for each of these groups.
    - We then analyzed the distribution of the `bad_flag=1` (target variable) for each group, focusing on larger groups where the count of rows was significant (> 400).
2. **Creating Binary Features Based on `NaN` Groups**:
    - Based on the identified `NaN` groups, we created new binary features (e.g., `nan_1`, `nan_2`, ...) where:
        - A value of `1` indicates that the row contains `NaN` for all columns in that group.
        - A value of `0` indicates that not all columns in the group have `NaN`.
    - This was done only for groups with more than 400 occurrences, thus optimizing computational efficiency.

3. **One-Hot Encoding for Categorical Columns**:
    - For columns with few distinct values, we applied one-hot encoding to represent each unique value (including `NaN`).
    - The one-hot encoding resulted in binary columns representing each distinct value, including a column for `NaN` values (using the `dummy_na=True` option).
    - The original columns were retained alongside these new one-hot encoded features.
4. **Data Output**:
    - The final dataset (`x_train_with_new_features`) includes both the original columns and the newly created binary features (for `NaN` groups and one-hot encoding).
    - We saved the updated dataset to CSV for future use in modeling.

Insights:

- **`NaN` Grouping and Target Distribution**: By grouping columns with `NaN` values together, we gained insights into which combinations of features might be more informative with respect to the target variable (`bad_flag=1`). This is valuable for feature engineering, as certain `NaN` patterns could be predictive of the target.

- **Efficiency Considerations**: Focusing only on groups with a high number of occurrences (i.e., > 400) and skipping smaller groups allowed us to optimize performance. This reduces redundant calculations and focuses attention on more meaningful patterns.

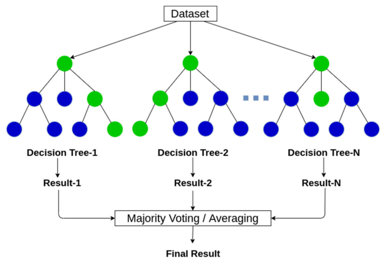
- **Feature Expansion**: The process of creating binary features based on the presence of `NaN` in specific columns (and one-hot encoding categorical columns) significantly expanded the feature set, which could help machine learning models identify more complex relationships in the data.

- **Handling Missing Data**: This approach gives us a robust way to handle missing values. The inclusion of `NaN` columns as features (both in the form of binary indicators and one-hot encoding) ensures that the models are informed about the presence of missing values, which can often carry useful information.

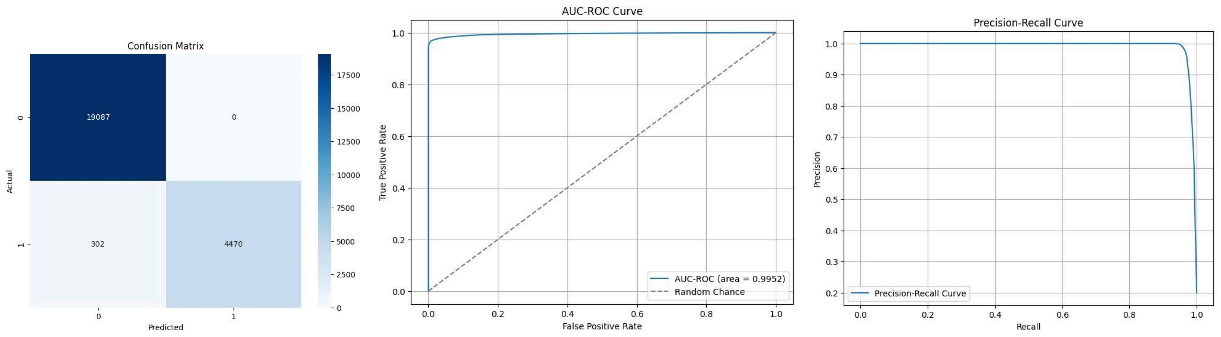
Model Selection and Training

In this project, three distinct models were trained and evaluated to determine the best-performing model for the given problem. These models are:

Random Forest: A robust ensemble learning method based on decision trees, Random Forest was selected for its ability to handle large datasets, capture non-linear relationships, and provide an efficient model with minimal hyperparameter tuning. This model has proven to be particularly effective in dealing with overfitting, making it a strong candidate for the task.

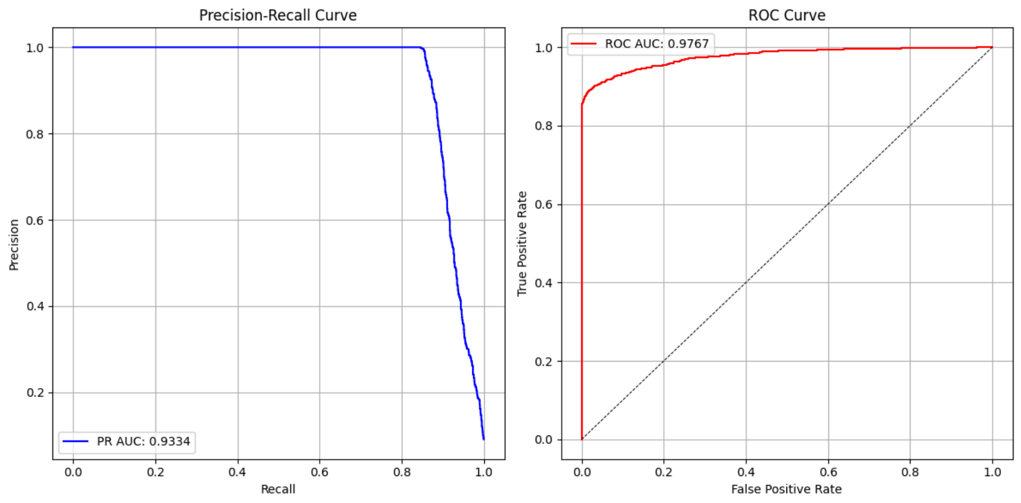


Result Overview:

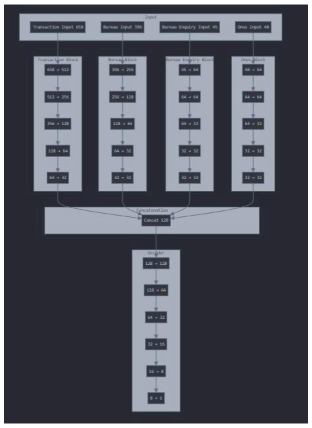


XGBoost: *XGBoost* , an implementation of gradient-boosted trees, was chosen for its impressive performance in structured data tasks. By optimizing the model’s parameters, we aimed to maximize predictive power and improve model accuracy. XGBoost's ability to reduce bias and variance through boosting makes it a highly reliable model for this task.

Result Overview:



Deep Learning Neural Network: A deep learning approach was also explored, using a fully connected neural network with multiple layers. Neural networks are known for their ability to learn complex patterns in data, and we aimed to leverage this power to achieve higher accuracy. The deep learning model was trained with a large set of data, utilizing techniques such as dropout and batch normalization to prevent overfitting.Different Head were made for different categories of attributes which were further combined to capture intra category and inter category patterns of the data.



Model Performance

After training all three models, the performance was evaluated using appropriate evaluation metrics (e.g., accuracy, precision, recall, F1-score). Among the models tested, **Random Forest** emerged as the most effective, outperforming both XGBoost and the deep learning model. Random Forest demonstrated the best balance between predictive accuracy and computational efficiency, making it the top choice for this project.

Documentation prepared by:

Arunangshu Karmakar, Aditya Debnath, Aiswani Mondal

(IIT Kharagpur)