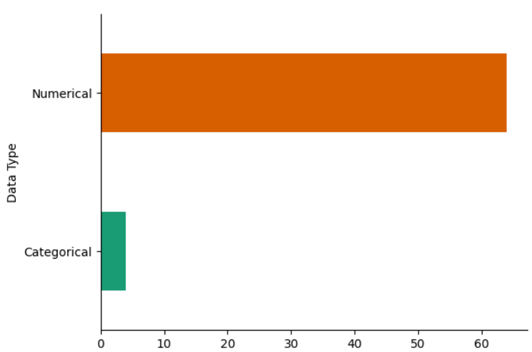
SynergyX2024 Datathon Competition
Team : Farid_Shaheb_Fan_Club

Member 1 : MD. Nayeem
Member 2 : MD. Jahid Hasan Jim
Member 3 : Shakhoyat Shujon

Defining the Problem

In SynergyX2024 Datathon Competition , the given Dataset contains train.csv , test.csv, sample.csv . In the train.csv file we have **621165** rows**, 68** columns**.** In these 68 columns one column is our target column. In this dataset this column is **v16. v16** column is basically a categorical column, which contains 2 categories (0 & 1). So, it's a classification problem, more specifically it's a binary classification problem.

Exploratory Data Analysis



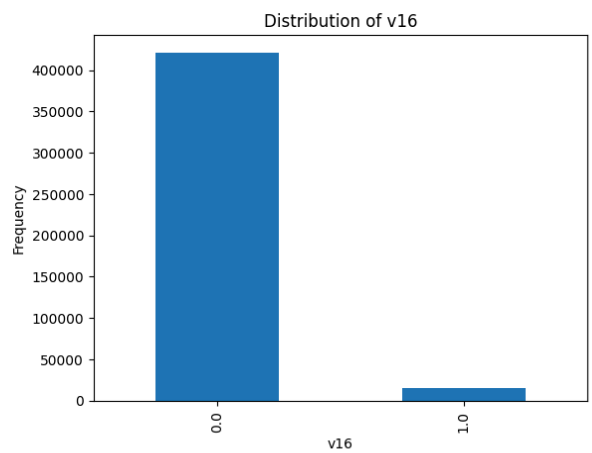


Fig 1: Plotting the number of Numerical and Categorical column

Fig 2 : Plotting the number of category and there values of **Target column**

The target column has **184700** missing columns. Since it was too much, we did not use resampling & other techniques. We **dropped** those rows.

- ☐ **Data Cleaning:** We handled missing data, outliers, skewness of data and inconsistencies of the dataset.

We identified several columns containing missing values. The red line indicates columns with more than 80% missing values in the original dataset, so we dropped those columns. Additionally, we reviewed the description of columns with null values.

Fig: Missing Values per Column

Columns	Missing_Values
v22	215685
v23	364241
v27	189
v39	6090
v42	21
v44	316088
v47	436156
v48	436157
v51	2642
v52	2642
v53	427224
v54	427224
v55	427224
v56	427224
v57	427478
v58	427224
v59	428388
v60	427224
v61	427224
v62	427224
v63	427224
v64	2642
v65	427224
v66	427224
v67	427224

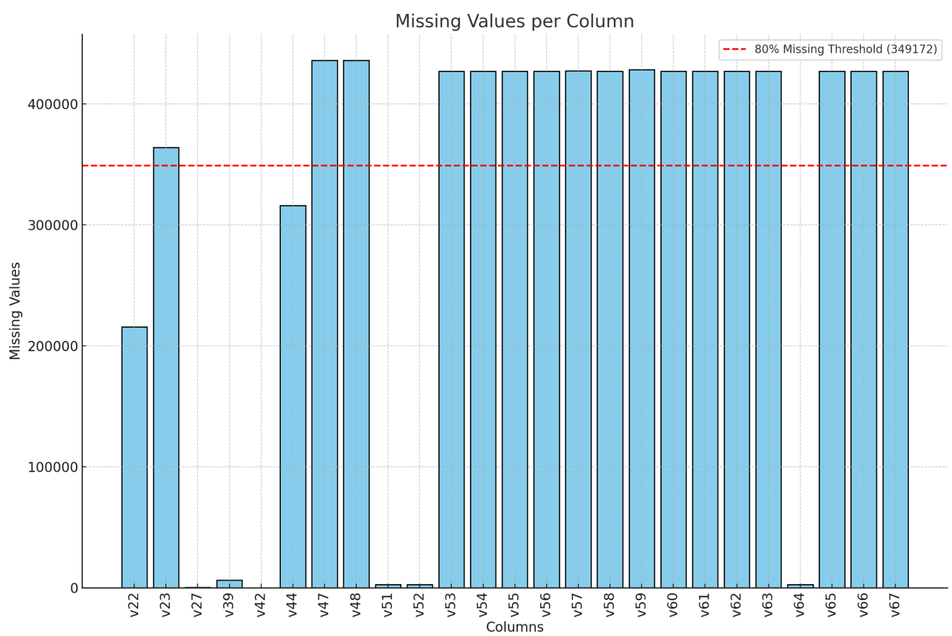


Fig: Details of columns with **less** missing data.

	missing	non-null	total	dtype	unique
v27	189	436276	436465	float64	1208
v39	6090	430375	436465	object	8
v42	21	436444	436465	object	31
v51	2642	433823	436465	float64	7945
v52	2642	433823	436465	float64	234211
v64	2642	433823	436465	float64	11817

We handled these columns with less missing values by first analyzing the skewness and correlation, and then applying the mean, mode, or KNNImputer where necessary.

Fig: Plot all the columns with categorical values in a grid with 4 columns

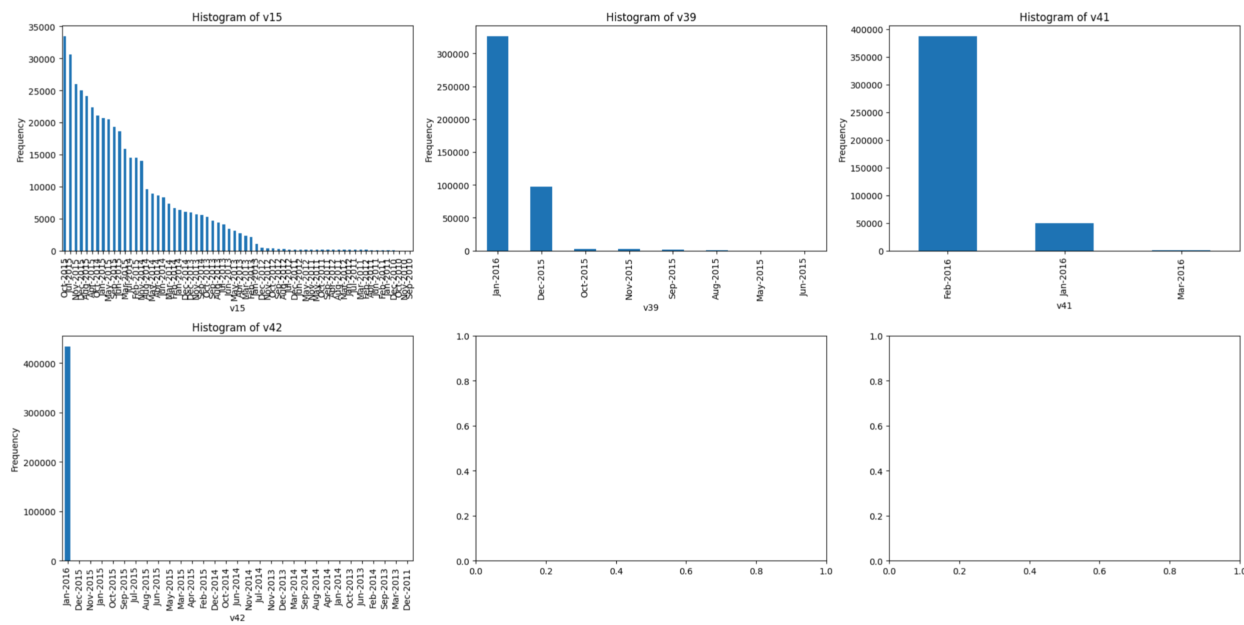
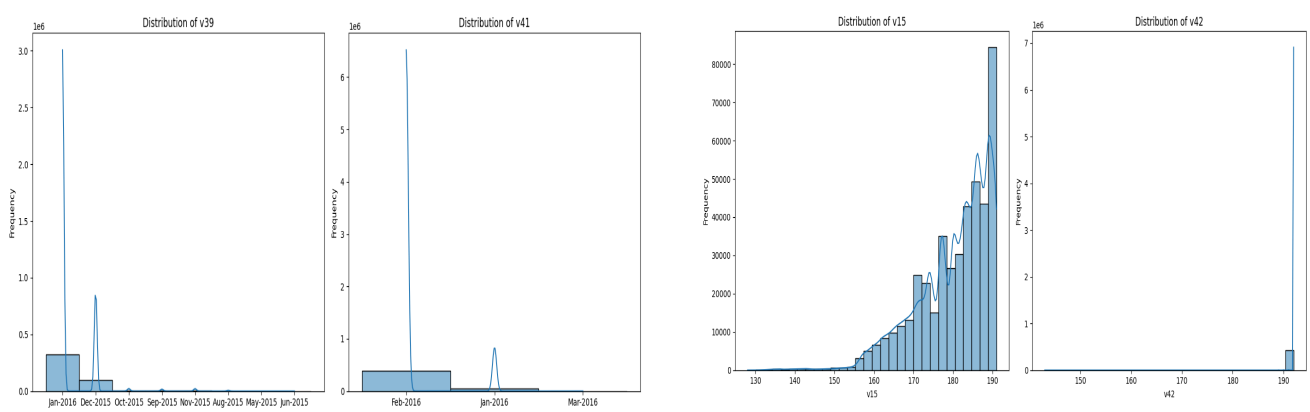


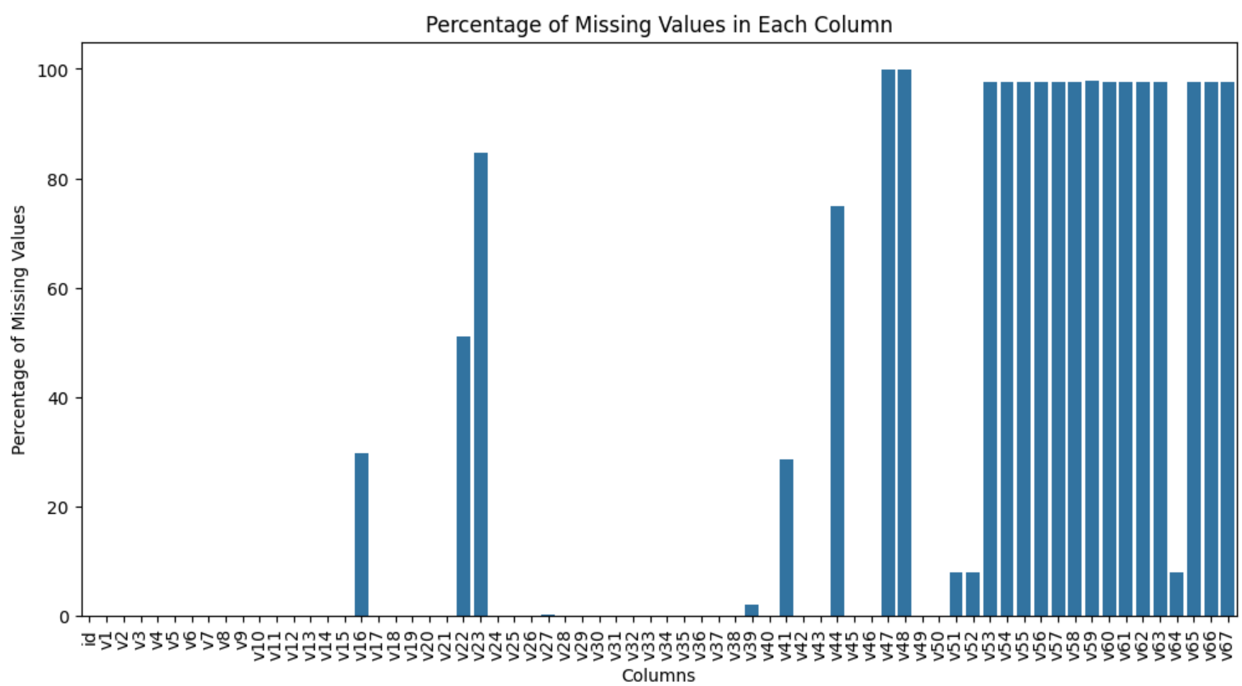
Fig: Data Distribution of those categorical column

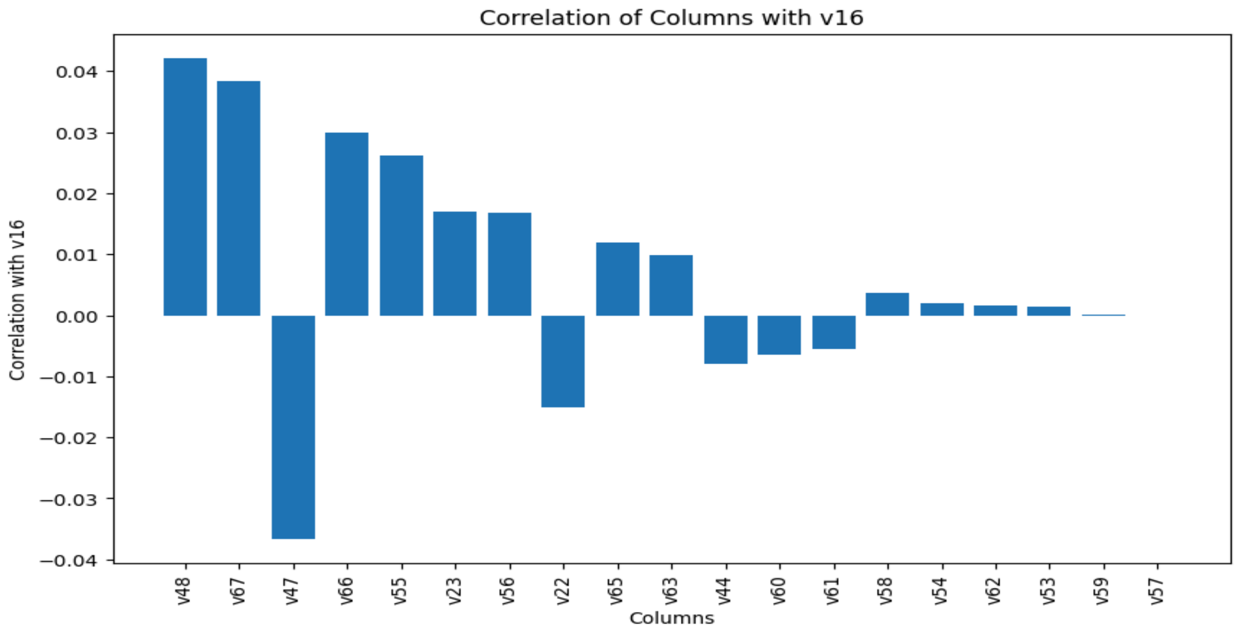


We dropped columns with more than 50,000 missing values, planning to re-investigate these columns later to assess their impact on the target column. For categorical columns with missing values, we filled in the missing entries with the **most frequent value** for each column. We also analyzed the correlation between columns with highly missing values and the target column.

Figure : Details of those columns with high missing value count.

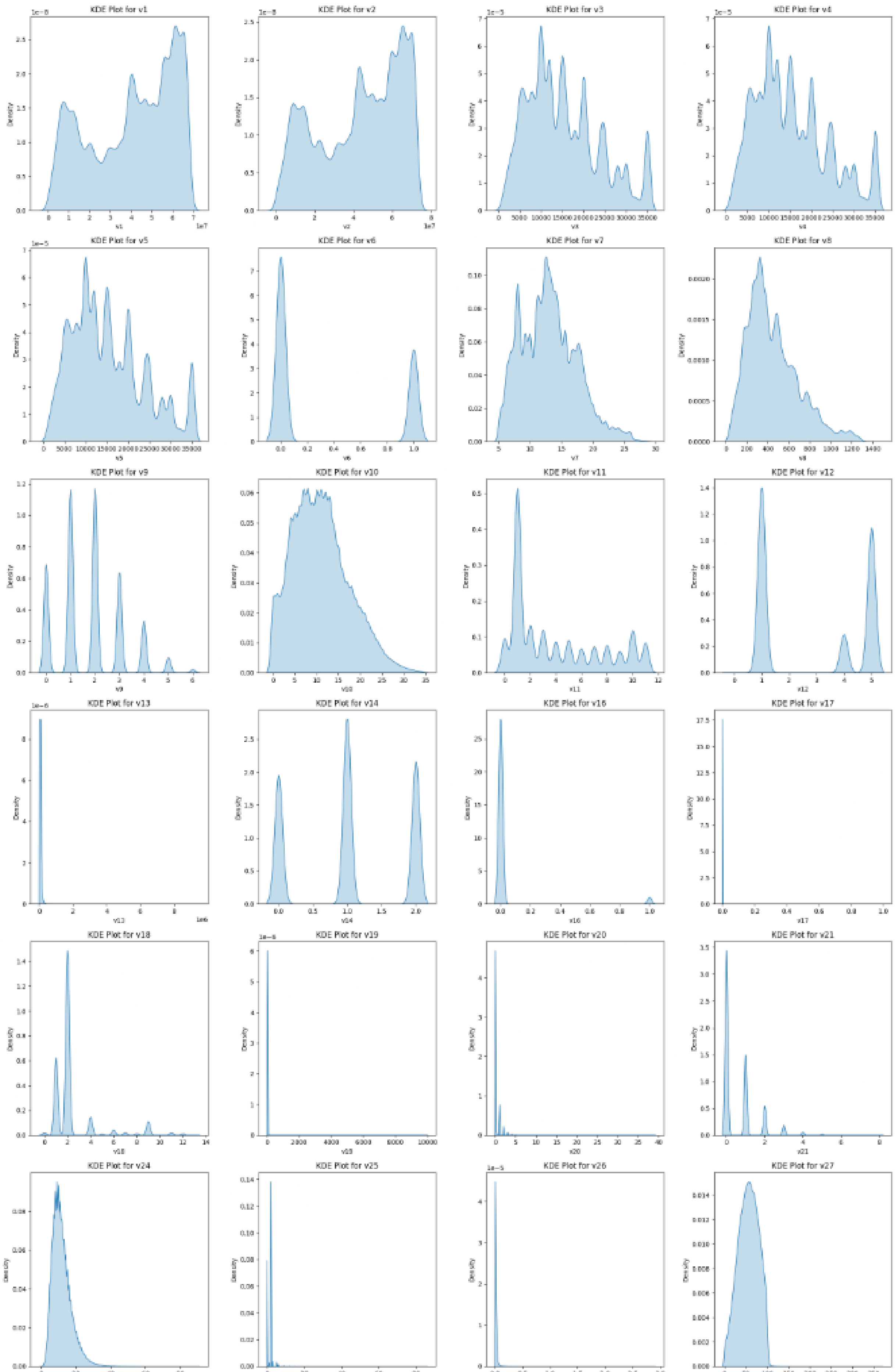
Column	Missing	Non-Null	Total	Dtype	Unique
v22	215685	220780	436465	float64	145
v23	364241	72224	436465	float64	121
v44	316088	120377	436465	float64	165
v47	436156	309	436465	float64	230
v48	436157	308	436465	float64	279
v53	427224	9241	436465	float64	10
v54	427224	9241	436465	float64	32
v55	427224	9241	436465	float64	11
v56	427224	9241	436465	float64	16
v57	427478	8987	436465	float64	179
v58	427224	9241	436465	float64	7888
v59	428388	8077	436465	float64	1129
v60	427224	9241	436465	float64	14
v61	427224	9241	436465	float64	24
v62	427224	9241	436465	float64	6428
v63	427224	9241	436465	float64	1021
v65	427224	9241	436465	float64	16
v66	427224	9241	436465	float64	26
v67	427224	9241	436465	float64	24

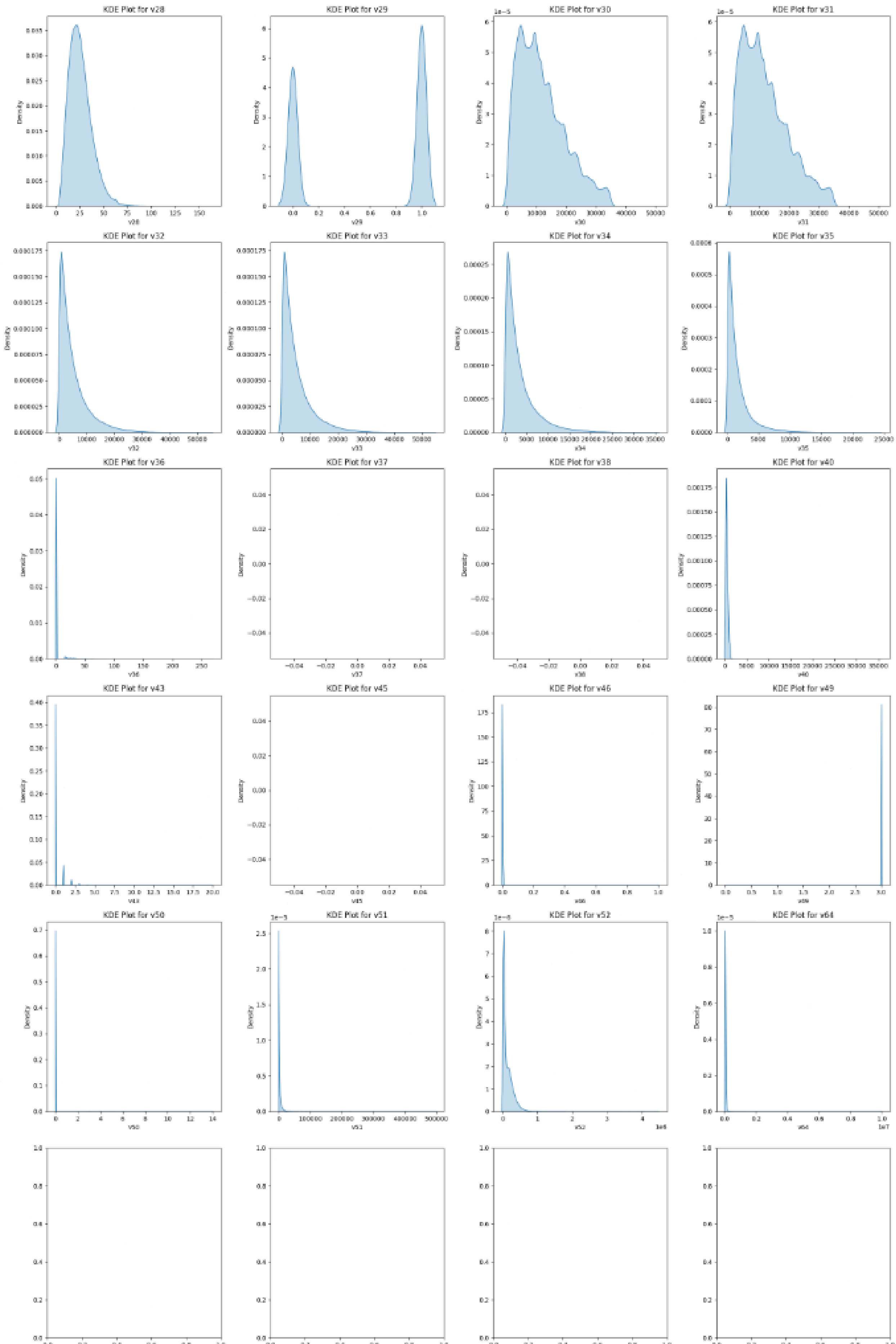




Finally, we selected relevant columns from these highly missing value columns & applied mean or mode to fill them up.

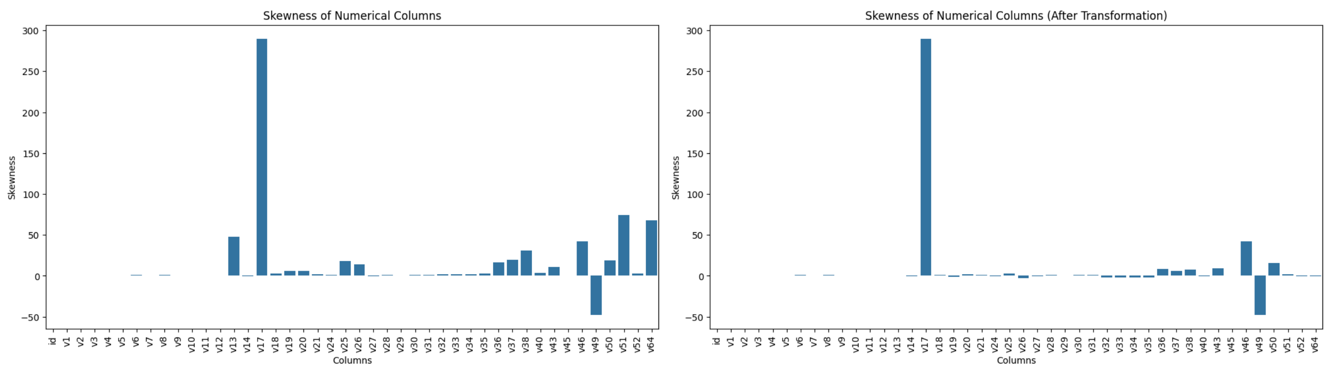
Fig: Plot KDE for numerical columns in a grid of 4 columns


KDE Plot for v1
KDE Plot for v2
KDE Plot for v3
KDE Plot for v4
KDE Plot for v5
KDE Plot for v6
KDE Plot for v7
KDE Plot for v8
KDE Plot for v9
KDE Plot for v10
KDE Plot for v11
KDE Plot for v12
KDE Plot for v13
KDE Plot for v14
KDE Plot for v16
KDE Plot for v17
KDE Plot for v18
KDE Plot for v19
KDE Plot for v20
KDE Plot for v21
KDE Plot for v24
KDE Plot for v25
KDE Plot for v26
KDE Plot for v27


KDE Plot for v28
KDE Plot for v29
KDE Plot for v30
KDE Plot for v31
KDE Plot for v32
KDE Plot for v33
KDE Plot for v34
KDE Plot for v35
KDE Plot for v36
KDE Plot for v37
KDE Plot for v38
KDE Plot for v40
KDE Plot for v43
KDE Plot for v45
KDE Plot for v46
KDE Plot for v49
KDE Plot for v50
KDE Plot for v51
KDE Plot for v52
KDE Plot for v64

Plot KDEs for numerical columns in a grid with 4 columns to show a smoothed view of data density for each variable. Many KDE plots have high peaks, which show where most data values are concentrated. Some plots look like normal distributions with a single, symmetrical peak, while others are skewed, showing uneven data concentration. Several KDEs have multiple peaks, suggesting there may be different clusters or groups within certain variables. The height and width of the peaks indicate how concentrated or spread out the data is, with taller peaks showing higher concentration around specific values.

Fig: Skewness of Numerical column (**before** & **after**) which have less than 20% missing values



Prepare Data for Modeling

- ☐ **Encode Categorical Data:**

We encoded the categorical columns using One-Hot Encoding (OHE) on columns ‘v39’ and ‘v41’. In these two columns, some categories have a large number of occurrences, while others have fewer. To address this, we first defined a threshold based on the value counts of these columns. Categories with fewer occurrences than the threshold were grouped into a single category called ‘other’. Afterward, we performed OHE and used the `drop_first` parameter to avoid the multicollinearity problem

- ☐ **Feature Selection:**

Now, The dataset contains **66** features. Using these features, we calculated their correlation with the target column. Features with a more correlation with the target feature were selected for the next step. We then applied **SelectKBest** with **chi2** to select the best features from **47** columns to **40** columns.

- ☐ **Feature Scaling:**

  We used **Standard Scalar** to scale down all the features.

Split the Data into Training and Testing Sets:

We performed an 80:20 split, where 80% of the data was allocated to the training set and 20% to the testing set.

Select, Train & Evaluate the Model:

For this binary classification task, we trained several models, including **Random Forest**, **Logistic Regression**, **XGBoost**, **CatBoost**, **Gaussian Naive Bayes (GNB)**, **Multinomial Naive Bayes (MNB)**, and **Decision Tree**. Our focus was to maximize the F1 score for class 1.

For this binary **classification task**, we found that the **CatBoostClassifier** model achieved the best results, with an overall accuracy of **98.4%** & with f1 score **0.6949** for class 1. Below is a detailed breakdown of the performance metrics:

Metric	Precision	Recall	F1-Score	Support
Class 0	0.98	1.00	0.99	84,389
Class 1	0.96	0.54	0.69	2,904
Accuracy			0.984	87,293
Macro Average	0.97	0.77	0.84	87,293
Weighted Average	0.98	0.98	0.98	87,293