Santander Customer Transaction Prediction

By

Kiran Kumar

18 June 2019

Contents

Chapter 1

Introduction

1.1 Background:

At Santander, mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as:

- is a customer satisfied?
- Will a customer buy this product?
- Can a customer pay this loan?

1.2 Problem Statement:

We need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

- Supervised: The labels are included in the training data and the goal is to train a model to learn to predict the labels from the features
- Classification: The label is a binary variable, 0 (will not make a specific transaction in the future), 1 (will make a specific transaction in the future)

1.3 Data:

The details of data attributes in the dataset are as follows -

- ID_code (string)
- Target (0 or 1)
- 200 numerical variables, named from var_0 to var_199

Given below is a sample of the data set that we are using to predict customer transactions:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_190	var_191	var_192	var_193	var_194	var_195	var_196
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.9642	3.1364	1.6910	18.5227	-2.3978	7.8784
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6421	7.7214	2.5837	10.9516	15.4305	2.0339	8.1267
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9057	9.7905	1.6704	1.6858	21.6042	3.1417	-6.5213
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4666	4.7433	0.7178	1.4214	23.0347	-1.2706	-2.9275
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4905	9.5214	-0.1508	9.1942	13.2876	-1.5121	3.9267

5 rows × 202 columns

As you can see from the info below we have 200 continuous variables (var_0 to var_199), using which we have to correctly predict whether customer will make a specific transaction in the future or not:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 202 entries, ID_code to var_199
dtypes: float64(200), int64(1), object(1)
memory usage: 308.2+ MB
```

Chapter 2

Methodology

2.1 Pre Processing

We begin by exploring the data, cleaning the data as well as visualizing the data through graphs and plots, which is often called as **Exploratory Data Analysis (EDA).**

To start this process we will first get a quick glance on the distributions of the variables. Most ML algorithms require the data to be normally distributed. We can visualize that in a glance by using **pandas_profiling** library to generate profile reports.

For each column the following statistics are presented in an interactive HTML report:

- Essentials: type, unique values, missing values
- Quantile statistics like minimum value, Q1, median, Q3, maximum, range, interquartile range
- Descriptive statistics like mean, mode, standard deviation, sum, median absolute deviation, coefficient of variation, kurtosis, skewness
- Most frequent values
- Histogram
- Correlations highlighting of highly correlated variables, Spearman and Pearson matrixes

We can access the statistics of each variable generated in the HTML report via our saved repository.

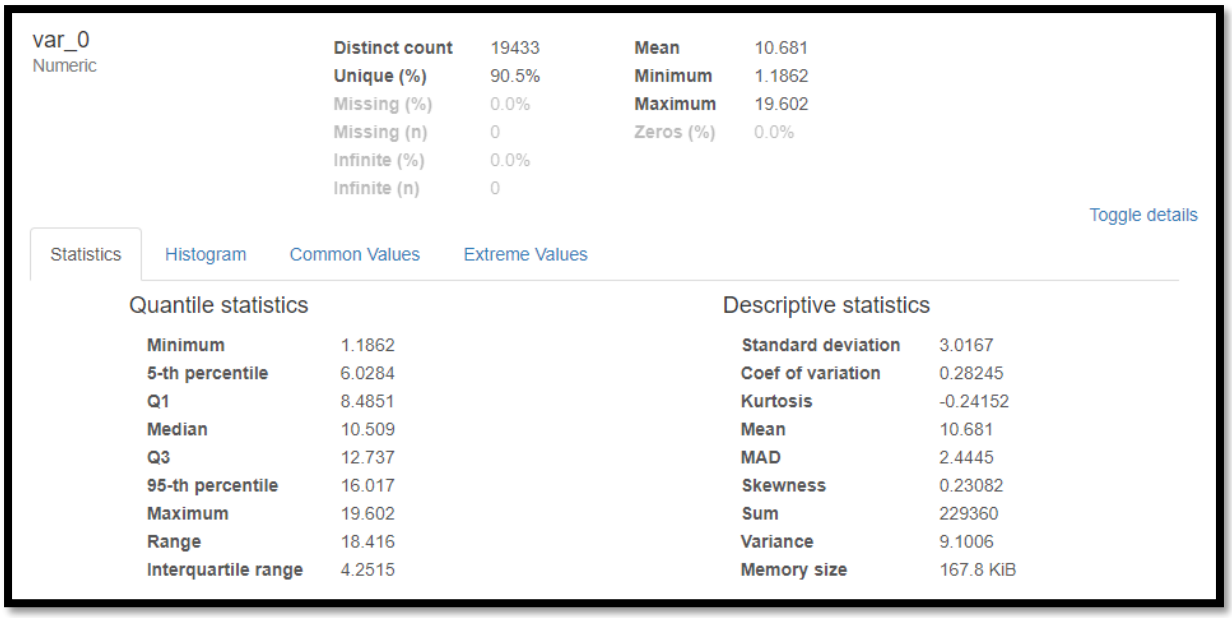


Fig 2.1 : Statistics of var_0 variable

For example, we have taken the var_0 variable here, in Fig 2.1 it shows a **Maximum value of 19.6** in Quantile Statistics.

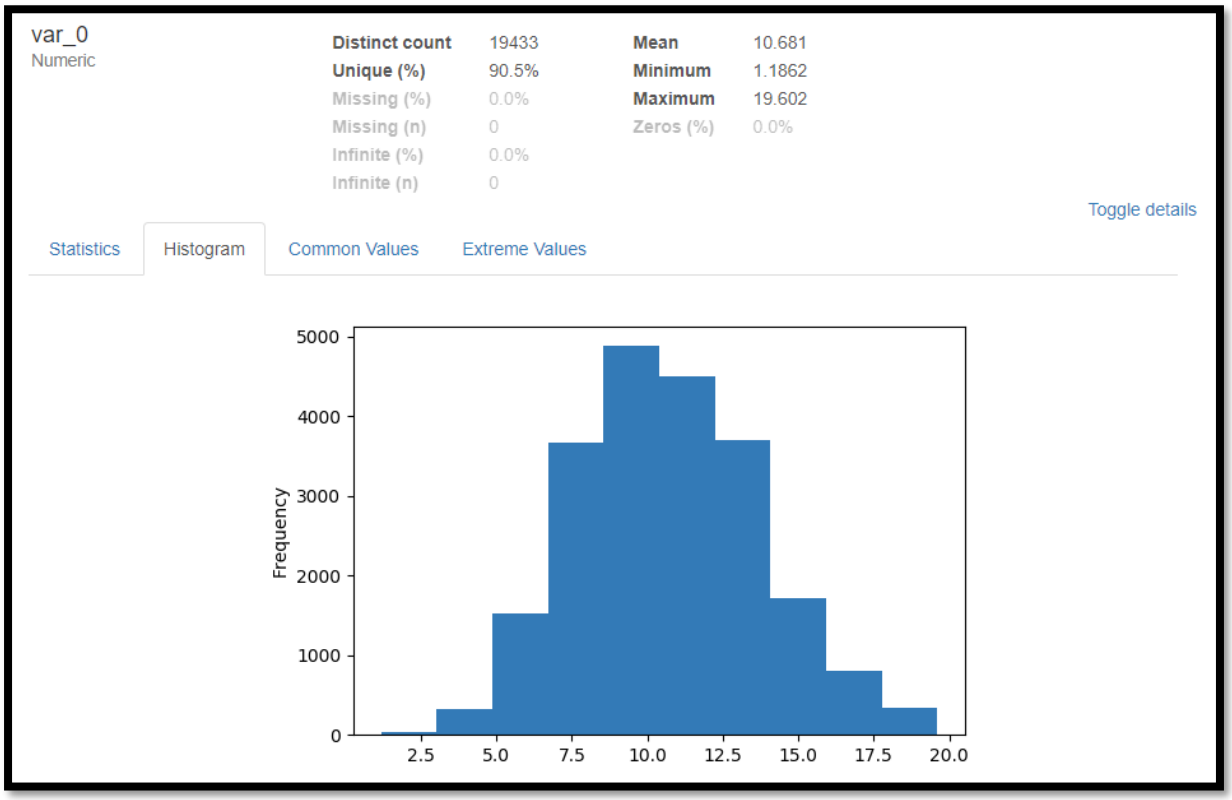


Fig 2.2 : Histogram of var_0 variable

The distribution of var_0 variable seems to be normally distributed. Other features like common values and Extreme values can also be accessed via toggle details option in HTML report.

2.1.1 Missing Value Analysis

- Visualisation of missing values done using the **missingno** library.
- No missing value were found.

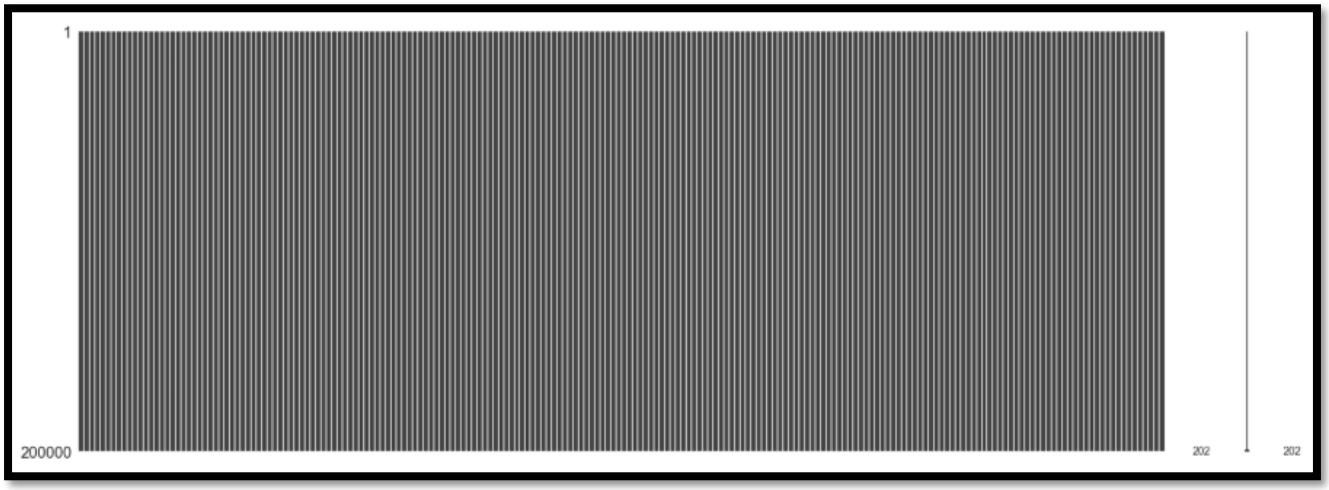


Fig 2.3 : Visualisation of missing values

2.1.2 Data Visualization

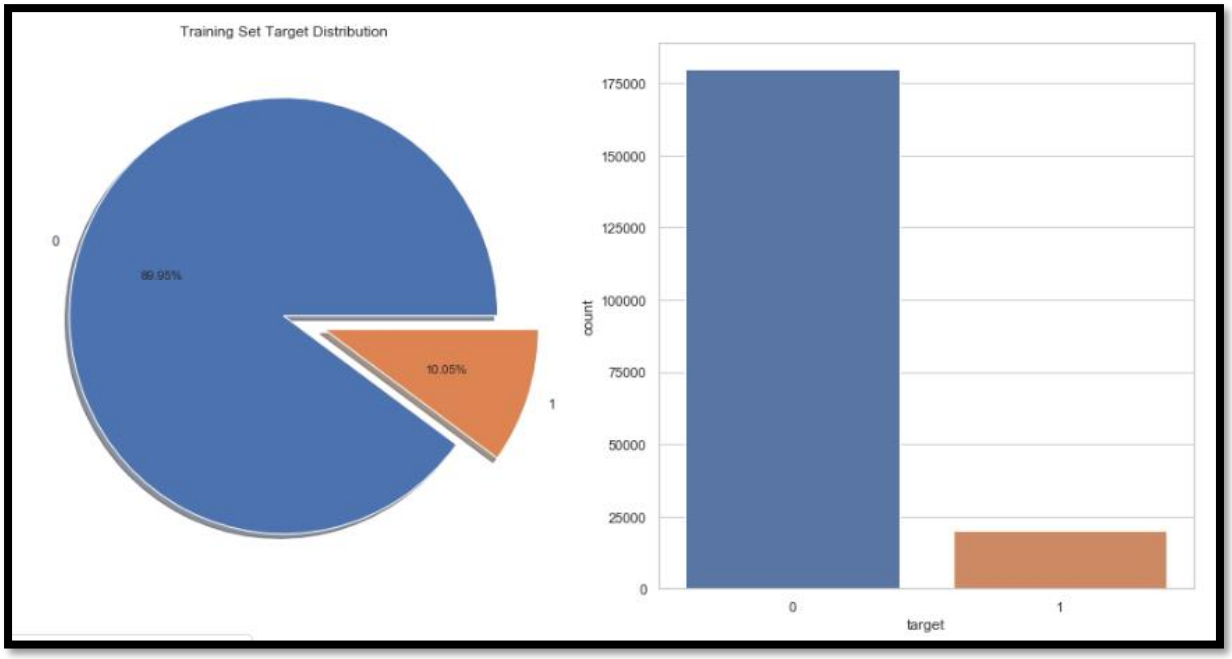


Fig 2.4 : Target Variable Distribution

- Number of transactions happening account to approx. 10% in the train dataset.
- The dataset is unbalanced with respect to the target, need to consider resampling methods.

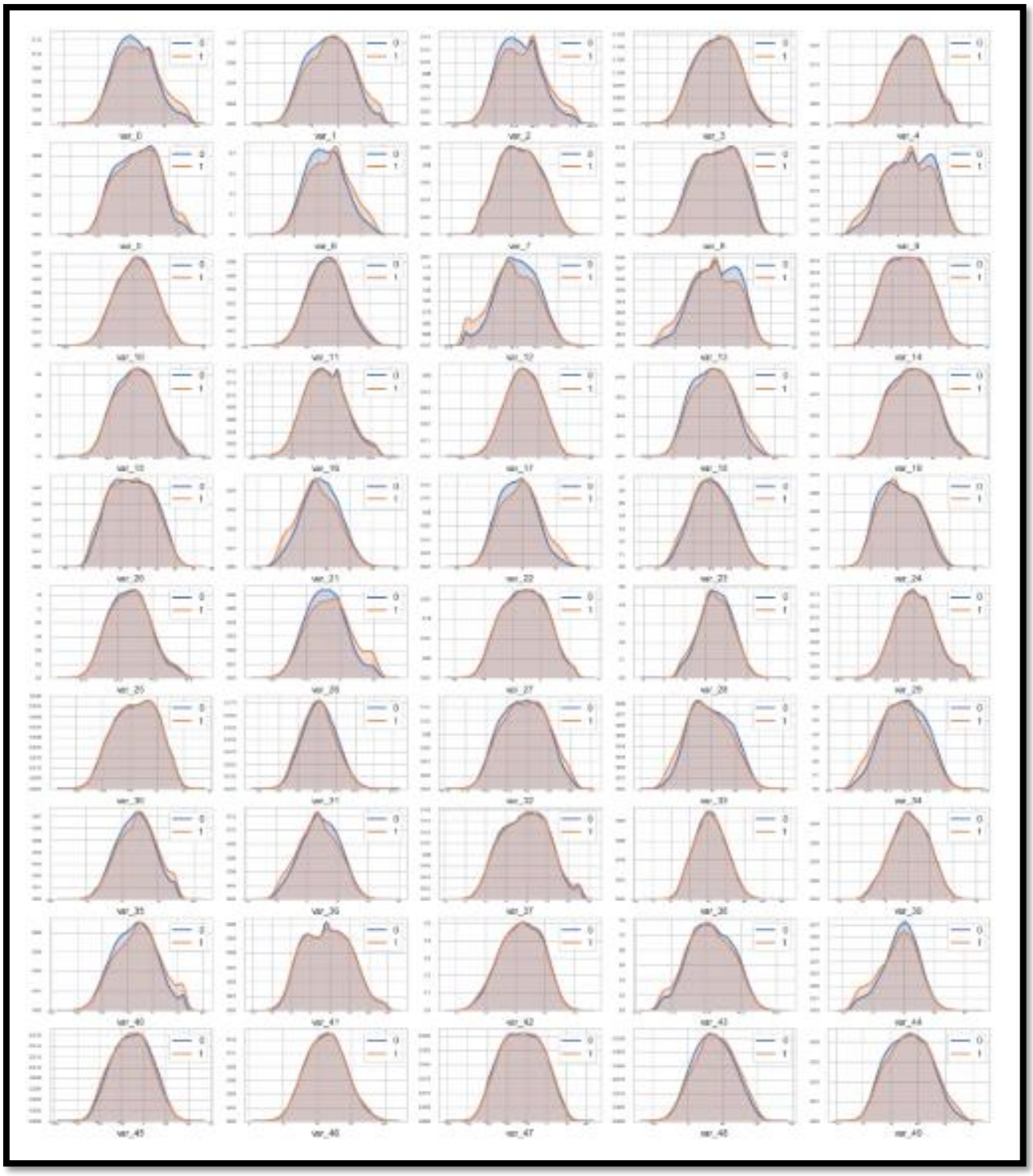


Fig 2.5 : KDE plots from var_0 to var_49 with respect to target variable(0 and 1)

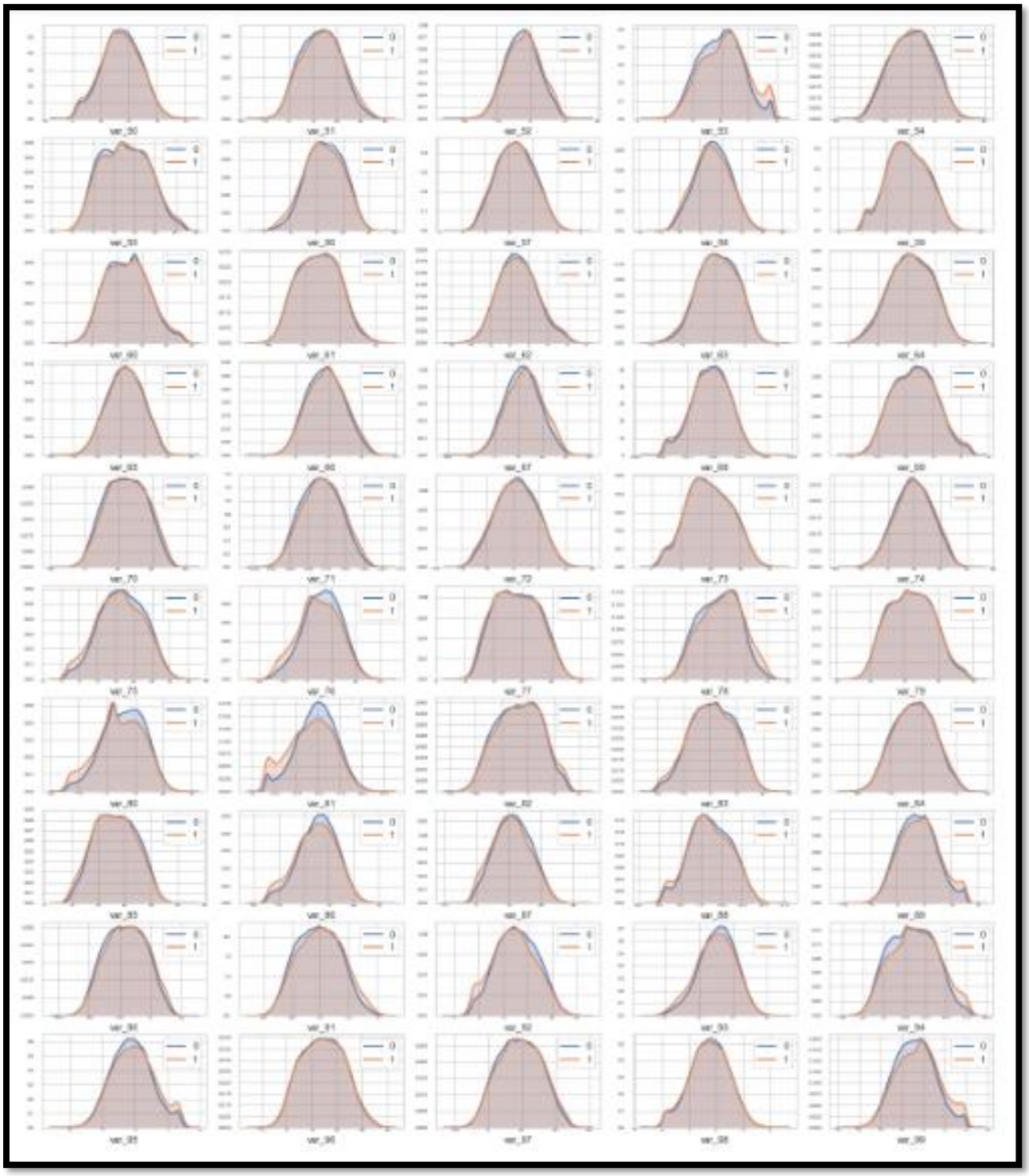


Fig 2.6 : KDE plots from var_50 to var_99 with respect to target variable(0 and 1)

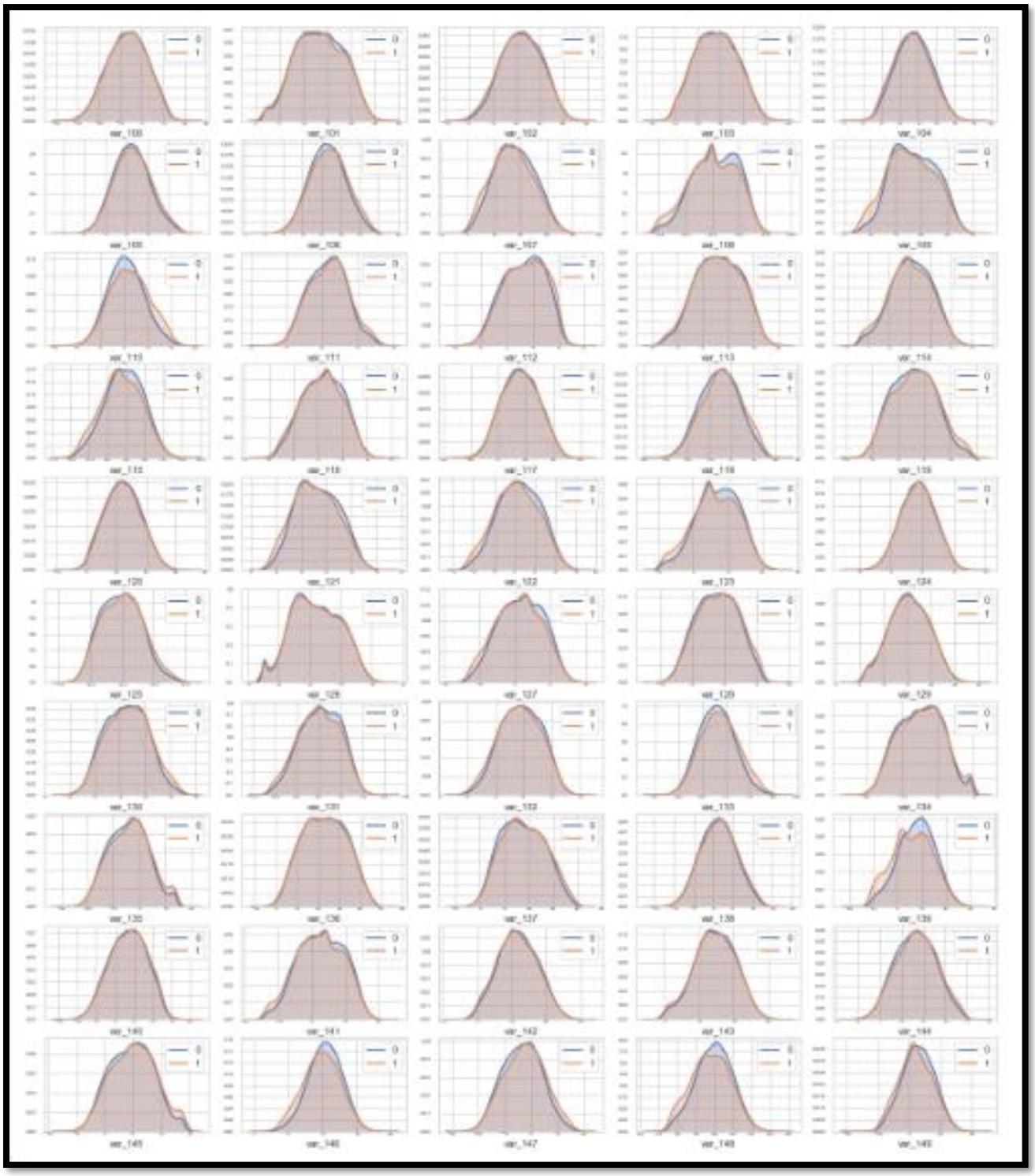


Fig 2.7 : KDE plots from var_100 to var_149 with respect to target variable(0 and 1)

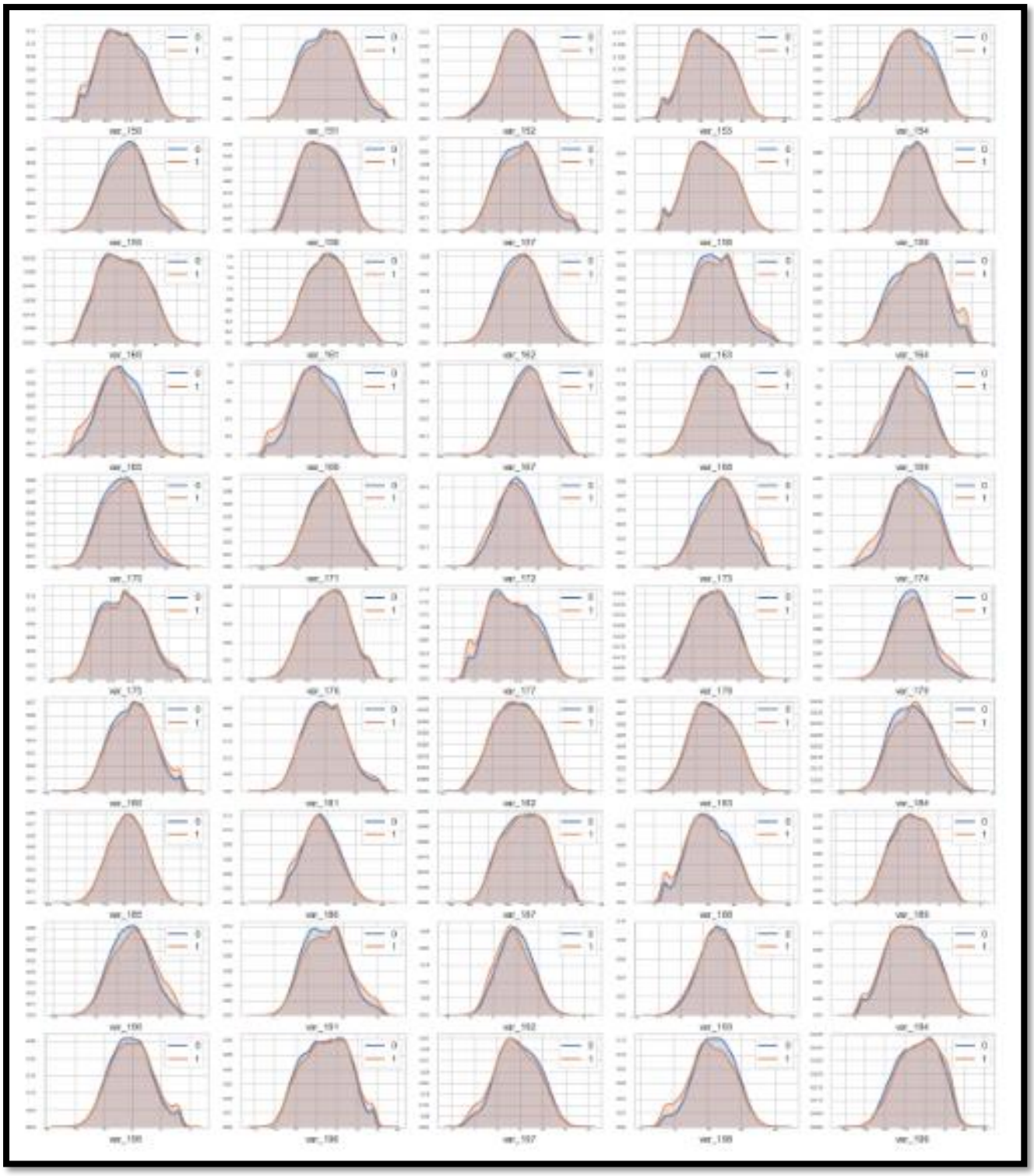


Fig 2.8 : KDE plots from var_150 to var_199 with respect to target variable(0 and 1)

- if we look closely var_2, var_9, var_12, var_13, var_26, var_40, var_53, var_81 and many others have **resemblance of a bi-modal type distribution**.
- All of these variables have a bump of frequency that matches the rising of the probability of making a transaction.
- if pdf(target = 1) - pdf(target = 0) > 0, then there is a high probability of the client making a transfer.

2.1.3 Outlier Analysis

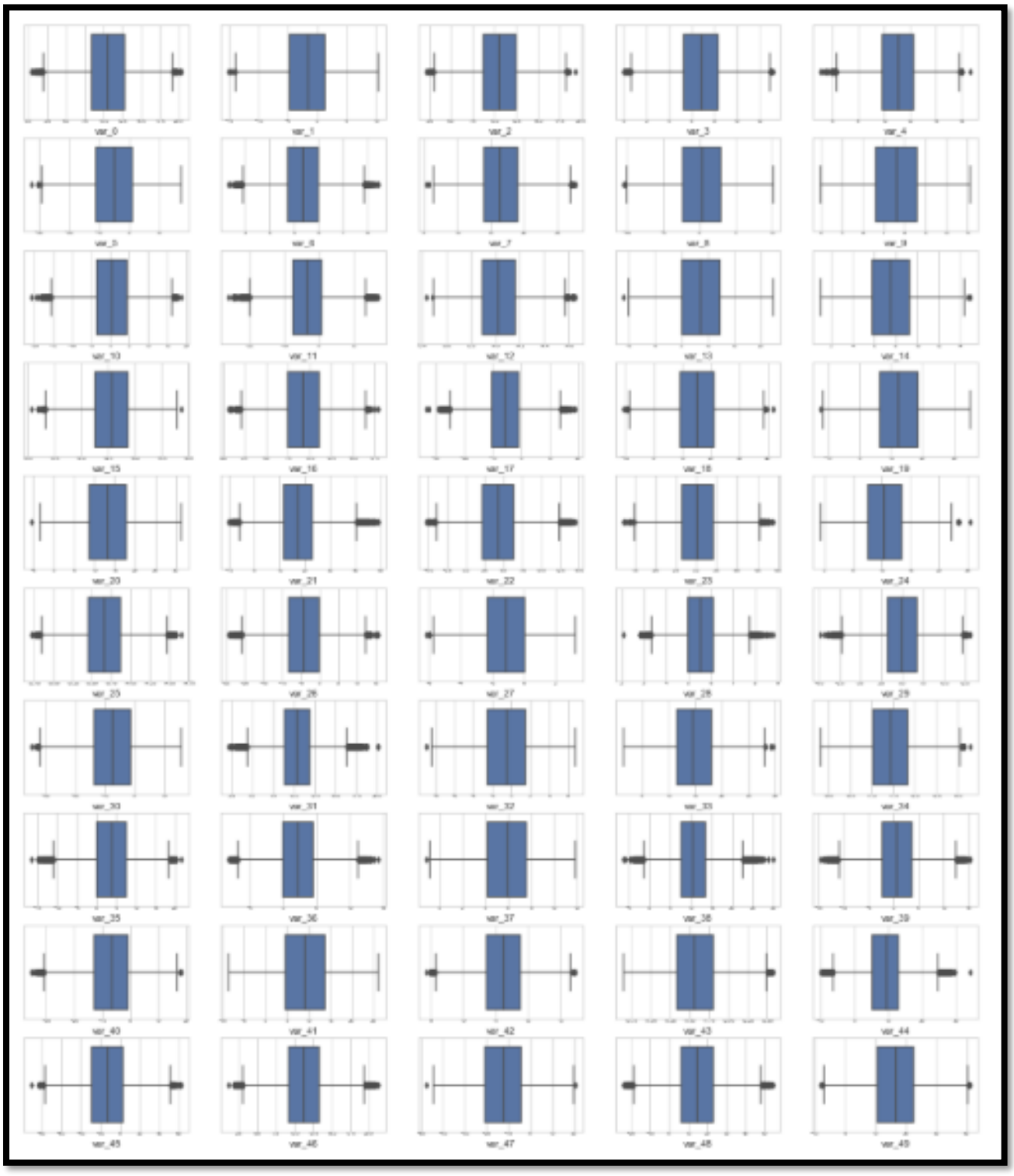


Fig 2.9 : Box plots from var_0 to var_49

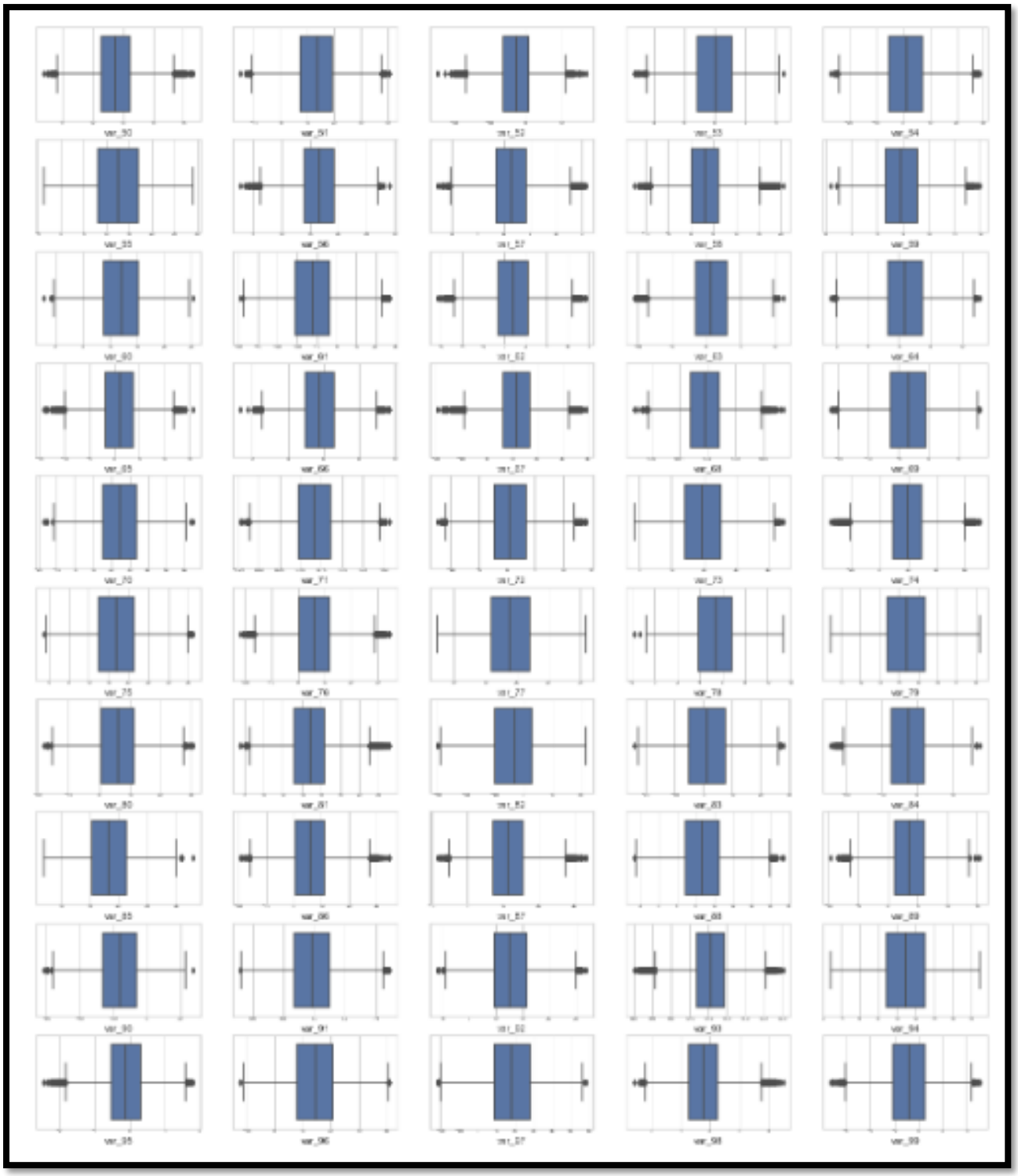


Fig 2.10 : Box plots from var_50 to var_99

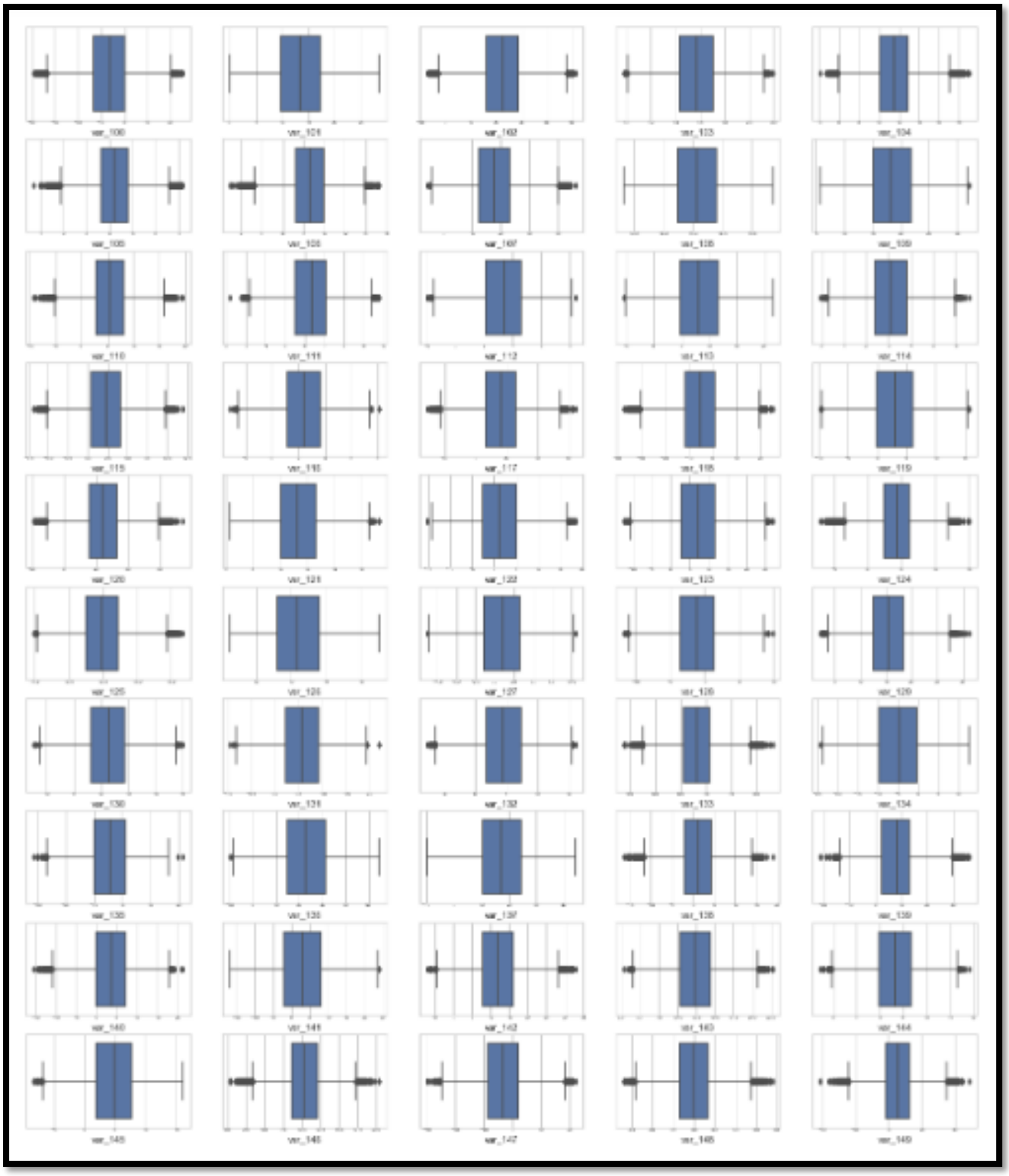


Fig 2.11 : Box plots from var_100 to var_149

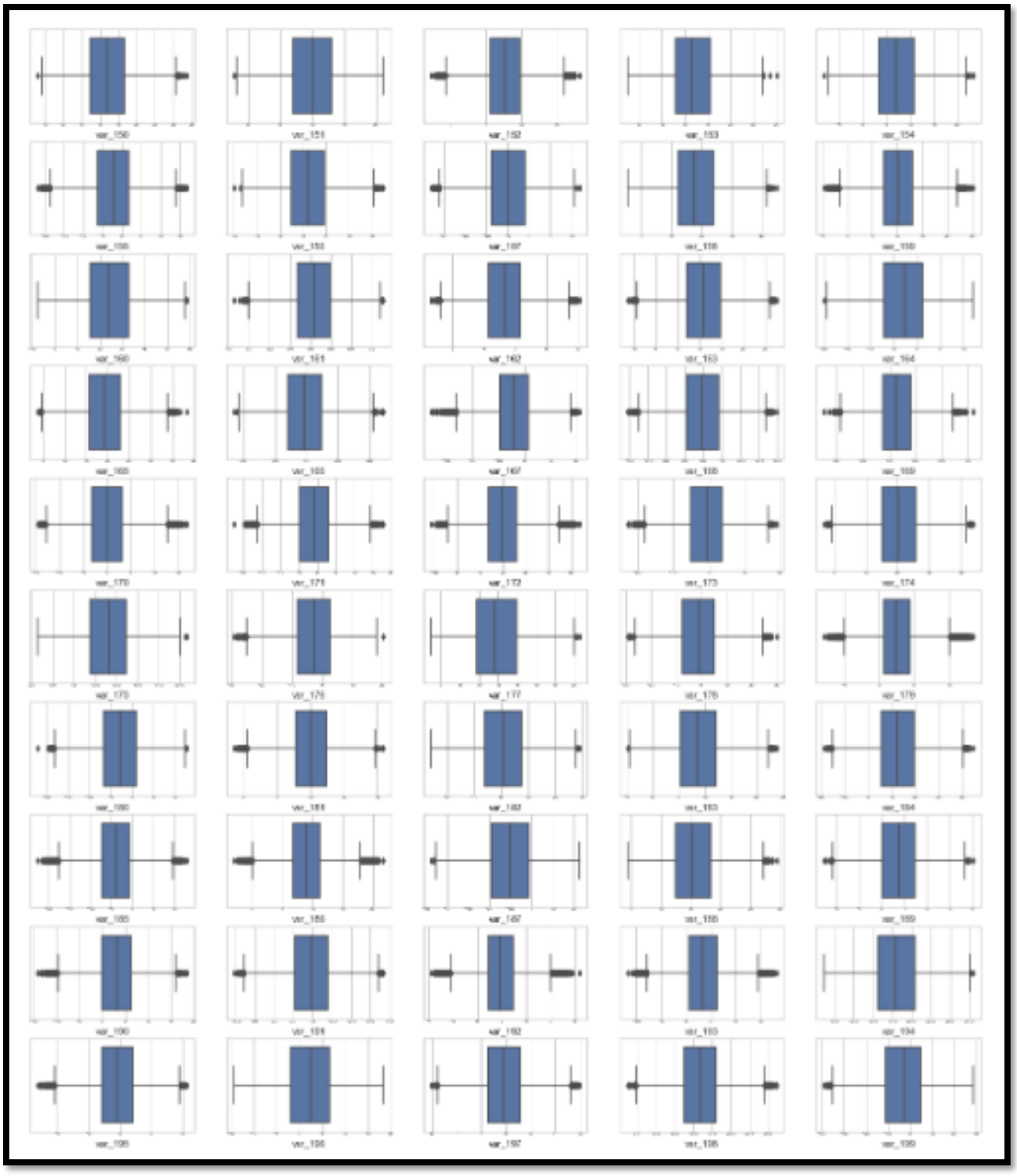


Fig 2.12 : Box plots from var_150 to var_199

- Almost all variables have outliers present from the box plots above.

- After separating outliers and inliers with IQR method we found that all the target variables with label as one are outliers.
- Outliers present in our data, are meaningful and thus can't be removed.

```
print("df.shape:",train.shape)
df_in = train[~((train < (Q1 - 1.5 * IQR)) |(train > (Q3 + 1.5 * IQR))).any(axis=1)]
df_out = train[((train < (Q1 - 1.5 * IQR)) |(train > (Q3 + 1.5 * IQR))).any(axis=1)]
print("df_in.shape:",df_in.shape)
print("df_out.shape:",df_out.shape)
```

```
df.shape: (200000, 202)
df_in.shape: (157999, 202)
df_out.shape: (42001, 202)
```

```
df_in['target'].value_counts()
```

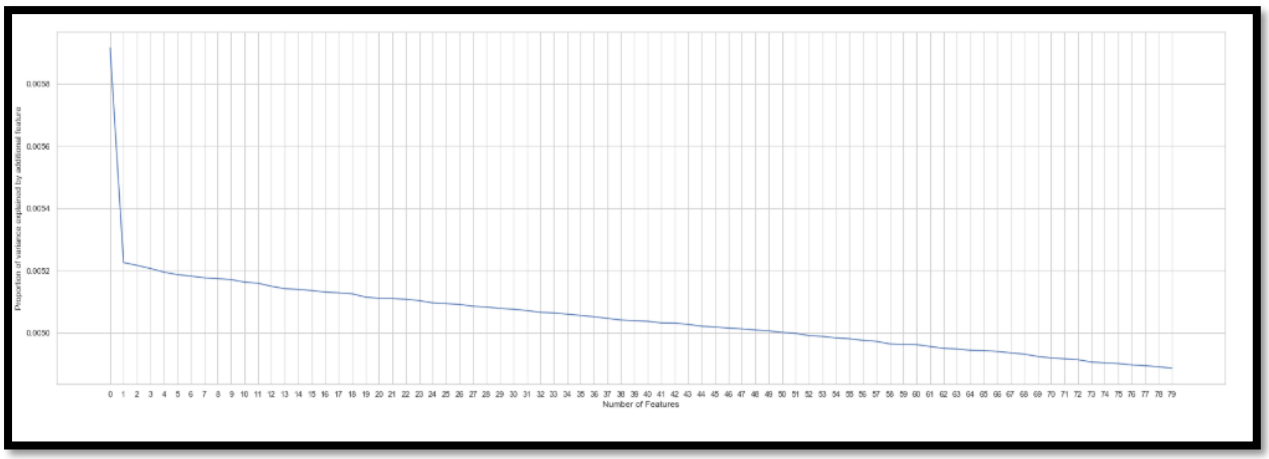
```
0    157999
Name: target, dtype: int64
```

```
# comparing the 'train' and 'df_out' dataset,
# we can say that all the data points with target equals to 1 are present as outliers
df_out['target'].value_counts()
```

```
0    21903
1    20098
Name: target, dtype: int64
```

2.1.4 Principal component analysis (PCA)

- PCA is a dimensionality reduction technique that reduces less-informative 'noise' features.
- But PCA is sensitive to variance and different scales, so standardizing will help PCA perform better.
- However, since we found that the correlation between different features in the training dataset is not that significant, so using PCA might not be meaningful.



2.1.5 Correlation Analysis

- The features are independent and not correlated to each other

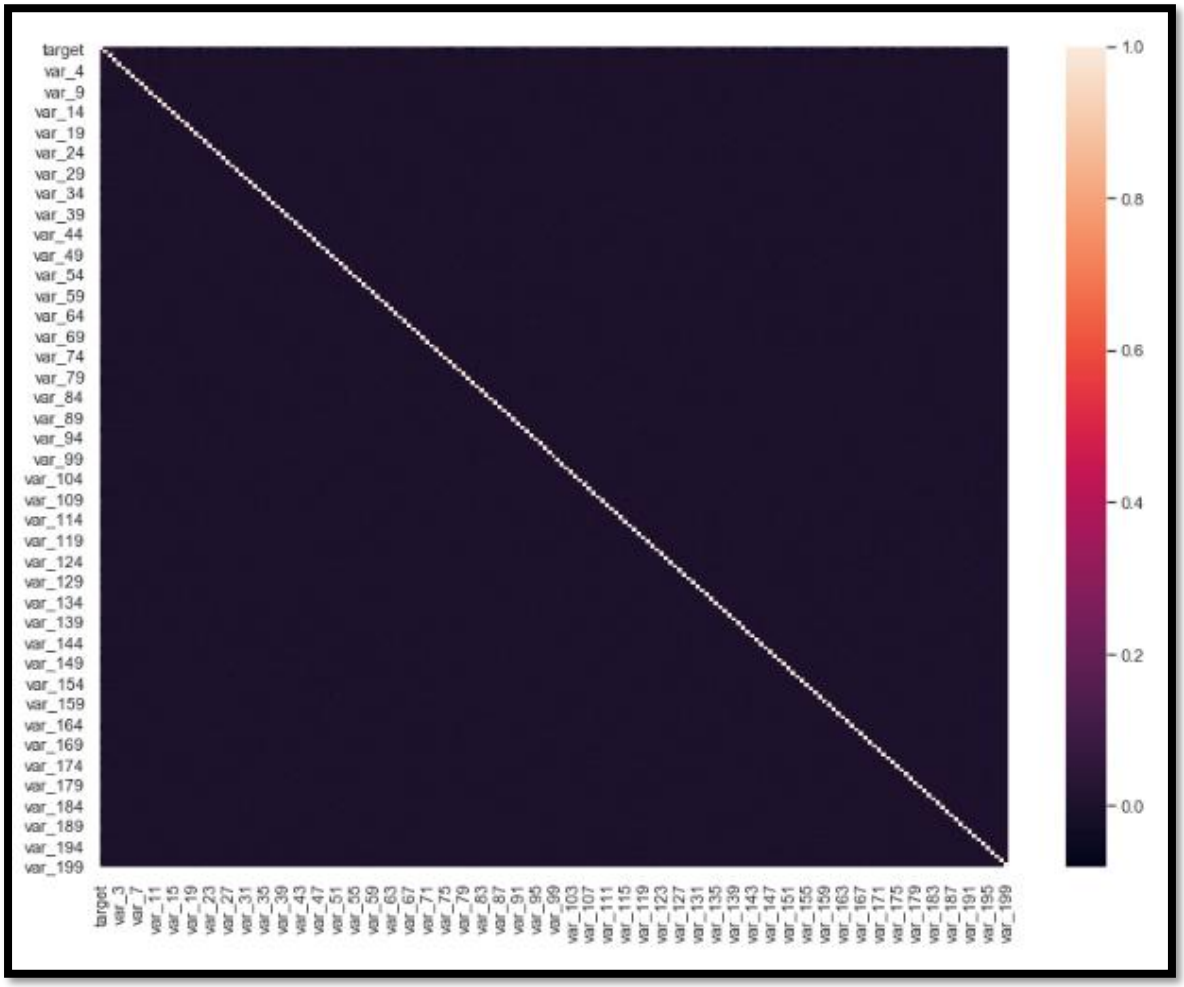


Fig 2.15 : Correlation Analysis of all features using HeatMap

2.1.6 Feature Engineering

```
#Created new columns with the unique values count
for feat in ['var_' + str(x) for x in range(200)]:
    train_count_values = train.groupby(feat)[feat].count()
    test_count_values = test.groupby(feat)[feat].count()
    train['new_' + feat] = train_count_values.loc[train[feat]].values
    test['new_' + feat] = test_count_values.loc[test[feat]].values
```

```
train.head(3)
```

var_6	var_7	...	new_var_190	new_var_191	new_var_192	new_var_193	new_var_194	new_var_195	new_var_196	new_var_197	new_var_198	new_var_199
5.1187	18.6266	...	3	6	7	3	4	4	3	13	5	2
5.6208	16.5338	...	5	4	6	1	1	2	2	13	2	1
6.9427	14.6155	...	3	4	3	1	2	2	3	8	2	2

- Created new features with the counts of unique values present in each variable.

Chapter 3

Modeling

3.1 Evaluation Metric

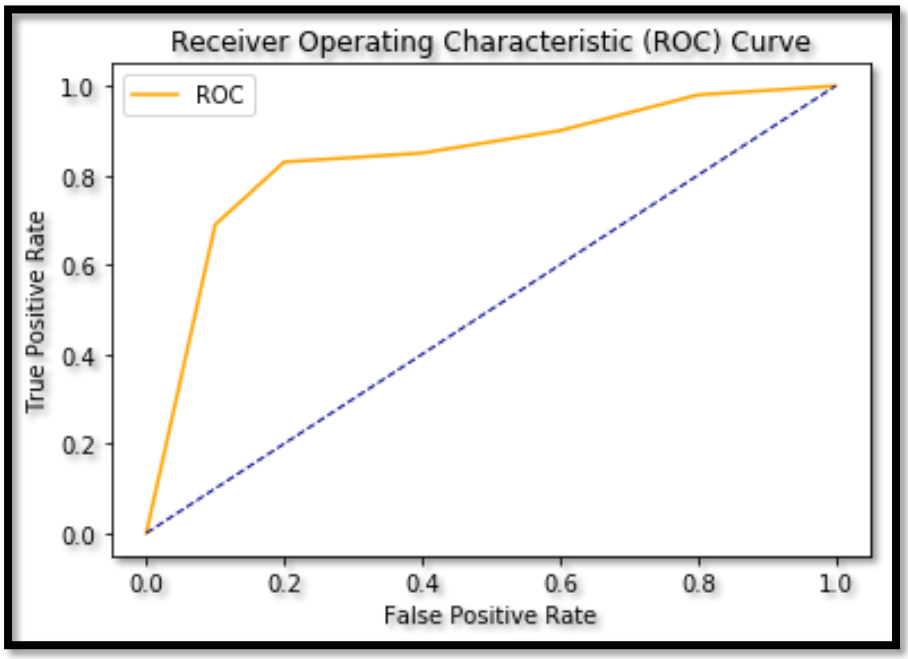
- It can be more flexible to predict probabilities of an observation belonging to each class in a classification problem rather than predicting classes directly.

- The reason for this is to provide our model the capability to choose and even calibrate the threshold for how to interpret the predicted probabilities.

- There are two diagnostic tools that help in the interpretation of probabilistic forecast for binary (two-class) classification predictive modelling problems are **ROC Curves** and **Precision-Recall** curves.

- **ROC** is a probability curve for different classes. ROC tells us how good the model is for distinguishing the given classes, in terms of the predicted probability.

- A typical ROC curve has False Positive Rate (FPR) on the X-axis and True Positive Rate (TPR) on the Y-axis.



- The area covered by the curve is the area between the orange line (ROC) and the axis. This area covered is AUC. The bigger the area covered, the better the machine learning models is at distinguishing the given classes. Ideal value for AUC is 1.

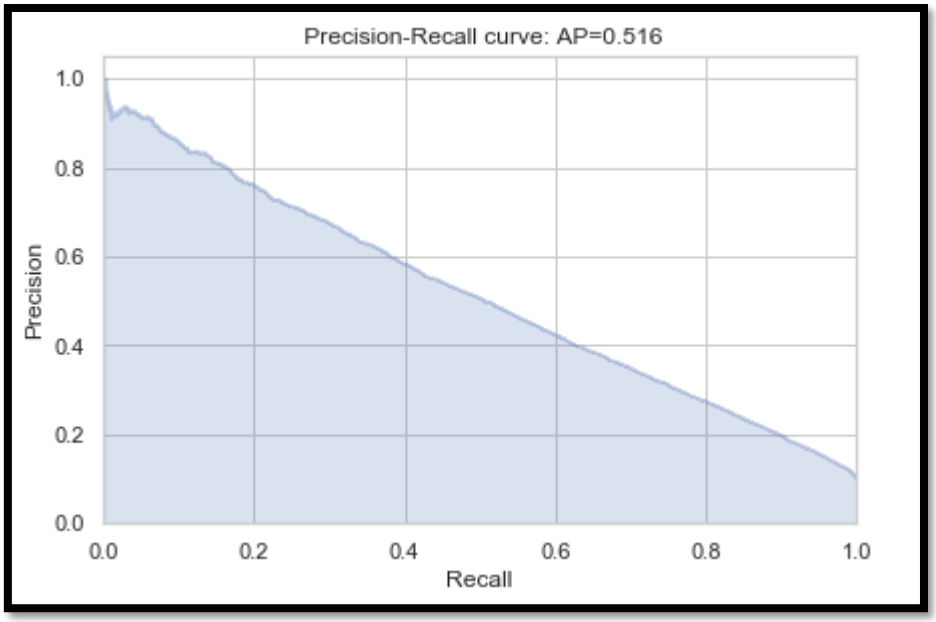
- Precision is a ratio of the number of true positives divided by the sum of the true positives and false positives. It describes how good a model is at predicting the positive class.

- Recall is calculated as the ratio of the number of true positives divided by the sum of the true positives and the false negatives. Recall is the same as sensitivity.

- **Precision-Recall** curves are useful in cases where there is an imbalance in the observations between the two classes. Specifically, there are many examples of no event (class 0) and only a few examples of an event (class 1).

- Key to the calculation of precision and recall is that the calculations do not make use of the true negatives. It is only concerned with the correct prediction of the minority class, class 1.

- A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds, much like the ROC curve.



- Hence, ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets.

3.2 Logistic Regression

- We will start our model building from the most simplest to more complex. Therefore we use Simple Logistic Regression first as our base model.
- Since this is an unbalanced dataset, we need to define parameter 'class_weight = balanced' which will give equal weights to both the targets irrespective of their representation in the training dataset.
- Pipeline concept is used to perform sequence of different transformations on our dataset before applying the final estimator.
- The precision for this model was found to be : 0.2943

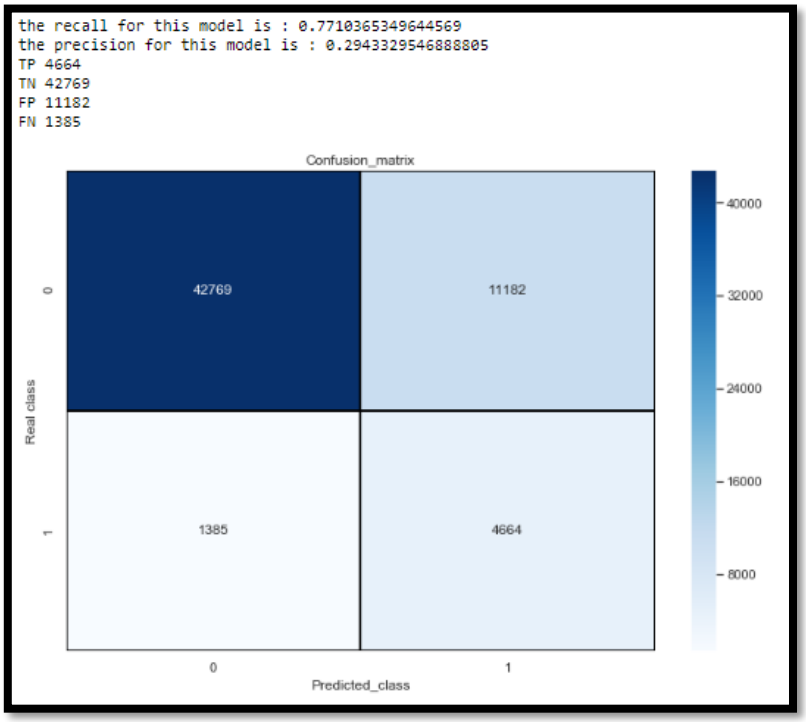


Fig 2.16 : Confusion Matrix

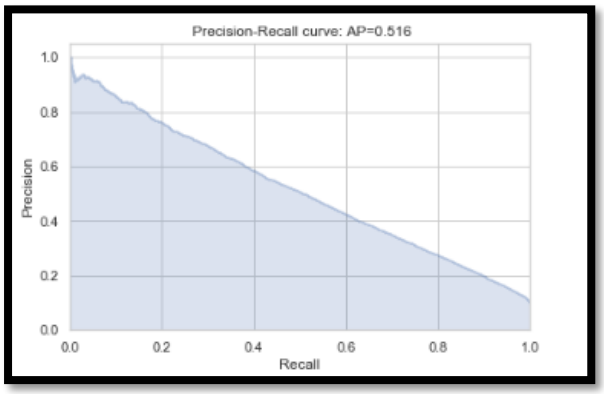


Fig 2.17 : PR curve

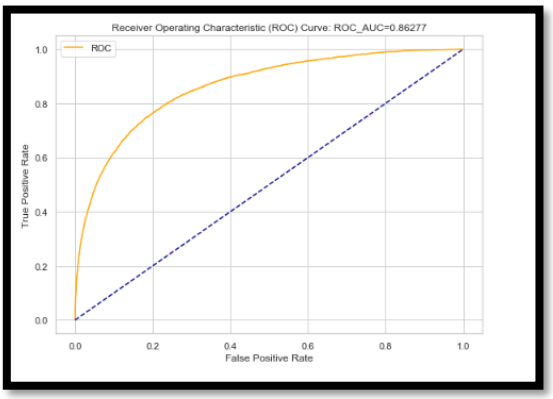
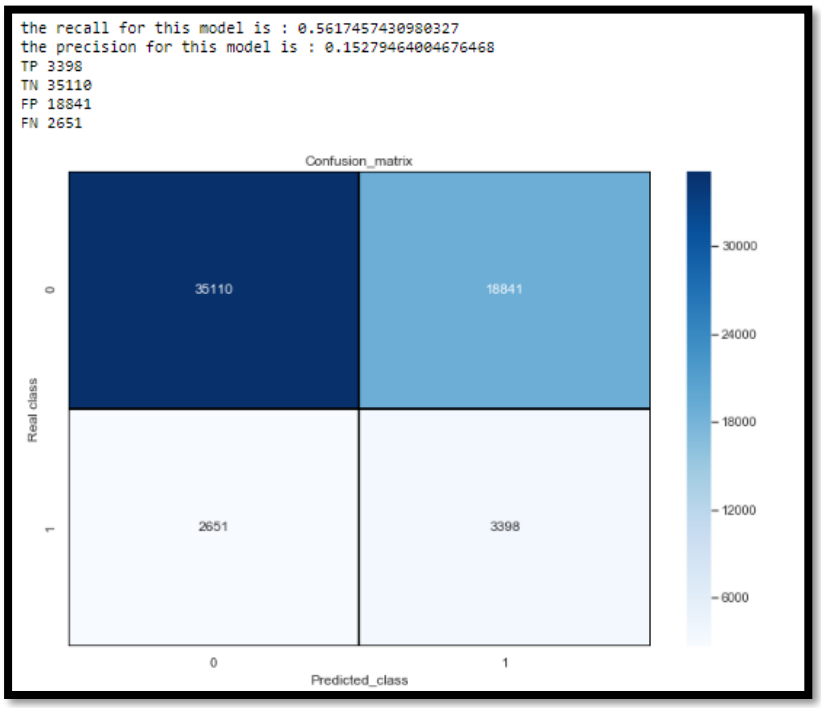


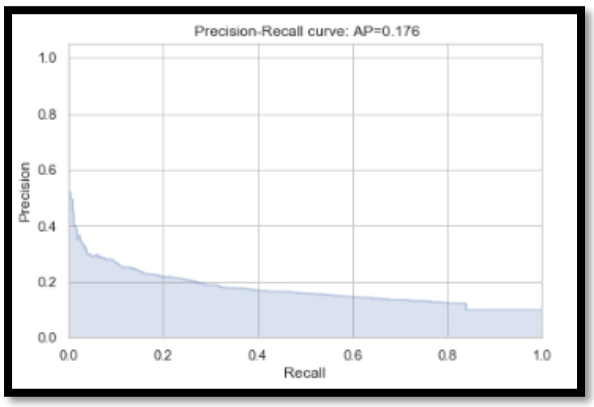
Fig 2.18 : ROC curve

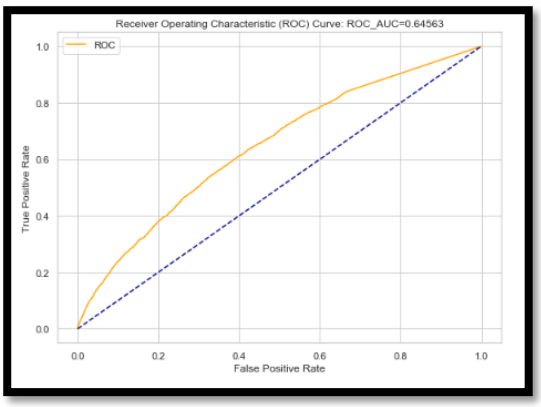
- From the plots above, we got an **average precision score of 0.516**, and **ROC_AUC** score of **0.86227.**

3.3 Decision Trees

- Moving on to a slightly advanced algorithm, decision trees. Again, the parameters here are class_weight to deal with unbalanced target variable, random_state for reproducibility of same trees.
- The feature max_features and min_sample_leaf are used to prune the tree and avoid overfitting to the training data.
- Max_features defines what proportion of available input features will be used to create tree.
- Min_sample_leaf restricts the minimum number of samples in a leaf node. If leaf nodes have less samples it implies we have grown the tree too much and trying to predict each sample very precisely, thus leading to overfitting.
- The precision for this model was found to be : 0.1527







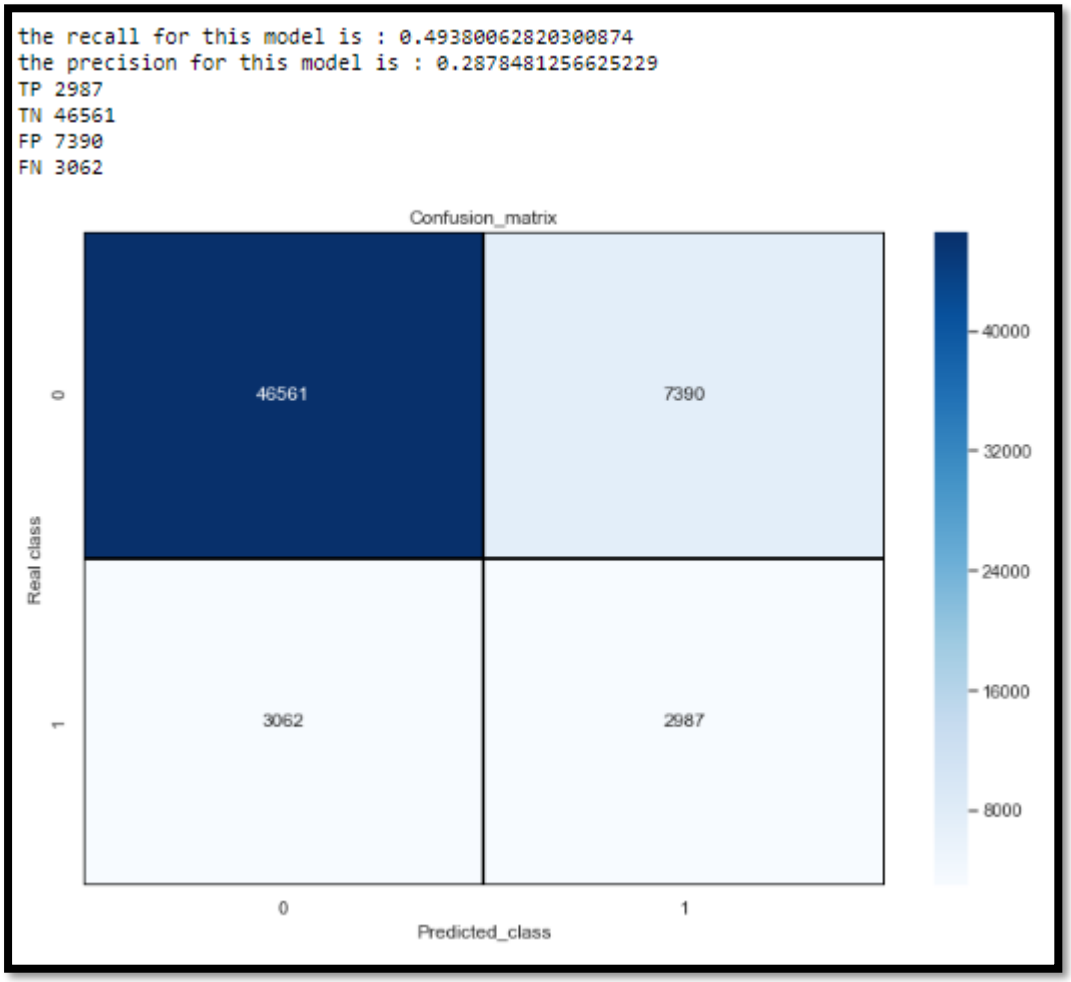
- From the plots above, we got an **average precision score of 0.176**, and **ROC_AUC** score of **0.64563.**

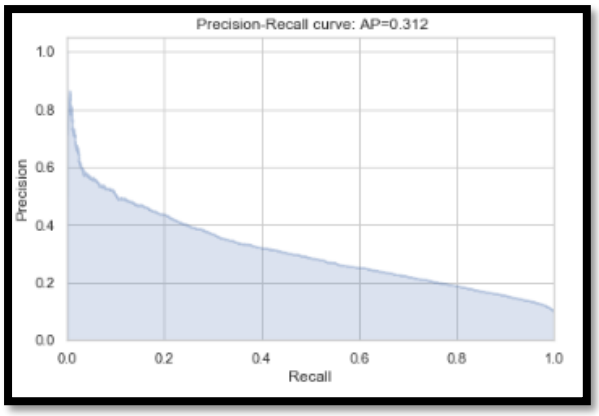
3.4 Ensemble Learning

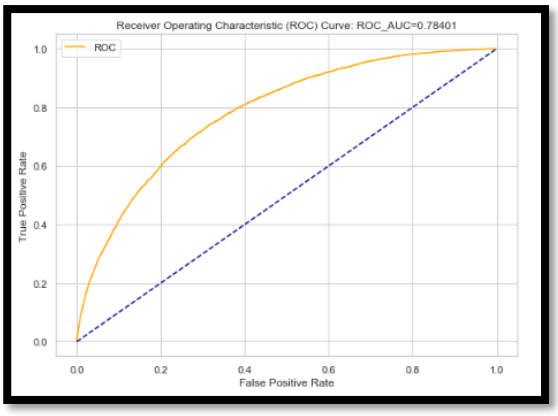
- Ensemble Learning refers to the algorithms that created using ensembles of various learning algorithms. For example, random forests are ensembles of many decision tree estimators.
- There are 2 types of ensemble learning algorithms:
- Bagging Algorithms: Bagging involves having each model in the ensemble vote with equal weight for the final output. In order to promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set
- Boosting Algorithms: Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models mis-classified.

Random Forest

- Let's start with building a random forest, with parameters like class_weight, random_state, and hyperparameters like max_features and min_sample_leaf as earlier.
- We have also defined the n_estimators which is a compulsory parameter. This defines the number of decision trees that will be present in the forest.
- The precision for this model was found to be : 0.2878, much better than the decision tree model.







- From the plots above, we got an **average precision score of 0.312**, and **ROC_AUC** score of **0.78401.**

3.5 Need for Bayesian approach of hyperparameter optimization

- An ML algorithm generally has a loss/cost function which needs to be minimized subject to the values of hyperparameters and parameters of the algorithm.

- Weights and Biases are decided on the basis of Optimization function of the ML algorithm. Selecting good hyperparameters further minimizes the loss function, and makes model more robust and accurate, increasing the overall efficiency of the model.

- For instance in Gradient Descent, it is very important to select a good learning rate to reach the convergence point in shortest time, because if it is large the cost function will overshoot and won't be able to find minima or if too small it will take forever to reach the minima.

- Now even though Scikit-Learn provides us Grid Search and Random Search, these algorithms are brute force and the computation time grows as grid becomes more dense. Even the best possible combination might be far from the optimal.

- For example, the **common implementation of random search completely ignores information on the trials already computed**, and each new sample is drawn from the same initial distribution.

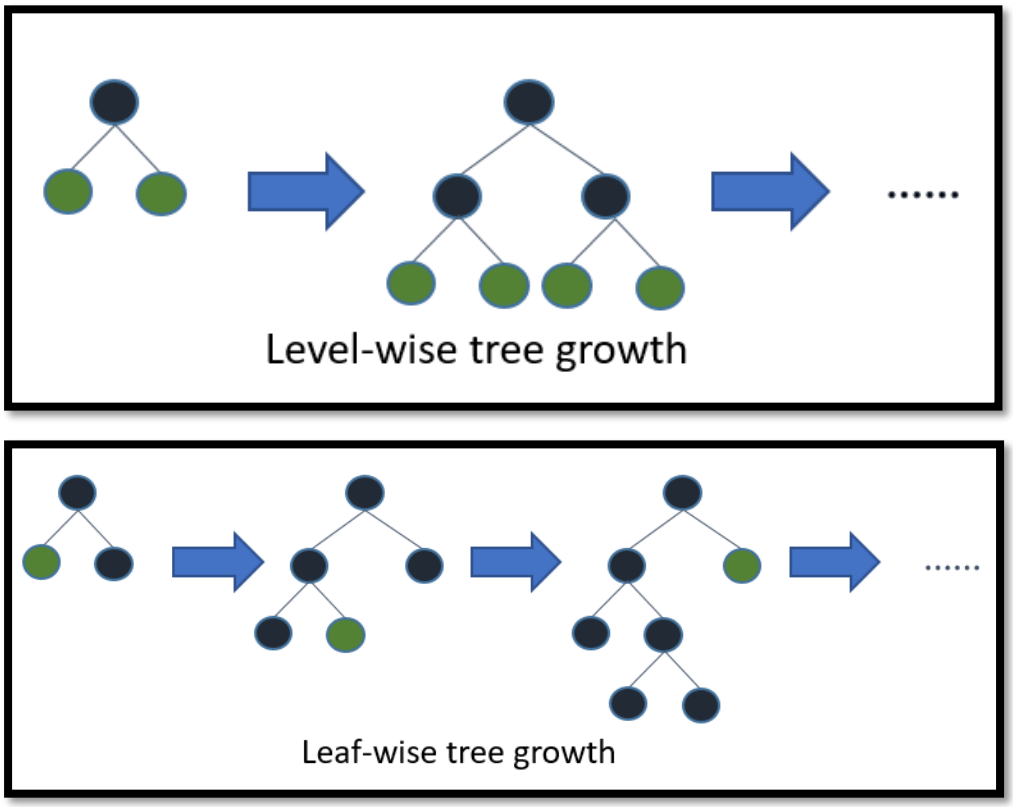
- Fortunately, there is a way to account for them. Say, you were tuning C - regularization parameter for logistic regression. If one particular value gives really bad results - the points in vicinity will also perform poorly, so there is really very little need to sample from this region.

- We would like to incorporate this information into our strategy - in other words, we want to get more points from the regions with high probability of yielding good result and get less points from elsewhere.

- **BayesianOptimization** library uses Bayesian approach for intelligent points sampling from search space. It adjusts prior distribution using history of target function evaluations, concentrating probability mass in the region where function performs better.

3.6 Light GBM

- Light GBM is a gradient boosting framework that uses tree based learning algorithm. It grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. Leaf-wise algorithm can reduce more loss than a level-wise algorithm.



- It is 'Light' because of its high speed. It can handle large data, requires low memory to run and focuses on accuracy of results.

- Since in our dataset, all the features are independent of each other, it's better to train an ensemble of 200 LGB models.

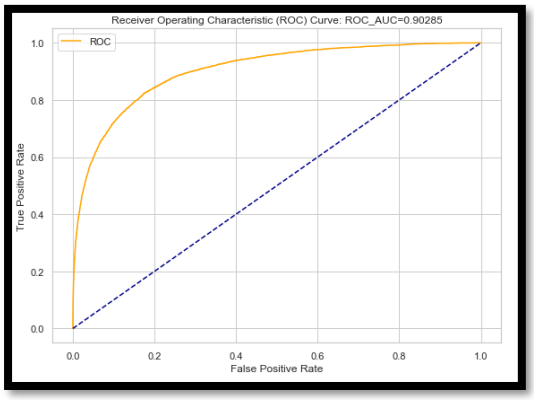
- Each training model with take 2 features at a time: The original one and an extra column with the unique values count. (For example: var_0 and new_var_0)

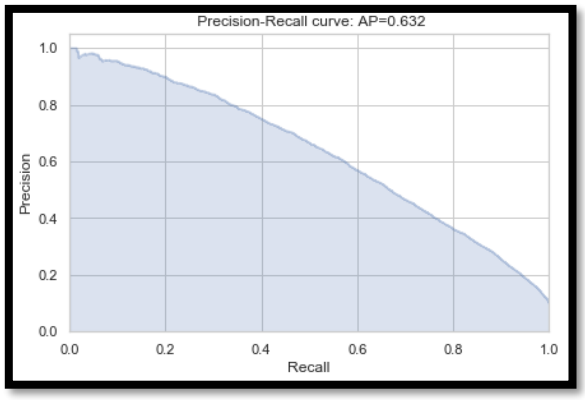
- Using this method, saved us a lot of time for hyperparameter optimisation, instead of running a single LGB model which takes into account the interaction of all 400 features.

- Without hyperparameter tuning of LGB model we got **average precision score of 0.593**, and **ROC_AUC** score of **0.88645.**

- BayesianOptimization gave us the following optimal parameters:
  {'bagging_fraction': 0.7, 'bagging_freq': 2, 'boost_from_average': False,
  'is_unbalance': True, 'lambda_l1': 0.7, 'lambda_l2': 1.9999999964082154,
  'learning_rate': 0.009999999503478726, 'max_depth': 5, 'metric': 'auc',
  'min_data_in_leaf': 25, 'min_gain_to_split': 0.9880270270985563,
  'min_sum_hessian_in_leaf': 20.0, 'num_leaves': 30, 'objective': 'binary'}

- After hyperparameter tuning of LGB model with **BayesianOptimization** library, we achieved the following results.





- From the plots above, we got an **average precision score of 0.632**, and **ROC_AUC** score of **0.90285.**

3.7 Auto ML (H2O)

- **AutoML** function of H2O.ai was also tried, which automates the process of building large number of models, with the goal of finding the "best" model without any prior knowledge.

- Stacked-ensemble model topped the AutoML leader board with an **precision_auc** score of **0.5672** and **ROC_AUC** score of **0.8808.**

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.195621 on test data:

	0	1	Error	Rate
0	51003.0	2924.0	0.0542	(2924.0/53927.0)
1	2687.0	3383.0	0.4427	(2687.0/6070.0)
Total	53690.0	6307.0	0.0935	(5611.0/59997.0)

Chapter 4

Summary

- Santander is interested in finding which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

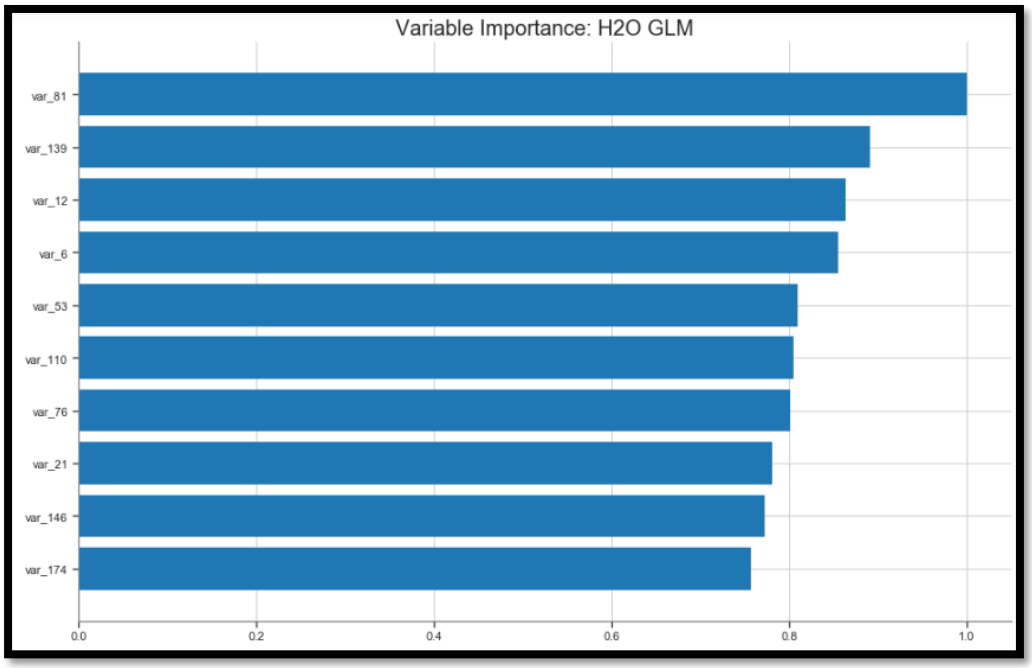
- Hence , it is interested in correctly identifying the customers with target label as 1, (i.e. customers who will make a specific transaction in the future)

- Since our dataset is an imbalance class dataset, where the proportion of positive samples is low **(around 10%)**, we should aim for **higher precision since it does not include True negatives in calculation, and hence it will not affected by class imbalance.**

- Therefore, **precision-recall (PR) curve** should be chosen as an evaluation metric instead of ROC curves in this scenario.

- Among all the models trained so far, **light GBM** performed the best with **PR_AUC** of **0.632**

- Top 10 features with their variable importance w.r.t to GLM model of H2O.



References

- https://medium.com/analytics-vidhya/gentle-introduction-to-automl-from-h2o-ai-a42b393b4ba2

- https://fizzylogic.nl/2018/08/21/5-must-have-tools-if-youre-serious-about-machine-learning/

- https://medium.com/spikelab/hyperparameter-optimization-using-bayesian-optimization-f1f393dcd36d

- https://lightgbm.readthedocs.io/en/latest/Python-Intro.html

- https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

- https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc