Santander Customer Transaction Prediction Problem

<u>Organization Context:</u>

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?

<u>Problem Statement:</u>

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

Data Set:

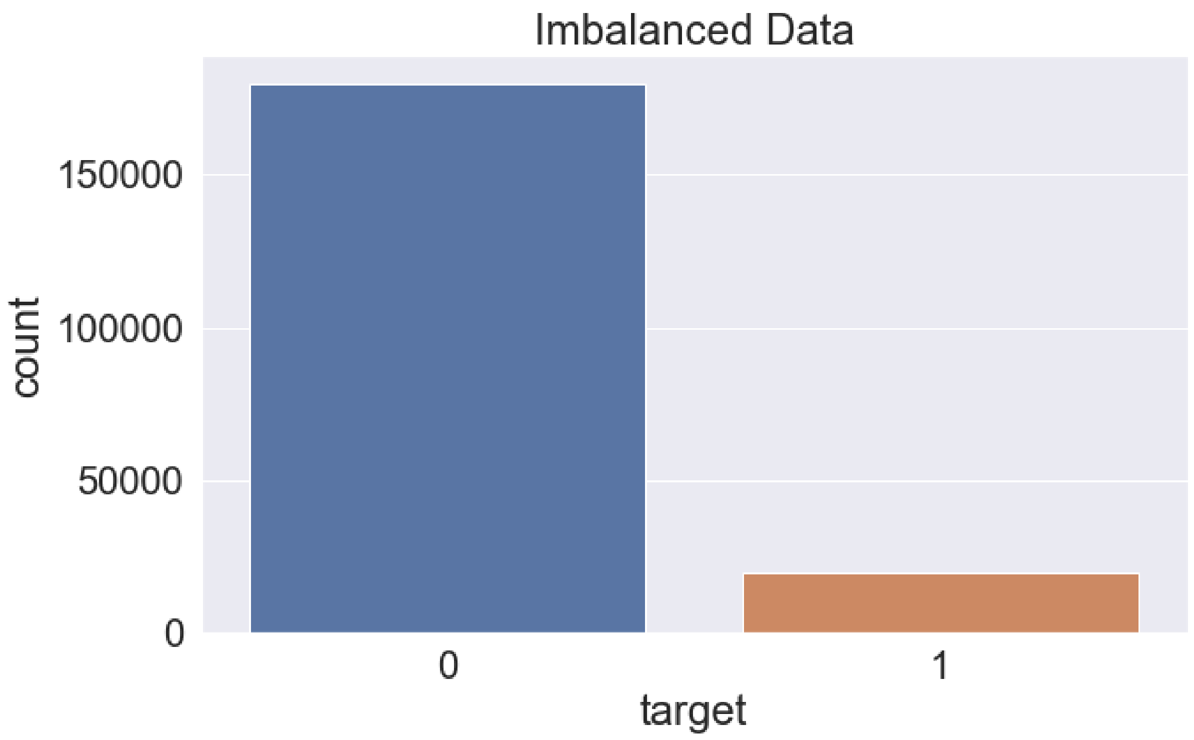
- test.csv
- train.csv

<u>Number of attributes:</u>

You are provided with an anonymized dataset containing numeric feature variables, the binary target column, and a string ID_code column. The task is to predict the value of target column in the test set.

Missing Values: No

<u>Exploratory Data Analysis (EDA):</u>



Preliminary Analysis:

- There are 200,000 rows of anonymized and redacted training data and testing data
- There are 200 variables of unknown significance.
- There are no missing values in the dataset.
- Dataset has class imbalance problem – around 90% of the training data has a target of 0 and 10% has target of 1.

Metadata verification:

```
<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
```

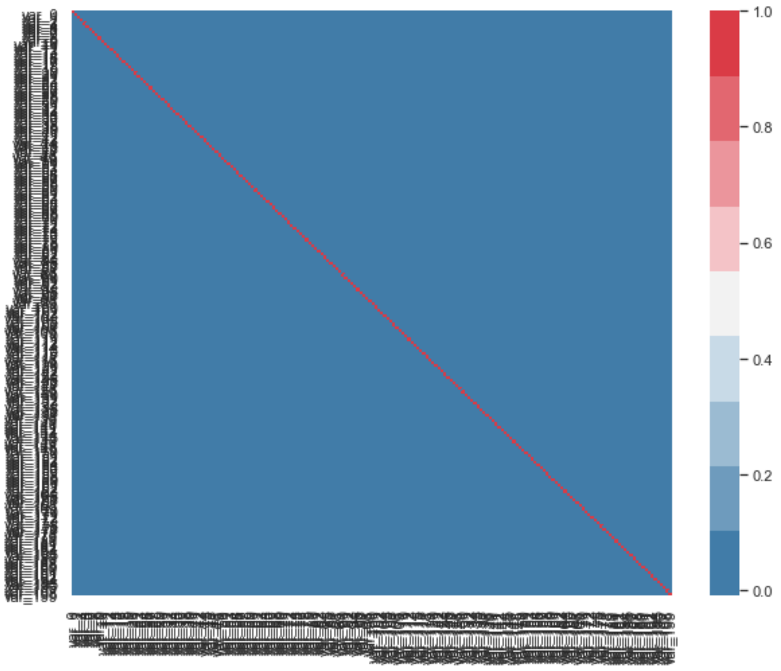
```
Datatype of target: int64
Datatype of variables: float64
```

- 200 columns of variables are floating point numbers
- They have a maximum of 4 decimal places
- Consists of both positive and negative numbers

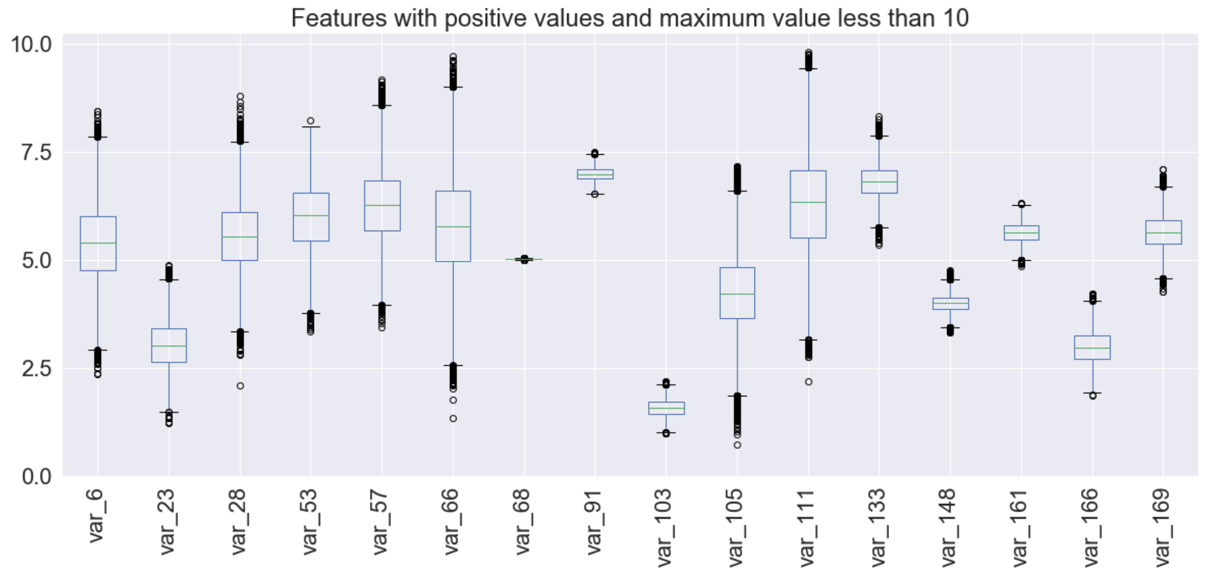
Feature analysis:

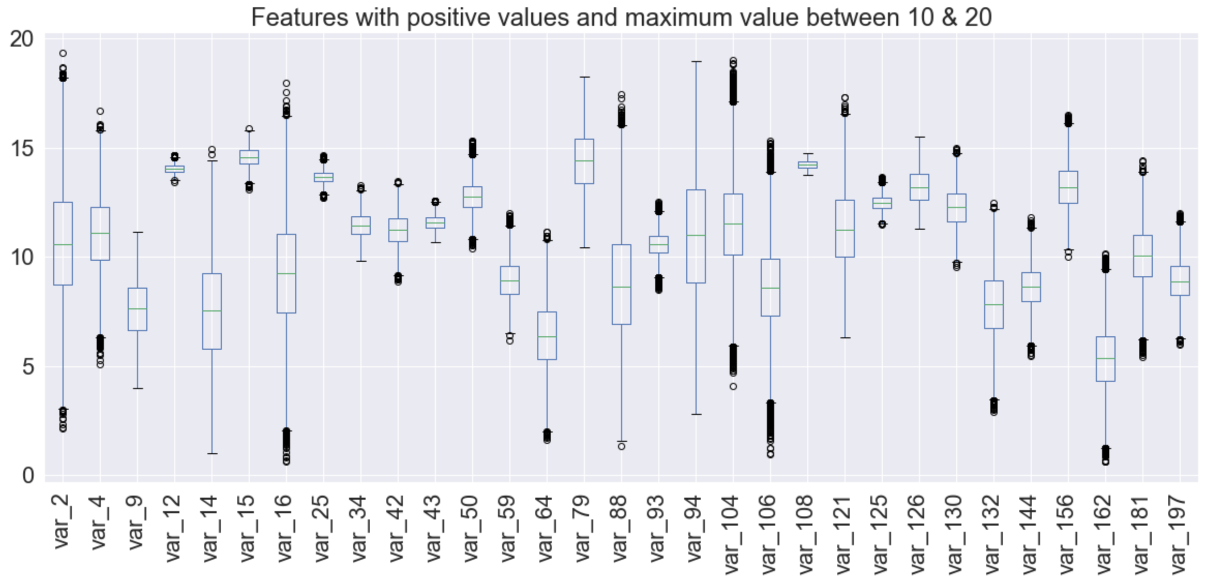
Feature Analysis:

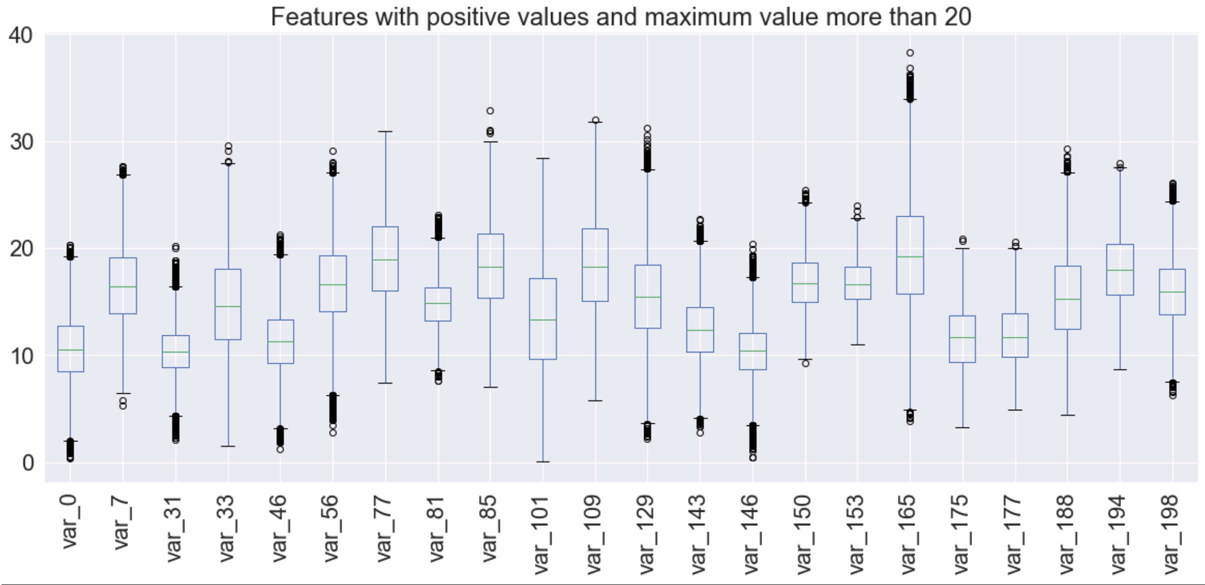
- Some boxplots show numerous outliers, and some show very small variation between samples.
- The most common case is positive features with a max between 10-20
- There are no purely negative features.
- Correlation analysis tells us that we can infer little to no correlations among variables.
- Highest correlation is not even 1%

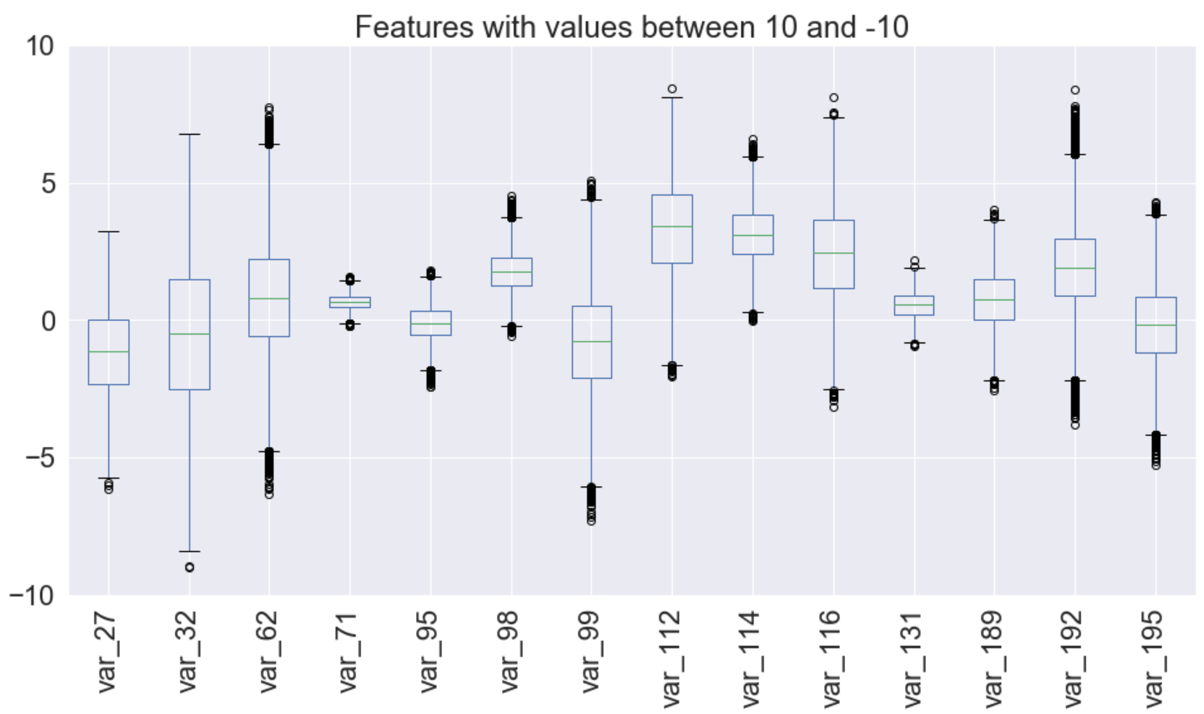


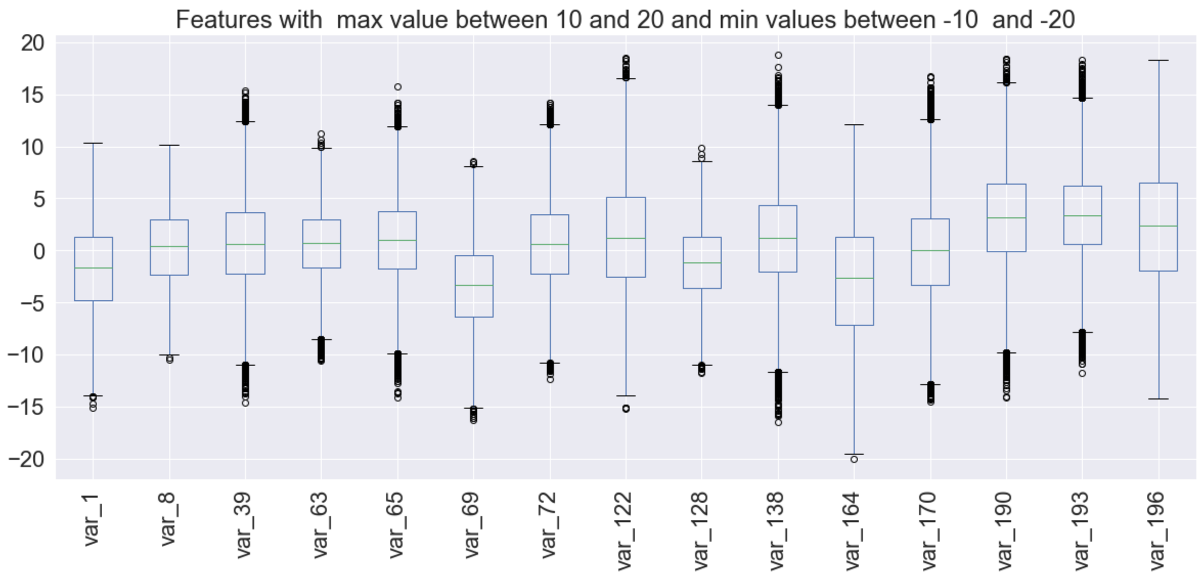
Boxplots:

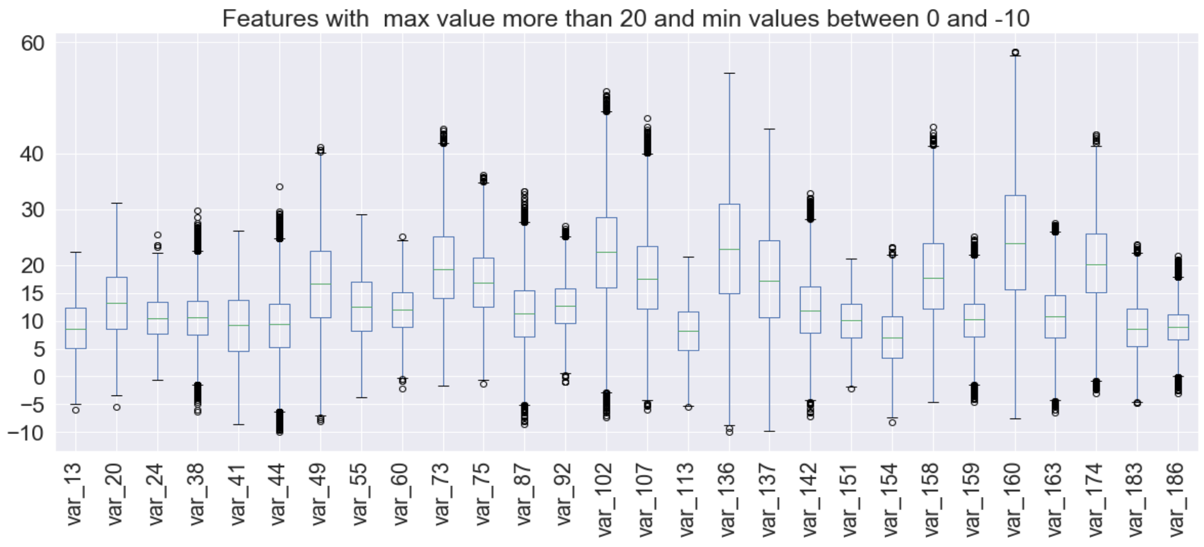

Features with positive values and maximum value less than 10
10.0
7.5
5.0
2.5
0.0
var_6
var_23
var_28
var_53
var_57
var_66
var_68
var_91
var_103
var_105
var_111
var_133
var_148
var_161
var_166
var_169

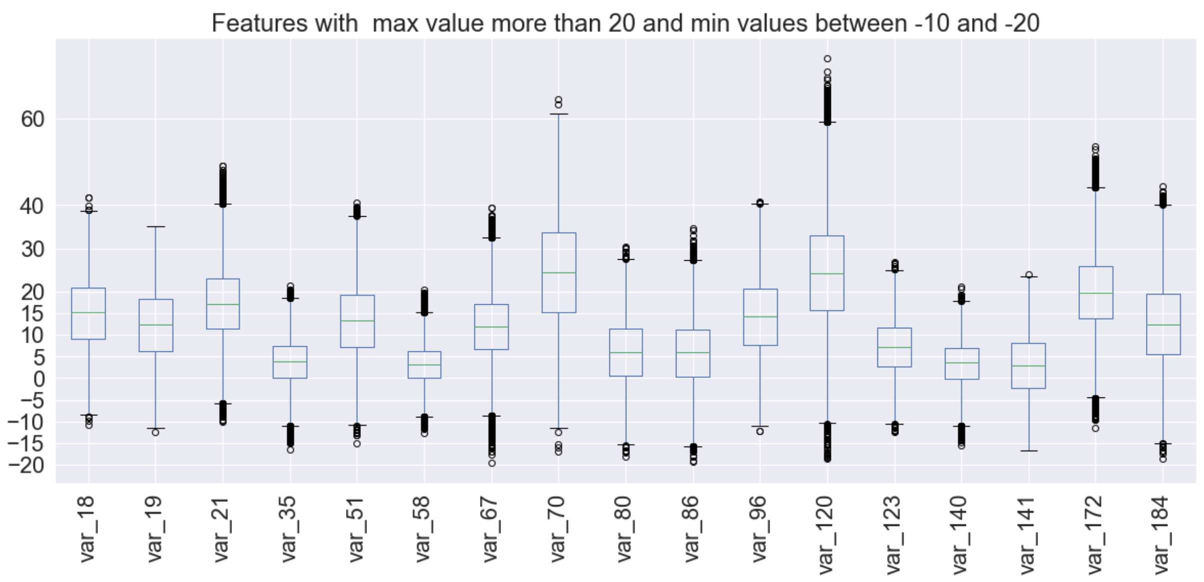

Features with positive values and maximum value between 10 & 20
20
15
10
5
0
var_2
var_4
var_9
var_12
var_14
var_15
var_16
var_25
var_34
var_42
var_43
var_50
var_59
var_64
var_79
var_88
var_93
var_94
var_104
var_106
var_108
var_121
var_125
var_126
var_130
var_132
var_144
var_156
var_162
var_181
var_197

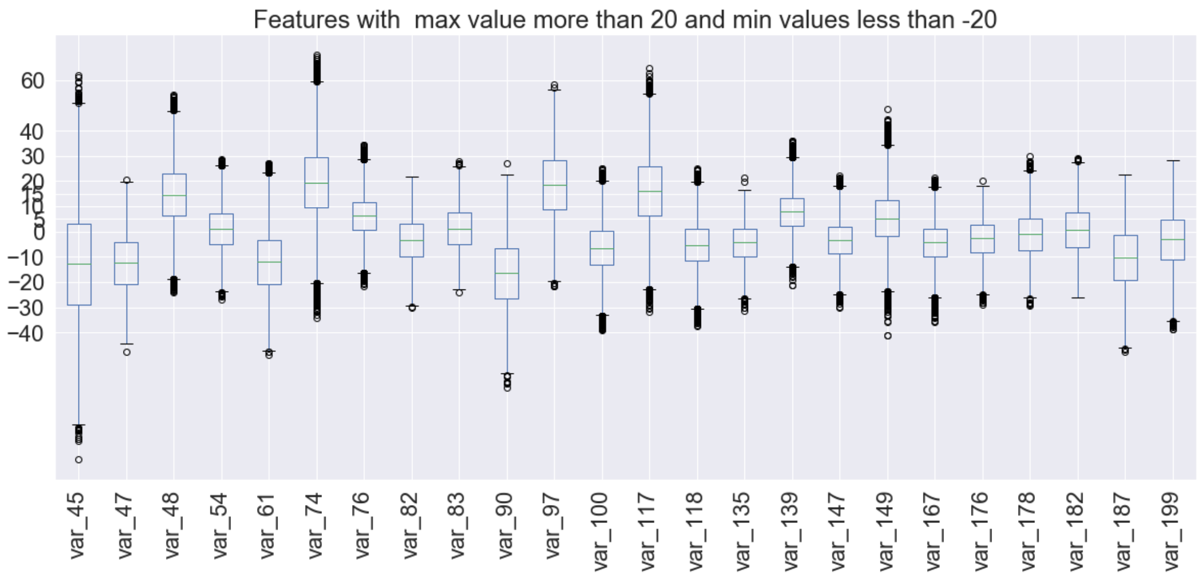

Features with positive values and maximum value more than 20
40
30
20
10
0
var_0
var_7
var_31
var_33
var_46
var_56
var_77
var_81
var_85
var_101
var_109
var_129
var_143
var_146
var_150
var_153
var_165
var_175
var_177
var_188
var_194
var_198


Features with values between 10 and -10
10
5
0
−5
−10
var_27
var_32
var_62
var_71
var_95
var_98
var_99
var_112
var_114
var_116
var_131
var_189
var_192
var_195


Features with max value between 10 and 20 and min values between -10 and -20
20
15
10
5
0
−5
−10
−15
−20
var_1
var_8
var_39
var_63
var_65
var_69
var_72
var_122
var_128
var_138
var_164
var_170
var_190
var_193
var_196


Features with max value more than 20 and min values between 0 and -10
60
40
30
20
15
10
5
0
−5
−10
var_13
var_20
var_24
var_38
var_41
var_44
var_49
var_55
var_60
var_73
var_75
var_87
var_92
var_102
var_107
var_113
var_136
var_137
var_142
var_151
var_154
var_158
var_159
var_160
var_163
var_174
var_183
var_186





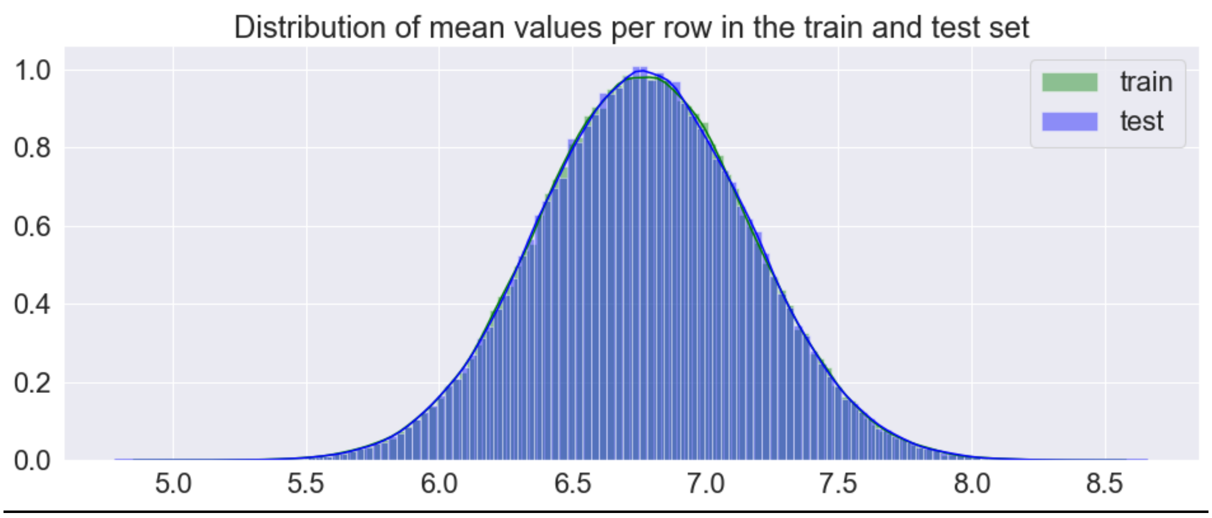
Summary Statistics Analysis:

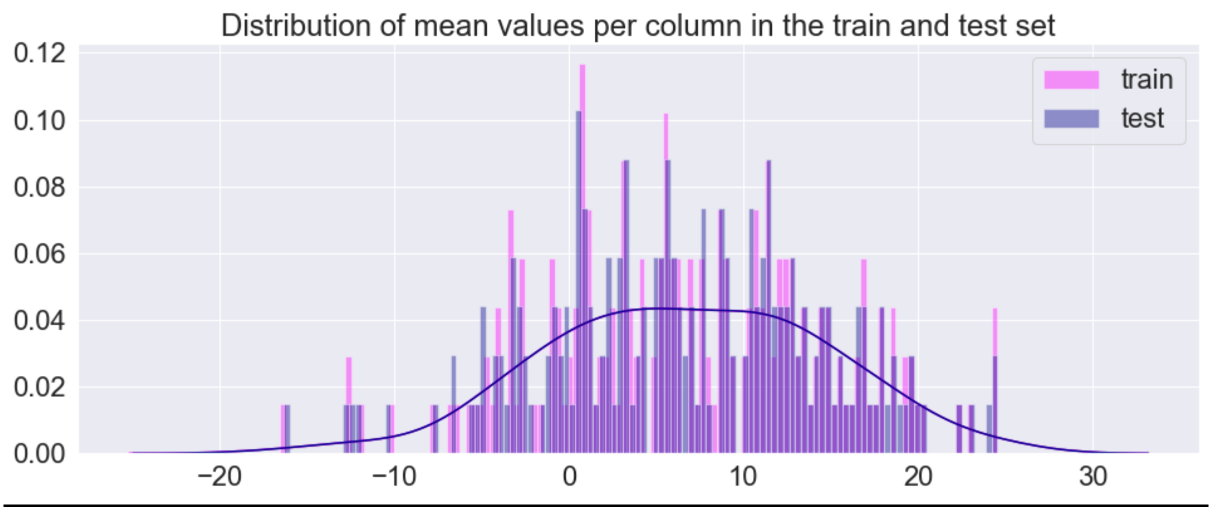
We set out to analyse the following statistics on both the train and test data:

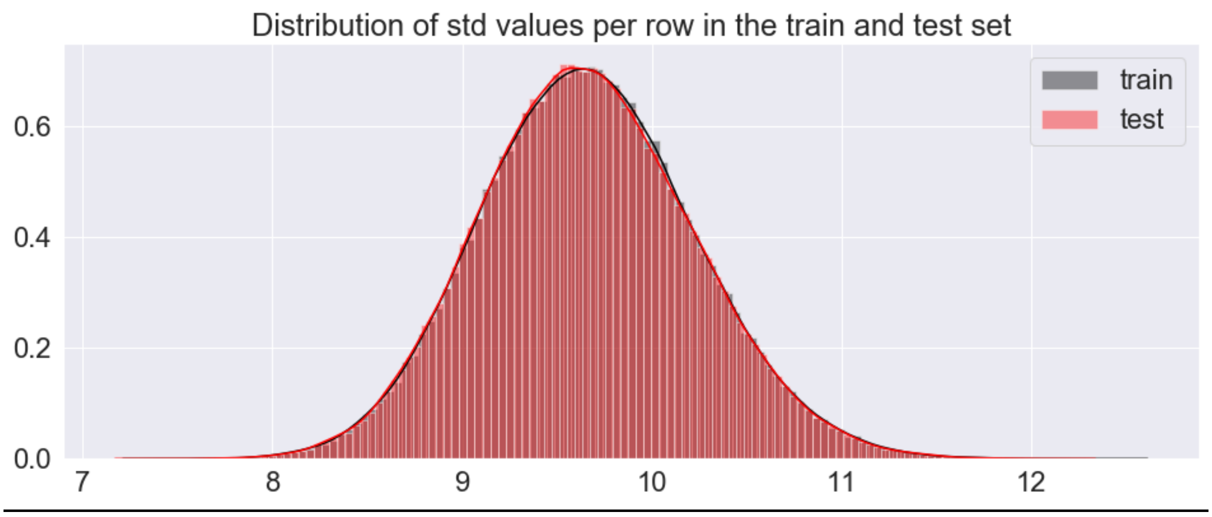
- mean
- std
- min
- max
- skew - *Is the distribution right-skewed (negative) or left-skewed (positive)*
- kurtosis - *Are the tails of the distribution large or small? below 0 means light tails.*

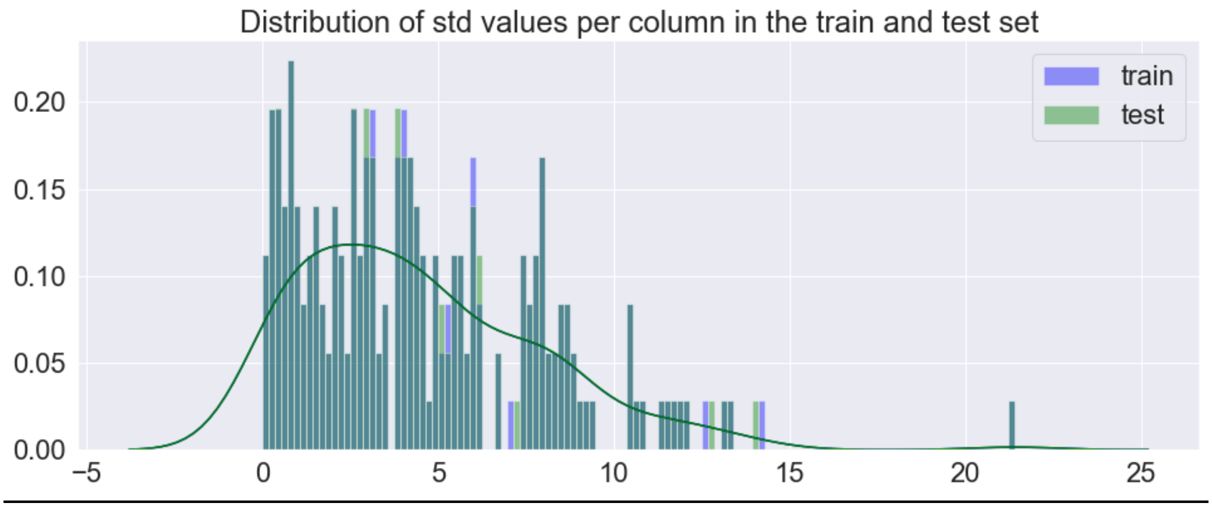
The observations from summary statistical analysis are as follows:

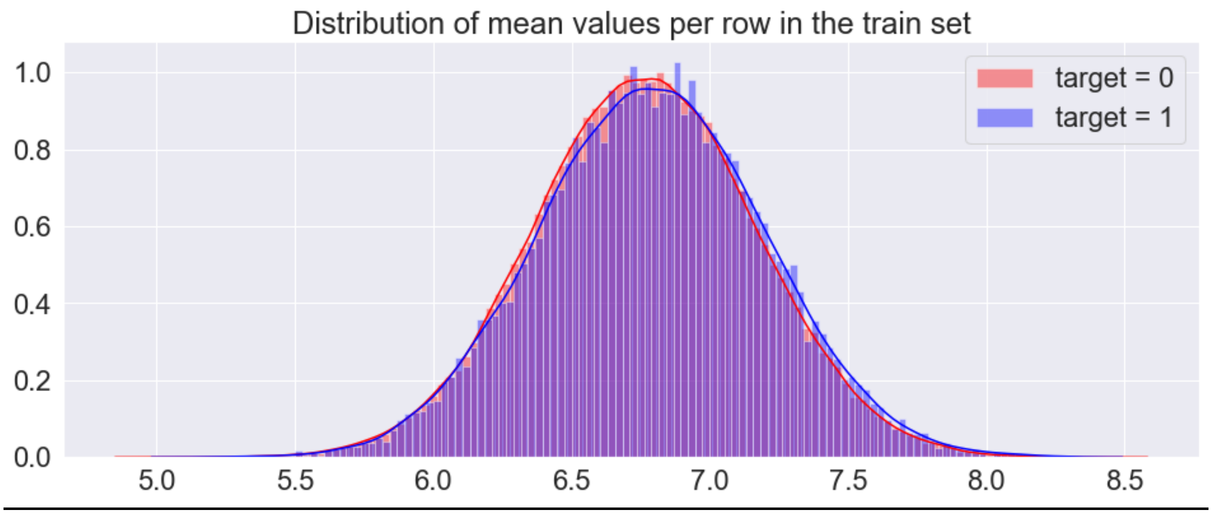
- There is little to no differences when examining features by row.
- There are some noticeable differences when examining features by column/target, but no significant patterns.
- The most notable difference overall is the kurtosis values per column in the training set.

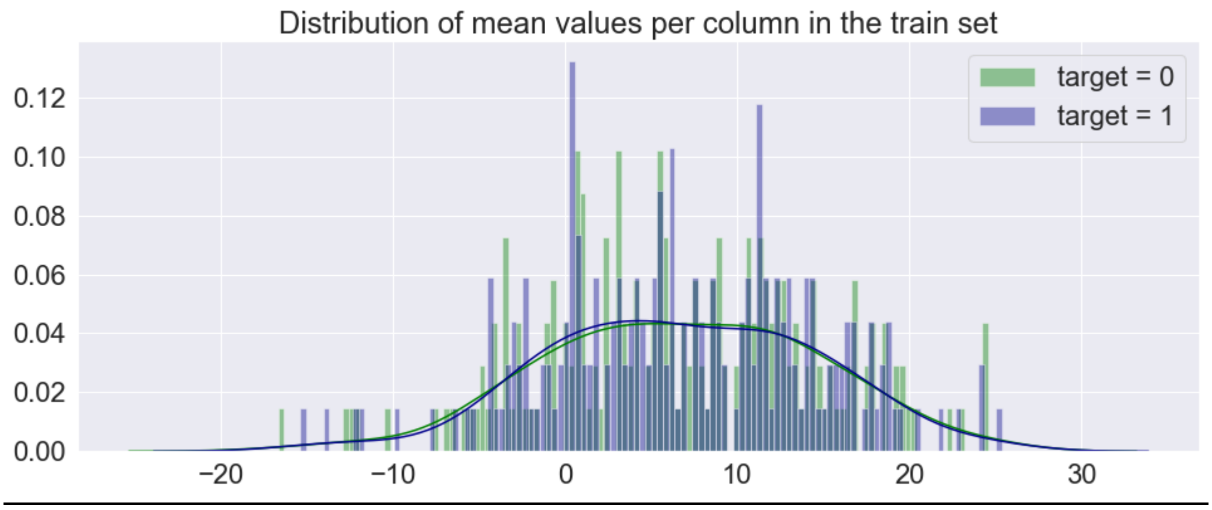

Distribution of mean values per row in the train and test set
train
test
1.0
0.8
0.6
0.4
0.2
0.0
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5


Distribution of mean values per column in the train and test set
train
test
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−20
−10
0
10
20
30

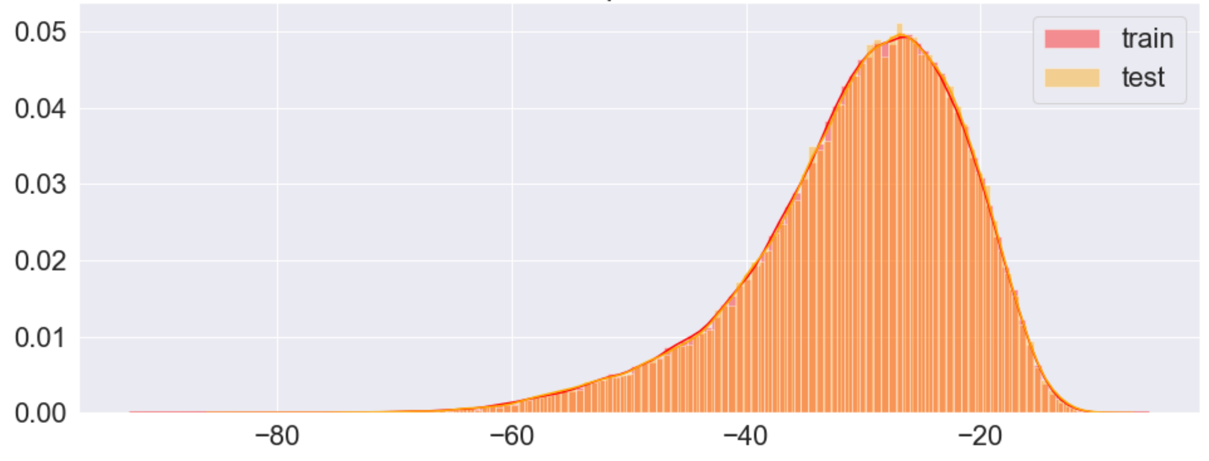

Distribution of std values per row in the train and test set
train
test
0.6
0.4
0.2
0.0
7
8
9
10
11
12


Distribution of std values per column in the train and test set
train
test
0.20
0.15
0.10
0.05
0.00
−5
0
5
10
15
20
25

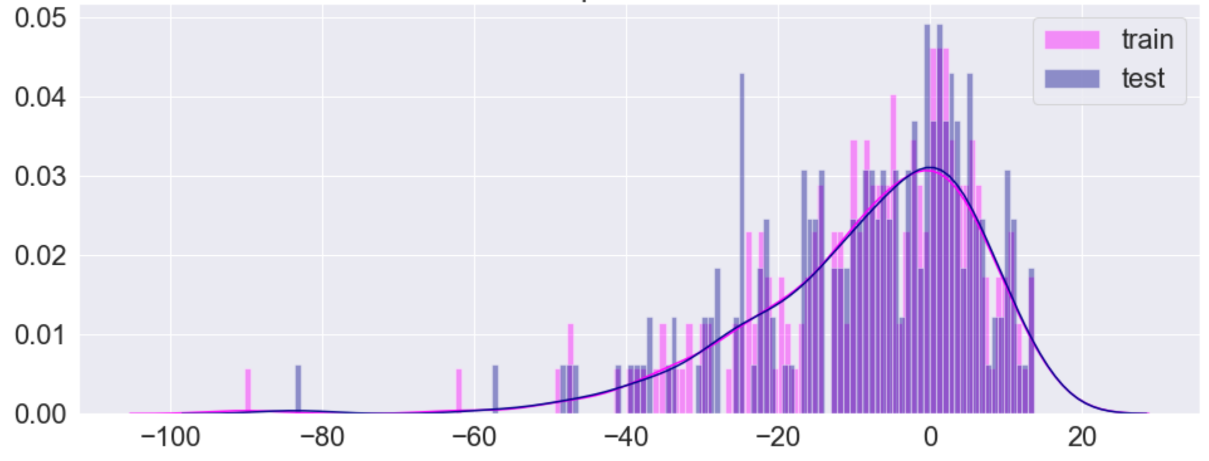

Distribution of mean values per row in the train set
target = 0
target = 1
1.0
0.8
0.6
0.4
0.2
0.0
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5


Distribution of mean values per column in the train set
target = 0
target = 1
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−20
−10
0
10
20
30

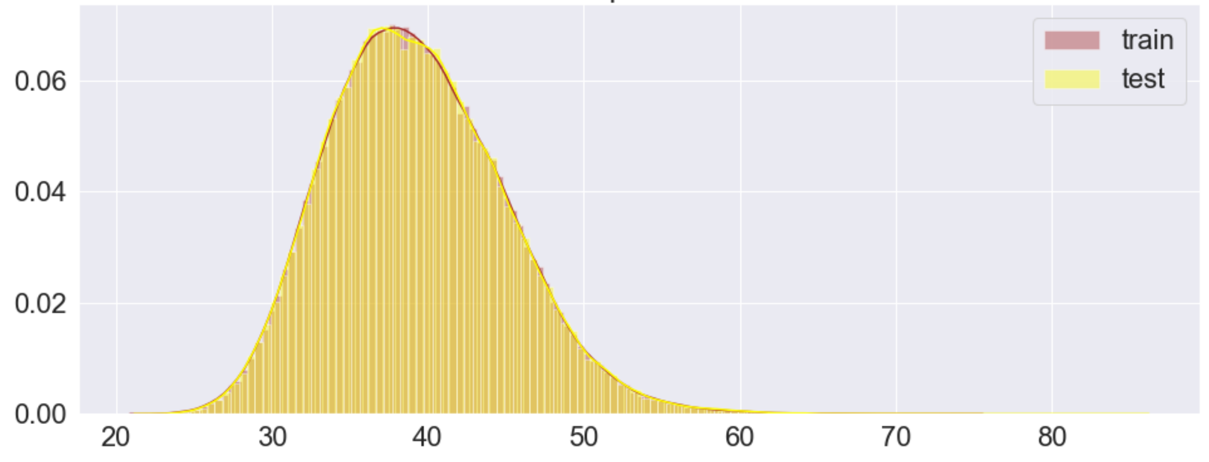
Distribution of min values per row in the train and test set

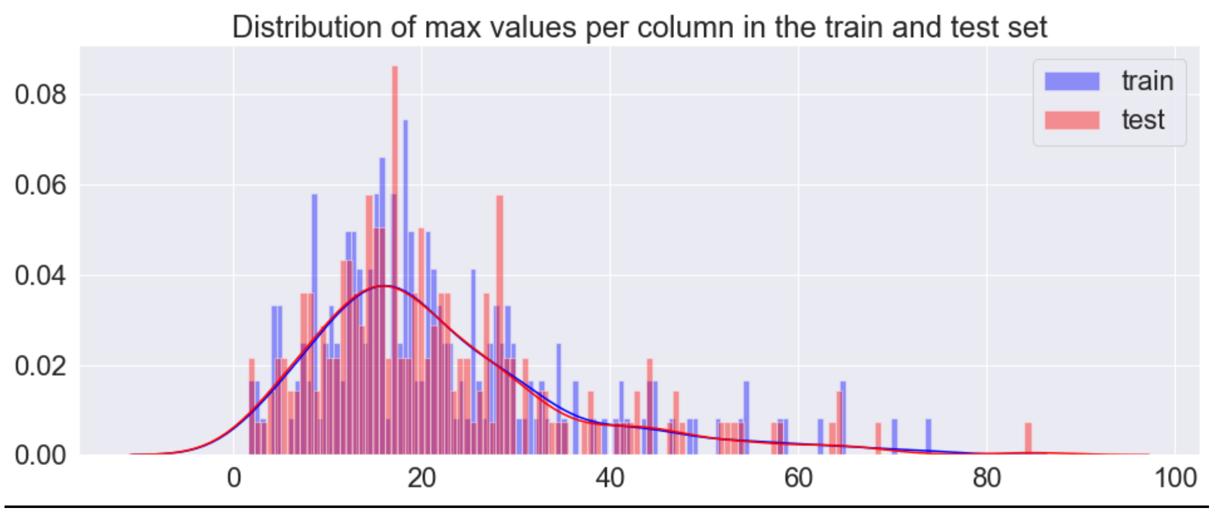


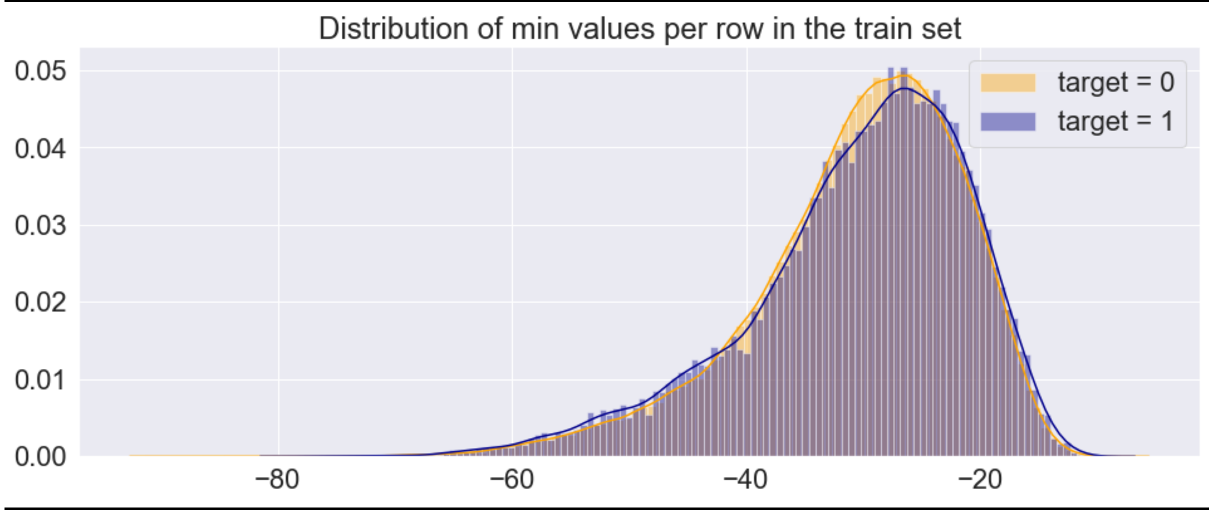
Distribution of min values per column in the train and test set

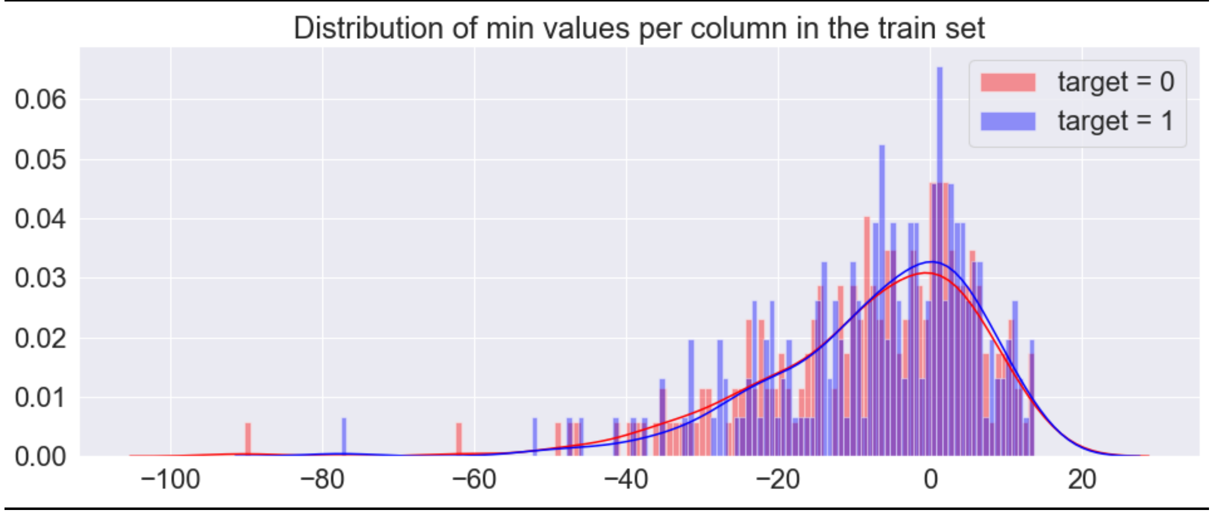


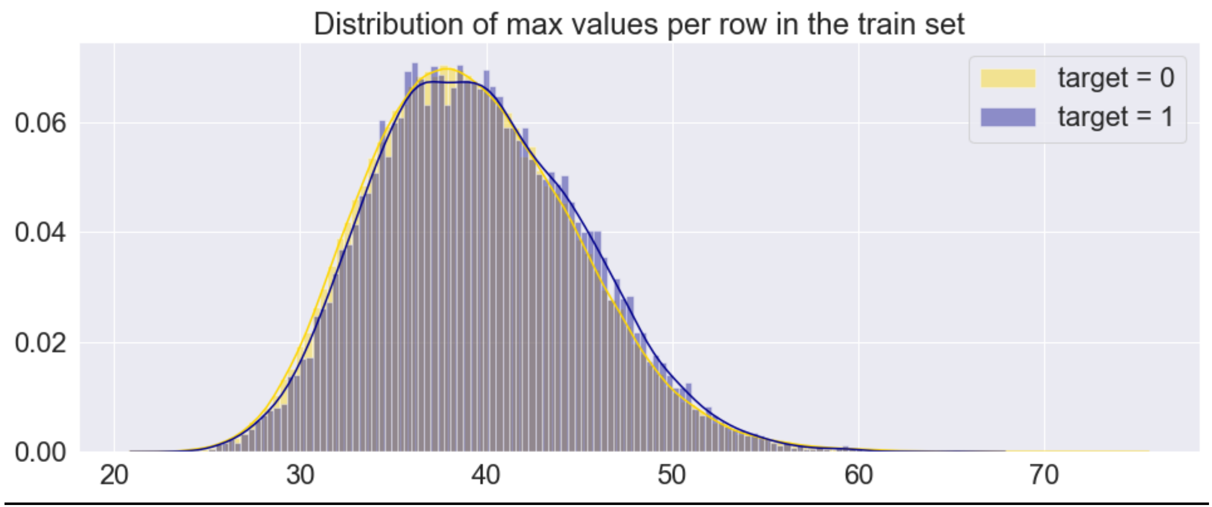
Distribution of max values per row in the train and test set

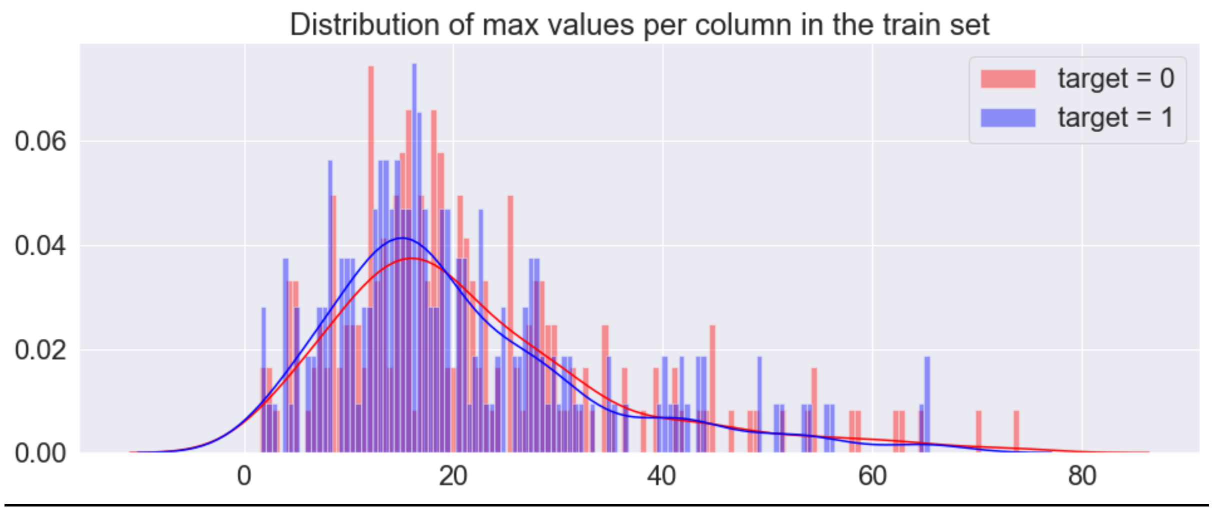


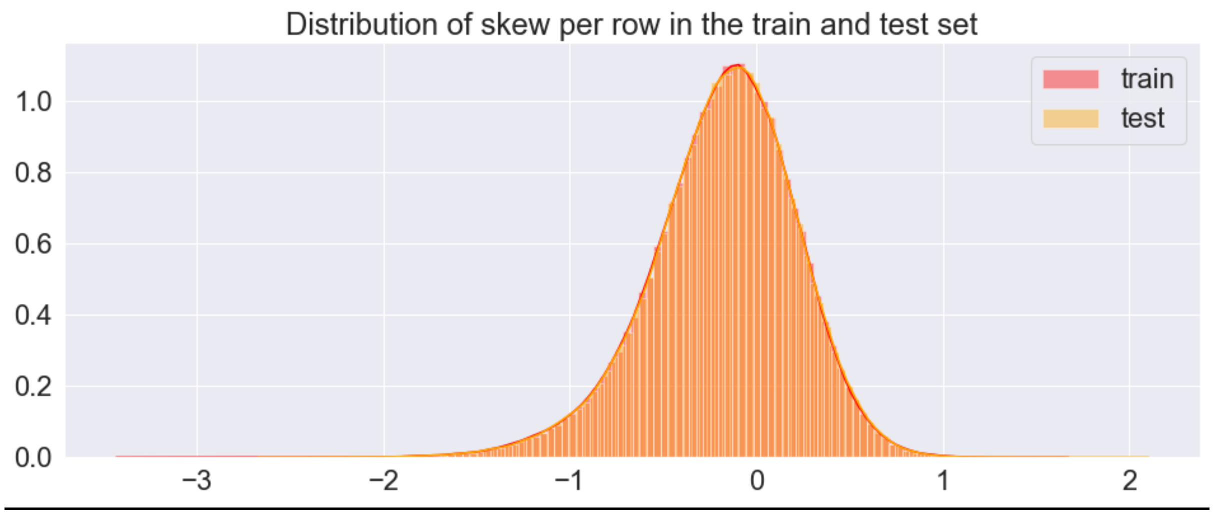

Distribution of max values per column in the train and test set
train
test
0.08
0.06
0.04
0.02
0.00
0
20
40
60
80
100

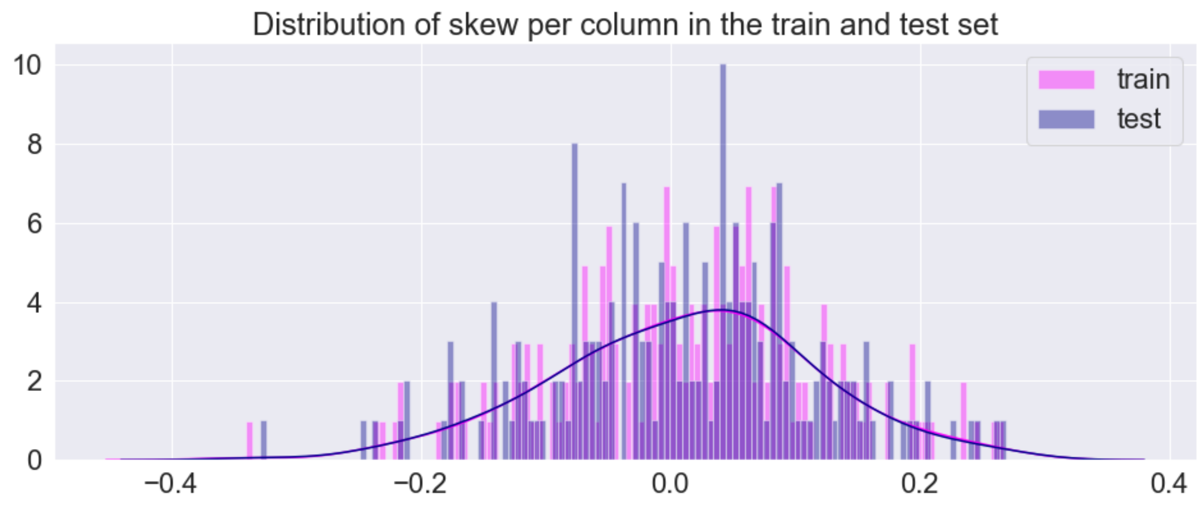

Distribution of min values per row in the train set
target = 0
target = 1
0.05
0.04
0.03
0.02
0.01
0.00
−80
−60
−40
−20

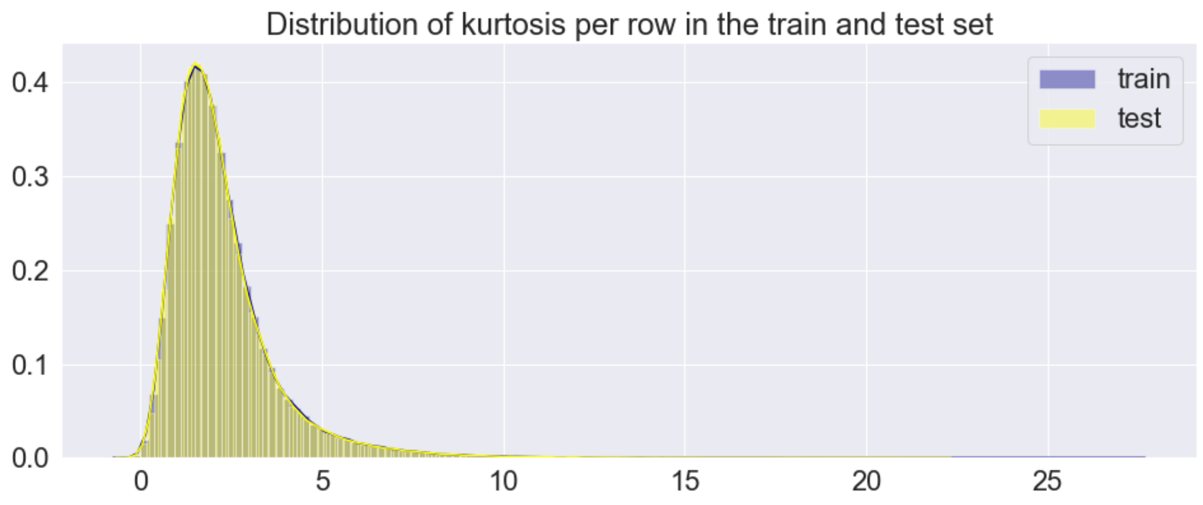

Distribution of min values per column in the train set
target = 0
target = 1
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−100
−80
−60
−40
−20
0
20

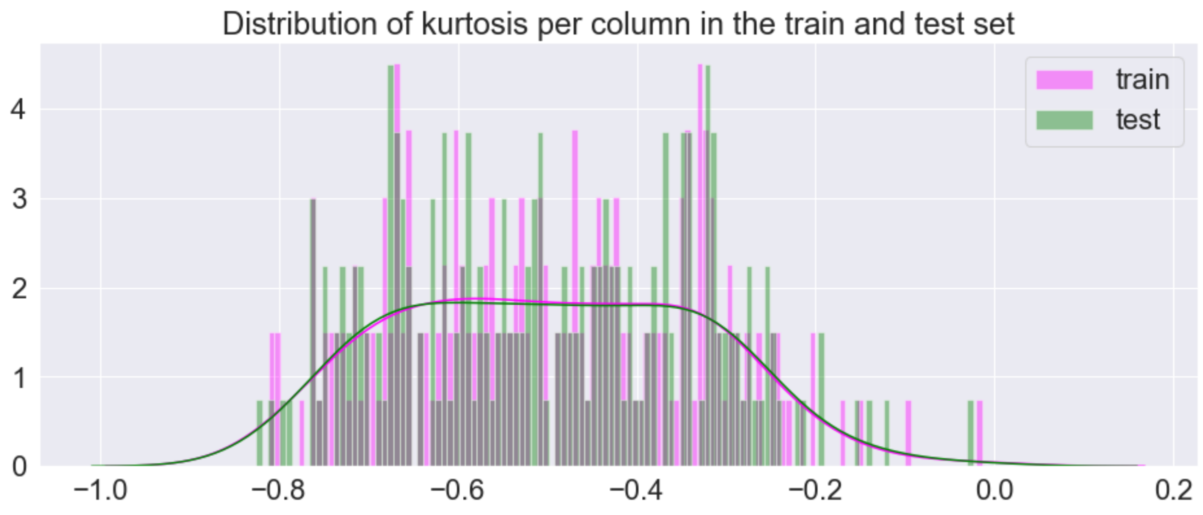

Distribution of max values per row in the train set
target = 0
target = 1
0.06
0.04
0.02
0.00
20
30
40
50
60
70

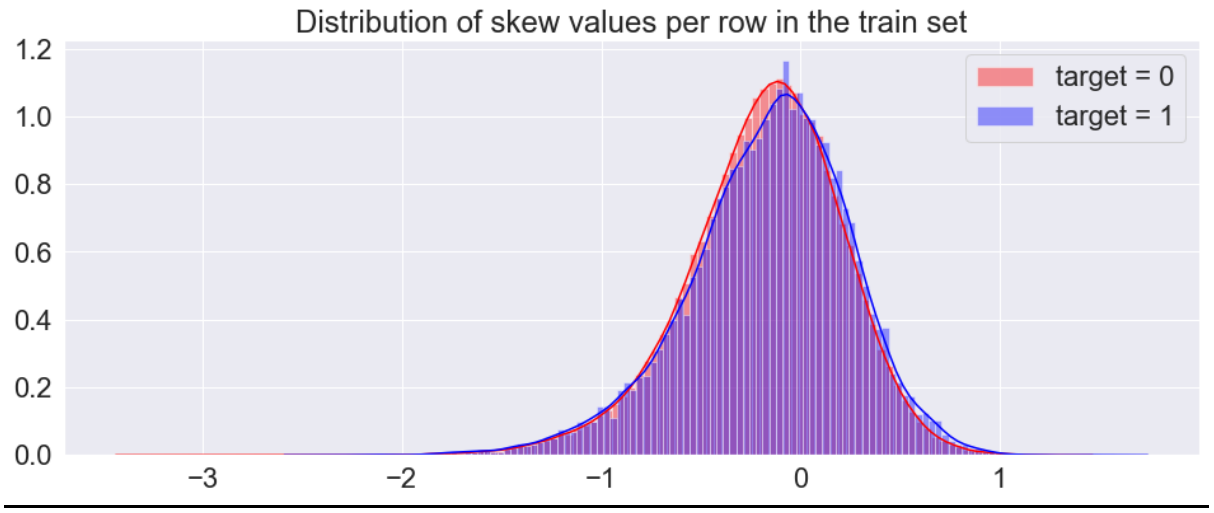

Distribution of max values per column in the train set
target = 0
target = 1
0.06
0.04
0.02
0.00
0
20
40
60
80

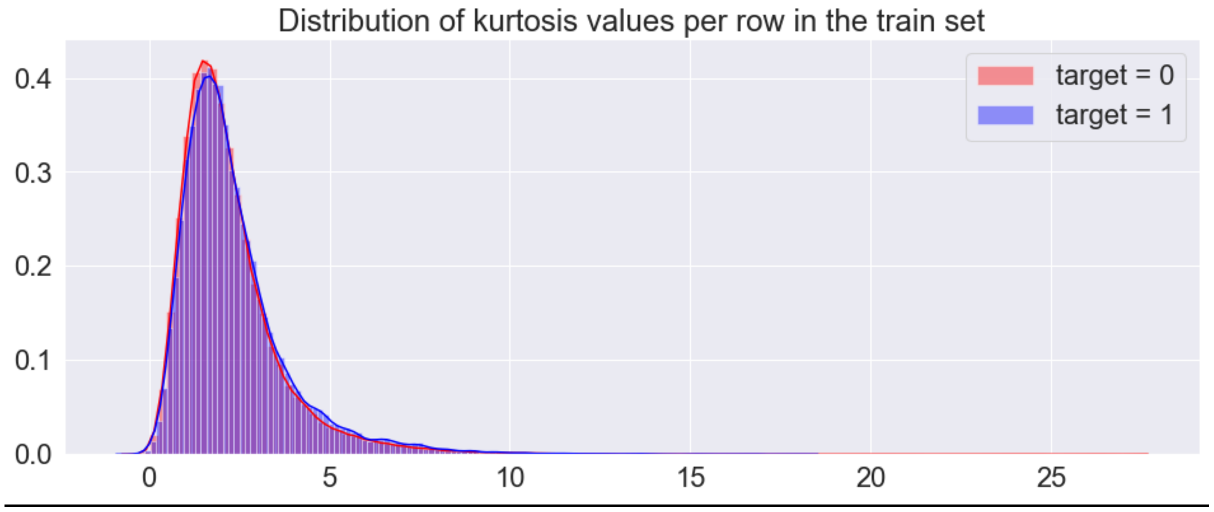

Distribution of skew per row in the train and test set
train
test
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2

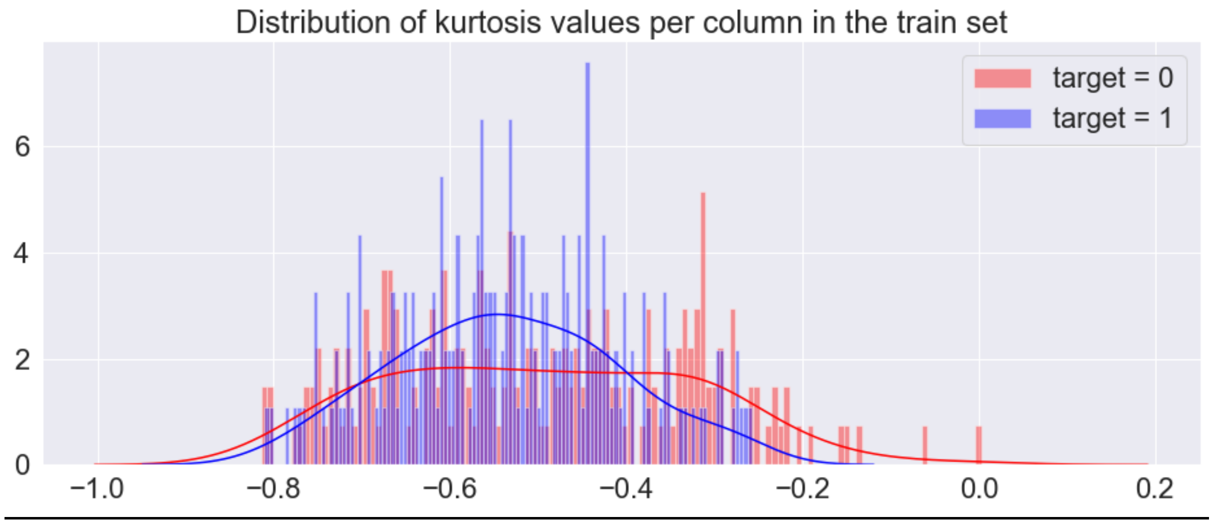

Distribution of skew per column in the train and test set
train
test
10
8
6
4
2
0
−0.4
−0.2
0.0
0.2
0.4


Distribution of kurtosis per row in the train and test set
train
test
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20
25


Distribution of kurtosis per column in the train and test set
train
test
4
3
2
1
0
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2


Distribution of skew values per row in the train set
target = 0
target = 1
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1


Distribution of kurtosis values per row in the train set
target = 0
target = 1
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20
25


Distribution of kurtosis values per column in the train set
target = 0
target = 1
6
4
2
0
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2

An early Logistic Regression Model:

First, we need to separate the TARGET values from the features on the TRAIN data ONLY; Next, we need to *train_test_split* the training data. Why?

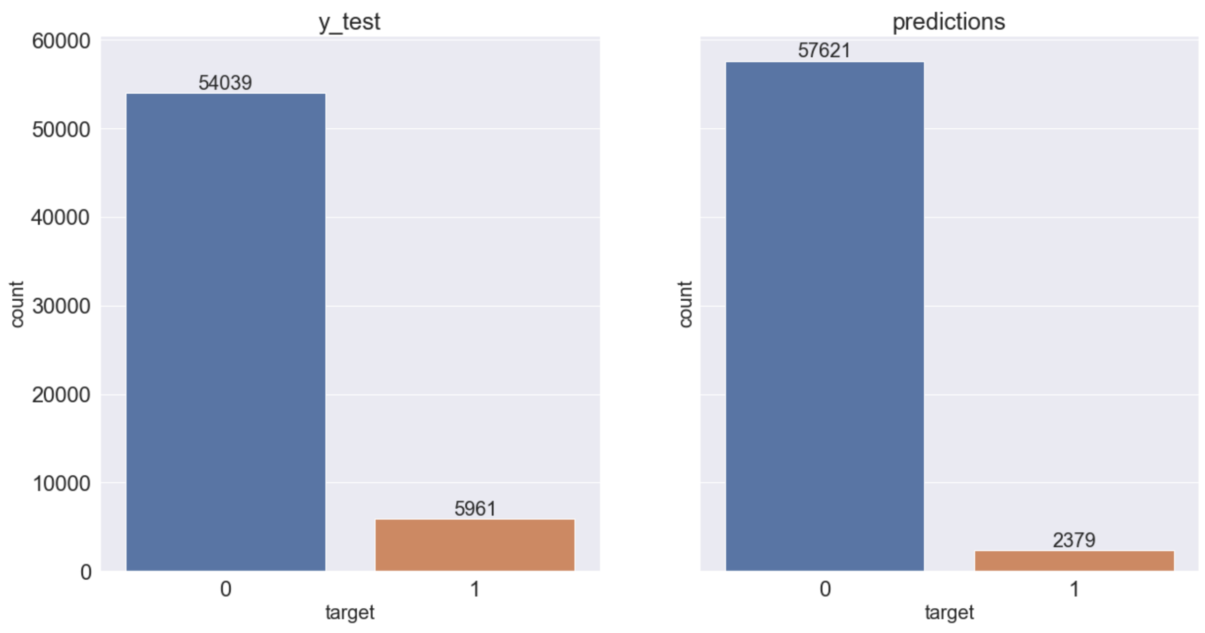
- Because, we want a supervised machine learning model (meaning we want to know the correct answers to judge if it is indeed a good model)
- Because, we can split the data into a train and test set. Where we'll use 70% of the data for training our model, and 30% of the data for testing our model accuracy.

Finally, we created a simple *LogisticRegression* model which shows the following output:

```
0.9136333333333333

             precision    recall  f1-score   support

           0       0.92      0.99      0.95     54039
           1       0.66      0.26      0.38      5961

    accuracy                           0.91     60000
   macro avg       0.79      0.63      0.67     60000
weighted avg       0.90      0.91      0.90     60000

Wall time: 2min 43s
```

Results of an early *LogisticRegression* regression:

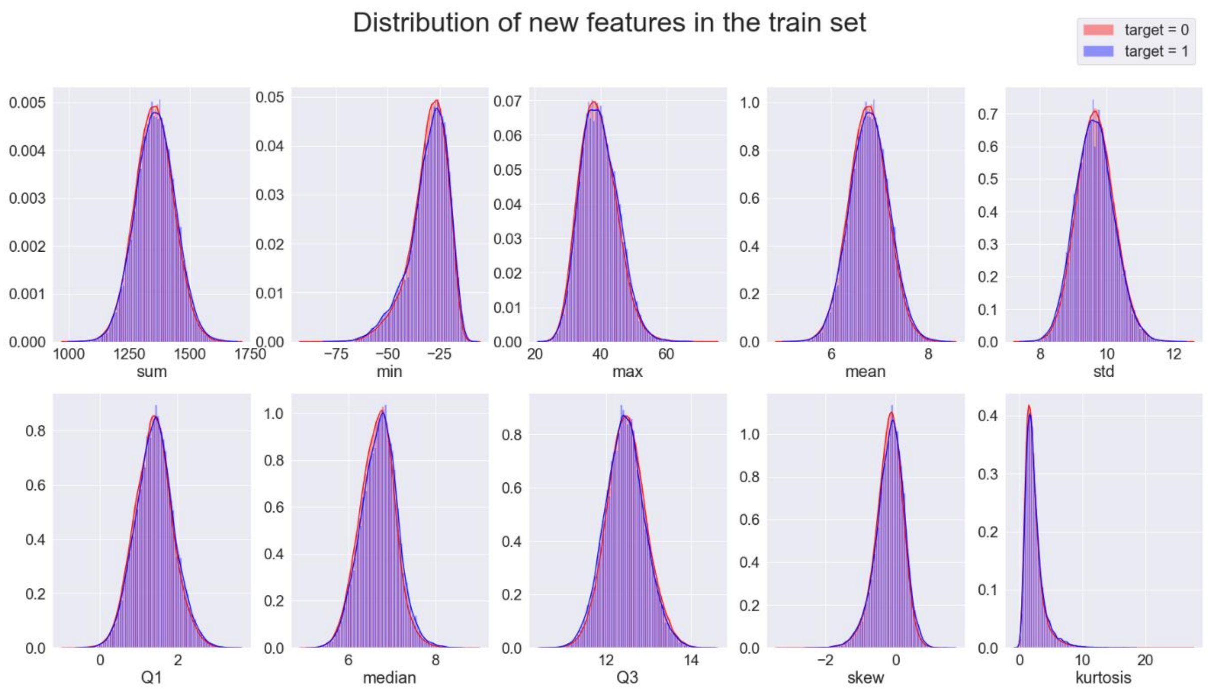
- The logistic regression model has an accuracy of 0.9136 or 91.3%, Why?
  - Only 3.8% of the predictions were guessed as 1 and y_test shows 10.2% to be 1
  - Logistic Regression algorithm cleverly decided to guess 0 most of the time
  - The 0's recall is very high, but 1's recall is dangerously low

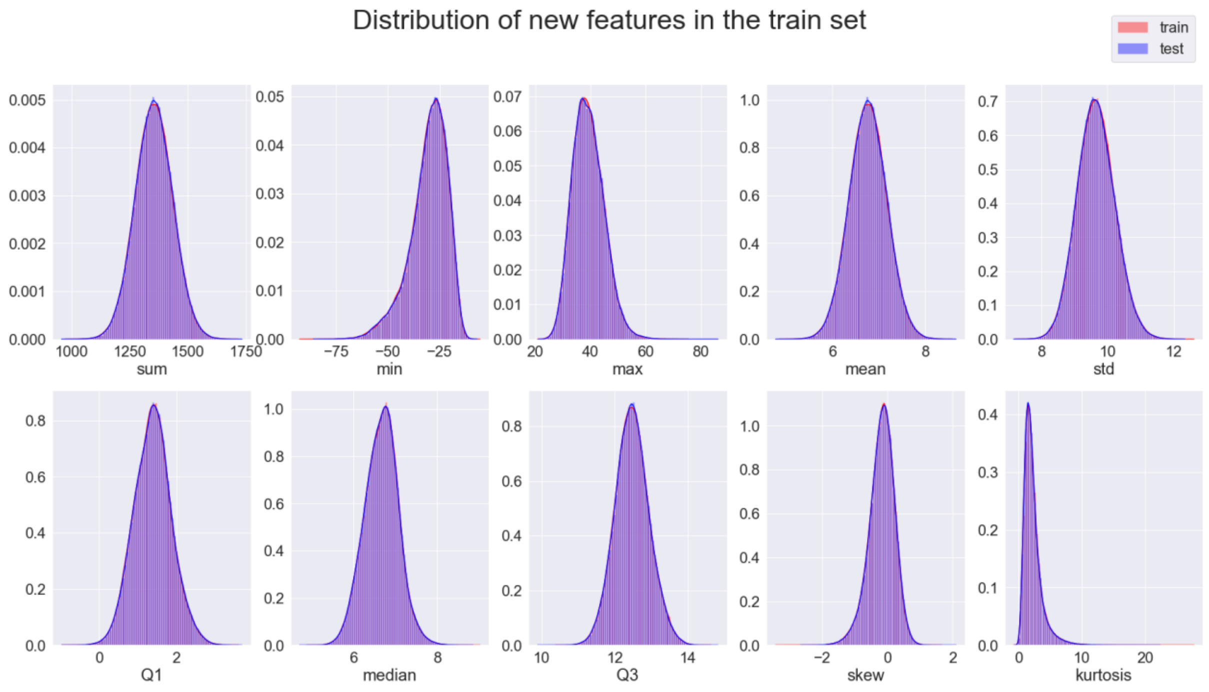


Feature Engineering:

Feature engineering is a process of transforming the given data into a form which is more separable for a Machine Learning model and it involves adding new features to each row in train and test data.

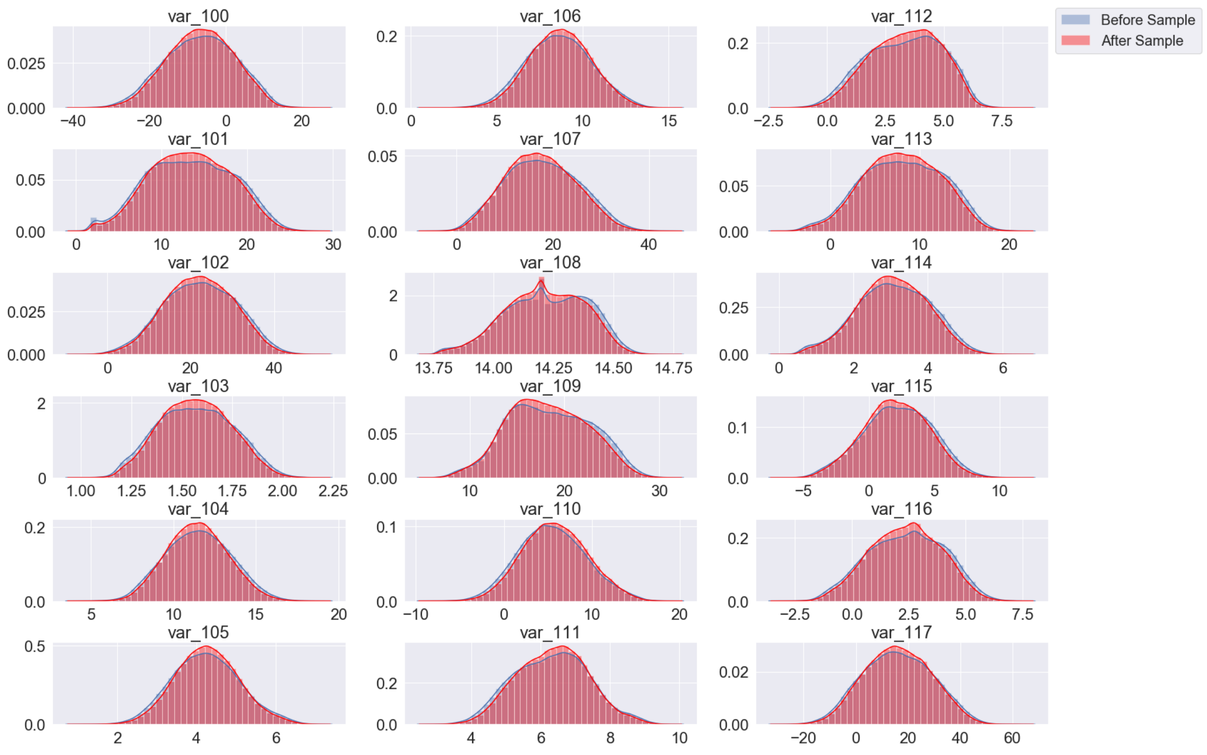
- We reviewed the distributions of our new features by TARGET
- We reviewed the distributions by Train and Test
- Tried scaling each variable via *StandardScaler*, *MinMaxScaler*, *Normalizer* as applicable
- Observed that there are minimal differences between 0 and 1 targets for newly generated features, but they add an extra unique element to each sample.
- For now, new features are worth adding to the dataset. Later, we will discover whether these features were of any importance.

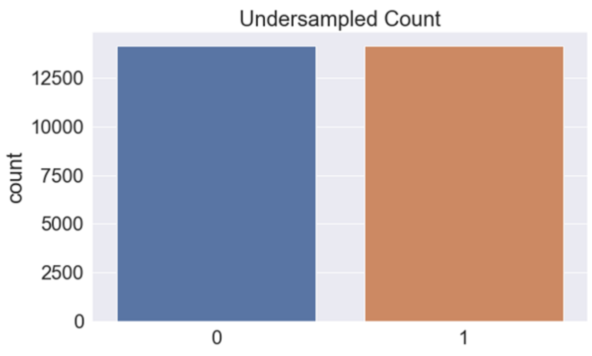


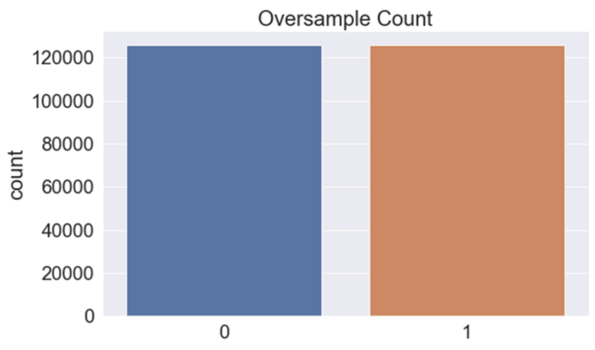


Under-sampling and Over-sampling

- Given the imbalanced Dataset, can we somehow make it balanced? Must use *X_train* data we have previously defined. Otherwise Our model will find exact matches when Oversampling.
- Under-sampling created an even distribution among target values; however, in this process we lost nearly 80% of the datapoints
- Oversampling gained several more features to the training data by nearly 80% distribution for each variable seems to be more concentrated around the mean of the distribution. (with some minor exceptions)
- SMOTE Oversampling was observed to be very effective.
- Subtle hints of red in the plots below indicate theses datasets are not the same.







Predictive Modelling

Predictive modelling is a process of using known results to create, process, and validate a model that can be used to forecast future outcomes. Each model is made up of several predictors, which are variables that are likely to influence future results.

Since, the problem is an attempt to correctly classify the TARGET feature, this is a classification problem, where we classify a given observation as having a target of either 0 or 1.

Model Classification Metrics used:

Following classification metrics have been chosen to evaluate various algorithms.

- *Accuracy*: Classification Accuracy is what we usually mean, when we use the term accuracy. It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- *Precision:* It is the number of correct positive results divided by the number of positive results predicted by the classifier.

- *Recall:* It is the number of correct positive results divided by the number of ***all*** relevant samples (all samples that should have been identified as positive).

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

- *F1-score:* F1 Score is the Harmonic Mean between *precision* and *recall*. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

  High precision but lower recall, gives you an extremely accurate, but it then misses several instances that are difficult to classify. The greater the F1-Score, the better is the performance of our model. Mathematically, it can be expressed as:

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

Classification models:

Logistic Regression: Logistic regression is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome measured with a dichotomous variable (in which there are only two possible outcomes). In logistic regression, the dependent variable is binary or dichotomous, i.e. it only contains data coded as 1 (TRUE, success, pregnant, etc.) or 0 (FALSE, failure, non-pregnant, etc.).

The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable, response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a *logit* transformation of the probability of presence of the characteristic of interest:

$$logit(p) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \ldots + b_kX_k$$

where p is the probability of presence of the characteristic of interest. The logit transformation is defined as the logged odds:

$$odds = \frac{p}{1-p} = \frac{probability\ of\ presence\ of\ characteristic}{probability\ of\ absence\ of\ characteristic}$$

and

$$logit(p) = \ln\left(\frac{p}{1-p}\right)$$

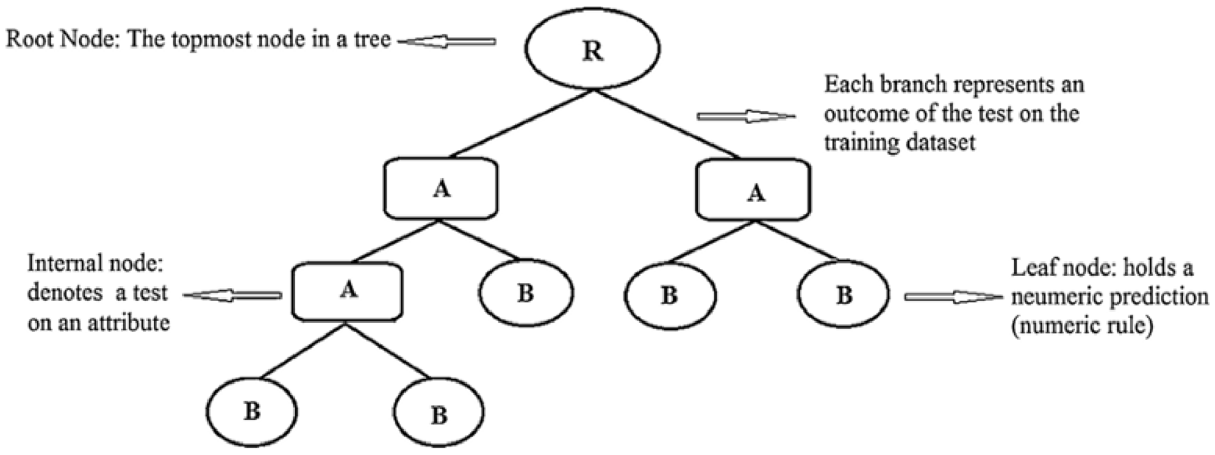
Rather than choosing parameters that minimize the sum of squared errors (like in ordinary regression), estimation in logistic regression chooses parameters that maximize the likelihood of observing the sample values.

We created a Logistic Regression model in Python using scikit-learn python package which shows the following output:

```
0.7831166666666667

              precision    recall  f1-score   support

           0       0.97      0.78      0.87     54039
           1       0.28      0.77      0.41      5961

    accuracy                           0.78     60000
   macro avg       0.63      0.78      0.64     60000
weighted avg       0.90      0.78      0.82     60000
```

Decision Tree: Decision tree is a machine learning algorithm having a pre-defined target variable that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

A decision tree is a flowchart-like structure in which each *internal node* represents a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each *branch* represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

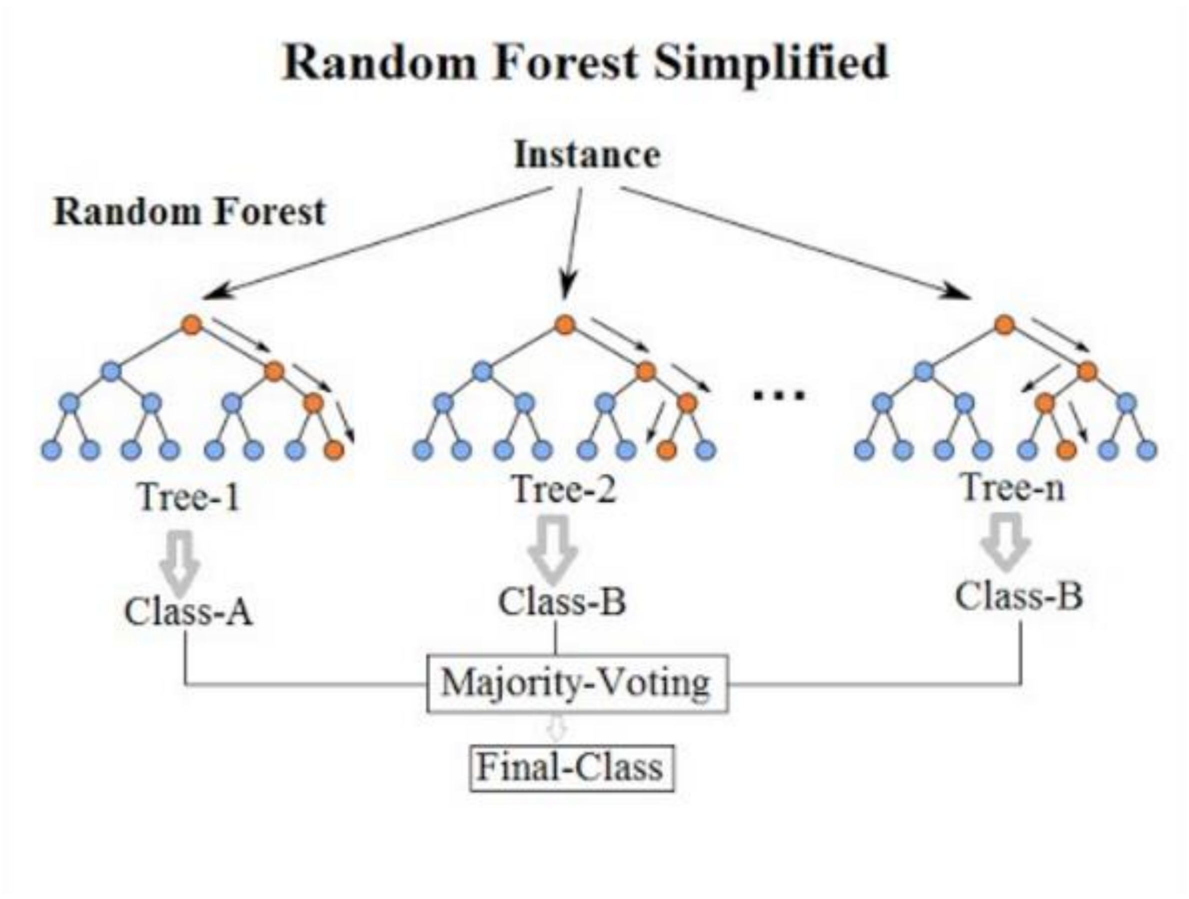

Root Node: The topmost node in a tree
R
Each branch represents an outcome of the test on the training dataset
A
A
Internal node: denotes a test on an attribute
A
B
B
B
Leaf node: holds a neumeric prediction (numeric rule)
B
B

We created a Decision Tree model in Python using scikit-learn python package which shows the following output:

```
0.8257333333333333

         precision   recall  f1-score   support

     0      0.91       0.90     0.90       54039
     1      0.16       0.17     0.17       5961

  accuracy                      0.83     60000
 macro avg     0.53     0.54     0.53     60000
weighted avg   0.83     0.83     0.83     60000
```

<u>Random Forest</u>: Random Forest, like its name implies, consists of many individual decision trees that operate as an ensemble, where each individual tree results in a class prediction and the class with the most votes becomes our model’s final prediction. The reason that the random forest model works so well is that, many relatively uncorrelated trees operating as a group will outperform any of the individual constituent tree.

The low correlation between models is the key to higher performance in Random Forest, because uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason behind this wonderful effect is that the trees protect each other from their individual errors. While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

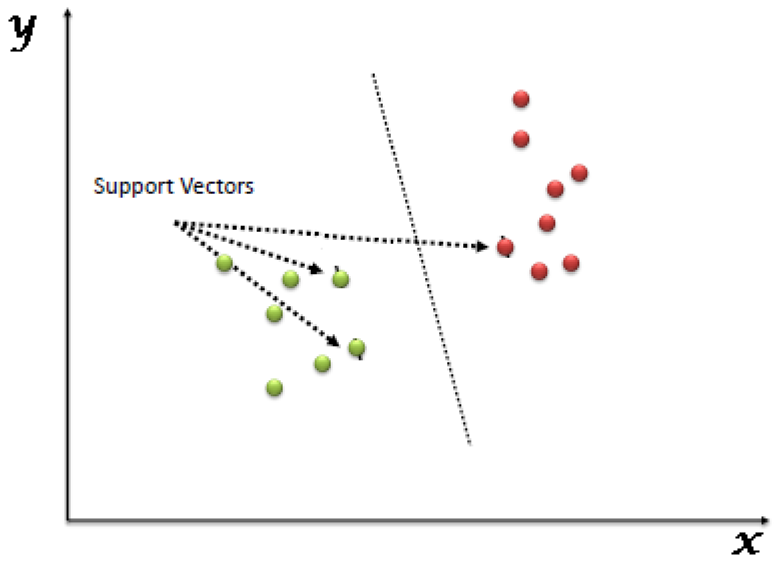


We created a Random Forest model with 50 trees in Python using scikit-learn python package which shows the following output:

```
0.90065

          precision    recall  f1-score   support

     0      0.90         1.00     0.95       54039
     1      0.00         0.00     0.00       5961

  accuracy                           0.90    60000
 macro avg       0.45     0.50    0.47    60000
weighted avg     0.81     0.90    0.85    60000
```

SVM (Linear Kernel): A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labelled training data (supervised learning), the algorithm outputs an optimal *hyperplane* which categorizes new examples. In two-dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

In SVM, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well. Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

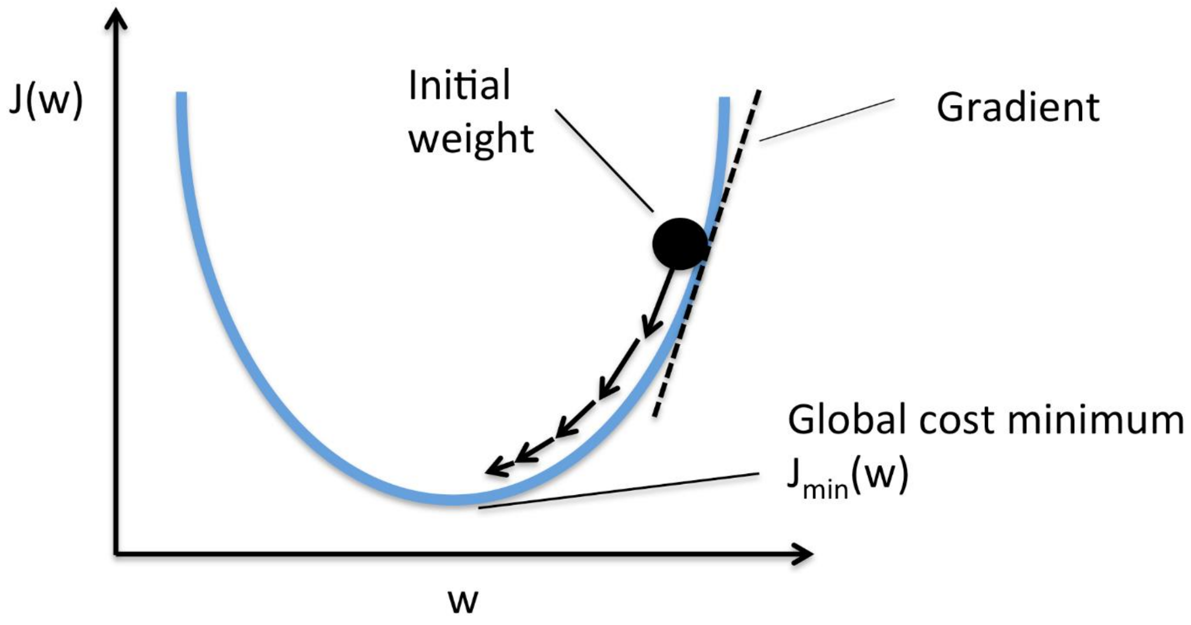


We created SVM (Linear Kernel) in Python using scikit-learn python package which shows the following output:

```
0.9136666666666666

          precision   recall   f1-score   support

       0      0.92         0.99     0.95        54039
       1      0.67        0.26     0.38        5961

   accuracy                            0.91     60000
  macro avg       0.80      0.62     0.66     60000
weighted avg     0.90      0.91     0.90     60000
```

SGDClassifier: “Gradient Descent” can be described as an iterative method which is used to find the values of the parameters of a function that minimizes the cost/loss function as much as possible. The parameters are initially defined a certain value and from that, Gradient Descent is run in an iterative fashion which uses calculus to find the optimal values of the parameters, or minimum possible value of the given cost/loss function. The word “stochastic” in Stochastic Gradient Descent (SGD) means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent (SGD), a few samples are selected randomly instead of the whole data set for each iteration.

SGDClassifier is a linear classifier that uses SGD for training, i.e. for finding the minima of the cost/loss function using SGD. The SGD estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated for each sample at a time and the model is updated along the way with a decreasing learning rate. For best results when using the default learning rate schedule, the data should have zero mean and unit variance.

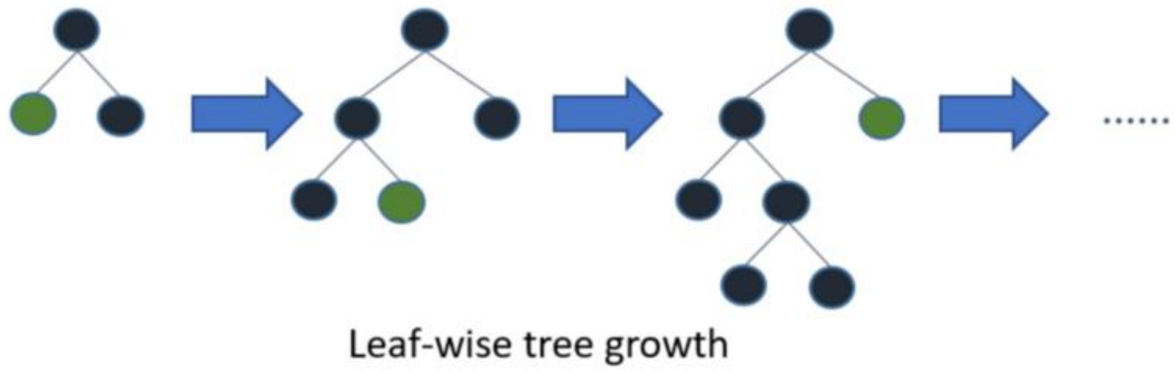


We created SVM (Linear Kernel) in Python using scikit-learn python package which shows the following output:

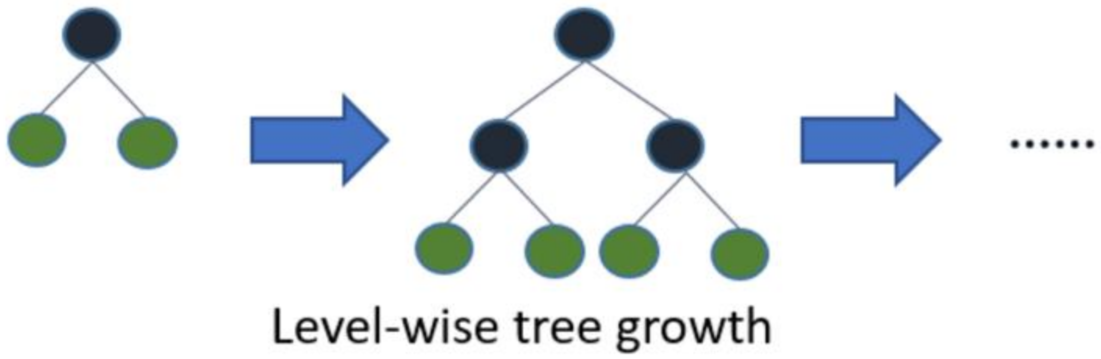
```
0.7616

             precision    recall     f1-score   support

       0      0.96         0.77       0.85       54039
       1      0.25         0.71       0.37       5961

  accuracy                            0.76     60000
 macro avg       0.61      0.74       0.61     60000
weighted avg     0.89      0.76       0.81     60000
```

LightGBM: Light GBM is a gradient boosting framework that uses tree-based learning algorithm. Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithms grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

The size of data is increasing day by day and it is becoming difficult for traditional data science algorithms to give faster results. Light GBM is prefixed as 'Light' because of its high speed. Light GBM can handle the large size of data and takes lower memory to run. Another reason of why Light GBM is popular is because it focuses on accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development. Below diagrams explain the implementation of LightGBM vis-a-vis other boosting algorithms.



Explains how LightGBM works



Other boosting algorithms

We created SVM (Linear Kernel) in Python using scikit-learn python package which shows the following output:

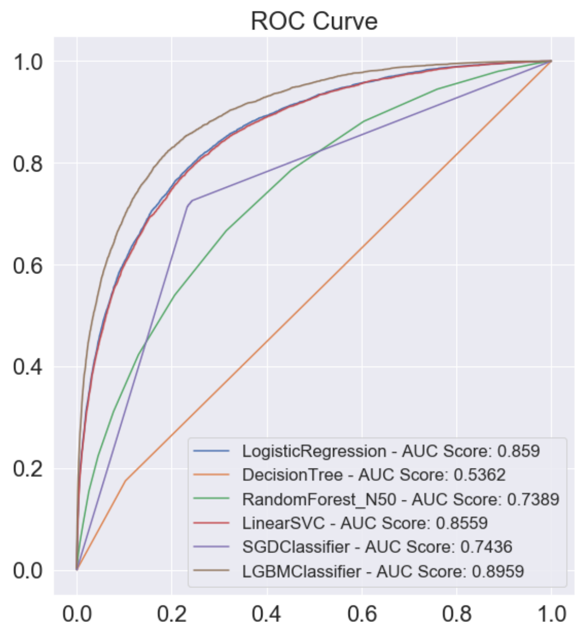
```
Training until validation scores don't improve for 200 rounds.
[1000]    training's auc: 0.865868        valid_1's auc: 0.846914
[2000]    training's auc: 0.893346        valid_1's auc: 0.871847
[3000]    training's auc: 0.906407        valid_1's auc: 0.882742
[4000]    training's auc: 0.914438        valid_1's auc: 0.888301
[5000]    training's auc: 0.920056        valid_1's auc: 0.891838
[6000]    training's auc: 0.924247        valid_1's auc: 0.893765
[7000]    training's auc: 0.927766        valid_1's auc: 0.894841
[8000]    training's auc: 0.931122        valid_1's auc: 0.895458
[9000]    training's auc: 0.934378        valid_1's auc: 0.895708
[10000]   training's auc: 0.937701        valid_1's auc: 0.895873
Early stopping, best iteration is:
[10094]   training's auc: 0.938005        valid_1's auc: 0.895897

0.8411
```

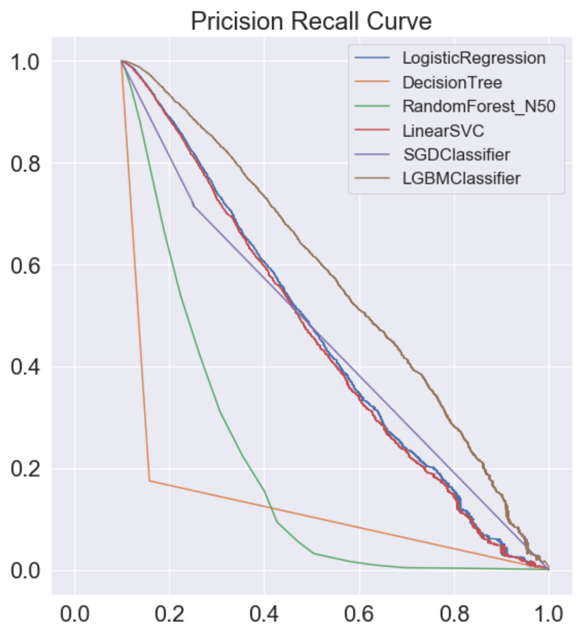
```
           precision   recall  f1-score  support

       0      0.97        0.85     0.91       54039
       1      0.36        0.77     0.49       5961

    accuracy                            0.84    60000
   macro avg       0.67      0.81     0.70    60000
weighted avg     0.91      0.84     0.86    60000
```

Conclusions:

Let's look the analysis at analysis of the Receiver Operating Characteristic (ROC), Area Under Curve(AUC) (called as ROC-AUC or sometimes referred to as AUC-ROC) which plots True Positive Rate (TPR) against False Positive Rate (FPR) where TPR is on y-axis and FPR is on the x-axis. The Precision Recall (PR AUC) curve plots Precision against Recall, for all classification algorithms used earlier and listed below.

- Logistic Regression
- Decision Tree
- Random Forest
- Linear SVC
- SGD Classifier
- Light GBM Classifier





ROC-AUC Analysis:

ROC-AUC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represent degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between patients with disease and no disease.

An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. When AUC is 0.5, it means model has no class separation capacity whatsoever.

Different classification models used measured in terms of AUC score can be ranked as follows:

LGBM Classifier (AUC: 0.8959) > Logistic Regression (AUC: 0.8590) > Linear SVM (AUC: 0.8559) > SGD Classifier (AUC: 0.7436) > Random Forest (AUC: 0.7389) > Decision Tree (AUC: 0.5362).

We can conclude that Light GBM Classifier is the best performing algorithm in terms of ROC-AUC score.

Precision-Recall Curve Analysis:

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. There is a trade-off between precision and recall which can be observed using the precision-recall curve, and an appropriate balance between the two obtained. A precision-recall curve can be noisy (a zigzag curve frequently going up and down) for small recall values. Therefore, precision-recall curves tend to cross each other much more frequently than ROC curves especially for small recall values.

We can conclude that Light GBM Classifier is the best performing algorithm in terms of trade-off between precision and recall.

Summary and Recommendations:

Santander Customer Transaction Prediction problem has a highly anonymized and redacted dataset most likely in favour of protecting the privacy of users involved. Also, there is almost no correlation between various variables and there is no central important set of features in this dataset. Under-sampling/over-sampling and feature engineering was done to deal with the class imbalance problem and to add uniqueness to the important features in training and test data.

Finally, considering the overall performance of 6 different classification models, the Light GBM model shows the best performance in terms of ROC-AUC and PR AUC which is the benchmarking classification metric for this project.

This project will help Santander bank to make better policy decisions regarding which of their customers makes certain specific choices regarding the nature of financial transaction that they are willing or unwilling to engage in.