

**Contents**

**1. Introduction**

[1.1 Problem Statement](#_Toc29919) 3

[1.2 Dataset](#_Toc29920) 3

[1.3 Exploratory Data Analysis](#_Toc29921) 3

[**2. Methodology**](#_Toc29922)

[2.1 Data Pre Processing](#_Toc29923) 9

[2.1.1 Missing Value Analysis](#_Toc29924) 9

[2.1.2 Feature Selection](#_Toc29925) 9

[2.1.3 Feature Engineering](#_Toc29926) 11

2.2 Splitting train and test Dataset 11

[2.2 Model Development](#_Toc29928) 12

[Logistic regression](#_Toc29929) 12

[KNN 13](#_Toc29930)

[Navie bayes 13](#_Toc29931)

[Random forest 14](#_Toc29932)

[Light GBM 14](#_Toc29933)

[**3. Coding**](#_Toc29937)

[4.1 Python Coding](#_Toc29939) 15

**1. Introduction**

##### **Problem Statement**

**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?

**Problem Statement** -

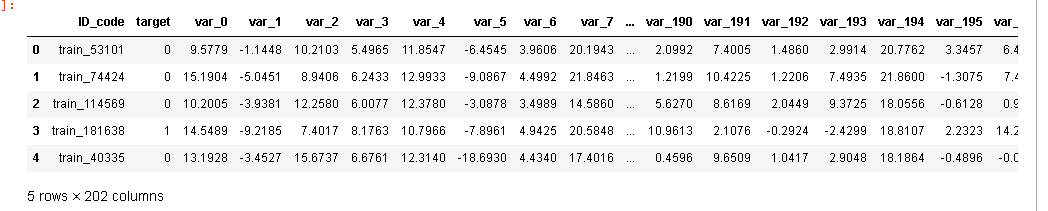
In this challenge, we need to identify which customers will make a specific transaction in

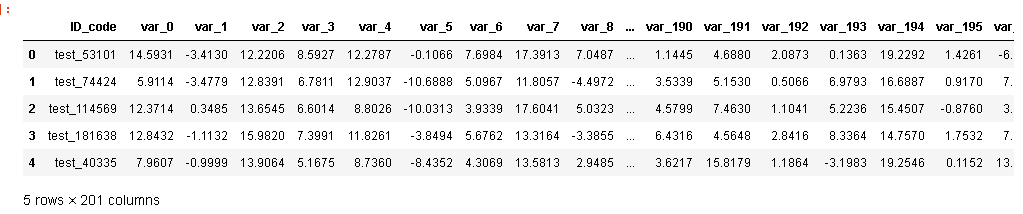
the future, irrespective of the amount of money transacted.

**1.2 Dataset**

We have train and test data sets:

Sample Dataset-





Attribute information

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.

### **1.3 Exploratory Data Analysis**

Any predictive modelling requires that we look at the data before we start modelling. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process, we will first try and look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable.

Exploratory Data Analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics. In the given data set there are 202 variables and 2 lakh Observations. Data types of all variables are object,float64 and int64. We take sample data from large data due to limited memory.

We start this process after importing the train\_cab.csv file in python jupyter notebook or

R Studio.Before importing the data, having a look on the excel file pf data can help us draw many conclusions.

After importing the csv file we start cleaning the data. We have also the the test.csv file along with it.

**Train contains:**

**ID\_code** (string);

**target**;

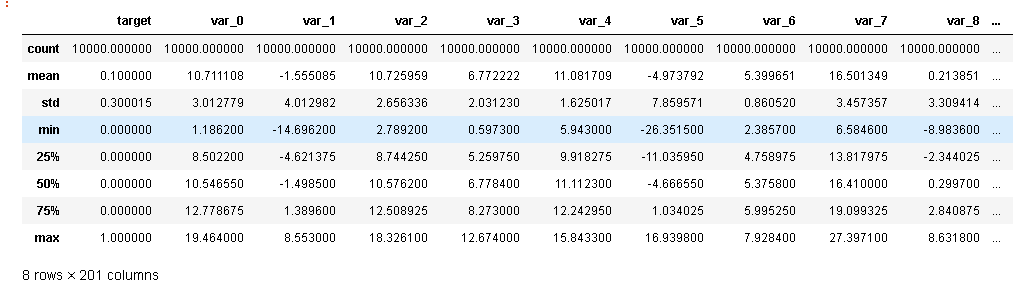
**200** numerical variables, named from **var\_0** to **var\_199**;

**Test contains:**

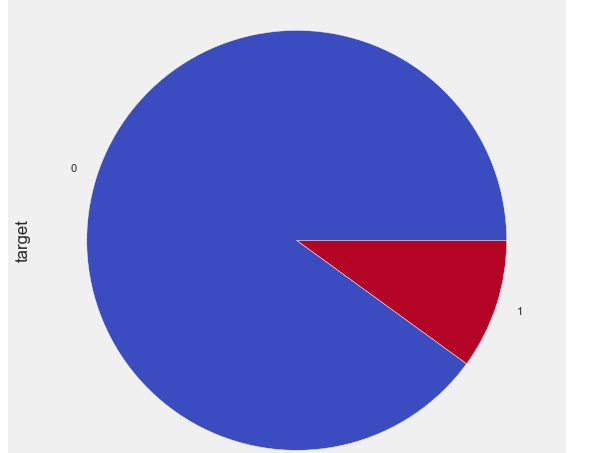
**ID\_code** (string);

**200** numerical variables, named from **var\_0** to **var\_199**;

The distribution of given attributes is like below:



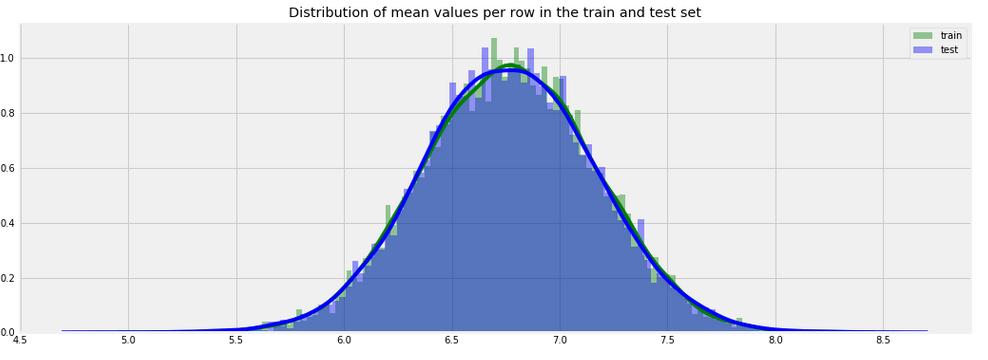
Distribution of Target variable



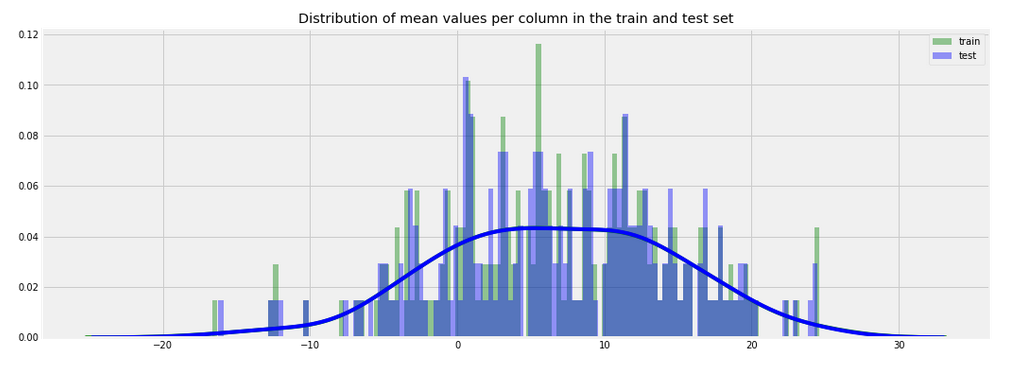
Here our data is suffering with imbalanced data set. We perform different techniques to improve classification performance in future steps.

Now we will see the Distribution of Variables

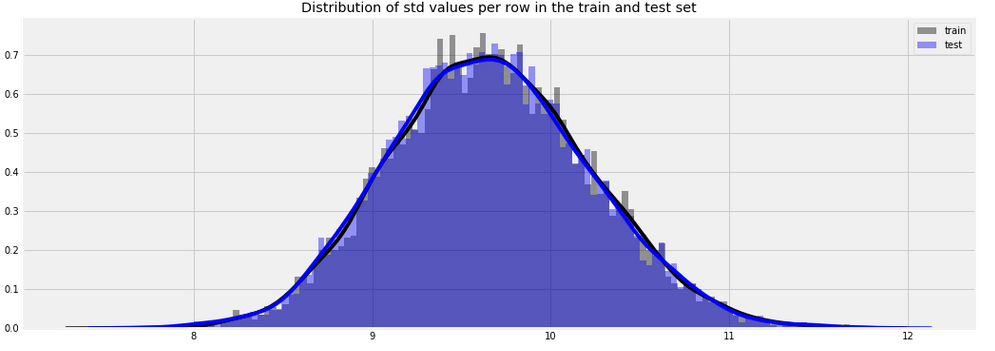
**Distribution of mean values per row in the train and test set**



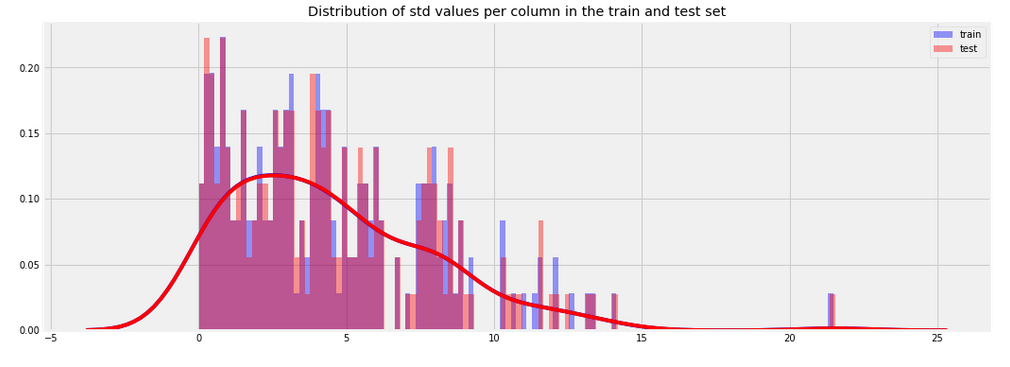
**Distribution of mean values per column in the train and test set**



**Distribution of std values per row in the train and test set**

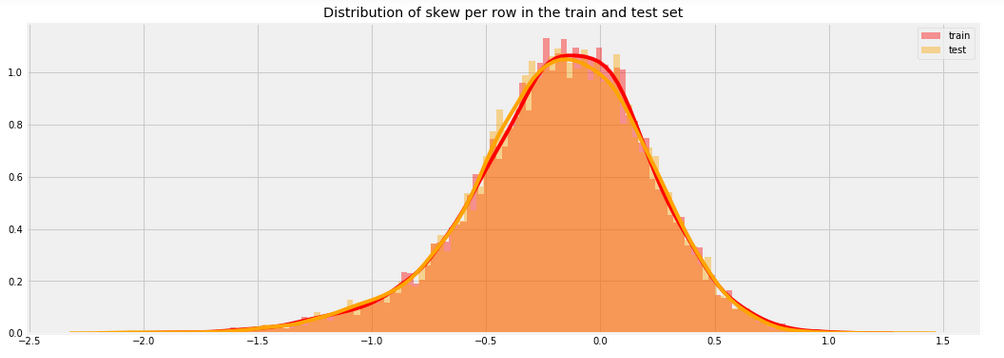


**Distribution of std values per column in the train and test set**

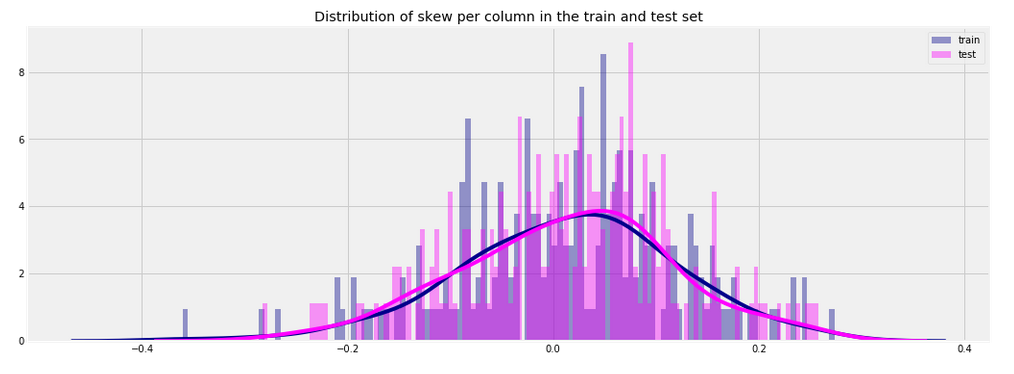


**Let's see now what is the distribution of skew values per rows and columns.**

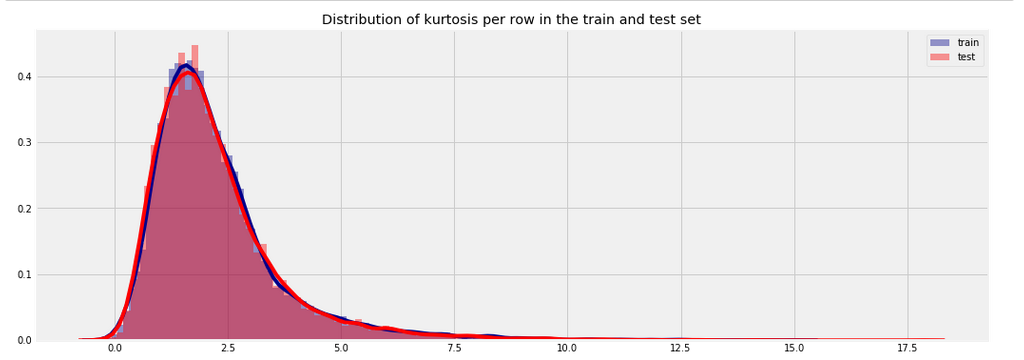
**Let's see first the distribution of skewness calculated per rows in train and test sets**



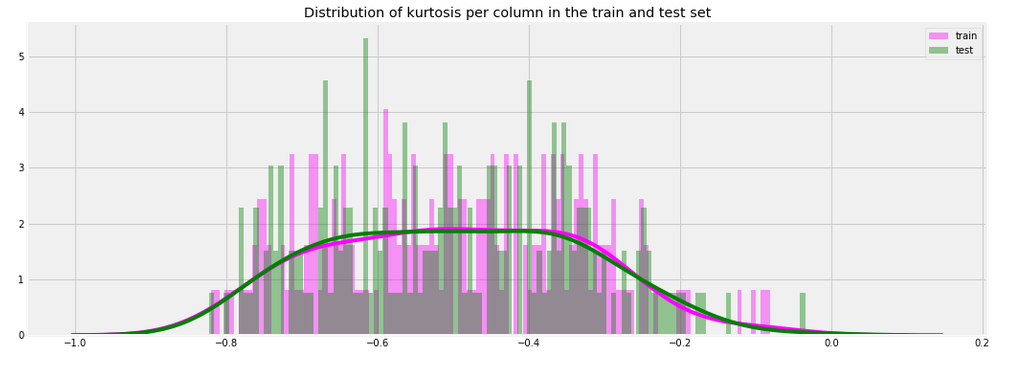
**Distribution of skew per column in the train and test set**



**Distribution of kurtosis per row in the train and test set**



**Distribution of kurtosis per column in the train and test set**

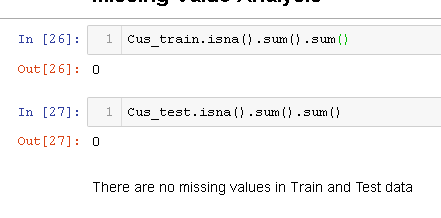


Both Skewness and Kurtosis show that the features distributions are like a normal one.

##### **2.1 Data Pre-Processing**

#### **2.1.1 Missing Value Analysis**

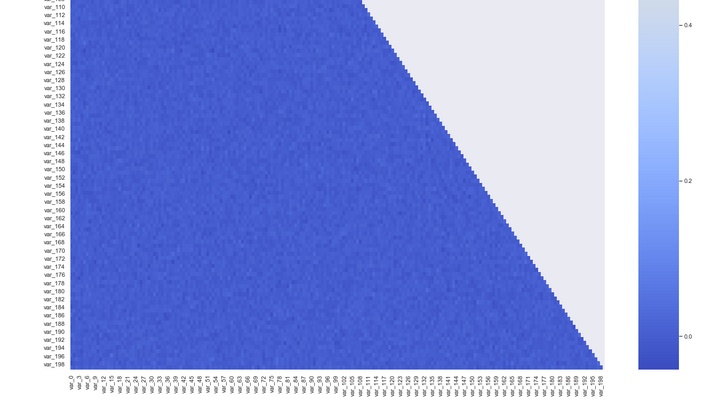
There are no missing values present in train and test data sets



**Feature Selection**

Before performing any type of modeling, we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of prediction. Selecting subset of relevant columns for the model construction is known as Feature Selection. We cannot use all the features because some features may be carrying the same information or irrelevant information which can increase overhead. To reduce overhead we adopt feature selection technique to extract meaningful features out of data. This in turn helps us to avoid the problem of multi collinearity.





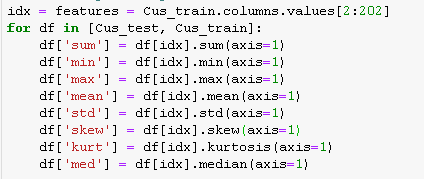
the above figure shows that most of the pearson correlations between the numerical data are close to zero, in fact is between 0 and 0.2. That means that most of the numerical data are almost uncorrelated between them.

**Feature Engineering**

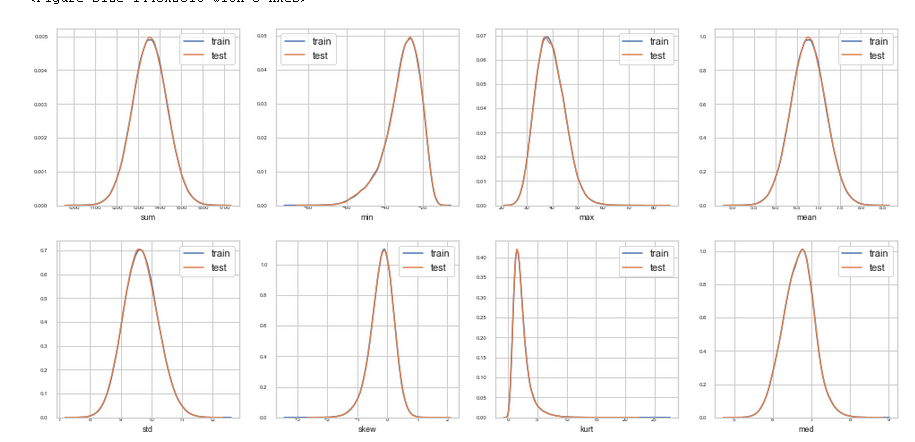
Feature Engineering is used to drive new features from existing features.

Let's calculate for starting few aggregated values for the existing features.

We create new feature like sum, minimum, maximum ,mean, standard deviation, skewness, kutosis ,median from our data.



**Distribution of New features**



**Splitting train and Validation Dataset**

1. a) We have used sklearn’s train\_test\_split() method to divide whole Dataset into train and validation datset.
2. b) 25% is in validation dataset and 75% is in training data.

**Oversample minority class:**

* It can be defined as adding more copies of minority class.
* It can be a good choice when we don't have a ton of data to work with.
* Drawback is that we are adding information.This may leads to overfitting and poor performance on test data.

**Undersample majority class:**

* It can be defined as removing some observations of the majority class.
* It can be a good choice when we have a ton of data -think million of rows.
* Drawback is that we are removing information that may be valuable.This may leads to underfitting and poor performance on test data.

Both Oversampling and undersampling techniques have some drawbacks. So, we are not going to use this models for this problem and also we will use other best algorithms.

**Random Oversampling Examples**

It creates a sample of synthetic data by enlarging the features space of minority and majority class examples.

##### **2.4 Model Development**

After Data pre-processing the next step is to develop a model using a train or historical data Which can perform to predict accurate result on test data or new data. Here we have tried with different model and will choose the model which will provide the most accurate values.

**Logistic Regression**

Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical.

Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratiolevel independent variables.

|  |  |  |
| --- | --- | --- |
| Parameter | R output | Python output |
| Accuracy | 1 | 77.52 |
| FNR | 30.67 | 29.55 |
| FPR | 25.8 | 21.7 |
| TPR | 96.8 | 70.5 |
| TNR | 78.6 | 78.3 |
| AUC | 0.94 | 0.84 |

**K- Nearest Neighbors :**

K-Nearest Neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

|  |  |  |
| --- | --- | --- |
| Parameter | R output | Python output |
| Accuracy | 53.15 | 77.56 |
| FNR | 68.57 | 85.15 |
| FPR | 26.09 | 15.24 |
| TPR | 53.48 | 14.54 |
| TNR | 73.67 | 84.57 |
| AUC | 0.53 | 0.51 |

In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors

**Naive Bayes:**

The Naïve Bayesian classifier is based on Bayes’ theorem with the independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods Algorithm:

Bayes theorem provides a way of calculating the posterior probability,P(c|x),fromP(c),P(x),and P(x|c). Naive Bayes classifier assumes that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.

We will implement all four models on our preprocessed data in both Python and R in

this chapter and then later on will select the final model

|  |  |  |
| --- | --- | --- |
| Parameter | R output | Python output |
| Accuracy | 74.35 | 80.96 |
| FNR | 28.34 | 28.34 |
| FPR | 23.56 | 18.65 |
| TPR | 74.67 | 71.67 |
| TNR | 76.14 | 81.34 |
| AUC | 0.74 | 0.86 |

**Random Forest**

Random Forest is an ensemble technique that consists of many decision trees. The idea behind Random Forest is to build n number of trees to have more accuracy in dataset. It is called random forest as we are building n number of trees randomly. In other words,

to build the decision trees it selects randomly n number of variables and n number of observations. It means to build each decision tree on random forest we are not going to use the same data. The higher no of trees in the random forest will give higher no of accuracy, so in random forest we can go for multiple trees. It can handle large no of independent variables without variable deletion and it will give the estimates that what variables are important.

|  |  |  |
| --- | --- | --- |
| Parameter | R output | Python output |
| Accuracy | 65.45 | 89.88 |
| FNR | 42..45 | 97.56 |
| FPR | 27 .45 | 0.49 |
| TPR | 66.34 | 2.02 |
| TNR | 72.56 | 97.98 |
| AUC | 0.65 | 0.63 |

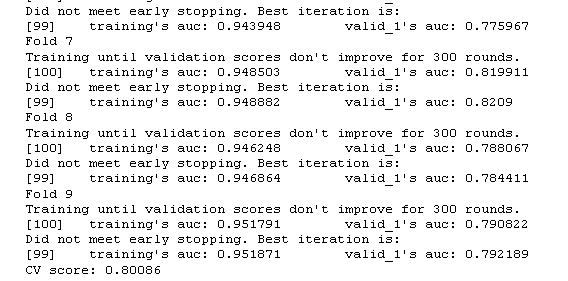
**Light GBM**

LightGBM is a gradient boosting framework that uses tree based learning algorithms. We are going to use LightGBM model.

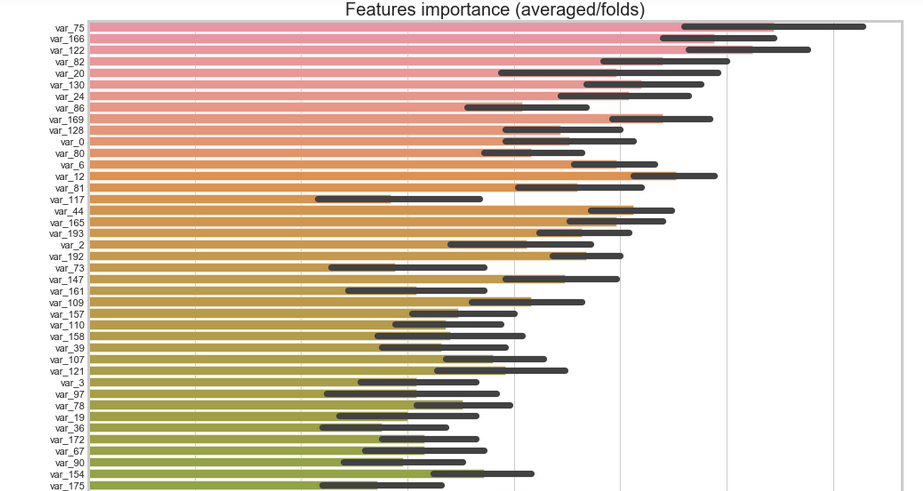
Will see **how it differs from other tree based algorithm.**

**Light GBM grows tree vertically**while other algorithm grows trees horizontally meaning that Light GBM grows tree **leaf-wise**while other algorithm 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.

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.



Feature Importance:



**4.Coding**

**4.1 Python Coding**

import os

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import warnings

from datetime import datetime

import calendar

from math import sin, cos, sqrt, atan2, radians,asin

import matplotlib.dates as mdates

import matplotlib as mpl

warnings.filterwarnings('ignore')

pd.set\_option('display.max\_colwidth', -1)

plt.style.use('fivethirtyeight')

from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import Imputer

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import StratifiedKFold

import lightgbm as lgb

from tqdm import tqdm\_notebook

from imblearn.over\_sampling import SMOTE

from sklearn.metrics import classification\_report

from sklearn.metrics import roc\_curve,auc,roc\_auc\_score

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.linear\_model import LogisticRegression

import gc

from collections import Counter

from sklearn.datasets import make\_classification

from imblearn.over\_sampling import RandomOverSampler

#Set working directory

os.chdir("C:/Data science/Project/Santander Customer Transaction")

os.getcwd()

'C:\\Data science\\Project\\Santander Customer Transaction'

#load the data

Cus\_train=pd.read\_csv("Santander\_sample\_train.csv")

Cus\_test=pd.read\_csv("Santander\_sample\_test.csv")

#Cus\_train=pd.read\_csv("train.csv")

#Cus\_test=pd.read\_csv("test.csv")

Cus\_train.head()

|  | **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** | **var\_197** | **var\_198** | **var\_199** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | train\_53101 | 0 | 9.5779 | -1.1448 | 10.2103 | 5.4965 | 11.8547 | -6.4545 | 3.9606 | 20.1943 | ... | 2.0992 | 7.4005 | 1.4860 | 2.9914 | 20.7762 | 3.3457 | 6.4024 | 8.5164 | 18.7698 | -11.6723 |
| **1** | train\_74424 | 0 | 15.1904 | -5.0451 | 8.9406 | 6.2433 | 12.9933 | -9.0867 | 4.4992 | 21.8463 | ... | 1.2199 | 10.4225 | 1.2206 | 7.4935 | 21.8600 | -1.3075 | 7.4112 | 10.6903 | 18.0640 | -1.2149 |
| **2** | train\_114569 | 0 | 10.2005 | -3.9381 | 12.2580 | 6.0077 | 12.3780 | -3.0878 | 3.4989 | 14.5860 | ... | 5.6270 | 8.6169 | 2.0449 | 9.3725 | 18.0556 | -0.6128 | 0.9046 | 9.6364 | 14.7799 | 11.6834 |
| **3** | train\_181638 | 1 | 14.5489 | -9.2185 | 7.4017 | 8.1763 | 10.7966 | -7.8961 | 4.9425 | 20.5848 | ... | 10.9613 | 2.1076 | -0.2924 | -2.4299 | 18.8107 | 2.2323 | 14.2430 | 8.4075 | 12.9844 | 2.5465 |
| **4** | train\_40335 | 0 | 13.1928 | -3.4527 | 15.6737 | 6.6761 | 12.3140 | -18.6930 | 4.4340 | 17.4016 | ... | 0.4596 | 9.6509 | 1.0417 | 2.9048 | 18.1864 | -0.4896 | -0.0624 | 9.5451 | 18.6588 | -19.0305 |

5 rows × 202 columns

Cus\_test.head()

|  | **ID\_code** | **var\_0** | **var\_1** | **var\_2** | **var\_3** | **var\_4** | **var\_5** | **var\_6** | **var\_7** | **var\_8** | **...** | **var\_190** | **var\_191** | **var\_192** | **var\_193** | **var\_194** | **var\_195** | **var\_196** | **var\_197** | **var\_198** | **var\_199** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | test\_53101 | 14.5931 | -3.4130 | 12.2206 | 8.5927 | 12.2787 | -0.1066 | 7.6984 | 17.3913 | 7.0487 | ... | 1.1445 | 4.6880 | 2.0873 | 0.1363 | 19.2292 | 1.4261 | -6.4313 | 9.0598 | 12.9742 | -3.1827 |
| **1** | test\_74424 | 5.9114 | -3.4779 | 12.8391 | 6.7811 | 12.9037 | -10.6888 | 5.0967 | 11.8057 | -4.4972 | ... | 3.5339 | 5.1530 | 0.5066 | 6.9793 | 16.6887 | 0.9170 | 7.1355 | 9.7038 | 13.1813 | 5.4344 |
| **2** | test\_114569 | 12.3714 | 0.3485 | 13.6545 | 6.6014 | 8.8026 | -10.0313 | 3.9339 | 17.6041 | 5.0323 | ... | 4.5799 | 7.4630 | 1.1041 | 5.2236 | 15.4507 | -0.8760 | 3.1354 | 9.3337 | 9.0674 | 14.2244 |
| **3** | test\_181638 | 12.8432 | -1.1132 | 15.9820 | 7.3991 | 11.8261 | -3.8494 | 5.6762 | 13.3164 | -3.3855 | ... | 6.4316 | 4.5648 | 2.8416 | 8.3364 | 14.7570 | 1.7532 | 7.8354 | 8.2817 | 12.5065 | 8.8211 |
| **4** | test\_40335 | 7.9607 | -0.9999 | 13.9064 | 5.1675 | 8.7360 | -8.4352 | 4.3069 | 13.5813 | 2.9485 | ... | 3.6217 | 15.8179 | 1.1864 | -3.1983 | 19.2546 | 0.1152 | 13.9836 | 7.9041 | 11.8314 | -3.1253 |

5 rows × 201 columns

Cus\_train.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10000 entries, 0 to 9999

Columns: 202 entries, ID\_code to var\_199

dtypes: float64(200), int64(1), object(1)

memory usage: 15.4+ MB

Cus\_test.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10000 entries, 0 to 9999

Columns: 201 entries, ID\_code to var\_199

dtypes: float64(200), object(1)

memory usage: 15.3+ MB

Cus\_train.describe()

|  | **target** | **var\_0** | **var\_1** | **var\_2** | **var\_3** | **var\_4** | **var\_5** | **var\_6** | **var\_7** | **var\_8** | **...** | **var\_190** | **var\_191** | **var\_192** | **var\_193** | **var\_194** | **var\_195** | **var\_196** | **var\_197** | **var\_198** | **var\_199** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | ... | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 |
| **mean** | 0.100000 | 10.711108 | -1.555085 | 10.725959 | 6.772222 | 11.081709 | -4.973792 | 5.399651 | 16.501349 | 0.213851 | ... | 3.220615 | 7.419514 | 1.923874 | 3.348032 | 17.967702 | -0.125835 | 2.325853 | 8.904356 | 15.889900 | -3.261658 |
| **std** | 0.300015 | 3.012779 | 4.012982 | 2.656336 | 2.031230 | 1.625017 | 7.859571 | 0.860520 | 3.457357 | 3.309414 | ... | 4.557838 | 3.024710 | 1.484084 | 4.008411 | 3.155545 | 1.440739 | 5.448067 | 0.928939 | 2.995748 | 10.367058 |
| **min** | 0.000000 | 1.186200 | -14.696200 | 2.789200 | 0.597300 | 5.943000 | -26.351500 | 2.385700 | 6.584600 | -8.983600 | ... | -13.454700 | -0.937600 | -3.515900 | -9.887000 | 9.649200 | -4.433200 | -13.508400 | 6.324800 | 6.558700 | -37.696200 |
| **25%** | 0.000000 | 8.502200 | -4.621375 | 8.744250 | 5.259750 | 9.918275 | -11.035950 | 4.758975 | 13.817975 | -2.344025 | ... | -0.042125 | 5.144700 | 0.879050 | 0.576950 | 15.606375 | -1.169225 | -1.822000 | 8.238075 | 13.859650 | -11.188925 |
| **50%** | 0.000000 | 10.546550 | -1.498500 | 10.576200 | 6.778400 | 11.112300 | -4.666550 | 5.375800 | 16.410000 | 0.299700 | ... | 3.135800 | 7.357400 | 1.878100 | 3.404250 | 17.935950 | -0.177600 | 2.447400 | 8.881500 | 15.910200 | -2.745050 |
| **75%** | 0.000000 | 12.778675 | 1.389600 | 12.508925 | 8.273000 | 12.242950 | 1.034025 | 5.995250 | 19.099325 | 2.840875 | ... | 6.414200 | 9.492425 | 2.952500 | 6.239000 | 20.376700 | 0.862725 | 6.546850 | 9.594850 | 18.071700 | 4.917875 |
| **max** | 1.000000 | 19.464000 | 8.553000 | 18.326100 | 12.674000 | 15.843300 | 16.939800 | 7.928400 | 27.397100 | 8.631800 | ... | 17.342600 | 16.289600 | 7.611600 | 16.237200 | 26.671200 | 4.082900 | 14.572400 | 11.832900 | 25.685200 | 23.876400 |

8 rows × 201 columns

Cus\_test.describe()

|  | **var\_0** | **var\_1** | **var\_2** | **var\_3** | **var\_4** | **var\_5** | **var\_6** | **var\_7** | **var\_8** | **var\_9** | **...** | **var\_190** | **var\_191** | **var\_192** | **var\_193** | **var\_194** | **var\_195** | **var\_196** | **var\_197** | **var\_198** | **var\_199** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | ... | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 |
| **mean** | 10.705713 | -1.579591 | 10.725522 | 6.790928 | 11.062078 | -5.071369 | 5.404619 | 16.541806 | 0.226165 | 7.560088 | ... | 3.203806 | 7.449769 | 1.939969 | 3.321719 | 18.033297 | -0.141370 | 2.362241 | 8.905582 | 15.931850 | -3.313788 |
| **std** | 3.051148 | 4.081132 | 2.650834 | 2.067572 | 1.609057 | 7.893001 | 0.865800 | 3.434089 | 3.323748 | 1.240798 | ... | 4.546520 | 3.049154 | 1.493199 | 4.014542 | 3.126021 | 1.441264 | 5.453217 | 0.920792 | 2.987404 | 10.468773 |
| **min** | 1.482700 | -14.321300 | 3.436800 | -0.022400 | 5.989200 | -27.191600 | 2.846600 | 7.631000 | -8.657000 | 4.243300 | ... | -11.927200 | -1.462700 | -2.874200 | -9.026000 | 9.849500 | -4.325900 | -13.610100 | 6.418300 | 7.436800 | -34.605000 |
| **25%** | 8.500175 | -4.694100 | 8.749125 | 5.212250 | 9.893650 | -11.186875 | 4.763100 | 13.971400 | -2.357575 | 6.610875 | ... | -0.105950 | 5.163250 | 0.875775 | 0.610175 | 15.668125 | -1.180275 | -1.962475 | 8.246300 | 13.861500 | -11.381875 |
| **50%** | 10.502550 | -1.522900 | 10.578200 | 6.823150 | 11.063600 | -4.969600 | 5.381450 | 16.430700 | 0.256400 | 7.604450 | ... | 3.176250 | 7.335800 | 1.911850 | 3.430900 | 17.988600 | -0.165900 | 2.517850 | 8.881450 | 16.014850 | -2.886250 |
| **75%** | 12.772125 | 1.445900 | 12.546700 | 8.361975 | 12.218500 | 0.910800 | 5.988575 | 19.094250 | 2.878350 | 8.581325 | ... | 6.365175 | 9.531725 | 2.967300 | 6.187375 | 20.409700 | 0.832525 | 6.612600 | 9.589275 | 18.094625 | 4.914850 |
| **max** | 20.064900 | 8.752900 | 18.714100 | 12.947600 | 16.037100 | 17.212700 | 7.995400 | 28.292800 | 8.756900 | 10.913000 | ... | 15.633800 | 16.288000 | 7.001400 | 16.590900 | 26.681500 | 3.942500 | 15.862700 | 11.981700 | 24.307000 | 27.531900 |

8 rows × 200 columns

**Exploratory Data Analysis**

At first glance we have many uncharacterized numerical features, their names has the prefix "var\_" and they are 200 in numbers. There are so many variables that some histograms will shed light to their numerical appearance.

Cus\_train.target.value\_counts()

0 9000

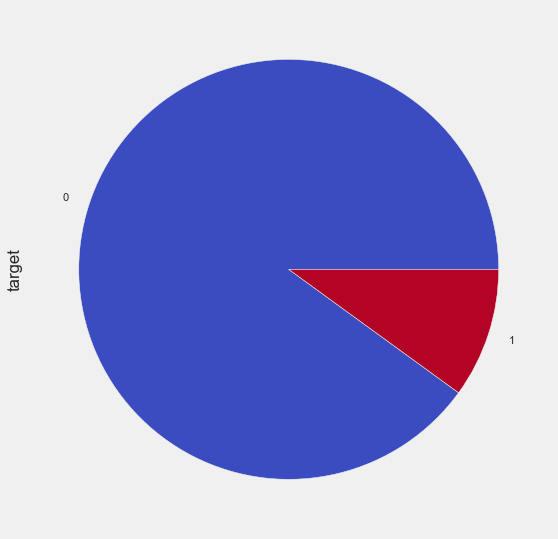
1 1000

Name: target, dtype: int64

#checking the target variable distribution with pie chart

Cus\_train['target'].value\_counts().plot(kind="pie", figsize=(12,9), colormap="coolwarm")

<matplotlib.axes.\_subplots.AxesSubplot at 0x10bcaba8>



Here we have imbalanced data set

numerical\_features = Cus\_train.columns[2:]

#checking the distribution of the variables

print('Distributions columns')

plt.figure(figsize=(30, 185))

for i, col in enumerate(numerical\_features):

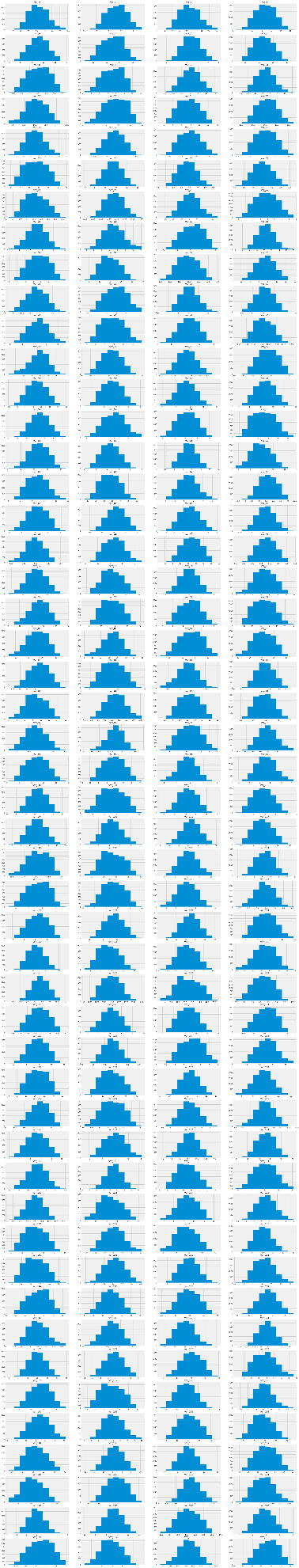
plt.subplot(50, 4, i + 1)

plt.hist(Cus\_train[col])

plt.title(col)

gc.collect();

Distributions columns



Almost all features shows a normal distribution shape. Lets see the distributions for all numerical features per each class.

print('Distributions columns')

plt.figure(figsize=(30, 185))

for i, col in enumerate(numerical\_features):

plt.subplot(50, 4, i + 1)

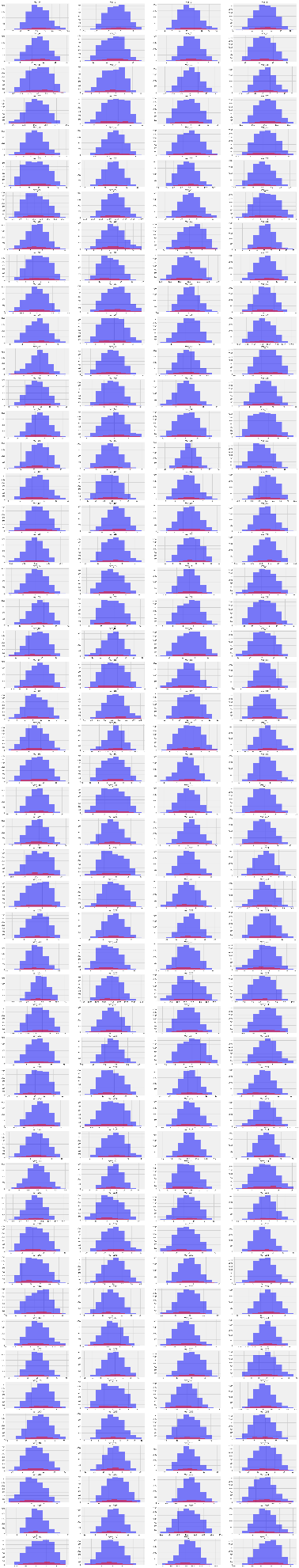
plt.hist(Cus\_train[Cus\_train["target"] == 0][col], alpha=0.5, label='0', color='b')

plt.hist(Cus\_train[Cus\_train["target"] == 1][col], alpha=0.5, label='1', color='r')

plt.title(col)

gc.collect();

Distributions columns



plt.figure(figsize=(16,6))

features = Cus\_train.columns.values[2:202]

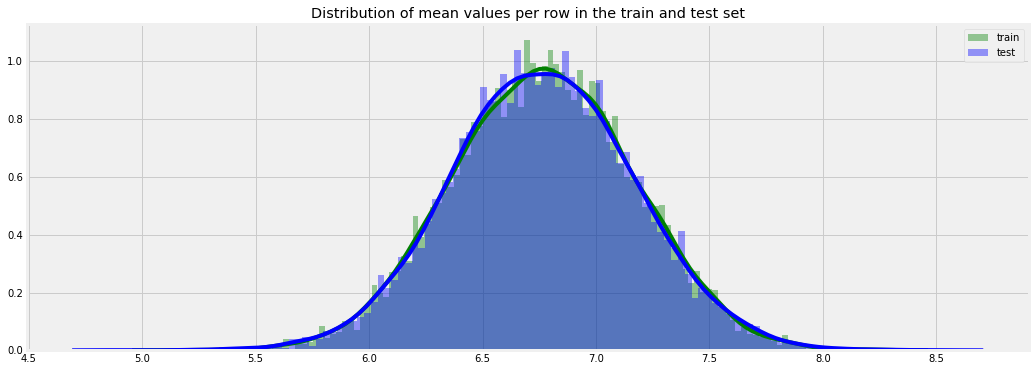
plt.title("Distribution of mean values per row in the train and test set")

sns.distplot(Cus\_train[features].mean(axis=1),color="green", kde=True,bins=120, label='train')

sns.distplot(Cus\_test[features].mean(axis=1),color="blue", kde=True,bins=120, label='test')

plt.legend()

plt.show()



plt.figure(figsize=(16,6))

features = Cus\_train.columns.values[2:202]

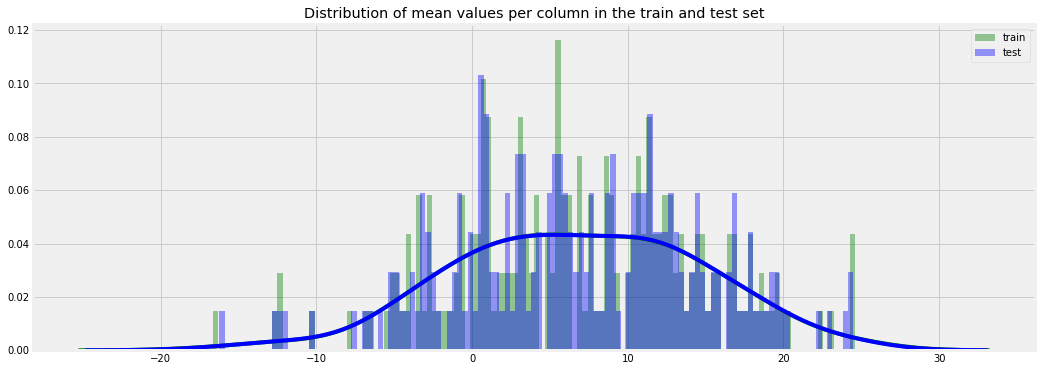
plt.title("Distribution of mean values per column in the train and test set")

sns.distplot(Cus\_train[features].mean(axis=0),color="green", kde=True,bins=120, label='train')

sns.distplot(Cus\_test[features].mean(axis=0),color="blue", kde=True,bins=120, label='test')

plt.legend()

plt.show()



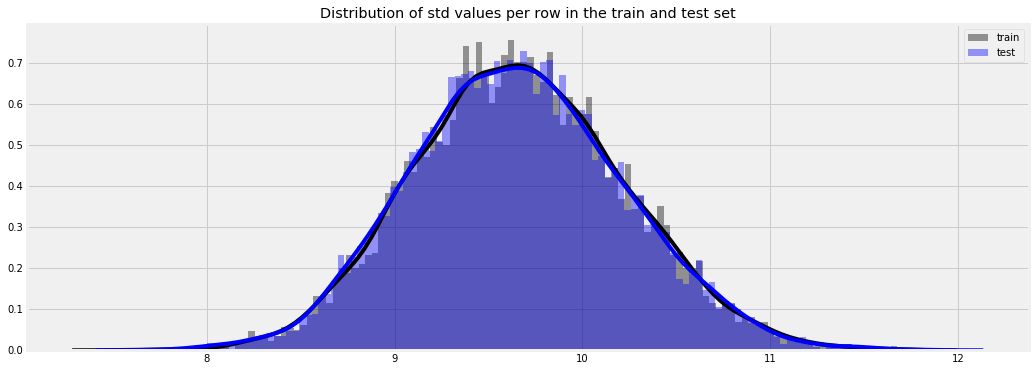
plt.figure(figsize=(16,6))

plt.title("Distribution of std values per row in the train and test set")

sns.distplot(Cus\_train[features].std(axis=1),color="black", kde=True,bins=120, label='train')

sns.distplot(Cus\_test[features].std(axis=1),color="blue", kde=True,bins=120, label='test')

plt.legend();plt.show()



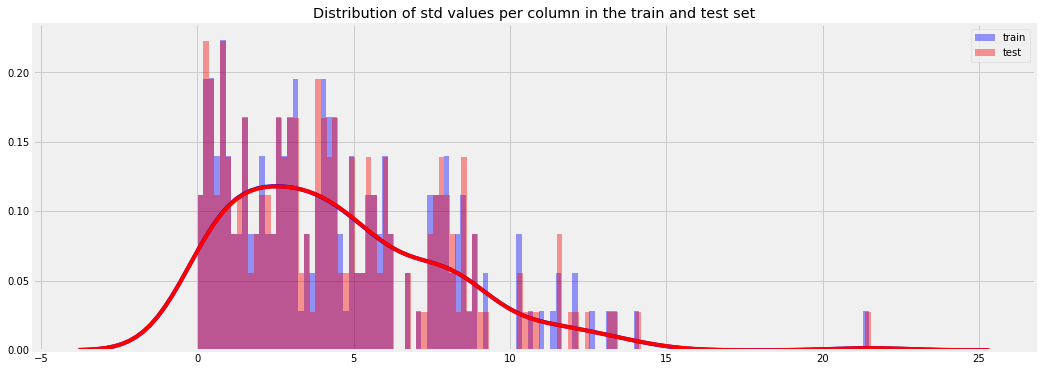
plt.figure(figsize=(16,6))

plt.title("Distribution of std values per column in the train and test set")

sns.distplot(Cus\_train[features].std(axis=0),color="blue", kde=True,bins=120, label='train')

sns.distplot(Cus\_test[features].std(axis=0),color="red", kde=True,bins=120, label='test')

plt.legend();plt.show()



Most of the distributions show small std.deviations, and very few more than 20. Maybe a log transformation or a scaling technique to all features will alter the graph above to a normal one.

**Distribution of skew and kurtosis**

Let's see now what is the distribution of skew values per rows and columns.

Let's see first the distribution of skewness calculated per rows in train and test sets

plt.figure(figsize=(16,6))

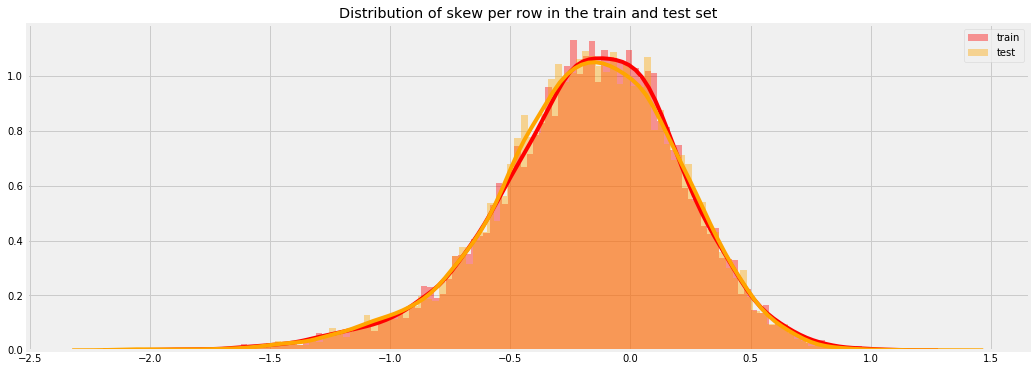
plt.title("Distribution of skew per row in the train and test set")

sns.distplot(Cus\_train[features].skew(axis=1),color="red", kde=True,bins=120, label='train')

sns.distplot(Cus\_test[features].skew(axis=1),color="orange", kde=True,bins=120, label='test')

plt.legend()

plt.show()



plt.figure(figsize=(16,6))

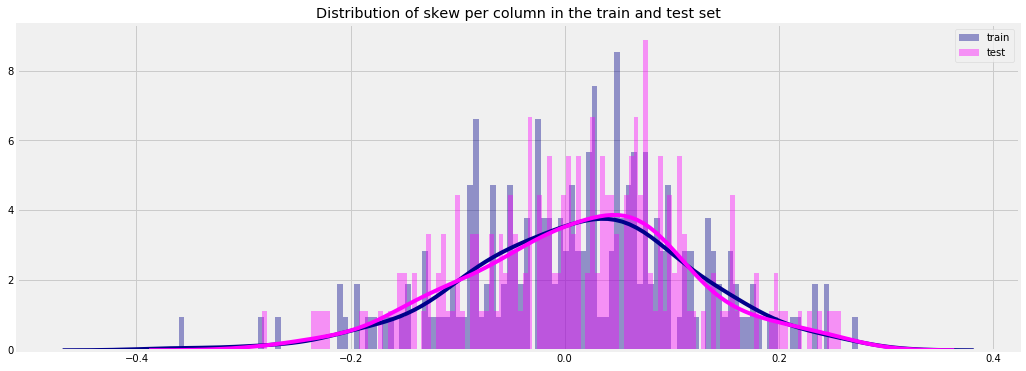
plt.title("Distribution of skew per column in the train and test set")

sns.distplot(Cus\_train[features].skew(axis=0),color="darkblue", kde=True,bins=120, label='train')

sns.distplot(Cus\_test[features].skew(axis=0),color="magenta", kde=True,bins=120, label='test')

plt.legend()

plt.show()



Let's see now what is the distribution of kurtosis values per rows and columns.

Let's see first the distribution of kurtosis calculated per rows in train and test sets.

plt.figure(figsize=(16,6))

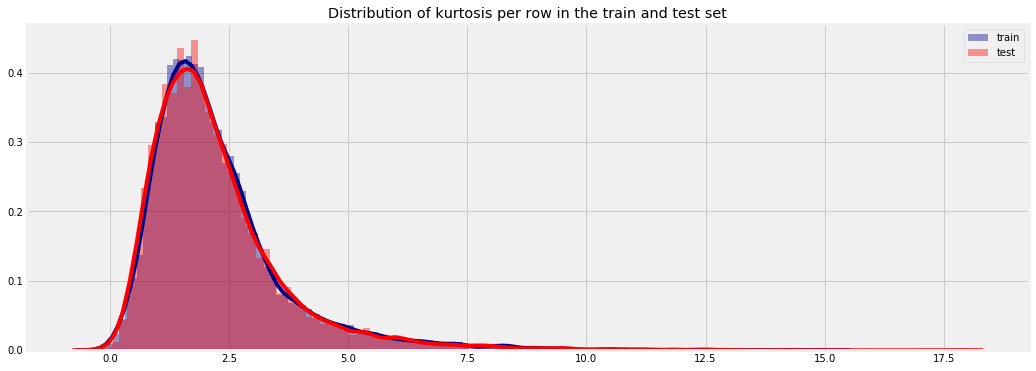
plt.title("Distribution of kurtosis per row in the train and test set")

sns.distplot(Cus\_train[features].kurtosis(axis=1),color="darkblue", kde=True,bins=120, label='train')

sns.distplot(Cus\_test[features].kurtosis(axis=1),color="red", kde=True,bins=120, label='test')

plt.legend()

plt.show()



plt.figure(figsize=(16,6))

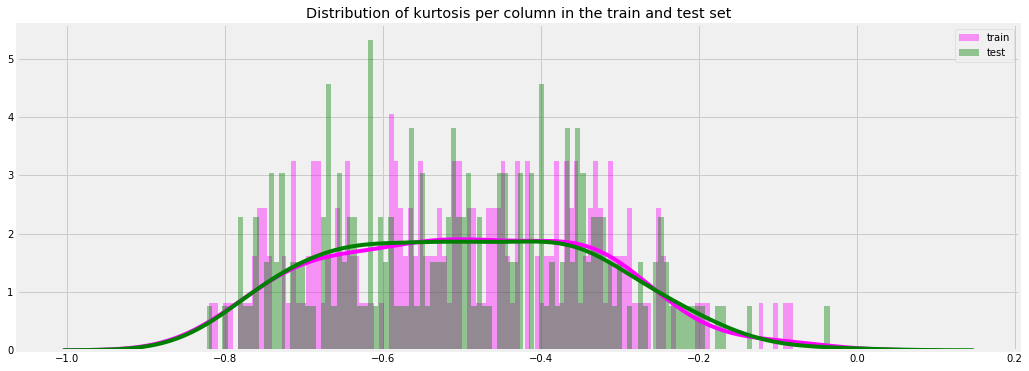
plt.title("Distribution of kurtosis per column in the train and test set")

sns.distplot(Cus\_train[features].kurtosis(axis=0),color="magenta", kde=True,bins=120, label='train')

sns.distplot(Cus\_test[features].kurtosis(axis=0),color="green", kde=True,bins=120, label='test')

plt.legend()

plt.show()



Both Skewness and Kurtosis show that the features distributions are like a normal one.

**Missing Value Analysis**

Cus\_train.isna().sum().sum()

0

Cus\_test.isna().sum().sum()

0

There are no missing values in Train and Test data

**Feature Selection**

df=Cus\_train.copy()

sns.set(rc={'figure.figsize':(20,28)})

# Compute the correlation matrix

corr = Cus\_train[numerical\_features].corr()

# Generate a mask for the upper triangle

mask = np.zeros\_like(corr, dtype=np.bool)

mask[np.triu\_indices\_from(mask)] = True

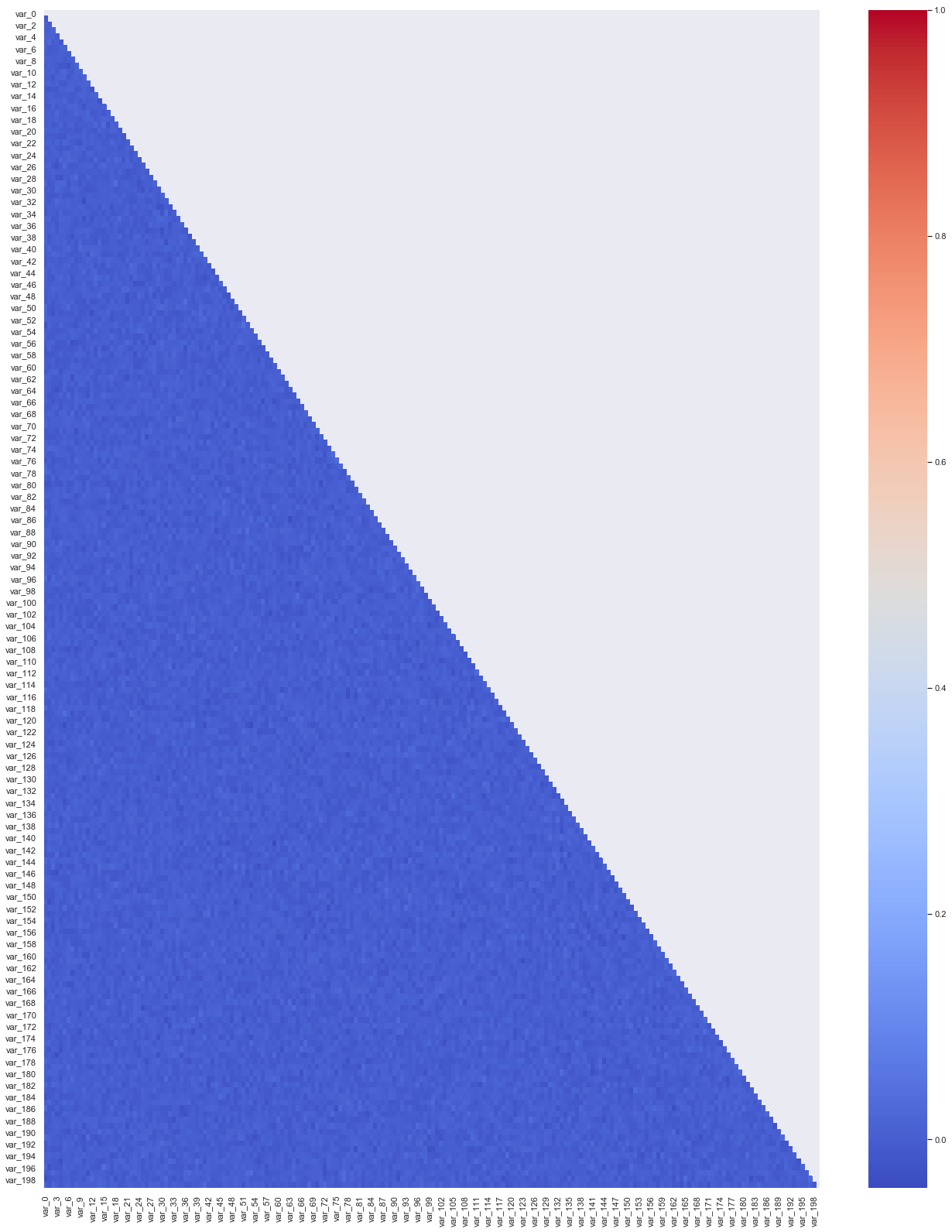
sns.heatmap(corr, mask=mask,

#annot=True,

#fmt=".2f",

cmap='coolwarm')

<matplotlib.axes.\_subplots.AxesSubplot at 0x16bb16a0>



the above figure shows that most of the pearson correlations between the numerical data are close to zero, in fact is between 0 and 0.2. That means that most of the numerical data are almost uncorrelated between them.

**Feature Engineering**

Let's calculate for starting few aggregated values for the existing features.

#Feature Engineering

idx = features = Cus\_train.columns.values[2:202]

for df in [Cus\_test, Cus\_train]:

df['sum'] = df[idx].sum(axis=1)

df['min'] = df[idx].min(axis=1)

df['max'] = df[idx].max(axis=1)

df['mean'] = df[idx].mean(axis=1)

df['std'] = df[idx].std(axis=1)

df['skew'] = df[idx].skew(axis=1)

df['kurt'] = df[idx].kurtosis(axis=1)

df['med'] = df[idx].median(axis=1)

Cus\_train[Cus\_train.columns[202:]].head()

|  | **sum** | **min** | **max** | **mean** | **std** | **skew** | **kurt** | **med** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1446.7378 | -37.8489 | 40.9121 | 7.233689 | 9.497786 | -0.132393 | 2.940025 | 6.78570 |
| **1** | 1349.0741 | -26.5076 | 36.2994 | 6.745370 | 9.259042 | -0.067781 | 1.321518 | 6.52180 |
| **2** | 1450.7626 | -31.4984 | 54.1926 | 7.253813 | 9.681573 | 0.272724 | 3.421256 | 6.41490 |
| **3** | 1262.4139 | -33.0995 | 27.5274 | 6.312069 | 9.055221 | -0.437886 | 1.440909 | 6.48625 |
| **4** | 1465.6899 | -23.8511 | 36.7349 | 7.328449 | 9.953071 | -0.002209 | 0.587409 | 6.51120 |

Cus\_train.head()

|  | **ID\_code** | **target** | **var\_0** | **var\_1** | **var\_2** | **var\_3** | **var\_4** | **var\_5** | **var\_6** | **var\_7** | **...** | **var\_198** | **var\_199** | **sum** | **min** | **max** | **mean** | **std** | **skew** | **kurt** | **med** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | train\_53101 | 0 | 9.5779 | -1.1448 | 10.2103 | 5.4965 | 11.8547 | -6.4545 | 3.9606 | 20.1943 | ... | 18.7698 | -11.6723 | 1446.7378 | -37.8489 | 40.9121 | 7.233689 | 9.497786 | -0.132393 | 2.940025 | 6.78570 |
| **1** | train\_74424 | 0 | 15.1904 | -5.0451 | 8.9406 | 6.2433 | 12.9933 | -9.0867 | 4.4992 | 21.8463 | ... | 18.0640 | -1.2149 | 1349.0741 | -26.5076 | 36.2994 | 6.745370 | 9.259042 | -0.067781 | 1.321518 | 6.52180 |
| **2** | train\_114569 | 0 | 10.2005 | -3.9381 | 12.2580 | 6.0077 | 12.3780 | -3.0878 | 3.4989 | 14.5860 | ... | 14.7799 | 11.6834 | 1450.7626 | -31.4984 | 54.1926 | 7.253813 | 9.681573 | 0.272724 | 3.421256 | 6.41490 |
| **3** | train\_181638 | 1 | 14.5489 | -9.2185 | 7.4017 | 8.1763 | 10.7966 | -7.8961 | 4.9425 | 20.5848 | ... | 12.9844 | 2.5465 | 1262.4139 | -33.0995 | 27.5274 | 6.312069 | 9.055221 | -0.437886 | 1.440909 | 6.48625 |
| **4** | train\_40335 | 0 | 13.1928 | -3.4527 | 15.6737 | 6.6761 | 12.3140 | -18.6930 | 4.4340 | 17.4016 | ... | 18.6588 | -19.0305 | 1465.6899 | -23.8511 | 36.7349 | 7.328449 | 9.953071 | -0.002209 | 0.587409 | 6.51120 |

5 rows × 210 columns

Cus\_test.head()

|  | **ID\_code** | **var\_0** | **var\_1** | **var\_2** | **var\_3** | **var\_4** | **var\_5** | **var\_6** | **var\_7** | **var\_8** | **...** | **var\_198** | **var\_199** | **sum** | **min** | **max** | **mean** | **std** | **skew** | **kurt** | **med** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | test\_53101 | 14.5931 | -3.4130 | 12.2206 | 8.5927 | 12.2787 | -0.1066 | 7.6984 | 17.3913 | 7.0487 | ... | 12.9742 | -3.1827 | 1463.5356 | -17.9367 | 40.2833 | 7.317678 | 9.480678 | 0.361158 | 0.902351 | 6.53130 |
| **1** | test\_74424 | 5.9114 | -3.4779 | 12.8391 | 6.7811 | 12.9037 | -10.6888 | 5.0967 | 11.8057 | -4.4972 | ... | 13.1813 | 5.4344 | 1226.6844 | -41.0584 | 45.0096 | 6.133422 | 9.785142 | -0.662492 | 4.453195 | 6.17810 |
| **2** | test\_114569 | 12.3714 | 0.3485 | 13.6545 | 6.6014 | 8.8026 | -10.0313 | 3.9339 | 17.6041 | 5.0323 | ... | 9.0674 | 14.2244 | 1389.9965 | -36.0183 | 39.3952 | 6.949983 | 9.706652 | -0.558886 | 3.063557 | 7.11910 |
| **3** | test\_181638 | 12.8432 | -1.1132 | 15.9820 | 7.3991 | 11.8261 | -3.8494 | 5.6762 | 13.3164 | -3.3855 | ... | 12.5065 | 8.8211 | 1322.4640 | -35.7617 | 37.1635 | 6.612320 | 9.644150 | -0.492677 | 2.617459 | 6.48785 |
| **4** | test\_40335 | 7.9607 | -0.9999 | 13.9064 | 5.1675 | 8.7360 | -8.4352 | 4.3069 | 13.5813 | 2.9485 | ... | 11.8314 | -3.1253 | 1323.7056 | -23.3404 | 43.2035 | 6.618528 | 9.404510 | 0.181288 | 1.824032 | 6.47330 |

5 rows × 209 columns

def plot\_feature\_distribution(df1, df2, label1, label2, features):

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(2,4,figsize=(15,8))

for feature in features:

i += 1

plt.subplot(2,4,i)

sns.distplot(df1[feature], hist=False,label=label1)

sns.distplot(df2[feature], hist=False,label=label2)

plt.xlabel(feature, fontsize=9)

locs, labels = plt.xticks()

plt.tick\_params(axis='x', which='major', labelsize=6, pad=-6)

plt.tick\_params(axis='y', which='major', labelsize=6)

plt.show();

Let's check the distribution of these new, engineered features.

We plot first the distribution of new features, grouped by value of corresponding target values

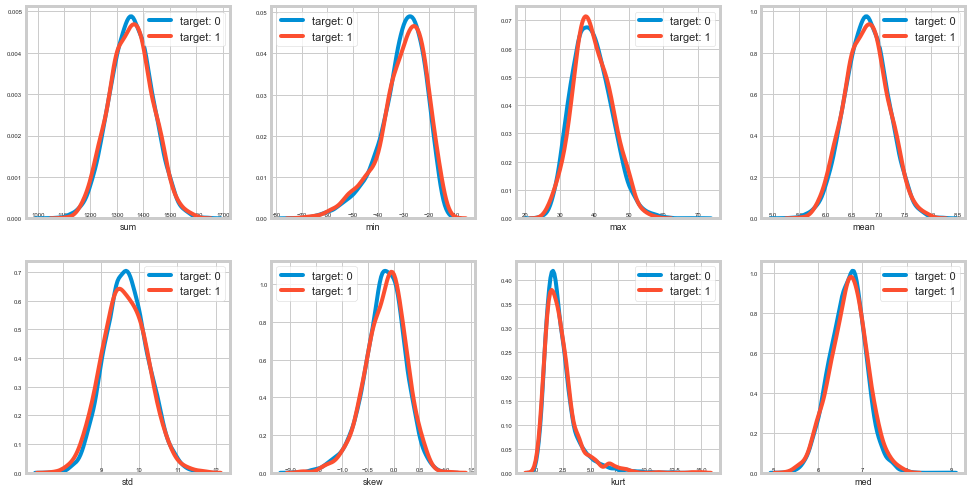
t0 = Cus\_train.loc[Cus\_train['target'] == 0]

t1 = Cus\_train.loc[Cus\_train['target'] == 1]

features = Cus\_train.columns.values[202:]

plot\_feature\_distribution(t0, t1, 'target: 0', 'target: 1', features)

<Figure size 1440x2016 with 0 Axes>

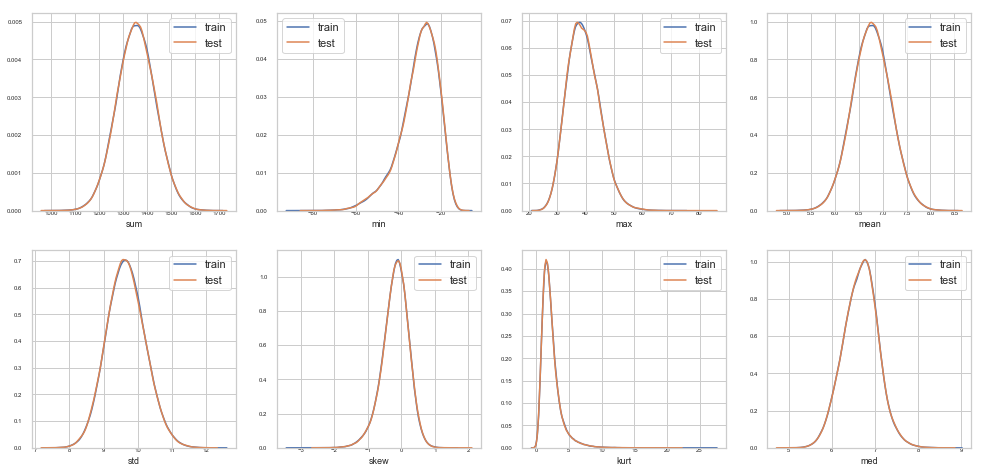


Let's show the distribution of new features values for train and test.

features = Cus\_train.columns.values[202:]

plot\_feature\_distribution(Cus\_train, Cus\_test, 'train', 'test', features)

<Figure size 1440x2016 with 0 Axes>



#Let's check how many features we have now.

print('Train and test columns: {} {}'.format(len(Cus\_train.columns), len(Cus\_test.columns)))

Train and test columns: 210 209

df1=Cus\_train.copy()

df2=Cus\_test.copy()

#Cus\_train=df1.copy()

#Cus\_train['ID\_code']=Cus\_train['ID\_code'].astype(float)

#Using train test split

y = Cus\_train['target']

X = Cus\_train.drop(['target','ID\_code'], axis=1)

y=y.astype('int')

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=101)

(X\_train.shape),(y\_train.shape)

((7500, 208), (7500,))

Handling Imbalanced Data

#SMOTE Oversampling

#print("Before OverSampling, counts of label '1': {}".format(sum(y\_train==1)))

#print("Before OverSampling, counts of label '0': {} \n".format(sum(y\_train==0)))

#sm = SMOTE(random\_state=2)

#X\_train\_res, y\_train\_res = sm.fit\_sample(X\_train, y\_train.ravel())

#print('After OverSampling, the shape of train\_X: {}'.format(X\_train\_res.shape))

#print('After OverSampling, the shape of train\_y: {} \n'.format(y\_train\_res.shape))

#print("After OverSampling, counts of label '1': {}".format(sum(y\_train\_res==1)))

#print("After OverSampling, counts of label '0': {}".format(sum(y\_train\_res==0)))

Before OverSampling, counts of label '1': 753

Before OverSampling, counts of label '0': 6747

After OverSampling, the shape of train\_X: (13494, 208)

After OverSampling, the shape of train\_y: (13494,)

After OverSampling, counts of label '1': 6747

After OverSampling, counts of label '0': 6747

**Random Over sampling**

X, y = make\_classification(n\_classes=2, class\_sep=2,weights=[0.1, 0.9], n\_informative=3, n\_redundant=1, flip\_y=0,

n\_features=20, n\_clusters\_per\_class=1, n\_samples=1000, random\_state=10)

print('Original dataset shape %s' % Counter(y))

ros = RandomOverSampler(random\_state=42)

X\_train\_res, y\_train\_res = ros.fit\_resample(X\_train, y\_train)

print('Resampled dataset shape %s' % Counter(y\_train\_res))

Original dataset shape Counter({1: 900, 0: 100})

Resampled dataset shape Counter({0: 6747, 1: 6747})

Prediction function

#Predicting & Stats Function

def pred(model\_object,predictors,compare):

"""1.model\_object = model name

2.predictors = data to be predicted

3.compare = y\_train"""

predicted = model\_object.predict(predictors)

# Determine the false positive and true positive rates

fpr, tpr, \_ = roc\_curve(compare, model\_object.predict\_proba(predictors)[:,1])

cm = pd.crosstab(compare,predicted)

TN = cm.iloc[0,0]

FN = cm.iloc[1,0]

TP = cm.iloc[1,1]

FP = cm.iloc[0,1]

print("CONFUSION MATRIX ------->> ")

print(cm)

print()

##check accuracy of model

print('Classification paradox :------->>')

print('Accuracy :- ', round(((TP+TN)\*100)/(TP+TN+FP+FN),2))

print()

print('True Negative Rate :- ',round((TN\*100)/(TN+FP),2))

print()

print('True Positive Rate / Recall :- ',round((TP\*100)/(FN+TP),2))

print()

print('False Negative Rate :- ',round((FN\*100)/(FN+TP),2))

print()

print('False Postive Rate :- ',round((FP\*100)/(FP+TN),2))

print()

print(classification\_report(compare,predicted))

print()

# Calculate the AUC

print ('AUC -: %0.2f' % auc(fpr, tpr))

**Model Level Approach**

Logistic Regression

#train data on logistic Regression

logit\_model = LogisticRegression(random\_state=101).fit(X\_train\_res,y\_train\_res)

#predict data using logistic Regression

pred(logit\_model,X\_test,y\_test)

# Output :------->>

# Accuracy :- 77.52

# True Negative Rate :- 78.3

# True Positive Rate / Recall :- 70.45

# False Negative Rate :- 29.55

# False Postive Rate :- 21.7

#AUC -: 0.84

CONFUSION MATRIX ------->>

col\_0 0 1

target

0 1764 489

1 73 174

Classification paradox :------->>

Accuracy :- 77.52

True Negative Rate :- 78.3

True Positive Rate / Recall :- 70.45

False Negative Rate :- 29.55

False Postive Rate :- 21.7

precision recall f1-score support

0 0.96 0.78 0.86 2253

1 0.26 0.70 0.38 247

accuracy 0.78 2500

macro avg 0.61 0.74 0.62 2500

weighted avg 0.89 0.78 0.82 2500

AUC -: 0.84

**KNN**

from sklearn.neighbors import KNeighborsClassifier

#KNN Model Development

KNN\_Model = KNeighborsClassifier(n\_neighbors=5).fit(X\_train\_res,y\_train\_res)

#train data using KNN

pred(KNN\_Model,X\_test,y\_test)

# Output :------->>

# Accuracy :- 77.56

# True Negative Rate :- 84.47

# True Positive Rate / Recall :- 14.57

# False Negative Rate :- 85.43

# False Postive Rate :- 15.53

# AUC = 0.51

CONFUSION MATRIX ------->>

col\_0 0 1

target

0 1903 350

1 211 36

Classification paradox :------->>

Accuracy :- 77.56

True Negative Rate :- 84.47

True Positive Rate / Recall :- 14.57

False Negative Rate :- 85.43

False Postive Rate :- 15.53

precision recall f1-score support

0 0.90 0.84 0.87 2253

1 0.09 0.15 0.11 247

accuracy 0.78 2500

macro avg 0.50 0.50 0.49 2500

weighted avg 0.82 0.78 0.80 2500

AUC -: 0.51

**Navie Bayes**

#Navie Model Development

Naive\_model = GaussianNB().fit(X\_train\_res,y\_train\_res)

#train data using Naive Bayes

pred(Naive\_model,X\_test,y\_test)

# Output :------->>

# Accuracy :- 80.96

# True Negative Rate :- 81.98

# True Positive Rate / Recall :- 71.66

# False Negative Rate :- 28.34

# False Postive Rate :- 18.02

# AUC = 0.86

CONFUSION MATRIX ------->>

col\_0 0 1

target

0 1847 406

1 70 177

Classification paradox :------->>

Accuracy :- 80.96

True Negative Rate :- 81.98

True Positive Rate / Recall :- 71.66

False Negative Rate :- 28.34

False Postive Rate :- 18.02

precision recall f1-score support

0 0.96 0.82 0.89 2253

1 0.30 0.72 0.43 247

accuracy 0.81 2500

macro avg 0.63 0.77 0.66 2500

weighted avg 0.90 0.81 0.84 2500

AUC -: 0.86

**Random Forest**

# Training Model With Optimum Parameters

final\_Model = RandomForestClassifier(random\_state=10, n\_estimators = 50,n\_jobs=-1)

final\_Model.fit(X\_train\_res,y\_train\_res)

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=50, n\_jobs=-1,

oob\_score=False, random\_state=10, verbose=0,

warm\_start=False)

#Validating Predictions

pred(final\_Model,X\_test,y\_test)

CONFUSION MATRIX ------->>

col\_0 0 1

target

0 2242 11

1 242 5

Classification paradox :------->>

Accuracy :- 89.88

True Negative Rate :- 99.51

True Positive Rate / Recall :- 2.02

False Negative Rate :- 97.98

False Postive Rate :- 0.49

precision recall f1-score support

0 0.90 1.00 0.95 2253

1 0.31 0.02 0.04 247

accuracy 0.90 2500

macro avg 0.61 0.51 0.49 2500

weighted avg 0.84 0.90 0.86 2500

AUC -: 0.63

**Light GBM**

features = [c for c in Cus\_train.columns if c not in ['ID\_code', 'target']]

target = Cus\_train['target']

param = {

'bagging\_freq': 5,

'bagging\_fraction': 0.4,

'boost\_from\_average':'false',

'boost': 'gbdt',

'feature\_fraction': 0.05,

'learning\_rate': 0.01,

'max\_depth': -1,

'metric':'auc',

'min\_data\_in\_leaf': 80,

'min\_sum\_hessian\_in\_leaf': 10.0,

'num\_leaves': 13,

'num\_threads': 8,

'tree\_learner': 'serial',

'objective': 'binary',

'verbosity': 1

}

folds = StratifiedKFold(n\_splits=10, shuffle=False, random\_state=101)

oof = np.zeros(len(Cus\_train))

predictions = np.zeros(len(Cus\_test))

feature\_importance\_df = pd.DataFrame()

for fold\_, (trn\_idx, val\_idx) in enumerate(folds.split(Cus\_train.values, target.values)):

print("Fold {}".format(fold\_))

trn\_data = lgb.Dataset(Cus\_train.iloc[trn\_idx][features], label=target.iloc[trn\_idx])

val\_data = lgb.Dataset(Cus\_train.iloc[val\_idx][features], label=target.iloc[val\_idx])

num\_round = 100

clf = lgb.train(param, trn\_data, num\_round, valid\_sets = [trn\_data, val\_data], verbose\_eval=100, early\_stopping\_rounds = 300)

oof[val\_idx] = clf.predict(Cus\_train.iloc[val\_idx][features], num\_iteration=clf.best\_iteration)

fold\_importance\_df = pd.DataFrame()

fold\_importance\_df["Feature"] = features

fold\_importance\_df["importance"] = clf.feature\_importance()

fold\_importance\_df["fold"] = fold\_ + 1

feature\_importance\_df = pd.concat([feature\_importance\_df, fold\_importance\_df], axis=0)

predictions += clf.predict(Cus\_test[features], num\_iteration=clf.best\_iteration) / folds.n\_splits

print("CV score: {:<8.5f}".format(roc\_auc\_score(target, oof)))

Fold 0

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.947539 valid\_1's auc: 0.798333

Did not meet early stopping. Best iteration is:

[100] training's auc: 0.947539 valid\_1's auc: 0.798333

Fold 1

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.949265 valid\_1's auc: 0.8131

Did not meet early stopping. Best iteration is:

[100] training's auc: 0.949265 valid\_1's auc: 0.8131

Fold 2

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.948218 valid\_1's auc: 0.783056

Did not meet early stopping. Best iteration is:

[99] training's auc: 0.949013 valid\_1's auc: 0.784567

Fold 3

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.950486 valid\_1's auc: 0.807378

Did not meet early stopping. Best iteration is:

[99] training's auc: 0.950605 valid\_1's auc: 0.808933

Fold 4

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.946816 valid\_1's auc: 0.784544

Did not meet early stopping. Best iteration is:

[99] training's auc: 0.947776 valid\_1's auc: 0.785744

Fold 5

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.947074 valid\_1's auc: 0.851822

Did not meet early stopping. Best iteration is:

[99] training's auc: 0.947688 valid\_1's auc: 0.850656

Fold 6

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.943621 valid\_1's auc: 0.774767

Did not meet early stopping. Best iteration is:

[99] training's auc: 0.943948 valid\_1's auc: 0.775967

Fold 7

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.948503 valid\_1's auc: 0.819911

Did not meet early stopping. Best iteration is:

[99] training's auc: 0.948882 valid\_1's auc: 0.8209

Fold 8

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.946248 valid\_1's auc: 0.788067

Did not meet early stopping. Best iteration is:

[99] training's auc: 0.946864 valid\_1's auc: 0.784411

Fold 9

Training until validation scores don't improve for 300 rounds.

[100] training's auc: 0.951791 valid\_1's auc: 0.790822

Did not meet early stopping. Best iteration is:

[99] training's auc: 0.951871 valid\_1's auc: 0.792189

CV score: 0.80086

cols = (feature\_importance\_df[["Feature", "importance"]]

.groupby("Feature")

.mean()

.sort\_values(by="importance", ascending=False)[:150].index)

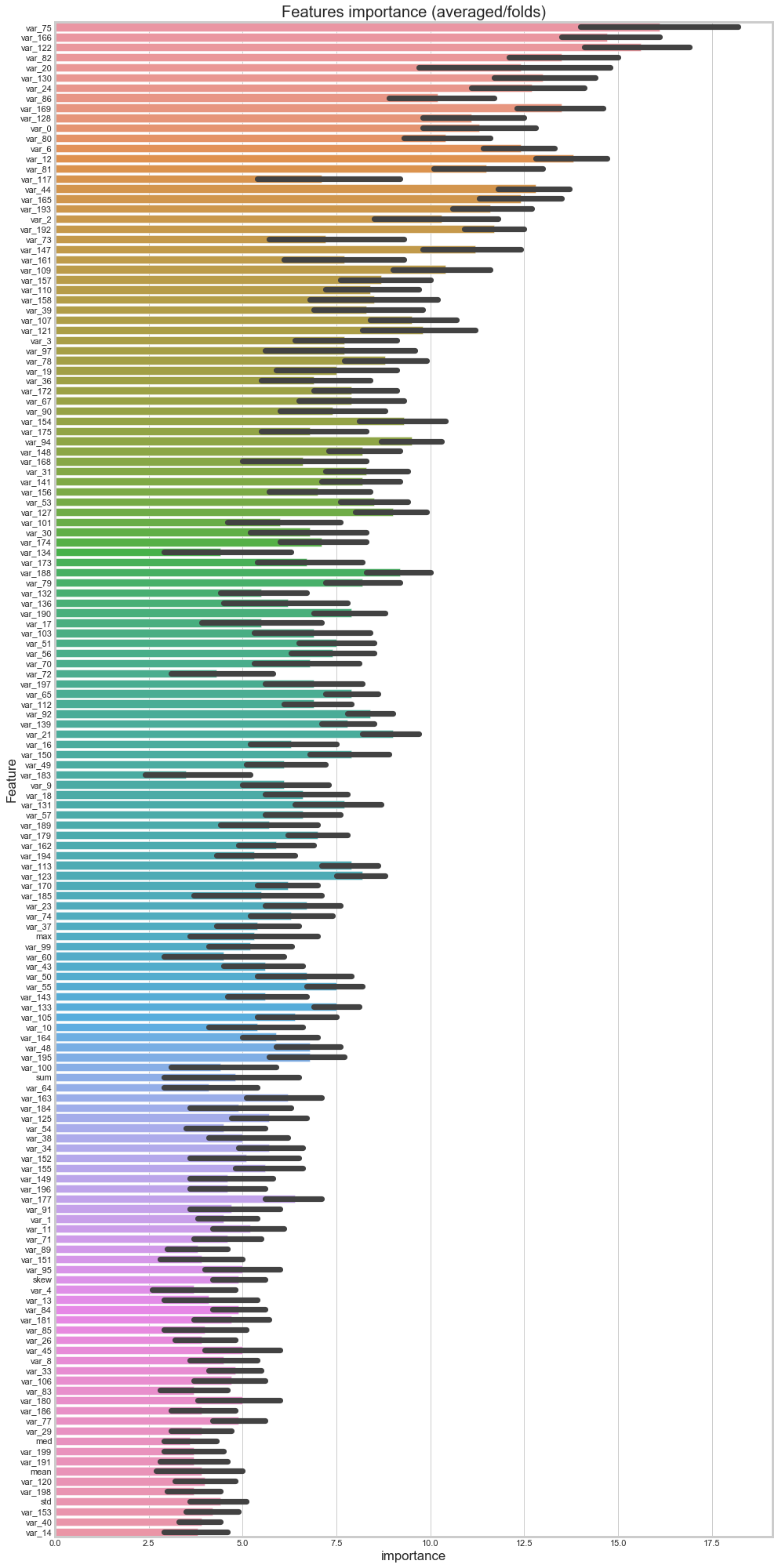
best\_features = feature\_importance\_df.loc[feature\_importance\_df.Feature.isin(cols)]

plt.figure(figsize=(14,28))

sns.barplot(x="importance", y="Feature", data=best\_features.sort\_values(by="importance",ascending=False))

plt.title('Features importance (averaged/folds)')

plt.tight\_layout()



sub\_df = pd.DataFrame({"ID\_code":Cus\_test["ID\_code"].values})

sub\_df["target"] = predictions

sub\_df.to\_csv("submission.csv", index=False)

Thank you