**Santander Customer Transaction Prediction Solution Using Python**

**Report by AMAN ARYA**

INDEX

1. Introduction
2. Prepare the data analysis
3. Data exploration
4. Check the data
5. Density plots of features
6. Distribution of mean and std
7. Distribution of min and max
8. Distribution of skew and kurtosis
9. Features correlations
10. Duplicate values
11. Feature engineering
12. Model
13. Submission

**Introduction**

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?

The data is anonimyzed, each row containing 200 numerical values identified just with a number.

In the following we will explore the data, prepare it for a model, train a model and predict the target value for the test set, then prepare a submission.

**Prepare for data analysis**

**Load Packages**

import gc

import os

import logging

import datetime

import warnings

import numpy as np

import pandas as pd

import seaborn as sns

import lightgbm as lgb

from tqdm import tqdm\_notebook

import matplotlib.pyplot as plt

from sklearn.metrics import mean\_squared\_error

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

from sklearn.model\_selection import StratifiedKFold

warnings.filterwarnings('ignore')

**Load Data**

os.listdir("C:/Users/DELL/Desktop/input")

['sample\_submission.csv', 'test.csv', 'train.csv']

Now load the train and test data files

%%time

train\_df = pd.read\_csv("C:/Users/DELL/Desktop/input/train.csv")

test\_df = pd.read\_csv("C:/Users/DELL/Desktop/input/test.csv")

Wall time: 31.2 s

**Data Exploration**

Check the train and test set.

train\_df.shape, test\_df.shape

((200000, 202), (200000, 201))

train\_df.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\_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 | 8.5635 | 12.7803 | -1.0914 |
| 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 | 8.7889 | 18.3560 | 1.9518 |
| 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 | 8.2675 | 14.7222 | 0.3965 |
| 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 | 10.2922 | 17.9697 | -8.9996 |
| 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 | 9.5031 | 17.9974 | -8.8104 |

5 rows × 202 columns

test\_df.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\_0 | 11.0656 | 7.7798 | 12.9536 | 9.4292 | 11.4327 | -2.3805 | 5.8493 | 18.2675 | 2.1337 | ... | -2.1556 | 11.8495 | -1.4300 | 2.4508 | 13.7112 | 2.4669 | 4.3654 | 10.7200 | 15.4722 | -8.7197 |
| 1 | test\_1 | 8.5304 | 1.2543 | 11.3047 | 5.1858 | 9.1974 | -4.0117 | 6.0196 | 18.6316 | -4.4131 | ... | 10.6165 | 8.8349 | 0.9403 | 10.1282 | 15.5765 | 0.4773 | -1.4852 | 9.8714 | 19.1293 | -20.9760 |
| 2 | test\_2 | 5.4827 | -10.3581 | 10.1407 | 7.0479 | 10.2628 | 9.8052 | 4.8950 | 20.2537 | 1.5233 | ... | -0.7484 | 10.9935 | 1.9803 | 2.1800 | 12.9813 | 2.1281 | -7.1086 | 7.0618 | 19.8956 | -23.1794 |
| 3 | test\_3 | 8.5374 | -1.3222 | 12.0220 | 6.5749 | 8.8458 | 3.1744 | 4.9397 | 20.5660 | 3.3755 | ... | 9.5702 | 9.0766 | 1.6580 | 3.5813 | 15.1874 | 3.1656 | 3.9567 | 9.2295 | 13.0168 | -4.2108 |
| 4 | test\_4 | 11.7058 | -0.1327 | 14.1295 | 7.7506 | 9.1035 | -8.5848 | 6.8595 | 10.6048 | 2.9890 | ... | 4.2259 | 9.1723 | 1.2835 | 3.3778 | 19.5542 | -0.2860 | -5.1612 | 7.2882 | 13.9260 | -9.1846 |

5 rows × 201 columns

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**;

Let's check if there are any missing data. We will also chech the type of data.

We check first train.

def missing\_data(data):

total = data.isnull().sum()

percent = (data.isnull().sum()/data.isnull().count()\*100)

tt = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

types = []

for col in data.columns:

dtype = str(data[col].dtype)

types.append(dtype)

tt['Types'] = types

return(np.transpose(tt))

%%time

missing\_data(train\_df)

Wall time: 1.72 s

|  | **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** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Total | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Percent | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Types | object | int64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | ... | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 |

3 rows × 202 columns

Here we check test dataset.

%%time

missing\_data(test\_df)

Wall time: 981 ms

| **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** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Total | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Percent | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Types | object | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | ... | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 |

3 rows × 201 columns

There are no missing data in train and test datasets. Let's check the numerical values in train and test dataset.

%%time

train\_df.describe()

Wall time: 4.23 s

|  | **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 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | ... | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 |
| mean | 0.100490 | 10.679914 | -1.627622 | 10.715192 | 6.796529 | 11.078333 | -5.065317 | 5.408949 | 16.545850 | 0.284162 | ... | 3.234440 | 7.438408 | 1.927839 | 3.331774 | 17.993784 | -0.142088 | 2.303335 | 8.908158 | 15.870720 | -3.326537 |
| std | 0.300653 | 3.040051 | 4.050044 | 2.640894 | 2.043319 | 1.623150 | 7.863267 | 0.866607 | 3.418076 | 3.332634 | ... | 4.559922 | 3.023272 | 1.478423 | 3.992030 | 3.135162 | 1.429372 | 5.454369 | 0.921625 | 3.010945 | 10.438015 |
| min | 0.000000 | 0.408400 | -15.043400 | 2.117100 | -0.040200 | 5.074800 | -32.562600 | 2.347300 | 5.349700 | -10.505500 | ... | -14.093300 | -2.691700 | -3.814500 | -11.783400 | 8.694400 | -5.261000 | -14.209600 | 5.960600 | 6.299300 | -38.852800 |
| 25% | 0.000000 | 8.453850 | -4.740025 | 8.722475 | 5.254075 | 9.883175 | -11.200350 | 4.767700 | 13.943800 | -2.317800 | ... | -0.058825 | 5.157400 | 0.889775 | 0.584600 | 15.629800 | -1.170700 | -1.946925 | 8.252800 | 13.829700 | -11.208475 |
| 50% | 0.000000 | 10.524750 | -1.608050 | 10.580000 | 6.825000 | 11.108250 | -4.833150 | 5.385100 | 16.456800 | 0.393700 | ... | 3.203600 | 7.347750 | 1.901300 | 3.396350 | 17.957950 | -0.172700 | 2.408900 | 8.888200 | 15.934050 | -2.819550 |
| 75% | 0.000000 | 12.758200 | 1.358625 | 12.516700 | 8.324100 | 12.261125 | 0.924800 | 6.003000 | 19.102900 | 2.937900 | ... | 6.406200 | 9.512525 | 2.949500 | 6.205800 | 20.396525 | 0.829600 | 6.556725 | 9.593300 | 18.064725 | 4.836800 |
| max | 1.000000 | 20.315000 | 10.376800 | 19.353000 | 13.188300 | 16.671400 | 17.251600 | 8.447700 | 27.691800 | 10.151300 | ... | 18.440900 | 16.716500 | 8.402400 | 18.281800 | 27.928800 | 4.272900 | 18.321500 | 12.000400 | 26.079100 | 28.500700 |

8 rows × 201 columns

%time

test\_df.describe()

Wall time: 0 ns

|  | **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 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | ... | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 |
| mean | 10.658737 | -1.624244 | 10.707452 | 6.788214 | 11.076399 | -5.050558 | 5.415164 | 16.529143 | 0.277135 | 7.569407 | ... | 3.189766 | 7.458269 | 1.925944 | 3.322016 | 17.996967 | -0.133657 | 2.290899 | 8.912428 | 15.869184 | -3.246342 |
| std | 3.036716 | 4.040509 | 2.633888 | 2.052724 | 1.616456 | 7.869293 | 0.864686 | 3.424482 | 3.333375 | 1.231865 | ... | 4.551239 | 3.025189 | 1.479966 | 3.995599 | 3.140652 | 1.429678 | 5.446346 | 0.920904 | 3.008717 | 10.398589 |
| min | 0.188700 | -15.043400 | 2.355200 | -0.022400 | 5.484400 | -27.767000 | 2.216400 | 5.713700 | -9.956000 | 4.243300 | ... | -14.093300 | -2.407000 | -3.340900 | -11.413100 | 9.382800 | -4.911900 | -13.944200 | 6.169600 | 6.584000 | -39.457800 |
| 25% | 8.442975 | -4.700125 | 8.735600 | 5.230500 | 9.891075 | -11.201400 | 4.772600 | 13.933900 | -2.303900 | 6.623800 | ... | -0.095000 | 5.166500 | 0.882975 | 0.587600 | 15.634775 | -1.160700 | -1.948600 | 8.260075 | 13.847275 | -11.124000 |
| 50% | 10.513800 | -1.590500 | 10.560700 | 6.822350 | 11.099750 | -4.834100 | 5.391600 | 16.422700 | 0.372000 | 7.632000 | ... | 3.162400 | 7.379000 | 1.892600 | 3.428500 | 17.977600 | -0.162000 | 2.403600 | 8.892800 | 15.943400 | -2.725950 |
| 75% | 12.739600 | 1.343400 | 12.495025 | 8.327600 | 12.253400 | 0.942575 | 6.005800 | 19.094550 | 2.930025 | 8.584825 | ... | 6.336475 | 9.531100 | 2.956000 | 6.174200 | 20.391725 | 0.837900 | 6.519800 | 9.595900 | 18.045200 | 4.935400 |
| max | 22.323400 | 9.385100 | 18.714100 | 13.142000 | 16.037100 | 17.253700 | 8.302500 | 28.292800 | 9.665500 | 11.003600 | ... | 20.359000 | 16.716500 | 8.005000 | 17.632600 | 27.947800 | 4.545400 | 15.920700 | 12.275800 | 26.538400 | 27.907400 |

8 rows × 200 columns

We can make few observations here:

* standard deviation is relatively large for both train and test variable data;
* min, max, mean, sdt values for train and test data looks quite close;
* mean values are distributed over a large range.

The number of values in train and test set is the same. Let's plot the scatter plot for train and test set for few of the features.

def plot\_feature\_scatter(df1, df2, features):

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(4,4,figsize=(14,14))

for feature in features:

i += 1

plt.subplot(4,4,i)

plt.scatter(df1[feature], df2[feature], marker='+')

plt.xlabel(feature, fontsize=9)

plt.show();

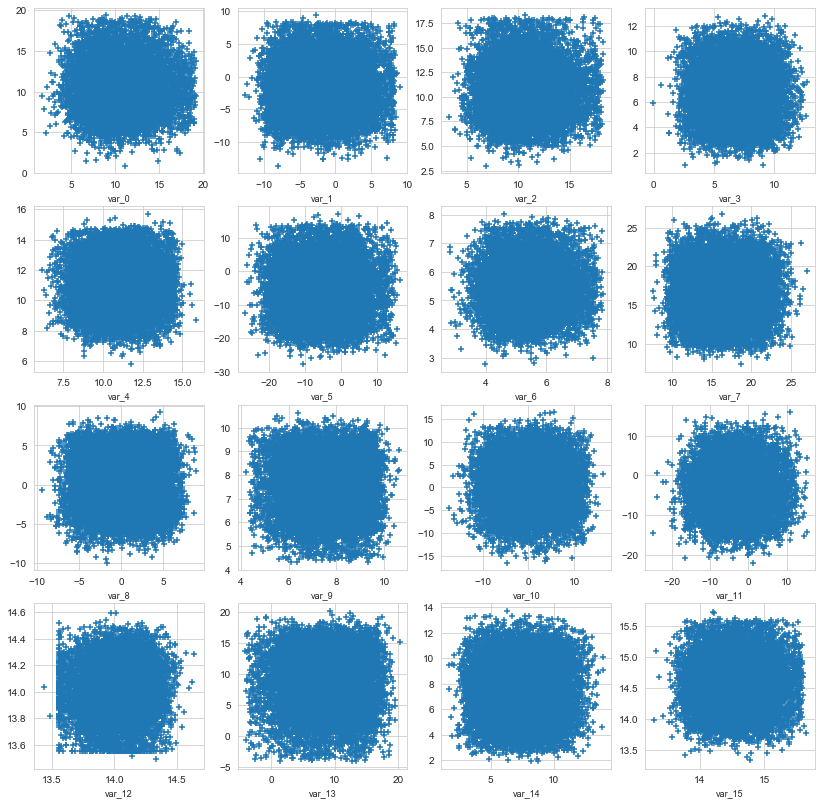
We will show just 5% of the data. On x axis we show train values and on the y axis we show the test values.

features = ['var\_0', 'var\_1','var\_2','var\_3', 'var\_4', 'var\_5', 'var\_6', 'var\_7',

'var\_8', 'var\_9', 'var\_10','var\_11','var\_12', 'var\_13', 'var\_14', 'var\_15',

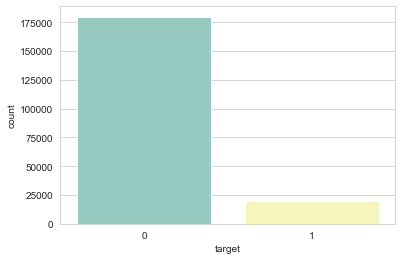
]

plot\_feature\_scatter(train\_df[::20],test\_df[::20], features)



Let's check the distribution of **target** value in train dataset.

sns.countplot(train\_df['target'], palette='Set3')



print("There are {}% target values with 1".format(100 \* train\_df["target"].value\_counts()[1]/train\_df.shape[0]))

There are 10.049% target values with 1

* The data is unbalanced with respect with **target** value.

**Density plots of features**

Let's show now the density plot of variables in train dataset.

We represent with different colors the distribution for values with **target** value **0** and **1**.

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

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(10,10,figsize=(18,22))

for feature in features:

i += 1

plt.subplot(10,10,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();

t0 = train\_df.loc[train\_df['target'] == 0]

t1 = train\_df.loc[train\_df['target'] == 1]

features = train\_df.columns.values[2:102]

plot\_feature\_distribution(t0, t1, '0', '1',

features)



features = train\_df.columns.values[102:202]

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



We can observe that there is a considerable number of features with significant different distribution for the two target values.  
For example, **var\_0**, **var\_1**, **var\_2**, **var\_5**, **var\_9**, **var\_13**, **var\_106**, **var\_109**, **var\_139** and many others.

Also some features, like **var\_2**, **var\_13**, **var\_26**, **var\_55**, **var\_175**, **var\_184**, **var\_196** shows a distribution that resambles to a bivariate distribution.

We will take this into consideration in the future for the selection of the features for our prediction model.

Le't s now look to the distribution of the same features in parallel in train and test datasets.

features = train\_df.columns.values[2:102]

plot\_feature\_distribution(train\_df, test\_df, 'train', 'test', features)



features = train\_df.columns.values[102:202]

plot\_feature\_distribution(train\_df, test\_df, 'train', 'test', features)



**Distribution of mean and std**

Let's check the distribution of the mean values per row in the train and test set.

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

features = train\_df.columns.values[2:202]

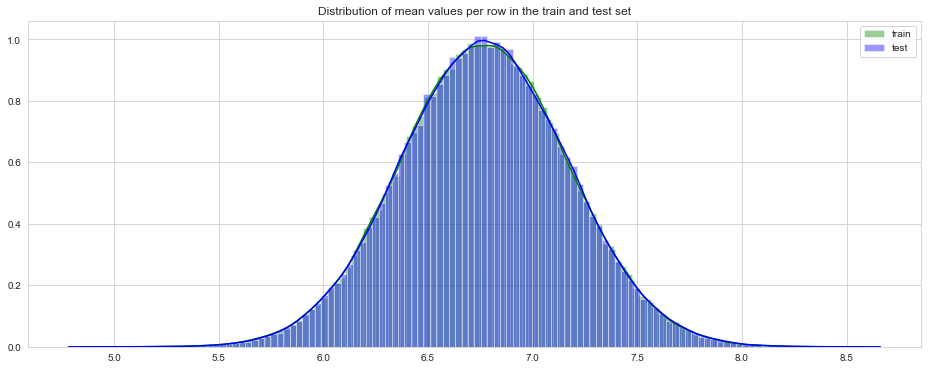
plt.title("Distribution of mean values per row in the train and test set")

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

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

plt.legend()

plt.show()



Let's check the distribution of the mean values per columns in the train and test set.

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

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

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

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

plt.legend()

plt.show()



Let's show the distribution of standard deviation of values per row for train and test datasets.

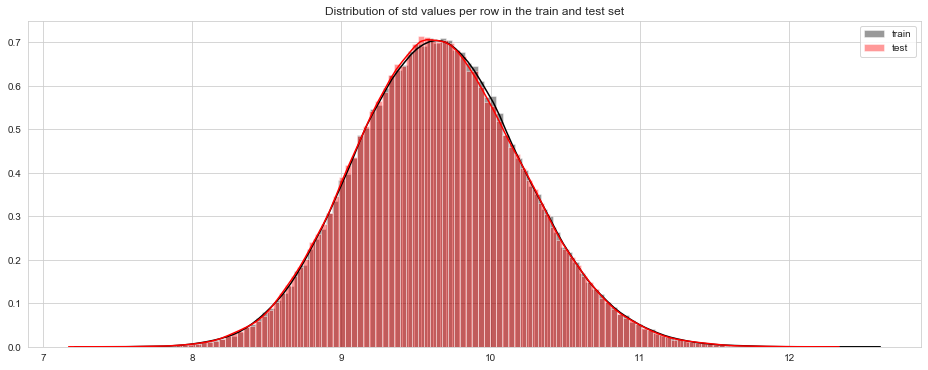
plt.figure(figsize=(16,6))

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

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

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

plt.legend();plt.show()



Let's check the distribution of the standard deviation of values per columns in the train and test datasets.

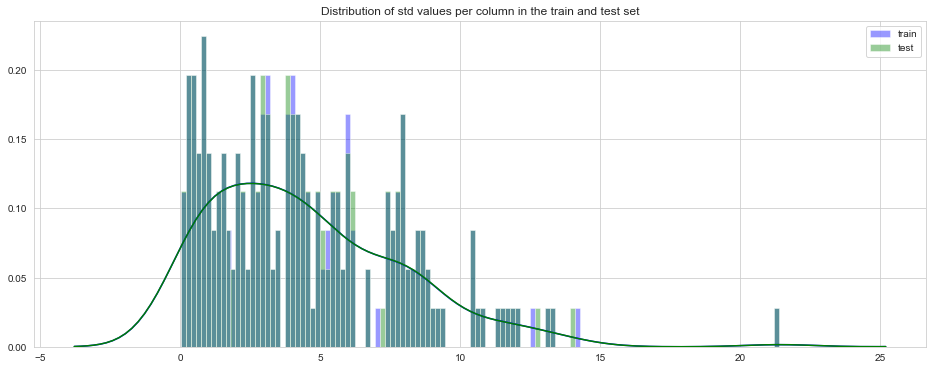
plt.figure(figsize=(16,6))

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

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

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

plt.legend(); plt.show()



Let's check now the distribution of the mean value per row in the train dataset, grouped by value of target.

t0 = train\_df.loc[train\_df['target'] == 0]

t1 = train\_df.loc[train\_df['target'] == 1]

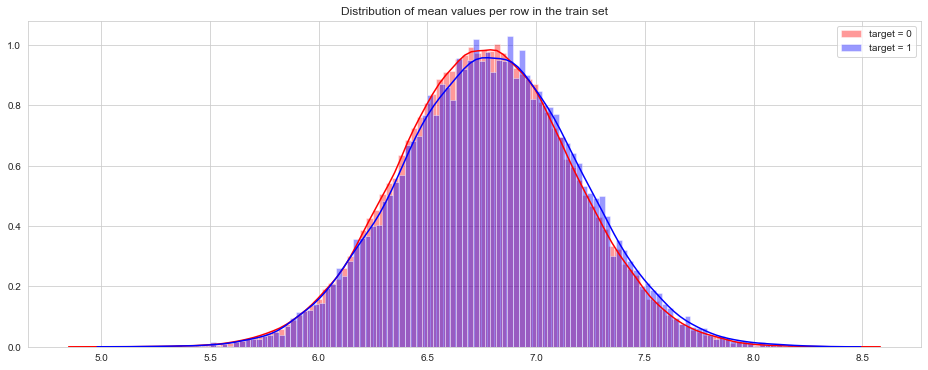
plt.figure(figsize=(16,6))

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

sns.distplot(t0[features].mean(axis=1),color="red", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].mean(axis=1),color="blue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



Let's check now the distribution of the mean value per column in the train dataset, grouped by value of target.

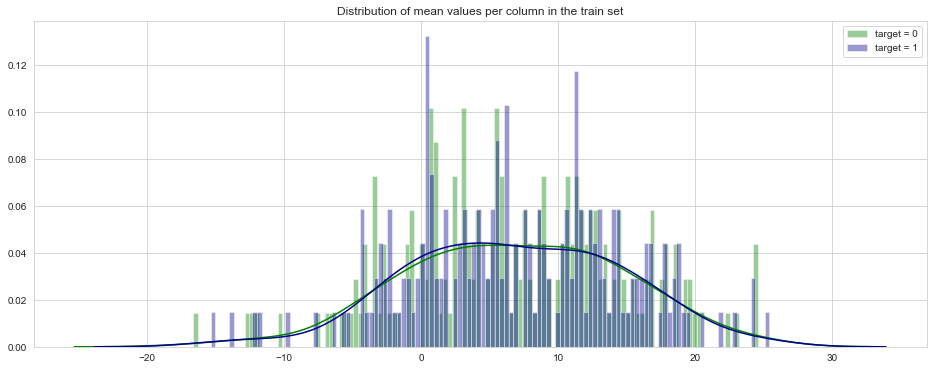
plt.figure(figsize=(16,6))

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

sns.distplot(t0[features].mean(axis=0),color="green", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].mean(axis=0),color="darkblue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



**Distribution of min and max**

Let's check the distribution of min per row in the train and test set.

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

features = train\_df.columns.values[2:202]

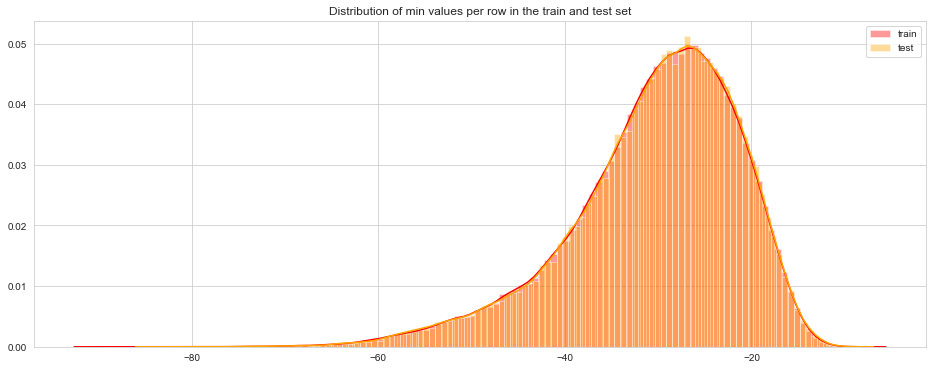
plt.title("Distribution of min values per row in the train and test set")

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

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

plt.legend()

plt.show()

****

A long queue to the lower values for both, extended as long as to -80 for test set, is observed.

Let's now show the distribution of min per column in the train and test set.

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

features = train\_df.columns.values[2:202]

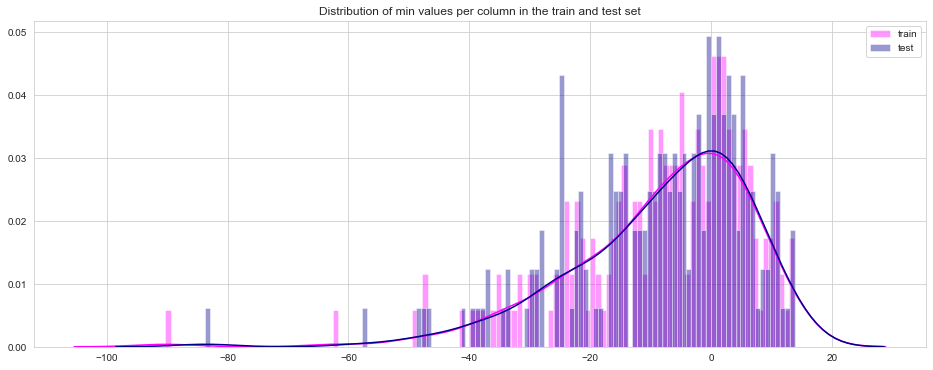
plt.title("Distribution of min values per column in the train and test set")

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

sns.distplot(test\_df[features].min(axis=0),color="darkblue", kde=True,bins=120, label='test')

plt.legend()

plt.show()



Let's check now the distribution of max values per rows for train and test set.

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

features = train\_df.columns.values[2:202]

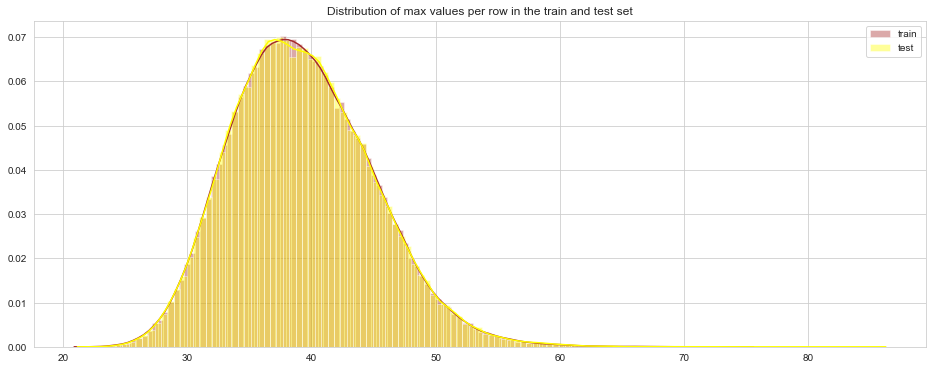
plt.title("Distribution of max values per row in the train and test set")

sns.distplot(train\_df[features].max(axis=1),color="brown", kde=True,bins=120, label='train')

sns.distplot(test\_df[features].max(axis=1),color="yellow", kde=True,bins=120, label='test')

plt.legend()

plt.show()



Let's show now the max distribution on columns for train and test set.

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

features = train\_df.columns.values[2:202]

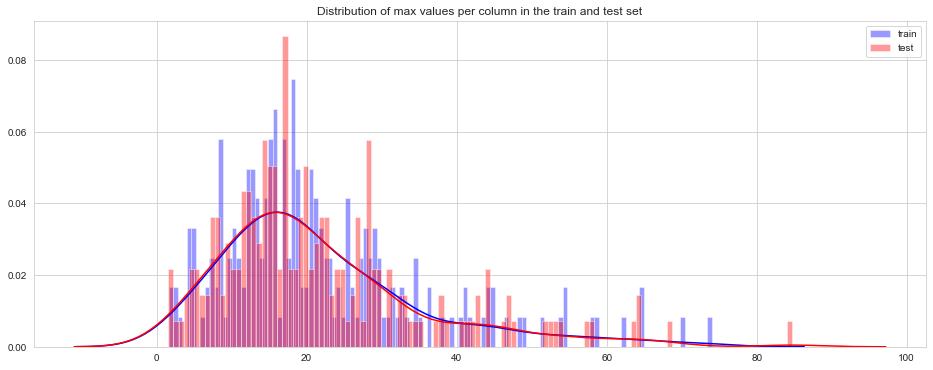
plt.title("Distribution of max values per column in the train and test set")

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

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

plt.legend()

plt.show()



Let's show now the distributions of min values per row in train set, separated on the values of target (0 and 1).

t0 = train\_df.loc[train\_df['target'] == 0]

t1 = train\_df.loc[train\_df['target'] == 1]

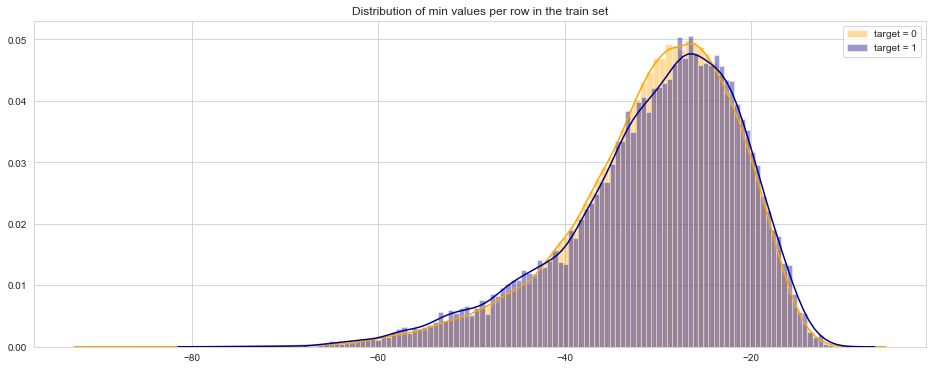
plt.figure(figsize=(16,6))

plt.title("Distribution of min values per row in the train set")

sns.distplot(t0[features].min(axis=1),color="orange", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].min(axis=1),color="darkblue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



We show here the distribution of min values per columns in train set.

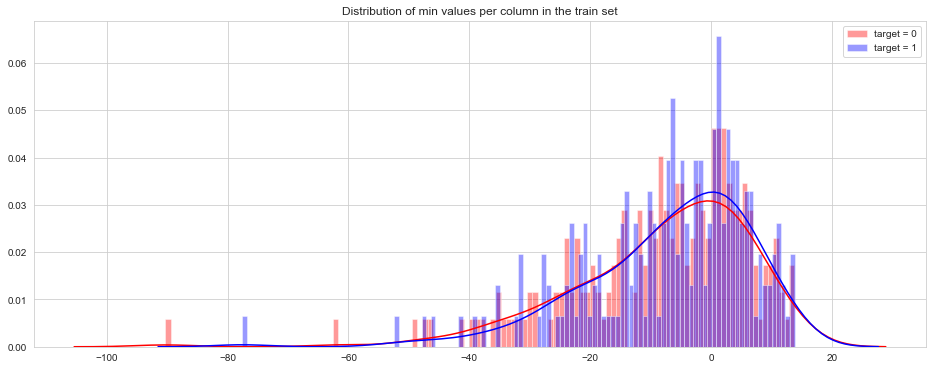
plt.figure(figsize=(16,6))

plt.title("Distribution of min values per column in the train set")

sns.distplot(t0[features].min(axis=0),color="red", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].min(axis=0),color="blue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



Let's show now the distribution of max values per row in the train set.

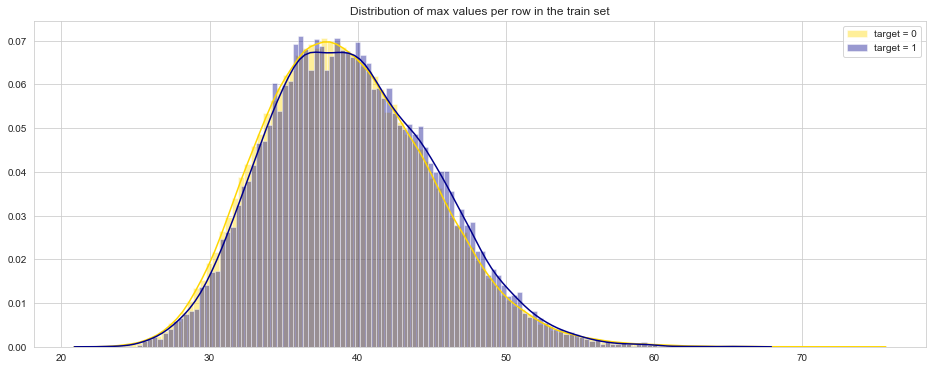
plt.figure(figsize=(16,6))

plt.title("Distribution of max values per row in the train set")

sns.distplot(t0[features].max(axis=1),color="gold", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].max(axis=1),color="darkblue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



Let's show also the distribution of max values per columns in the train set.

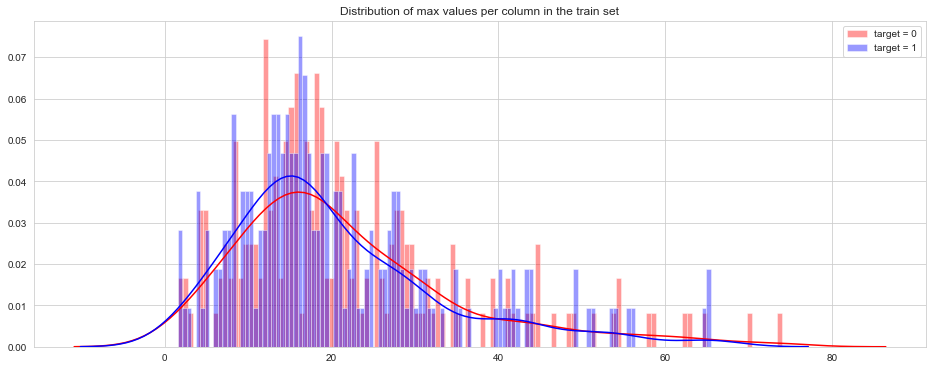
plt.figure(figsize=(16,6))

plt.title("Distribution of max values per column in the train set")

sns.distplot(t0[features].max(axis=0),color="red", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].max(axis=0),color="blue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



**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))

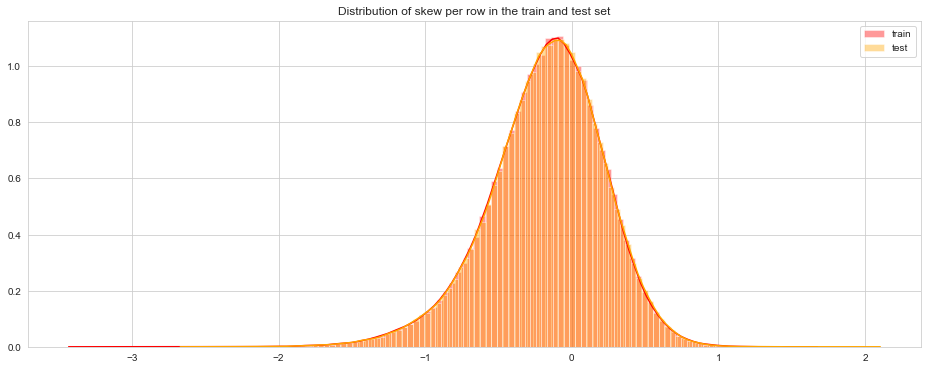
plt.title("Distribution of skew per row in the train and test set")

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

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

plt.legend()

plt.show()



Let's see first the distribution of skewness calculated per columns in train and test set.

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

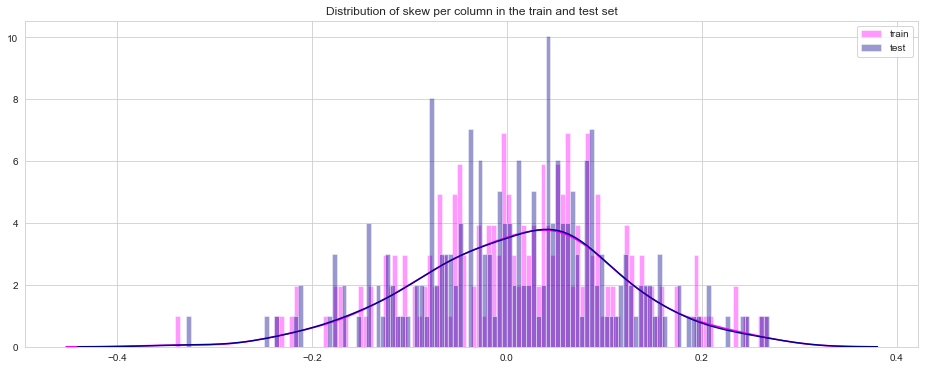
plt.title("Distribution of skew per column in the train and test set")

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

sns.distplot(test\_df[features].skew(axis=0),color="darkblue", 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))

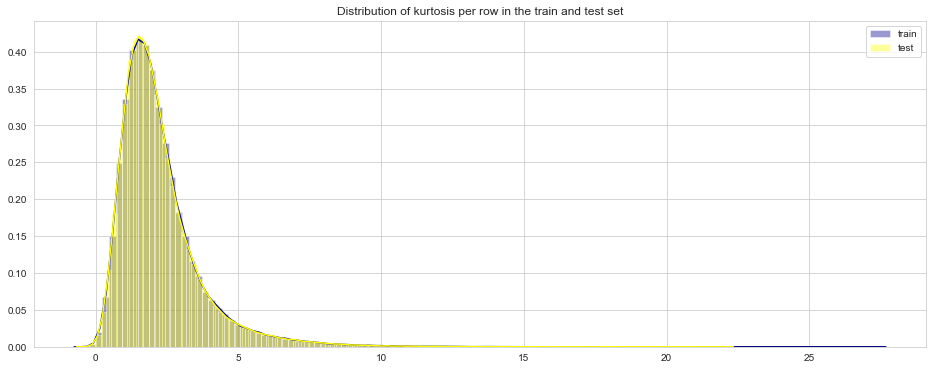
plt.title("Distribution of kurtosis per row in the train and test set")

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

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

plt.legend()

plt.show()



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

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

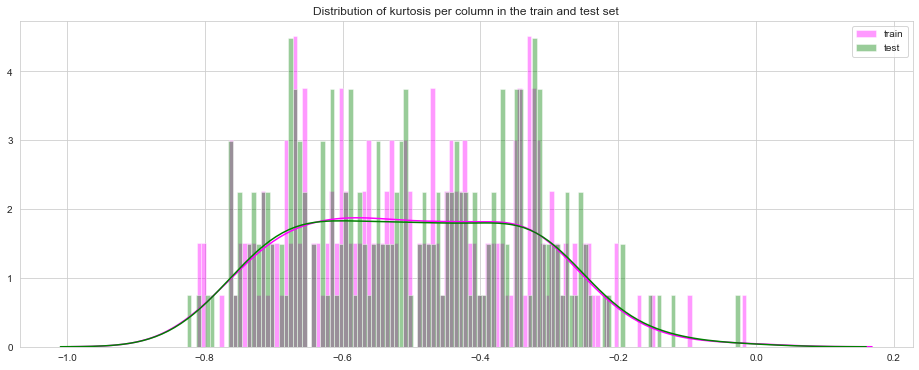
plt.title("Distribution of kurtosis per column in the train and test set")

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

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

plt.legend()

plt.show()



Let's see now the distribution of skewness on rows in train separated for values of target 0 and 1.

t0 = train\_df.loc[train\_df['target'] == 0]

t1 = train\_df.loc[train\_df['target'] == 1]

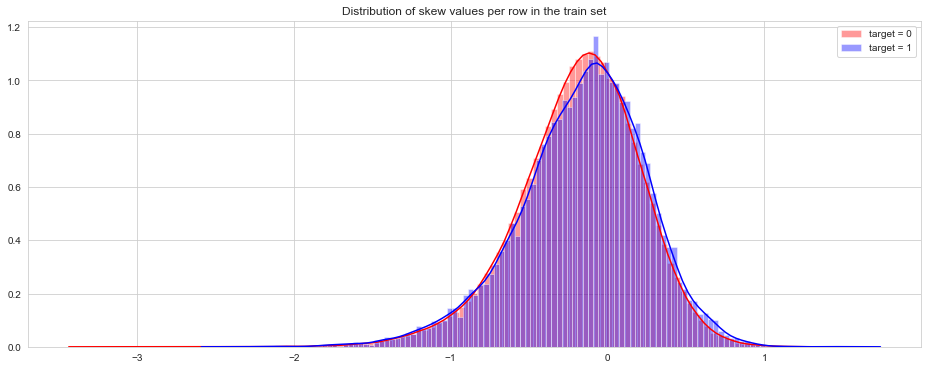
plt.figure(figsize=(16,6))

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

sns.distplot(t0[features].skew(axis=1),color="red", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].skew(axis=1),color="blue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



Let's see now the distribution of skewness on columns in train separated for values of target 0 and 1.

t0 = train\_df.loc[train\_df['target'] == 0]

t1 = train\_df.loc[train\_df['target'] == 1]

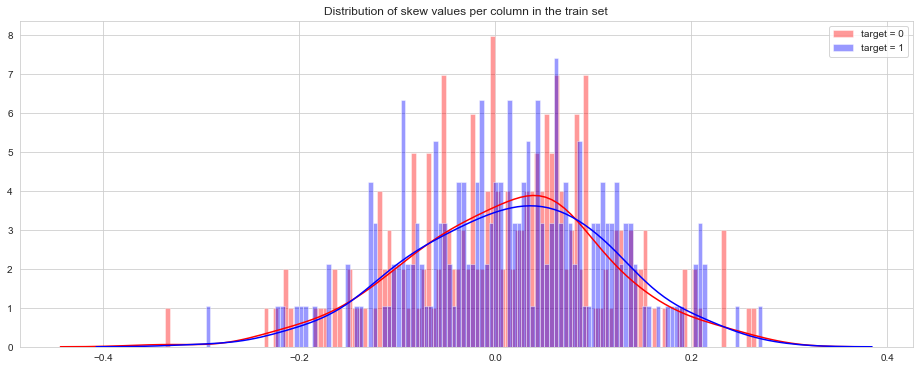
plt.figure(figsize=(16,6))

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

sns.distplot(t0[features].skew(axis=0),color="red", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].skew(axis=0),color="blue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



Let's see now the distribution of kurtosis on rows in train separated for values of target 0 and 1.

t0 = train\_df.loc[train\_df['target'] == 0]

t1 = train\_df.loc[train\_df['target'] == 1]

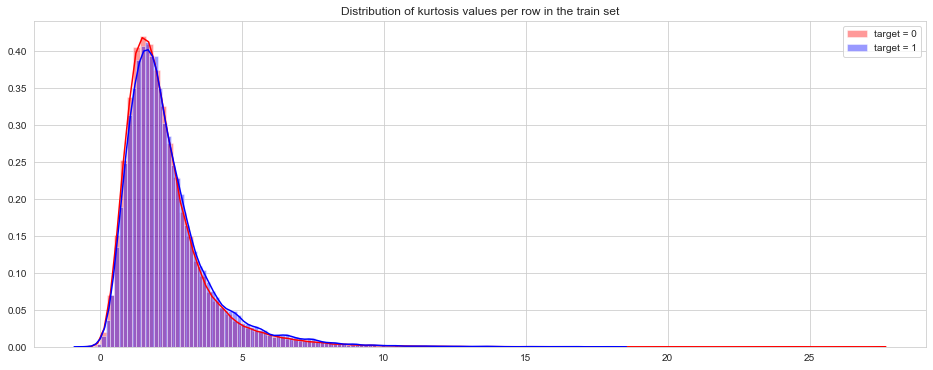
plt.figure(figsize=(16,6))

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

sns.distplot(t0[features].kurtosis(axis=1),color="red", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].kurtosis(axis=1),color="blue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



Let's see now the distribution of kurtosis on columns in train separated for values of target 0 and 1

t0 = train\_df.loc[train\_df['target'] == 0]

t1 = train\_df.loc[train\_df['target'] == 1]

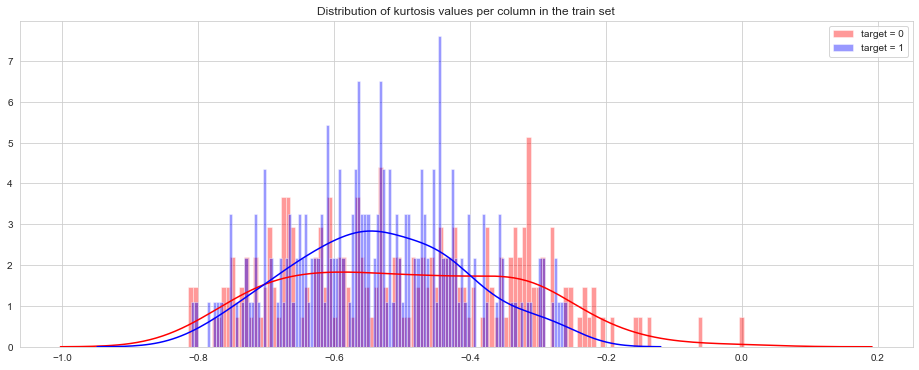
plt.figure(figsize=(16,6))

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

sns.distplot(t0[features].kurtosis(axis=0),color="red", kde=True,bins=120, label='target = 0')

sns.distplot(t1[features].kurtosis(axis=0),color="blue", kde=True,bins=120, label='target = 1')

plt.legend(); plt.show()



**Feature Correlation**

We calculate now the correlations between the features in train set.  
The following table shows the first 10 the least correlated features.

%%time

correlations = train\_df[features].corr().abs().unstack().sort\_values(kind="quicksort").reset\_index()

correlations = correlations[correlations['level\_0'] != correlations['level\_1']]

correlations.head(10)

Wall time: 45.6 s

Parser : 130 ms

|  | **level\_0** | **level\_1** | **0** |
| --- | --- | --- | --- |
| 0 | var\_75 | var\_191 | 2.703975e-08 |
| 1 | var\_191 | var\_75 | 2.703975e-08 |
| 2 | var\_173 | var\_6 | 5.942735e-08 |
| 3 | var\_6 | var\_173 | 5.942735e-08 |
| 4 | var\_126 | var\_109 | 1.313947e-07 |
| 5 | var\_109 | var\_126 | 1.313947e-07 |
| 6 | var\_144 | var\_27 | 1.772502e-07 |
| 7 | var\_27 | var\_144 | 1.772502e-07 |
| 8 | var\_177 | var\_100 | 3.116544e-07 |
| 9 | var\_100 | var\_177 | 3.116544e-07 |

correlations.tail(10)

|  | **level\_0** | **level\_1** | **0** |
| --- | --- | --- | --- |
| 39790 | var\_183 | var\_189 | 0.009359 |
| 39791 | var\_189 | var\_183 | 0.009359 |
| 39792 | var\_174 | var\_81 | 0.009490 |
| 39793 | var\_81 | var\_174 | 0.009490 |
| 39794 | var\_81 | var\_165 | 0.009714 |
| 39795 | var\_165 | var\_81 | 0.009714 |
| 39796 | var\_53 | var\_148 | 0.009788 |
| 39797 | var\_148 | var\_53 | 0.009788 |
| 39798 | var\_26 | var\_139 | 0.009844 |
| 39799 | var\_139 | var\_26 | 0.009844 |

correlations.head(10)

|  | **level\_0** | **level\_1** | **0** |
| --- | --- | --- | --- |
| 0 | var\_75 | var\_191 | 2.703975e-08 |
| 1 | var\_191 | var\_75 | 2.703975e-08 |
| 2 | var\_173 | var\_6 | 5.942735e-08 |
| 3 | var\_6 | var\_173 | 5.942735e-08 |
| 4 | var\_126 | var\_109 | 1.313947e-07 |
| 5 | var\_109 | var\_126 | 1.313947e-07 |
| 6 | var\_144 | var\_27 | 1.772502e-07 |
| 7 | var\_27 | var\_144 | 1.772502e-07 |
| 8 | var\_177 | var\_100 | 3.116544e-07 |
| 9 | var\_100 | var\_177 | 3.116544e-07 |

**Duplicate values**

%%time

features = train\_df.columns.values[2:202]

unique\_max\_train = []

unique\_max\_test = []

for feature in features:

values = train\_df[feature].value\_counts()

unique\_max\_train.append([feature, values.max(), values.idxmax()])

values = test\_df[feature].value\_counts()

unique\_max\_test.append([feature, values.max(), values.idxmax()])

Wall time: 16.5 s

np.transpose((pd.DataFrame(unique\_max\_train, columns=['Feature', 'Max duplicates', 'Value'])).\

sort\_values(by = 'Max duplicates', ascending=False).head(15))

|  | **68** | **108** | **126** | **12** | **91** | **103** | **148** | **71** | **161** | **25** | **125** | **169** | **166** | **133** | **43** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Feature | var\_68 | var\_108 | var\_126 | var\_12 | var\_91 | var\_103 | var\_148 | var\_71 | var\_161 | var\_25 | var\_125 | var\_169 | var\_166 | var\_133 | var\_43 |
| Max duplicates | 1084 | 313 | 305 | 203 | 66 | 61 | 59 | 54 | 52 | 41 | 40 | 39 | 39 | 39 | 39 |
| Value | 5.0214 | 14.1999 | 11.5356 | 13.5545 | 6.9785 | 1.6662 | 4.0456 | 0.7031 | 5.7688 | 13.6723 | 12.5159 | 5.6941 | 2.7306 | 6.8632 | 11.4522 |

np.transpose((pd.DataFrame(unique\_max\_test, columns=['Feature', 'Max duplicates', 'Value'])).\

sort\_values(by = 'Max duplicates', ascending=False).head(15))

|  | **68** | **126** | **108** | **12** | **91** | **103** | **148** | **161** | **25** | **71** | **43** | **166** | **125** | **169** | **133** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Feature** | var\_68 | var\_126 | var\_108 | var\_12 | var\_91 | var\_103 | var\_148 | var\_161 | var\_25 | var\_71 | var\_43 | var\_166 | var\_125 | var\_169 | var\_133 |
| **Max duplicates** | 1104 | 307 | 302 | 188 | 86 | 78 | 74 | 69 | 60 | 60 | 58 | 53 | 53 | 51 | 50 |
| **Value** | 5.0197 | 11.5357 | 14.1999 | 13.5546 | 6.9939 | 1.4659 | 4.0004 | 5.7114 | 13.5965 | 0.5389 | 11.5738 | 2.8446 | 12.2189 | 5.8455 | 6.6873 |

**Feature Engineering**

%%time

idx = features = train\_df.columns.values[2:202]

for df in [test\_df, train\_df]:

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)

Wall time: 31.6 s

train\_df[train\_df.columns[202:]].head()

|  | **sum** | **min** | **max** | **mean** | **std** | **skew** | **kurt** | **med** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1456.3182 | -21.4494 | 43.1127 | 7.281591 | 9.331540 | 0.101580 | 1.331023 | 6.77040 |
| 1 | 1415.3636 | -47.3797 | 40.5632 | 7.076818 | 10.336130 | -0.351734 | 4.110215 | 7.22315 |
| 2 | 1240.8966 | -22.4038 | 33.8820 | 6.204483 | 8.753387 | -0.056957 | 0.546438 | 5.89940 |
| 3 | 1288.2319 | -35.1659 | 38.1015 | 6.441159 | 9.594064 | -0.480116 | 2.630499 | 6.70260 |
| 4 | 1354.2310 | -65.4863 | 41.1037 | 6.771155 | 11.287122 | -1.463426 | 9.787399 | 6.94735 |

test\_df[test\_df.columns[201:]].head()

|  | **sum** | **min** | **max** | **mean** | **std** | **skew** | **kurt** | **med** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1416.6404 | -31.9891 | 42.0248 | 7.083202 | 9.910632 | -0.088518 | 1.871262 | 7.31440 |
| 1 | 1249.6860 | -41.1924 | 35.6020 | 6.248430 | 9.541267 | -0.559785 | 3.391068 | 6.43960 |
| 2 | 1430.2599 | -34.3488 | 39.3654 | 7.151299 | 9.967466 | -0.135084 | 2.326901 | 7.26355 |
| 3 | 1411.4447 | -21.4797 | 40.3383 | 7.057223 | 8.257204 | -0.167741 | 2.253054 | 6.89675 |
| 4 | 1423.7364 | -24.8254 | 45.5510 | 7.118682 | 10.043542 | 0.293484 | 2.044943 | 6.83375 |

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

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(2,4,figsize=(18,8))

for feature in features:

i += 1

plt.subplot(2,4,i)

sns.kdeplot(df1[feature], bw=0.5,label=label1)

sns.kdeplot(df2[feature], bw=0.5,label=label2)

plt.xlabel(feature, fontsize=11)

locs, labels = plt.xticks()

plt.tick\_params(axis='x', which='major', labelsize=8)

plt.tick\_params(axis='y', which='major', labelsize=8)

plt.show();

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

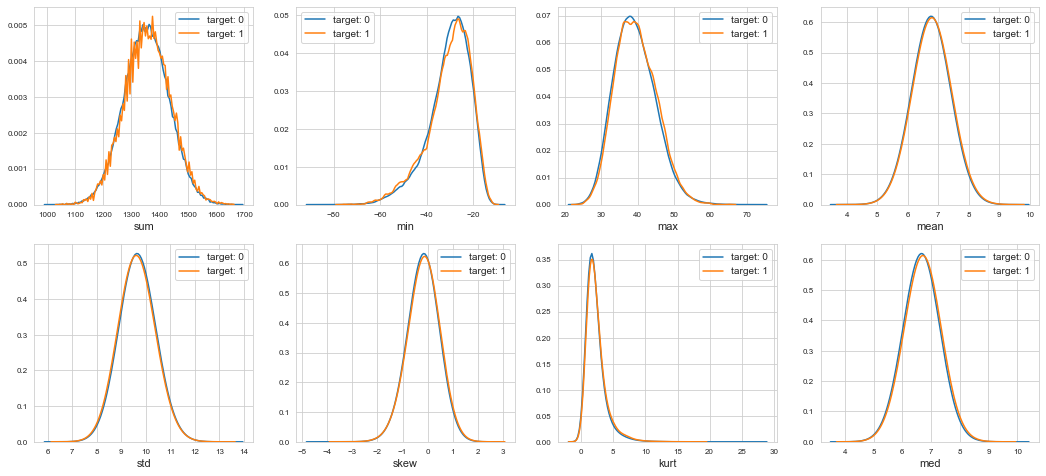
We plot first the distribution of new features, grouped by value of corresponding target values.

t0 = train\_df.loc[train\_df['target'] == 0]

t1 = train\_df.loc[train\_df['target'] == 1]

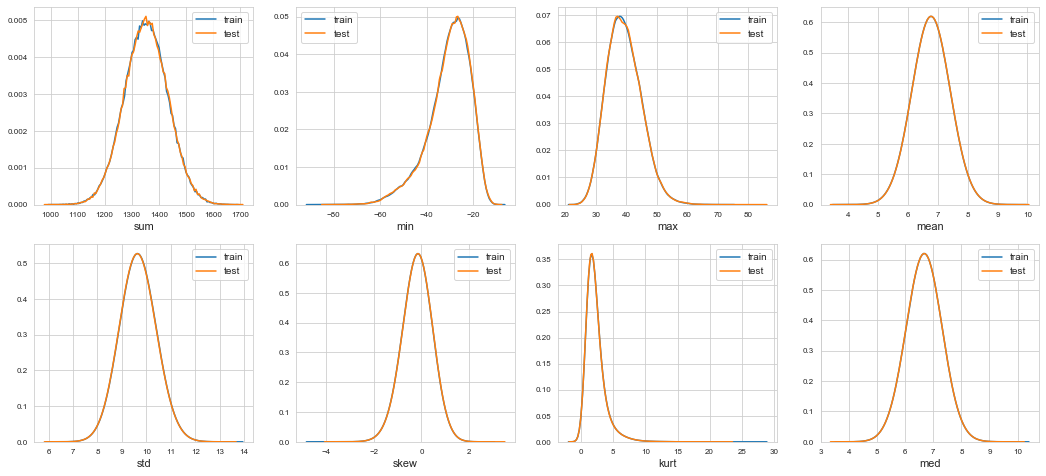
features = train\_df.columns.values[202:]

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



features = train\_df.columns.values[202:]

plot\_new\_feature\_distribution(train\_df, test\_df, 'train', 'test', features)



features = [c for c in train\_df.columns if c not in ['ID\_code', 'target']] for feature in features: train\_df['r2\_'+feature] = np.round(train\_df[feature], 2) test\_df['r2\_'+feature] = np.round(test\_df[feature], 2) train\_df['r1\_'+feature] = np.round(train\_df[feature], 1) test\_df['r1\_'+feature] = np.round(test\_df[feature], 1)

print('Train and test columns: {} {}'.format(len(train\_df.columns), len(test\_df.columns)))

Train and test columns: 626 625

**Model**

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

target = train\_df['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

}

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)]

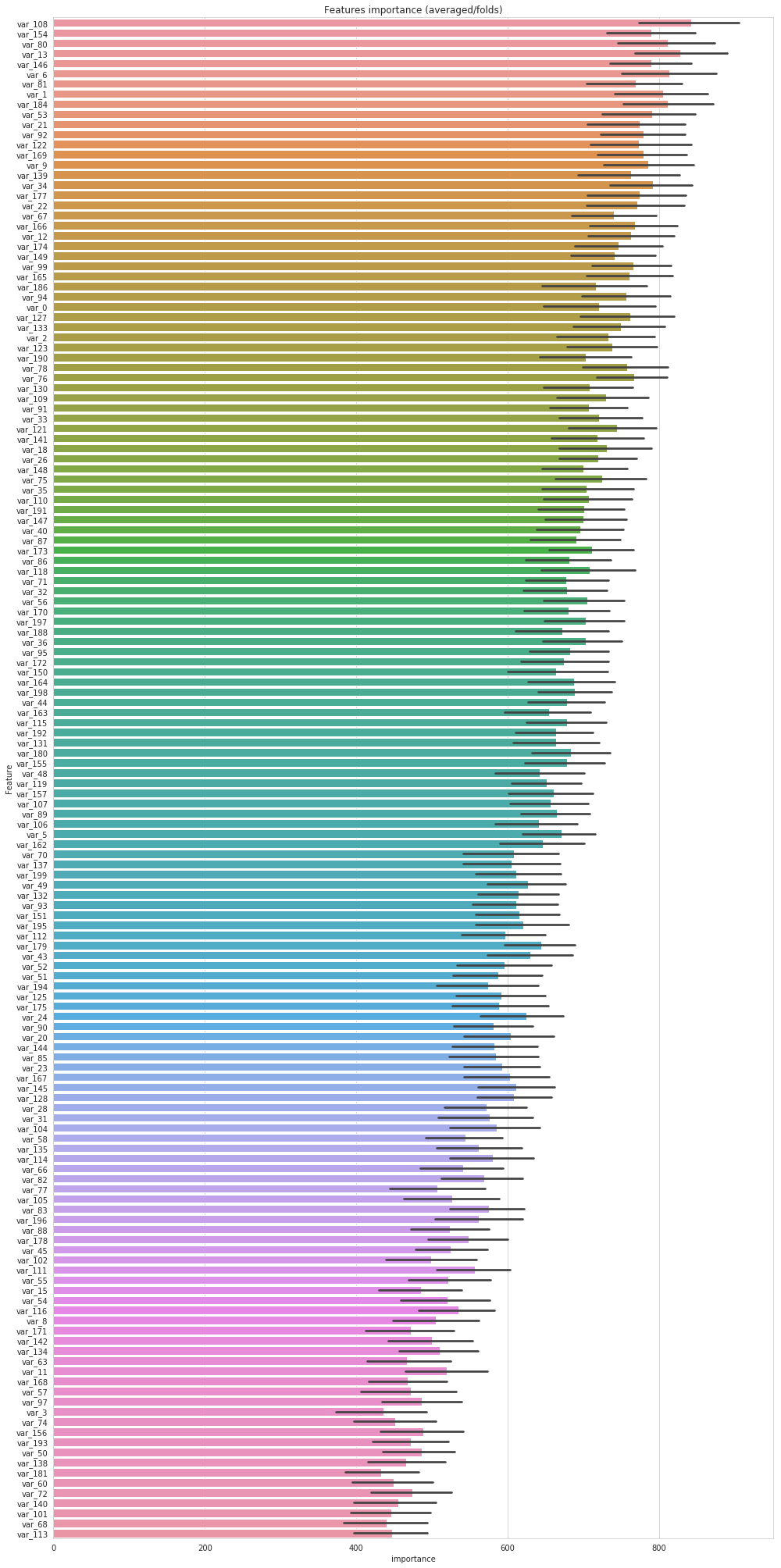
plt.figure(figsize=(14,28))

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

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

plt.tight\_layout()

plt.savefig('FI.png')

****

**Submission**

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

sub\_df["target"] = predictions

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