LOAN DEFAULT ANALYSIS

Data analysis report

Edward Zhang

Word count: 2196

Data preprocessing

The dataset consists of 50000 observations and 54 valid variables. It has a lot of missing values, outliers, and wrong types, etc. The dataset needs some work before we started the data cleaning process.

In columns, we noticed that some of them do not make sense. The CLERK_TYPE only has one category, QUANT_ADDITIONAL_CARD has only zero value, EDUCATION_LEVEL(even has two columns) has almost half of the observation as missing values, and CITY_OF_BIRTH has too many empty strings. The same problems also happened to RESIDENCIAL_PHONE_AREA_CODE, FLAG_MOBILE_PHONE, PROFESSIONAL_STATE, PROFESSIONAL_CITY, PROFESSIONAL_BOROUGH, and so on. There are some binary categorical variables in this dataset, for example, sex, nationality(brazil or not brazil), and a bunch of flags. First I capitalized those columns, so that we won't treat any lower case string (ex. "y","n") as missing. I used 0 and 1 to represent them. For any binary variables that contain the value it should not have, we replace it with a missing value. The reason I did not drop the identity variables is, I am personally curious if that really had a significant influence on the default rate. For some of the variables that have inconsistent missing value types, for example, MARITAL_STATUS and RESIDENCE_TYPE should not have a category of 0, meanwhile, some of the observations have a value of 0, in STATE_OF_BIRTH, it has a lot of empty strings, in RESIDENCIAL_ZIP_3, it has excel type of missing value as a string. We would drop PROFESSIONAL_ZIP_3 as well because it has the same information as the RESIDENCIAL_ZIP_3. APPLICATION_SUBMISSION_TYPE, OCCUPATION_TYPE, and PROFESSION_CODE have too many missing values(>>1%), we need to drop these columns as well.

Now we are ready to split the data into test and train. In the train set, we used the function isnull() to conclude the missing values in the dataset. The categorical variables, SEX, MARITAL_STATUS, RESIDENCE_TYPE, and RESIDENCIAL_ZIP_3, all have missing values, and we replaced them with the mode of each column in the train set. And for missing values in MONTHS_IN_RESIDENCE, we replaced them by the median of this column in the train set. The reason we chose median instead of mean is, the median is more robust to outliers. Another thing we have done here is, transform the multi-categories variable into fewer groups that make much more sense. The variables are, STATE_OF_BIRTH, RESIDENCIAL_STATE, RESIDENCIAL_CITY, MARITAL_STATUS, RESIDENCE_TYPE, RESIDENCIAL_ZIP_3, and RESIDENCIAL_BOROUGH. For each of these columns, we first ranked each category by the default rate, then, re-categorize them into fewer categories based on the rate(ex. Level 1 to 4).

For the test set, we have done the same thing for the missing values as the train set, however, for the multi-category variables, we still ranked them by train set default rate, because we want to keep the decision factor consistent so that each new categories still has the same old multi-categories.

For outliers, we do not perform outliers cleaning on the test set, because we assumed that's part of the reality, we need to take the real-life outliers into account. For the train set, there are some specific variables that we would like to check the outliers, they are QUANT_DEPENDANTS, MONTHS_IN_RESIDENCE, PERSONAL_MONTHLY_INCOME, OTHER_INCOMES, PERSONAL_ASSETS_VALUE, MONTHS_IN_THE_JOB, and AGE. We did not clean all of the outliers, but only the extreme value. Boxplot, violin plot, and distplot helped us a lot to have a general idea of how the data looks like. It has a lot of outliers. By cleaning out the extreme values that I deem it would affect our analysis, the violin plots looked better.

The three variables I designed myself are Total_Income, Total_bank_accounts, and total_type_of_cards. The total_Income, which sums the monthly income and other income, because some people, for example, brokers, may have a low monthly income, but support themselves with commissions. I think the total income can better represent the wealthiness of individuals. The total bank accounts sum the banking accounts and special accounts. People with multiple bank accounts usually have higher savings. By common sense, if you left empty bank accounts, banks would charge you a lot of fees, otherwise, they will terminate them. For this reason, you must have enough savings to support multi-accounts. The assumption we made is, the wealthier group would be less likely to default. The total type of cards sums all the flags of credit cards. The reason is, this proved the person had been approved by different financial institutions. We assume these people would be more willing to repay the loan.

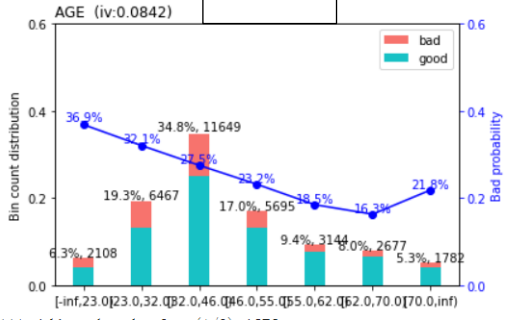
Weight of Evidence and Bins

Weight of Evidence is one way we can normalize the data and make the data perform better. First, we divided data into multiple partitions, called bins. For each observation within each bin, divide the percentage of goods by the percentage of bad. Then we can take natural logarithm of the ratio, the result is our weight of evidence in favour of goods**(BHALLA, 1)**. If woe is larger than 0, hence the bin has more goods than bads, if woe is smaller than 0, hence the bin has fewer goods than bads.

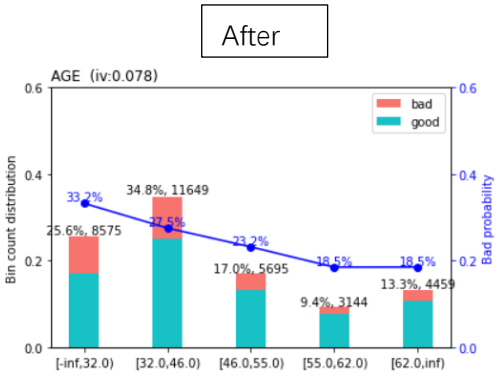
When creating the woe bins for the data, I used the method of tree, and after several modifications, a min percentage of coarse bins equal to 0.05 would give me a decent woe. The total number of observations is not a lot, I believe a minimum percentage of a bin of 0.01 would work. We do not have a lot of variables, so a stop limit 0.02 for information value was set. After testing different max numbers of bins between 5 and 15, 10 is the optimal number for this.

By applying the automatic bins and woe methods over the train set, we noticed that there are some variables of bins we wish to alter. For AGE, the new partition we had was [32,46,55,62], in this way, the higher the bins, the lower the default rate.

Before

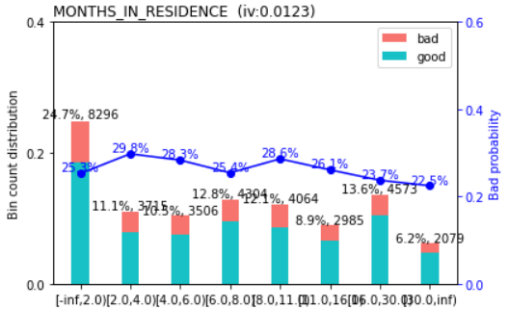


After

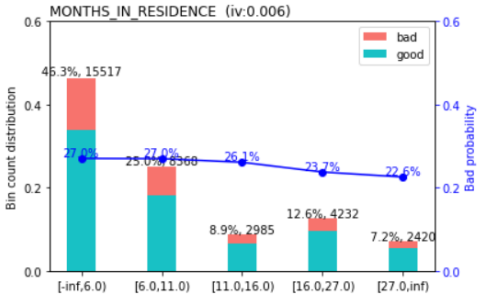


For MONTHS_IN_RESIDENCE, cuts were altered a bit, with the new cuts [6,11,16,27], the default rate shows a minor decrease as the bins got higher.

Before

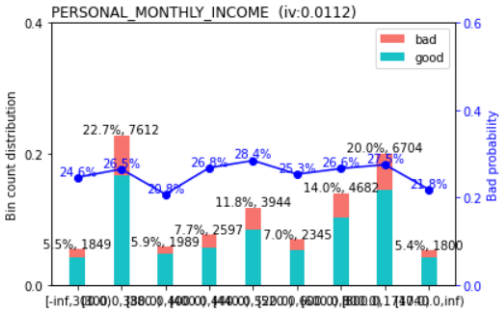


After

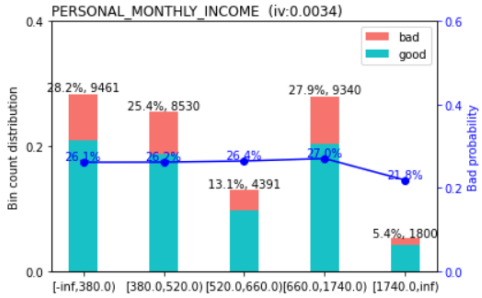


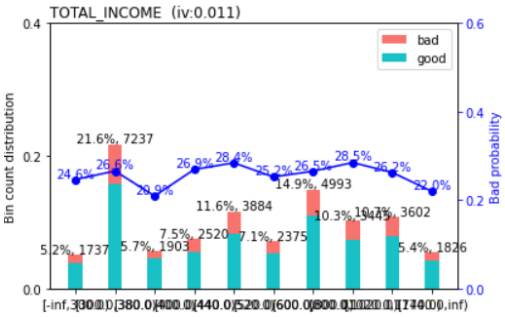
For PERSONAL_MONTHLY_INCOME, many different partitions were tried, the one that most makes sense is [380,520,660,1740]. When the income is over 1740, the bad rate drastically decreased. This trend also applied to the TOTAL_INCOME.

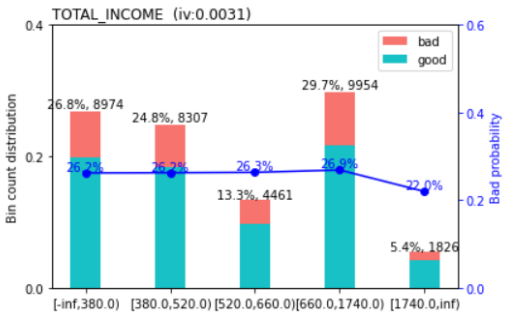
Before



After





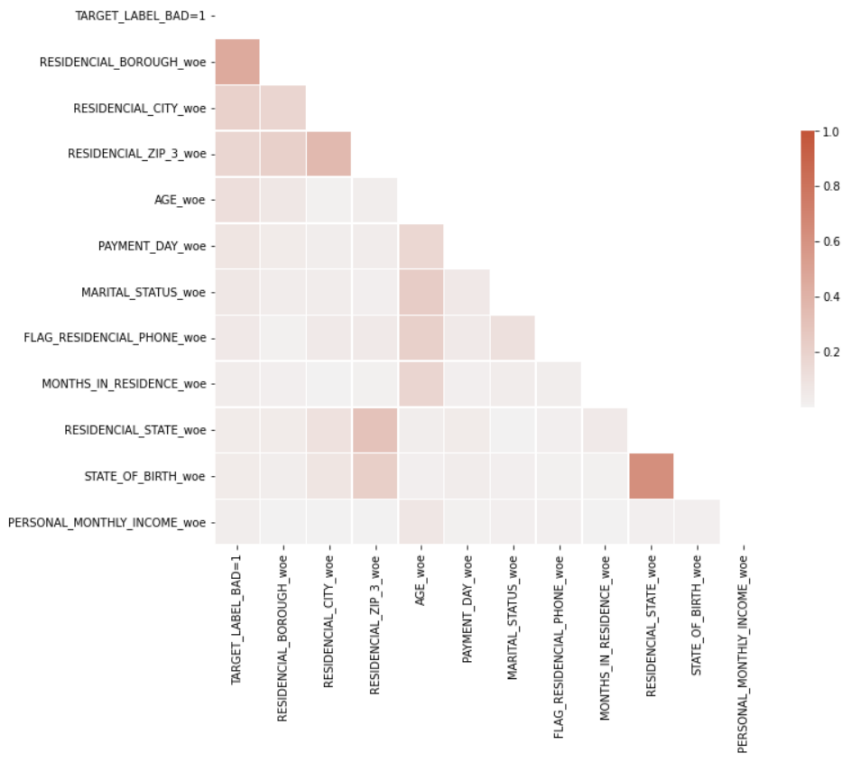


For other variables, the bins look just fine. We select the variables based on Information Value. The information value is a very useful measure to perform variable selections, it also ranks the variables based on importance**(BHALLA, 1)**. We noticed most variables on the list have relatively small information values.

	variable	info_value
7	RESIDENCIAL_BOROUGH_woe	1.029684
29	RESIDENCIAL_CITY_woe	0.364442
3	RESIDENCIAL_ZIP_3_woe	0.165772
4	AGE_woe	0.078019
23	PAYMENT_DAY_woe	0.032268
6	MARITAL_STATUS_woe	0.023044
25	FLAG_RESIDENCIAL_PHONE_woe	0.017929
14	RESIDENCIAL_STATE_woe	0.011005
26	STATE_OF_BIRTH_woe	0.008732
1	MONTHS_IN_RESIDENCE_woe	0.006032
28	RESIDENCE_TYPE_woe	0.003936
20	PERSONAL_MONTHLY_INCOME_woe	0.003447
19	TOTAL_INCOME_woe	0.003141
21	QUANT_DEPENDANTS_woe	0.002054
9	FLAG_MASTERCARD_woe	0.002036
2	FLAG_PROFESSIONAL_PHONE_woe	0.001410
24	SEX_woe	0.001336
5	TOTAL_TYPE_OF_CARDS_woe	0.000809
12	COMPANY_woe	0.000757
15	QUANT_CARS_woe	0.000340
11	FLAG_EMAIL_woe	0.000232
8	QUANT_BANKING_ACCOUNTS_woe	0.000225
13	QUANT_SPECIAL_BANKING_ACCOUNTS_woe	0.000225
10	TOTAL_BANKING_ACCOUNTS_woe	0.000225
31	FLAG_VISA_woe	0.000181
22	PRODUCT_woe	0.000078
30	NACIONALITY_woe	0.000000
0	FLAG_OTHER_CARDS_woe	0.000000
16	POSTAL_ADDRESS_TYPE_woe	0.000000
27	OTHER_INCOMES_woe	0.000000
18	FLAG_AMERICAN_EXPRESS_woe	0.000000
17	PERSONAL_ASSETS_VALUE_woe	0.000000
32	FLAG_DINERS_woe	0.000000

Only 6 variables will be kept for a strict 0.02 cut-off point. Since we do not have any other stronger variables in this dataset, we set the cut-off point to 0.003 instead. We also discard the TOTAL_INCOME, because it has a serious correlation problem with PERSONAL_MONTHLY_INCOME, we kept the one with a higher information value.

In the correlation matrix, we found out the correlation is not a serious problem in this case, except for STATE_OF_BIRTH and RESIDENCIAL_STATE. However, we still keep them, the logistic regression penalty will regularize them later if necessary.



Logistic Regression

Logistic Regression is perfect for the binary response variable. When building our model, I tried two penalty methods, LASSO, and Elastic Net, it turns out that LASSO is not the optimal method for our data as it regularized most of our coefficients to 0. Elastic Net was chosen. We set the lambda to 13:1, as we tried different lambda, and 13 would give the most accurate prediction based on the confusion matrix. Again, we do not have many observations, a tolerance for the lambda is set to 0, and 5 folds for cross-validation is enough. We fixed the random state to studentID, and max iterations to 150, LASSO ratio to 0 to 1.001 with 0.01 as increase would let the model converge after I gave several tries.

The model regularized some of the variables and coefficients are shown below.

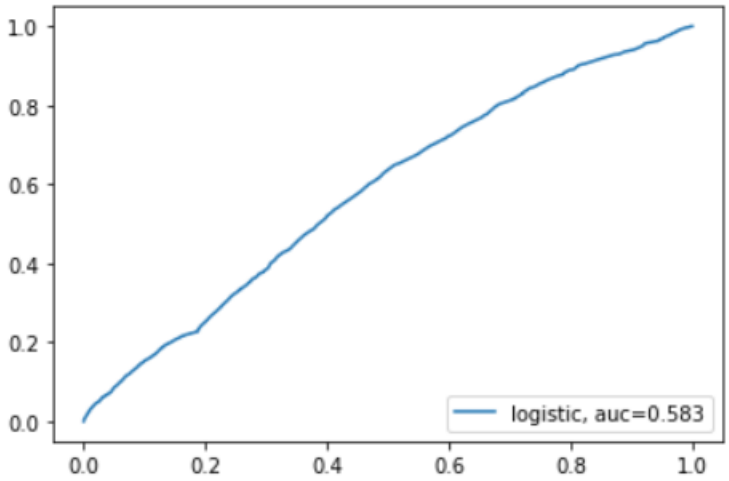
	column	0
0	RESIDENCIAL_BOROUGH_woe	0.871406
1	RESIDENCIAL_CITY_woe	0.573707
2	RESIDENCIAL_ZIP_3_woe	0.159891
3	AGE_woe	0.412604
4	PAYMENT_DAY_woe	0.134145
5	MARITAL_STATUS_woe	0.024869
6	FLAG_RESIDENCIAL_PHONE_woe	0.235135
7	MONTHS_IN_RESIDENCE_woe	0.000000
8	RESIDENCIAL_STATE_woe	0.000000
9	STATE_OF_BIRTH_woe	0.000000
10	PERSONAL_MONTHLY_INCOME_woe	0.000000

The coefficient returned shows that every variable with an information value smaller than and not near to 0.02 was regularized to zero. Except that, I considered our variable selections are appropriate. However, based on the result, I only recommend using the top 7 variables on this list.

The intercept it returns is 0.00377898. That is an excellent result as it is very close to 0. The final LASSO ratio and lambda it returns are 0.2 and 0.00215443 respectively. Hence, the model has 0.2 LASSO penalty and 0.8 for RIDGE. The confusion matrix does not look very balanced, however, that's the best one we can obtain after multiple tries.

```
array([[9009, 2079],
       [3020,  892]], dtype=int64)
```

We can plot a ROC curve to check the accuracy of the model.



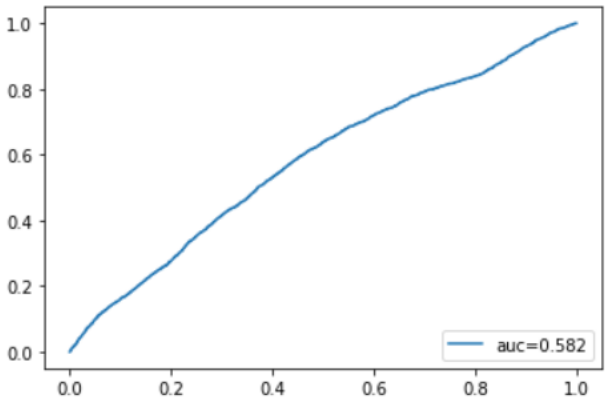
The AUC we obtained is 0.583, hence, the model is not very predictive, but still generally makes sense. For the scorecard, after multiple tries, a base point of 700, base odds as 0.5 bads to goods, and a PDO(points to double the odds) as 50 would give us a reasonable range of scorecards. We then applied it to the test set. The result summary is shown below.

	score
count	15000.000000
mean	678.310533
std	67.147868
min	510.000000
25%	669.000000
50%	701.000000
75%	712.000000
max	833.000000

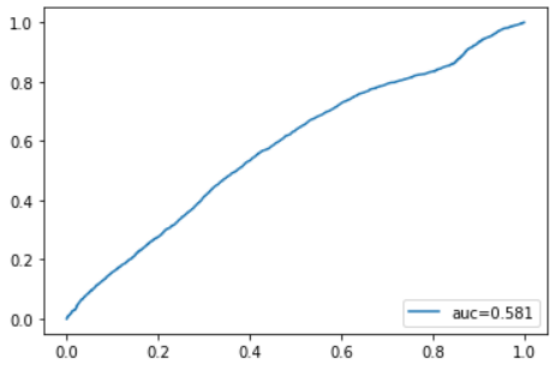
XGB and RF

Random Forest is a method that was based on decision trees, but with less variance by bootstrapping**(Yiu, 2019)**. For RF, I used cross-validation to find the appropriate parameter. 3000 is the optimal result for

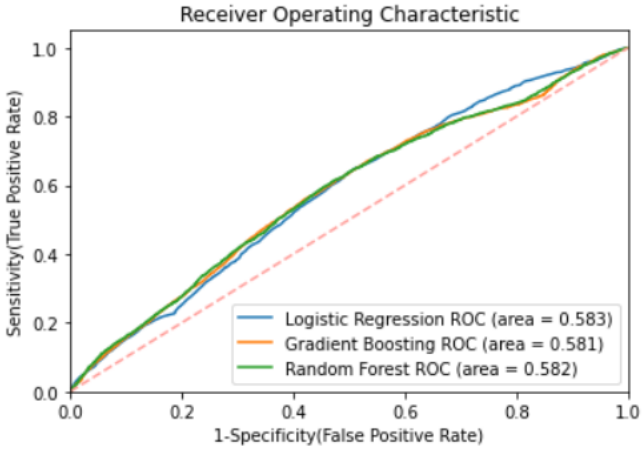
the number of trees to train after performing the CV, which gives the best AUC. After applying the model to the test data without woe, we have a ROC curve with AUC as 0.582 as below.



XGBoosting makes a few small correlated trees instead of using a large number of trees. We also use cross-validation to find the appropriate parameters, learning_rate, trees to train, and max depth. After two times 5-folds CV, it turns out that 0.15 learning_rate, 150 trees with a max depth of 2 would have the highest accuracy for our data. After applying the model to the test data without woe, we have a ROC curve with AUC as 0.581 as below.



By comparing the AUC of all three models, we noticed that the best performing one is the logistic regression with an AUC of 0.583, meanwhile, the XGBoosting has the lowest AUC of 0.581, even though the difference is extremely small.



The random forest requires a large amount of data to perform well, while XGBoosting only needs small samples, it is normal for those two models having similar AUC with 35000 observations. In theory, they should have similar accuracy as well. One interesting finding is, when the rate got higher, the log model outperforms the other two models, while the other two outperform the log model when the rate is low. And the ROC of RF and XGB look very similar.

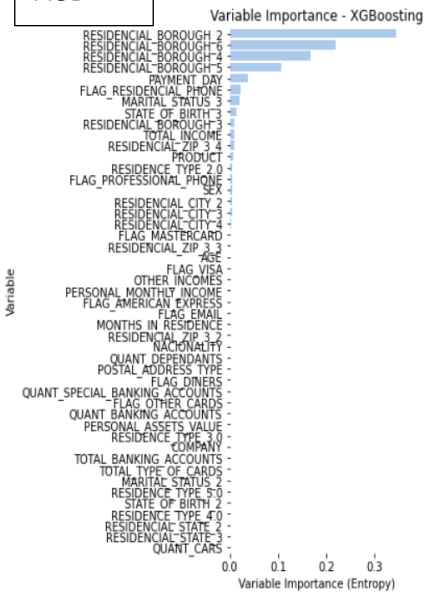
Variable Importance

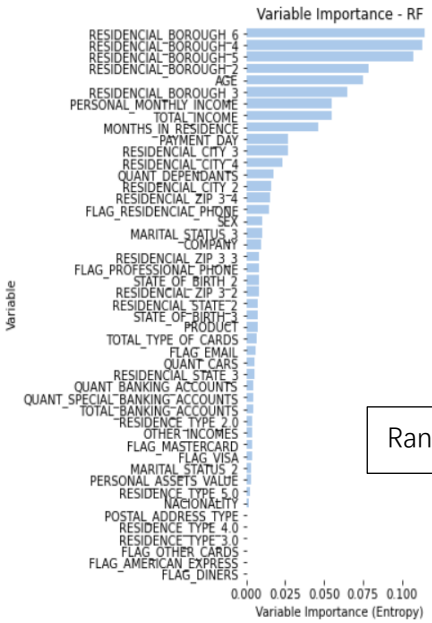
We use coefficients to determine the variables' importance in logistic regression instead, and Entropy for RF and XGB.

Logistic Regression

	column	0
0	RESIDENCIAL_BOROUGH_woe	0.871406
1	RESIDENCIAL_CITY_woe	0.573707
2	RESIDENCIAL_ZIP_3_woe	0.159891
3	AGE_woe	0.412604
4	PAYMENT_DAY_woe	0.134145
5	MARITAL_STATUS_woe	0.024869
6	FLAG_RESIDENCIAL_PHONE_woe	0.235135
7	MONTHS_IN_RESIDENCE_woe	0.000000
8	RESIDENCIAL_STATE_woe	0.000000
9	STATE_OF_BIRTH_woe	0.000000
10	PERSONAL_MONTHLY_INCOME_woe	0.000000

XGB





Random Forest

The rank of importance does not agree with each other except the borough is always the most important one. The RF has kept a lot of variables which the Entropies are closer than other models, this is expected since the RF is the model with the most complexity. XGBoosting is usually a simpler model than the RF and more robust to smaller data set, thus it surely has less importance for variables ranked behind. The

logistic regression consists of the LASSO penalty, which will largely eliminate less useful variables, this is the reason that it has only non-zero coefficients for variables ranked higher and the fewest features.

Two-cut-off point

Scorecard range for the test set.

	score
count	15000.000000
mean	678.310533
std	67.147868
min	510.000000
25%	669.000000
50%	701.000000
75%	712.000000
max	833.000000

	cutoff	Accepted_rate	Accuracy_good	Accuracy_bad	Accuracy_Total	Avg_cost_g	Avg_cost_b	total_cost_g	total_cost_b	total_cost
0	520.0	0.9942	0.996392	0.0120143	0.739667	4.3945	3.8343	136.23	14819.6	14955.8
1	530.0	0.975933	0.982413	0.0421779	0.7372	4.3413	3.71781	833.529	13926.9	14760.4
2	540.0	0.958933	0.96645	0.0608384	0.730267	4.26847	3.63509	1532.38	13333.5	14865.9
3	550.0	0.947267	0.954816	0.0738753	0.725067	4.22516	3.59315	2083	13014.4	15097.4
4	560.0	0.902867	0.915404	0.132413	0.7112	4.09893	3.39594	3828.4	11522.4	15350.8
5	570.0	0.8476	0.862825	0.187883	0.6868	3.90422	3.14824	5610.36	9907.52	15517.9
6	580.0	0.812467	0.824766	0.22137	0.6674	3.56697	3.0263	6869.99	9206.02	16076
7	590.0	0.802133	0.8125	0.227249	0.659867	3.51917	3.00787	7316.36	9092.8	16409.2
8	600.0	0.802133	0.8125	0.227249	0.659867	3.51917	3.00787	7316.36	9092.8	16409.2
9	610.0	0.802133	0.8125	0.227249	0.659867	3.51917	3.00787	7316.36	9092.8	16409.2
10	620.0	0.802133	0.8125	0.227249	0.659867	3.51917	3.00787	7316.36	9092.8	16409.2
11	630.0	0.802133	0.8125	0.227249	0.659867	3.51917	3.00787	7316.36	9092.8	16409.2
12	640.0	0.802133	0.8125	0.227249	0.659867	3.51917	3.00787	7316.36	9092.8	16409.2
13	650.0	0.801933	0.8125	0.227761	0.66	3.51917	3.00617	7316.36	9078.62	16395
14	660.0	0.7864	0.799964	0.249489	0.6564	3.47876	2.92889	7663.72	8569.92	16233.6
15	670.0	0.742867	0.761364	0.307515	0.643	3.15289	2.73478	8314.18	7386.65	15700.8
16	680.0	0.701267	0.722763	0.350716	0.625733	2.97125	2.55644	8982.08	6403.89	15386
17	690.0	0.6726	0.694715	0.388548	0.614867	2.86865	2.4644	9644.42	5880.06	15524.5
18	700.0	0.508067	0.541126	0.582822	0.552	2.37293	1.72362	12021.3	2793.99	14815.3
19	710.0	0.2662	0.294913	0.787065	0.423267	1.31451	0.67466	9766.84	487.779	10254.6
20	720.0	0.116733	0.131584	0.922035	0.337733	0.542966	0.265061	5170.12	77.3979	5247.52
21	730.0	0.051	0.0557359	0.962423	0.2922	0.216072	0.118407	2262.27	17.4058	2279.68
22	740.0	0.051	0.0557359	0.962423	0.2922	0.216072	0.118407	2262.27	17.4058	2279.68
23	750.0	0.051	0.0557359	0.962423	0.2922	0.216072	0.118407	2262.27	17.4058	2279.68
24	760.0	0.051	0.0557359	0.962423	0.2922	0.216072	0.118407	2262.27	17.4058	2279.68
25	770.0	0.051	0.0557359	0.962423	0.2922	0.216072	0.118407	2262.27	17.4058	2279.68
26	780.0	0.051	0.0557359	0.962423	0.2922	0.216072	0.118407	2262.27	17.4058	2279.68
27	790.0	0.0509333	0.0556457	0.962423	0.292133	0.215921	0.118407	2260.69	17.4058	2278.1
28	800.0	0.0466	0.0515873	0.96728	0.2904	0.181332	0.105433	1906.52	13.39	1919.91
29	810.0	0.0291333	0.0330988	0.978016	0.279533	0.118773	0.0639326	1268.74	4.47528	1273.21
30	820.0	0.0124	0.0147006	0.992843	0.2698	0.0535452	0.0278808	583.857	0.641258	584.498
31	830.0	0.00333333	0.00414863	0.998978	0.2636	0.0130791	0.00253175	144.419	0.010127	144.429

- The accepted rate is accepted customers divided by total customers
- The accuracy of good is accepted good customers divided by total good customers

- The accuracy of bad is rejected bad customers divided by total bad customers
- The accuracy of Total is the sum of accepted good customers and rejected bad customers divided by total customers
- The average cost of goods is the sum of all accepted good customers monthly income multiplied by utilization and the monthly interest rate and divided by total good customers, that is, the average cost for all rejected good customers
- The average cost of bad is the sum of all accepted bad customers monthly income multiplied by utilization and the monthly interest rate and divided by total bad customers, that is, the average cost for all accepted bad customers
- The Total cost of good is the average cost of good multiplied by all rejected good
- The Total cost of bad is the average cost of bad multiplied by all accepted bad
- The total cost is simply the sum of the above two

Here we only care about the cost.
By observing the cut-off table, 660 is the point where I designed for immediate rejection, while 720 is the point I designed for immediate acceptance. Any individuals between the range will need further investigation.
For points lower than 660, the total cost drastically increased to 16000 while for points higher than 720, the total cost is as low as 5250. And the points in the range are ambiguous because the accuracy of bad started to drastically decrease in that range. This is the basic rationale of how I make the cut.

Word count: 2196

REFERENCE:

Bhalla, D. (n.d.). *Weight of evidence (WOE) and information value (iv) explained*. ListenData. Retrieved November 21, 2021, from https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html#What-is-Information-Value-IV-.

Yiu, T. (2021, September 29). *Understanding random forest*. Medium. Retrieved November 21, 2021, from https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

coursework2_ba

November 25, 2021

Assignmen 2

[1]:
```
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
```

0.0.1 Question 1: Data cleaning

[9]:
```
data = pd.read_csv('CC_Modelling_Data.csv')
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2718:
DtypeWarning: Columns (51,52) have mixed types.Specify dtype option on import or
set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[5]:
```
columns = pd.ExcelFile(r"CC_VariablesList.xls")
```

[6]:
```
data.columns = columns.parse(0).iloc[:,1]
```

[11]:
```
data.describe(include='all')
```

[11]:

	0	1	2	3	4	5	\
count	50000.000000	50000	50000.000000	50000	50000.0	50000.000000	
unique	NaN	1	NaN	3	NaN	NaN	
top	NaN	C	NaN	Web	NaN	NaN	
freq	NaN	50000	NaN	28206	NaN	NaN	
mean	25000.500000	NaN	12.869920	NaN	0.0	1.006540	
std	14433.901067	NaN	6.608385	NaN	0.0	0.080606	
min	1.000000	NaN	1.000000	NaN	0.0	1.000000	
25%	12500.750000	NaN	10.000000	NaN	0.0	1.000000	
50%	25000.500000	NaN	10.000000	NaN	0.0	1.000000	
75%	37500.250000	NaN	15.000000	NaN	0.0	1.000000	
max	50000.000000	NaN	25.000000	NaN	0.0	2.000000	

	6	7	8	9	10	11	12	\

	0	1	2	3	4	5	6
count	50000	50000.00000	50000.000000	50000.0	50000	50000	50000.000000
unique	4	NaN	NaN	NaN	29	9910	NaN
top	F	NaN	NaN	NaN	BA		NaN
freq	30805	NaN	NaN	NaN	5717	2064	NaN
mean	NaN	2.14840	0.650520	0.0	NaN	NaN	0.961600
std	NaN	1.32285	1.193655	0.0	NaN	NaN	0.202105
min	NaN	0.00000	0.000000	0.0	NaN	NaN	0.000000
25%	NaN	1.00000	0.000000	0.0	NaN	NaN	1.000000
50%	NaN	2.00000	0.000000	0.0	NaN	NaN	1.000000
75%	NaN	2.00000	1.000000	0.0	NaN	NaN	1.000000
max	NaN	7.00000	53.000000	0.0	NaN	NaN	2.000000

	13	14	15	16	17	18	19
count	50000	50000	50000	50000	50000	48651.000000	46223.000000
unique	27	3529	14511	2	102	NaN	NaN
top	SP	Sao Paulo	CENTRO	Y		NaN	NaN
freq	8773	894	4169	41809	8212	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	1.252225	9.727149
std	NaN	NaN	NaN	NaN	NaN	0.867833	10.668841
min	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000
25%	NaN	NaN	NaN	NaN	NaN	1.000000	1.000000
50%	NaN	NaN	NaN	NaN	NaN	1.000000	6.000000
75%	NaN	NaN	NaN	NaN	NaN	1.000000	15.000000
max	NaN	NaN	NaN	NaN	NaN	5.000000	228.000000

	20	21	22	23	24
count	50000	50000.000000	50000.000000	50000.000000	50000.000000
unique	1	NaN	NaN	NaN	NaN
top	N	NaN	NaN	NaN	NaN
freq	50000	NaN	NaN	NaN	NaN
mean	NaN	0.802280	886.678437	35.434760	0.111440
std	NaN	0.398284	7846.959327	891.515142	0.314679
min	NaN	0.000000	60.000000	0.000000	0.000000
25%	NaN	1.000000	360.000000	0.000000	0.000000
50%	NaN	1.000000	500.000000	0.000000	0.000000
75%	NaN	1.000000	800.000000	0.000000	0.000000
max	NaN	1.000000	959000.000000	194344.000000	1.000000

	25	26	27	28	29
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000
unique	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN
mean	0.097460	0.001320	0.001740	0.002040	0.357840
std	0.296586	0.036308	0.041677	0.045121	0.479953
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000

```
50%         0.000000      0.000000      0.000000      0.000000      0.000000
75%         0.000000      0.000000      0.000000      0.000000      1.000000
max         1.000000      1.000000      1.000000      1.000000      2.000000

                  30            31            32     33     34          35  \
count    50000.000000  5.000000e+04  50000.000000  50000  50000       16217
unique            NaN           NaN           NaN      2     28        2236
top               NaN           NaN           NaN      N          FORTALEZA
freq              NaN           NaN           NaN  27959  34307         419
mean         0.357840  2.322372e+03      0.336140    NaN    NaN         NaN
std          0.479953  4.235798e+04      0.472392    NaN    NaN         NaN
min          0.000000  0.000000e+00      0.000000    NaN    NaN         NaN
25%          0.000000  0.000000e+00      0.000000    NaN    NaN         NaN
50%          0.000000  0.000000e+00      0.000000    NaN    NaN         NaN
75%          1.000000  0.000000e+00      1.000000    NaN    NaN         NaN
max          2.000000  6.000000e+06      1.000000    NaN    NaN         NaN

            36     37     38            39            40            41  \
count    16217  50000  50000  50000.000000  42244.000000  42687.000000
unique    5057      2     87           NaN           NaN           NaN
top     CENTRO      N                  NaN           NaN           NaN
freq      3727  36510  36532           NaN           NaN           NaN
mean       NaN    NaN    NaN      0.009320      8.061784      2.484316
std        NaN    NaN    NaN      0.383453      3.220104      1.532261
min        NaN    NaN    NaN      0.000000      0.000000      0.000000
25%        NaN    NaN    NaN      0.000000      9.000000      1.000000
50%        NaN    NaN    NaN      0.000000      9.000000      2.000000
75%        NaN    NaN    NaN      0.000000      9.000000      4.000000
max        NaN    NaN    NaN     35.000000     18.000000      5.000000

                  42            43       44       45       46       47  \
count   21116.000000  17662.000000  50000.0  50000.0  50000.0  50000.0
unique           NaN           NaN      NaN      NaN      NaN      NaN
top              NaN           NaN      NaN      NaN      NaN      NaN
freq             NaN           NaN      NaN      NaN      NaN      NaN
mean        3.797926      0.296003      0.0      0.0      0.0      0.0
std         5.212168      0.955688      0.0      0.0      0.0      0.0
min         0.000000      0.000000      0.0      0.0      0.0      0.0
25%         0.000000      0.000000      0.0      0.0      0.0      0.0
50%         0.000000      0.000000      0.0      0.0      0.0      0.0
75%        11.000000      0.000000      0.0      0.0      0.0      0.0
max        17.000000      5.000000      0.0      0.0      0.0      0.0

                  48     49           50       51       52            53
count   50000.000000  50000  50000.00000  50000.0  50000.0  50000.000000
unique           NaN      1          NaN   1481.0   1481.0           NaN
top              NaN      N          NaN    960.0    960.0           NaN
```

```
freq             NaN  50000          NaN    721.0    721.0          NaN
mean        1.275700    NaN     43.24852      NaN      NaN     0.260820
std         0.988286    NaN     14.98905      NaN      NaN     0.439086
min         1.000000    NaN      6.00000      NaN      NaN     0.000000
25%         1.000000    NaN     31.00000      NaN      NaN     0.000000
50%         1.000000    NaN     41.00000      NaN      NaN     0.000000
75%         1.000000    NaN     53.00000      NaN      NaN     1.000000
max         7.000000    NaN    106.00000      NaN      NaN     1.000000
```

[165]:
```
#drop clerk type
df=data
df=df.drop(columns='CLERK_TYPE')
```

[166]:
```
#web=0, carga=1
df['APPLICATION_SUBMISSION_TYPE']=df.loc[:,'APPLICATION_SUBMISSION_TYPE'].str.
 ↪upper()
for i in range(50000):
    if df.loc[i,'APPLICATION_SUBMISSION_TYPE']=='WEB':
        df.loc[i,'APPLICATION_SUBMISSION_TYPE']=0
    elif df.loc[i,'APPLICATION_SUBMISSION_TYPE']=='CARGA':
        df.loc[i,'APPLICATION_SUBMISSION_TYPE']=1
    else:
        df.loc[i,'APPLICATION_SUBMISSION_TYPE']=np.nan
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1763:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(loc, value)
```

[167]:
```
df.loc[:,'APPLICATION_SUBMISSION_TYPE']=df.loc[:,'APPLICATION_SUBMISSION_TYPE'].
 ↪astype(float)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1743:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(ilocs[0], value)
```

[168]:
```
#quant additional card
df=df.drop(columns='QUANT_ADDITIONAL_CARDS')
```

[169]:
```
#drop postal address type
df.loc[:,'POSTAL_ADDRESS_TYPE']=df.loc[:,'POSTAL_ADDRESS_TYPE'].astype(float)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1743:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(ilocs[0], value)
```

[170]:
```
#sex M=0,F=1
df['SEX']=df.loc[:,'SEX'].str.upper()
for i in range(50000):
    if df.loc[i,'SEX']=='M':
        df.loc[i,'SEX']=0
    elif df.loc[i,'SEX']=='F':
        df.loc[i,'SEX']=1
    else:
        df.loc[i,'SEX']=np.nan
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1763:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(loc, value)
```

[171]:
```
df.loc[:,'SEX']=df.loc[:,'SEX'].astype(float)
```

[172]:
```
#MARITAL STATUS needs more work after splitting
df.loc[:,'MARITAL_STATUS']=df.loc[:,'MARITAL_STATUS'].astype(str)
for i in range(50000):
    if df.loc[i,'MARITAL_STATUS']=='0':
        df.loc[i,'MARITAL_STATUS']=np.nan
```

[173]:
```
#drop Education level(duplicate and too many missing)
df=df.drop(columns='EDUCATION_LEVEL')
```

[174]:
```
#STATE_OF_BIRTH needs work after splitting

#replace empty strings with 'NA'
for i in range(50000):
    if not df.loc[i,'STATE_OF_BIRTH'].strip():
        df.loc[i,'STATE_OF_BIRTH']='NA'
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1763:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(loc, value)
```

[175]:
```
#drop CITY_OF_BIRTH (too many empty strings)
df=df.drop(columns='CITY_OF_BIRTH')
```

[176]:
```
#NACIONALITY  brazil=1, foreign=0
df.loc[:,'NACIONALITY']=df.loc[:,'NACIONALITY'].astype(int)
for i in range(50000):
    if df.loc[i,'NACIONALITY']==1:
        df.loc[i,'NACIONALITY']=1
    elif df.loc[i,'NACIONALITY']==0:
        df.loc[i,'NACIONALITY']=0
    elif df.loc[i,'NACIONALITY']==2:
        df.loc[i,'NACIONALITY']=0
    else:
        df.loc[i,'NACIONALITY']=np.nan
```

[177]:
```
#RESIDENCIAL_BOROUGH(needs more work after spliting)
for i in range(50000):
    if not df.loc[i,'RESIDENCIAL_BOROUGH'].strip():
        df.loc[i,'RESIDENCIAL_BOROUGH']='NA'
```

[178]:
```
#FLAG_RESIDENCIAL_PHONE Y=1, N=0
df['FLAG_RESIDENCIAL_PHONE']=df.loc[:,'FLAG_RESIDENCIAL_PHONE'].str.upper()
for i in range(50000):
```

```
    if df.loc[i,'FLAG_RESIDENCIAL_PHONE']=='N':
        df.loc[i,'FLAG_RESIDENCIAL_PHONE']=0
    elif df.loc[i,'FLAG_RESIDENCIAL_PHONE']=='Y':
        df.loc[i,'FLAG_RESIDENCIAL_PHONE']=1
    else:
        df.loc[i,'FLAG_RESIDENCIAL_PHONE']=np.nan
```

[179]:
```
df.loc[:,'FLAG_RESIDENCIAL_PHONE']=df.loc[:,'FLAG_RESIDENCIAL_PHONE'].
↪astype(float)
```

[180]:
```
#drop RESIDENCIAL_PHONE_AREA_CODE(Not helpful, hard to conclude, too many␣
↪missing)
df=df.drop(columns='RESIDENCIAL_PHONE_AREA_CODE')
```

[181]:
```
#RESIDENCE_TYPE (needs work after splitting)
for i in range(50000):
    if df.loc[i,'RESIDENCE_TYPE']==0:
        df.loc[i,'RESIDENCE_TYPE']=np.nan
```

[182]:
```
#MONTHS_IN_RESIDENCE (needs work after splitting)
df.loc[:,'MONTHS_IN_RESIDENCE']=df.loc[:,'MONTHS_IN_RESIDENCE'].astype(float)
```

[183]:
```
#drop FLAG_MOBILE_PHONE (all of them are 'N')
df=df.drop(columns='FLAG_MOBILE_PHONE')
```

[184]:
```
#FLAG_EMAIL
df.loc[:,'FLAG_EMAIL']=df.loc[:,'FLAG_EMAIL'].astype(float)
```

[185]:
```
#PERSONAL_MONTHLY_INCOME
df.loc[:,'PERSONAL_MONTHLY_INCOME']=df.loc[:,'PERSONAL_MONTHLY_INCOME'].
↪astype(float)
```

[186]:
```
#OTHER_INCOMES
df.loc[:,'OTHER_INCOMES']=df.loc[:,'OTHER_INCOMES'].astype(float)
```

[187]:
```
#FLAG_VISA
df.loc[:,'FLAG_VISA']=df.loc[:,'FLAG_VISA'].astype(float)
```

[188]:
```
#FLAG_MASTERCARD
df.loc[:,'FLAG_MASTERCARD']=df.loc[:,'FLAG_MASTERCARD'].astype(float)
```

[189]:
```
#FLAG_DINERS
df.loc[:,'FLAG_DINERS']=df.loc[:,'FLAG_DINERS'].astype(float)
```

[190]:
```
#FLAG_AMERICAN_EXPRESS
df.loc[:,'FLAG_AMERICAN_EXPRESS']=df.loc[:,'FLAG_AMERICAN_EXPRESS'].
↪astype(float)
```

[191]:
```
#FLAG_OTHER_CARDS
df.loc[:,'FLAG_OTHER_CARDS']=df.loc[:,'FLAG_OTHER_CARDS'].astype(float)
```

[192]:
```
#QUANT_BANKING_ACCOUNTS
df.loc[:,'QUANT_BANKING_ACCOUNTS']=df.loc[:,'QUANT_BANKING_ACCOUNTS'].
 ↪astype(float)
```

[193]:
```
#QUANT_SPECIAL_BANKING_ACCOUNTS
df.loc[:,'QUANT_SPECIAL_BANKING_ACCOUNTS']=df.loc[:
 ↪,'QUANT_SPECIAL_BANKING_ACCOUNTS'].astype(float)
```

[194]:
```
#PERSONAL_ASSETS_VALUE
df.loc[:,'PERSONAL_ASSETS_VALUE']=df.loc[:,'PERSONAL_ASSETS_VALUE'].
 ↪astype(float)
```

[195]:
```
#QUANT_CARS
df.loc[:,'QUANT_CARS']=df.loc[:,'QUANT_CARS'].astype(float)
```

[196]:
```
#COMPANY Y=1, N=0
df['COMPANY']=df.loc[:,'COMPANY'].str.upper()
for i in range(50000):
    if df.loc[i,'COMPANY']=='N':
        df.loc[i,'COMPANY']=float(0)
    elif df.loc[i,'COMPANY']=='Y':
        df.loc[i,'COMPANY']=float(1)
    else:
        df.loc[i,'COMPANY']=np.nan
```

[197]:
```
df.loc[:,'COMPANY']=df.loc[:,'COMPANY'].astype(float)
```

[198]:
```
#drop PROFESSIONAL_STATE (most are blank)
df=df.drop(columns='PROFESSIONAL_STATE')
```

[199]:
```
#drop PROFESSIONAL_CITY (a lot of them are missing)
df=df.drop(columns='PROFESSIONAL_CITY')
```

[200]:
```
#drop PROFESSIONAL_BOROUGH for the same reason
df=df.drop(columns='PROFESSIONAL_BOROUGH')
```

[201]:
```
#FLAG_PROFESSIONAL_PHONE N=0, Y=1
df['FLAG_PROFESSIONAL_PHONE']=df.loc[:,'FLAG_PROFESSIONAL_PHONE'].str.upper()
for i in range(50000):
    if df.loc[i,'FLAG_PROFESSIONAL_PHONE']=='N':
        df.loc[i,'FLAG_PROFESSIONAL_PHONE']=float(0)
    elif df.loc[i,'FLAG_PROFESSIONAL_PHONE']=='Y':
        df.loc[i,'FLAG_PROFESSIONAL_PHONE']=float(1)
    else:
```

```
        df.loc[i,'FLAG_PROFESSIONAL_PHONE']=np.nan
```

[202]:
```
df.loc[:,'FLAG_PROFESSIONAL_PHONE']=df.loc[:,'FLAG_PROFESSIONAL_PHONE'].
↪astype(float)
```

[203]:
```
#drop PROFESSIONAL_PHONE_AREA_CODE (a lot of them are missing and hard to␣
↪conclude)
df=df.drop(columns='PROFESSIONAL_PHONE_AREA_CODE')
```

[204]:
```
#MONTHS_IN_THE_JOB
df.loc[:,'MONTHS_IN_THE_JOB']=df.loc[:,'MONTHS_IN_THE_JOB'].astype(float)
```

[205]:
```
#PROFESSION_CODE (also needs work after splitting)
df.loc[:,'PROFESSION_CODE']=df.loc[:,'PROFESSION_CODE'].astype(float)
```

[206]:
```
#OCCUPATION_TYPE(needs work after splitting)
df.loc[:,'OCCUPATION_TYPE']=df.loc[:,'OCCUPATION_TYPE'].astype(float)
```

[207]:
```
#drop MATE_PROFESSION_CODE (too many missing)
df=df.drop(columns='MATE_PROFESSION_CODE')
```

[208]:
```
#FLAG_HOME_ADDRESS_DOCUMENT(all 0)
df=df.drop(columns='FLAG_HOME_ADDRESS_DOCUMENT')
```

[209]:
```
#FLAG_RG(for the same reason)
df=df.drop(columns='FLAG_RG')
```

[210]:
```
#FLAG_CPF(same reason)
df=df.drop(columns='FLAG_CPF')
```

[211]:
```
#FLAG_INCOME_PROOF(same reason)
df=df.drop(columns='FLAG_INCOME_PROOF')
```

[212]:
```
#PRODUCT (Change level 7 to 3)
df.loc[:,'PRODUCT']=df.loc[:,'PRODUCT'].astype(float)
for i in range(50000):
    if df.loc[i,'PRODUCT']==7.0:
        df.loc[i,'PRODUCT']=float(3)
```

[213]:
```
#FLAG_ACSP_RECORD(all N)
df=df.drop(columns='FLAG_ACSP_RECORD')
```

[214]:
```
#Age
df.loc[:,'AGE']=df.loc[:,'AGE'].astype(float)
```

[215]:
```
#RESIDENCIAL_ZIP_3 (have something other than number and null, need modify one␣
↪by one)
df.loc[:,'RESIDENCIAL_ZIP_3']=df.loc[:,'RESIDENCIAL_ZIP_3'].astype(str)
```

```
for i in range(50000):
    if not df.loc[i,'RESIDENCIAL_ZIP_3'].isdigit():
        df.loc[i,'RESIDENCIAL_ZIP_3']=np.nan
```

[216]:
```
#PROFESSIONAL_ZIP_3
df.loc[:,'PROFESSIONAL_ZIP_3']=df.loc[:,'PROFESSIONAL_ZIP_3'].astype(str)
for i in range(50000):
    if not df.loc[i,'PROFESSIONAL_ZIP_3'].isdigit():
        df.loc[i,'PROFESSIONAL_ZIP_3']=np.nan
```

[217]:
```
#TARGET_LABEL_BAD=1
df.loc[:,'TARGET_LABEL_BAD=1']=df.loc[:,'TARGET_LABEL_BAD=1'].astype(float)
```

[218]:
```
df.describe(include='all')
```

[218]:

Var_Title	ID_CLIENT	PAYMENT_DAY	APPLICATION_SUBMISSION_TYPE
count	50000.000000	50000.000000	30539.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	25000.500000	12.869920	0.076394
std	14433.901067	6.608385	0.265632
min	1.000000	1.000000	0.000000
25%	12500.750000	10.000000	0.000000
50%	25000.500000	10.000000	0.000000
75%	37500.250000	15.000000	0.000000
max	50000.000000	25.000000	1.000000

Var_Title	POSTAL_ADDRESS_TYPE	SEX	MARITAL_STATUS	QUANT_DEPENDANTS
count	50000.000000	49935.000000	49798	50000.000000
unique	NaN	NaN	7	NaN
top	NaN	NaN	2	NaN
freq	NaN	NaN	25967	NaN
mean	1.006540	0.616902	NaN	0.650520
std	0.080606	0.486147	NaN	1.193655
min	1.000000	0.000000	NaN	0.000000
25%	1.000000	0.000000	NaN	0.000000
50%	1.000000	1.000000	NaN	0.000000
75%	1.000000	1.000000	NaN	1.000000
max	2.000000	1.000000	NaN	53.000000

Var_Title	STATE_OF_BIRTH	NACIONALITY	RESIDENCIAL_STATE	RESIDENCIAL_CITY
count	50000	50000.00000	50000	50000
unique	29	NaN	27	3529
top	BA	NaN	SP	Sao Paulo
freq	5717	NaN	8773	894
mean	NaN	0.95768	NaN	NaN

std	NaN	0.20132	NaN	NaN
min	NaN	0.00000	NaN	NaN
25%	NaN	1.00000	NaN	NaN
50%	NaN	1.00000	NaN	NaN
75%	NaN	1.00000	NaN	NaN
max	NaN	1.00000	NaN	NaN

Var_Title	RESIDENCIAL_BOROUGH	FLAG_RESIDENCIAL_PHONE	RESIDENCE_TYPE	\
count	50000	50000.000000	47891.000000	
unique	14511	NaN	NaN	
top	CENTRO	NaN	NaN	
freq	4169	NaN	NaN	
mean	NaN	0.836180	1.272097	
std	NaN	0.370116	0.860120	
min	NaN	0.000000	1.000000	
25%	NaN	1.000000	1.000000	
50%	NaN	1.000000	1.000000	
75%	NaN	1.000000	1.000000	
max	NaN	1.000000	5.000000	

Var_Title	MONTHS_IN_RESIDENCE	FLAG_EMAIL	PERSONAL_MONTHLY_INCOME	\
count	46223.000000	50000.000000	50000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	9.727149	0.802280	886.678437	
std	10.668841	0.398284	7846.959327	
min	0.000000	0.000000	60.000000	
25%	1.000000	1.000000	360.000000	
50%	6.000000	1.000000	500.000000	
75%	15.000000	1.000000	800.000000	
max	228.000000	1.000000	959000.000000	

Var_Title	OTHER_INCOMES	FLAG_VISA	FLAG_MASTERCARD	FLAG_DINERS	\
count	50000.000000	50000.000000	50000.000000	50000.000000	
unique	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	
mean	35.434760	0.111440	0.097460	0.001320	
std	891.515142	0.314679	0.296586	0.036308	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	
75%	0.000000	0.000000	0.000000	0.000000	
max	194344.000000	1.000000	1.000000	1.000000	

Var_Title	FLAG_AMERICAN_EXPRESS	FLAG_OTHER_CARDS	QUANT_BANKING_ACCOUNTS	\

count	50000.000000	50000.000000	50000.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	0.001740	0.002040	0.357840
std	0.041677	0.045121	0.479953
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.000000	1.000000
max	1.000000	1.000000	2.000000

Var_Title	QUANT_SPECIAL_BANKING_ACCOUNTS	PERSONAL_ASSETS_VALUE	\
count	50000.000000	5.000000e+04	
unique	NaN	NaN	
top	NaN	NaN	
freq	NaN	NaN	
mean	0.357840	2.322372e+03	
std	0.479953	4.235798e+04	
min	0.000000	0.000000e+00	
25%	0.000000	0.000000e+00	
50%	0.000000	0.000000e+00	
75%	1.000000	0.000000e+00	
max	2.000000	6.000000e+06	

Var_Title	QUANT_CARS	COMPANY	FLAG_PROFESSIONAL_PHONE	\
count	50000.000000	50000.00000	50000.00000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	0.336140	0.44082	0.26980	
std	0.472392	0.49649	0.44386	
min	0.000000	0.00000	0.00000	
25%	0.000000	0.00000	0.00000	
50%	0.000000	0.00000	0.00000	
75%	1.000000	1.00000	1.00000	
max	1.000000	1.00000	1.00000	

Var_Title	MONTHS_IN_THE_JOB	PROFESSION_CODE	OCCUPATION_TYPE	PRODUCT	\
count	50000.000000	42244.000000	42687.000000	50000.000000	
unique	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	
mean	0.009320	8.061784	2.484316	1.172100	
std	0.383453	3.220104	1.532261	0.440778	
min	0.000000	0.000000	0.000000	1.000000	
25%	0.000000	9.000000	1.000000	1.000000	

```
50%                 0.000000         9.000000         2.000000      1.000000
75%                 0.000000         9.000000         4.000000      1.000000
max                35.000000        18.000000         5.000000      3.000000

Var_Title           AGE RESIDENCIAL_ZIP_3 PROFESSIONAL_ZIP_3  \
count      50000.00000             49999               49999
unique             NaN               793                 793
top                NaN               960                 960
freq               NaN              1085                1085
mean           43.24852               NaN                 NaN
std            14.98905               NaN                 NaN
min             6.00000               NaN                 NaN
25%            31.00000               NaN                 NaN
50%            41.00000               NaN                 NaN
75%            53.00000               NaN                 NaN
max           106.00000               NaN                 NaN

Var_Title  TARGET_LABEL_BAD=1
count            50000.000000
unique                     NaN
top                        NaN
freq                       NaN
mean                  0.260820
std                   0.439086
min                   0.000000
25%                   0.000000
50%                   0.000000
75%                   1.000000
max                   1.000000
```

Now we have the dataframe left with clear numeric value. We need to delete more variables before split it into test and train dataset. From the describe table, we noticed that we might want to drop some variables based on the description. APPLICATION_SUBMISSION_TYPE with too many missing value (»1%),OCCUPATION_TYPE and PROFESSION_CODE with too many missing values. We drop PROFESSIONAL_ZIP_3 as well, because it has the same information as the residencial one.

[219]:
```
df=df.drop(columns='APPLICATION_SUBMISSION_TYPE')
df=df.drop(columns='OCCUPATION_TYPE')
df=df.drop(columns='PROFESSION_CODE')
df=df.drop(columns='PROFESSIONAL_ZIP_3')
```

[220]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import scorecardpy as sc
%matplotlib inline
```

[221]:
```
train, test = sc.split_df(df.iloc[:,1:],
                          y = 'TARGET_LABEL_BAD=1',
                          ratio = 0.7,
                          seed = 251238783).values()
```

0.0.2 TRAIN Missing Value

[222]:
```
train.isnull().any()
```

[222]:
```
Var_Title
PAYMENT_DAY                         False
POSTAL_ADDRESS_TYPE                 False
SEX                                  True
MARITAL_STATUS                       True
QUANT_DEPENDANTS                    False
STATE_OF_BIRTH                      False
NACIONALITY                         False
RESIDENCIAL_STATE                   False
RESIDENCIAL_CITY                    False
RESIDENCIAL_BOROUGH                 False
FLAG_RESIDENCIAL_PHONE              False
RESIDENCE_TYPE                       True
MONTHS_IN_RESIDENCE                  True
FLAG_EMAIL                          False
PERSONAL_MONTHLY_INCOME             False
OTHER_INCOMES                       False
FLAG_VISA                           False
FLAG_MASTERCARD                     False
FLAG_DINERS                         False
FLAG_AMERICAN_EXPRESS               False
FLAG_OTHER_CARDS                    False
QUANT_BANKING_ACCOUNTS              False
QUANT_SPECIAL_BANKING_ACCOUNTS      False
PERSONAL_ASSETS_VALUE               False
QUANT_CARS                          False
COMPANY                             False
FLAG_PROFESSIONAL_PHONE             False
MONTHS_IN_THE_JOB                   False
PRODUCT                             False
AGE                                 False
RESIDENCIAL_ZIP_3                    True
TARGET_LABEL_BAD=1                  False
dtype: bool
```

[223]:
```
train.describe(include='all')
```

[223]:

Var_Title	PAYMENT_DAY	POSTAL_ADDRESS_TYPE	SEX	MARITAL_STATUS
count	35000.000000	35000.000000	34955.000000	34869
unique	NaN	NaN	NaN	7
top	NaN	NaN	NaN	2
freq	NaN	NaN	NaN	18277
mean	12.861486	1.006800	0.619168	NaN
std	6.613217	0.082182	0.485598	NaN
min	1.000000	1.000000	0.000000	NaN
25%	10.000000	1.000000	0.000000	NaN
50%	10.000000	1.000000	1.000000	NaN
75%	15.000000	1.000000	1.000000	NaN
max	25.000000	2.000000	1.000000	NaN

Var_Title	QUANT_DEPENDANTS	STATE_OF_BIRTH	NACIONALITY	RESIDENCIAL_STATE
count	35000.000000	35000	35000.000000	35000
unique	NaN	29	NaN	27
top	NaN	BA	NaN	SP
freq	NaN	4022	NaN	6157
mean	0.657486	NaN	0.957486	NaN
std	1.176888	NaN	0.201762	NaN
min	0.000000	NaN	0.000000	NaN
25%	0.000000	NaN	1.000000	NaN
50%	0.000000	NaN	1.000000	NaN
75%	1.000000	NaN	1.000000	NaN
max	15.000000	NaN	1.000000	NaN

Var_Title	RESIDENCIAL_CITY	RESIDENCIAL_BOROUGH	FLAG_RESIDENCIAL_PHONE
count	35000	35000	35000.000000
unique	3023	11579	NaN
top	FORTALEZA	CENTRO	NaN
freq	612	2942	NaN
mean	NaN	NaN	0.835543
std	NaN	NaN	0.370695
min	NaN	NaN	0.000000
25%	NaN	NaN	1.000000
50%	NaN	NaN	1.000000
75%	NaN	NaN	1.000000
max	NaN	NaN	1.000000

Var_Title	RESIDENCE_TYPE	MONTHS_IN_RESIDENCE	FLAG_EMAIL
count	33535.000000	32343.000000	35000.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	1.274907	9.723217	0.803771
std	0.866131	10.699172	0.397149
min	1.000000	0.000000	0.000000

25%	1.000000	1.000000	1.000000
50%	1.000000	6.000000	1.000000
75%	1.000000	15.000000	1.000000
max	5.000000	228.000000	1.000000

Var_Title	PERSONAL_MONTHLY_INCOME	OTHER_INCOMES	FLAG_VISA	\
count	35000.000000	35000.000000	35000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	923.880094	31.585403	0.111400	
std	9108.778645	195.674286	0.314631	
min	69.000000	0.000000	0.000000	
25%	360.000000	0.000000	0.000000	
50%	500.000000	0.000000	0.000000	
75%	800.000000	0.000000	0.000000	
max	959000.000000	10200.000000	1.000000	

Var_Title	FLAG_MASTERCARD	FLAG_DINERS	FLAG_AMERICAN_EXPRESS	\
count	35000.000000	35000.000000	35000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	0.096657	0.001457	0.001571	
std	0.295495	0.038145	0.039611	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	
75%	0.000000	0.000000	0.000000	
max	1.000000	1.000000	1.000000	

Var_Title	FLAG_OTHER_CARDS	QUANT_BANKING_ACCOUNTS	\
count	35000.000000	35000.000000	
unique	NaN	NaN	
top	NaN	NaN	
freq	NaN	NaN	
mean	0.001971	0.357771	
std	0.044358	0.479887	
min	0.000000	0.000000	
25%	0.000000	0.000000	
50%	0.000000	0.000000	
75%	0.000000	1.000000	
max	1.000000	2.000000	

Var_Title	QUANT_SPECIAL_BANKING_ACCOUNTS	PERSONAL_ASSETS_VALUE	\
count	35000.000000	3.500000e+04	
unique	NaN	NaN	

top	NaN	NaN
freq	NaN	NaN
mean	0.357771	2.401439e+03
std	0.479887	4.885616e+04
min	0.000000	0.000000e+00
25%	0.000000	0.000000e+00
50%	0.000000	0.000000e+00
75%	1.000000	0.000000e+00
max	2.000000	6.000000e+06

Var_Title	QUANT_CARS	COMPANY	FLAG_PROFESSIONAL_PHONE	\
count	35000.000000	35000.000000	35000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	0.335543	0.440200	0.270343	
std	0.472187	0.496418	0.444143	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	
75%	1.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	

Var_Title	MONTHS_IN_THE_JOB	PRODUCT	AGE	RESIDENCIAL_ZIP_3	\
count	35000.0000	35000.000000	35000.000000	34999	
unique	NaN	NaN	NaN	763	
top	NaN	NaN	NaN	960	
freq	NaN	NaN	NaN	735	
mean	0.0094	1.173057	43.180229	NaN	
std	0.4015	0.441717	14.953427	NaN	
min	0.0000	1.000000	6.000000	NaN	
25%	0.0000	1.000000	31.000000	NaN	
50%	0.0000	1.000000	41.000000	NaN	
75%	0.0000	1.000000	53.000000	NaN	
max	35.0000	3.000000	101.000000	NaN	

Var_Title	TARGET_LABEL_BAD=1
count	35000.000000
unique	NaN
top	NaN
freq	NaN
mean	0.260829
std	0.439093
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000

```
max                    1.000000
```

We are starting by cleaning the train dataset

[224]:
```
#we starting by cleaning the SEX variable.
#Note, from the description, we have an mean for sex is above 0.6, hence the␣
 ↪mode of this data is 1
#We replace the missing value by 1
train.SEX.fillna(value = 1, inplace=True)
```

[225]:
```
train.loc[:,'MARITAL_STATUS'].mode()
```

[225]:
```
0    2
dtype: object
```

[226]:
```
#For MARITAL_STATUS we need to treat all 0 as NaN, since we dont have a␣
 ↪category of 0 for this one.
train.MARITAL_STATUS.fillna(value = '2', inplace=True)
```

[227]:
```
#For RESIDENCE_TYPE
#train.RESIDENCE_TYPE.fillna(train.RESIDENCE_TYPE.mode(), inplace=True)
train.RESIDENCE_TYPE.mode()
```

[227]:
```
0    1.0
dtype: float64
```

[228]:
```
train.loc[:,'RESIDENCE_TYPE']=train.loc[:,'RESIDENCE_TYPE'].astype(float)
train.RESIDENCE_TYPE.fillna(value=float(1), inplace=True)
```

[229]:
```
train.MONTHS_IN_RESIDENCE.median()
```

[229]:
```
6.0
```

[230]:
```
#Now for the MONTHS_IN_RESIDENCE, we replace the missing value by the median
train.MONTHS_IN_RESIDENCE.fillna(value=6, inplace=True)
```

[231]:
```
train.RESIDENCIAL_ZIP_3.mode()
```

[231]:
```
0    960
dtype: object
```

[232]:
```
#Now for the RESIDENCIAL_ZIP_3, we replace the missing value by the median
train.RESIDENCIAL_ZIP_3.fillna(value='960', inplace=True)
```

[233]:
```
train['STATE_OF_BIRTH']=train.loc[:,'STATE_OF_BIRTH'].str.upper()
c = train.groupby(['STATE_OF_BIRTH'])['TARGET_LABEL_BAD=1'].mean().sort_values()
c.count()
```

[233]: 29

[234]:
```
#Now we will be working on the string categories variables and we will create␣
 ↪levels for them based on bad rates
#Start with STATE_OF_BIRTH
train['STATE_OF_BIRTH']=train.loc[:,'STATE_OF_BIRTH'].str.upper()
c = train.groupby(['STATE_OF_BIRTH'])['TARGET_LABEL_BAD=1'].mean().sort_values()
for i in train.index:
    if train.loc[i,'STATE_OF_BIRTH'] in c.index[:9]:
        train.loc[i,'STATE_OF_BIRTH']='3'
    elif train.loc[i,'STATE_OF_BIRTH'] in c.index[9:19]:
        train.loc[i,'STATE_OF_BIRTH']='2'
    else:
        train.loc[i,'STATE_OF_BIRTH']='1'

train.loc[:,'STATE_OF_BIRTH']=train.loc[:,'STATE_OF_BIRTH'].astype(str)
```

[235]:
```
#RESIDENCIAL_STATE
train['RESIDENCIAL_STATE']=train.loc[:,'RESIDENCIAL_STATE'].str.upper()
m = train.groupby(['RESIDENCIAL_STATE'])['TARGET_LABEL_BAD=1'].mean().
 ↪sort_values()
for i in train.index:
    if train.loc[i,'RESIDENCIAL_STATE'] in m.index[:9]:
        train.loc[i,'RESIDENCIAL_STATE']='3'
    elif train.loc[i,'RESIDENCIAL_STATE'] in m.index[9:18]:
        train.loc[i,'RESIDENCIAL_STATE']='2'
    else:
        train.loc[i,'RESIDENCIAL_STATE']='1'
train.loc[:,'RESIDENCIAL_STATE']=train.loc[:,'RESIDENCIAL_STATE'].astype(str)
```

[236]:
```
# RESIDENCIAL_CITY like the state
train['RESIDENCIAL_CITY']=train.loc[:,'RESIDENCIAL_CITY'].str.upper()
v = train.groupby(['RESIDENCIAL_CITY'])['TARGET_LABEL_BAD=1'].mean().
 ↪sort_values()
for i in train.index:
    if train.loc[i,'RESIDENCIAL_CITY'] in v.index[:500]:
        train.loc[i,'RESIDENCIAL_CITY']='4'
    elif train.loc[i,'RESIDENCIAL_CITY'] in v.index[500:1000]:
        train.loc[i,'RESIDENCIAL_CITY']='3'
    elif train.loc[i,'RESIDENCIAL_CITY'] in v.index[1000:1500]:
        train.loc[i,'RESIDENCIAL_CITY']='2'
    else:
        train.loc[i,'RESIDENCIAL_CITY']='1'
train.loc[:,'RESIDENCIAL_CITY']=train.loc[:,'RESIDENCIAL_CITY'].astype(str)
```

[237]:
```
v.count()
```

[237]:
```
2131
```

[238]:
```
train['MARITAL_STATUS']=train.loc[:,'MARITAL_STATUS'].str.upper()
k = train.groupby(['MARITAL_STATUS'])['TARGET_LABEL_BAD=1'].mean().sort_values()
k
```

[238]:
```
MARITAL_STATUS
4    0.208833
3    0.218341
2    0.243101
5    0.252796
7    0.269122
6    0.288636
1    0.304528
Name: TARGET_LABEL_BAD=1, dtype: float64
```

[239]:
```
#MARITAL_STATUS
for i in train.index:
    if train.loc[i,'MARITAL_STATUS'] in k.index[:3]:
        train.loc[i,'MARITAL_STATUS']='3'
    elif train.loc[i,'MARITAL_STATUS'] in k.index[3:6]:
        train.loc[i,'MARITAL_STATUS']='2'
    else:
        train.loc[i,'MARITAL_STATUS']='1'
train.loc[:,'MARITAL_STATUS']=train.loc[:,'MARITAL_STATUS'].astype(str)
```

[240]:
```
train.loc[:,'RESIDENCE_TYPE']=train.loc[:,'RESIDENCE_TYPE'].astype(str)
p = train.groupby(['RESIDENCE_TYPE'])['TARGET_LABEL_BAD=1'].mean().sort_values()
p
```

[240]:
```
RESIDENCE_TYPE
3.0    0.230769
1.0    0.256094
4.0    0.276786
5.0    0.282979
2.0    0.302705
Name: TARGET_LABEL_BAD=1, dtype: float64
```

[241]:
```
t = train.groupby(['RESIDENCIAL_ZIP_3'])['TARGET_LABEL_BAD=1'].mean().
 ↪sort_values()
t.count()
```

[241]:
```
763
```

[242]:
```
t = train.groupby(['RESIDENCIAL_ZIP_3'])['TARGET_LABEL_BAD=1'].mean().
 ↪sort_values()
for i in train.index:
    if train.loc[i,'RESIDENCIAL_ZIP_3'] in t.index[:200]:
```

```
        train.loc[i,'RESIDENCIAL_ZIP_3']='4'
    elif train.loc[i,'RESIDENCIAL_ZIP_3'] in t.index[200:400]:
        train.loc[i,'RESIDENCIAL_ZIP_3']='3'
    elif train.loc[i,'RESIDENCIAL_ZIP_3'] in t.index[400:600]:
        train.loc[i,'RESIDENCIAL_ZIP_3']='2'
    else:
        train.loc[i,'RESIDENCIAL_ZIP_3']='1'
train.loc[:,'RESIDENCIAL_ZIP_3']=train.loc[:,'RESIDENCIAL_ZIP_3'].astype(str)
```

[243]:
```
o = train.groupby(['RESIDENCIAL_BOROUGH'])['TARGET_LABEL_BAD=1'].mean().
 ↪sort_values()
o.count()
```

[243]:
```
11579
```

[244]:
```
for i in train.index:
    if train.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[:2000]:
        train.loc[i,'RESIDENCIAL_BOROUGH']='6'
    elif train.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[2000:4000]:
        train.loc[i,'RESIDENCIAL_BOROUGH']='5'
    elif train.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[4000:6000]:
        train.loc[i,'RESIDENCIAL_BOROUGH']='4'
    elif train.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[6000:8000]:
        train.loc[i,'RESIDENCIAL_BOROUGH']='3'
    elif train.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[8000:10000]:
        train.loc[i,'RESIDENCIAL_BOROUGH']='2'
    else:
        train.loc[i,'RESIDENCIAL_BOROUGH']='1'
train.loc[:,'RESIDENCIAL_BOROUGH']=train.loc[:,'RESIDENCIAL_BOROUGH'].
 ↪astype(str)
```

[245]:
```
train.isnull().any()
```

[245]:
```
Var_Title
PAYMENT_DAY                    False
POSTAL_ADDRESS_TYPE            False
SEX                            False
MARITAL_STATUS                 False
QUANT_DEPENDANTS               False
STATE_OF_BIRTH                 False
NACIONALITY                    False
RESIDENCIAL_STATE              False
RESIDENCIAL_CITY               False
RESIDENCIAL_BOROUGH            False
FLAG_RESIDENCIAL_PHONE         False
RESIDENCE_TYPE                 False
MONTHS_IN_RESIDENCE            False
```

```
FLAG_EMAIL                          False
PERSONAL_MONTHLY_INCOME             False
OTHER_INCOMES                       False
FLAG_VISA                           False
FLAG_MASTERCARD                     False
FLAG_DINERS                         False
FLAG_AMERICAN_EXPRESS               False
FLAG_OTHER_CARDS                    False
QUANT_BANKING_ACCOUNTS              False
QUANT_SPECIAL_BANKING_ACCOUNTS      False
PERSONAL_ASSETS_VALUE               False
QUANT_CARS                          False
COMPANY                             False
FLAG_PROFESSIONAL_PHONE             False
MONTHS_IN_THE_JOB                   False
PRODUCT                             False
AGE                                 False
RESIDENCIAL_ZIP_3                   False
TARGET_LABEL_BAD=1                  False
dtype: bool
```

[246]:
```
train.describe(include='all')
```

[246]:

Var_Title	PAYMENT_DAY	POSTAL_ADDRESS_TYPE	SEX	MARITAL_STATUS
count	35000.000000	35000.000000	35000.000000	35000
unique	NaN	NaN	NaN	3
top	NaN	NaN	NaN	3
freq	NaN	NaN	NaN	21787
mean	12.861486	1.006800	0.619657	NaN
std	6.613217	0.082182	0.485478	NaN
min	1.000000	1.000000	0.000000	NaN
25%	10.000000	1.000000	0.000000	NaN
50%	10.000000	1.000000	1.000000	NaN
75%	15.000000	1.000000	1.000000	NaN
max	25.000000	2.000000	1.000000	NaN

Var_Title	QUANT_DEPENDANTS	STATE_OF_BIRTH	NACIONALITY	RESIDENCIAL_STATE
count	35000.000000	35000	35000.000000	35000
unique	NaN	3	NaN	3
top	NaN	2	NaN	2
freq	NaN	17958	NaN	22417
mean	0.657486	NaN	0.957486	NaN
std	1.176888	NaN	0.201762	NaN
min	0.000000	NaN	0.000000	NaN
25%	0.000000	NaN	1.000000	NaN
50%	0.000000	NaN	1.000000	NaN
75%	1.000000	NaN	1.000000	NaN

max	15.000000	NaN	1.000000	NaN

Var_Title	RESIDENCIAL_CITY	RESIDENCIAL_BOROUGH	FLAG_RESIDENCIAL_PHONE	\
count	35000	35000	35000.000000	
unique	4	6	NaN	
top	2	3	NaN	
freq	24388	17174	NaN	
mean	NaN	NaN	0.835543	
std	NaN	NaN	0.370695	
min	NaN	NaN	0.000000	
25%	NaN	NaN	1.000000	
50%	NaN	NaN	1.000000	
75%	NaN	NaN	1.000000	
max	NaN	NaN	1.000000	

Var_Title	RESIDENCE_TYPE	MONTHS_IN_RESIDENCE	FLAG_EMAIL	\
count	35000	35000.000000	35000.000000	
unique	5	NaN	NaN	
top	1.0	NaN	NaN	
freq	30563	NaN	NaN	
mean	NaN	9.440571	0.803771	
std	NaN	10.332204	0.397149	
min	NaN	0.000000	0.000000	
25%	NaN	2.000000	1.000000	
50%	NaN	6.000000	1.000000	
75%	NaN	14.000000	1.000000	
max	NaN	228.000000	1.000000	

Var_Title	PERSONAL_MONTHLY_INCOME	OTHER_INCOMES	FLAG_VISA	\
count	35000.000000	35000.000000	35000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	923.880094	31.585403	0.111400	
std	9108.778645	195.674286	0.314631	
min	69.000000	0.000000	0.000000	
25%	360.000000	0.000000	0.000000	
50%	500.000000	0.000000	0.000000	
75%	800.000000	0.000000	0.000000	
max	959000.000000	10200.000000	1.000000	

Var_Title	FLAG_MASTERCARD	FLAG_DINERS	FLAG_AMERICAN_EXPRESS	\
count	35000.000000	35000.000000	35000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	0.096657	0.001457	0.001571	

```
std              0.295495      0.038145              0.039611
min              0.000000      0.000000              0.000000
25%              0.000000      0.000000              0.000000
50%              0.000000      0.000000              0.000000
75%              0.000000      0.000000              0.000000
max              1.000000      1.000000              1.000000

Var_Title  FLAG_OTHER_CARDS  QUANT_BANKING_ACCOUNTS  \
count          35000.000000            35000.000000
unique                  NaN                     NaN
top                     NaN                     NaN
freq                    NaN                     NaN
mean               0.001971                0.357771
std                0.044358                0.479887
min                0.000000                0.000000
25%                0.000000                0.000000
50%                0.000000                0.000000
75%                0.000000                1.000000
max                1.000000                2.000000

Var_Title  QUANT_SPECIAL_BANKING_ACCOUNTS  PERSONAL_ASSETS_VALUE  \
count                        35000.000000           3.500000e+04
unique                                NaN                    NaN
top                                   NaN                    NaN
freq                                  NaN                    NaN
mean                             0.357771           2.401439e+03
std                              0.479887           4.885616e+04
min                              0.000000           0.000000e+00
25%                              0.000000           0.000000e+00
50%                              0.000000           0.000000e+00
75%                              1.000000           0.000000e+00
max                              2.000000           6.000000e+06

Var_Title    QUANT_CARS       COMPANY  FLAG_PROFESSIONAL_PHONE  \
count      35000.000000  35000.000000             35000.000000
unique              NaN           NaN                      NaN
top                 NaN           NaN                      NaN
freq                NaN           NaN                      NaN
mean           0.335543      0.440200                 0.270343
std            0.472187      0.496418                 0.444143
min            0.000000      0.000000                 0.000000
25%            0.000000      0.000000                 0.000000
50%            0.000000      0.000000                 0.000000
75%            1.000000      1.000000                 1.000000
max            1.000000      1.000000                 1.000000

Var_Title  MONTHS_IN_THE_JOB       PRODUCT           AGE RESIDENCIAL_ZIP_3  \
```

count	35000.0000	35000.000000	35000.000000	35000
unique	NaN	NaN	NaN	4
top	NaN	NaN	NaN	2
freq	NaN	NaN	NaN	14085
mean	0.0094	1.173057	43.180229	NaN
std	0.4015	0.441717	14.953427	NaN
min	0.0000	1.000000	6.000000	NaN
25%	0.0000	1.000000	31.000000	NaN
50%	0.0000	1.000000	41.000000	NaN
75%	0.0000	1.000000	53.000000	NaN
max	35.0000	3.000000	101.000000	NaN

Var_Title	TARGET_LABEL_BAD=1
count	35000.000000
unique	NaN
top	NaN
freq	NaN
mean	0.260829
std	0.439093
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

0.0.3 Test Missing Value

[247]:
```
test.isnull().any()
```

[247]:
```
Var_Title
PAYMENT_DAY                    False
POSTAL_ADDRESS_TYPE            False
SEX                             True
MARITAL_STATUS                  True
QUANT_DEPENDANTS               False
STATE_OF_BIRTH                 False
NACIONALITY                    False
RESIDENCIAL_STATE              False
RESIDENCIAL_CITY               False
RESIDENCIAL_BOROUGH            False
FLAG_RESIDENCIAL_PHONE         False
RESIDENCE_TYPE                  True
MONTHS_IN_RESIDENCE             True
FLAG_EMAIL                     False
PERSONAL_MONTHLY_INCOME        False
OTHER_INCOMES                  False
FLAG_VISA                      False
```

```
FLAG_MASTERCARD                   False
FLAG_DINERS                       False
FLAG_AMERICAN_EXPRESS             False
FLAG_OTHER_CARDS                  False
QUANT_BANKING_ACCOUNTS            False
QUANT_SPECIAL_BANKING_ACCOUNTS    False
PERSONAL_ASSETS_VALUE             False
QUANT_CARS                        False
COMPANY                           False
FLAG_PROFESSIONAL_PHONE           False
MONTHS_IN_THE_JOB                 False
PRODUCT                           False
AGE                               False
RESIDENCIAL_ZIP_3                 False
TARGET_LABEL_BAD=1                False
dtype: bool
```

[248]:
```
test.describe(include='all')
```

[248]:
```
Var_Title   PAYMENT_DAY  POSTAL_ADDRESS_TYPE           SEX MARITAL_STATUS  \
count      15000.000000         15000.000000  14980.000000          14929
unique              NaN                  NaN           NaN              7
top                 NaN                  NaN           NaN              2
freq                NaN                  NaN           NaN           7690
mean          12.889600             1.005933      0.611615            NaN
std            6.597276             0.076802      0.487399            NaN
min            1.000000             1.000000      0.000000            NaN
25%           10.000000             1.000000      0.000000            NaN
50%           10.000000             1.000000      1.000000            NaN
75%           15.000000             1.000000      1.000000            NaN
max           25.000000             2.000000      1.000000            NaN

Var_Title  QUANT_DEPENDANTS STATE_OF_BIRTH   NACIONALITY RESIDENCIAL_STATE  \
count          15000.000000          15000  15000.000000              15000
unique                  NaN             28           NaN                 27
top                     NaN             SP           NaN                 SP
freq                    NaN           1695           NaN               2616
mean               0.634267            NaN      0.958133                NaN
std                1.231777            NaN      0.200291                NaN
min                0.000000            NaN      0.000000                NaN
25%                0.000000            NaN      1.000000                NaN
50%                0.000000            NaN      1.000000                NaN
75%                1.000000            NaN      1.000000                NaN
max               53.000000            NaN      1.000000                NaN

Var_Title RESIDENCIAL_CITY RESIDENCIAL_BOROUGH  FLAG_RESIDENCIAL_PHONE  \
count                15000               15000            15000.000000
```

unique	2083	6559	NaN
top	Sao Paulo	CENTRO	NaN
freq	283	1227	NaN
mean	NaN	NaN	0.837667
std	NaN	NaN	0.368769
min	NaN	NaN	0.000000
25%	NaN	NaN	1.000000
50%	NaN	NaN	1.000000
75%	NaN	NaN	1.000000
max	NaN	NaN	1.000000

Var_Title	RESIDENCE_TYPE	MONTHS_IN_RESIDENCE	FLAG_EMAIL	\
count	14356.000000	13880.000000	15000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	1.265534	9.736311	0.798800	
std	0.845905	10.598207	0.400911	
min	1.000000	0.000000	0.000000	
25%	1.000000	1.000000	1.000000	
50%	1.000000	6.000000	1.000000	
75%	1.000000	15.000000	1.000000	
max	5.000000	99.000000	1.000000	

Var_Title	PERSONAL_MONTHLY_INCOME	OTHER_INCOMES	FLAG_VISA	\
count	15000.000000	15000.000000	15000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	799.874569	44.416593	0.111533	
std	3412.383497	1599.998798	0.314802	
min	60.000000	0.000000	0.000000	
25%	360.000000	0.000000	0.000000	
50%	500.000000	0.000000	0.000000	
75%	800.000000	0.000000	0.000000	
max	198183.000000	194344.000000	1.000000	

Var_Title	FLAG_MASTERCARD	FLAG_DINERS	FLAG_AMERICAN_EXPRESS	\
count	15000.000000	15000.000000	15000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	0.099333	0.001000	0.002133	
std	0.299119	0.031608	0.046140	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	

```
75%              0.000000      0.000000               0.000000
max              1.000000      1.000000               1.000000

Var_Title  FLAG_OTHER_CARDS  QUANT_BANKING_ACCOUNTS  \
count          15000.000000            15000.000000
unique                  NaN                     NaN
top                     NaN                     NaN
freq                    NaN                     NaN
mean               0.002200                0.358000
std                0.046854                0.480123
min                0.000000                0.000000
25%                0.000000                0.000000
50%                0.000000                0.000000
75%                0.000000                1.000000
max                1.000000                2.000000

Var_Title  QUANT_SPECIAL_BANKING_ACCOUNTS  PERSONAL_ASSETS_VALUE  \
count                        15000.000000           1.500000e+04
unique                                NaN                    NaN
top                                   NaN                    NaN
freq                                  NaN                    NaN
mean                             0.358000           2.137884e+03
std                              0.480123           2.027787e+04
min                              0.000000           0.000000e+00
25%                              0.000000           0.000000e+00
50%                              0.000000           0.000000e+00
75%                              1.000000           0.000000e+00
max                              2.000000           1.800000e+06

Var_Title    QUANT_CARS       COMPANY  FLAG_PROFESSIONAL_PHONE  \
count      15000.000000  15000.000000             15000.000000
unique              NaN           NaN                      NaN
top                 NaN           NaN                      NaN
freq                NaN           NaN                      NaN
mean           0.337533      0.442267                 0.268533
std            0.472884      0.496672                 0.443211
min            0.000000      0.000000                 0.000000
25%            0.000000      0.000000                 0.000000
50%            0.000000      0.000000                 0.000000
75%            1.000000      1.000000                 1.000000
max            1.000000      1.000000                 1.000000

Var_Title  MONTHS_IN_THE_JOB       PRODUCT           AGE RESIDENCIAL_ZIP_3  \
count           15000.000000  15000.000000  15000.000000             15000
unique                   NaN           NaN           NaN               704
top                      NaN           NaN           NaN               960
freq                     NaN           NaN           NaN               350
```

```
mean                 0.009133      1.169867     43.407867                NaN
std                  0.337625      0.438587     15.071138                NaN
min                  0.000000      1.000000     17.000000                NaN
25%                  0.000000      1.000000     32.000000                NaN
50%                  0.000000      1.000000     42.000000                NaN
75%                  0.000000      1.000000     54.000000                NaN
max                 26.000000      3.000000    106.000000                NaN

Var_Title  TARGET_LABEL_BAD=1
count            15000.000000
unique                     NaN
top                        NaN
freq                       NaN
mean                  0.260800
std                   0.439086
min                   0.000000
25%                   0.000000
50%                   0.000000
75%                   1.000000
max                   1.000000
```

[249]:
```
#we starting by cleaning the SEX variable.
#Note, from the description, we have an mean for sex is above 0.6, hence the␣
 ↪mode of this data is 1
#We replace the missing value by 1
test.SEX.fillna(value = 1, inplace=True)
```

[250]:
```
test.loc[:,'MARITAL_STATUS'].mode()
```

[250]:
```
0    2
dtype: object
```

[251]:
```
#For MARITAL_STATUS we need to treat all 0 as NaN, since we dont have a␣
 ↪category of 0 for this one.
test.MARITAL_STATUS.fillna(value = '2', inplace=True)
```

[252]:
```
test.MONTHS_IN_RESIDENCE.median()
```

[252]:
```
6.0
```

[253]:
```
#Now for the MONTHS_IN_RESIDENCE, we replace the missing value by the median
test.MONTHS_IN_RESIDENCE.fillna(value=6, inplace=True)
```

[254]:
```
test.RESIDENCIAL_ZIP_3.mode()
```

[254]:
```
0    960
dtype: object
```

[255]:
```
test.RESIDENCIAL_ZIP_3.fillna(value='960', inplace=True)
```

[256]:
```
m.count()
```

[256]:
```
27
```

[257]:
```
#Now we will be working on the string categories variables and we will create␣
 ↪levels for them based on bad rates
#Start with STATE_OF_BIRTH
#We need to re-categorize them using the level we have from Train set,␣
 ↪otherwise the level will be inconsistent
#For those categories that did not exist in Train set, we mark them as level 0␣
 ↪since we do not train our model with them
for i in test.index:
    if test.loc[i,'STATE_OF_BIRTH'] in c.index[:9]:
        test.loc[i,'STATE_OF_BIRTH']='3'
    elif test.loc[i,'STATE_OF_BIRTH'] in c.index[9:19]:
        test.loc[i,'STATE_OF_BIRTH']='2'
    elif test.loc[i,'STATE_OF_BIRTH'] in c.index[19:]:
        test.loc[i,'STATE_OF_BIRTH']='1'
    else:
        test.loc[i,'STATE_OF_BIRTH']=np.nan
```

[258]:
```
#RESIDENCIAL_STATE
for i in test.index:
    if test.loc[i,'RESIDENCIAL_STATE'] in m.index[:9]:
        test.loc[i,'RESIDENCIAL_STATE']='3'
    elif test.loc[i,'RESIDENCIAL_STATE'] in m.index[9:18]:
        test.loc[i,'RESIDENCIAL_STATE']='2'
    elif test.loc[i,'RESIDENCIAL_STATE'] in m.index[18:]:
        test.loc[i,'RESIDENCIAL_STATE']='1'
    else:
        test.loc[i,'RESIDENCIAL_STATE']=np.nan
```

[259]:
```
# RESIDENCIAL_CITY like the state
for i in test.index:
    if test.loc[i,'RESIDENCIAL_CITY'] in v.index[:500]:
        test.loc[i,'RESIDENCIAL_CITY']='4'
    elif test.loc[i,'RESIDENCIAL_CITY'] in v.index[500:1000]:
        test.loc[i,'RESIDENCIAL_CITY']='3'
    elif test.loc[i,'RESIDENCIAL_CITY'] in v.index[1000:1500]:
        test.loc[i,'RESIDENCIAL_CITY']='2'
    elif test.loc[i,'RESIDENCIAL_CITY'] in v.index[1500:]:
        test.loc[i,'RESIDENCIAL_CITY']='1'
    else:
        test.loc[i,'RESIDENCIAL_CITY']=np.nan
```

[260]:
```
#marital status
for i in test.index:
    if test.loc[i,'MARITAL_STATUS'] in k.index[:3]:
        test.loc[i,'MARITAL_STATUS']='3'
    elif test.loc[i,'MARITAL_STATUS'] in k.index[3:6]:
        test.loc[i,'MARITAL_STATUS']='2'
    elif test.loc[i,'MARITAL_STATUS'] in k.index[6:]:
        test.loc[i,'MARITAL_STATUS']='1'
    else:
        test.loc[i,'MARITAL_STATUS']=np.nan
```

[261]:
```
#residence type
for i in test.index:
    if test.loc[i,'RESIDENCIAL_ZIP_3'] in t.index[:200]:
        test.loc[i,'RESIDENCIAL_ZIP_3']='4'
    elif test.loc[i,'RESIDENCIAL_ZIP_3'] in t.index[200:400]:
        test.loc[i,'RESIDENCIAL_ZIP_3']='3'
    elif test.loc[i,'RESIDENCIAL_ZIP_3'] in t.index[400:600]:
        test.loc[i,'RESIDENCIAL_ZIP_3']='2'
    elif test.loc[i,'RESIDENCIAL_ZIP_3'] in t.index[600:]:
        test.loc[i,'RESIDENCIAL_ZIP_3']='1'
    else:
        test.loc[i,'RESIDENCIAL_ZIP_3']=np.nan
```

[262]:
```
for i in test.index:
    if test.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[:2000]:
        test.loc[i,'RESIDENCIAL_BOROUGH']='6'
    elif test.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[2000:4000]:
        test.loc[i,'RESIDENCIAL_BOROUGH']='5'
    elif test.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[4000:6000]:
        test.loc[i,'RESIDENCIAL_BOROUGH']='4'
    elif test.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[6000:8000]:
        test.loc[i,'RESIDENCIAL_BOROUGH']='3'
    elif test.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[8000:10000]:
        test.loc[i,'RESIDENCIAL_BOROUGH']='2'
    elif test.loc[i,'RESIDENCIAL_BOROUGH'] in o.index[10000:]:
        test.loc[i,'RESIDENCIAL_BOROUGH']='1'
    else:
        test.loc[i,'RESIDENCIAL_BOROUGH']=np.nan
```

[263]:
```
test.loc[:,'RESIDENCE_TYPE'].mode()
```

[263]:
```
0    1.0
dtype: float64
```

[264]:
```
#For RESIDENCE_TYPE (Filling the missing value).
test.RESIDENCE_TYPE.fillna(value = float(1), inplace=True)
```

[265]:
```
test.loc[:,'RESIDENCE_TYPE']=test.loc[:,'RESIDENCE_TYPE'].astype(str)
```

[266]:
```
test.RESIDENCIAL_CITY.mode()
```

[266]:
```
0    2
dtype: object
```

[267]:
```
test.RESIDENCIAL_BOROUGH.mode()
```

[267]:
```
0    3
dtype: object
```

[268]:
```
test.RESIDENCIAL_ZIP_3.mode()
```

[268]:
```
0    2
dtype: object
```

[269]:
```
test.RESIDENCIAL_CITY.fillna(value = '2', inplace=True)
```

[270]:
```
test.RESIDENCIAL_BOROUGH.fillna(value = '3', inplace=True)
```

[271]:
```
test.loc[:,'RESIDENCIAL_BOROUGH']=test.loc[:,'RESIDENCIAL_BOROUGH'].astype(str)
```

[272]:
```
test.RESIDENCIAL_BOROUGH.unique()
```

[272]:
```
array(['3', '6', '2', '5', '4', '1'], dtype=object)
```

[273]:
```
test.RESIDENCIAL_ZIP_3.fillna(value = '2', inplace=True)
```

[274]:
```
test.isnull().any()
```

[274]:
```
Var_Title
PAYMENT_DAY                    False
POSTAL_ADDRESS_TYPE            False
SEX                            False
MARITAL_STATUS                 False
QUANT_DEPENDANTS               False
STATE_OF_BIRTH                 False
NACIONALITY                    False
RESIDENCIAL_STATE              False
RESIDENCIAL_CITY               False
RESIDENCIAL_BOROUGH            False
FLAG_RESIDENCIAL_PHONE         False
RESIDENCE_TYPE                 False
MONTHS_IN_RESIDENCE            False
FLAG_EMAIL                     False
PERSONAL_MONTHLY_INCOME        False
OTHER_INCOMES                  False
```

```
FLAG_VISA                          False
FLAG_MASTERCARD                    False
FLAG_DINERS                        False
FLAG_AMERICAN_EXPRESS              False
FLAG_OTHER_CARDS                   False
QUANT_BANKING_ACCOUNTS             False
QUANT_SPECIAL_BANKING_ACCOUNTS     False
PERSONAL_ASSETS_VALUE              False
QUANT_CARS                         False
COMPANY                            False
FLAG_PROFESSIONAL_PHONE            False
MONTHS_IN_THE_JOB                  False
PRODUCT                            False
AGE                                False
RESIDENCIAL_ZIP_3                  False
TARGET_LABEL_BAD=1                 False
dtype: bool
```

[275]:
```
test.describe(include='all')
```

[275]:
```
Var_Title    PAYMENT_DAY  POSTAL_ADDRESS_TYPE           SEX MARITAL_STATUS  \
count      15000.000000         15000.000000  15000.000000          15000
unique              NaN                  NaN           NaN              3
top                 NaN                  NaN           NaN              3
freq                NaN                  NaN           NaN           9220
mean          12.889600             1.005933      0.612133            NaN
std            6.597276             0.076802      0.487280            NaN
min            1.000000             1.000000      0.000000            NaN
25%           10.000000             1.000000      0.000000            NaN
50%           10.000000             1.000000      1.000000            NaN
75%           15.000000             1.000000      1.000000            NaN
max           25.000000             2.000000      1.000000            NaN

Var_Title  QUANT_DEPENDANTS STATE_OF_BIRTH   NACIONALITY RESIDENCIAL_STATE  \
count          15000.000000          15000  15000.000000             15000
unique                  NaN              3           NaN                 3
top                     NaN              2           NaN                 2
freq                    NaN           7713           NaN              9566
mean               0.634267            NaN      0.958133               NaN
std                1.231777            NaN      0.200291               NaN
min                0.000000            NaN      0.000000               NaN
25%                0.000000            NaN      1.000000               NaN
50%                0.000000            NaN      1.000000               NaN
75%                1.000000            NaN      1.000000               NaN
max               53.000000            NaN      1.000000               NaN

Var_Title RESIDENCIAL_CITY RESIDENCIAL_BOROUGH  FLAG_RESIDENCIAL_PHONE  \
```

count	15000	15000	15000.000000
unique	4	6	NaN
top	2	3	NaN
freq	11725	9801	NaN
mean	NaN	NaN	0.837667
std	NaN	NaN	0.368769
min	NaN	NaN	0.000000
25%	NaN	NaN	1.000000
50%	NaN	NaN	1.000000
75%	NaN	NaN	1.000000
max	NaN	NaN	1.000000

Var_Title	RESIDENCE_TYPE	MONTHS_IN_RESIDENCE	FLAG_EMAIL	\
count	15000	15000.000000	15000.000000	
unique	5	NaN	NaN	
top	1.0	NaN	NaN	
freq	13118	NaN	NaN	
mean	NaN	9.457333	0.798800	
std	NaN	10.242036	0.400911	
min	NaN	0.000000	0.000000	
25%	NaN	2.000000	1.000000	
50%	NaN	6.000000	1.000000	
75%	NaN	14.000000	1.000000	
max	NaN	99.000000	1.000000	

Var_Title	PERSONAL_MONTHLY_INCOME	OTHER_INCOMES	FLAG_VISA	\
count	15000.000000	15000.000000	15000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	799.874569	44.416593	0.111533	
std	3412.383497	1599.998798	0.314802	
min	60.000000	0.000000	0.000000	
25%	360.000000	0.000000	0.000000	
50%	500.000000	0.000000	0.000000	
75%	800.000000	0.000000	0.000000	
max	198183.000000	194344.000000	1.000000	

Var_Title	FLAG_MASTERCARD	FLAG_DINERS	FLAG_AMERICAN_EXPRESS	\
count	15000.000000	15000.000000	15000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	0.099333	0.001000	0.002133	
std	0.299119	0.031608	0.046140	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	

```
50%              0.000000      0.000000              0.000000
75%              0.000000      0.000000              0.000000
max              1.000000      1.000000              1.000000

Var_Title  FLAG_OTHER_CARDS  QUANT_BANKING_ACCOUNTS  \
count          15000.000000            15000.000000
unique                  NaN                     NaN
top                     NaN                     NaN
freq                    NaN                     NaN
mean               0.002200                0.358000
std                0.046854                0.480123
min                0.000000                0.000000
25%                0.000000                0.000000
50%                0.000000                0.000000
75%                0.000000                1.000000
max                1.000000                2.000000

Var_Title  QUANT_SPECIAL_BANKING_ACCOUNTS  PERSONAL_ASSETS_VALUE  \
count                        15000.000000           1.500000e+04
unique                                NaN                    NaN
top                                   NaN                    NaN
freq                                  NaN                    NaN
mean                             0.358000           2.137884e+03
std                              0.480123           2.027787e+04
min                              0.000000           0.000000e+00
25%                              0.000000           0.000000e+00
50%                              0.000000           0.000000e+00
75%                              1.000000           0.000000e+00
max                              2.000000           1.800000e+06

Var_Title    QUANT_CARS       COMPANY  FLAG_PROFESSIONAL_PHONE  \
count      15000.000000  15000.000000             15000.000000
unique              NaN           NaN                      NaN
top                 NaN           NaN                      NaN
freq                NaN           NaN                      NaN
mean           0.337533      0.442267                 0.268533
std            0.472884      0.496672                 0.443211
min            0.000000      0.000000                 0.000000
25%            0.000000      0.000000                 0.000000
50%            0.000000      0.000000                 0.000000
75%            1.000000      1.000000                 1.000000
max            1.000000      1.000000                 1.000000

Var_Title  MONTHS_IN_THE_JOB       PRODUCT           AGE RESIDENCIAL_ZIP_3  \
count           15000.000000  15000.000000  15000.000000             15000
unique                   NaN           NaN           NaN                 4
top                      NaN           NaN           NaN                 2
```

freq	NaN	NaN	NaN	6069
mean	0.009133	1.169867	43.407867	NaN
std	0.337625	0.438587	15.071138	NaN
min	0.000000	1.000000	17.000000	NaN
25%	0.000000	1.000000	32.000000	NaN
50%	0.000000	1.000000	42.000000	NaN
75%	0.000000	1.000000	54.000000	NaN
max	26.000000	3.000000	106.000000	NaN

Var_Title	TARGET_LABEL_BAD=1
count	15000.000000
unique	NaN
top	NaN
freq	NaN
mean	0.260800
std	0.439086
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

0.0.4 Outliers

By browsing the train set description, we noticed that there are a few variables that has extreme outliers that may affect our modelling, we picked them out. The variables that we need to specifically look at are QUANT_DEPENDANTS, MONTHS_IN_RESIDENCE, PERSONAL_MONTHLY_INCOME, OTHER_INCOMES, PERSONAL_ASSETS_VALUE, MONTHS_IN_THE_JOB, AGE

[276]:
```
train.describe(include='all')
```

[276]:

Var_Title	PAYMENT_DAY	POSTAL_ADDRESS_TYPE	SEX	MARITAL_STATUS	\
count	35000.000000	35000.000000	35000.000000	35000	
unique	NaN	NaN	NaN	3	
top	NaN	NaN	NaN	3	
freq	NaN	NaN	NaN	21787	
mean	12.861486	1.006800	0.619657	NaN	
std	6.613217	0.082182	0.485478	NaN	
min	1.000000	1.000000	0.000000	NaN	
25%	10.000000	1.000000	0.000000	NaN	
50%	10.000000	1.000000	1.000000	NaN	
75%	15.000000	1.000000	1.000000	NaN	
max	25.000000	2.000000	1.000000	NaN	

Var_Title	QUANT_DEPENDANTS	STATE_OF_BIRTH	NACIONALITY	RESIDENCIAL_STATE	\
count	35000.000000	35000	35000.000000	35000	
unique	NaN	3	NaN	3	

top	NaN	2	NaN	2
freq	NaN	17958	NaN	22417
mean	0.657486	NaN	0.957486	NaN
std	1.176888	NaN	0.201762	NaN
min	0.000000	NaN	0.000000	NaN
25%	0.000000	NaN	1.000000	NaN
50%	0.000000	NaN	1.000000	NaN
75%	1.000000	NaN	1.000000	NaN
max	15.000000	NaN	1.000000	NaN

Var_Title	RESIDENCIAL_CITY	RESIDENCIAL_BOROUGH	FLAG_RESIDENCIAL_PHONE	\
count	35000	35000	35000.000000	
unique	4	6	NaN	
top	2	3	NaN	
freq	24388	17174	NaN	
mean	NaN	NaN	0.835543	
std	NaN	NaN	0.370695	
min	NaN	NaN	0.000000	
25%	NaN	NaN	1.000000	
50%	NaN	NaN	1.000000	
75%	NaN	NaN	1.000000	
max	NaN	NaN	1.000000	

Var_Title	RESIDENCE_TYPE	MONTHS_IN_RESIDENCE	FLAG_EMAIL	\
count	35000	35000.000000	35000.000000	
unique	5	NaN	NaN	
top	1.0	NaN	NaN	
freq	30563	NaN	NaN	
mean	NaN	9.440571	0.803771	
std	NaN	10.332204	0.397149	
min	NaN	0.000000	0.000000	
25%	NaN	2.000000	1.000000	
50%	NaN	6.000000	1.000000	
75%	NaN	14.000000	1.000000	
max	NaN	228.000000	1.000000	

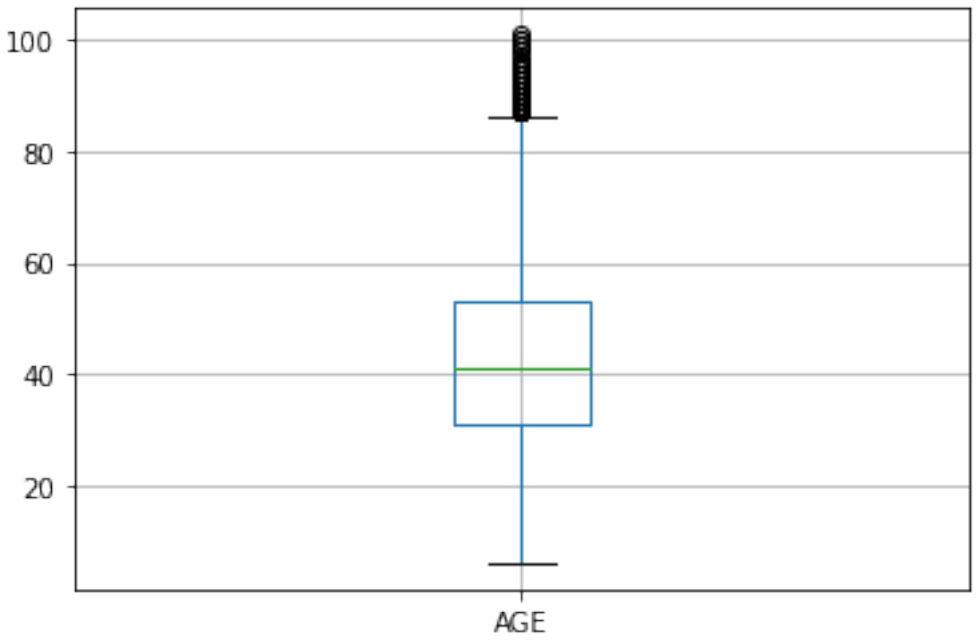
Var_Title	PERSONAL_MONTHLY_INCOME	OTHER_INCOMES	FLAG_VISA	\
count	35000.000000	35000.000000	35000.000000	
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	923.880094	31.585403	0.111400	
std	9108.778645	195.674286	0.314631	
min	69.000000	0.000000	0.000000	
25%	360.000000	0.000000	0.000000	
50%	500.000000	0.000000	0.000000	
75%	800.000000	0.000000	0.000000	

```
max                 959000.000000   10200.000000      1.000000

Var_Title  FLAG_MASTERCARD   FLAG_DINERS  FLAG_AMERICAN_EXPRESS  \
count         35000.000000  35000.000000           35000.000000
unique                 NaN           NaN                    NaN
top                    NaN           NaN                    NaN
freq                   NaN           NaN                    NaN
mean              0.096657      0.001457               0.001571
std               0.295495      0.038145               0.039611
min               0.000000      0.000000               0.000000
25%               0.000000      0.000000               0.000000
50%               0.000000      0.000000               0.000000
75%               0.000000      0.000000               0.000000
max               1.000000      1.000000               1.000000

Var_Title  FLAG_OTHER_CARDS  QUANT_BANKING_ACCOUNTS  \
count          35000.000000            35000.000000
unique                  NaN                     NaN
top                     NaN                     NaN
freq                    NaN                     NaN
mean               0.001971                0.357771
std                0.044358                0.479887
min                0.000000                0.000000
25%                0.000000                0.000000
50%                0.000000                0.000000
75%                0.000000                1.000000
max                1.000000                2.000000

Var_Title  QUANT_SPECIAL_BANKING_ACCOUNTS  PERSONAL_ASSETS_VALUE  \
count                        35000.000000           3.500000e+04
unique                                NaN                    NaN
top                                   NaN                    NaN
freq                                  NaN                    NaN
mean                             0.357771           2.401439e+03
std                              0.479887           4.885616e+04
min                              0.000000           0.000000e+00
25%                              0.000000           0.000000e+00
50%                              0.000000           0.000000e+00
75%                              1.000000           0.000000e+00
max                              2.000000           6.000000e+06

Var_Title    QUANT_CARS       COMPANY  FLAG_PROFESSIONAL_PHONE  \
count      35000.000000  35000.000000             35000.000000
unique              NaN           NaN                      NaN
top                 NaN           NaN                      NaN
freq                NaN           NaN                      NaN
mean           0.335543      0.440200                 0.270343
```

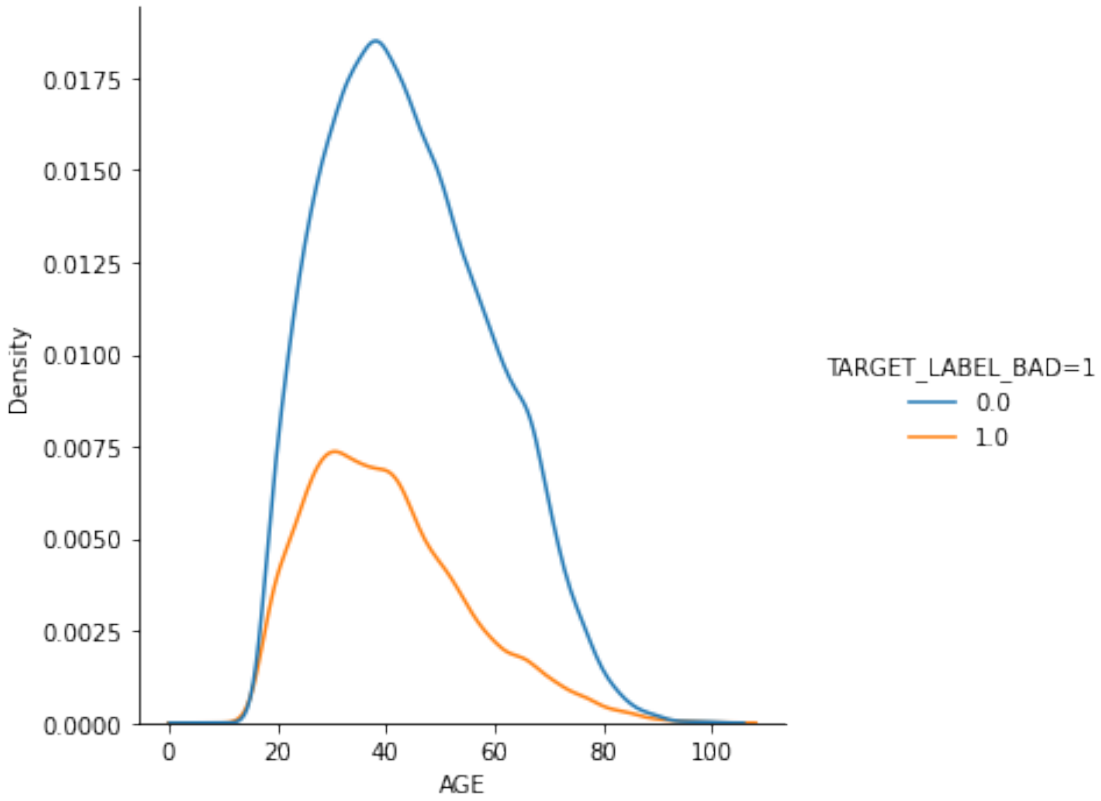
```
std            0.472187      0.496418                0.444143
min            0.000000      0.000000                0.000000
25%            0.000000      0.000000                0.000000
50%            0.000000      0.000000                0.000000
75%            1.000000      1.000000                1.000000
max            1.000000      1.000000                1.000000

Var_Title  MONTHS_IN_THE_JOB       PRODUCT           AGE RESIDENCIAL_ZIP_3  \
count            35000.0000  35000.000000  35000.000000             35000
unique                  NaN           NaN           NaN                 4
top                     NaN           NaN           NaN                 2
freq                    NaN           NaN           NaN             14085
mean                 0.0094      1.173057     43.180229               NaN
std                  0.4015      0.441717     14.953427               NaN
min                  0.0000      1.000000      6.000000               NaN
25%                  0.0000      1.000000     31.000000               NaN
50%                  0.0000      1.000000     41.000000               NaN
75%                  0.0000      1.000000     53.000000               NaN
max                 35.0000      3.000000    101.000000               NaN

Var_Title  TARGET_LABEL_BAD=1
count            35000.000000
unique                    NaN
top                       NaN
freq                      NaN
mean                 0.260829
std                  0.439093
min                  0.000000
25%                  0.000000
50%                  0.000000
75%                  1.000000
max                  1.000000
```

[277]:
```
train.boxplot(column='AGE')
```

[277]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f406a794150>
```

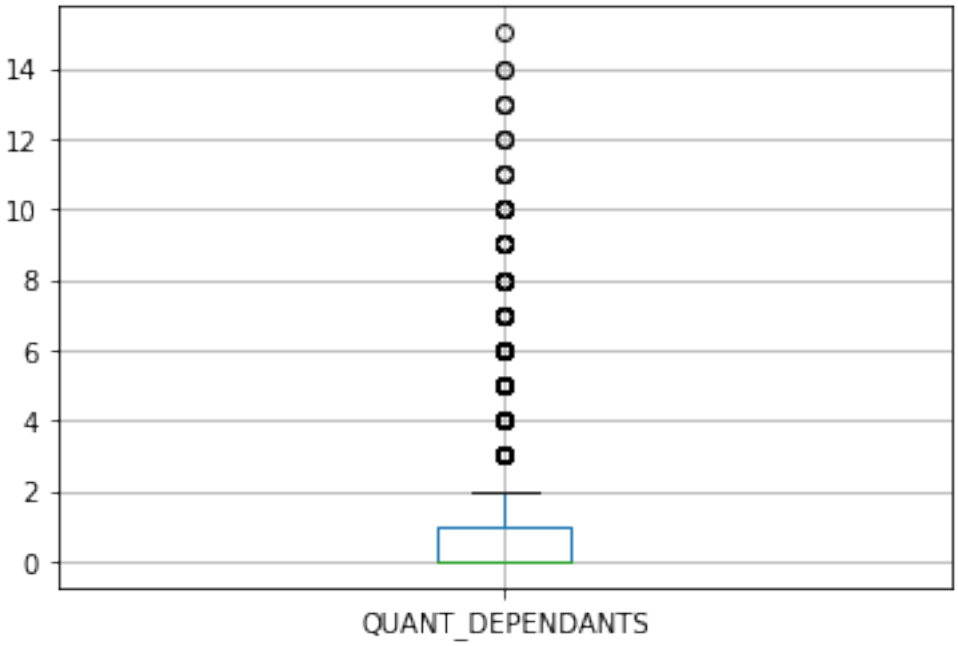


[278]:
```
sns.displot(data = train, x = 'AGE', kind = 'kde', hue = 'TARGET_LABEL_BAD=1')
plt.show()
```

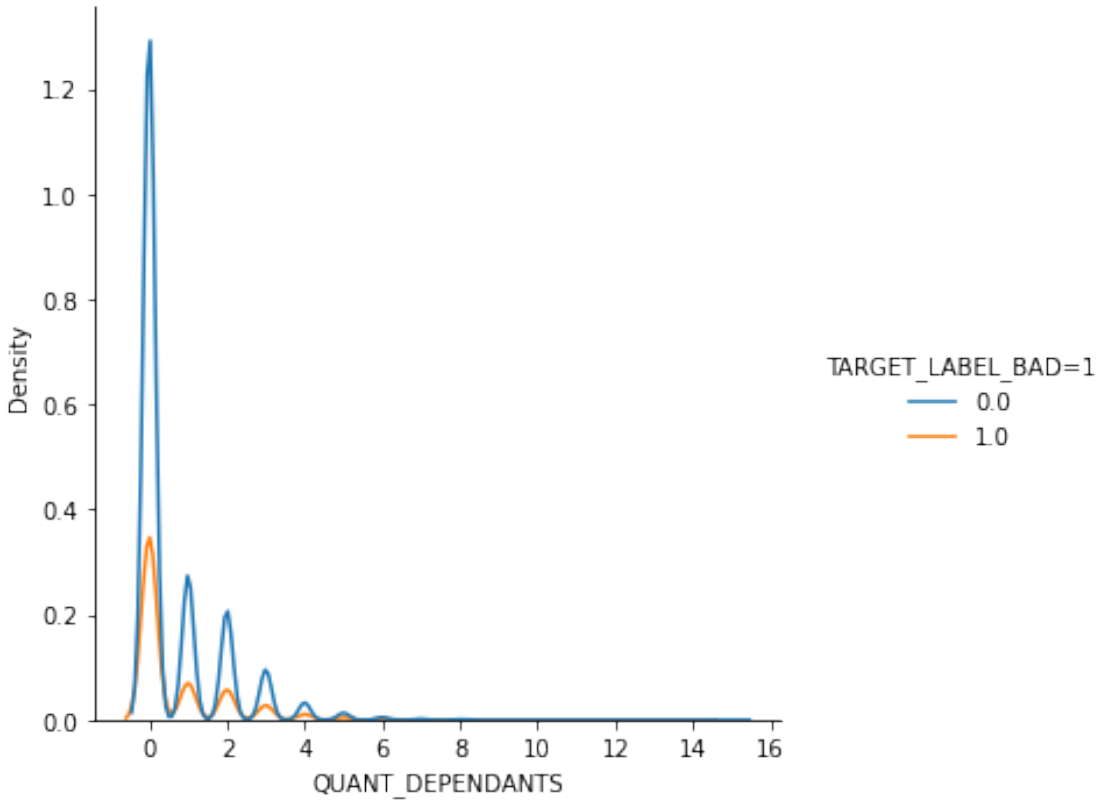


[279]:
```
train.boxplot(column='QUANT_DEPENDANTS')
```

[279]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f406a5f6650>
```

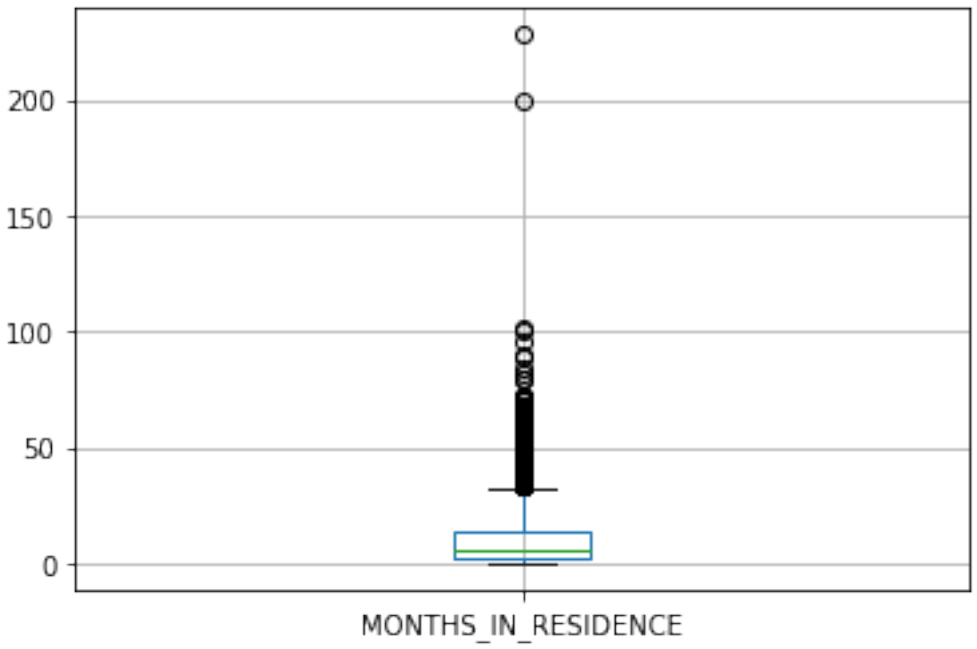


[280]:
```
sns.displot(data = train, x = 'QUANT_DEPENDANTS', kind = 'kde', hue =␣
↪'TARGET_LABEL_BAD=1')
plt.show()
```

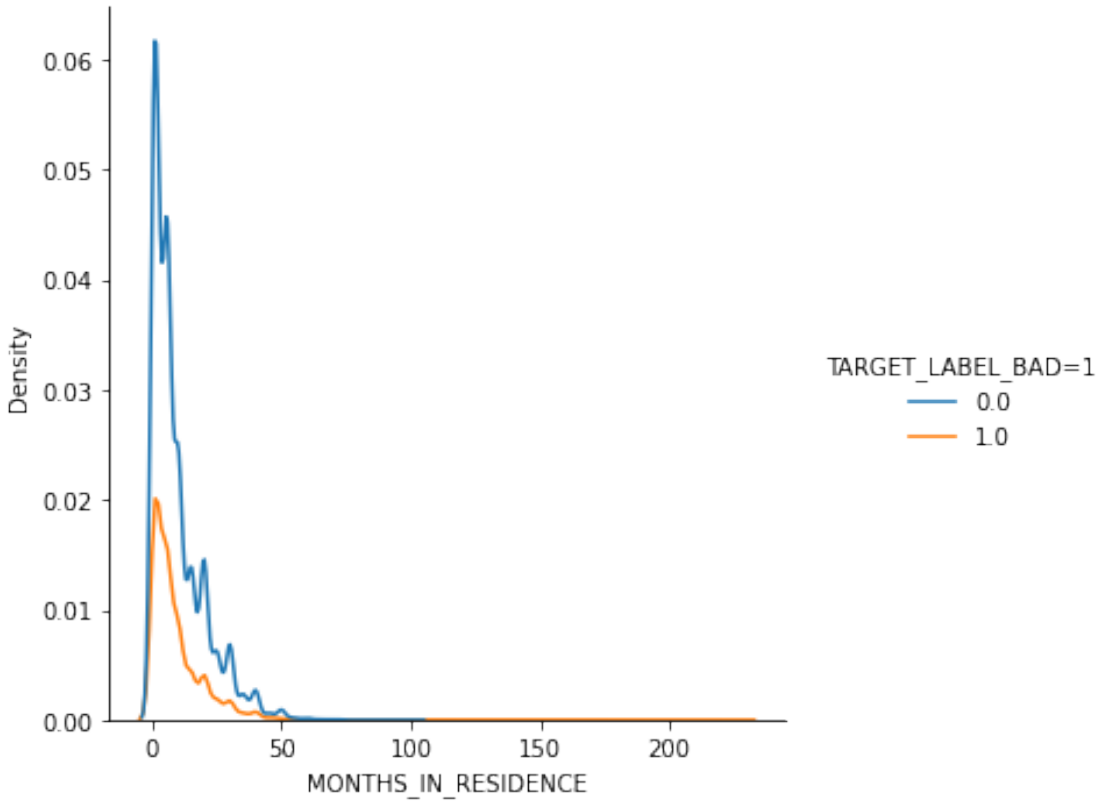


[281]:
```
train.boxplot(column='MONTHS_IN_RESIDENCE')
```

[281]: <matplotlib.axes._subplots.AxesSubplot at 0x7f406a544110>

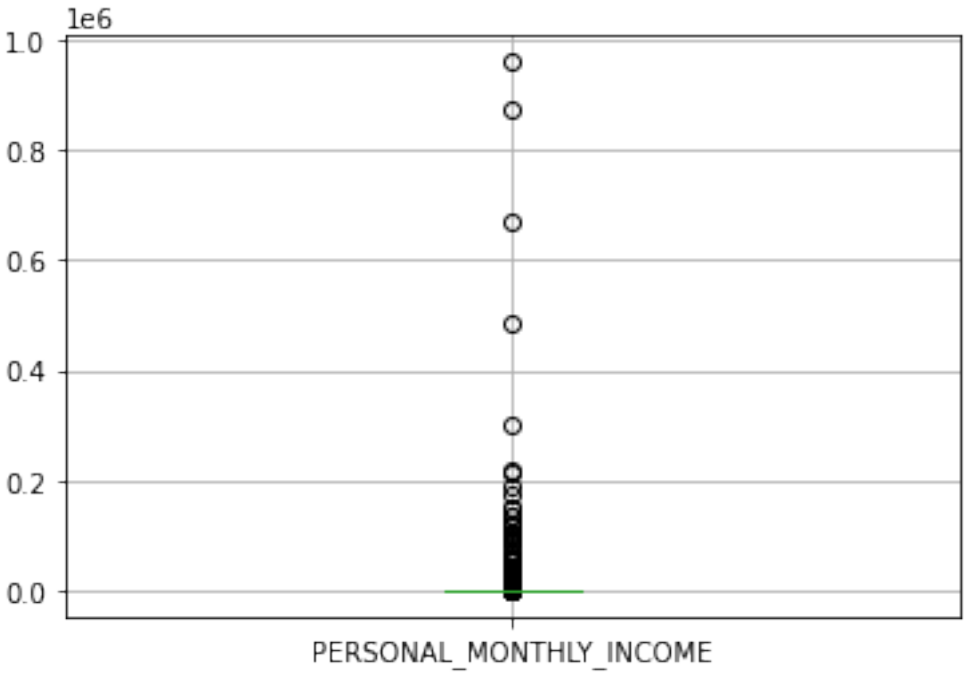


[282]:
```
sns.displot(data = train, x = 'MONTHS_IN_RESIDENCE', kind = 'kde', hue =␣
↪'TARGET_LABEL_BAD=1')
plt.show()
```

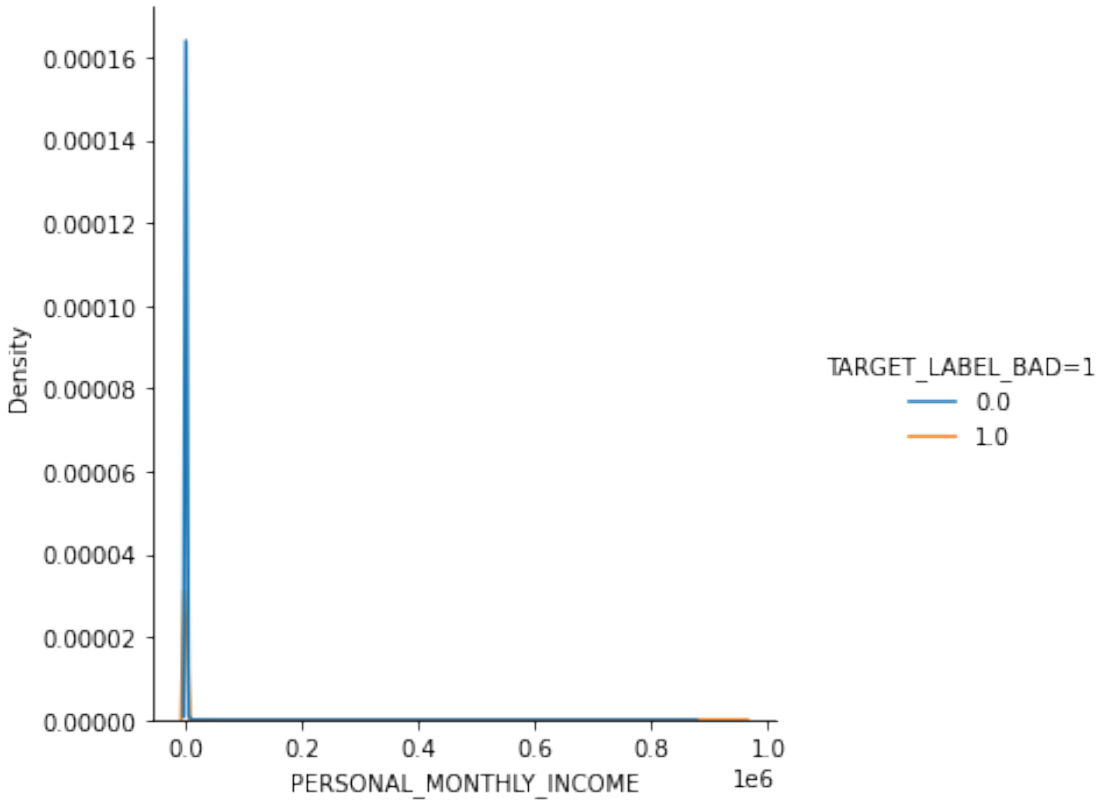


[283]:
```
train.boxplot(column='PERSONAL_MONTHLY_INCOME')
```

[283]: <matplotlib.axes._subplots.AxesSubplot at 0x7f406a66a910>

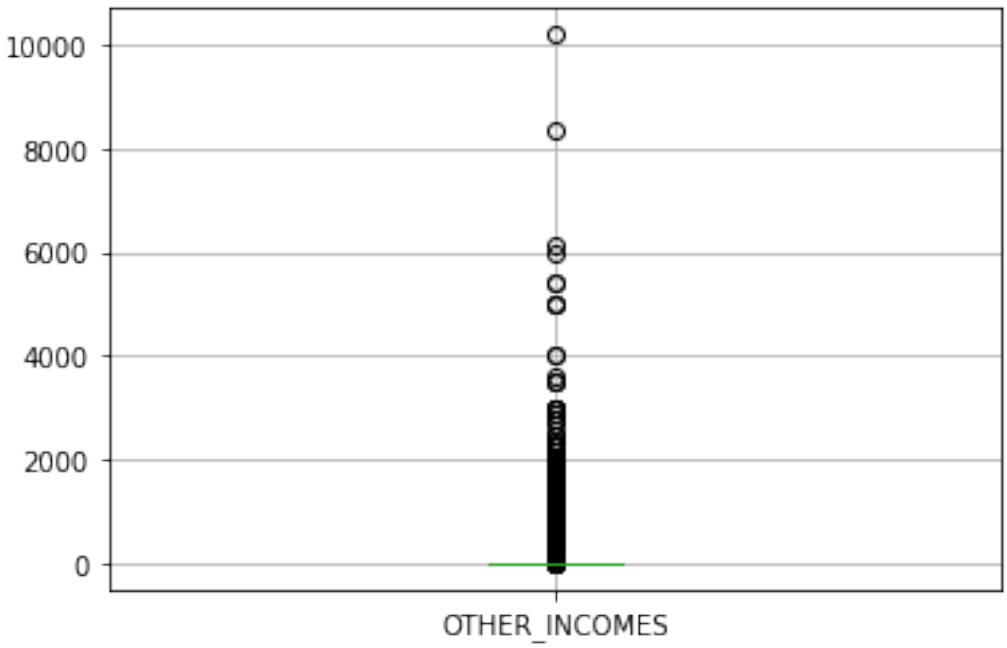


[284]:
```
sns.displot(data = train, x = 'PERSONAL_MONTHLY_INCOME', kind = 'kde', hue =␣
↪'TARGET_LABEL_BAD=1')
plt.show()
```

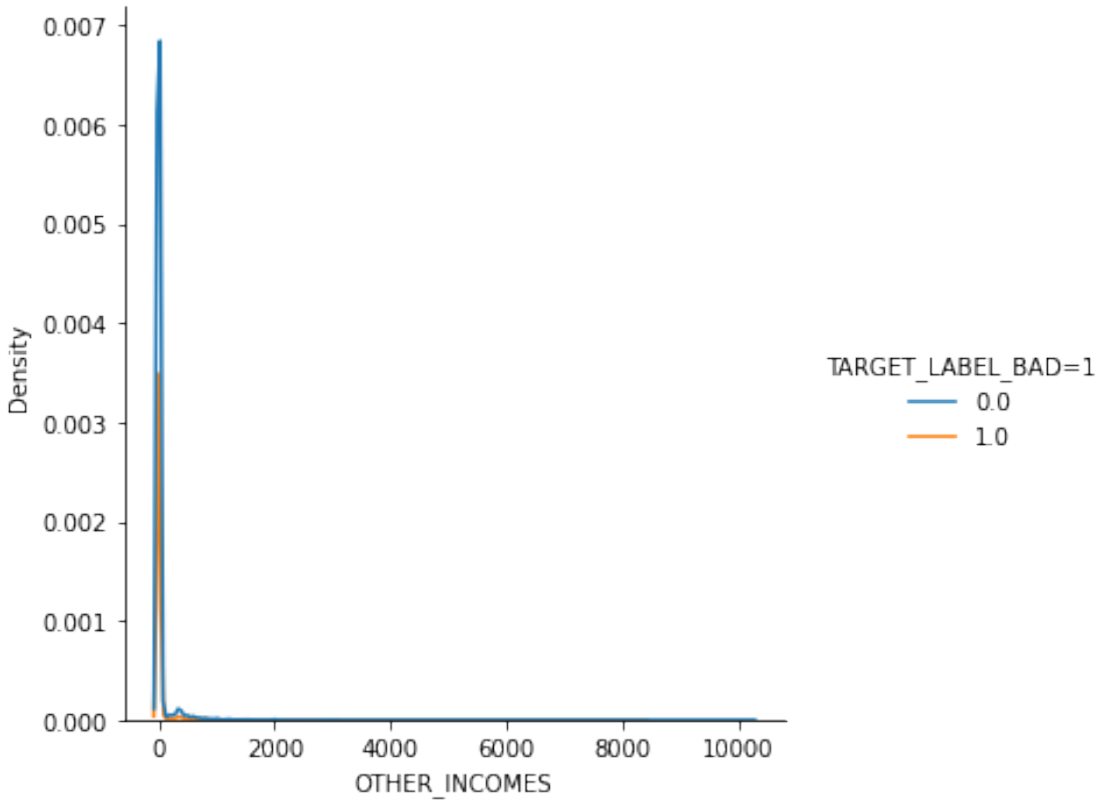


[285]:
```
train.boxplot(column='OTHER_INCOMES')
```

[285]: <matplotlib.axes._subplots.AxesSubplot at 0x7f406a5cbfd0>

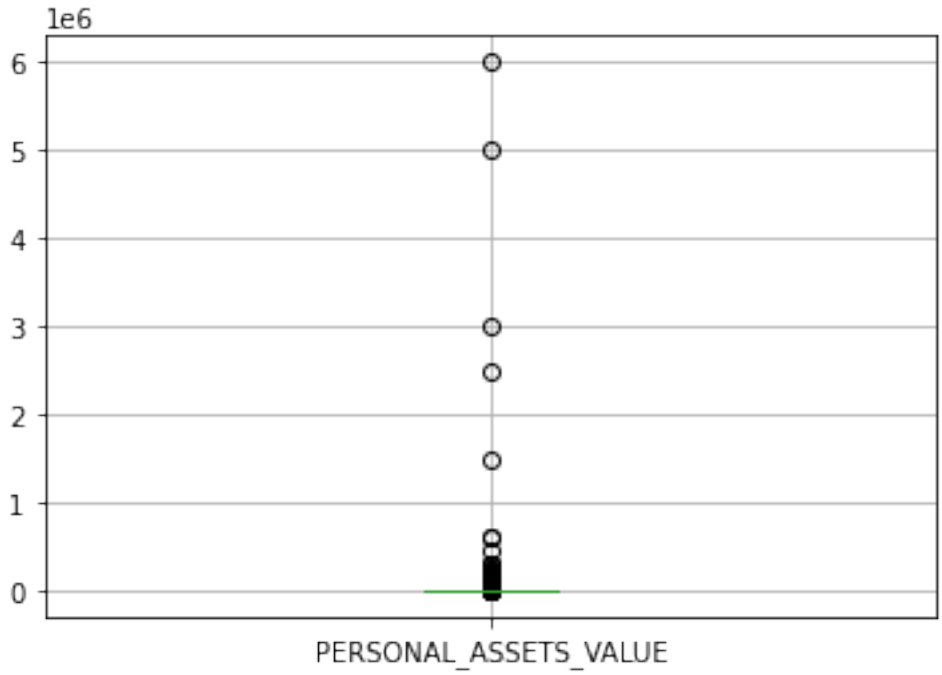


[286]:
```
sns.displot(data = train, x = 'OTHER_INCOMES', kind = 'kde', hue =␣
↪'TARGET_LABEL_BAD=1')
plt.show()
```

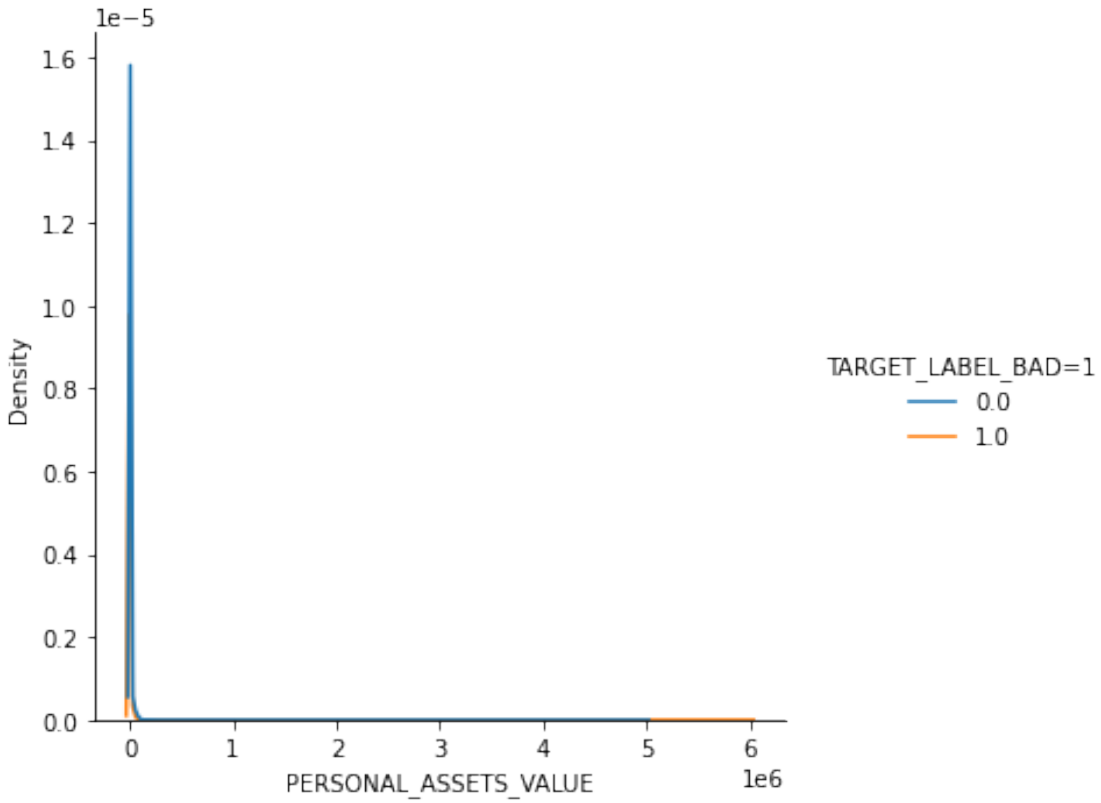


[287]:
```
train.boxplot(column='PERSONAL_ASSETS_VALUE')
```

[287]: <matplotlib.axes._subplots.AxesSubplot at 0x7f406a4b9c90>

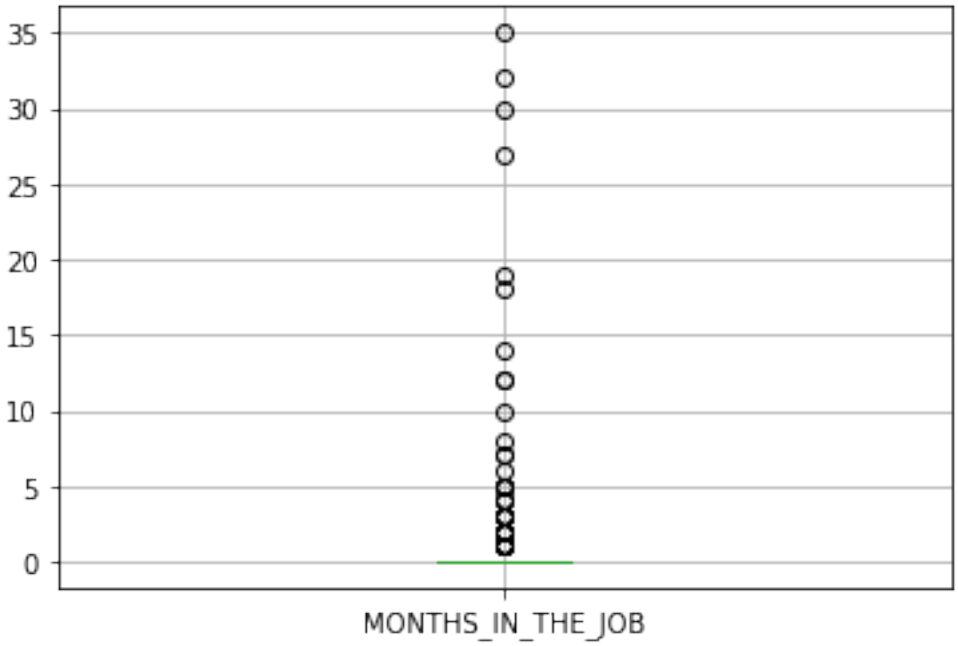


[288]:
```
sns.displot(data = train, x = 'PERSONAL_ASSETS_VALUE', kind = 'kde', hue =␣
↪'TARGET_LABEL_BAD=1')
plt.show()
```

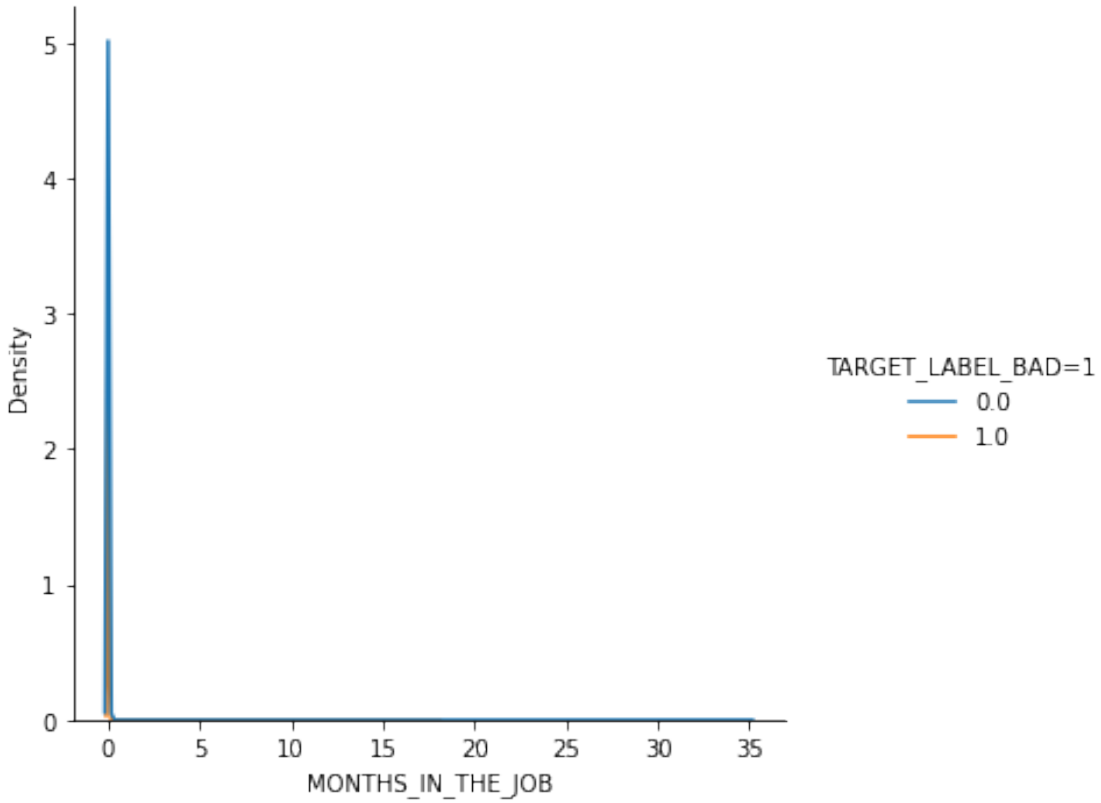


[289]:
```
train.boxplot(column='MONTHS_IN_THE_JOB')
```

[289]: <matplotlib.axes._subplots.AxesSubplot at 0x7f406a4d2ad0>



[290]:
```
sns.displot(data = train, x = 'MONTHS_IN_THE_JOB', kind = 'kde', hue =␣
↪'TARGET_LABEL_BAD=1')
plt.show()
```



[291]:
```
df1=train
```

[292]:
```
#Age looks actually not that bad, we will just leave it there as how it was
```

[293]:
```
#QUANT_DEPENDANTS, cut is at 6
df1 = df1.loc[df1['QUANT_DEPENDANTS'] < 7]
```

[294]:
```
#MONTHS_IN_RESIDENCE, 50 is a reasonable cut
df1 = df1.loc[df1['MONTHS_IN_RESIDENCE'] < 51]
```

[295]:
```
#Personal income is a disaster, try 40000 at first
df1 = df1.loc[df1['PERSONAL_MONTHLY_INCOME'] < 20000]
```

[296]:
```
#OTHER_INCOMES is a disaster too, try 800 at first
df1 = df1.loc[df1['OTHER_INCOMES'] < 800]
```

[297]:
```
#PERSONAL_ASSETS_VALUE... Try 40000
df1 = df1.loc[df1['PERSONAL_ASSETS_VALUE'] < 40000]
```

[298]:
```
#MONTHS_IN_THE_JOB, drop, too many 0s
df1=df1.drop(columns='MONTHS_IN_THE_JOB')
```

[299]:
```
df1.describe()
```

[299]:

Var_Title	PAYMENT_DAY	POSTAL_ADDRESS_TYPE	SEX	QUANT_DEPENDANTS
count	33739.000000	33739.000000	33739.000000	33739.000000
mean	12.884555	1.006610	0.620854	0.632947
std	6.619158	0.081031	0.485182	1.097916
min	1.000000	1.000000	0.000000	0.000000
25%	10.000000	1.000000	0.000000	0.000000
50%	10.000000	1.000000	1.000000	0.000000
75%	15.000000	1.000000	1.000000	1.000000
max	25.000000	2.000000	1.000000	6.000000

Var_Title	NACIONALITY	FLAG_RESIDENCIAL_PHONE	MONTHS_IN_RESIDENCE
count	33739.000000	33739.000000	33739.000000
mean	0.956163	0.837399	9.123685
std	0.204734	0.369007	9.706580
min	0.000000	0.000000	0.000000
25%	1.000000	1.000000	2.000000
50%	1.000000	1.000000	6.000000
75%	1.000000	1.000000	13.000000
max	1.000000	1.000000	50.000000

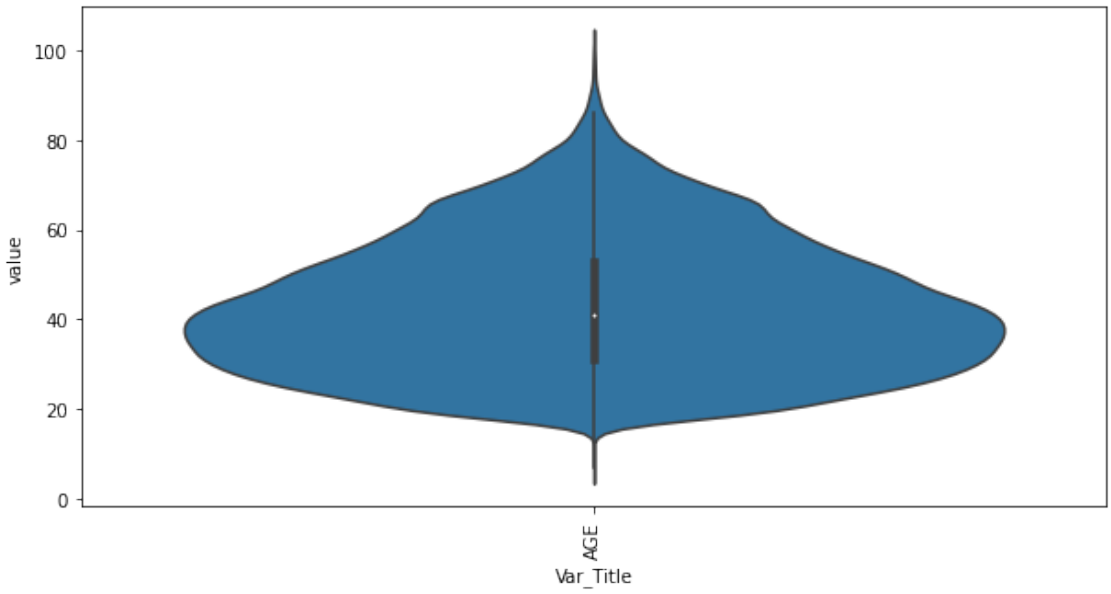
Var_Title	FLAG_EMAIL	PERSONAL_MONTHLY_INCOME	OTHER_INCOMES	FLAG_VISA
count	33739.000000	33739.000000	33739.000000	33739.000000
mean	0.805537	712.456575	15.811122	0.106672
std	0.395793	717.893486	82.584115	0.308700
min	0.000000	69.000000	0.000000	0.000000
25%	1.000000	358.000000	0.000000	0.000000
50%	1.000000	500.000000	0.000000	0.000000
75%	1.000000	800.000000	0.000000	0.000000
max	1.000000	17510.000000	798.000000	1.000000

Var_Title	FLAG_MASTERCARD	FLAG_DINERS	FLAG_AMERICAN_EXPRESS
count	33739.000000	33739.000000	33739.000000
mean	0.093512	0.001245	0.001482
std	0.291153	0.035261	0.038468
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000

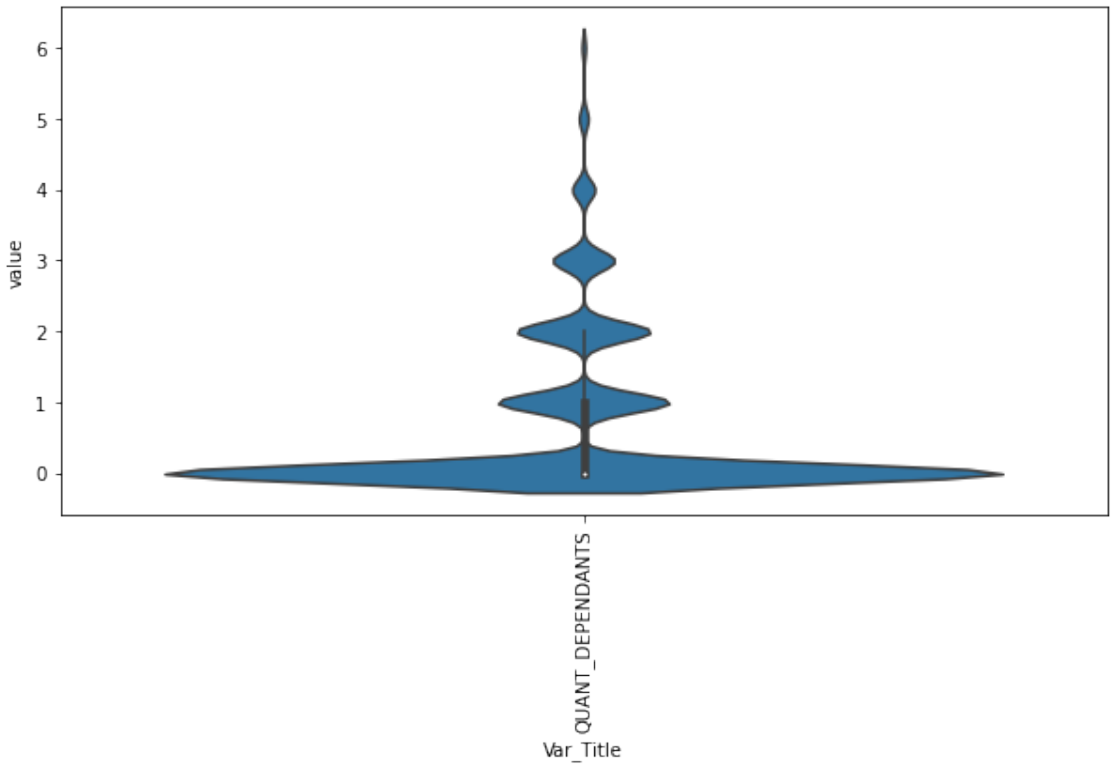
Var_Title	FLAG_OTHER_CARDS	QUANT_BANKING_ACCOUNTS
count	33739.000000	33739.000000
mean	0.001986	0.356324
std	0.044519	0.479476
min	0.000000	0.000000

```
25%                0.000000                0.000000
50%                0.000000                0.000000
75%                0.000000                1.000000
max                1.000000                2.000000

Var_Title  QUANT_SPECIAL_BANKING_ACCOUNTS  PERSONAL_ASSETS_VALUE  \
count                       33739.000000           33739.000000
mean                            0.356324             612.939625
std                             0.479476            3882.065075
min                             0.000000               0.000000
25%                             0.000000               0.000000
50%                             0.000000               0.000000
75%                             1.000000               0.000000
max                             2.000000           38000.000000

Var_Title    QUANT_CARS       COMPANY  FLAG_PROFESSIONAL_PHONE       PRODUCT  \
count      33739.000000  33739.000000             33739.000000  33739.000000
mean           0.332434      0.438335                 0.268502      1.170930
std            0.471093      0.496190                 0.443187      0.438569
min            0.000000      0.000000                 0.000000      1.000000
25%            0.000000      0.000000                 0.000000      1.000000
50%            0.000000      0.000000                 0.000000      1.000000
75%            1.000000      1.000000                 1.000000      1.000000
max            1.000000      1.000000                 1.000000      3.000000

Var_Title           AGE  TARGET_LABEL_BAD=1
count      33739.000000        33739.000000
mean          42.987018            0.262130
std           14.913321            0.439799
min            7.000000            0.000000
25%           31.000000            0.000000
50%           41.000000            0.000000
75%           53.000000            1.000000
max          101.000000            1.000000
```

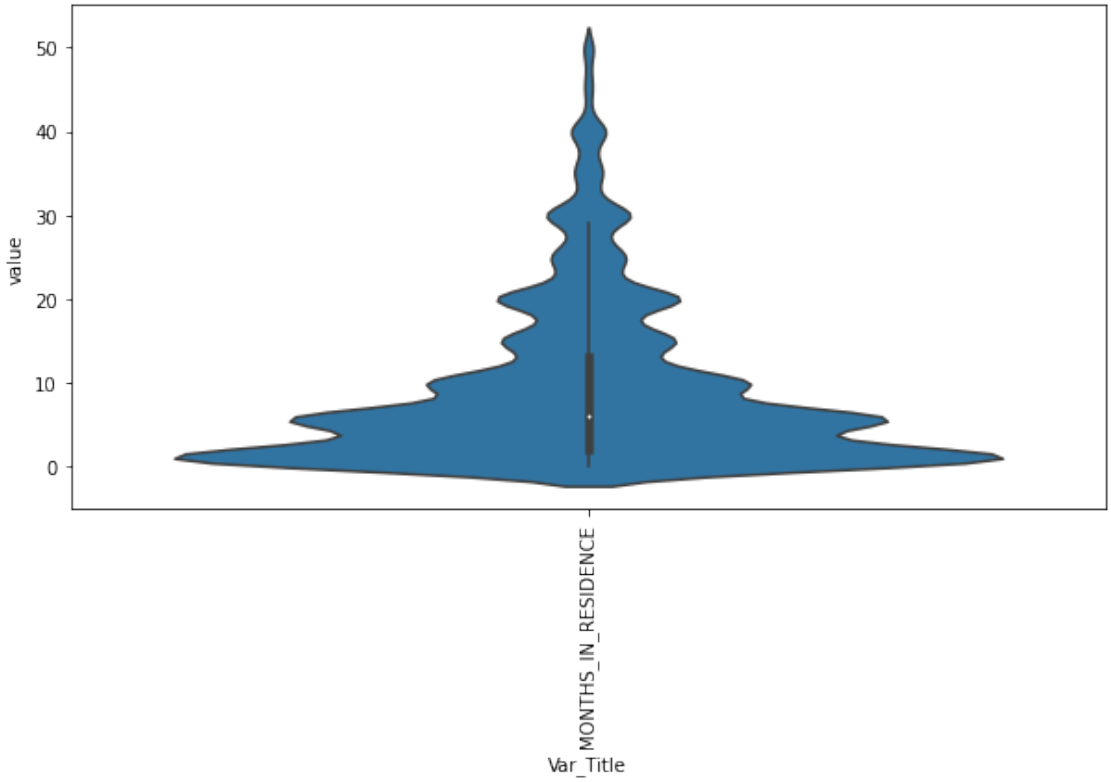
[300]:
```
#AGE Violin
fig, ax = plt.subplots(figsize=(10,5))
a = sns.violinplot(x='Var_Title', y='value',data=pd.melt(df1.loc[:,['AGE']]),␣
 ↪ax=ax)
a.set_xticklabels(a.get_xticklabels(), rotation=90);
```



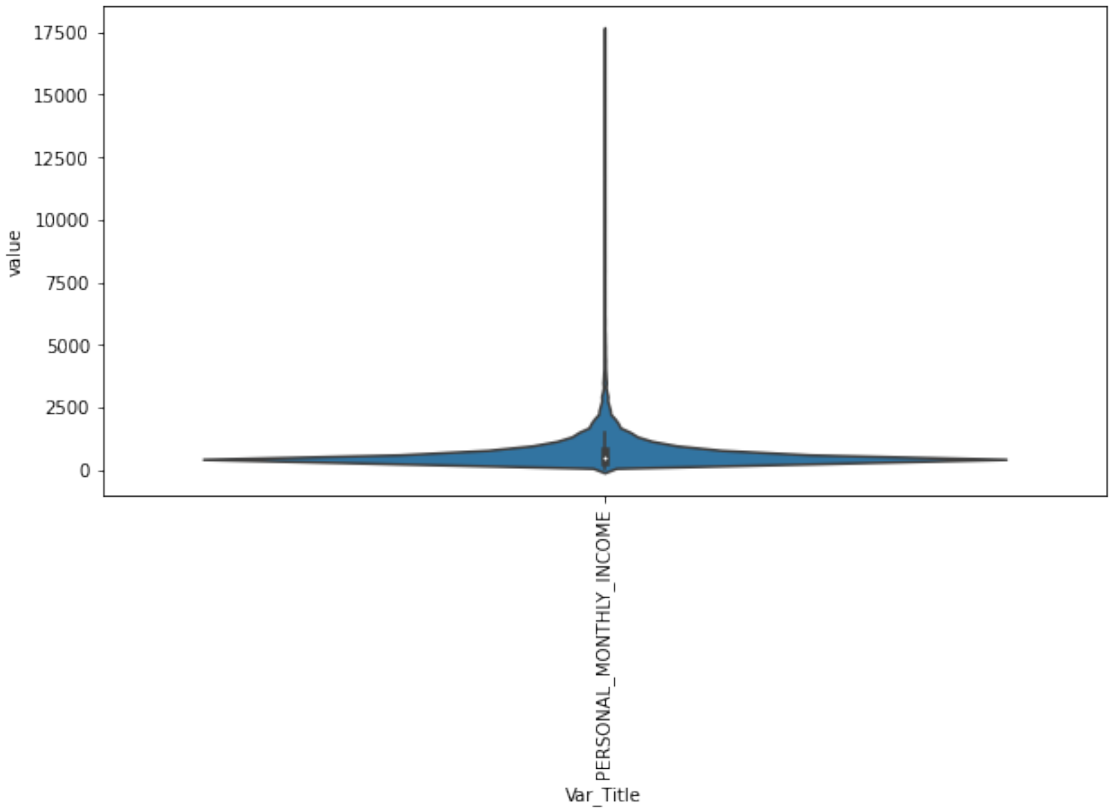
[301]:
```
#QUANT_DEPENDANTS
fig, ax = plt.subplots(figsize=(10,5))
a = sns.violinplot(x='Var_Title', y='value',data=pd.melt(df1.loc[:
 ↪,['QUANT_DEPENDANTS']]), ax=ax)
a.set_xticklabels(a.get_xticklabels(), rotation=90);
```



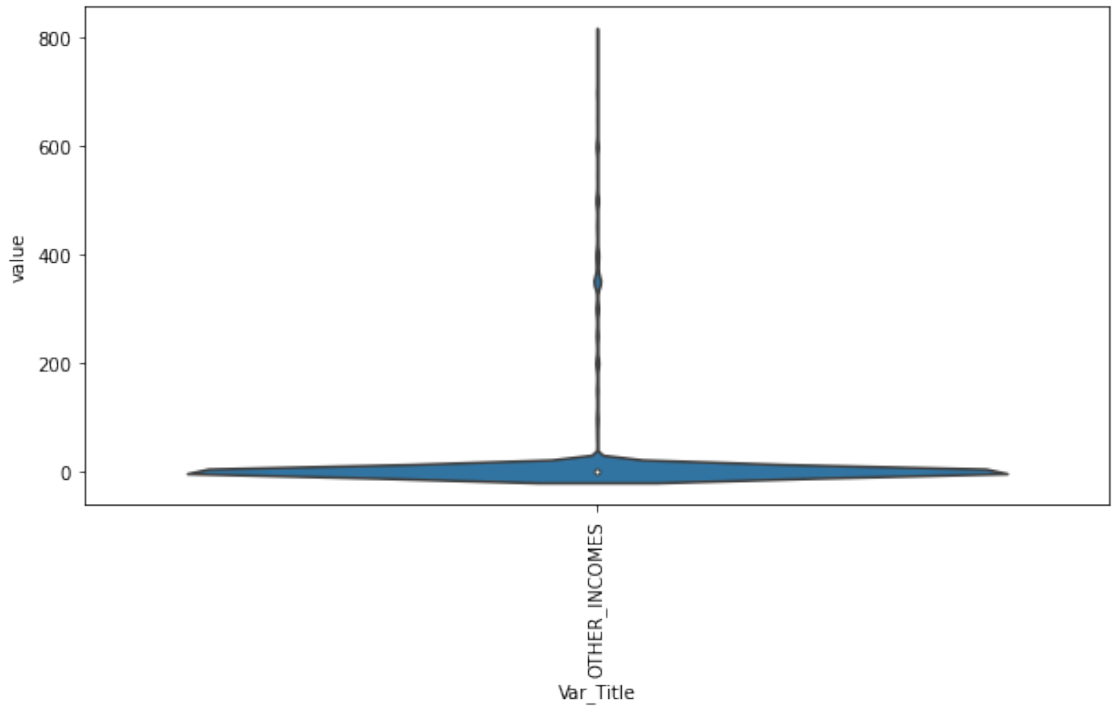
[302]:
```
#MONTHS_IN_RESIDENCE
fig, ax = plt.subplots(figsize=(10,5))
a = sns.violinplot(x='Var_Title', y='value',data=pd.melt(df1.loc[:
 ↪,['MONTHS_IN_RESIDENCE']]), ax=ax)
a.set_xticklabels(a.get_xticklabels(), rotation=90);
```



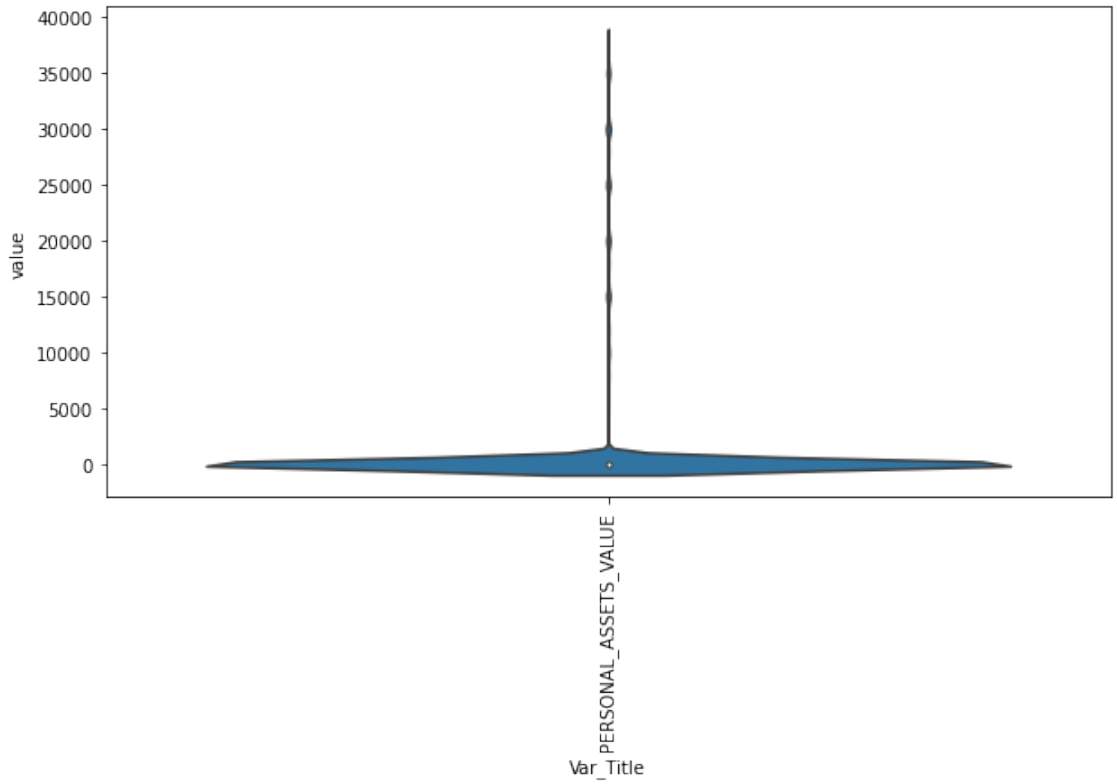
[303]:
```
#Personal Monthly income violin
fig, ax = plt.subplots(figsize=(10,5))
a = sns.violinplot(x='Var_Title', y='value',data=pd.melt(df1.loc[:
 ↪,['PERSONAL_MONTHLY_INCOME']]), ax=ax)
a.set_xticklabels(a.get_xticklabels(), rotation=90);
```



[304]:
```
#Other income violin
fig, ax = plt.subplots(figsize=(10,5))
a = sns.violinplot(x='Var_Title', y='value',data=pd.melt(df1.loc[:
 ↪,['OTHER_INCOMES']]), ax=ax)
a.set_xticklabels(a.get_xticklabels(), rotation=90);
```



[305]:
```
#PERSONAL_ASSETS_VALUE
fig, ax = plt.subplots(figsize=(10,5))
a = sns.violinplot(x='Var_Title', y='value',data=pd.melt(df1.loc[:
 ↪,['PERSONAL_ASSETS_VALUE']]), ax=ax)
a.set_xticklabels(a.get_xticklabels(), rotation=90);
```



We noticed that the data is still not good enough with the outliers, here we start with the personal monthly income again, cause we believe thats the most important factor to bring the data back to normal.

[306]:
```
#15000 is the proper cut I think, from the violin plot, for PERSONAL MONTHLY␣
 ↪INCOME, we cannot cut too many
df1 = df1.loc[df1['PERSONAL_MONTHLY_INCOME'] < 15000]
```

[307]:
```
#We need to modify OTHER_INCOMES as well
df1 = df1.loc[df1['OTHER_INCOMES'] < 600]
```

[308]:
```
#PERSONAL_ASSETS_VALUE: Leave it there, the data is strange, we will figure out␣
 ↪if we should delete this in later steps
```

[309]:
```
df1.describe()
```

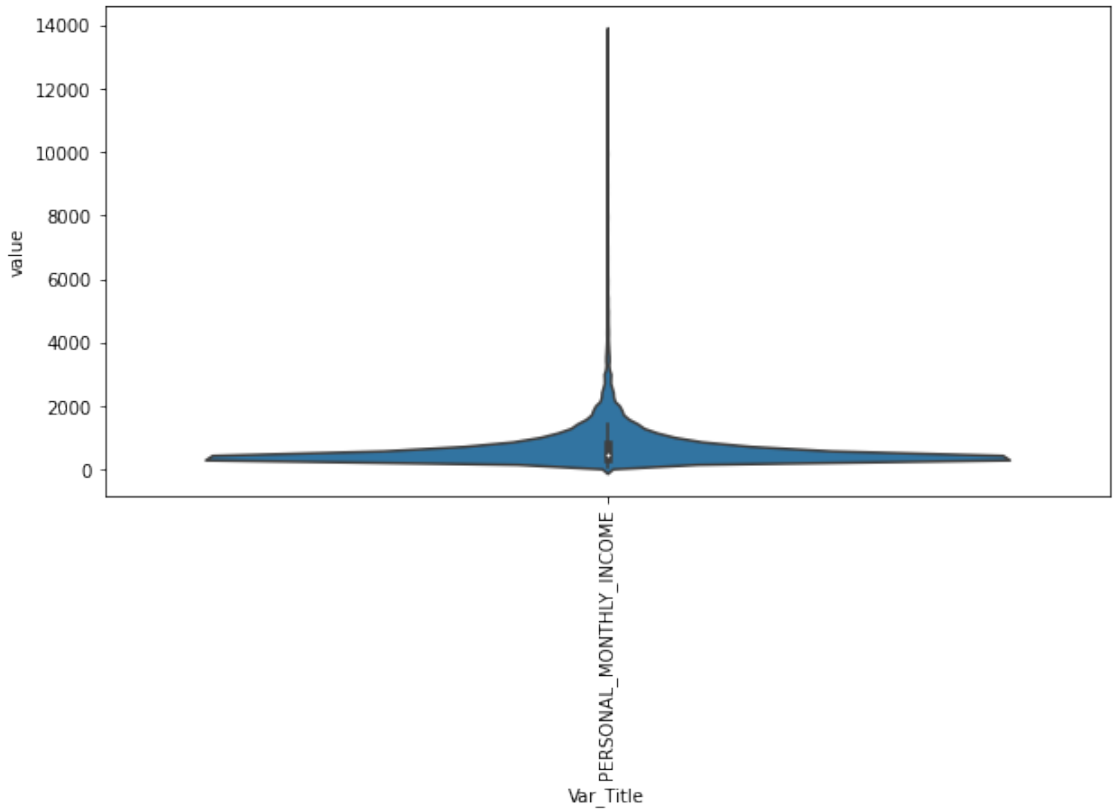
[309]:

Var_Title	PAYMENT_DAY	POSTAL_ADDRESS_TYPE	SEX	QUANT_DEPENDANTS	\
count	33522.000000	33522.000000	33522.000000	33522.000000	
mean	12.883480	1.006623	0.620607	0.631824	
std	6.616225	0.081110	0.485243	1.096871	
min	1.000000	1.000000	0.000000	0.000000	
25%	10.000000	1.000000	0.000000	0.000000	

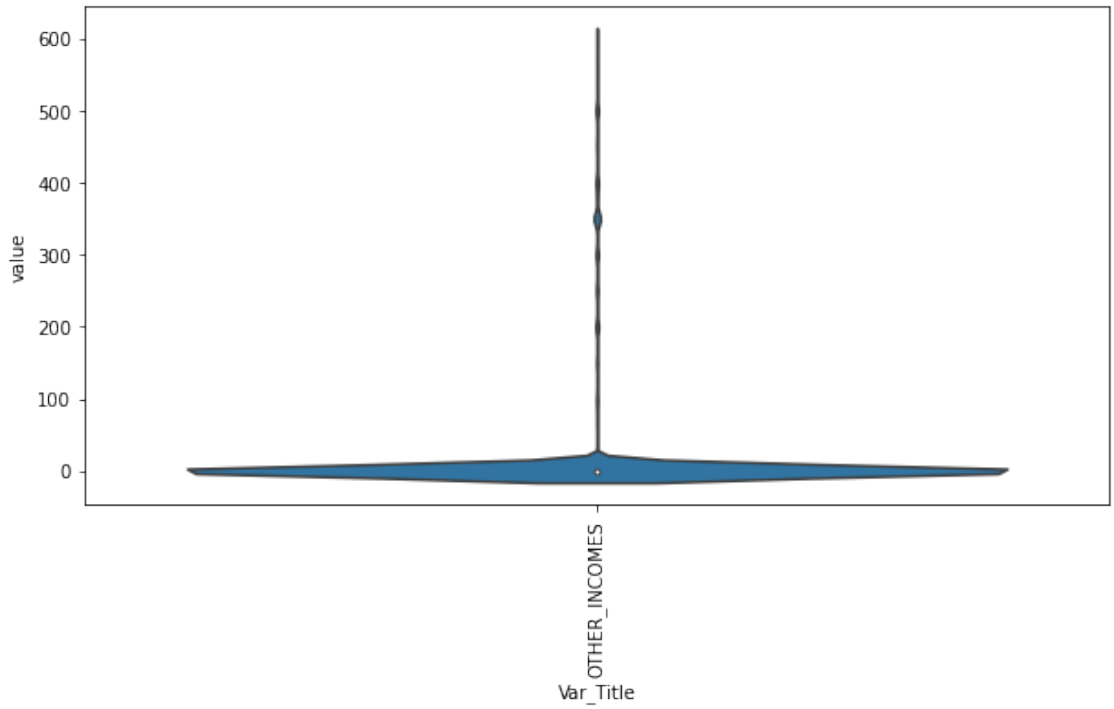
```
50%          10.000000             1.000000      1.000000          0.000000
75%          15.000000             1.000000      1.000000          1.000000
max          25.000000             2.000000      1.000000          6.000000

Var_Title   NACIONALITY  FLAG_RESIDENCIAL_PHONE  MONTHS_IN_RESIDENCE  \
count      33522.000000            33522.000000         33522.000000
mean           0.955910                0.837032             9.108138
std            0.205299                0.369342             9.704732
min            0.000000                0.000000             0.000000
25%            1.000000                1.000000             2.000000
50%            1.000000                1.000000             6.000000
75%            1.000000                1.000000            13.000000
max            1.000000                1.000000            50.000000

Var_Title    FLAG_EMAIL  PERSONAL_MONTHLY_INCOME  OTHER_INCOMES     FLAG_VISA  \
count      33522.000000             33522.000000   33522.000000  33522.000000
mean           0.806097               709.721933      11.760512      0.105751
std            0.395359               689.427599      64.932670      0.307524
min            0.000000                69.000000       0.000000      0.000000
25%            1.000000               358.000000       0.000000      0.000000
50%            1.000000               500.000000       0.000000      0.000000
75%            1.000000               800.000000       0.000000      0.000000
max            1.000000             13757.000000     599.000000      1.000000

Var_Title  FLAG_MASTERCARD   FLAG_DINERS  FLAG_AMERICAN_EXPRESS  \
count         33522.000000  33522.000000           33522.000000
mean              0.092864      0.001253               0.001492
std               0.290247      0.035375               0.038592
min               0.000000      0.000000               0.000000
25%               0.000000      0.000000               0.000000
50%               0.000000      0.000000               0.000000
75%               0.000000      0.000000               0.000000
max               1.000000      1.000000               1.000000

Var_Title  FLAG_OTHER_CARDS  QUANT_BANKING_ACCOUNTS  \
count          33522.000000            33522.000000
mean               0.001999                0.355587
std                0.044663                0.479259
min                0.000000                0.000000
25%                0.000000                0.000000
50%                0.000000                0.000000
75%                0.000000                1.000000
max                1.000000                2.000000

Var_Title  QUANT_SPECIAL_BANKING_ACCOUNTS  PERSONAL_ASSETS_VALUE  \
count                        33522.000000           33522.000000
mean                             0.355587             607.105483
```

```
std                              0.479259           3860.848213
min                              0.000000              0.000000
25%                              0.000000              0.000000
50%                              0.000000              0.000000
75%                              1.000000              0.000000
max                              2.000000          38000.000000

Var_Title    QUANT_CARS       COMPANY  FLAG_PROFESSIONAL_PHONE       PRODUCT  \
count      33522.000000  33522.000000             33522.000000  33522.000000
mean           0.331543      0.438220                 0.268242      1.170843
std            0.470775      0.496176                 0.443051      0.438333
min            0.000000      0.000000                 0.000000      1.000000
25%            0.000000      0.000000                 0.000000      1.000000
50%            0.000000      0.000000                 0.000000      1.000000
75%            1.000000      1.000000                 1.000000      1.000000
max            1.000000      1.000000                 1.000000      3.000000

Var_Title           AGE  TARGET_LABEL_BAD=1
count      33522.000000        33522.000000
mean          42.986099            0.261977
std           14.917561            0.439717
min            7.000000            0.000000
25%           31.000000            0.000000
50%           41.000000            0.000000
75%           53.000000            1.000000
max          101.000000            1.000000
```

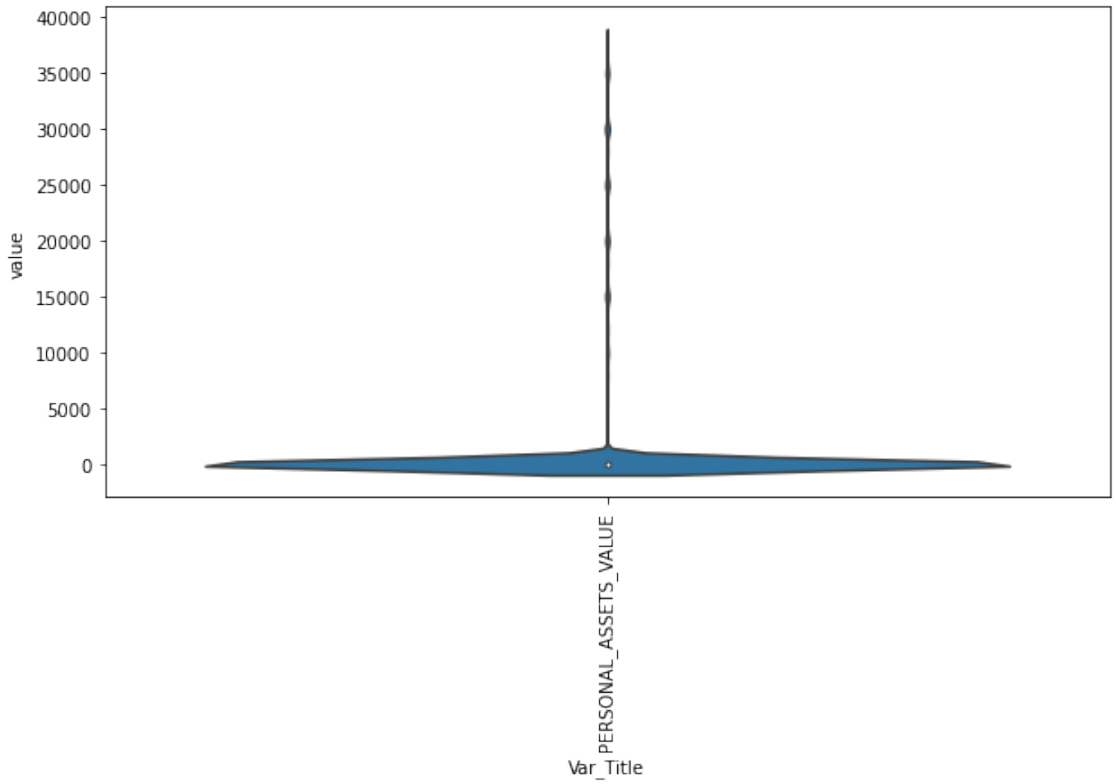
[310]:
```
#PERSONAL_MONTHLY_INCOME violin
fig, ax = plt.subplots(figsize=(10,5))
a = sns.violinplot(x='Var_Title', y='value',data=pd.melt(df1.loc[:
 ↪,['PERSONAL_MONTHLY_INCOME']]), ax=ax)
a.set_xticklabels(a.get_xticklabels(), rotation=90);
```



[311]:
```
#OTHER_INCOMES violin
fig, ax = plt.subplots(figsize=(10,5))
a = sns.violinplot(x='Var_Title', y='value',data=pd.melt(df1.loc[:
 ↪,['OTHER_INCOMES']]), ax=ax)
a.set_xticklabels(a.get_xticklabels(), rotation=90);
```



[312]:
```
#PERSONAL_ASSETS_VALUE
fig, ax = plt.subplots(figsize=(10,5))
a = sns.violinplot(x='Var_Title', y='value',data=pd.melt(df1.loc[:
 ↪,['PERSONAL_ASSETS_VALUE']]), ax=ax)
a.set_xticklabels(a.get_xticklabels(), rotation=90);
```



Still does not look too good, but thats the best we can do for here, if we cut too much for the outliers, our sample amount will be drastcally decreased.

[313]:
```
cleantrain=df1
#record the clean train set
```

Now we implement three new variables that we think might be important to the data

[314]:
```
cleantrain['TOTAL_INCOME'] = cleantrain['PERSONAL_MONTHLY_INCOME'] +␣
 ↪cleantrain['OTHER_INCOMES']
cleantrain['TOTAL_BANKING_ACCOUNTS'] = cleantrain['QUANT_BANKING_ACCOUNTS'] +␣
 ↪cleantrain['QUANT_SPECIAL_BANKING_ACCOUNTS']
cleantrain['TOTAL_TYPE_OF_CARDS'] = cleantrain['FLAG_VISA'] +␣
 ↪cleantrain['FLAG_MASTERCARD']+cleantrain['FLAG_DINERS']+cleantrain['FLAG_AMERICAN_EXPRESS']
```

[315]:
```
cleantrain.describe()
```

[315]:

Var_Title	PAYMENT_DAY	POSTAL_ADDRESS_TYPE	SEX	QUANT_DEPENDANTS	\
count	33522.000000	33522.000000	33522.000000	33522.000000	
mean	12.883480	1.006623	0.620607	0.631824	
std	6.616225	0.081110	0.485243	1.096871	
min	1.000000	1.000000	0.000000	0.000000	
25%	10.000000	1.000000	0.000000	0.000000	

```
50%          10.000000             1.000000      1.000000          0.000000
75%          15.000000             1.000000      1.000000          1.000000
max          25.000000             2.000000      1.000000          6.000000

Var_Title   NACIONALITY  FLAG_RESIDENCIAL_PHONE  MONTHS_IN_RESIDENCE  \
count      33522.000000            33522.000000         33522.000000
mean           0.955910                0.837032             9.108138
std            0.205299                0.369342             9.704732
min            0.000000                0.000000             0.000000
25%            1.000000                1.000000             2.000000
50%            1.000000                1.000000             6.000000
75%            1.000000                1.000000            13.000000
max            1.000000                1.000000            50.000000

Var_Title    FLAG_EMAIL  PERSONAL_MONTHLY_INCOME  OTHER_INCOMES     FLAG_VISA  \
count      33522.000000             33522.000000   33522.000000  33522.000000
mean           0.806097               709.721933      11.760512      0.105751
std            0.395359               689.427599      64.932670      0.307524
min            0.000000                69.000000       0.000000      0.000000
25%            1.000000               358.000000       0.000000      0.000000
50%            1.000000               500.000000       0.000000      0.000000
75%            1.000000               800.000000       0.000000      0.000000
max            1.000000             13757.000000     599.000000      1.000000

Var_Title  FLAG_MASTERCARD   FLAG_DINERS  FLAG_AMERICAN_EXPRESS  \
count         33522.000000  33522.000000           33522.000000
mean              0.092864      0.001253               0.001492
std               0.290247      0.035375               0.038592
min               0.000000      0.000000               0.000000
25%               0.000000      0.000000               0.000000
50%               0.000000      0.000000               0.000000
75%               0.000000      0.000000               0.000000
max               1.000000      1.000000               1.000000

Var_Title  FLAG_OTHER_CARDS  QUANT_BANKING_ACCOUNTS  \
count          33522.000000            33522.000000
mean               0.001999                0.355587
std                0.044663                0.479259
min                0.000000                0.000000
25%                0.000000                0.000000
50%                0.000000                0.000000
75%                0.000000                1.000000
max                1.000000                2.000000

Var_Title  QUANT_SPECIAL_BANKING_ACCOUNTS  PERSONAL_ASSETS_VALUE  \
count                        33522.000000           33522.000000
mean                             0.355587             607.105483
```

```
std                              0.479259           3860.848213
min                              0.000000              0.000000
25%                              0.000000              0.000000
50%                              0.000000              0.000000
75%                              1.000000              0.000000
max                              2.000000          38000.000000

Var_Title    QUANT_CARS       COMPANY  FLAG_PROFESSIONAL_PHONE       PRODUCT  \
count      33522.000000  33522.000000             33522.000000  33522.000000
mean           0.331543      0.438220                 0.268242      1.170843
std            0.470775      0.496176                 0.443051      0.438333
min            0.000000      0.000000                 0.000000      1.000000
25%            0.000000      0.000000                 0.000000      1.000000
50%            0.000000      0.000000                 0.000000      1.000000
75%            1.000000      1.000000                 1.000000      1.000000
max            1.000000      1.000000                 1.000000      3.000000

Var_Title           AGE  TARGET_LABEL_BAD=1  TOTAL_INCOME  \
count      33522.000000        33522.000000  33522.000000
mean          42.986099            0.261977    721.482444
std           14.917561            0.439717    689.720611
min            7.000000            0.000000     69.000000
25%           31.000000            0.000000    368.000000
50%           41.000000            0.000000    500.000000
75%           53.000000            1.000000    800.000000
max          101.000000            1.000000  13757.000000

Var_Title  TOTAL_BANKING_ACCOUNTS  TOTAL_TYPE_OF_CARDS
count                33522.000000         33522.000000
mean                     0.711175             0.203359
std                      0.958517             0.510172
min                      0.000000             0.000000
25%                      0.000000             0.000000
50%                      0.000000             0.000000
75%                      2.000000             0.000000
max                      4.000000             4.000000
```

0.0.5 Outliers for Test set

We do not clean the outliers for test set, as sometimes the outliers are the real situation that we need to faced, it is part of the future. If we truly want to see how our model can perform the prediction, we need to keep the outliers in the test set

[316]:
```
df2=test
df2.isnull().any()
```

[316]:
```
Var_Title
PAYMENT_DAY                          False
POSTAL_ADDRESS_TYPE                  False
SEX                                  False
MARITAL_STATUS                       False
QUANT_DEPENDANTS                     False
STATE_OF_BIRTH                       False
NACIONALITY                          False
RESIDENCIAL_STATE                    False
RESIDENCIAL_CITY                     False
RESIDENCIAL_BOROUGH                  False
FLAG_RESIDENCIAL_PHONE               False
RESIDENCE_TYPE                       False
MONTHS_IN_RESIDENCE                  False
FLAG_EMAIL                           False
PERSONAL_MONTHLY_INCOME              False
OTHER_INCOMES                        False
FLAG_VISA                            False
FLAG_MASTERCARD                      False
FLAG_DINERS                          False
FLAG_AMERICAN_EXPRESS                False
FLAG_OTHER_CARDS                     False
QUANT_BANKING_ACCOUNTS               False
QUANT_SPECIAL_BANKING_ACCOUNTS       False
PERSONAL_ASSETS_VALUE                False
QUANT_CARS                           False
COMPANY                              False
FLAG_PROFESSIONAL_PHONE              False
MONTHS_IN_THE_JOB                    False
PRODUCT                              False
AGE                                  False
RESIDENCIAL_ZIP_3                    False
TARGET_LABEL_BAD=1                   False
dtype: bool
```

[ ]:
```
df2=df2.drop(columns='MONTHS_IN_THE_JOB')
```

[334]:
```
cleantest=df2
cleantest['TOTAL_INCOME'] = cleantest['PERSONAL_MONTHLY_INCOME'] +␣
 ↪cleantest['OTHER_INCOMES']
cleantest['TOTAL_BANKING_ACCOUNTS'] = cleantest['QUANT_BANKING_ACCOUNTS'] +␣
 ↪cleantest['QUANT_SPECIAL_BANKING_ACCOUNTS']
cleantest['TOTAL_TYPE_OF_CARDS'] = cleantest['FLAG_VISA'] +␣
 ↪cleantest['FLAG_MASTERCARD']+cleantest['FLAG_DINERS']+cleantest['FLAG_AMERICAN_EXPRESS']+cl
```

[319]:
```
cleantest.describe(include='all')
```

[319]:

Var_Title	PAYMENT_DAY	POSTAL_ADDRESS_TYPE	SEX	MARITAL_STATUS
count	15000.000000	15000.000000	15000.000000	15000
unique	NaN	NaN	NaN	3
top	NaN	NaN	NaN	3
freq	NaN	NaN	NaN	9220
mean	12.889600	1.005933	0.612133	NaN
std	6.597276	0.076802	0.487280	NaN
min	1.000000	1.000000	0.000000	NaN
25%	10.000000	1.000000	0.000000	NaN
50%	10.000000	1.000000	1.000000	NaN
75%	15.000000	1.000000	1.000000	NaN
max	25.000000	2.000000	1.000000	NaN

Var_Title	QUANT_DEPENDANTS	STATE_OF_BIRTH	NACIONALITY	RESIDENCIAL_STATE
count	15000.000000	15000	15000.000000	15000
unique	NaN	3	NaN	3
top	NaN	2	NaN	2
freq	NaN	7713	NaN	9566
mean	0.634267	NaN	0.958133	NaN
std	1.231777	NaN	0.200291	NaN
min	0.000000	NaN	0.000000	NaN
25%	0.000000	NaN	1.000000	NaN
50%	0.000000	NaN	1.000000	NaN
75%	1.000000	NaN	1.000000	NaN
max	53.000000	NaN	1.000000	NaN

Var_Title	RESIDENCIAL_CITY	RESIDENCIAL_BOROUGH	FLAG_RESIDENCIAL_PHONE
count	15000	15000	15000.000000
unique	4	6	NaN
top	2	3	NaN
freq	11725	9801	NaN
mean	NaN	NaN	0.837667
std	NaN	NaN	0.368769
min	NaN	NaN	0.000000
25%	NaN	NaN	1.000000
50%	NaN	NaN	1.000000
75%	NaN	NaN	1.000000
max	NaN	NaN	1.000000

Var_Title	RESIDENCE_TYPE	MONTHS_IN_RESIDENCE	FLAG_EMAIL
count	15000	15000.000000	15000.000000
unique	5	NaN	NaN
top	1.0	NaN	NaN
freq	13118	NaN	NaN
mean	NaN	9.457333	0.798800
std	NaN	10.242036	0.400911
min	NaN	0.000000	0.000000

25%	NaN	2.000000	1.000000
50%	NaN	6.000000	1.000000
75%	NaN	14.000000	1.000000
max	NaN	99.000000	1.000000

Var_Title	PERSONAL_MONTHLY_INCOME	OTHER_INCOMES	FLAG_VISA
count	15000.000000	15000.000000	15000.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	799.874569	44.416593	0.111533
std	3412.383497	1599.998798	0.314802
min	60.000000	0.000000	0.000000
25%	360.000000	0.000000	0.000000
50%	500.000000	0.000000	0.000000
75%	800.000000	0.000000	0.000000
max	198183.000000	194344.000000	1.000000

Var_Title	FLAG_MASTERCARD	FLAG_DINERS	FLAG_AMERICAN_EXPRESS
count	15000.000000	15000.000000	15000.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	0.099333	0.001000	0.002133
std	0.299119	0.031608	0.046140
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000

Var_Title	FLAG_OTHER_CARDS	QUANT_BANKING_ACCOUNTS
count	15000.000000	15000.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	0.002200	0.358000
std	0.046854	0.480123
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	1.000000
max	1.000000	2.000000

Var_Title	QUANT_SPECIAL_BANKING_ACCOUNTS	PERSONAL_ASSETS_VALUE
count	15000.000000	1.500000e+04
unique	NaN	NaN

top	NaN	NaN
freq	NaN	NaN
mean	0.358000	2.137884e+03
std	0.480123	2.027787e+04
min	0.000000	0.000000e+00
25%	0.000000	0.000000e+00
50%	0.000000	0.000000e+00
75%	1.000000	0.000000e+00
max	2.000000	1.800000e+06

Var_Title	QUANT_CARS	COMPANY	FLAG_PROFESSIONAL_PHONE	PRODUCT	\
count	15000.000000	15000.000000	15000.000000	15000.000000	
unique	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	
mean	0.337533	0.442267	0.268533	1.169867	
std	0.472884	0.496672	0.443211	0.438587	
min	0.000000	0.000000	0.000000	1.000000	
25%	0.000000	0.000000	0.000000	1.000000	
50%	0.000000	0.000000	0.000000	1.000000	
75%	1.000000	1.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	3.000000	

Var_Title	AGE	RESIDENCIAL_ZIP_3	TARGET_LABEL_BAD=1	TOTAL_INCOME	\
count	15000.000000	15000	15000.000000	15000.000000	
unique	NaN	4	NaN	NaN	
top	NaN	2	NaN	NaN	
freq	NaN	6069	NaN	NaN	
mean	43.407867	NaN	0.260800	844.291162	
std	15.071138	NaN	0.439086	3768.966713	
min	17.000000	NaN	0.000000	60.000000	
25%	32.000000	NaN	0.000000	372.000000	
50%	42.000000	NaN	0.000000	510.000000	
75%	54.000000	NaN	1.000000	826.240000	
max	106.000000	NaN	1.000000	198183.000000	

Var_Title	TOTAL_BANKING_ACCOUNTS	TOTAL_TYPE_OF_CARDS
count	15000.000000	15000.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	0.716000	0.216200
std	0.960246	0.523331
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	2.000000	0.000000

```
max                      4.000000            4.000000
```

[320]:
```
#saving the result
cleantrain.to_csv("train_clean.csv", index = False)
cleantest.to_csv("test_clean.csv", index = False)
```

0.0.6 Question2: WOE

[131]:
```
cleantrain=pd.read_csv('train_clean.csv')
cleantest=pd.read_csv('test_clean.csv')
```

[54]:
```
bins = sc.woebin(cleantrain, y = 'TARGET_LABEL_BAD=1',
                 min_perc_fine_bin=0.01, # How many bins to cut initially into
                 min_perc_coarse_bin=0.05,  # Minimum percentage per final bin
                 stop_limit=0.02, # Minimum information value
                 max_num_bin=10, # Maximum number of bins
                 method='tree'
                 )
```

```
[INFO] creating woe binning …
Binning on 33522 rows and 34 columns in 00:00:16
```

[55]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

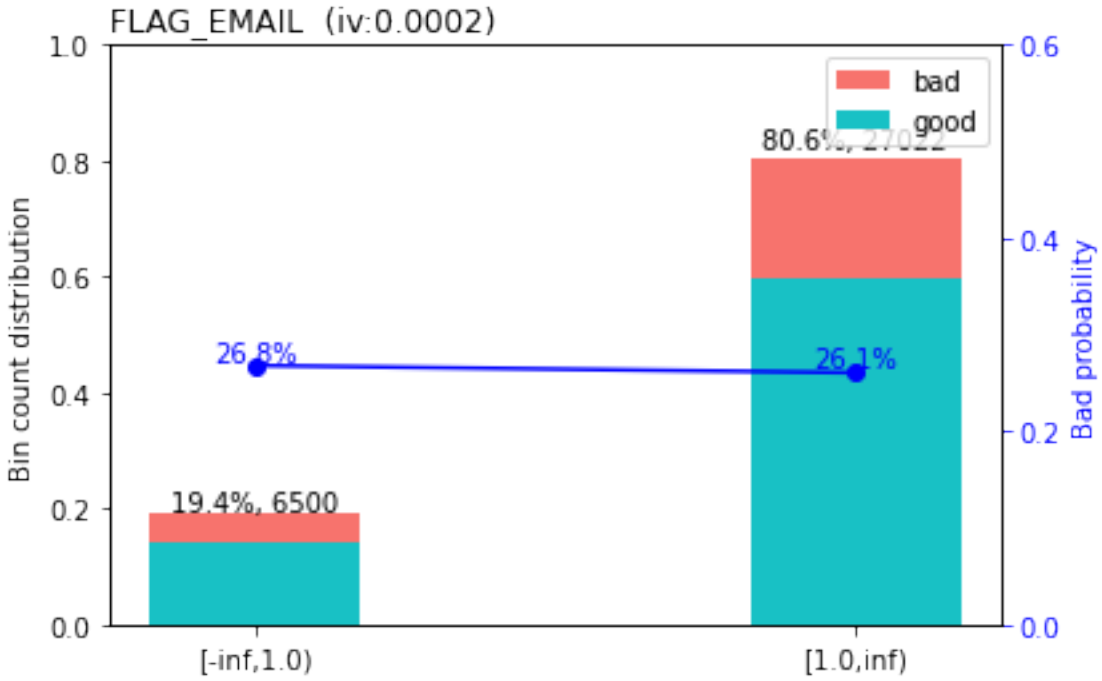
[56]:
```
sc.woebin_plot(bins)
```

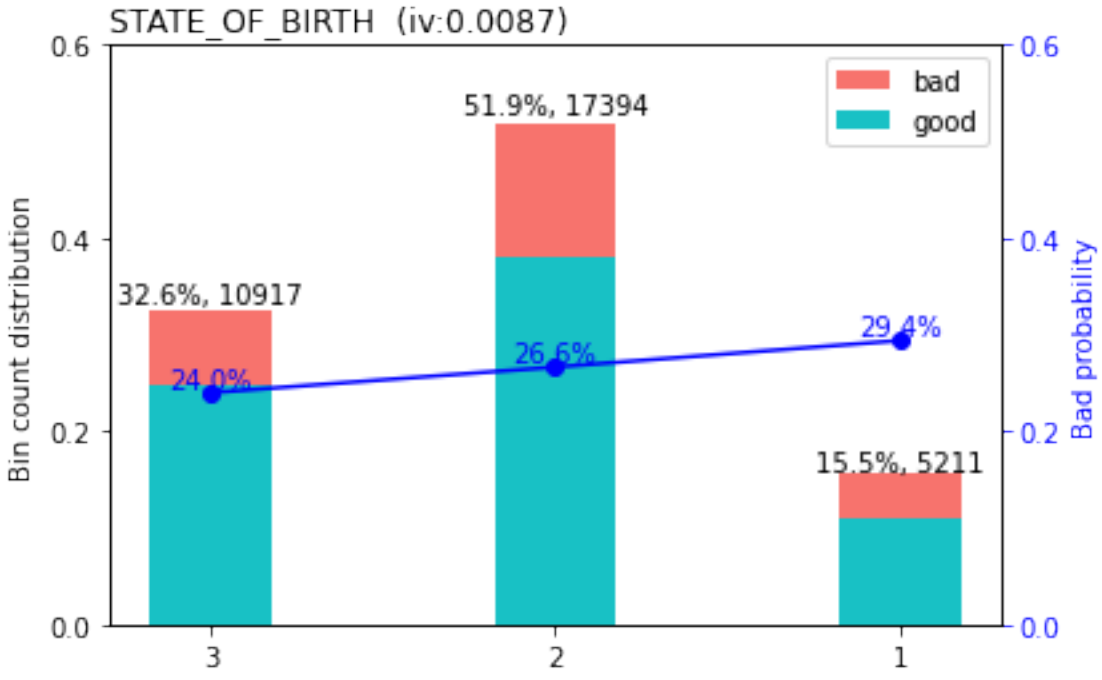
```
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
```

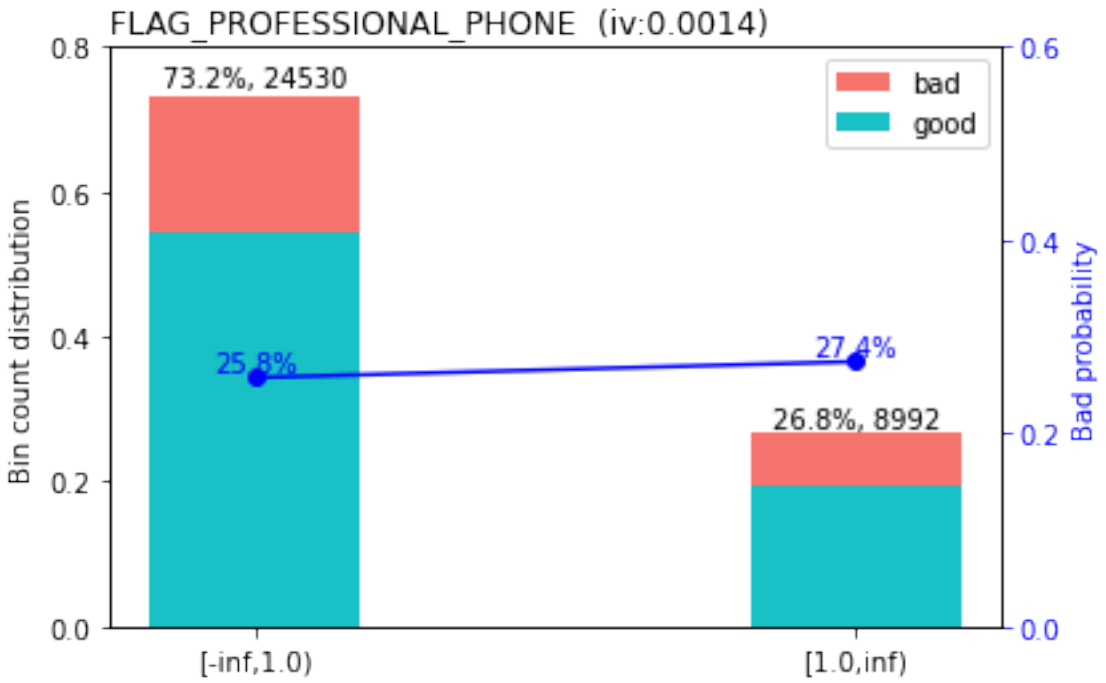
```
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
```

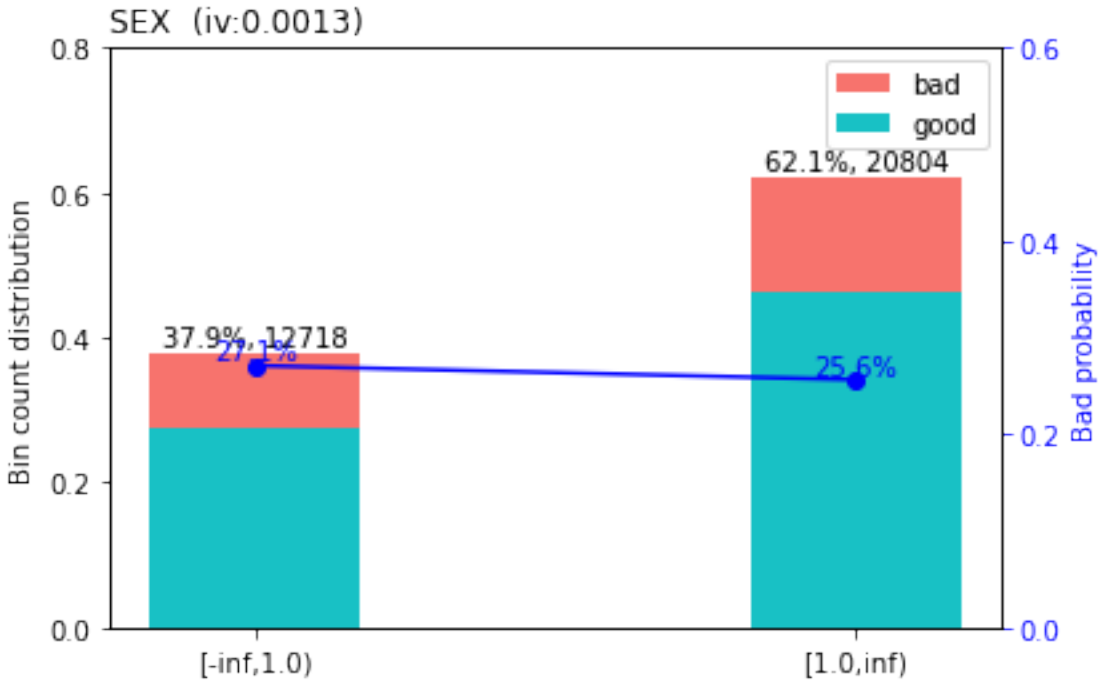
```
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
/usr/local/lib/python3.7/dist-packages/scorecardpy/woebin.py:1203:
RuntimeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the
rcParam `figure.max_open_warning`).
  fig, ax1 = plt.subplots()
```

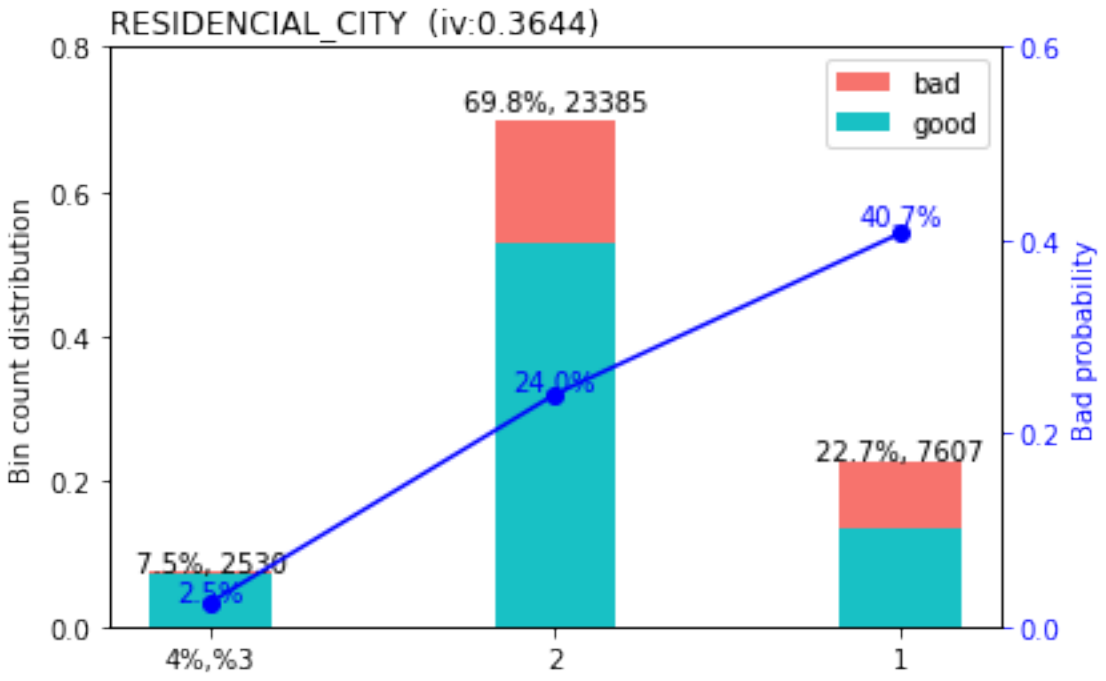
[56]:
```
{'AGE': <Figure size 432x288 with 2 Axes>,
 'COMPANY': <Figure size 432x288 with 2 Axes>,
 'FLAG_AMERICAN_EXPRESS': <Figure size 432x288 with 2 Axes>,
 'FLAG_DINERS': <Figure size 432x288 with 2 Axes>,
 'FLAG_EMAIL': <Figure size 432x288 with 2 Axes>,
 'FLAG_MASTERCARD': <Figure size 432x288 with 2 Axes>,
 'FLAG_OTHER_CARDS': <Figure size 432x288 with 2 Axes>,
 'FLAG_PROFESSIONAL_PHONE': <Figure size 432x288 with 2 Axes>,
 'FLAG_RESIDENCIAL_PHONE': <Figure size 432x288 with 2 Axes>,
 'FLAG_VISA': <Figure size 432x288 with 2 Axes>,
 'MARITAL_STATUS': <Figure size 432x288 with 2 Axes>,
 'MONTHS_IN_RESIDENCE': <Figure size 432x288 with 2 Axes>,
 'NACIONALITY': <Figure size 432x288 with 2 Axes>,
 'OTHER_INCOMES': <Figure size 432x288 with 2 Axes>,
 'PAYMENT_DAY': <Figure size 432x288 with 2 Axes>,
 'PERSONAL_ASSETS_VALUE': <Figure size 432x288 with 2 Axes>,
 'PERSONAL_MONTHLY_INCOME': <Figure size 432x288 with 2 Axes>,
 'POSTAL_ADDRESS_TYPE': <Figure size 432x288 with 2 Axes>,
 'PRODUCT': <Figure size 432x288 with 2 Axes>,
 'QUANT_BANKING_ACCOUNTS': <Figure size 432x288 with 2 Axes>,
 'QUANT_CARS': <Figure size 432x288 with 2 Axes>,
 'QUANT_DEPENDANTS': <Figure size 432x288 with 2 Axes>,
 'QUANT_SPECIAL_BANKING_ACCOUNTS': <Figure size 432x288 with 2 Axes>,
 'RESIDENCE_TYPE': <Figure size 432x288 with 2 Axes>,
 'RESIDENCIAL_BOROUGH': <Figure size 432x288 with 2 Axes>,
 'RESIDENCIAL_CITY': <Figure size 432x288 with 2 Axes>,
 'RESIDENCIAL_STATE': <Figure size 432x288 with 2 Axes>,
 'RESIDENCIAL_ZIP_3': <Figure size 432x288 with 2 Axes>,
 'SEX': <Figure size 432x288 with 2 Axes>,
 'STATE_OF_BIRTH': <Figure size 432x288 with 2 Axes>,
 'TOTAL_BANKING_ACCOUNTS': <Figure size 432x288 with 2 Axes>,
 'TOTAL_INCOME': <Figure size 432x288 with 2 Axes>,
 'TOTAL_TYPE_OF_CARDS': <Figure size 432x288 with 2 Axes>}
```

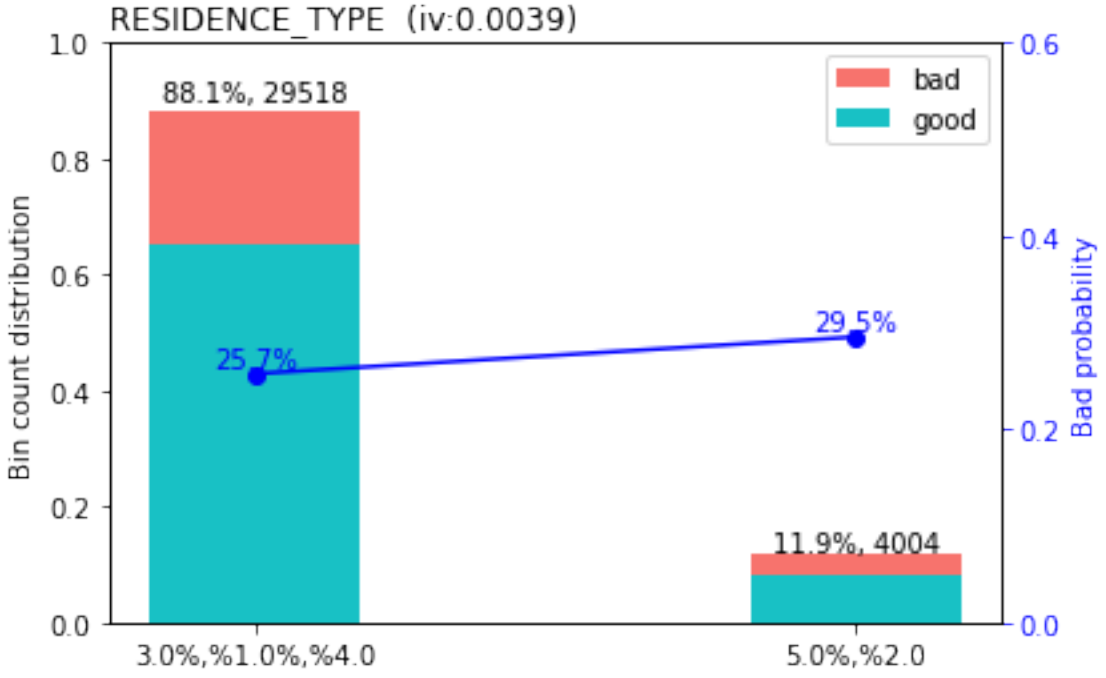

FLAG_EMAIL (iv:0.0002)
bad
good
80.6%, 27022
26.8%
26.1%
19.4%, 6500
Bin count distribution
Bad probability
[-inf,1.0)
[1.0,inf)

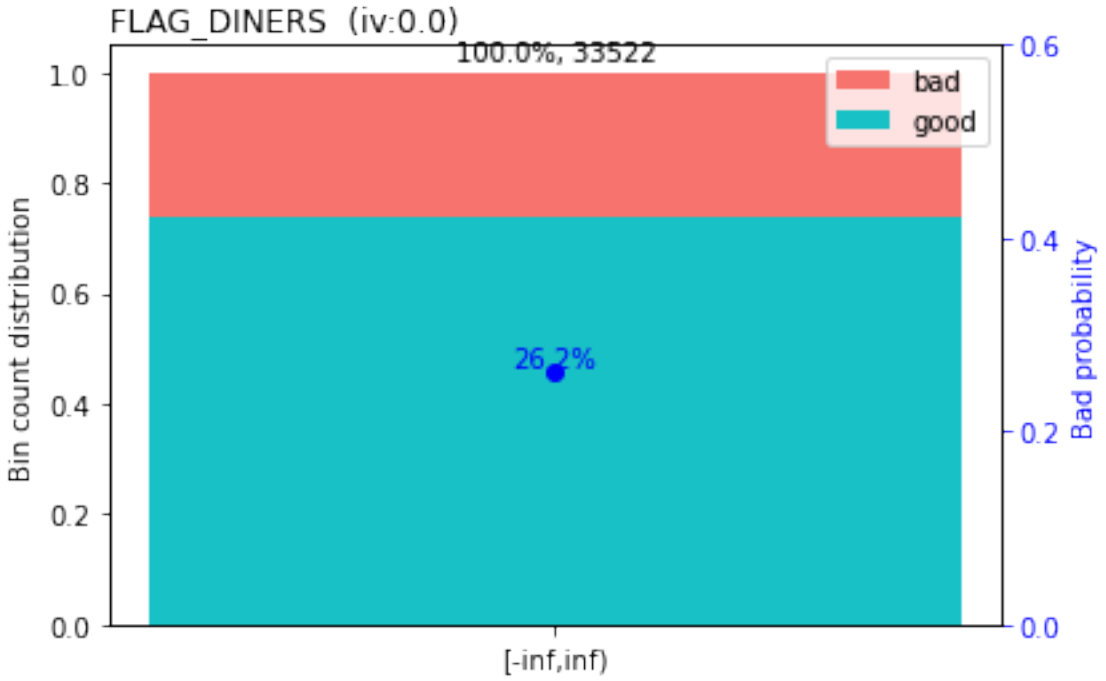

STATE_OF_BIRTH (iv:0.0087)
bad
good
51.9%, 17394
32.6%, 10917
24.0%
26.6%
29.4%
15.5%, 5211
Bin count distribution
Bad probability
3
2
1

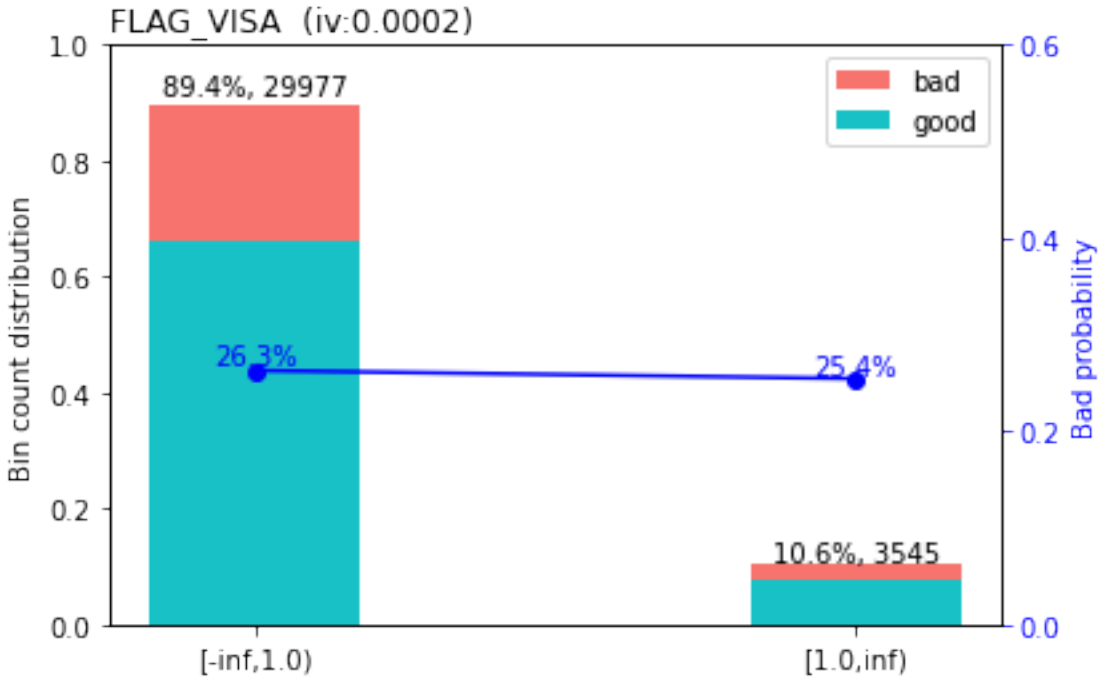

FLAG_PROFESSIONAL_PHONE (iv:0.0014)
Bin count distribution
Bad probability
bad
good
73.2%, 24530
26.8%, 8992
25.8%
27.4%
[-inf,1.0)
[1.0,inf)

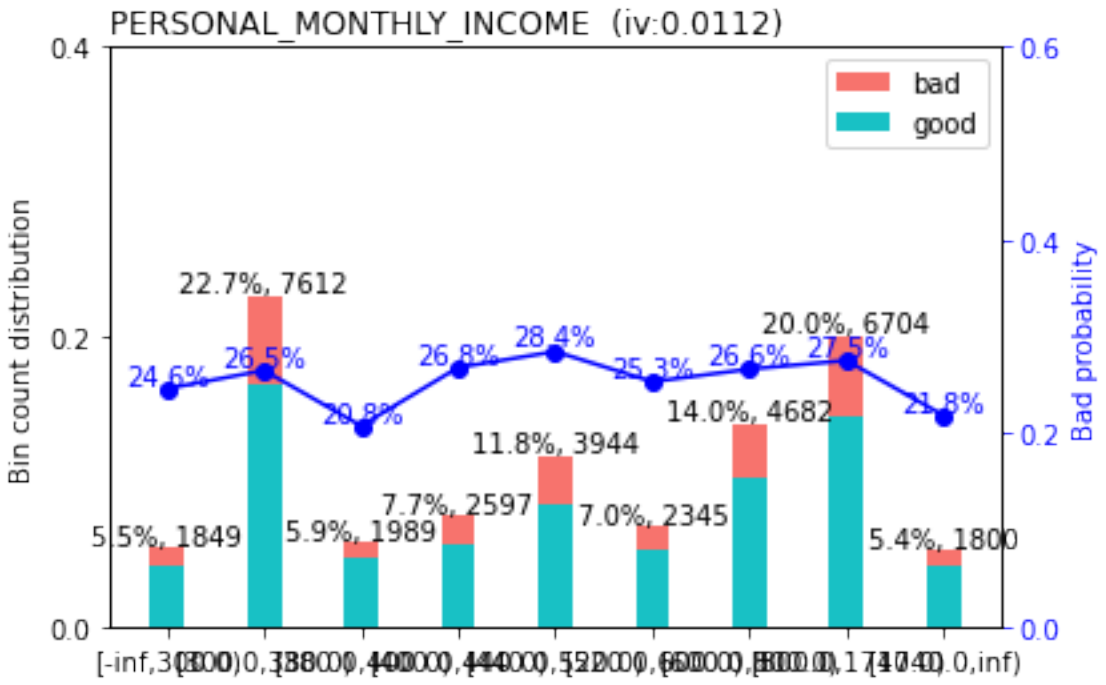

SEX (iv:0.0013)
Bin count distribution
Bad probability
bad
good
62.1%, 20804
37.9%, 12718
27.1%
25.6%
[-inf,1.0)
[1.0,inf)

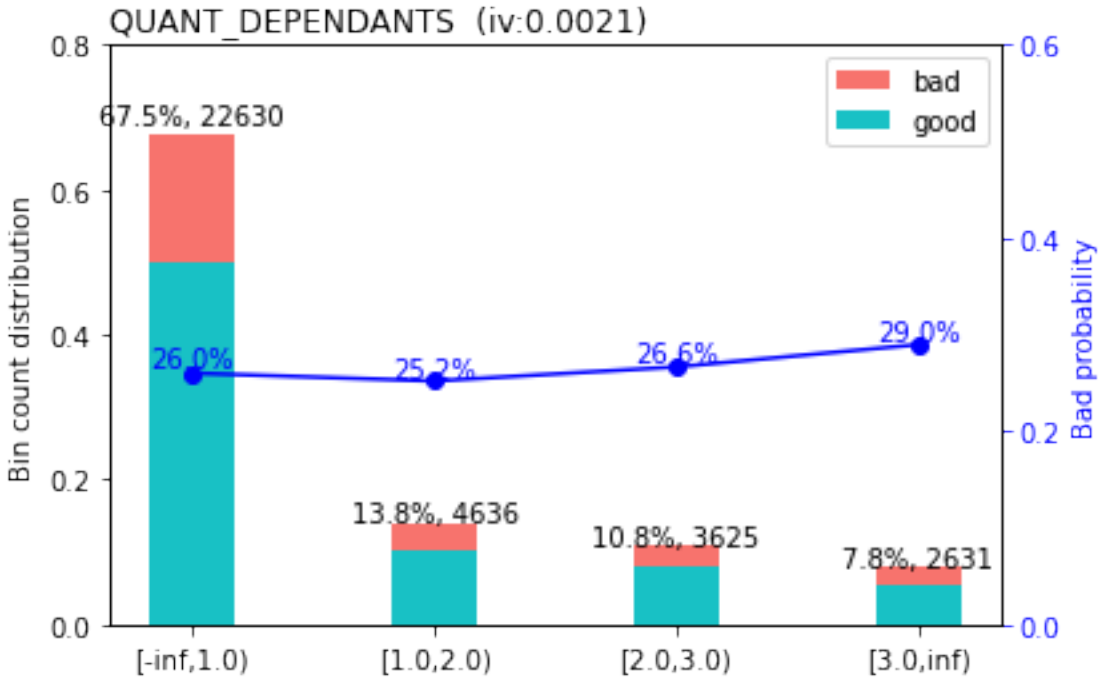

RESIDENCIAL_CITY (iv:0.3644)
Bin count distribution
Bad probability
bad
good
69.8%, 23385
40.7%
24.0%
22.7%, 7607
7.5%, 2530
2.5%
4%,%3
2
1

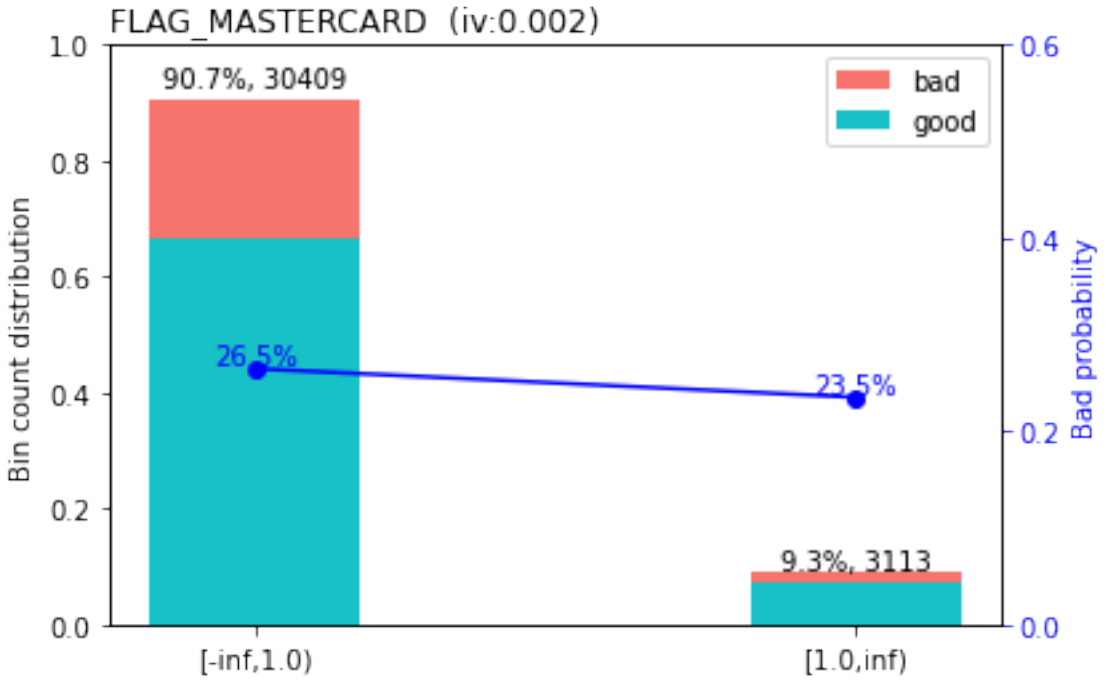

RESIDENCE_TYPE (iv:0.0039)
Bin count distribution
Bad probability
bad
good
88.1%, 29518
29.5%
25.7%
11.9%, 4004
3.0%,%1.0%,%4.0
5.0%,%2.0

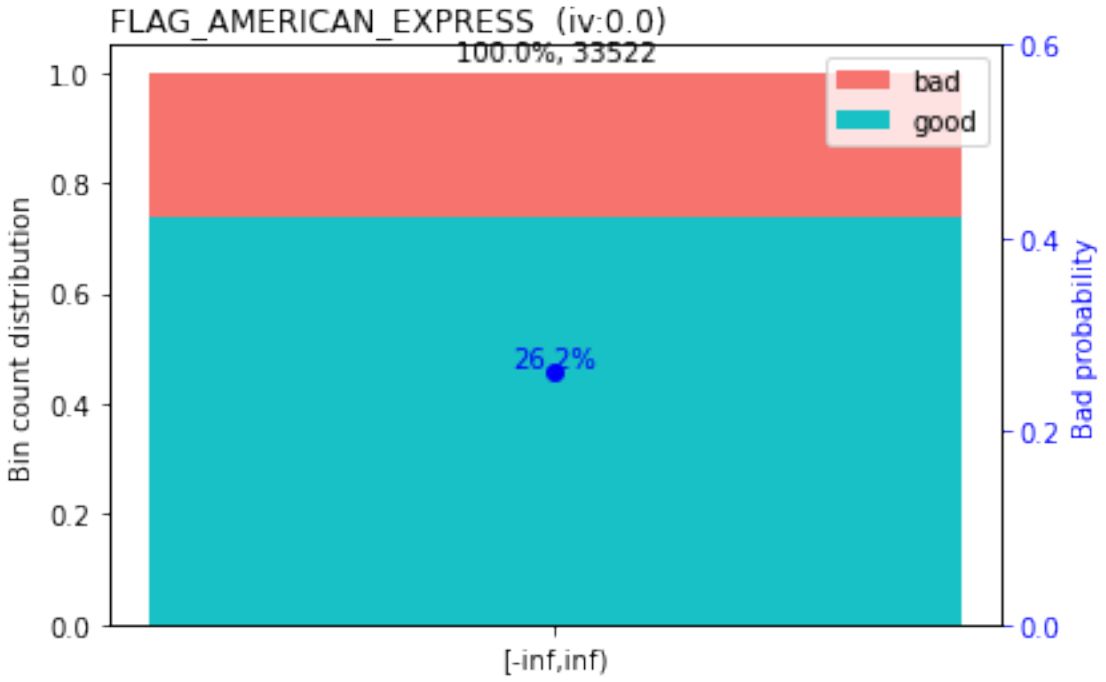

FLAG_DINERS (iv:0.0)
100.0%, 33522
bad
good
26.2%
Bin count distribution
Bad probability
[-inf,inf)

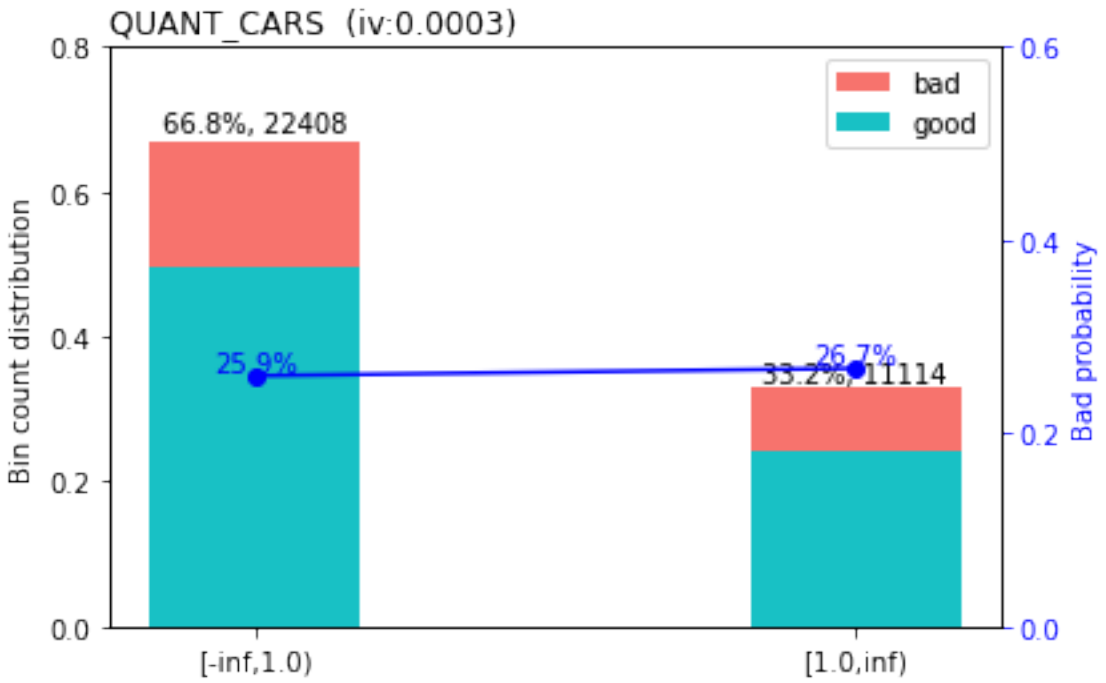

FLAG_VISA (iv:0.0002)
89.4%, 29977
bad
good
26.3%
25.4%
10.6%, 3545
Bin count distribution
Bad probability
[-inf,1.0)
[1.0,inf)

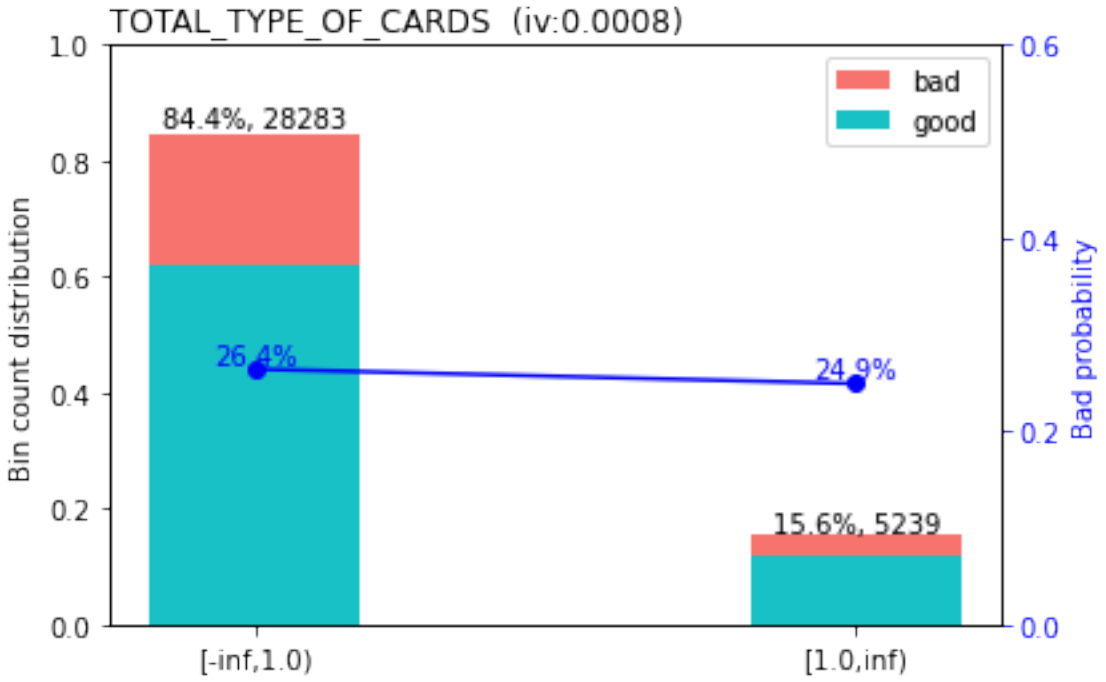

PERSONAL_MONTHLY_INCOME (iv:0.0112)
bad
good
Bin count distribution
Bad probability
0.0
0.2
0.4
0.6
22.7%, 7612
20.0%, 6704
14.0%, 4682
11.8%, 3944
7.7%, 2597
7.0%, 2345
5.9%, 1989
5.5%, 1849
5.4%, 1800
24.6%
26.5%
20.8%
26.8%
28.4%
25.3%
26.6%
27.5%
21.8%
[-inf,3000.0)
[3800.0,3800.0)
[3800.0,4400.0)
[4400.0,4400.0)
[4400.0,5200.0)
[5200.0,6600.0)
[6600.0,8000.0)
[8000.0,17400.0)
[17400.0,inf)

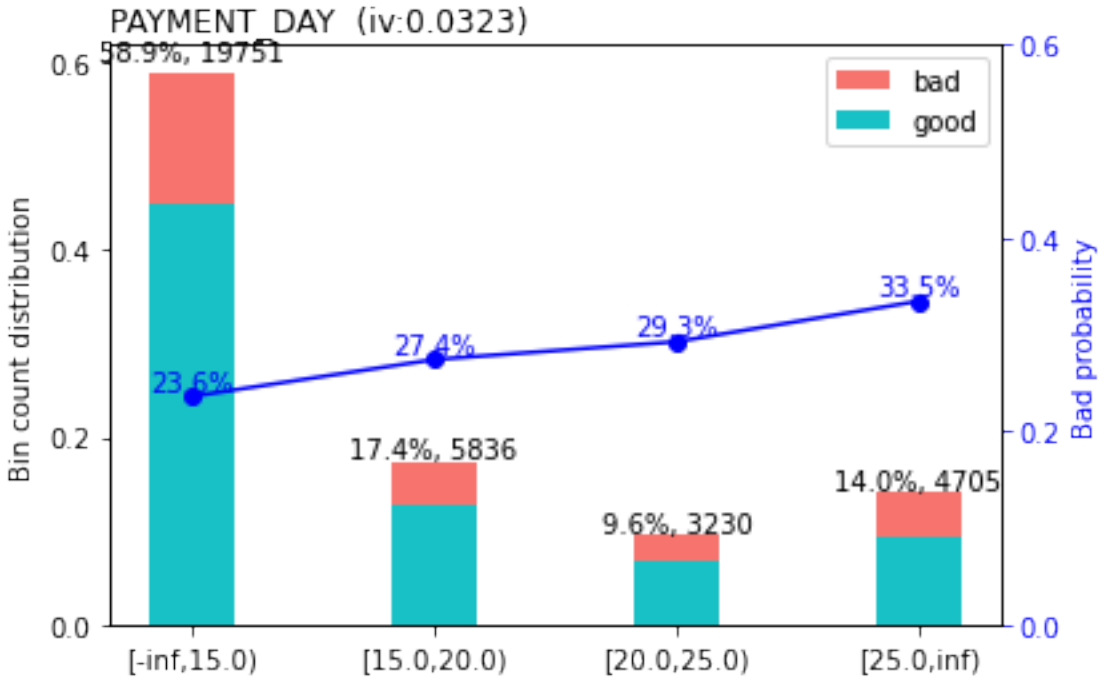

QUANT_DEPENDANTS (iv:0.0021)
bad
good
Bin count distribution
Bad probability
0.0
0.2
0.4
0.6
0.8
67.5%, 22630
13.8%, 4636
10.8%, 3625
7.8%, 2631
26.0%
25.2%
26.6%
29.0%
[-inf,1.0)
[1.0,2.0)
[2.0,3.0)
[3.0,inf)

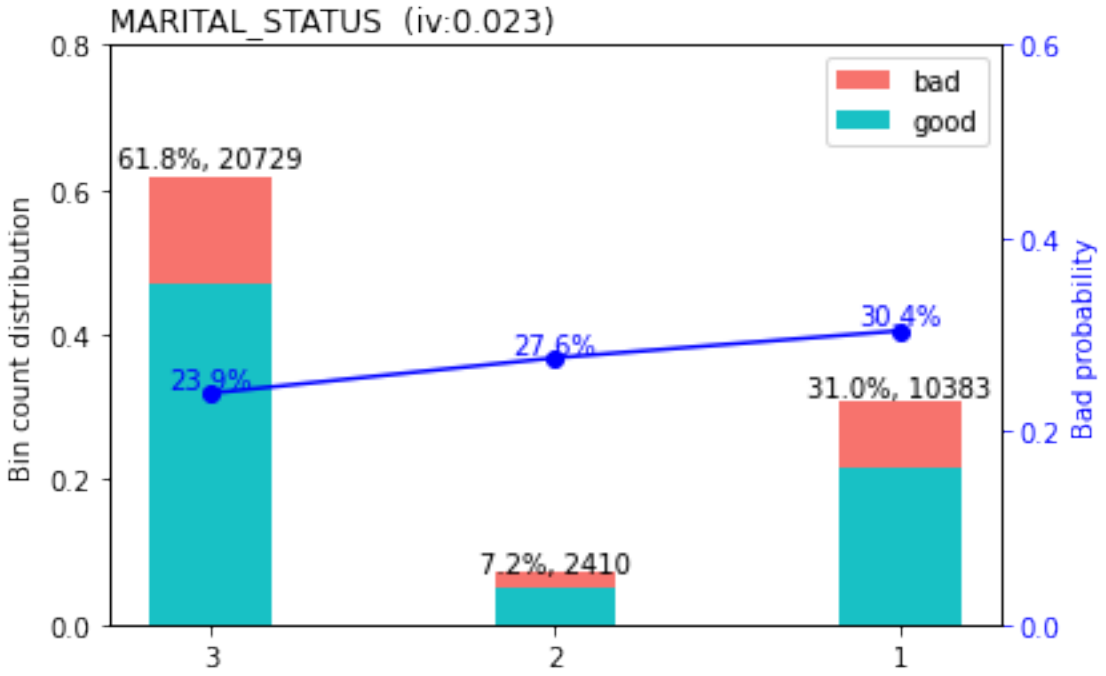

FLAG_MASTERCARD (iv:0.002)
90.7%, 30409
9.3%, 3113
26.5%
23.5%
bad
good
Bin count distribution
Bad probability
[-inf,1.0)
[1.0,inf)

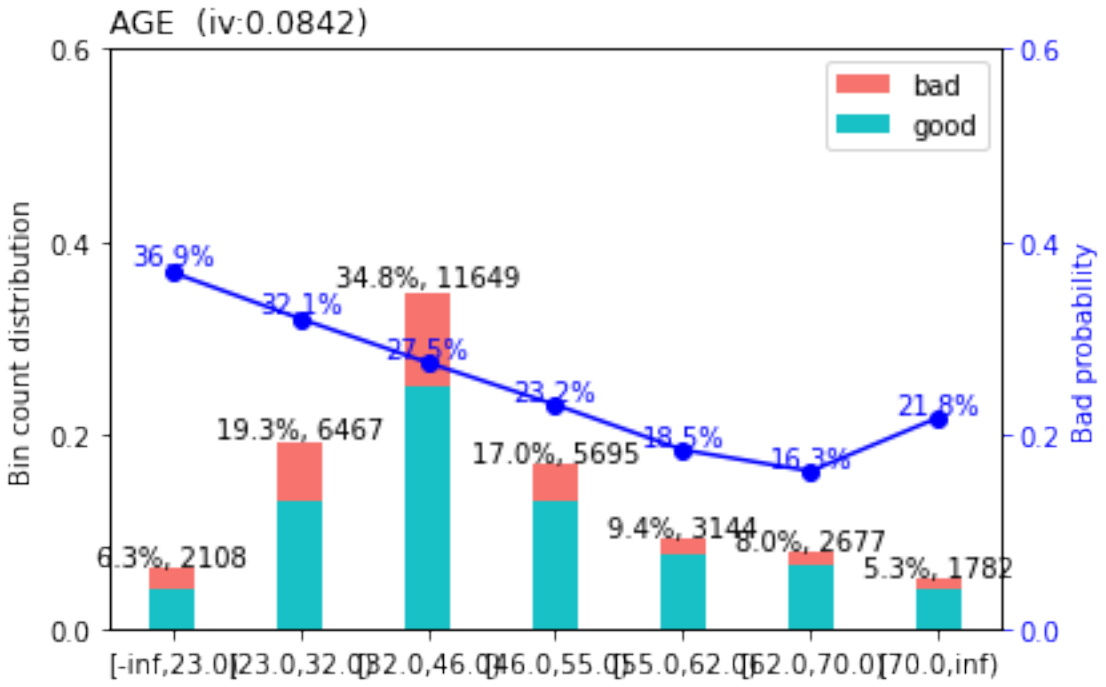

FLAG_AMERICAN_EXPRESS (iv:0.0)
100.0%, 33522
26.2%
bad
good
Bin count distribution
Bad probability
[-inf,inf)

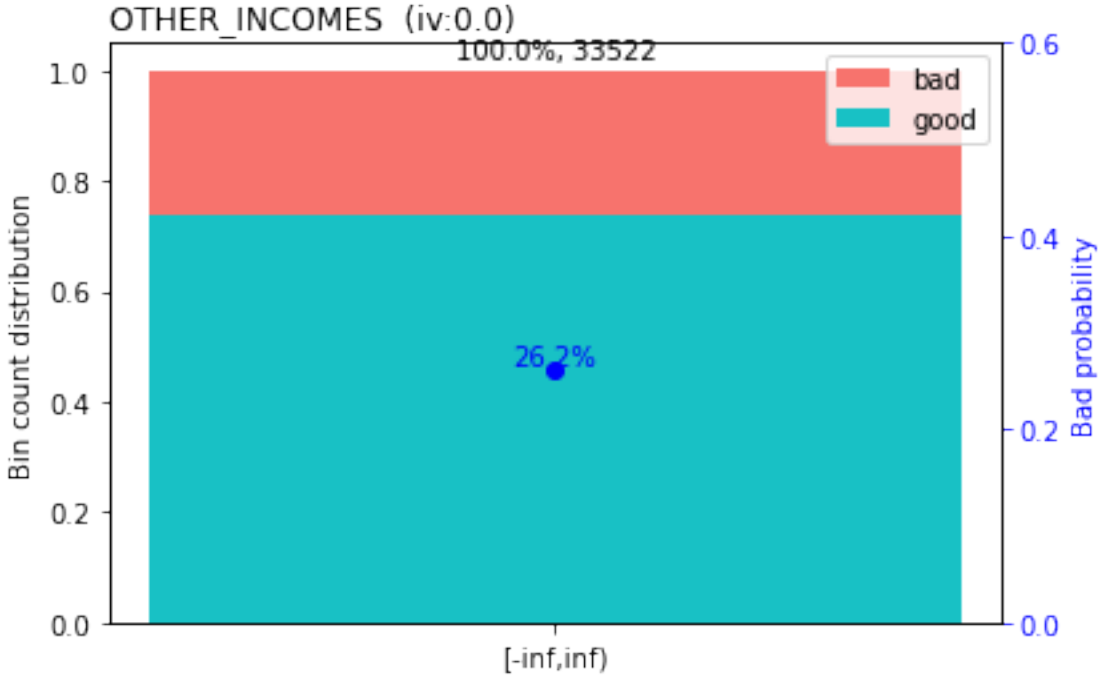

QUANT_CARS (iv:0.0003)
bad
good
66.8%, 22408
25.9%
26.7%
33.2%, 11114
Bin count distribution
Bad probability
0.0
0.2
0.4
0.6
0.8
[-inf,1.0)
[1.0,inf)

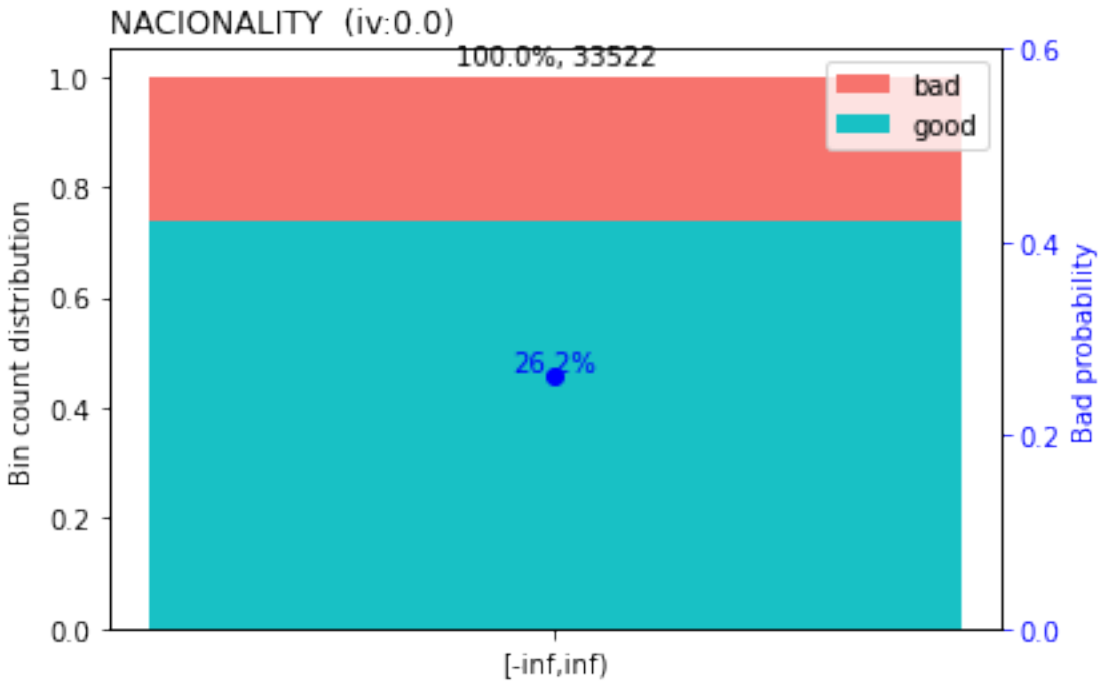

TOTAL_TYPE_OF_CARDS (iv:0.0008)
bad
good
84.4%, 28283
26.4%
24.9%
15.6%, 5239
Bin count distribution
Bad probability
0.0
0.2
0.4
0.6
0.8
1.0
[-inf,1.0)
[1.0,inf)

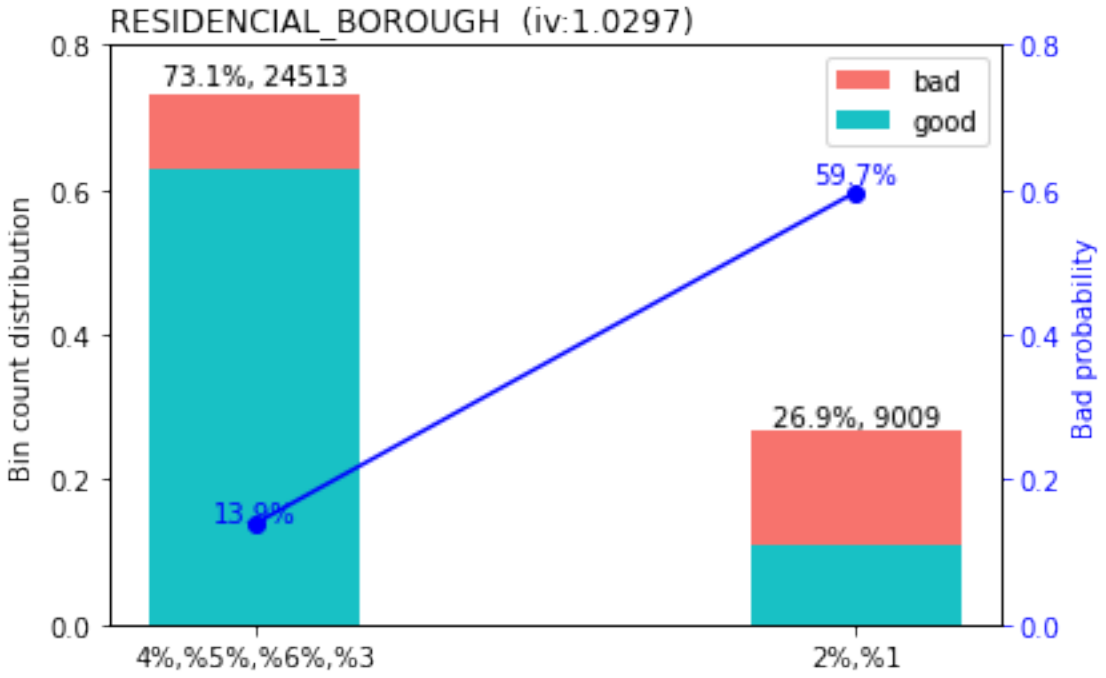

PAYMENT_DAY (iv:0.0323)
bad
good
Bin count distribution
Bad probability
58.9%, 19751
23.6%
27.4%
29.3%
33.5%
17.4%, 5836
9.6%, 3230
14.0%, 4705
[-inf,15.0)
[15.0,20.0)
[20.0,25.0)
[25.0,inf)

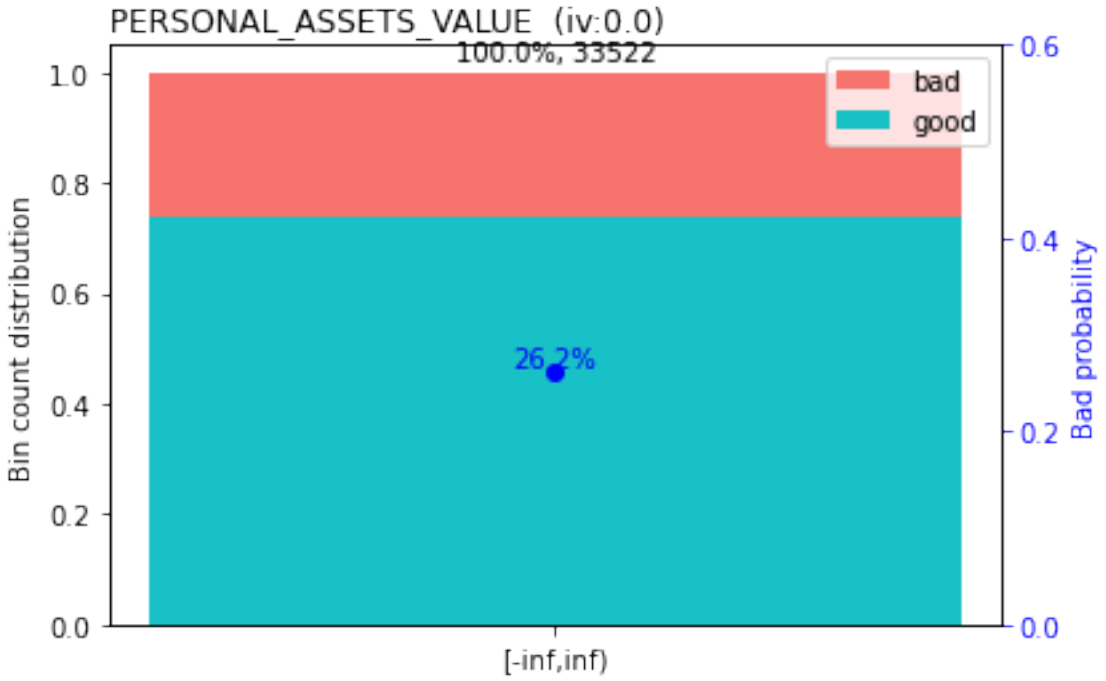

MARITAL_STATUS (iv:0.023)
bad
good
Bin count distribution
Bad probability
61.8%, 20729
23.9%
27.6%
30.4%
31.0%, 10383
7.2%, 2410
3
2
1

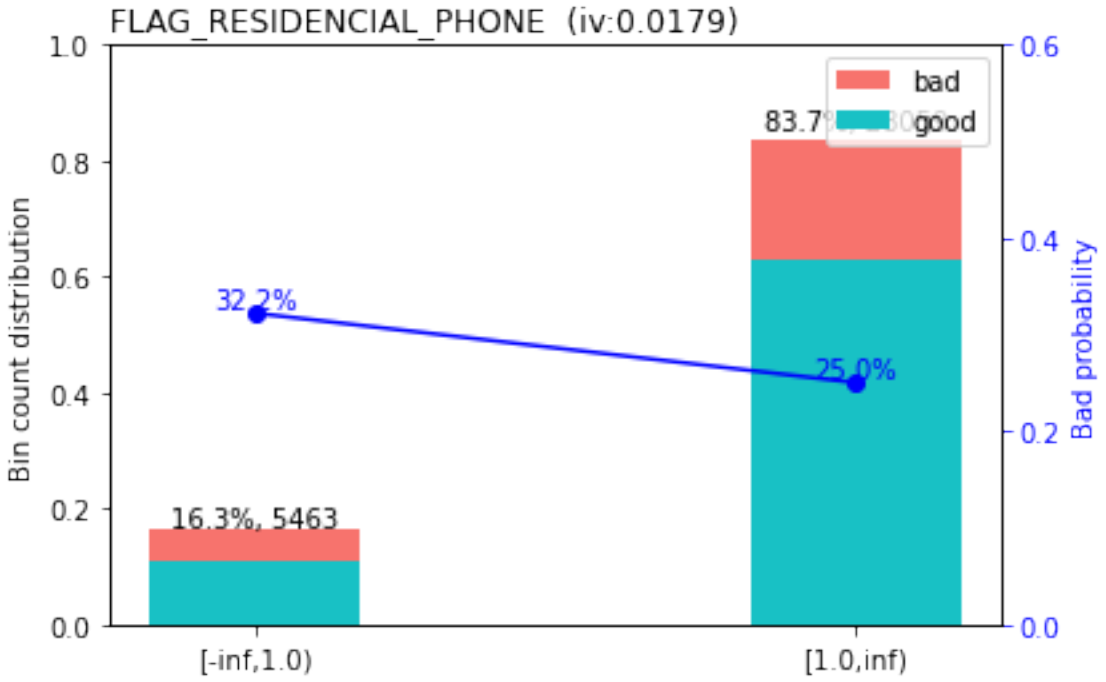

AGE (iv:0.0842)
bad
good
Bin count distribution
Bad probability
36.9%
32.1%
34.8%, 11649
27.5%
23.2%
19.3%, 6467
17.0%, 5695
18.5%
16.3%
21.8%
9.4%, 3144
8.0%, 2677
6.3%, 2108
5.3%, 1782
[-inf,23.0)
[23.0,32.0)
[32.0,46.0)
[46.0,55.0)
[55.0,62.0)
[62.0,70.0)
[70.0,inf)

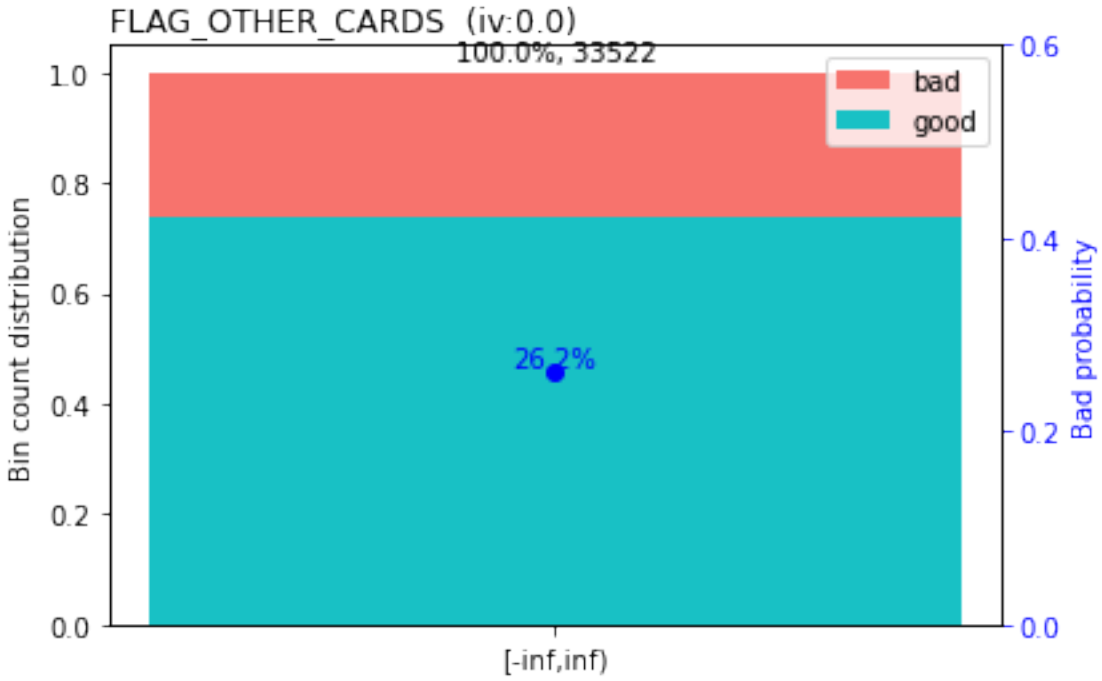

OTHER_INCOMES (iv:0.0)
100.0%, 33522
bad
good
Bin count distribution
Bad probability
26.2%
[-inf,inf)

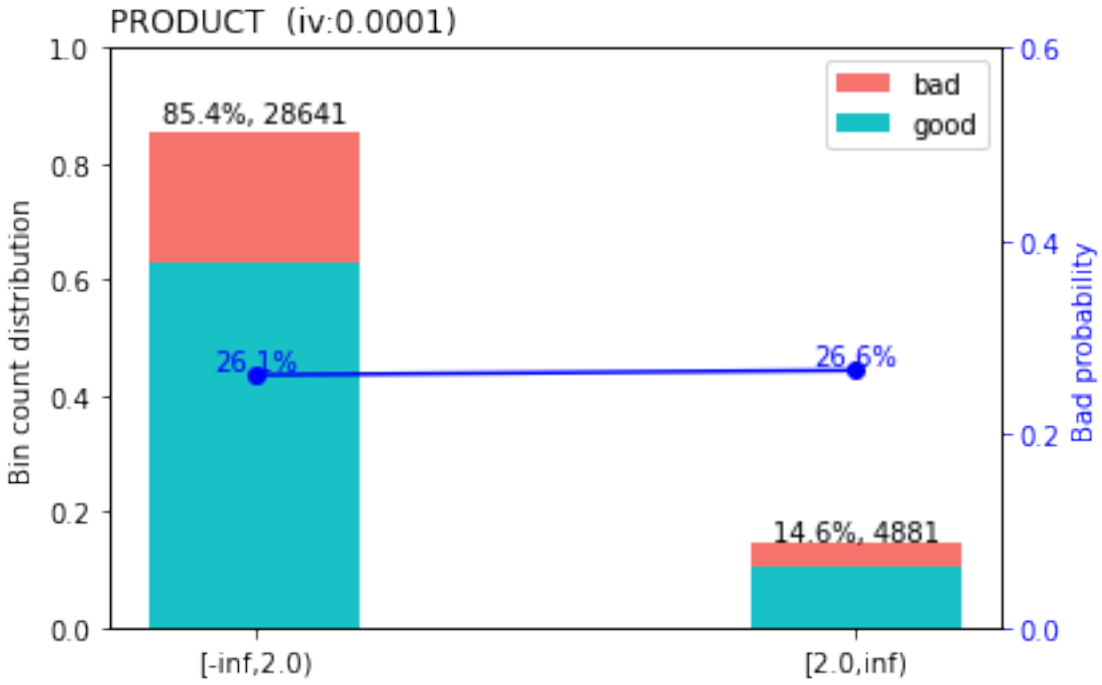

NACIONALITY (iv:0.0)
100.0%, 33522
bad
good
26.2%
Bin count distribution
Bad probability
[-inf,inf)

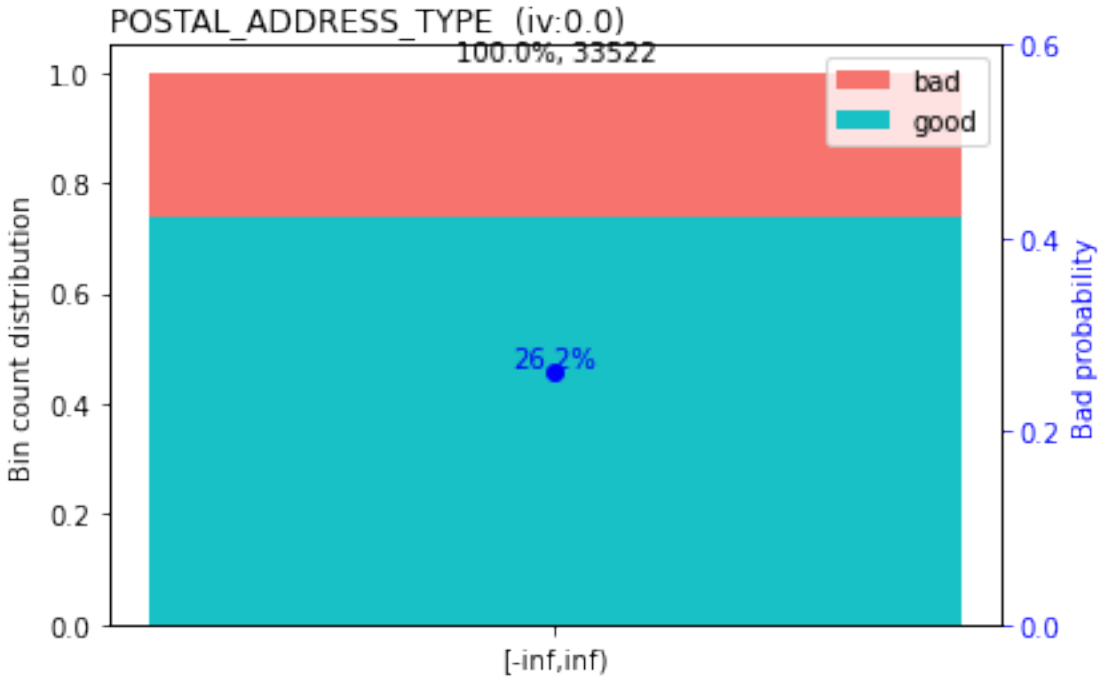

RESIDENCIAL_BOROUGH (iv:1.0297)
73.1%, 24513
bad
good
59.7%
26.9%, 9009
13.8%
Bin count distribution
Bad probability
4%,%5%,%6%,%3
2%,%1

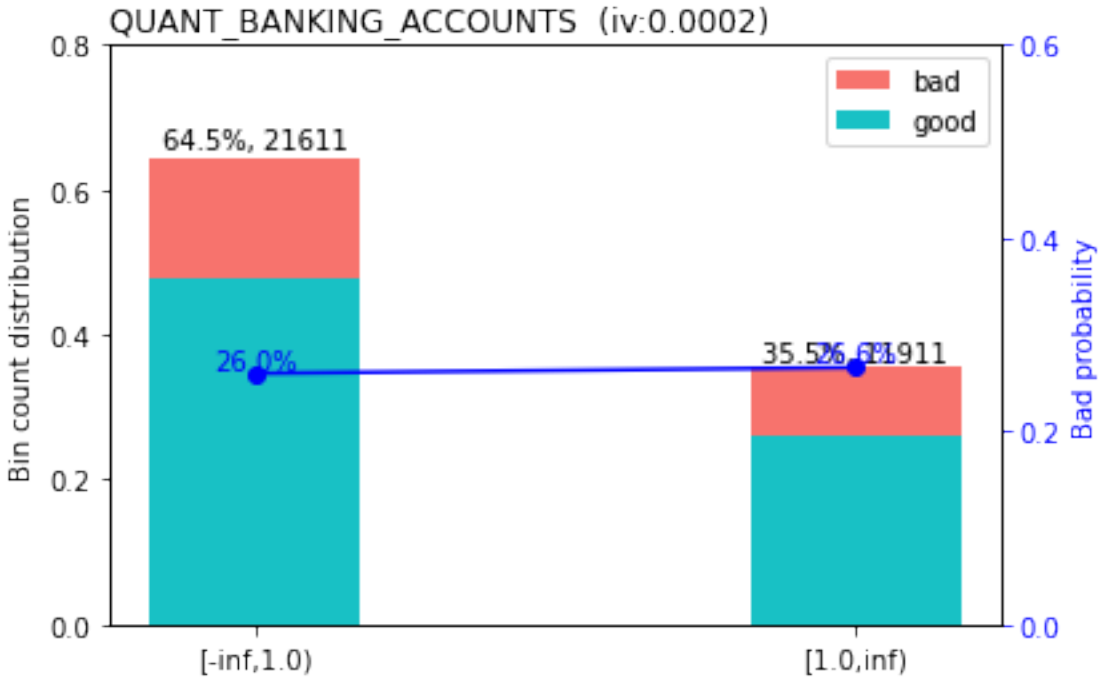

PERSONAL_ASSETS_VALUE (iv:0.0)
100.0%, 33522
bad
good
26.2%
Bin count distribution
Bad probability
[-inf,inf)

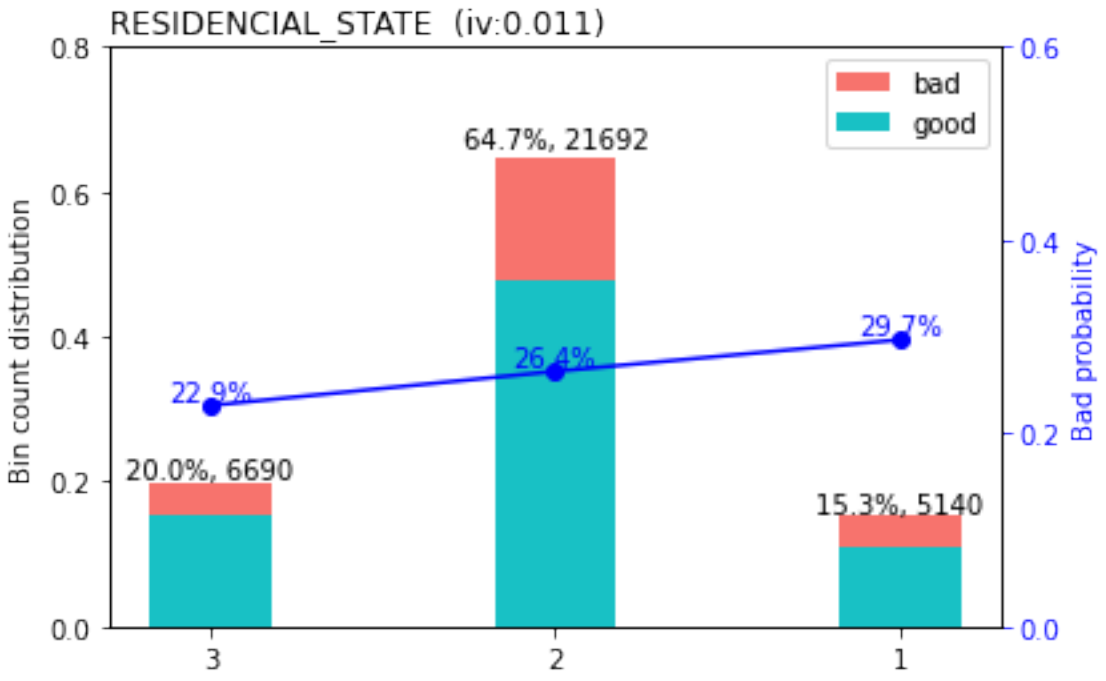

FLAG_RESIDENCIAL_PHONE (iv:0.0179)
bad
good
32.2%
25.0%
16.3%, 5463
Bin count distribution
Bad probability
[-inf,1.0)
[1.0,inf)

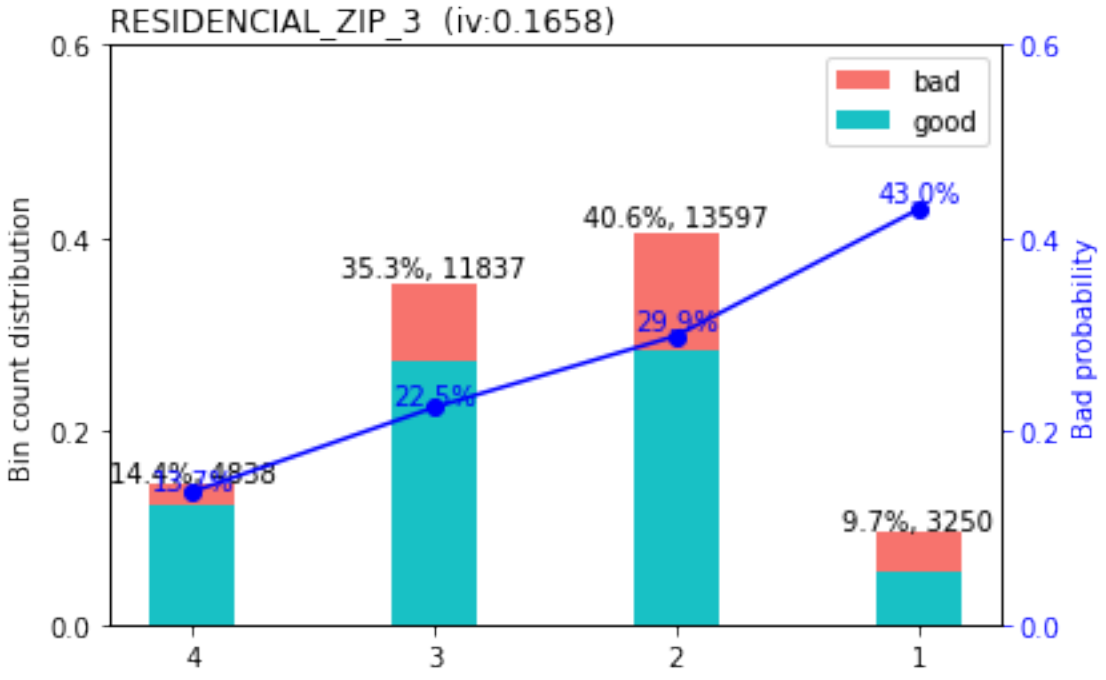

FLAG_OTHER_CARDS (iv:0.0)
100.0%, 33522
bad
good
26.2%
Bin count distribution
Bad probability
[-inf,inf)

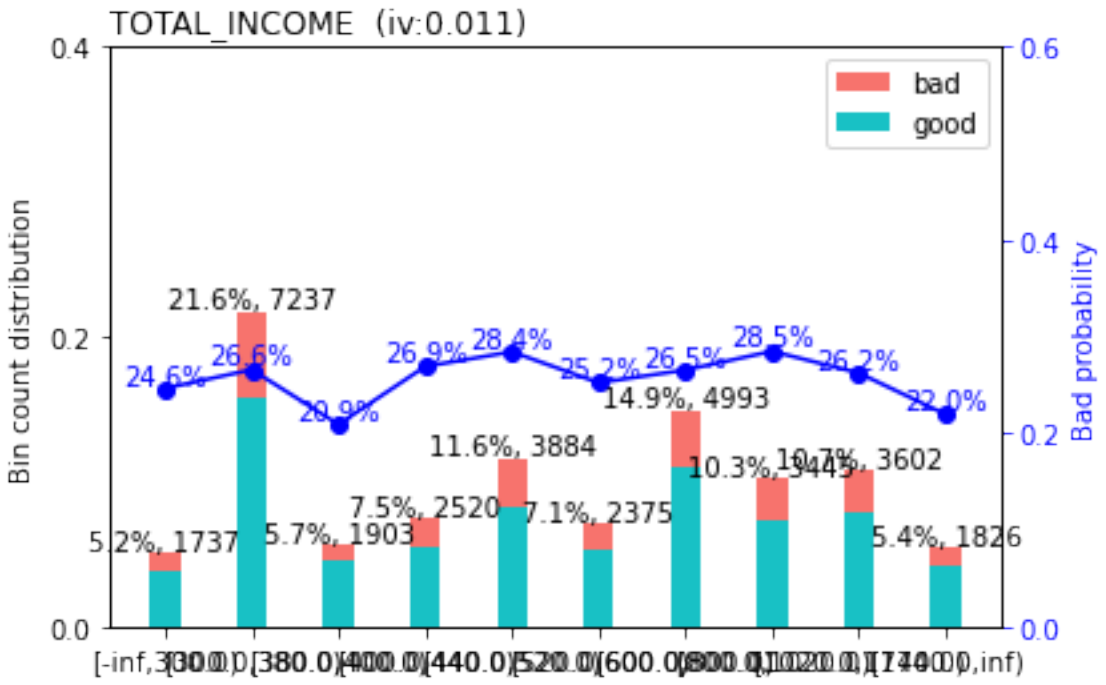

PRODUCT (iv:0.0001)
85.4%, 28641
bad
good
26.1%
26.6%
14.6%, 4881
Bin count distribution
Bad probability
[-inf,2.0)
[2.0,inf)

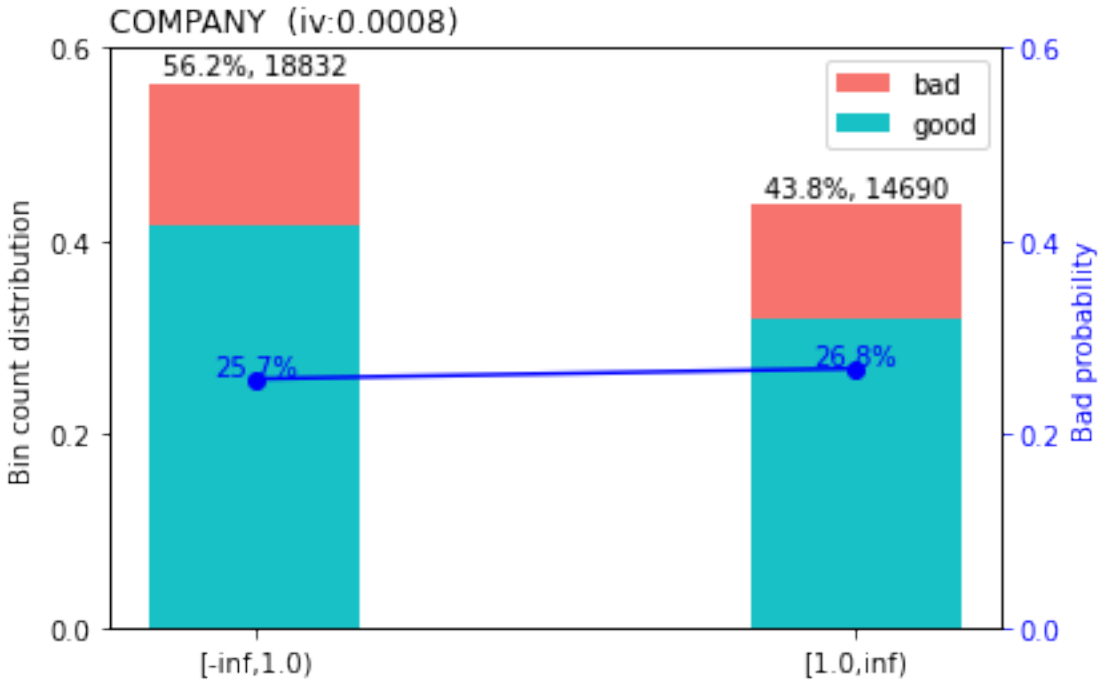

POSTAL_ADDRESS_TYPE (iv:0.0)
100.0%, 33522
bad
good
26.2%
Bin count distribution
Bad probability
[-inf,inf)

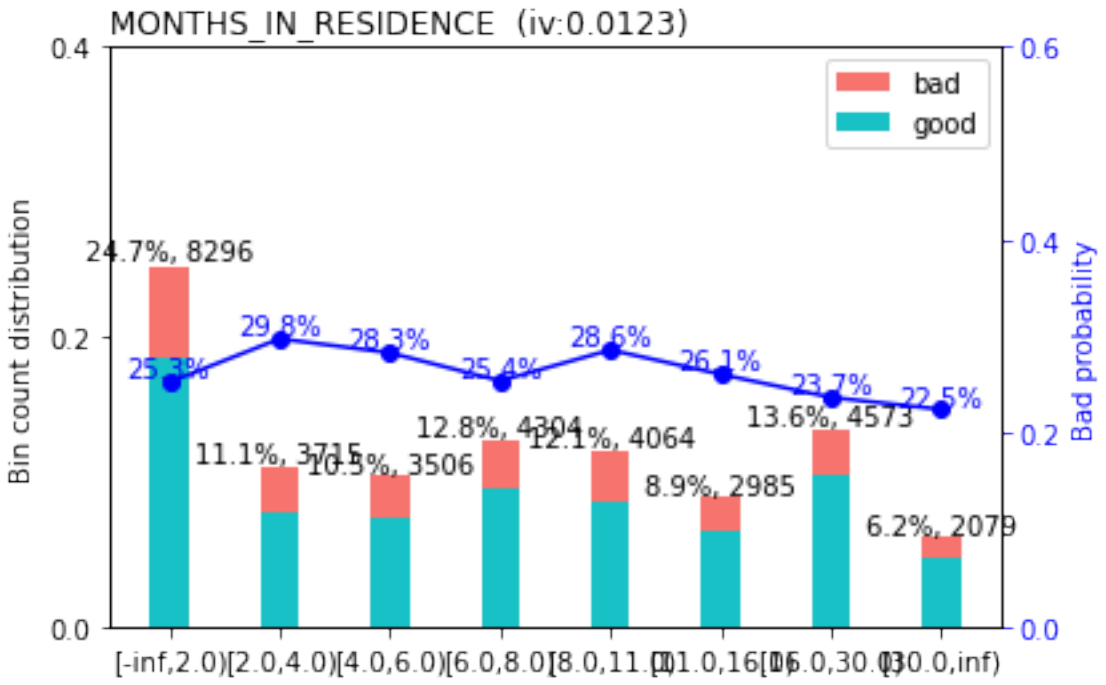

QUANT_BANKING_ACCOUNTS (iv:0.0002)
64.5%, 21611
bad
good
26.0%
Bin count distribution
Bad probability
[-inf,1.0)
[1.0,inf)

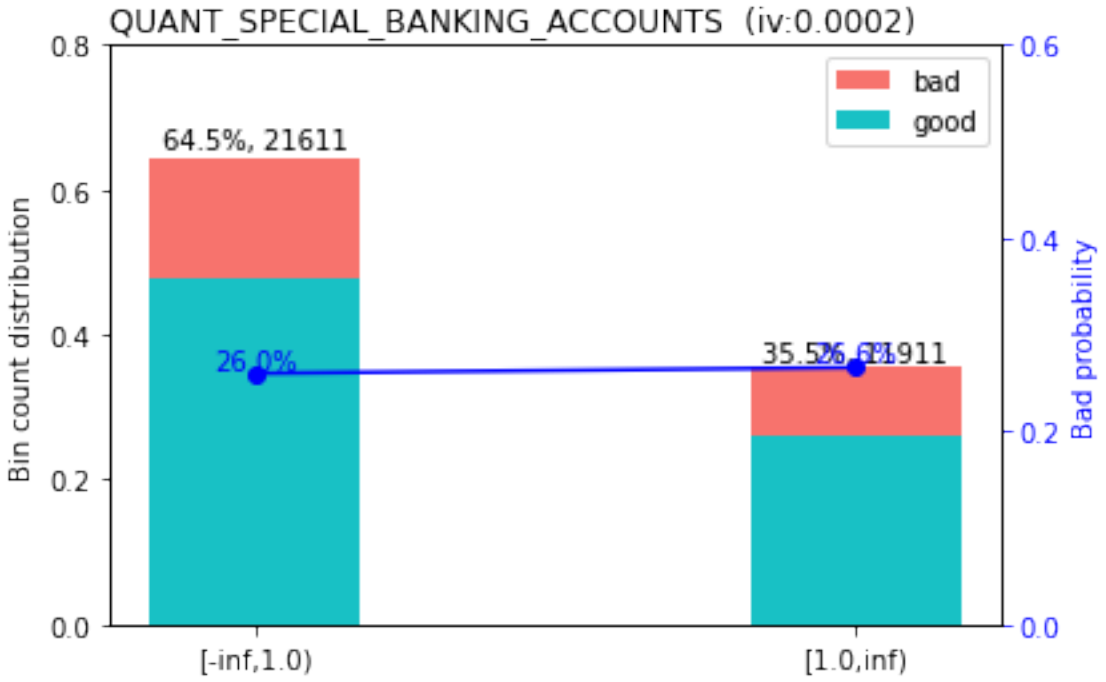

RESIDENCIAL_STATE (iv:0.011)
bad
good
64.7%, 21692
20.0%, 6690
15.3%, 5140
22.9%
26.4%
29.7%
Bin count distribution
Bad probability
0.0
0.2
0.4
0.6
0.8
3
2
1

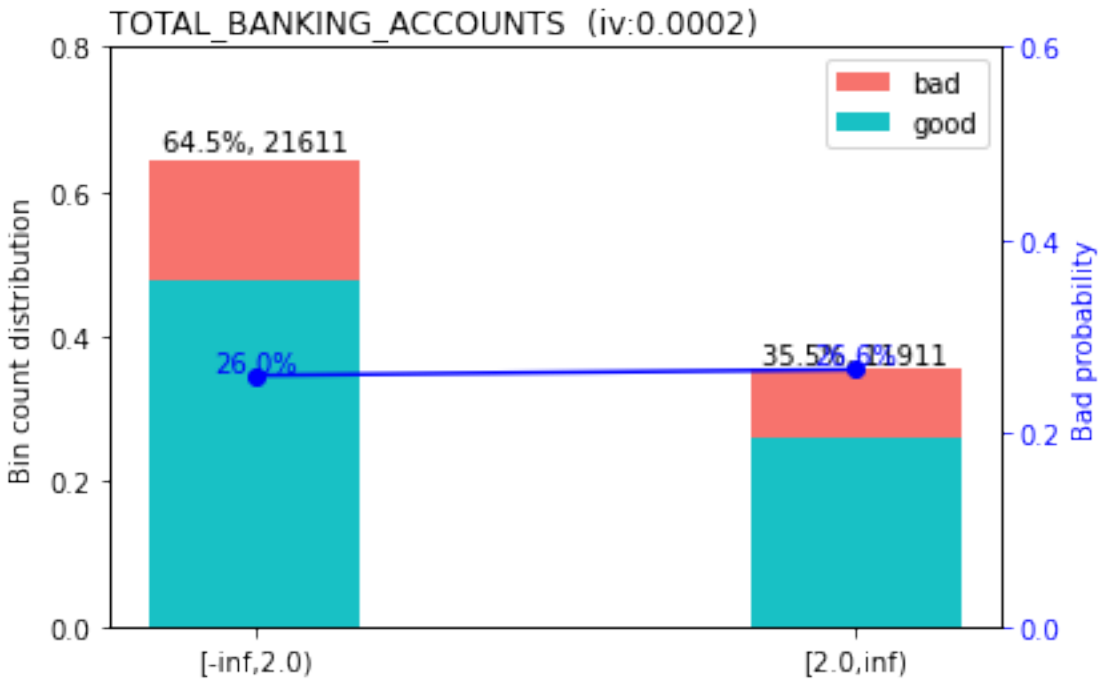

RESIDENCIAL_ZIP_3 (iv:0.1658)
bad
good
40.6%, 13597
35.3%, 11837
43.0%
29.9%
22.5%
9.7%, 3250
Bin count distribution
Bad probability
0.0
0.2
0.4
0.6
4
3
2
1


TOTAL_INCOME (iv:0.011)
bad
good
Bin count distribution
Bad probability
0.0
0.2
0.4
0.6
21.6%, 7237
24.6%
26.6%
20.9%
26.9%
28.4%
25.2%
26.5%
28.5%
26.2%
22.0%
14.9%, 4993
11.6%, 3884
10.3%, 3445
10.7%, 3602
7.5%, 2520
7.1%, 2375
5.2%, 1737
5.7%, 1903
5.4%, 1826
[-inf,380.0)
[380.0,440.0)
[600.0,inf)


COMPANY (iv:0.0008)
bad
good
Bin count distribution
Bad probability
0.0
0.2
0.4
0.6
56.2%, 18832
43.8%, 14690
25.7%
26.8%
[-inf,1.0)
[1.0,inf)

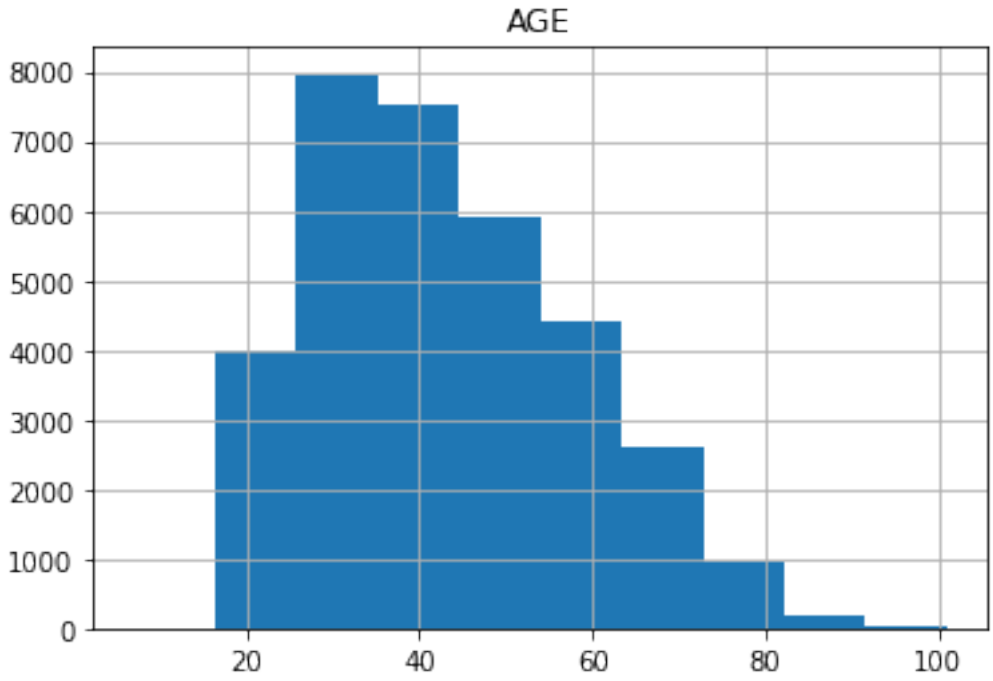

MONTHS_IN_RESIDENCE (iv:0.0123)
bad
good
Bin count distribution
Bad probability
24.7%, 8296
29.8%
28.3%
25.4%
28.6%
26.1%
23.7%
22.5%
11.1%, 3715
10.5%, 3506
12.8%, 4304
12.1%, 4064
8.9%, 2985
13.6%, 4573
6.2%, 2079
[-inf,2.0)
[2.0,4.0)
[4.0,6.0)
[6.0,8.0)
[8.0,11.0)
[11.0,16.0)
[16.0,30.0)
[30.0,inf)


QUANT_SPECIAL_BANKING_ACCOUNTS (iv:0.0002)
bad
good
Bin count distribution
Bad probability
64.5%, 21611
26.0%
35.5%
[-inf,1.0)
[1.0,inf)

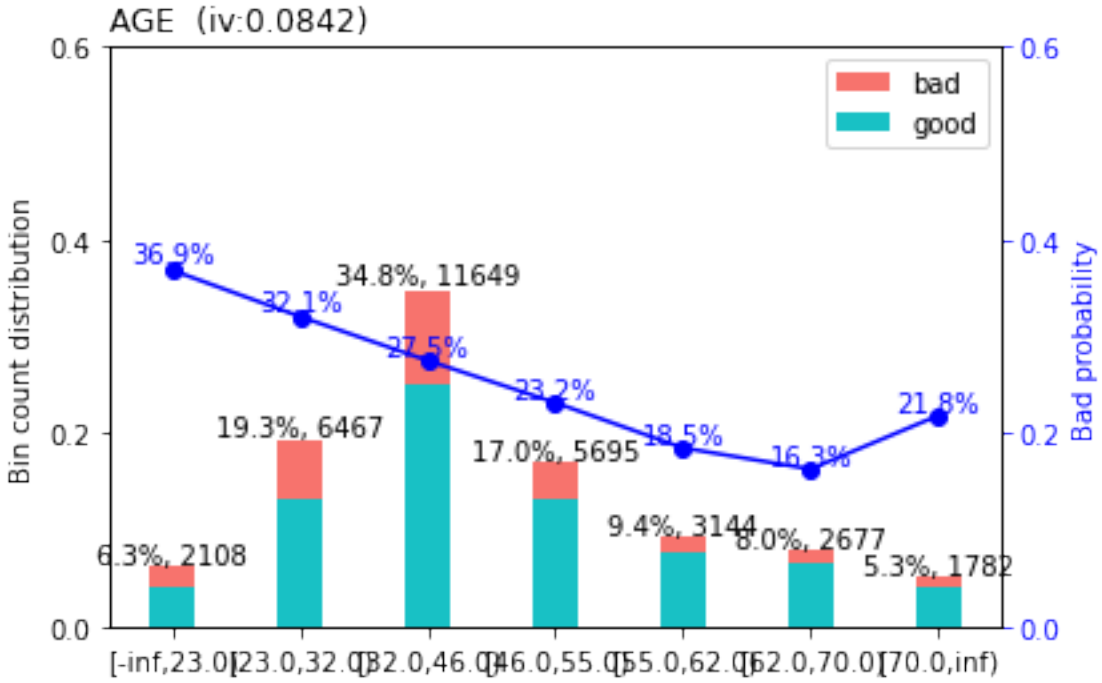


[57]:
```
breaks_adj = sc.woebin_adj(cleantrain, "TARGET_LABEL_BAD=1", bins, adj_all_var␣
↪= False)
```

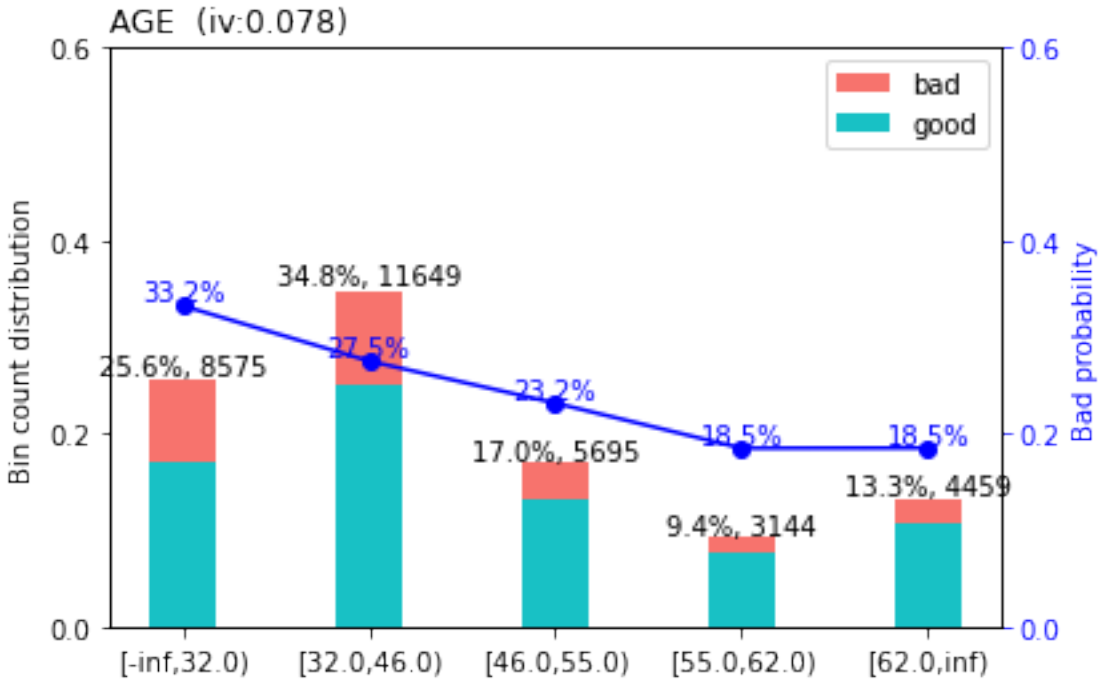
```
-------- 1/9 AGE --------
>>> dt[AGE].describe():
count    33522.000000
mean        42.986099
std         14.917561
min          7.000000
25%         31.000000
50%         41.000000
75%         53.000000
max        101.000000
Name: AGE, dtype: float64
```



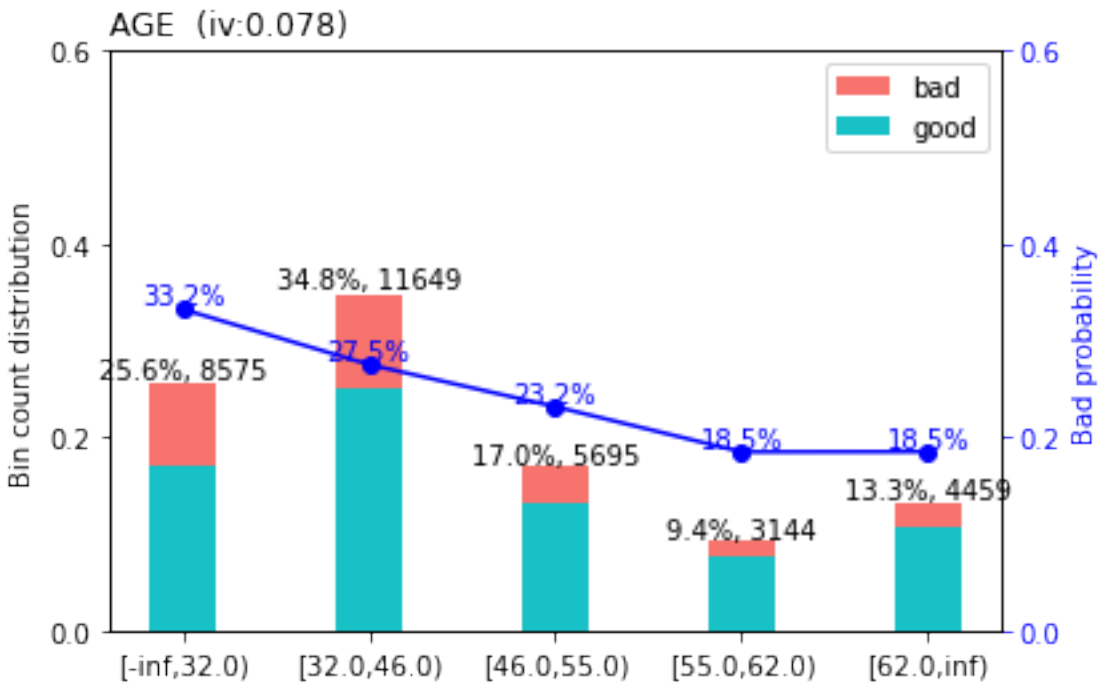

```
>>> Current breaks:
23.0,32.0,46.0,55.0,62.0,70.0
```



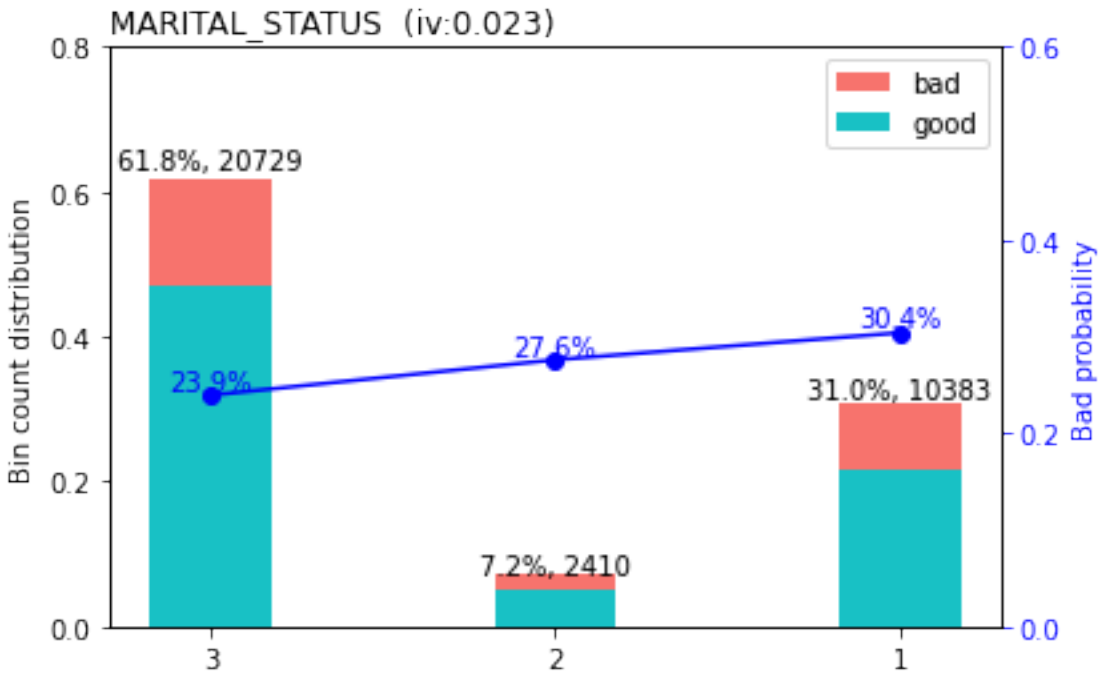
```
>>> Adjust breaks for (1/9) AGE?
1: next
2: yes
3: back
Selection: 2
>>> Enter modified breaks: 32,46,55,62
[INFO] creating woe binning …
>>> Current breaks:
32.0, 62.0, 46.0, 55.0
```



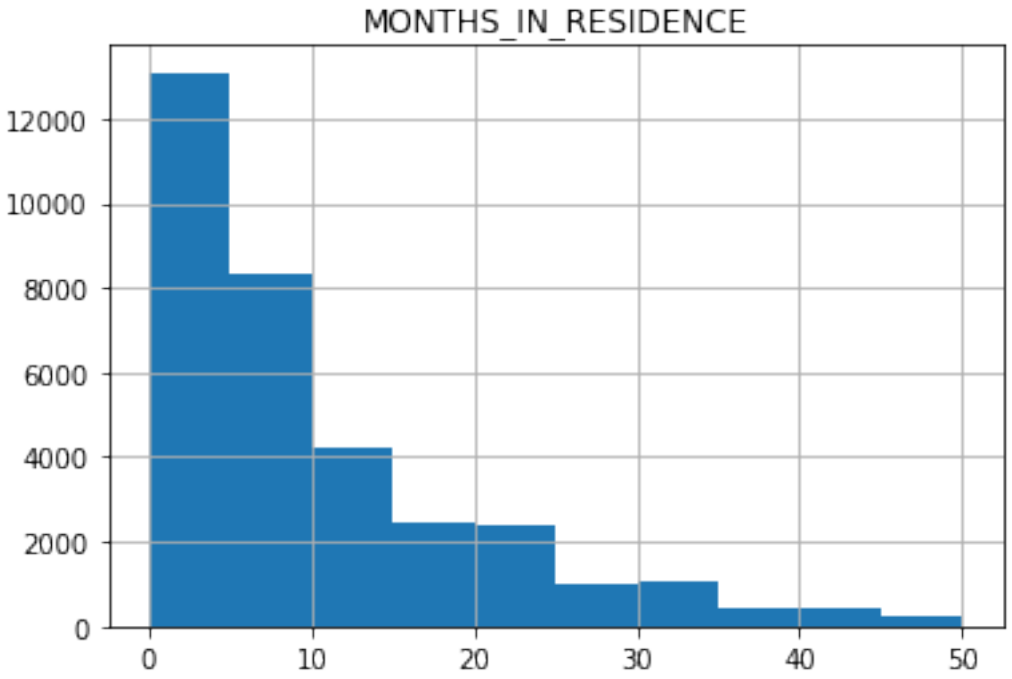

```
>>> Adjust breaks for (1/9) AGE?
1: next
2: yes
3: back
Selection: 2
>>> Enter modified breaks: 32,46,55,62
[INFO] creating woe binning …
>>> Current breaks:
32.0, 62.0, 46.0, 55.0
```



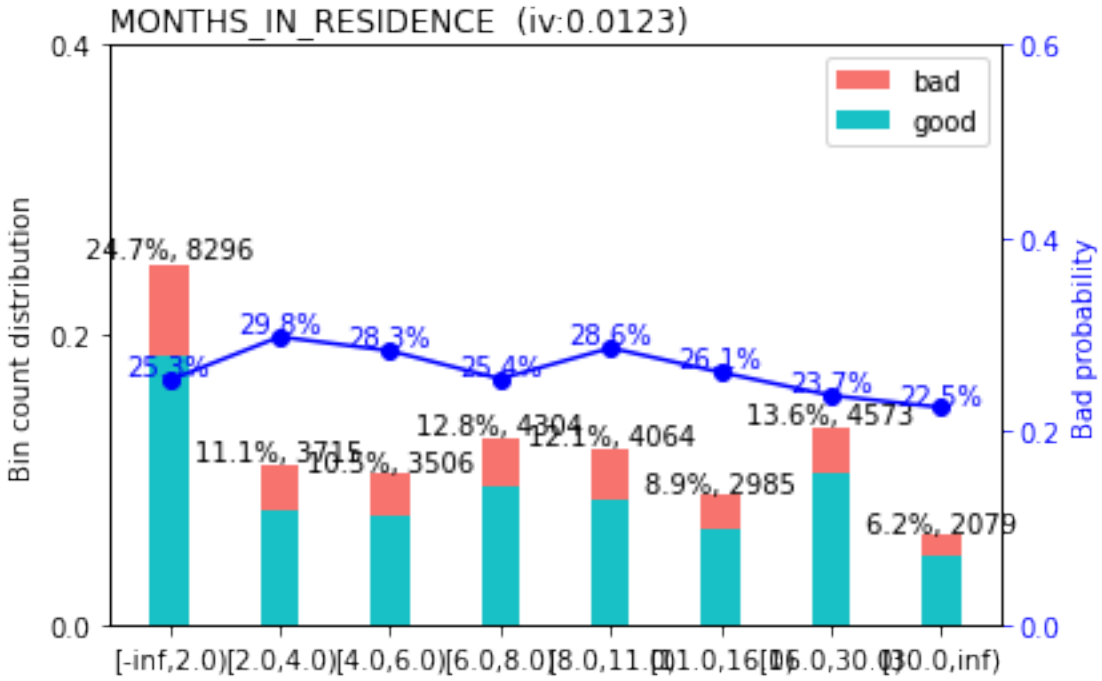

```
>>> Adjust breaks for (1/9) AGE?
1: next
2: yes
3: back
Selection: 1
-------- 2/9 MARITAL_STATUS --------
>>> dt[MARITAL_STATUS].describe():
count     33522
unique        3
top           3
freq      20729
Name: MARITAL_STATUS, dtype: object

>>> dt[MARITAL_STATUS].value_counts():
3    20729
1    10383
2     2410
Name: MARITAL_STATUS, dtype: int64

>>> Current breaks:
'3','2','1'
```



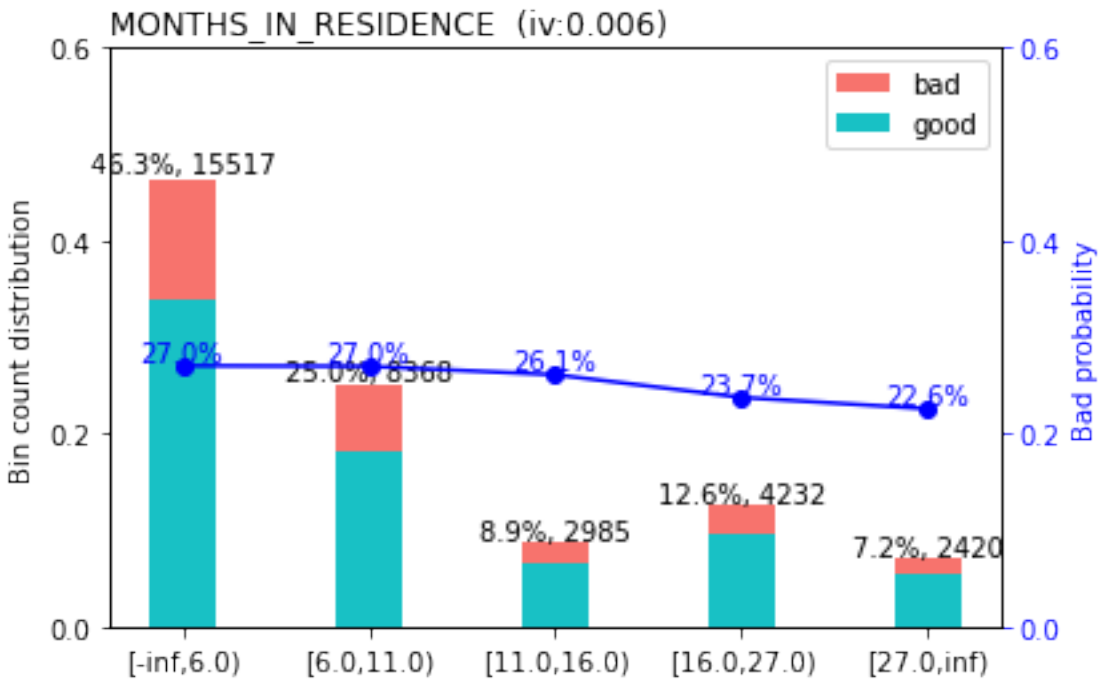

```
>>> Adjust breaks for (2/9) MARITAL_STATUS?
1: next
2: yes
3: back
Selection: 1
-------- 3/9 MONTHS_IN_RESIDENCE --------
>>> dt[MONTHS_IN_RESIDENCE].describe():
count    33522.000000
mean         9.108138
std          9.704732
min          0.000000
25%          2.000000
50%          6.000000
75%         13.000000
max         50.000000
Name: MONTHS_IN_RESIDENCE, dtype: float64
```




```
>>> Current breaks:
2.0,4.0,6.0,8.0,11.0,16.0,30.0
```

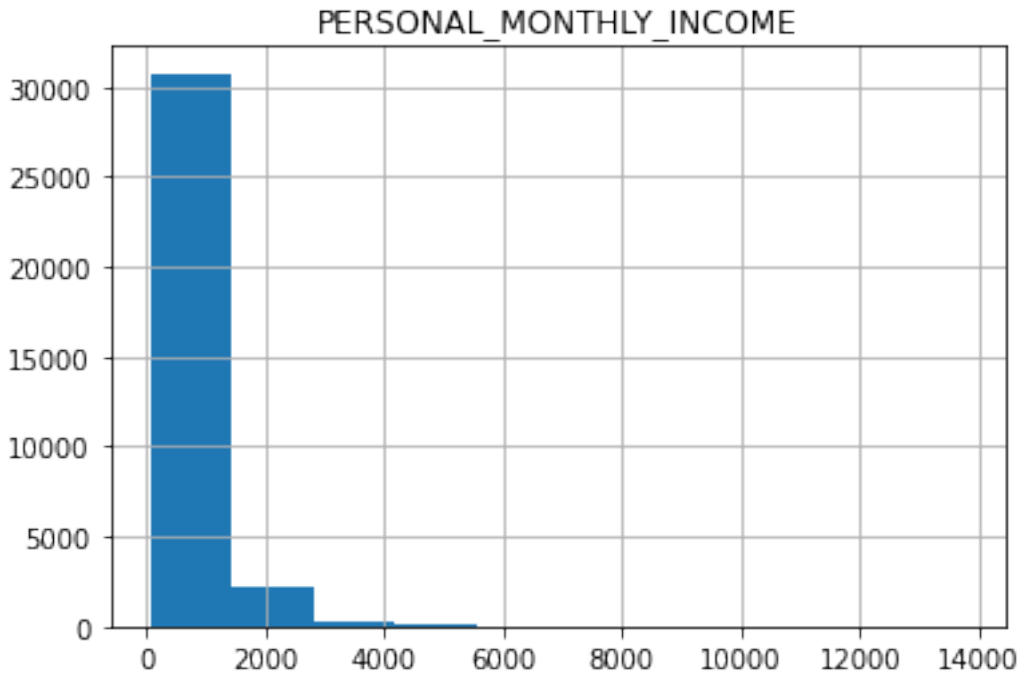


```
>>> Adjust breaks for (3/9) MONTHS_IN_RESIDENCE?
1: next
2: yes
3: back
Selection: 2
>>> Enter modified breaks: 6,11,16,27
[INFO] creating woe binning …
>>> Current breaks:
16.0, 11.0, 27.0, 6.0
```

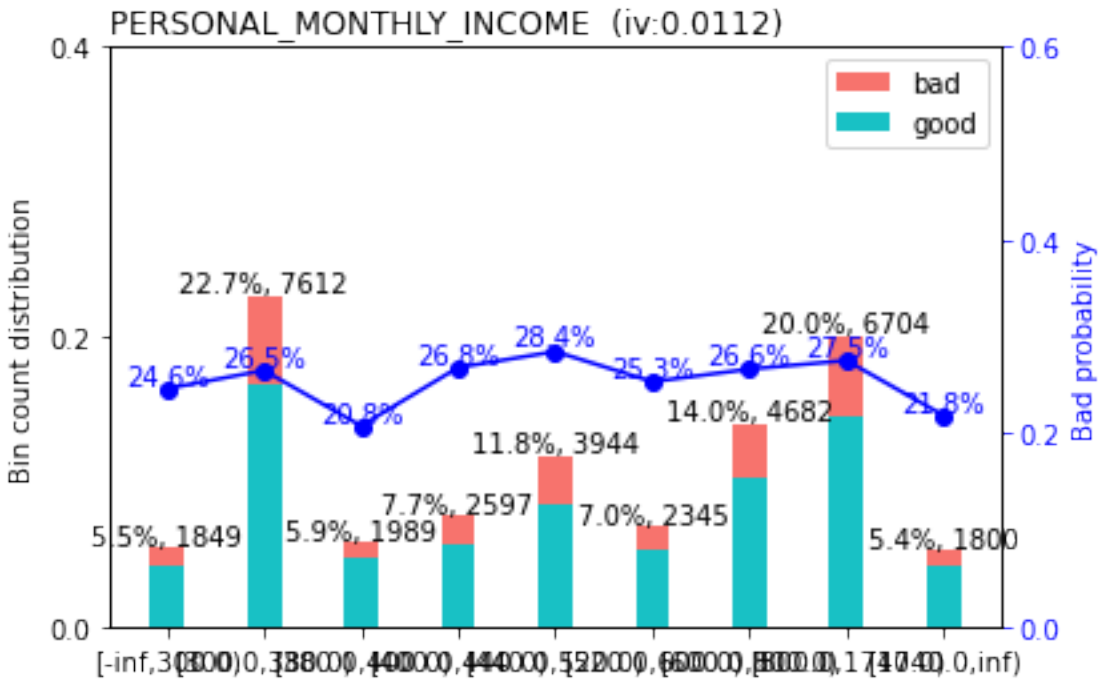



```
>>> Adjust breaks for (3/9) MONTHS_IN_RESIDENCE?
1: next
2: yes
3: back
Selection: 1
-------- 4/9 PERSONAL_MONTHLY_INCOME --------
>>> dt[PERSONAL_MONTHLY_INCOME].describe():
count    33522.000000
mean       709.721933
std        689.427599
min         69.000000
25%        358.000000
50%        500.000000
```

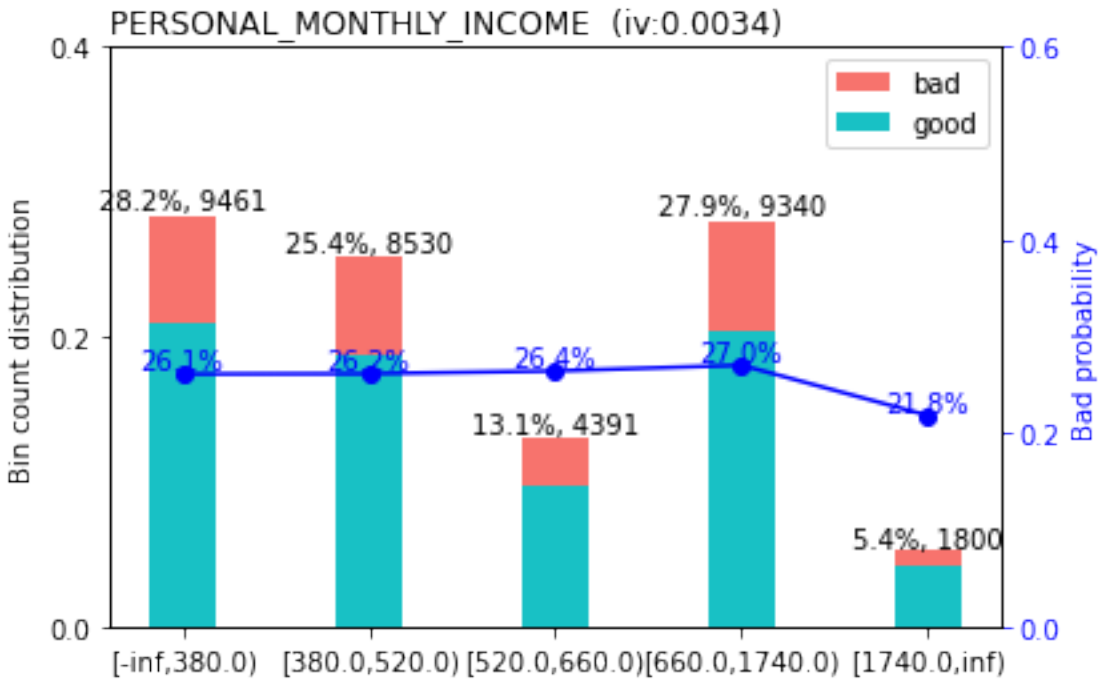
```
75%         800.000000
max       13757.000000
Name: PERSONAL_MONTHLY_INCOME, dtype: float64
```




```
>>> Current breaks:
300.0,380.0,400.0,440.0,520.0,600.0,800.0,1740.0
```

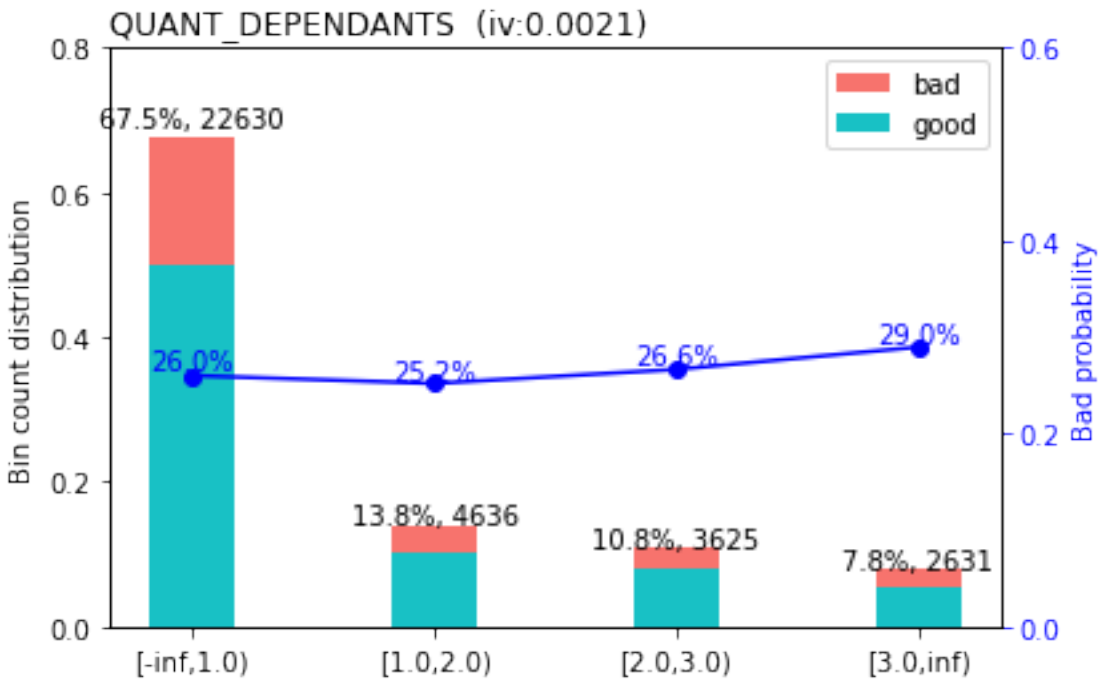



```
>>> Adjust breaks for (4/9) PERSONAL_MONTHLY_INCOME?
1: next
2: yes
3: back
Selection: 2
>>> Enter modified breaks: 380,520,660,1740
[INFO] creating woe binning …
>>> Current breaks:
380.0, 1740.0, 660.0, 520.0
```



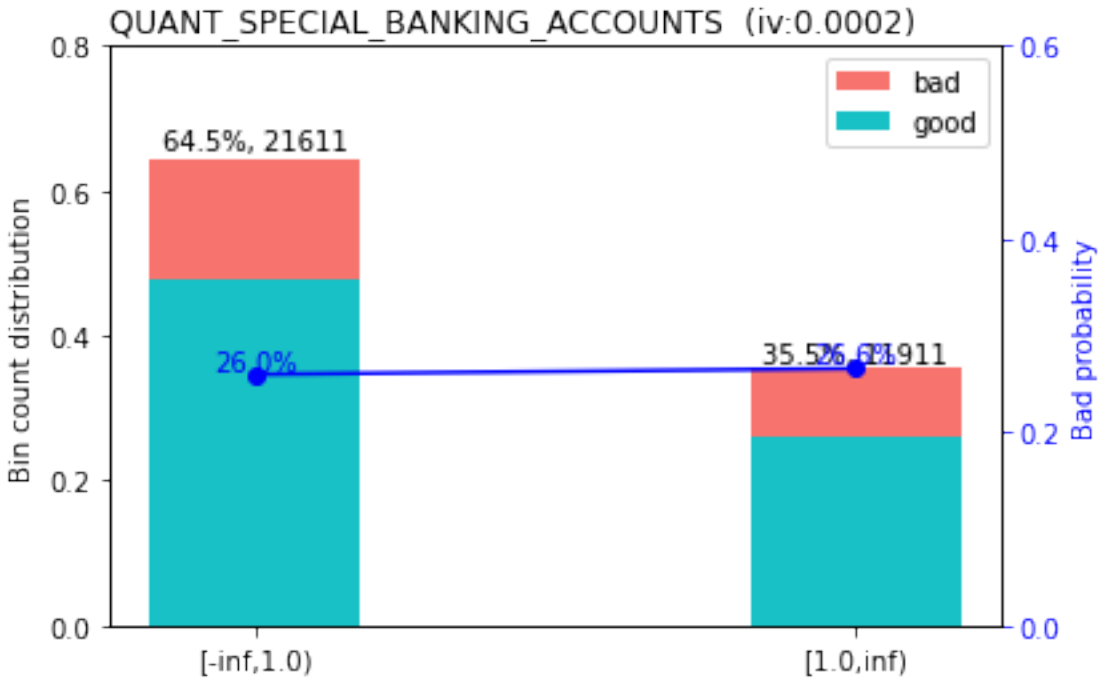

```
>>> Adjust breaks for (4/9) PERSONAL_MONTHLY_INCOME?
1: next
2: yes
3: back
Selection: 1
-------- 5/9 QUANT_DEPENDANTS --------
>>> dt[QUANT_DEPENDANTS].describe():
count    33522.000000
mean         0.631824
std          1.096871
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          6.000000
Name: QUANT_DEPENDANTS, dtype: float64

>>> dt[QUANT_DEPENDANTS].value_counts():
0    22630
1     4636
2     3625
3     1675
4      611
5      245
6      100
```

```
Name: QUANT_DEPENDANTS, dtype: int64

>>> Current breaks:
1.0,2.0,3.0
```



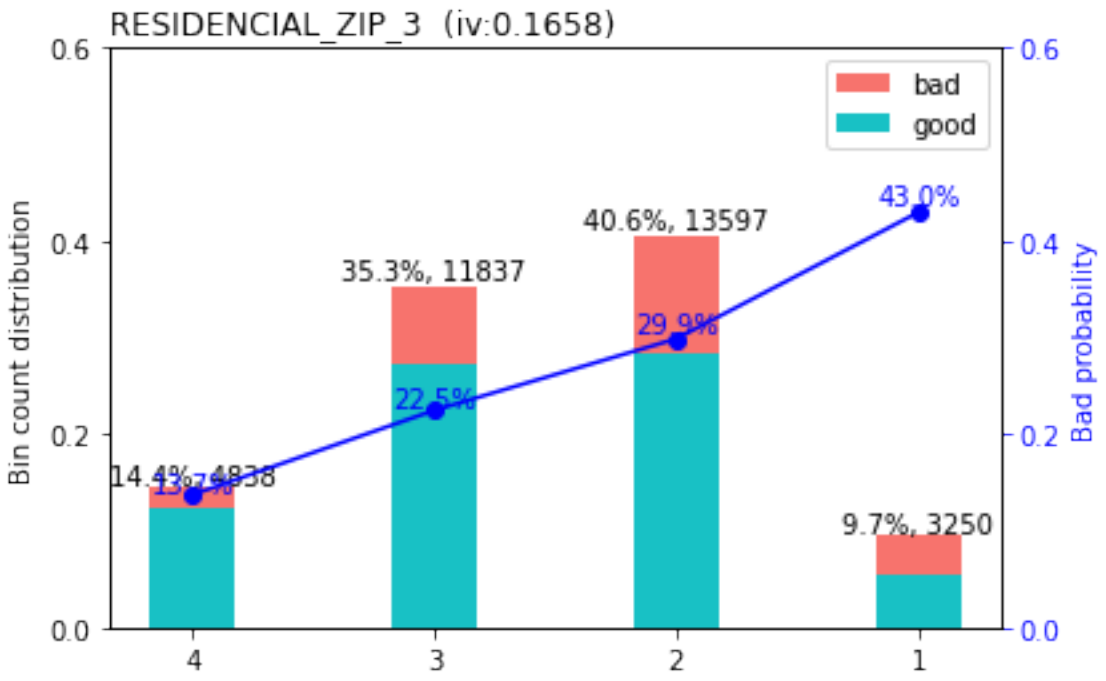

```
>>> Adjust breaks for (5/9) QUANT_DEPENDANTS?
1: next
2: yes
3: back
Selection: 1
-------- 6/9 QUANT_SPECIAL_BANKING_ACCOUNTS --------
>>> dt[QUANT_SPECIAL_BANKING_ACCOUNTS].describe():
count    33522.000000
mean         0.355587
std          0.479259
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          2.000000
Name: QUANT_SPECIAL_BANKING_ACCOUNTS, dtype: float64

>>> dt[QUANT_SPECIAL_BANKING_ACCOUNTS].value_counts():
0.0    21611
```

```
1.0    11902
2.0        9
Name: QUANT_SPECIAL_BANKING_ACCOUNTS, dtype: int64

>>> Current breaks:
1.0
```



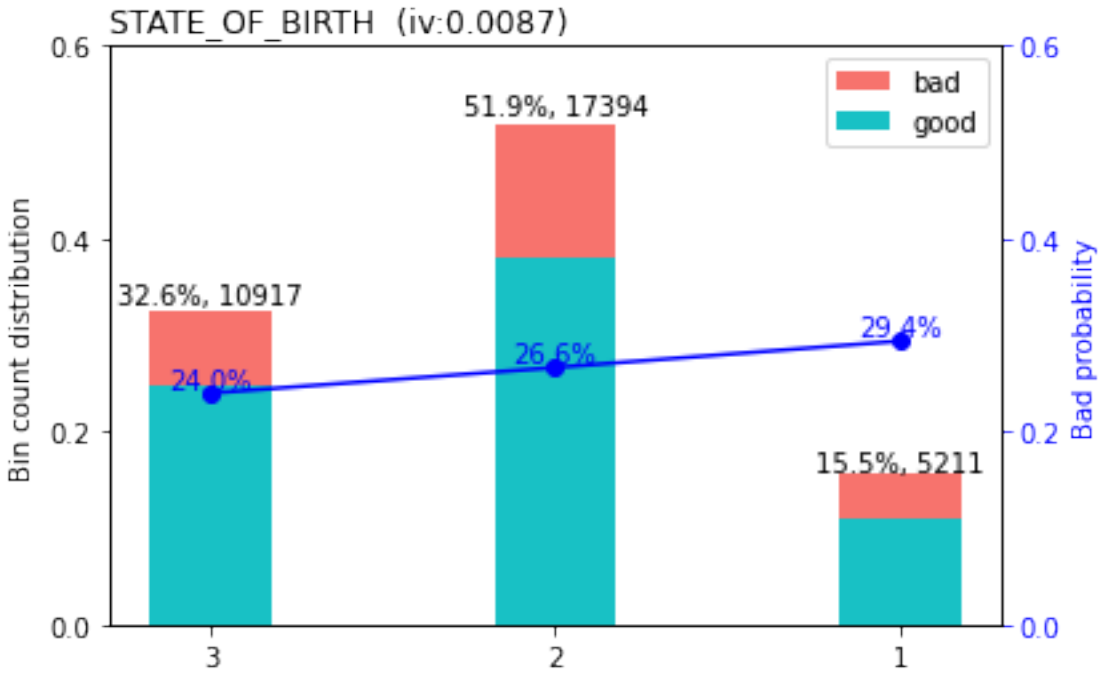

```
>>> Adjust breaks for (6/9) QUANT_SPECIAL_BANKING_ACCOUNTS?
1: next
2: yes
3: back
Selection: 1
-------- 7/9 RESIDENCIAL_ZIP_3 --------
>>> dt[RESIDENCIAL_ZIP_3].describe():
count     33522
unique        4
top           2
freq      13597
Name: RESIDENCIAL_ZIP_3, dtype: object

>>> dt[RESIDENCIAL_ZIP_3].value_counts():
2    13597
3    11837
4     4838
```

```
1     3250
Name: RESIDENCIAL_ZIP_3, dtype: int64

>>> Current breaks:
'4','3','2','1'
```

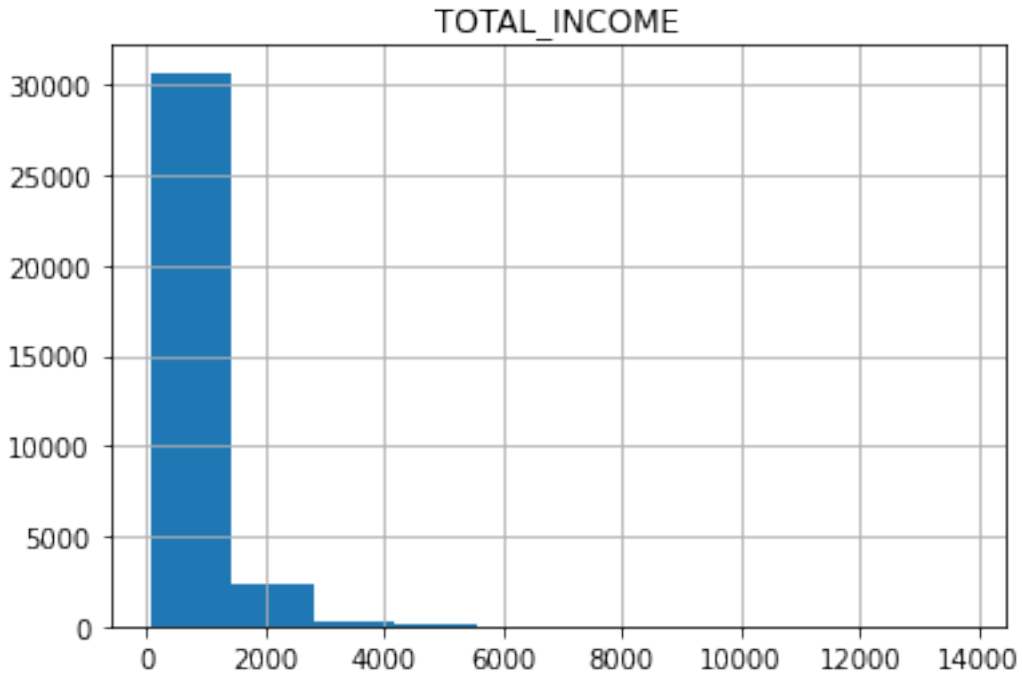



```
>>> Adjust breaks for (7/9) RESIDENCIAL_ZIP_3?
1: next
2: yes
3: back
Selection: 1
-------- 8/9 STATE_OF_BIRTH --------
>>> dt[STATE_OF_BIRTH].describe():
count     33522
unique        3
top           2
freq      17394
Name: STATE_OF_BIRTH, dtype: object

>>> dt[STATE_OF_BIRTH].value_counts():
2    17394
3    10917
1     5211
Name: STATE_OF_BIRTH, dtype: int64
```

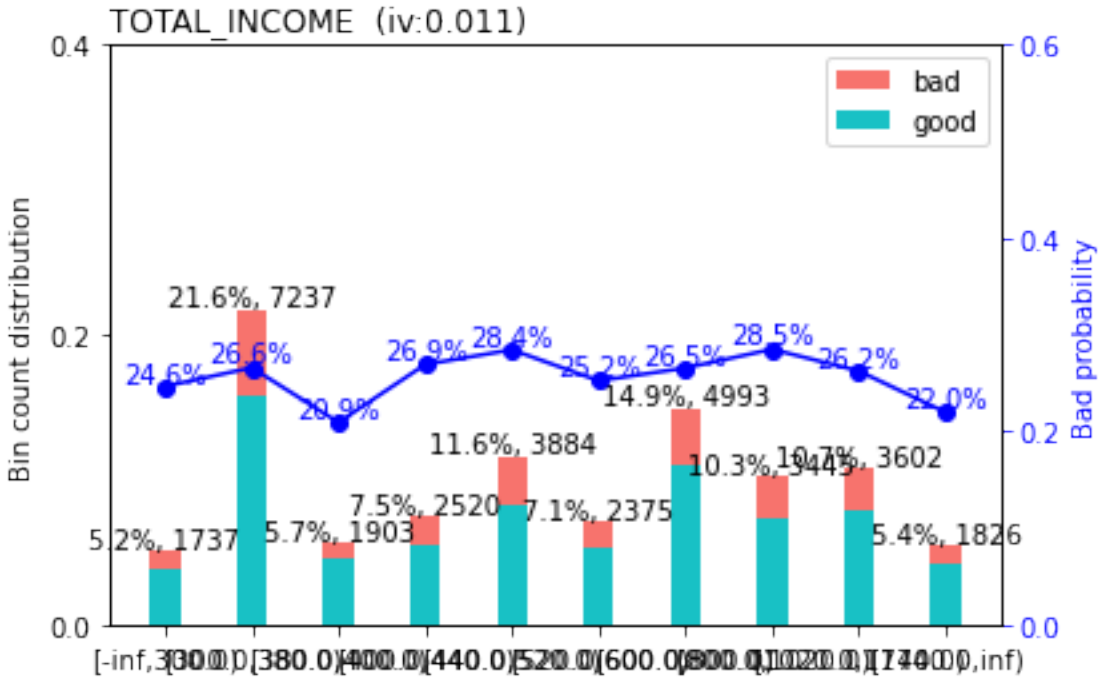
```
>>> Current breaks:
'3','2','1'
```



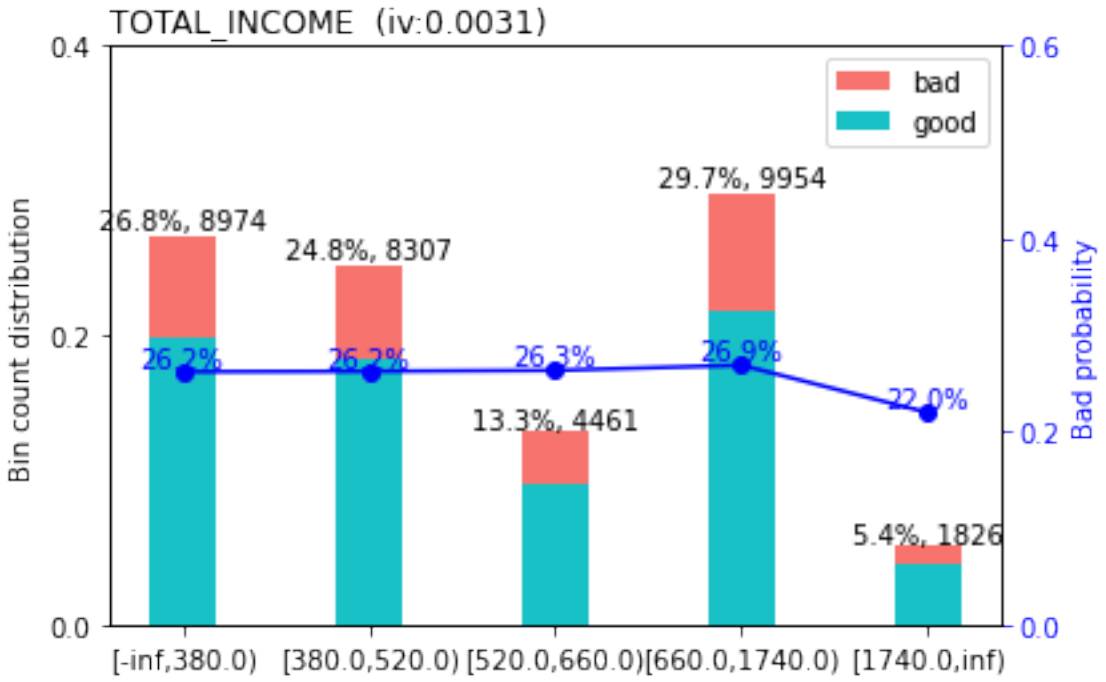

```
>>> Adjust breaks for (8/9) STATE_OF_BIRTH?
1: next
2: yes
3: back
Selection: 1
-------- 9/9 TOTAL_INCOME --------
>>> dt[TOTAL_INCOME].describe():
count    33522.000000
mean       721.482444
std        689.720611
min         69.000000
25%        368.000000
50%        500.000000
75%        800.000000
max      13757.000000
Name: TOTAL_INCOME, dtype: float64
```




```
>>> Current breaks:
300.0,380.0,400.0,440.0,520.0,600.0,800.0,1020.0,1740.0
```



```
>>> Adjust breaks for (9/9) TOTAL_INCOME?
1: next
2: yes
3: back
Selection: 2
>>> Enter modified breaks: 380,520,660,1740
[INFO] creating woe binning …
>>> Current breaks:
380.0, 1740.0, 660.0, 520.0
```




```
>>> Adjust breaks for (9/9) TOTAL_INCOME?
1: next
2: yes
3: back
Selection: 1
```

[66]:
```
bins_adj = sc.woebin(cleantrain, y="TARGET_LABEL_BAD=1",␣
↪breaks_list=breaks_adj) # Apply new cuts
train_woe = sc.woebin_ply(cleantrain, bins_adj) # Calculate WoE dataset (train)
test_woe = sc.woebin_ply(cleantest, bins_adj) # Calculate WoE dataset (test)
```

```
[INFO] creating woe binning …
[INFO] converting into woe values …
```

```
[INFO] converting into woe values …
```

[340]:
```
train_woe.to_csv("train_woe.csv", index = False)
test_woe.to_csv("test_woe.csv", index = False)
```

0.0.7 Information value filter

[67]:
```
sc.iv(train_woe, 'TARGET_LABEL_BAD=1')
```

[67]:

	variable	info_value
28	RESIDENCIAL_BOROUGH_woe	1.029684
15	RESIDENCIAL_CITY_woe	0.364442
32	RESIDENCIAL_ZIP_3_woe	0.165772
18	AGE_woe	0.078019
12	PAYMENT_DAY_woe	0.032268
19	MARITAL_STATUS_woe	0.023044
10	FLAG_RESIDENCIAL_PHONE_woe	0.017929
3	RESIDENCIAL_STATE_woe	0.011005
25	STATE_OF_BIRTH_woe	0.008732
14	MONTHS_IN_RESIDENCE_woe	0.006032
16	RESIDENCE_TYPE_woe	0.003936
6	PERSONAL_MONTHLY_INCOME_woe	0.003447
23	TOTAL_INCOME_woe	0.003141
21	QUANT_DEPENDANTS_woe	0.002054
2	FLAG_MASTERCARD_woe	0.002036
26	FLAG_PROFESSIONAL_PHONE_woe	0.001410
17	SEX_woe	0.001336
31	TOTAL_TYPE_OF_CARDS_woe	0.000809
30	COMPANY_woe	0.000757
9	QUANT_CARS_woe	0.000340
24	FLAG_EMAIL_woe	0.000232
11	TOTAL_BANKING_ACCOUNTS_woe	0.000225
29	QUANT_SPECIAL_BANKING_ACCOUNTS_woe	0.000225
20	QUANT_BANKING_ACCOUNTS_woe	0.000225
4	FLAG_VISA_woe	0.000181
0	PRODUCT_woe	0.000078
5	FLAG_OTHER_CARDS_woe	0.000000
27	NACIONALITY_woe	0.000000
1	OTHER_INCOMES_woe	0.000000
7	FLAG_DINERS_woe	0.000000
8	POSTAL_ADDRESS_TYPE_woe	0.000000
13	PERSONAL_ASSETS_VALUE_woe	0.000000
22	FLAG_AMERICAN_EXPRESS_woe	0.000000

Clearly we observed that, most variables in this list have small information value, if we were just trying to be purist, only 6 variables will be kept for a strict 0.02 cutoff point. In this case, since we do not have any other stronger variables in this dataset, we set the cutoff point at 0.003 instead, that is, drop everything below that.

[68]:
```
train_woe =␣
↪train_woe[['TARGET_LABEL_BAD=1','RESIDENCIAL_BOROUGH_woe','RESIDENCIAL_CITY_woe','RESIDENCI
                        ␣
↪'PAYMENT_DAY_woe','MARITAL_STATUS_woe','FLAG_RESIDENCIAL_PHONE_woe','MONTHS_IN_RESIDENCE_wo
                        ␣
↪'RESIDENCIAL_STATE_woe','STATE_OF_BIRTH_woe','PERSONAL_MONTHLY_INCOME_woe']]
test_woe =␣
↪test_woe[['TARGET_LABEL_BAD=1','RESIDENCIAL_BOROUGH_woe','RESIDENCIAL_CITY_woe','RESIDENCIA
                        ␣
↪'PAYMENT_DAY_woe','MARITAL_STATUS_woe','FLAG_RESIDENCIAL_PHONE_woe','MONTHS_IN_RESIDENCE_wo
                        ␣
↪'RESIDENCIAL_STATE_woe','STATE_OF_BIRTH_woe','PERSONAL_MONTHLY_INCOME_woe']]
#No total income, correlations
```

train_woe = train_woe[['TARGET_LABEL_BAD=1','RESIDENCIAL_CITY_woe','AGE_woe','PAYMENT_
test_woe = test_woe[['TARGET_LABEL_BAD=1','RESIDENCIAL_CITY_woe','AGE_woe','PAYMENT_DA

[69]:
```
sc.iv(train_woe, 'TARGET_LABEL_BAD=1')
```

[69]:

	variable	info_value
5	RESIDENCIAL_BOROUGH_woe	1.029684
2	RESIDENCIAL_CITY_woe	0.364442
10	RESIDENCIAL_ZIP_3_woe	0.165772
6	AGE_woe	0.078019
3	PAYMENT_DAY_woe	0.032268
7	MARITAL_STATUS_woe	0.023044
0	FLAG_RESIDENCIAL_PHONE_woe	0.017929
8	RESIDENCIAL_STATE_woe	0.011005
1	STATE_OF_BIRTH_woe	0.008732
4	MONTHS_IN_RESIDENCE_woe	0.006032
9	PERSONAL_MONTHLY_INCOME_woe	0.003447

[14]:
```
train_woe.to_csv("train1woe.csv", index = False)
test_woe.to_csv("test1woe.csv", index = False)
```

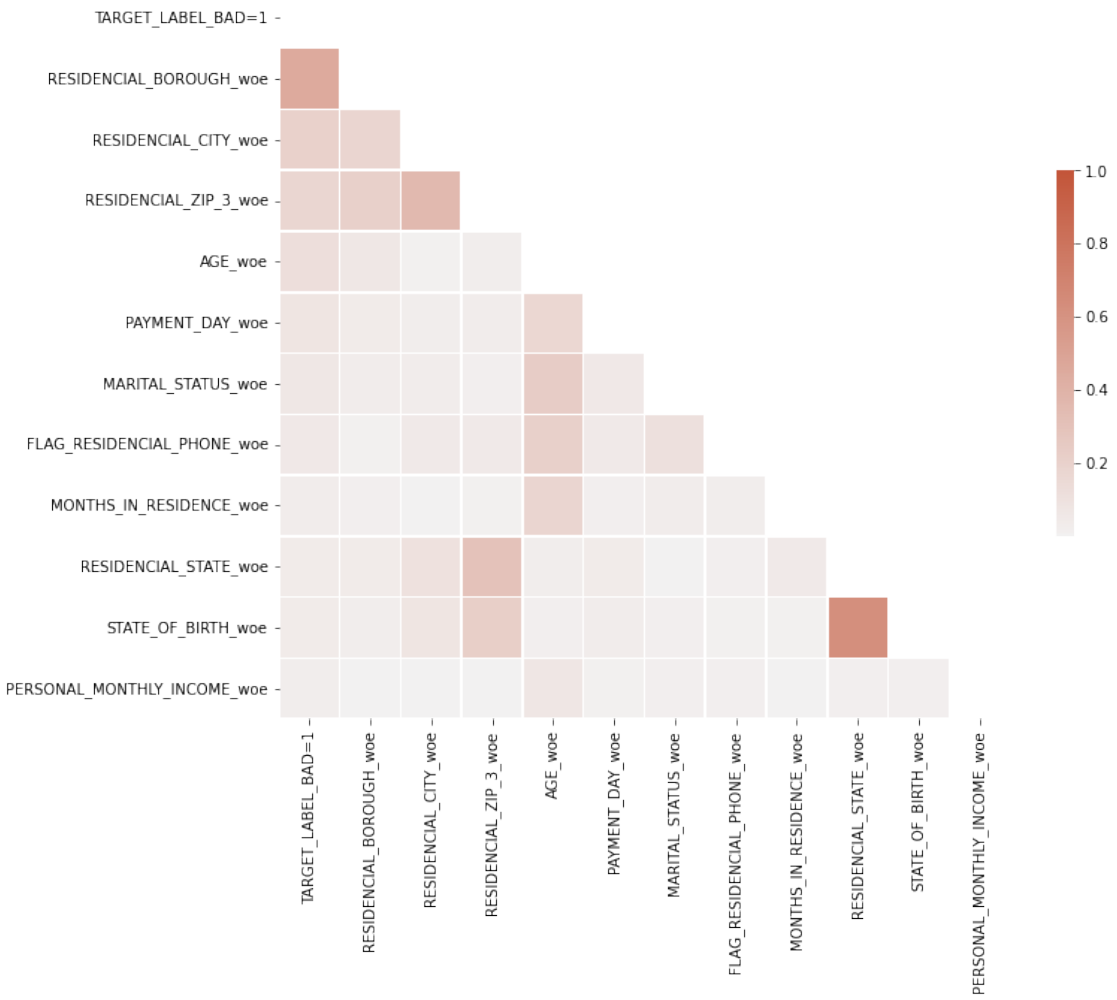
0.0.8 Correlation Analysis

[70]:
```
train_woe=pd.read_csv("train1woe.csv")
test_woe=pd.read_csv("test1woe.csv")
```

[71]:
```
corr = np.abs(train_woe.corr())
# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))
```

```
# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

[71]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2f453cdcd0>



[ ]:
```
#train_woe=train_woe.drop(columns=['RESIDENCIAL_STATE_woe'])
#test_woe=test_woe.drop(columns=['RESIDENCIAL_STATE_woe'])
```

0.0.9 Question3:

[75]:
```
from sklearn.linear_model import LogisticRegressionCV
```

[76]:
```
logreg = LogisticRegressionCV(penalty='elasticnet', # Type of penalization l1 =␣
↪lasso, l2 = ridge, elasticnet
                                    Cs = 13,        # How many parameters to␣
↪try. Can also be a vector with parameters to try.
                                    tol=0.00001, # Tolerance for parameters
                                    cv = 5,     # How many CV folds to try. 3␣
↪or 5 should be enough.
                                    fit_intercept=True, # Use constant?
                                    class_weight='balanced',# Weights, see␣
↪below
                                    random_state=251238783, # Random seed
                                    max_iter=150, # Maximum iterations
                                    verbose=0, # Show process. 1 is yes.
                                    solver = 'saga', # How to optimize.
                                    n_jobs = -1,      # Processes to use. Set␣
↪to number of physical cores.
                                    refit = True,     # If to retrain with the␣
↪best parameter and all data after finishing.
                                    l1_ratios = np.arange(0, 1.001, 0.01) #␣
↪The LASSO / Ridge ratios.
                                   )
```

[78]:
```
logreg.fit(X = train_woe.iloc[:,1:], # All rows and from the second var to end
           y = train_woe['TARGET_LABEL_BAD=1'] # The target
          )
```

[78]:
```
LogisticRegressionCV(Cs=13, class_weight='balanced', cv=5,
                     l1_ratios=array([0.  , 0.01, 0.02, 0.03, 0.04, 0.05, 0.06,
0.07, 0.08, 0.09, 0.1 ,
       0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 , 0.21,
       0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 , 0.31, 0.32,
       0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 , 0.41, 0.42, 0.43,
       0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5 , 0.51, 0.52, 0.53, 0.54,
       0.55, 0.56, 0.57, 0.58, 0.59, 0.6 , 0.61, 0.62, 0.63, 0.64, 0.65,
       0.66, 0.67, 0.68, 0.69, 0.7 , 0.71, 0.72, 0.73, 0.74, 0.75, 0.76,
       0.77, 0.78, 0.79, 0.8 , 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87,
       0.88, 0.89, 0.9 , 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98,
       0.99, 1.  ]),
                     max_iter=150, n_jobs=-1, penalty='elasticnet',
                     random_state=251238783, solver='saga', tol=1e-05)
```

[79]:
```
coef_df = pd.concat([pd.DataFrame({'column': train_woe.columns[1:]}),
                     pd.DataFrame(np.transpose(logreg.coef_))],
```

```
                    axis = 1
                   )
coef_df
```

[79]:

	column	0
0	RESIDENCIAL_BOROUGH_woe	0.871406
1	RESIDENCIAL_CITY_woe	0.573707
2	RESIDENCIAL_ZIP_3_woe	0.159891
3	AGE_woe	0.412604
4	PAYMENT_DAY_woe	0.134145
5	MARITAL_STATUS_woe	0.024869
6	FLAG_RESIDENCIAL_PHONE_woe	0.235135
7	MONTHS_IN_RESIDENCE_woe	0.000000
8	RESIDENCIAL_STATE_woe	0.000000
9	STATE_OF_BIRTH_woe	0.000000
10	PERSONAL_MONTHLY_INCOME_woe	0.000000

[80]:
```
logreg.intercept_
```

[80]:
```
array([0.00377898])
```

[81]:
```
print(logreg.l1_ratio_)
print(logreg.C_)
```

```
[0.2]
[0.00215443]
```

[82]:
```
pred_class_test = logreg.predict(test_woe.iloc[:, 1:])
probs_test = logreg.predict_proba(test_woe.iloc[:, 1:])
print(probs_test[0:5], pred_class_test[0:5])
```

```
[[0.67314508 0.32685492]
 [0.66825381 0.33174619]
 [0.56385367 0.43614633]
 [0.6551533  0.3448467 ]
 [0.67239938 0.32760062]] [0. 0. 0. 0. 0.]
```

[83]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true = test_woe['TARGET_LABEL_BAD=1'], y_pred =␣
 ↪pred_class_test)
```

[83]:
```
array([[9009, 2079],
       [3020,  892]])
```

[84]:
```
!pip install scorecardpy
import scorecardpy as sc
```

```
Requirement already satisfied: scorecardpy in /usr/local/lib/python3.7/dist-
packages (0.1.9.2)
```

```
Requirement already satisfied: scikit-learn>=0.19.1 in
/usr/local/lib/python3.7/dist-packages (from scorecardpy) (1.0.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from scorecardpy) (1.19.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-
packages (from scorecardpy) (3.2.2)
Requirement already satisfied: pandas>=0.25.0 in /usr/local/lib/python3.7/dist-
packages (from scorecardpy) (1.1.5)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=0.25.0->scorecardpy)
(2.8.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas>=0.25.0->scorecardpy) (2018.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.7.3->pandas>=0.25.0->scorecardpy) (1.15.0)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn>=0.19.1->scorecardpy) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn>=0.19.1->scorecardpy) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.19.1->scorecardpy)
(3.0.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib->scorecardpy) (3.0.6)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib->scorecardpy) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-
packages (from matplotlib->scorecardpy) (0.11.0)
```

[85]:
```
data_sc = sc.scorecard(bins_adj,
                       logreg,
                       train_woe.columns[1:],
                       points0=700,
                       odds0=0.5,
                       pdo=50)
```

[86]:
```
train_score = sc.scorecard_ply(cleantrain,data_sc,
                               print_step=0)
test_score = sc.scorecard_ply(cleantest, data_sc,
                               print_step=0)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1596:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  self.obj[key] = _infer_fill_value(value)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1781:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[item_labels[indexer[info_axis]]] = value
```

[87]:
```
train_score.describe()
```

[87]:
```
              score
count  33522.000000
mean     668.748822
std       77.918294
min      506.000000
25%      578.000000
50%      699.000000
75%      712.000000
max      833.000000
```

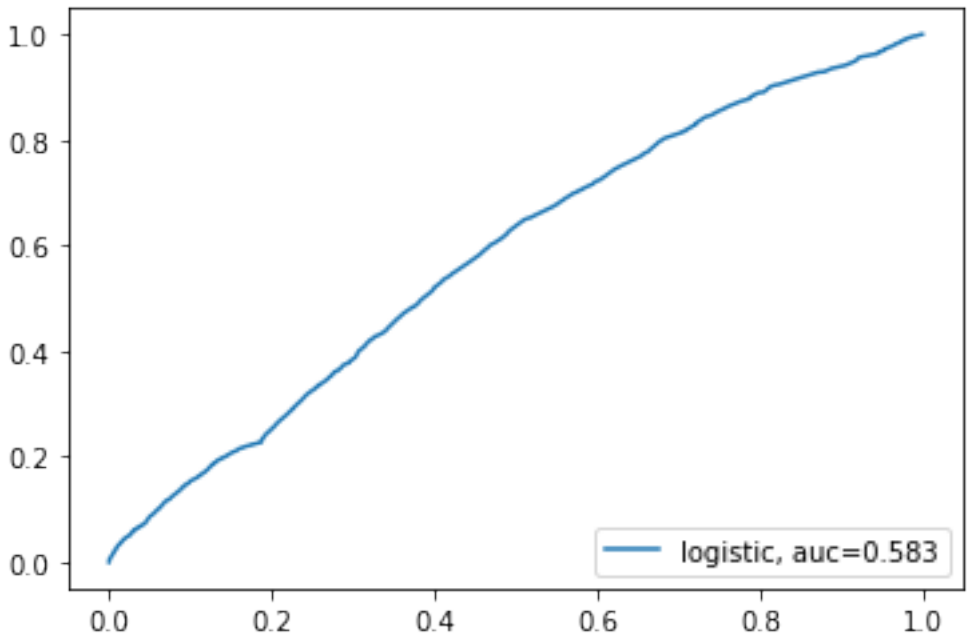
[88]:
```
test_score.describe()
```

[88]:
```
              score
count  15000.000000
mean     678.310533
std       67.147868
min      510.000000
25%      669.000000
50%      701.000000
75%      712.000000
max      833.000000
```

[89]:
```
from sklearn.metrics import roc_auc_score, confusion_matrix, roc_curve
```

[90]:
```
#AUC
fpr, tpr, thresholds = roc_curve(test_woe['TARGET_LABEL_BAD=1'], probs_test[:
 ↪,1])

# Save the AUC in a variable to display it. Round it first
auc = np.round(roc_auc_score(y_true = test_woe['TARGET_LABEL_BAD=1'],
                             y_score = probs_test[:,1]),
               decimals = 3)

# Create and show the plot
plt.plot(fpr,tpr,label="logistic, auc="+str(auc))
plt.legend(loc=4)
```

```
plt.show()
```



0.0.10 Question 4

[63]:
```
cleantrain = pd.read_csv('train_clean.csv')
```

[65]:
```
cleantrain.loc[:,'MARITAL_STATUS']=cleantrain.loc[:,'MARITAL_STATUS'].
 ↪astype(str)
cleantrain.loc[:,'STATE_OF_BIRTH']=cleantrain.loc[:,'STATE_OF_BIRTH'].
 ↪astype(str)
cleantrain.loc[:,'RESIDENCIAL_STATE']=cleantrain.loc[:,'RESIDENCIAL_STATE'].
 ↪astype(str)
cleantrain.loc[:,'RESIDENCIAL_CITY']=cleantrain.loc[:,'RESIDENCIAL_CITY'].
 ↪astype(str)
cleantrain.loc[:,'RESIDENCIAL_BOROUGH']=cleantrain.loc[:,'RESIDENCIAL_BOROUGH'].
 ↪astype(str)
cleantrain.loc[:,'RESIDENCE_TYPE']=cleantrain.loc[:,'RESIDENCE_TYPE'].
 ↪astype(str)
cleantrain.loc[:,'RESIDENCIAL_ZIP_3']=cleantrain.loc[:,'RESIDENCIAL_ZIP_3'].
 ↪astype(str)
```

[36]:
```
cleantest = pd.read_csv('test_clean.csv')
```

[37]:
```
cleantest.loc[:,'MARITAL_STATUS']=cleantest.loc[:,'MARITAL_STATUS'].astype(str)
cleantest.loc[:,'STATE_OF_BIRTH']=cleantest.loc[:,'STATE_OF_BIRTH'].astype(str)
```

```
cleantest.loc[:,'RESIDENCIAL_STATE']=cleantest.loc[:,'RESIDENCIAL_STATE'].
 ↪astype(str)
cleantest.loc[:,'RESIDENCIAL_CITY']=cleantest.loc[:,'RESIDENCIAL_CITY'].
 ↪astype(str)
cleantest.loc[:,'RESIDENCIAL_BOROUGH']=cleantest.loc[:,'RESIDENCIAL_BOROUGH'].
 ↪astype(str)
cleantest.loc[:,'RESIDENCE_TYPE']=cleantest.loc[:,'RESIDENCE_TYPE'].astype(str)
cleantest.loc[:,'RESIDENCIAL_ZIP_3']=cleantest.loc[:,'RESIDENCIAL_ZIP_3'].
 ↪astype(str)
```

When we first time clean the dataset, we took the random forest into account. We divided the crazy amount of categorical variables' categories into smaller amount of categories that make much more sense. In this case, we do not need to specifically do another data cleaning just for the random forest, all we need to do here is making sure the training and test data looks fine, then apply the dummy variables to it.

The data looks good, all the variables with categories more than two have string values, now we are able to apply the dummies.

[38]:
```
rf_train=pd.get_dummies(cleantrain,drop_first=True)
rf_test=pd.get_dummies(cleantest,drop_first=True)
```

[39]:
```
rf_train.describe(include='all')
```

[39]:

	PAYMENT_DAY	POSTAL_ADDRESS_TYPE	SEX	QUANT_DEPENDANTS	\
count	33522.000000	33522.000000	33522.000000	33522.000000	
mean	12.883480	1.006623	0.620607	0.631824	
std	6.616225	0.081110	0.485243	1.096871	
min	1.000000	1.000000	0.000000	0.000000	
25%	10.000000	1.000000	0.000000	0.000000	
50%	10.000000	1.000000	1.000000	0.000000	
75%	15.000000	1.000000	1.000000	1.000000	
max	25.000000	2.000000	1.000000	6.000000	

	NACIONALITY	FLAG_RESIDENCIAL_PHONE	MONTHS_IN_RESIDENCE	\
count	33522.000000	33522.000000	33522.000000	
mean	0.955910	0.837032	9.108138	
std	0.205299	0.369342	9.704732	
min	0.000000	0.000000	0.000000	
25%	1.000000	1.000000	2.000000	
50%	1.000000	1.000000	6.000000	
75%	1.000000	1.000000	13.000000	
max	1.000000	1.000000	50.000000	

	FLAG_EMAIL	PERSONAL_MONTHLY_INCOME	OTHER_INCOMES	FLAG_VISA	\
count	33522.000000	33522.000000	33522.000000	33522.000000	
mean	0.806097	709.721933	11.760512	0.105751	
std	0.395359	689.427599	64.932670	0.307524	

```
min        0.000000               69.000000       0.000000      0.000000
25%        1.000000              358.000000       0.000000      0.000000
50%        1.000000              500.000000       0.000000      0.000000
75%        1.000000              800.000000       0.000000      0.000000
max        1.000000            13757.000000     599.000000      1.000000

       FLAG_MASTERCARD   FLAG_DINERS  FLAG_AMERICAN_EXPRESS  FLAG_OTHER_CARDS  \
count     33522.000000  33522.000000           33522.000000      33522.000000
mean          0.092864      0.001253               0.001492          0.001999
std           0.290247      0.035375               0.038592          0.044663
min           0.000000      0.000000               0.000000          0.000000
25%           0.000000      0.000000               0.000000          0.000000
50%           0.000000      0.000000               0.000000          0.000000
75%           0.000000      0.000000               0.000000          0.000000
max           1.000000      1.000000               1.000000          1.000000

       QUANT_BANKING_ACCOUNTS  QUANT_SPECIAL_BANKING_ACCOUNTS  \
count            33522.000000                    33522.000000
mean                0.355587                        0.355587
std                 0.479259                        0.479259
min                 0.000000                        0.000000
25%                 0.000000                        0.000000
50%                 0.000000                        0.000000
75%                 1.000000                        1.000000
max                 2.000000                        2.000000

       PERSONAL_ASSETS_VALUE    QUANT_CARS       COMPANY  \
count           33522.000000  33522.000000  33522.000000
mean              607.105483      0.331543      0.438220
std              3860.848213      0.470775      0.496176
min                 0.000000      0.000000      0.000000
25%                 0.000000      0.000000      0.000000
50%                 0.000000      0.000000      0.000000
75%                 0.000000      1.000000      1.000000
max             38000.000000      1.000000      1.000000

       FLAG_PROFESSIONAL_PHONE       PRODUCT           AGE  \
count             33522.000000  33522.000000  33522.000000
mean                  0.268242      1.170843     42.986099
std                   0.443051      0.438333     14.917561
min                   0.000000      1.000000      7.000000
25%                   0.000000      1.000000     31.000000
50%                   0.000000      1.000000     41.000000
75%                   1.000000      1.000000     53.000000
max                   1.000000      3.000000    101.000000

       TARGET_LABEL_BAD=1  TOTAL_INCOME  TOTAL_BANKING_ACCOUNTS  \
```

```
       33522.000000  33522.000000           33522.000000
mean       0.261977    721.482444               0.711175
std        0.439717    689.720611               0.958517
min        0.000000     69.000000               0.000000
25%        0.000000    368.000000               0.000000
50%        0.000000    500.000000               0.000000
75%        1.000000    800.000000               2.000000
max        1.000000  13757.000000               4.000000

       TOTAL_TYPE_OF_CARDS  MARITAL_STATUS_2  MARITAL_STATUS_3  \
count         33522.000000      33522.000000      33522.000000
mean              0.203359          0.071893          0.618370
std               0.510172          0.258315          0.485794
min               0.000000          0.000000          0.000000
25%               0.000000          0.000000          0.000000
50%               0.000000          0.000000          1.000000
75%               0.000000          0.000000          1.000000
max               4.000000          1.000000          1.000000

       STATE_OF_BIRTH_2  STATE_OF_BIRTH_3  RESIDENCIAL_STATE_2  \
count      33522.000000      33522.000000         33522.000000
mean           0.518883          0.325667             0.647097
std            0.499651          0.468630             0.477880
min            0.000000          0.000000             0.000000
25%            0.000000          0.000000             0.000000
50%            1.000000          0.000000             1.000000
75%            1.000000          1.000000             1.000000
max            1.000000          1.000000             1.000000

       RESIDENCIAL_STATE_3  RESIDENCIAL_CITY_2  RESIDENCIAL_CITY_3  \
count         33522.000000        33522.000000        33522.000000
mean              0.199570            0.697602            0.050206
std               0.399683            0.459304            0.218373
min               0.000000            0.000000            0.000000
25%               0.000000            0.000000            0.000000
50%               0.000000            1.000000            0.000000
75%               0.000000            1.000000            0.000000
max               1.000000            1.000000            1.000000

       RESIDENCIAL_CITY_4  RESIDENCIAL_BOROUGH_2  RESIDENCIAL_BOROUGH_3  \
count        33522.000000           33522.000000           33522.000000
mean             0.025267               0.219527               0.490514
std              0.156937               0.413933               0.499917
min              0.000000               0.000000               0.000000
25%              0.000000               0.000000               0.000000
50%              0.000000               0.000000               0.000000
75%              0.000000               0.000000               1.000000
```

```
max              1.000000              1.000000              1.000000

       RESIDENCIAL_BOROUGH_4  RESIDENCIAL_BOROUGH_5  RESIDENCIAL_BOROUGH_6  \
count           33522.000000           33522.000000           33522.000000
mean                0.081529               0.076636               0.082573
std                 0.273649               0.266017               0.275239
min                 0.000000               0.000000               0.000000
25%                 0.000000               0.000000               0.000000
50%                 0.000000               0.000000               0.000000
75%                 0.000000               0.000000               0.000000
max                 1.000000               1.000000               1.000000

       RESIDENCE_TYPE_2.0  RESIDENCE_TYPE_3.0  RESIDENCE_TYPE_4.0  \
count        33522.000000        33522.000000        33522.000000
mean             0.078217            0.003073            0.006384
std              0.268517            0.055347            0.079645
min              0.000000            0.000000            0.000000
25%              0.000000            0.000000            0.000000
50%              0.000000            0.000000            0.000000
75%              0.000000            0.000000            0.000000
max              1.000000            1.000000            1.000000

       RESIDENCE_TYPE_5.0  RESIDENCIAL_ZIP_3_2  RESIDENCIAL_ZIP_3_3  \
count        33522.000000         33522.000000         33522.000000
mean             0.041227             0.405614             0.353111
std              0.198817             0.491018             0.477944
min              0.000000             0.000000             0.000000
25%              0.000000             0.000000             0.000000
50%              0.000000             0.000000             0.000000
75%              0.000000             1.000000             1.000000
max              1.000000             1.000000             1.000000

       RESIDENCIAL_ZIP_3_4
count         33522.000000
mean              0.144323
std               0.351422
min               0.000000
25%               0.000000
50%               0.000000
75%               0.000000
max               1.000000
```

[40]:
```
rf_test.describe(include='all')
```

[40]:
```
        PAYMENT_DAY  POSTAL_ADDRESS_TYPE           SEX  QUANT_DEPENDANTS  \
count  15000.000000         15000.000000  15000.000000      15000.000000
mean      12.889600             1.005933      0.612133          0.634267
```

```
std        6.597276             0.076802      0.487280          1.231777
min        1.000000             1.000000      0.000000          0.000000
25%       10.000000             1.000000      0.000000          0.000000
50%       10.000000             1.000000      1.000000          0.000000
75%       15.000000             1.000000      1.000000          1.000000
max       25.000000             2.000000      1.000000         53.000000

        NACIONALITY  FLAG_RESIDENCIAL_PHONE  MONTHS_IN_RESIDENCE  \
count  15000.000000            15000.000000         15000.000000
mean       0.958133                0.837667             9.457333
std        0.200291                0.368769            10.242036
min        0.000000                0.000000             0.000000
25%        1.000000                1.000000             2.000000
50%        1.000000                1.000000             6.000000
75%        1.000000                1.000000            14.000000
max        1.000000                1.000000            99.000000

         FLAG_EMAIL  PERSONAL_MONTHLY_INCOME  OTHER_INCOMES     FLAG_VISA  \
count  15000.000000             15000.000000   15000.000000  15000.000000
mean       0.798800               799.874569      44.416593      0.111533
std        0.400911              3412.383497    1599.998798      0.314802
min        0.000000                60.000000       0.000000      0.000000
25%        1.000000               360.000000       0.000000      0.000000
50%        1.000000               500.000000       0.000000      0.000000
75%        1.000000               800.000000       0.000000      0.000000
max        1.000000            198183.000000  194344.000000      1.000000

       FLAG_MASTERCARD   FLAG_DINERS  FLAG_AMERICAN_EXPRESS  FLAG_OTHER_CARDS  \
count     15000.000000  15000.000000           15000.000000      15000.000000
mean          0.099333      0.001000               0.002133          0.002200
std           0.299119      0.031608               0.046140          0.046854
min           0.000000      0.000000               0.000000          0.000000
25%           0.000000      0.000000               0.000000          0.000000
50%           0.000000      0.000000               0.000000          0.000000
75%           0.000000      0.000000               0.000000          0.000000
max           1.000000      1.000000               1.000000          1.000000

       QUANT_BANKING_ACCOUNTS  QUANT_SPECIAL_BANKING_ACCOUNTS  \
count            15000.000000                    15000.000000
mean                 0.358000                        0.358000
std                  0.480123                        0.480123
min                  0.000000                        0.000000
25%                  0.000000                        0.000000
50%                  0.000000                        0.000000
75%                  1.000000                        1.000000
max                  2.000000                        2.000000
```

```
       PERSONAL_ASSETS_VALUE    QUANT_CARS       COMPANY  \
count           1.500000e+04  15000.000000  15000.000000
mean            2.137884e+03      0.337533      0.442267
std             2.027787e+04      0.472884      0.496672
min             0.000000e+00      0.000000      0.000000
25%             0.000000e+00      0.000000      0.000000
50%             0.000000e+00      0.000000      0.000000
75%             0.000000e+00      1.000000      1.000000
max             1.800000e+06      1.000000      1.000000

       FLAG_PROFESSIONAL_PHONE       PRODUCT           AGE  \
count             15000.000000  15000.000000  15000.000000
mean                  0.268533      1.169867     43.407867
std                   0.443211      0.438587     15.071138
min                   0.000000      1.000000     17.000000
25%                   0.000000      1.000000     32.000000
50%                   0.000000      1.000000     42.000000
75%                   1.000000      1.000000     54.000000
max                   1.000000      3.000000    106.000000

       TARGET_LABEL_BAD=1   TOTAL_INCOME  TOTAL_BANKING_ACCOUNTS  \
count        15000.000000   15000.000000            15000.000000
mean             0.260800     844.291162                0.716000
std              0.439086    3768.966713                0.960246
min              0.000000      60.000000                0.000000
25%              0.000000     372.000000                0.000000
50%              0.000000     510.000000                0.000000
75%              1.000000     826.240000                2.000000
max              1.000000  198183.000000                4.000000

       TOTAL_TYPE_OF_CARDS  MARITAL_STATUS_2  MARITAL_STATUS_3  \
count         15000.000000      15000.000000      15000.000000
mean              0.216200          0.076000          0.614667
std               0.523331          0.265007          0.486690
min               0.000000          0.000000          0.000000
25%               0.000000          0.000000          0.000000
50%               0.000000          0.000000          1.000000
75%               0.000000          0.000000          1.000000
max               4.000000          1.000000          1.000000

       STATE_OF_BIRTH_2  STATE_OF_BIRTH_3  RESIDENCIAL_STATE_2  \
count      15000.000000      15000.000000         15000.000000
mean           0.514200          0.328867             0.637733
std            0.499815          0.469817             0.480671
min            0.000000          0.000000             0.000000
25%            0.000000          0.000000             0.000000
50%            1.000000          0.000000             1.000000
```

```
75%            1.000000          1.000000             1.000000
max            1.000000          1.000000             1.000000

       RESIDENCIAL_STATE_3  RESIDENCIAL_CITY_2  RESIDENCIAL_CITY_3  \
count         15000.000000        15000.000000        15000.000000
mean              0.204400            0.781667            0.038333
std               0.403276            0.413129            0.192006
min               0.000000            0.000000            0.000000
25%               0.000000            1.000000            0.000000
50%               0.000000            1.000000            0.000000
75%               0.000000            1.000000            0.000000
max               1.000000            1.000000            1.000000

       RESIDENCIAL_CITY_4  RESIDENCIAL_BOROUGH_2  RESIDENCIAL_BOROUGH_3  \
count        15000.000000            15000.00000            15000.000000
mean             0.016000                0.17860                0.653400
std              0.125479                0.38303                0.475903
min              0.000000                0.00000                0.000000
25%              0.000000                0.00000                0.000000
50%              0.000000                0.00000                1.000000
75%              0.000000                0.00000                1.000000
max              1.000000                1.00000                1.000000

       RESIDENCIAL_BOROUGH_4  RESIDENCIAL_BOROUGH_5  RESIDENCIAL_BOROUGH_6  \
count           15000.000000           15000.000000           15000.000000
mean                0.049400               0.041467               0.054533
std                 0.216709               0.199374               0.227075
min                 0.000000               0.000000               0.000000
25%                 0.000000               0.000000               0.000000
50%                 0.000000               0.000000               0.000000
75%                 0.000000               0.000000               0.000000
max                 1.000000               1.000000               1.000000

       RESIDENCE_TYPE_2.0  RESIDENCE_TYPE_3.0  RESIDENCE_TYPE_4.0  \
count        15000.000000        15000.000000        15000.000000
mean             0.079000            0.002467            0.005800
std              0.269748            0.049606            0.075939
min              0.000000            0.000000            0.000000
25%              0.000000            0.000000            0.000000
50%              0.000000            0.000000            0.000000
75%              0.000000            0.000000            0.000000
max              1.000000            1.000000            1.000000

       RESIDENCE_TYPE_5.0  RESIDENCIAL_ZIP_3_2  RESIDENCIAL_ZIP_3_3  \
count        15000.000000         15000.000000         15000.000000
mean             0.038200             0.404600             0.352933
std              0.191685             0.490831             0.477898
```

```
min              0.000000            0.000000            0.000000
25%              0.000000            0.000000            0.000000
50%              0.000000            0.000000            0.000000
75%              0.000000            1.000000            1.000000
max              1.000000            1.000000            1.000000

       RESIDENCIAL_ZIP_3_4
count         15000.000000
mean              0.147800
std               0.354913
min               0.000000
25%               0.000000
50%               0.000000
75%               0.000000
max               1.000000
```

0.0.11 Random Forest

[41]:
```
from sklearn.ensemble import RandomForestClassifier
default_rf = RandomForestClassifier(n_estimators=3000,
                        criterion='entropy',
                        max_depth=None,
                        min_samples_split=2,
                        min_samples_leaf=0.0001,
                        min_weight_fraction_leaf=0.0,
                        max_features='auto',
                        max_leaf_nodes=None,
                        min_impurity_decrease=0.00001,
                        bootstrap=True,
                        oob_score=True,
                        n_jobs=-1,
                        random_state=251238783,
                        verbose=0,
                        warm_start=False,
                        class_weight='balanced'
                                    )
```

0.0.12 CV grid search

[42]:
```
param_grid_rf = dict({'n_estimators': [2000,3000,4000,5000,6000]})
```

[43]:
```
val_train_rf = rf_train.sample(frac = 0.2, random_state = 251238783)
```

[44]:
```
from sklearn.model_selection import GridSearchCV
```

[42]:
```
GridRF = GridSearchCV(default_rf,
                        param_grid_rf,
```

```
                        cv = 5,
                        scoring = 'roc_auc',
                        n_jobs = -1,
                        refit = False,
                        verbose = 0
                       )
```

[28]:
```
val_train_rf_x=val_train_rf.drop(columns='TARGET_LABEL_BAD=1')
val_train_rf_y=val_train_rf.loc[:,'TARGET_LABEL_BAD=1']
```

[44]:
```
GridRF.fit(val_train_rf_x, val_train_rf_y)
```

```
/usr/local/lib/python3.7/dist-
packages/joblib/externals/loky/process_executor.py:705: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

[44]:
```
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(class_weight='balanced',
                                              criterion='entropy',
                                              min_impurity_decrease=1e-05,
                                              min_samples_leaf=0.0001,
                                              n_estimators=3000, n_jobs=-1,
                                              oob_score=True,
                                              random_state=251238783),
             n_jobs=-1,
             param_grid={'n_estimators': [2000, 3000, 4000, 5000, 6000]},
             refit=False, scoring='roc_auc')
```

[45]:
```
print('The best AUC is %.3f' % GridRF.best_score_)
GridRF.best_params_
```

```
The best AUC is 0.836
```

[45]:
```
{'n_estimators': 3000}
```

0.0.13 Fit the model

[45]:
```
rf_train_x=rf_train.drop(columns='TARGET_LABEL_BAD=1')
rf_train_y=rf_train.loc[:,'TARGET_LABEL_BAD=1']
```

[46]:
```
default_rf.fit(rf_train_x, # X
               rf_train_y    # y
                )
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:446: UserWarning: X does
not have valid feature names, but RandomForestClassifier was fitted with feature
```

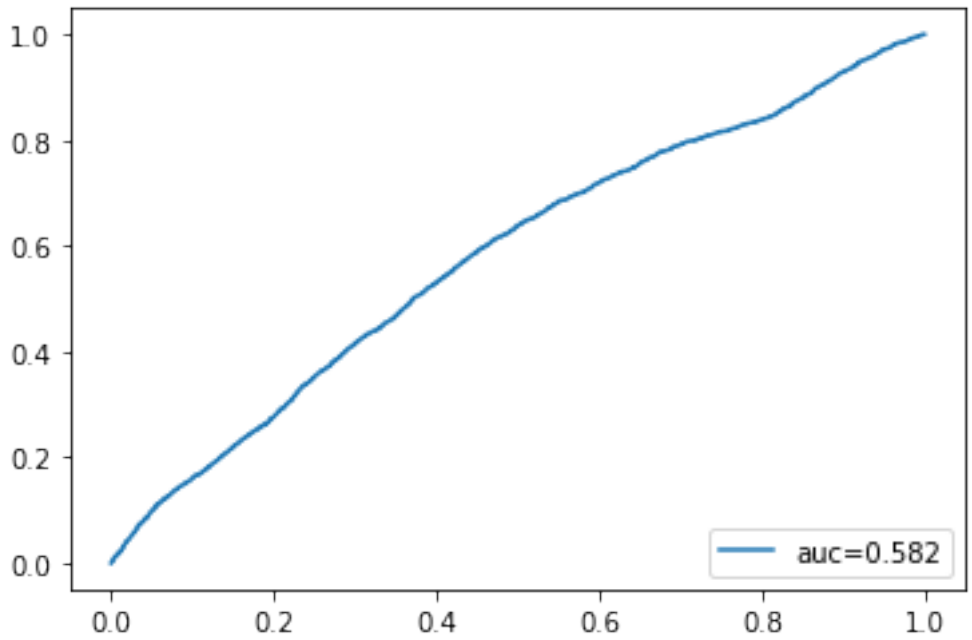
```
names
  "X does not have valid feature names, but"
```

[46]:
```
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       min_impurity_decrease=1e-05, min_samples_leaf=0.0001,
                       n_estimators=3000, n_jobs=-1, oob_score=True,
                       random_state=251238783)
```

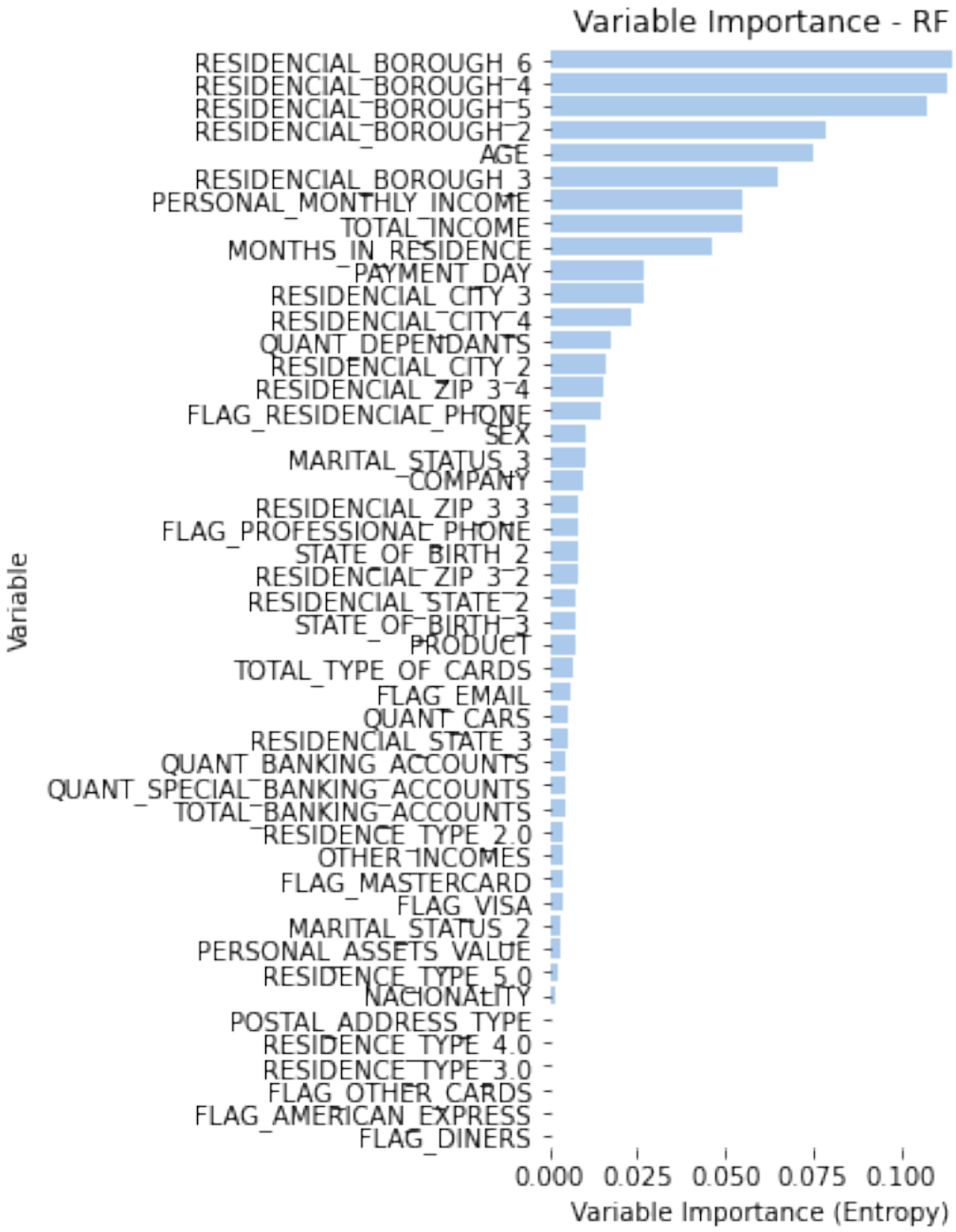
[47]:
```
rf_test_x=rf_test.drop(columns='TARGET_LABEL_BAD=1')
rf_test_y=rf_test.loc[:,'TARGET_LABEL_BAD=1']

rf_pred_class_test = default_rf.predict(rf_test_x)
rf_probs_test = default_rf.predict_proba(rf_test_x)
```

[48]:
```
from sklearn.metrics import roc_auc_score, confusion_matrix, roc_curve
```

[49]:
```
fpr, tpr, thresholds = roc_curve(rf_test['TARGET_LABEL_BAD=1'], rf_probs_test[:
 ↪,1])

# Save the AUC in a variable to display it. Round it first
auc = np.round(roc_auc_score(y_true = rf_test['TARGET_LABEL_BAD=1'],
                             y_score = rf_probs_test[:,1]),
               decimals = 3)

# Create and show the plot
plt.plot(fpr,tpr,label="auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



[51]:
```
importances = default_rf.feature_importances_
indices = np.argsort(importances)[::-1]

f, ax = plt.subplots(figsize=(3, 8))
plt.title("Variable Importance - RF")
sns.set_color_codes("pastel")
sns.barplot(y=[rf_train_x.columns[i] for i in indices], x=importances[indices],
            label="Total", color="b")
ax.set(ylabel="Variable",
       xlabel="Variable Importance (Entropy)")
sns.despine(left=True, bottom=True)
```



0.0.14 XGBoosting

For this part, we can directly took the dataset that we used for Random Forest to model

[3]:
```
from xgboost import XGBClassifier
```

[4]:
```
XGB_default = XGBClassifier(max_depth=2,
                            learning_rate=0.15,
                            n_estimators=150,
                            verbosity=0,
                            objective='binary:logistic',
                            booster='gbtree',
                            #n_jobs=8,
                            gamma=0.001,
                            subsample=0.632,
                            colsample_bytree=1,
                            colsample_bylevel=1,
                            colsample_bynode=1,
                            reg_alpha=1,
                            reg_lambda=0,
                            scale_pos_weight=1,
                            base_score=0.5,
                            random_state=251238783,
                            missing=None,
                            tree_method='gpu_hist',
                            gpu_id=0,
                            use_label_encoder=False
                            )
```

[20]:
```
param_gridXG = dict({'n_estimators': [50, 100, 150],
                     'max_depth': [2, 3, 4],
                   'learning_rate' : [0.01, 0.05, 0.1, 0.15]
                    })
```

[24]:
```
GridXGB = GridSearchCV(XGB_default,
                       param_gridXG,
                       cv = 5,
                       scoring = 'roc_auc',
                       refit = False,
                       verbose = 1,
                      )
```

[29]:
```
val_train_rf_y=val_train_rf_y.astype(int)
```

[30]:
```
GridXGB.fit(val_train_rf_x, val_train_rf_y)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

[30]:
```
GridSearchCV(cv=5,
             estimator=XGBClassifier(gamma=0.001, gpu_id=0, learning_rate=0.15,
                                     max_depth=2, n_estimators=150,
                                     random_state=251238783, reg_alpha=1,
                                     reg_lambda=0, subsample=0.632,
                                     tree_method='gpu_hist',
```

```
                                use_label_encoder=False, verbosity=0),
             param_grid={'learning_rate': [0.01, 0.05, 0.1, 0.15],
                         'max_depth': [2, 3, 4],
                         'n_estimators': [50, 100, 150]},
             refit=False, scoring='roc_auc', verbose=1)
```

[31]:
```
print('The best AUC is %.3f' % GridXGB.best_score_)
GridXGB.best_params_
```

```
The best AUC is 0.854
```

[31]:
```
{'learning_rate': 0.15, 'max_depth': 2, 'n_estimators': 150}
```

[32]:
```
param_gridXG = dict({'n_estimators': [150],
                     'max_depth': [2],
                    'learning_rate' : [0.15,0.2]
                     })
```

[33]:
```
GridXGB.fit(val_train_rf_x, val_train_rf_y)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

[33]:
```
GridSearchCV(cv=5,
             estimator=XGBClassifier(gamma=0.001, gpu_id=0, learning_rate=0.15,
                                     max_depth=2, n_estimators=150,
                                     random_state=251238783, reg_alpha=1,
                                     reg_lambda=0, subsample=0.632,
                                     tree_method='gpu_hist',
                                     use_label_encoder=False, verbosity=0),
             param_grid={'learning_rate': [0.01, 0.05, 0.1, 0.15],
                         'max_depth': [2, 3, 4],
                         'n_estimators': [50, 100, 150]},
             refit=False, scoring='roc_auc', verbose=1)
```

[34]:
```
print('The best AUC is %.3f' % GridXGB.best_score_)
GridXGB.best_params_
```

```
The best AUC is 0.854
```

[34]:
```
{'learning_rate': 0.15, 'max_depth': 2, 'n_estimators': 150}
```

Now we find our three best parameters: 'learning_rate': 0.15, 'max_depth': 2, 'n_estimators': 150

[62]:
```
XGB_final = XGBClassifier(max_depth=2,
                          learning_rate=0.15,
                          n_estimators=150,
                          verbosity=0,
                          objective='binary:logistic',
                          booster='gbtree',
```

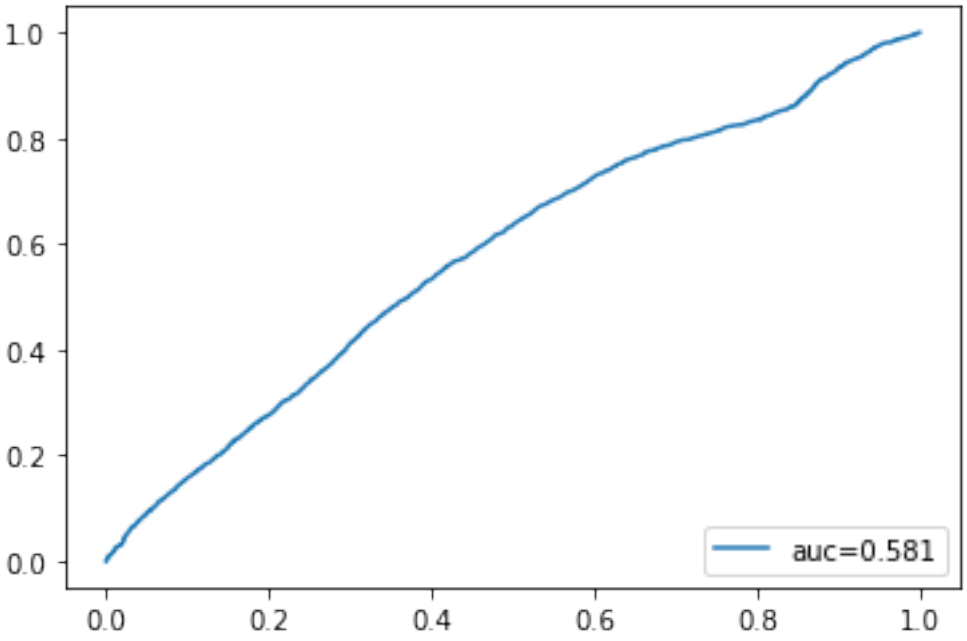
```
                            #n_jobs=8,
                            gamma=0.001,
                            subsample=0.632,
                            colsample_bytree=1,
                            colsample_bylevel=1,
                            colsample_bynode=1,
                            reg_alpha=1,
                            reg_lambda=0,
                            scale_pos_weight=1,
                            base_score=0.5,
                            random_state=251238783,
                            missing=None,
                            tree_method='gpu_exact',
                            gpu_id=0,
                            use_label_encoder=False
                            )
```

[63]:
```
XGB_final.fit(rf_train_x, rf_train_y)
```

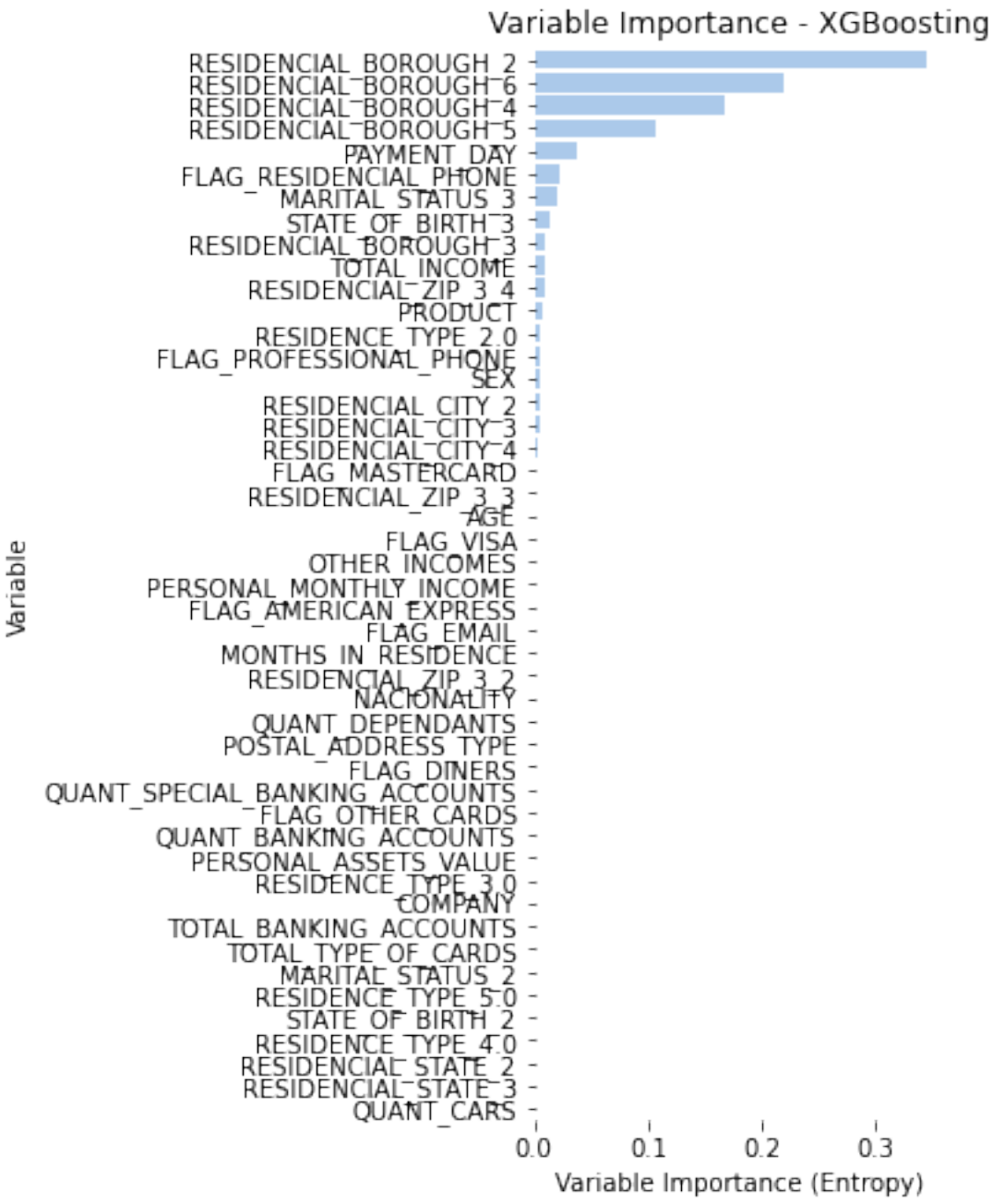
[63]:
```
XGBClassifier(gamma=0.001, gpu_id=0, learning_rate=0.15, max_depth=2,
              n_estimators=150, random_state=251238783, reg_alpha=1,
              reg_lambda=0, subsample=0.632, tree_method='gpu_exact',
              use_label_encoder=False, verbosity=0)
```

[64]:
```
rf_test_x=rf_test.drop(columns='TARGET_LABEL_BAD=1')
rf_test_y=rf_test.loc[:,'TARGET_LABEL_BAD=1']

XG_pred_class_test = XGB_final.predict(rf_test_x)
XG_probs_test = XGB_final.predict_proba(rf_test_x)
```

[65]:
```
fpr, tpr, thresholds = roc_curve(rf_test['TARGET_LABEL_BAD=1'], XG_probs_test[:
 ↪,1])

# Save the AUC in a variable to display it. Round it first
auc = np.round(roc_auc_score(y_true = rf_test['TARGET_LABEL_BAD=1'],
                             y_score = XG_probs_test[:,1]),
               decimals = 3)

# Create and show the plot
plt.plot(fpr,tpr,label="auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



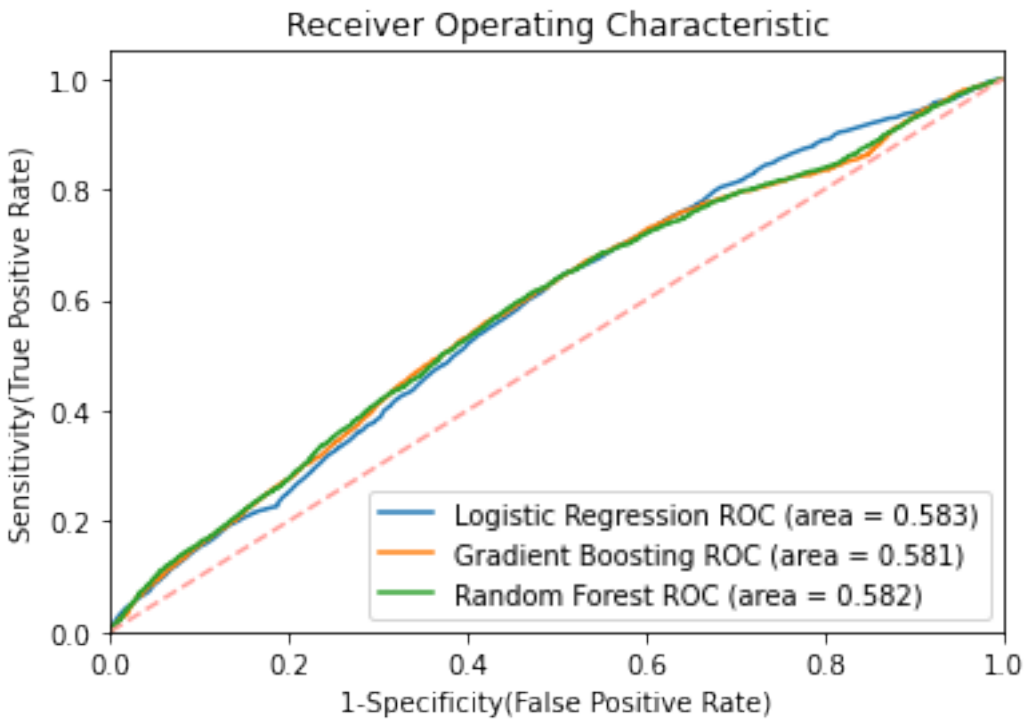
[66]:
```
importances = XGB_final.feature_importances_
indices = np.argsort(importances)[::-1]

f, ax = plt.subplots(figsize=(3, 8))
plt.title("Variable Importance - XGBoosting")
sns.set_color_codes("pastel")
sns.barplot(y=[rf_train_x.columns[i] for i in indices], x=importances[indices],
            label="Total", color="b")
ax.set(ylabel="Variable",
       xlabel="Variable Importance (Entropy)")
sns.despine(left=True, bottom=True)
```



[68]:
```
# Predict probabilities of scorecard.
logreg_probs_test = logreg.predict_proba(test_woe.iloc[:, 1:])
```

[81]:
```
models = [
{
    'label': 'Logistic Regression',
    'probs': probs_test[:,1]
```

```
},
{
    'label': 'Gradient Boosting',
    'probs': XG_probs_test[:,1]
},
{
    'label': 'Random Forest',
    'probs': rf_probs_test[:,1]
}
]

# Loop that creates the plot. I will pass each ROC curve one by one.
for m in models:
  auc = roc_auc_score(y_true = rf_test['TARGET_LABEL_BAD=1'],
                              y_score = m['probs'])
  fpr, tpr, thresholds = roc_curve(rf_test['TARGET_LABEL_BAD=1'],
                                           m['probs'])
  plt.plot(fpr, tpr, label='%s ROC (area = %0.3f)' % (m['label'], auc))


# Settings
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity(False Positive Rate)')
plt.ylabel('Sensitivity(True Positive Rate)')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")

# Plot!
plt.show()
```



0.0.15 Question 6

[92]:
```
test_score.describe()
```

[92]:
```
              score
count  15000.000000
mean     678.310533
std       67.147868
min      510.000000
25%      669.000000
50%      701.000000
75%      712.000000
max      833.000000
```

[122]:
```
c=np.linspace(520,830,32)
```

[115]:
```
cleantest['score']=test_score
```

[116]:
```
cotable = pd.DataFrame([], columns=['cutoff', 'Accepted_rate', 'Accuracy_good',
                   'Accuracy_bad', 'Accuracy_Total', 'Avg_cost_g',
                    'Avg_cost_b', 'total_cost_g','total_cost_b',
                    'total_cost'])
```

[117]:
```
cotable['cutoff']=c
```

[118]:
```
# total amount of good people
ttg=(1-cleantest.loc[:,'TARGET_LABEL_BAD=1'].mean())*15000
```

[119]:
```
cleantest.loc[:,'TARGET_LABEL_BAD=1']=cleantest.loc[:,'TARGET_LABEL_BAD=1'].
↪astype(int)
```

[120]:
```
cleantest[cleantest['TARGET_LABEL_BAD=1']==0][cleantest['score']>700]['PERSONAL_MONTHLY_INCOME
↪sum()
cleantest[cleantest['TARGET_LABEL_BAD=1']==0][cleantest['score']<600]['score'].
↪count()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
```

[120]:
```
2079
```

[126]:
```
w=0
for i in c:
  cotable.iloc[w,1]=cleantest.loc[cleantest['score']>i,'score'].count()/15000
  cotable.iloc[w,2]=((1-cleantest.
↪loc[cleantest['score']>i,'TARGET_LABEL_BAD=1'].mean())*cleantest.
↪loc[cleantest['score']>i,'score'].count())/ttg
  cotable.iloc[w,3]=(cleantest.loc[cleantest['score']<i,'TARGET_LABEL_BAD=1'].
↪mean()*cleantest.loc[cleantest['score']<i,'score'].count())/(15000-ttg)
  cotable.iloc[w,4]=(cotable.iloc[w,2]*ttg+cotable.iloc[w,3]*(15000-ttg))/15000
  cotable.
↪iloc[w,5]=(cleantest[cleantest['TARGET_LABEL_BAD=1']==0][cleantest['score']>i]['PERSONAL_MO
↪sum()*0.32*(0.2/12))/ttg
  cotable.
↪iloc[w,6]=(cleantest[cleantest['TARGET_LABEL_BAD=1']==1][cleantest['score']>i]['PERSONAL_MO
↪sum()*0.32*(0.2/12))/(15000-ttg)
  cotable.iloc[w,7]=cotable.
↪iloc[w,5]*(cleantest[cleantest['TARGET_LABEL_BAD=1']==0][cleantest['score']<i]['score'].
↪count())
  cotable.iloc[w,8]=cotable.
↪iloc[w,6]*(cleantest[cleantest['TARGET_LABEL_BAD=1']==1][cleantest['score']>i]['score'].
↪count())
  cotable.iloc[w,9]=cotable.iloc[w,7]+cotable.iloc[w,8]
  w+=1
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
```

```
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
```

```
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
```

```
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
```

```
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
```

```
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
```

```
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
```

```
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
```

```
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  import sys
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  if __name__ == '__main__':
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: UserWarning:
Boolean Series key will be reindexed to match DataFrame index.
  # Remove the CWD from sys.path while we load stuff.
```

[128]:
```
cotable
```

[128]:

	cutoff	Accepted_rate	Accuracy_good	Accuracy_bad	Accuracy_Total	Avg_cost_g
0	520.0	0.9942	0.996392	0.0120143	0.739667	4.3945
1	530.0	0.975933	0.982413	0.0421779	0.7372	4.3413
2	540.0	0.958933	0.96645	0.0608384	0.730267	4.26847
3	550.0	0.947267	0.954816	0.0738753	0.725067	4.22516
4	560.0	0.902867	0.915404	0.132413	0.7112	4.09893
5	570.0	0.8476	0.862825	0.187883	0.6868	3.90422
6	580.0	0.812467	0.824766	0.22137	0.6674	3.56697
7	590.0	0.802133	0.8125	0.227249	0.659867	3.51917
8	600.0	0.802133	0.8125	0.227249	0.659867	3.51917
9	610.0	0.802133	0.8125	0.227249	0.659867	3.51917
10	620.0	0.802133	0.8125	0.227249	0.659867	3.51917
11	630.0	0.802133	0.8125	0.227249	0.659867	3.51917
12	640.0	0.802133	0.8125	0.227249	0.659867	3.51917
13	650.0	0.801933	0.8125	0.227761	0.66	3.51917
14	660.0	0.7864	0.799964	0.249489	0.6564	3.47876
15	670.0	0.742867	0.761364	0.307515	0.643	3.15289
16	680.0	0.701267	0.722763	0.350716	0.625733	2.97125
17	690.0	0.6726	0.694715	0.388548	0.614867	2.86865
18	700.0	0.508067	0.541126	0.582822	0.552	2.37293
19	710.0	0.2662	0.294913	0.787065	0.423267	1.31451
20	720.0	0.116733	0.131584	0.922035	0.337733	0.542966
21	730.0	0.051	0.0557359	0.962423	0.2922	0.216072
22	740.0	0.051	0.0557359	0.962423	0.2922	0.216072
23	750.0	0.051	0.0557359	0.962423	0.2922	0.216072
24	760.0	0.051	0.0557359	0.962423	0.2922	0.216072
25	770.0	0.051	0.0557359	0.962423	0.2922	0.216072
26	780.0	0.051	0.0557359	0.962423	0.2922	0.216072
27	790.0	0.0509333	0.0556457	0.962423	0.292133	0.215921
28	800.0	0.0466	0.0515873	0.96728	0.2904	0.181332
29	810.0	0.0291333	0.0330988	0.978016	0.279533	0.118773
30	820.0	0.0124	0.0147006	0.992843	0.2698	0.0535452
31	830.0	0.00333333	0.00414863	0.998978	0.2636	0.0130791

	Avg_cost_b	total_cost_g	total_cost_b	total_cost
0	3.8343	136.23	14819.6	14955.8
1	3.71781	833.529	13926.9	14760.4
2	3.63509	1532.38	13333.5	14865.9
3	3.59315	2083	13014.4	15097.4
4	3.39594	3828.4	11522.4	15350.8
5	3.14824	5610.36	9907.52	15517.9
6	3.0263	6869.99	9206.02	16076
7	3.00787	7316.36	9092.8	16409.2
8	3.00787	7316.36	9092.8	16409.2
9	3.00787	7316.36	9092.8	16409.2

10	3.00787	7316.36	9092.8	16409.2
11	3.00787	7316.36	9092.8	16409.2
12	3.00787	7316.36	9092.8	16409.2
13	3.00617	7316.36	9078.62	16395
14	2.92889	7663.72	8569.92	16233.6
15	2.73478	8314.18	7386.65	15700.8
16	2.55644	8982.08	6403.89	15386
17	2.4644	9644.42	5880.06	15524.5
18	1.72362	12021.3	2793.99	14815.3
19	0.67466	9766.84	487.779	10254.6
20	0.265061	5170.12	77.3979	5247.52
21	0.118407	2262.27	17.4058	2279.68
22	0.118407	2262.27	17.4058	2279.68
23	0.118407	2262.27	17.4058	2279.68
24	0.118407	2262.27	17.4058	2279.68
25	0.118407	2262.27	17.4058	2279.68
26	0.118407	2262.27	17.4058	2279.68
27	0.118407	2260.69	17.4058	2278.1
28	0.105433	1906.52	13.39	1919.91
29	0.0639326	1268.74	4.47528	1273.21
30	0.0278808	583.857	0.641258	584.498
31	0.00253175	144.419	0.010127	144.429

[ ]: