project

December 28, 2017

1 0. Project Overview

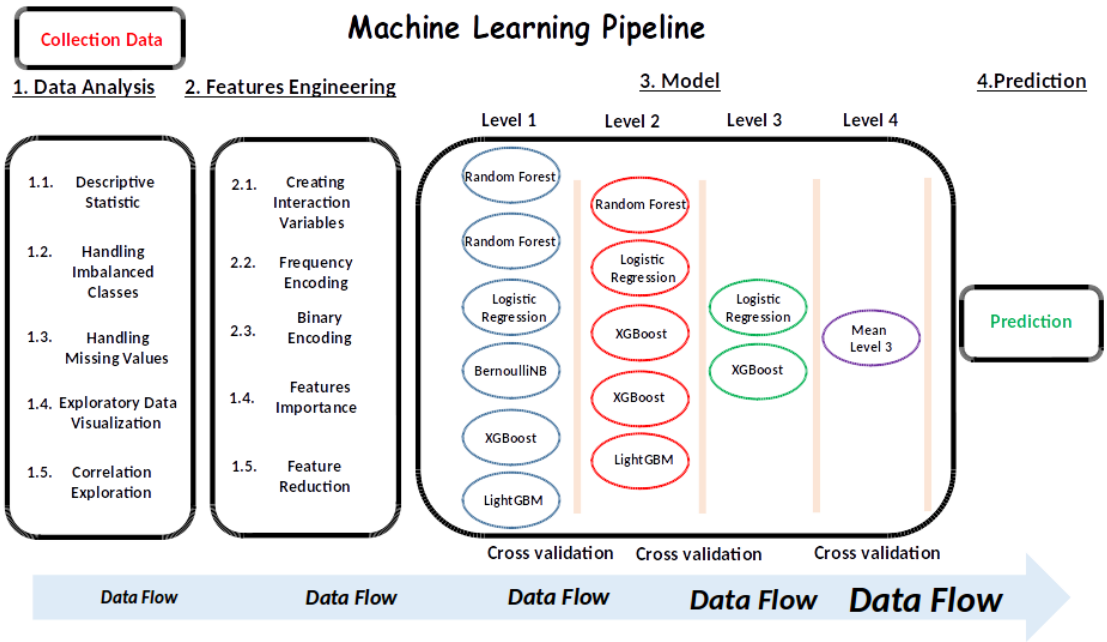
1.1 Porto Seguro's Safe Driver Prediction

1.1.1 Predict if a driver will file an insurance claim next year.

1.2 0.1.Project road map

```
In [1]: from IPython.display import Image
        Image(filename='drawing.png')
```

Out[1]:



2 1. Data Analysis

2.1 1.1. Descriptive statistics

2.1.1 1.1.1 Loading packages

```
In [2]: # Python libarary ################################################
        import sys
        import os
        import pandas as pd
        import numpy as np
        from IPython.display import display
        import seaborn as sns
        import matplotlib.pyplot as plt
        % matplotlib inline

        ###################################################################

In [3]: #
        import analysis.traintest.traintest as TT
        import analysis.missingvalue.missingvalue as MV
        import analysis.featureAnalysis.featureAnalysis as FA
        import analysis.multifeaurecomparision.multicomparision as MC
        import preparing.categoricalEncoding as PCAT
        import preparing.sampling as SA
        import preparing.featuressel as FSEL
        import model.model as ML
        ###################################################################

In [4]: # putting apple on the search path -- robust method
        sys.path.insert(0, os.path.join(os.path.split('__file__')[0], '..'))
```

2.1.2 1.1.2. Loading Data

```
In [5]: def LoadingData(path_to_train, path_to_test):
            try:
                train = pd.read_csv(path_to_train)
                test = pd.read_csv(path_to_test)
                return train, test
            except FileNotFoundError:
                print("The path does not exist")

In [6]: df_train, df_test = LoadingData('./data/train.csv', './data/test.csv')
```

2.1.3 1.1.3. Data at First Sight

Here is an excerpt of the the data description for the competition:

- Features that belong to **similar groupings are tagged** as such in the feature names (e.g., ind, reg, car, calc).

- Feature names include the postfix **bin** to indicate binary features and cat to indicate categorical features.
- Features **without these designations are either continuous or ordinal**.
- Values of **-1** indicate that the feature was missing from the observation.
- The **target** columns signifies whether or not a claim was filed for that policy holder.

```
In [7]: print("The shape of train is: ", df_train.shape)
        print("The shape of test is: ", df_test.shape)
```

```
The shape of train is:  (595212, 59)
The shape of test is:  (892816, 58)
```

```
In [8]: tt = TT.TrainTestAnalysis(df_train, df_test)
        tt.TrainTestCompare()
```

```
We have 595212 training rows and 892816 test rows.
We have 59 training columns and 58 test columns.
Train and test sets are distinct.
There is no NaN
```

```
In [9]: df_train.head()
```

Out[9]:

	id	target	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	\
0	7	0	2	2	5	1	
1	9	0	1	1	7	0	
2	13	0	5	4	9	1	
3	16	0	0	1	2	0	
4	17	0	0	2	0	1	

	ps_ind_05_cat	ps_ind_06_bin	ps_ind_07_bin	ps_ind_08_bin	...	\
0	0	0	1	0	...	
1	0	0	0	1	...	
2	0	0	0	1	...	
3	0	1	0	0	...	
4	0	1	0	0	...	

	ps_calc_11	ps_calc_12	ps_calc_13	ps_calc_14	ps_calc_15_bin	\
0	9	1	5	8	0	
1	3	1	1	9	0	
2	4	2	7	7	0	
3	2	2	4	9	0	
4	3	1	1	3	0	

	ps_calc_16_bin	ps_calc_17_bin	ps_calc_18_bin	ps_calc_19_bin	\
0	1	1	0	0	
1	1	1	0	1	
2	1	1	0	1	

```
3               0               0               0               0
4               0               0               1               1

   ps_calc_20_bin
0               1
1               0
2               0
3               0
4               0

[5 rows x 59 columns]
```

```
In [10]: df_test.head()
```

```
Out[10]:    id  ps_ind_01  ps_ind_02_cat  ps_ind_03  ps_ind_04_cat  ps_ind_05_cat  \
         0   0          0              1          8              1              0
         1   1          4              2          5              1              0
         2   2          5              1          3              0              0
         3   3          0              1          6              0              0
         4   4          5              1          7              0              0

            ps_ind_06_bin  ps_ind_07_bin  ps_ind_08_bin  ps_ind_09_bin       ...        \
         0              0              1              0              0       ...
         1              0              0              0              1       ...
         2              0              0              0              1       ...
         3              1              0              0              0       ...
         4              0              0              0              1       ...

            ps_calc_11  ps_calc_12  ps_calc_13  ps_calc_14  ps_calc_15_bin  \
         0           1           1           1          12               0
         1           2           0           3          10               0
         2           4           0           2           4               0
         3           5           1           0           5               1
         4           4           0           0           4               0

            ps_calc_16_bin  ps_calc_17_bin  ps_calc_18_bin  ps_calc_19_bin  \
         0               1               1               0               0
         1               0               1               1               0
         2               0               0               0               0
         3               0               1               0               0
         4               1               1               0               0

            ps_calc_20_bin
         0               1
         1               1
         2               0
         3               0
         4               1
```

```
[5 rows x 58 columns]
```

We indeed see the following

- binary variables
- categorical variables of which the category values are integers
- other variables with integer or float values
- variables with -1 representing missing values
- the target variable and an ID variable

2.1.4 1.1.4. Descriptive Statistics

** Meta Data:** To facilitate the data management, we'll store meta-information about the variables in a DataFrame. This will be helpful when we want to select specific variables for analysis, visualization, modeling, ...

We can also apply the describe method on the dataframe. However, it doesn't make much sense to calculate the mean, std, ... on categorical variables and the id variable. We'll explore the categorical variables visually later.

Thanks to our meta file we can easily select the variables on which we want to compute the descriptive statistics. To keep things clear, we'll do this per data type.

```
In [11]: fa = FA.DataExploration(df_train)
         feature_inofrmation = fa.DesAnalysis(True)
```

	Features	Dtype	Nunique	nduplicate	freq1	freq1_val	freq2	\
0	id	int64	595212	0	1050623	1	492634	
1	target	int64	2	595210	0	573518	1	
2	ps_ind_01	int64	8	595204	0	187594	1	
3	ps_ind_02_cat	int64	5	595207	1	431859	2	
4	ps_ind_03	int64	12	595200	2	96110	3	
5	ps_ind_04_cat	int64	3	595209	0	346965	1	
6	ps_ind_05_cat	int64	8	595204	0	528009	6	
7	ps_ind_06_bin	int64	2	595210	0	360852	1	
8	ps_ind_07_bin	int64	2	595210	0	442223	1	
9	ps_ind_08_bin	int64	2	595210	0	497644	1	
10	ps_ind_09_bin	int64	2	595210	0	484917	1	
11	ps_ind_10_bin	int64	2	595210	0	594990	1	
12	ps_ind_11_bin	int64	2	595210	0	594205	1	
13	ps_ind_12_bin	int64	2	595210	0	589594	1	
14	ps_ind_13_bin	int64	2	595210	0	594648	1	
15	ps_ind_14	int64	5	595207	0	588832	1	
16	ps_ind_15	int64	14	595198	7	65336	8	
17	ps_ind_16_bin	int64	2	595210	1	393330	0	
18	ps_ind_17_bin	int64	2	595210	0	523143	1	
19	ps_ind_18_bin	int64	2	595210	0	503879	1	
20	ps_reg_01	float64	10	595202	0.9	194608	0.7	
21	ps_reg_02	float64	19	595193	0.2	114886	0.3	
22	ps_reg_03	float64	5013	590199	-1	107772	0.633936	

23	ps_car_01_cat	int64	13	595199	11	207573	7
24	ps_car_02_cat	int64	3	595209	1	493990	0
25	ps_car_03_cat	int64	3	595209	-1	411231	1
26	ps_car_04_cat	int64	10	595202	0	496581	1
27	ps_car_05_cat	int64	3	595209	-1	266551	1
28	ps_car_06_cat	int64	18	595194	11	131527	1
29	ps_car_07_cat	int64	3	595209	1	553148	0
30	ps_car_08_cat	int64	2	595210	1	495264	0
31	ps_car_09_cat	int64	6	595206	2	353482	0
32	ps_car_10_cat	int64	3	595209	1	590179	0
33	ps_car_11_cat	int64	104	595108	104	85083	103
34	ps_car_11	int64	5	595207	3	318919	2
35	ps_car_12	float64	184	595028	0.316228	170579	0.4
36	ps_car_13	float64	70482	524730	0.674583	386	0.741689
37	ps_car_14	float64	850	594362	-1	42620	0.361525
38	ps_car_15	float64	15	595197	3.60555	109765	3.4641
39	ps_calc_01	float64	10	595202	0.6	59837	0
40	ps_calc_02	float64	10	595202	0.5	60070	0.4
41	ps_calc_03	float64	10	595202	0.1	60036	0.5
42	ps_calc_04	int64	6	595206	2	193977	3
43	ps_calc_05	int64	7	595205	2	195160	1
44	ps_calc_06	int64	11	595201	8	175015	7
45	ps_calc_07	int64	10	595202	3	162414	2
46	ps_calc_08	int64	11	595201	9	151746	10
47	ps_calc_09	int64	8	595204	2	182519	3
48	ps_calc_10	int64	26	595186	8	82043	7
49	ps_calc_11	int64	20	595192	5	102512	4
50	ps_calc_12	int64	11	595201	1	203280	2
51	ps_calc_13	int64	14	595198	2	139334	3
52	ps_calc_14	int64	24	595188	7	86673	8
53	ps_calc_15_bin	int64	2	595210	0	522342	1
54	ps_calc_16_bin	int64	2	595210	1	373698	0
55	ps_calc_17_bin	int64	2	595210	1	329856	0
56	ps_calc_18_bin	int64	2	595210	0	424278	1
57	ps_calc_19_bin	int64	2	595210	0	387469	1
58	ps_calc_20_bin	int64	2	595210	0	503955	1

	freq2_val	freq3	freq3_val	mean	std	min	\
0	1	320566	1	743803.558435	429367.820429	7.000000	
1	21694	NaN	NaN	0.036448	0.187401	0.000000	
2	143984	2	82468	1.900378	1.983789	0.000000	
3	123573	3	28186	1.358943	0.664594	-1.000000	
4	81973	1	67994	4.423318	2.699902	0.000000	
5	248164	-1	83	0.416794	0.493311	-1.000000	
6	20662	4	18344	0.405188	1.350642	-1.000000	
7	234360	NaN	NaN	0.393742	0.488579	0.000000	
8	152989	NaN	NaN	0.257033	0.436998	0.000000	
9	97568	NaN	NaN	0.163921	0.370205	0.000000	

10	110295	NaN	NaN	0.185304	0.388544	0.000000
11	222	NaN	NaN	0.000373	0.019309	0.000000
12	1007	NaN	NaN	0.001692	0.041097	0.000000
13	5618	NaN	NaN	0.009439	0.096693	0.000000
14	564	NaN	NaN	0.000948	0.030768	0.000000
15	5495	2	744	0.012451	0.127545	0.000000
16	59600	6	58408	7.299922	3.546042	0.000000
17	201882	NaN	NaN	0.660823	0.473430	0.000000
18	72069	NaN	NaN	0.121081	0.326222	0.000000
19	91333	NaN	NaN	0.153446	0.360417	0.000000
20	67897	0.8	60277	0.610991	0.287643	0.000000
21	95033	0	89297	0.439184	0.404264	0.000000
22	664	0.602599	637	0.551102	0.793506	-1.000000
23	179247	6	62393	8.295933	2.508270	-1.000000
24	101217	-1	5	0.829931	0.375716	-1.000000
25	110709	0	73272	-0.504899	0.788654	-1.000000
26	32115	2	23770	0.725192	2.153463	0.000000
27	172667	0	155994	-0.157732	0.844417	-1.000000
28	118386	0	110420	6.555340	5.501445	0.000000
29	30575	-1	11489	0.910027	0.347106	-1.000000
30	99948	NaN	NaN	0.832080	0.373796	0.000000
31	194518	1	29080	1.328890	0.978747	-1.000000
32	4857	2	176	0.992136	0.091619	0.000000
33	24262	64	22278	62.215674	33.012455	1.000000
34	189353	1	60952	2.346072	0.832548	-1.000000
35	111873	0.374166	98598	0.379945	0.058327	-1.000000
36	377	0.692776	363	0.813265	0.224588	0.250619
37	17696	0.358329	15523	0.276256	0.357154	-1.000000
38	77200	3.31662	68737	3.065899	0.731366	0.000000
39	59780	0.8	59710	0.449756	0.287198	0.000000
40	59823	0	59618	0.449589	0.286893	0.000000
41	59832	0.3	59819	0.449849	0.287153	0.000000
42	175512	1	108012	2.372081	1.117219	0.000000
43	170860	3	119192	1.885886	1.134927	0.000000
44	139771	9	129207	7.689445	1.334312	0.000000
45	139101	4	122039	3.005823	1.414564	0.000000
46	151330	8	102944	9.225904	1.459672	2.000000
47	152829	1	121391	2.339034	1.246949	0.000000
48	78009	9	76772	8.433590	2.904597	0.000000
49	93890	6	93110	5.441382	2.332871	0.000000
50	145321	0	141001	1.441918	1.202963	0.000000
51	132412	1	96661	2.872288	1.694887	0.000000
52	82177	6	80897	7.539026	2.746652	0.000000
53	72870	NaN	NaN	0.122427	0.327779	0.000000
54	221514	NaN	NaN	0.627840	0.483381	0.000000
55	265356	NaN	NaN	0.554182	0.497056	0.000000
56	170934	NaN	NaN	0.287182	0.452447	0.000000
57	207743	NaN	NaN	0.349024	0.476662	0.000000

```
58     91257        NaN        NaN        0.153318        0.360295  0.000000
```

	25%	50%	75%	max
0	371991.500000	743547.500000	1.115549e+06	1.488027e+06
1	0.000000	0.000000	0.000000e+00	1.000000e+00
2	0.000000	1.000000	3.000000e+00	7.000000e+00
3	1.000000	1.000000	2.000000e+00	4.000000e+00
4	2.000000	4.000000	6.000000e+00	1.100000e+01
5	0.000000	0.000000	1.000000e+00	1.000000e+00
6	0.000000	0.000000	0.000000e+00	6.000000e+00
7	0.000000	0.000000	1.000000e+00	1.000000e+00
8	0.000000	0.000000	1.000000e+00	1.000000e+00
9	0.000000	0.000000	0.000000e+00	1.000000e+00
10	0.000000	0.000000	0.000000e+00	1.000000e+00
11	0.000000	0.000000	0.000000e+00	1.000000e+00
12	0.000000	0.000000	0.000000e+00	1.000000e+00
13	0.000000	0.000000	0.000000e+00	1.000000e+00
14	0.000000	0.000000	0.000000e+00	1.000000e+00
15	0.000000	0.000000	0.000000e+00	4.000000e+00
16	5.000000	7.000000	1.000000e+01	1.300000e+01
17	0.000000	1.000000	1.000000e+00	1.000000e+00
18	0.000000	0.000000	0.000000e+00	1.000000e+00
19	0.000000	0.000000	0.000000e+00	1.000000e+00
20	0.400000	0.700000	9.000000e-01	9.000000e-01
21	0.200000	0.300000	6.000000e-01	1.800000e+00
22	0.525000	0.720677	1.000000e+00	4.037945e+00
23	7.000000	7.000000	1.100000e+01	1.100000e+01
24	1.000000	1.000000	1.000000e+00	1.000000e+00
25	-1.000000	-1.000000	0.000000e+00	1.000000e+00
26	0.000000	0.000000	0.000000e+00	9.000000e+00
27	-1.000000	0.000000	1.000000e+00	1.000000e+00
28	1.000000	7.000000	1.100000e+01	1.700000e+01
29	1.000000	1.000000	1.000000e+00	1.000000e+00
30	1.000000	1.000000	1.000000e+00	1.000000e+00
31	0.000000	2.000000	2.000000e+00	4.000000e+00
32	1.000000	1.000000	1.000000e+00	2.000000e+00
33	32.000000	65.000000	9.300000e+01	1.040000e+02
34	2.000000	3.000000	3.000000e+00	3.000000e+00
35	0.316228	0.374166	4.000000e-01	1.264911e+00
36	0.670867	0.765811	9.061904e-01	3.720626e+00
37	0.333167	0.368782	3.964846e-01	6.363961e-01
38	2.828427	3.316625	3.605551e+00	3.741657e+00
39	0.200000	0.500000	7.000000e-01	9.000000e-01
40	0.200000	0.400000	7.000000e-01	9.000000e-01
41	0.200000	0.500000	7.000000e-01	9.000000e-01
42	2.000000	2.000000	3.000000e+00	5.000000e+00
43	1.000000	2.000000	3.000000e+00	6.000000e+00
44	7.000000	8.000000	9.000000e+00	1.000000e+01

```
45      2.000000      3.000000  4.000000e+00  9.000000e+00
46      8.000000      9.000000  1.000000e+01  1.200000e+01
47      1.000000      2.000000  3.000000e+00  7.000000e+00
48      6.000000      8.000000  1.000000e+01  2.500000e+01
49      4.000000      5.000000  7.000000e+00  1.900000e+01
50      1.000000      1.000000  2.000000e+00  1.000000e+01
51      2.000000      3.000000  4.000000e+00  1.300000e+01
52      6.000000      7.000000  9.000000e+00  2.300000e+01
53      0.000000      0.000000  0.000000e+00  1.000000e+00
54      0.000000      1.000000  1.000000e+00  1.000000e+00
55      0.000000      1.000000  1.000000e+00  1.000000e+00
56      0.000000      0.000000  1.000000e+00  1.000000e+00
57      0.000000      0.000000  1.000000e+00  1.000000e+00
58      0.000000      0.000000  0.000000e+00  1.000000e+00
```

** reg variables** - only ps_reg_03 has missing values - the range (min to max) differs between the variables. We could apply scaling (e.g. StandardScaler), but it depends on the classifier we will want to use.

** car variables** - ps_car_12 and ps_car_15 have missing values - again, the range differs and we could apply scaling.

** calc variables** - no missing values - this seems to be some kind of ratio as the maximum is 0.9 - all three _calc variables have very similar distributions

** Ordinal variables** - Only one missing variable: ps_car_11 - We could apply scaling to deal with the different ranges

** Binary variables ** - A priori in the train data is 3.645%, which is strongly imbalanced. - From the means we can conclude that for most variables the value is zero in most cases.

** Target ** - A priori in the train data is 3.645%, which is strongly imbalanced. - From the means we can conclude that for most variables the value is zero in most cases.

2.2 1.2. Handling imbalanced classes

As we mentioned above the proportion of records with target=1 is far less than target=0. This can lead to a model that has great accuracy but does have any added value in practice. Two possible strategies to deal with this problem are:

oversampling records with target=1 undersampling records with target=0

2.2.1 1.2.1. Plot Target

```
In [12]: fa.PlotBinary(df_train,['target'])
```

2.2.2 1.2.3. Sampeling Data

As we have a rather large training set, we can go for undersampling.

```
In [13]: sa = SA.Sampling(df_train)
         train = sa.UnderSampling('target', .1)
```

```
Rate to undersample records with target=0: 0.34043569687437886
Number of records with target=0 after undersampling: 195246
```

```
In [14]: # Sample analysis
         sfa = FA.DataExploration(train)
         sample_feature_inofrmation = sfa.DesAnalysis(False)
```

```
In [15]: print('Percentage of {0} in train {1}'.format(feature_inofrmation['freq1'][1],
                                                      feature_inofrmation['freq1_val'][1]/feature_
         print('Percentage of {0} in train {1}'.format(feature_inofrmation['freq2'][1],
                                                      feature_inofrmation['freq2_val'][1]/feature_
         print('Percentage of {0} in sample train {1}'.format(feature_inofrmation['freq1'][1],
                                                             sample_feature_inofrmation['
         print('Percentage of {0} in sample train {1}'.format(feature_inofrmation['freq2'][1],
                                                             sample_feature_inofrmation['freq2_
```

```
Percentage of 0 in train 0.963552482140817
Percentage of 1 in train 0.036447517859182946
Percentage of 0 in sample train 0.9
Percentage of 1 in sample train 0.1
```

```
In [16]: fa.PlotBinary(train,['target'])
```

2.3 1.3. Handling Missing Values

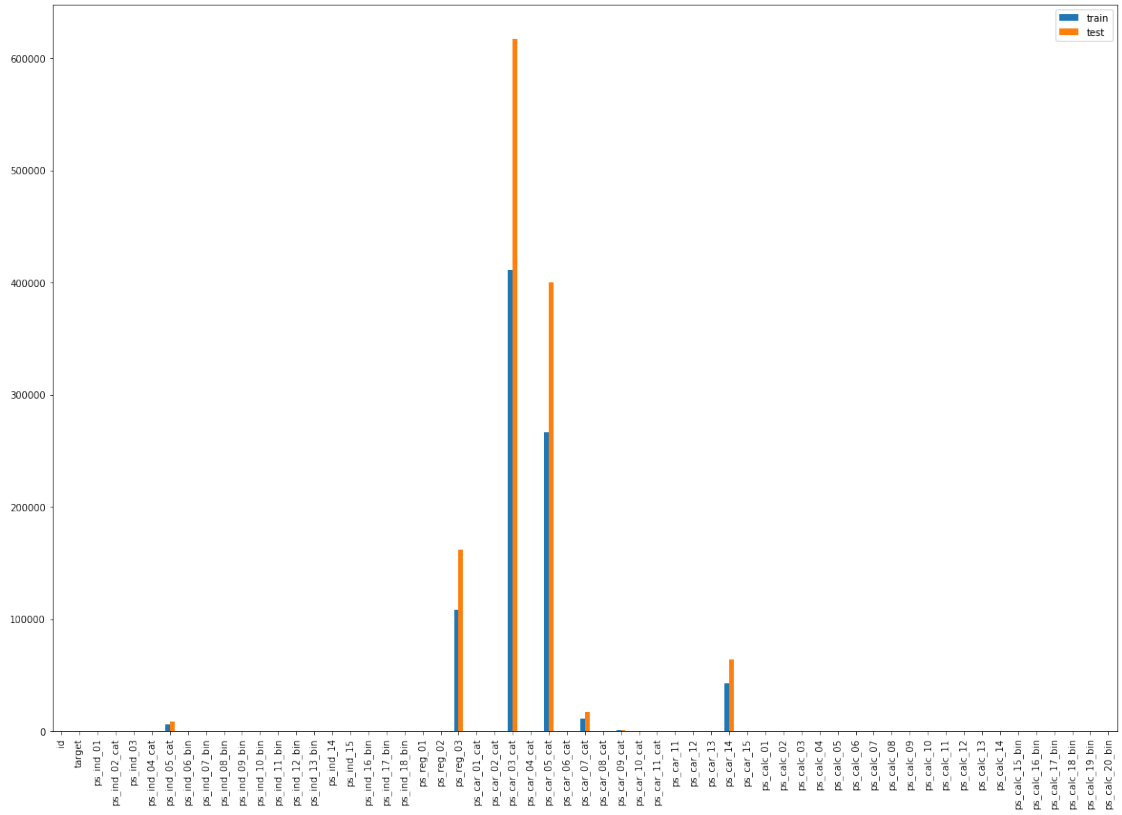
2.3.1 1.3.2. Missing Values Percentage

```
In [17]: # sample missing value
         missing = MV.MissingValue()
         print('-*-'*30)
         missing.DesMissingValue(df_train,-1)
```

```
-*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*--*-
Variable ps_ind_02_cat has 216 records (0.04%) with missing values
Variable ps_ind_04_cat has 83 records (0.01%) with missing values
Variable ps_ind_05_cat has 5809 records (0.98%) with missing values
Variable ps_reg_03 has 107772 records (18.11%) with missing values
Variable ps_car_01_cat has 107 records (0.02%) with missing values
Variable ps_car_02_cat has 5 records (0.00%) with missing values
Variable ps_car_03_cat has 411231 records (69.09%) with missing values
Variable ps_car_05_cat has 266551 records (44.78%) with missing values
Variable ps_car_07_cat has 11489 records (1.93%) with missing values
Variable ps_car_09_cat has 569 records (0.10%) with missing values
Variable ps_car_11 has 5 records (0.00%) with missing values
Variable ps_car_12 has 1 records (0.00%) with missing values
Variable ps_car_14 has 42620 records (7.16%) with missing values
In total, there are 13 variables with missing values
```

2.3.2 1.3.2. Plot MissingValue

```
In [18]: # train and test missing value
         tt.PlotTrainTestMissing(-1, np.nan)
```



- **ps_car_03_cat and ps_car_05_cat** have a large proportion of records with missing values. Remove these variables.
- For the other categorical variables with missing values, we can leave the missing value -1 as such.
- **ps_reg_03** (continuous) has missing values for 18% of all records. Replace by the mean.
- **ps_car_11** (ordinal) has only 5 records with misisng values. Replace by the mode.
- **ps_car_12** (continuous) has only 1 records with missing value. Replace by the mean.
- **ps_car_14** (continuous) has missing values for 7% of all records. Replace by the mean.

```
In [19]: from sklearn.preprocessing import Imputer
         # initiate test
         test = df_test
         # Dropping the variables with too many missing values

         vars_to_drop = ['ps_car_03_cat', 'ps_car_05_cat']
         train.drop(vars_to_drop, inplace=True, axis=1)
         test.drop(vars_to_drop, inplace=True, axis=1)
```

```
# Imputing with the mean or mode
mean_imp = Imputer(missing_values=-1, strategy='mean', axis=0)
mode_imp = Imputer(missing_values=-1, strategy='most_frequent', axis=0)
train['ps_reg_03'] = mean_imp.fit_transform(train[['ps_reg_03']]).ravel()
train['ps_car_12'] = mean_imp.fit_transform(train[['ps_car_12']]).ravel()
train['ps_car_14'] = mean_imp.fit_transform(train[['ps_car_14']]).ravel()
train['ps_car_11'] = mode_imp.fit_transform(train[['ps_car_11']]).ravel()
```

2.4 1.4. Exploratory Data Visualization

```
In [20]: print('Number of unique value: ',np.sort(feature_inofrmation.Nunique.unique()))
```

```
Number of unique value:  [2 3 5 6 7 8 10 11 12 13 14 15 18 19 20 24 26 104 184 850 5013 70482 59
```

```
In [21]: feature2 = fa.Describecat(2,True)
```

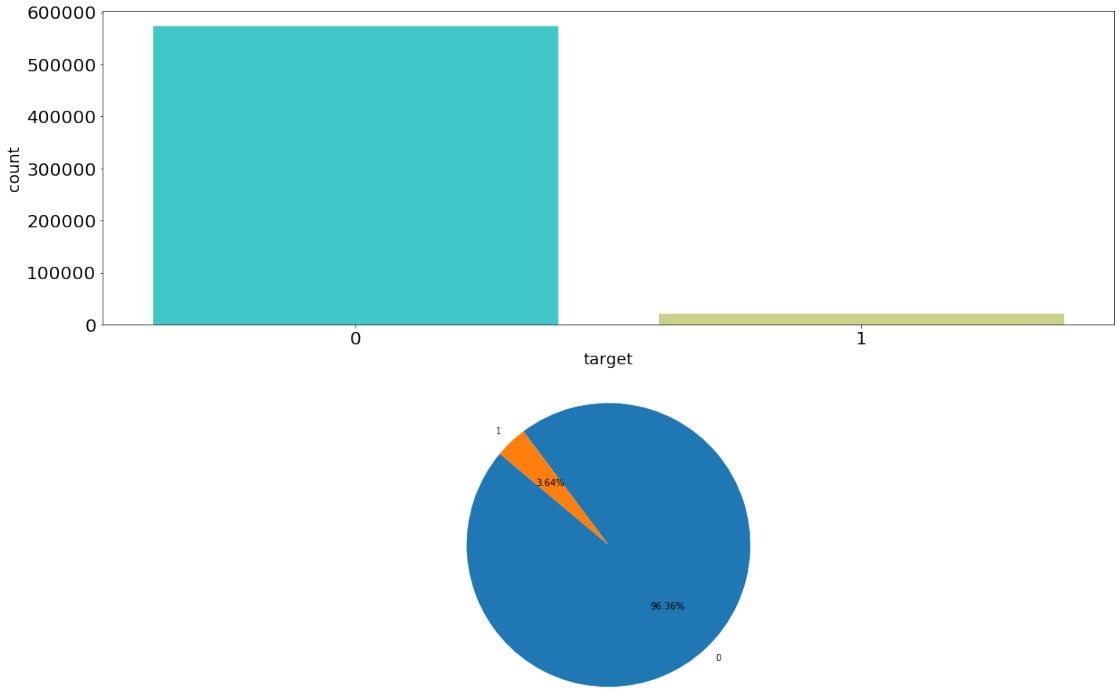
	Features	Dtype	Nunique	nduplicate	freq1	freq1_val	freq2	freq2_val	\
1	target	int64	2	595210	0	573518	1	21694	
7	ps_ind_06_bin	int64	2	595210	0	360852	1	234360	
8	ps_ind_07_bin	int64	2	595210	0	442223	1	152989	
9	ps_ind_08_bin	int64	2	595210	0	497644	1	97568	
10	ps_ind_09_bin	int64	2	595210	0	484917	1	110295	
11	ps_ind_10_bin	int64	2	595210	0	594990	1	222	
12	ps_ind_11_bin	int64	2	595210	0	594205	1	1007	
13	ps_ind_12_bin	int64	2	595210	0	589594	1	5618	
14	ps_ind_13_bin	int64	2	595210	0	594648	1	564	
17	ps_ind_16_bin	int64	2	595210	1	393330	0	201882	
18	ps_ind_17_bin	int64	2	595210	0	523143	1	72069	
19	ps_ind_18_bin	int64	2	595210	0	503879	1	91333	
30	ps_car_08_cat	int64	2	595210	1	495264	0	99948	
53	ps_calc_15_bin	int64	2	595210	0	522342	1	72870	
54	ps_calc_16_bin	int64	2	595210	1	373698	0	221514	
55	ps_calc_17_bin	int64	2	595210	1	329856	0	265356	
56	ps_calc_18_bin	int64	2	595210	0	424278	1	170934	
57	ps_calc_19_bin	int64	2	595210	0	387469	1	207743	
58	ps_calc_20_bin	int64	2	595210	0	503955	1	91257	

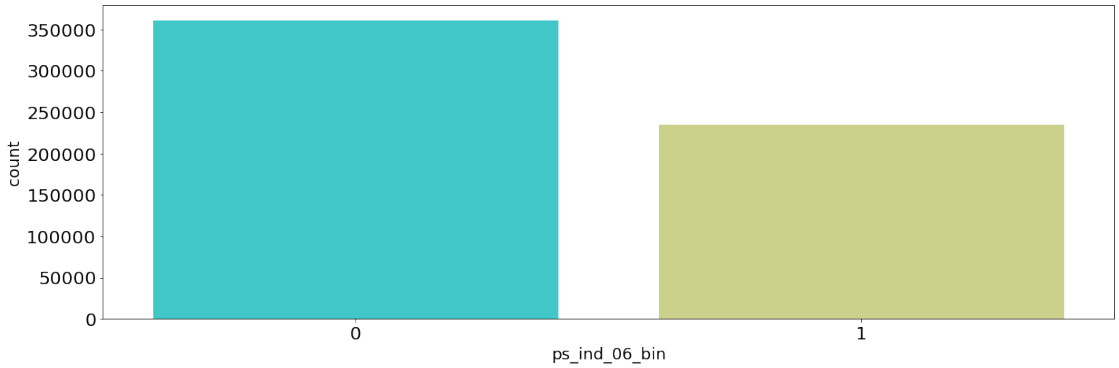
	freq3	freq3_val	mean	std	min	25%	50%	75%	max
1	NaN	NaN	0.036448	0.187401	0.0	0.0	0.0	0.0	1.0
7	NaN	NaN	0.393742	0.488579	0.0	0.0	0.0	1.0	1.0
8	NaN	NaN	0.257033	0.436998	0.0	0.0	0.0	1.0	1.0
9	NaN	NaN	0.163921	0.370205	0.0	0.0	0.0	0.0	1.0
10	NaN	NaN	0.185304	0.388544	0.0	0.0	0.0	0.0	1.0
11	NaN	NaN	0.000373	0.019309	0.0	0.0	0.0	0.0	1.0
12	NaN	NaN	0.001692	0.041097	0.0	0.0	0.0	0.0	1.0
13	NaN	NaN	0.009439	0.096693	0.0	0.0	0.0	0.0	1.0
14	NaN	NaN	0.000948	0.030768	0.0	0.0	0.0	0.0	1.0

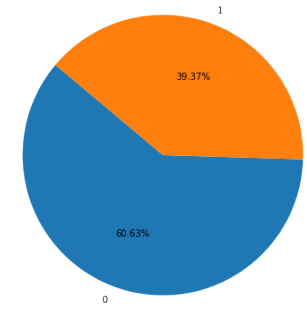
17	NaN	NaN	0.660823	0.473430	0.0	0.0	1.0	1.0	1.0
18	NaN	NaN	0.121081	0.326222	0.0	0.0	0.0	0.0	1.0
19	NaN	NaN	0.153446	0.360417	0.0	0.0	0.0	0.0	1.0
30	NaN	NaN	0.832080	0.373796	0.0	1.0	1.0	1.0	1.0
53	NaN	NaN	0.122427	0.327779	0.0	0.0	0.0	0.0	1.0
54	NaN	NaN	0.627840	0.483381	0.0	0.0	1.0	1.0	1.0
55	NaN	NaN	0.554182	0.497056	0.0	0.0	1.0	1.0	1.0
56	NaN	NaN	0.287182	0.452447	0.0	0.0	0.0	1.0	1.0
57	NaN	NaN	0.349024	0.476662	0.0	0.0	0.0	1.0	1.0
58	NaN	NaN	0.153318	0.360295	0.0	0.0	0.0	0.0	1.0

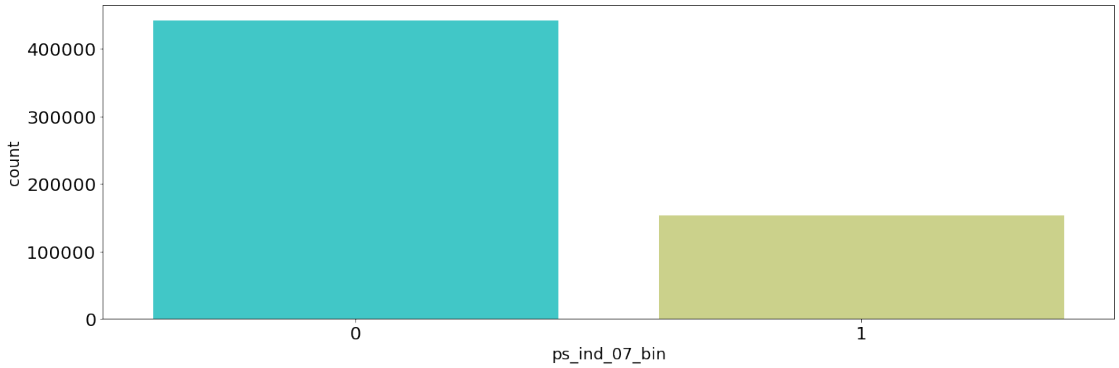
plot number of unique = 2

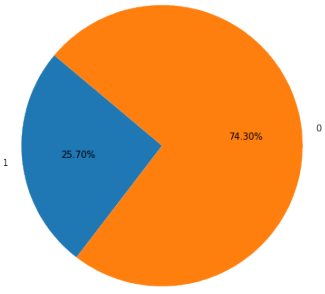
```
In [22]: fa.PlotCat(2)
```

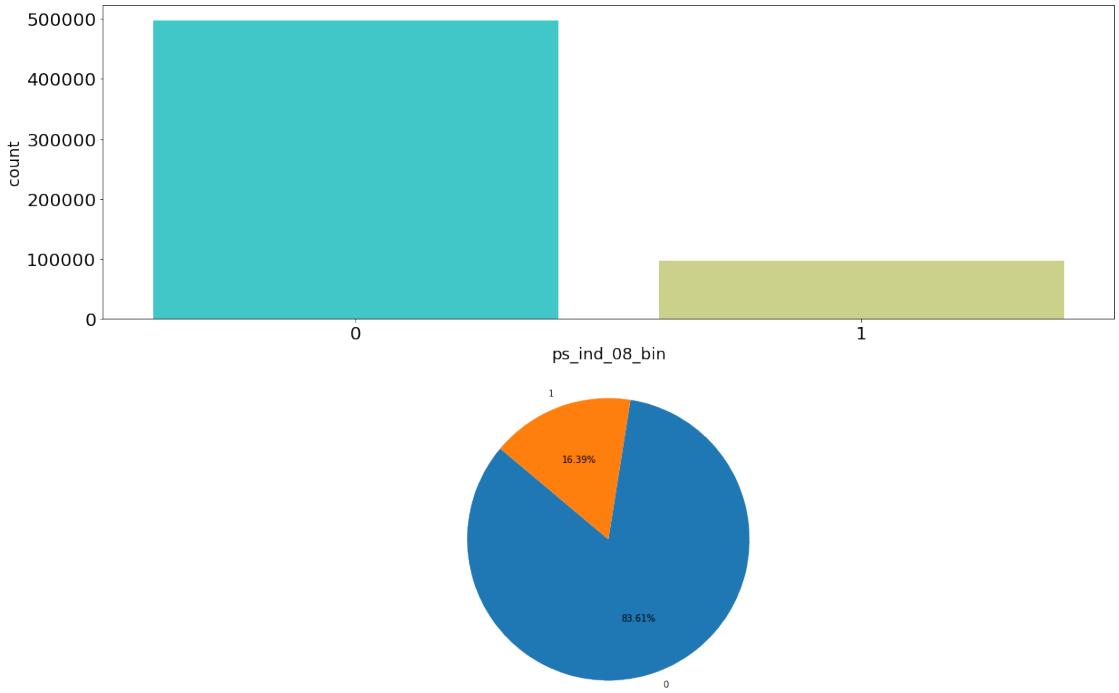


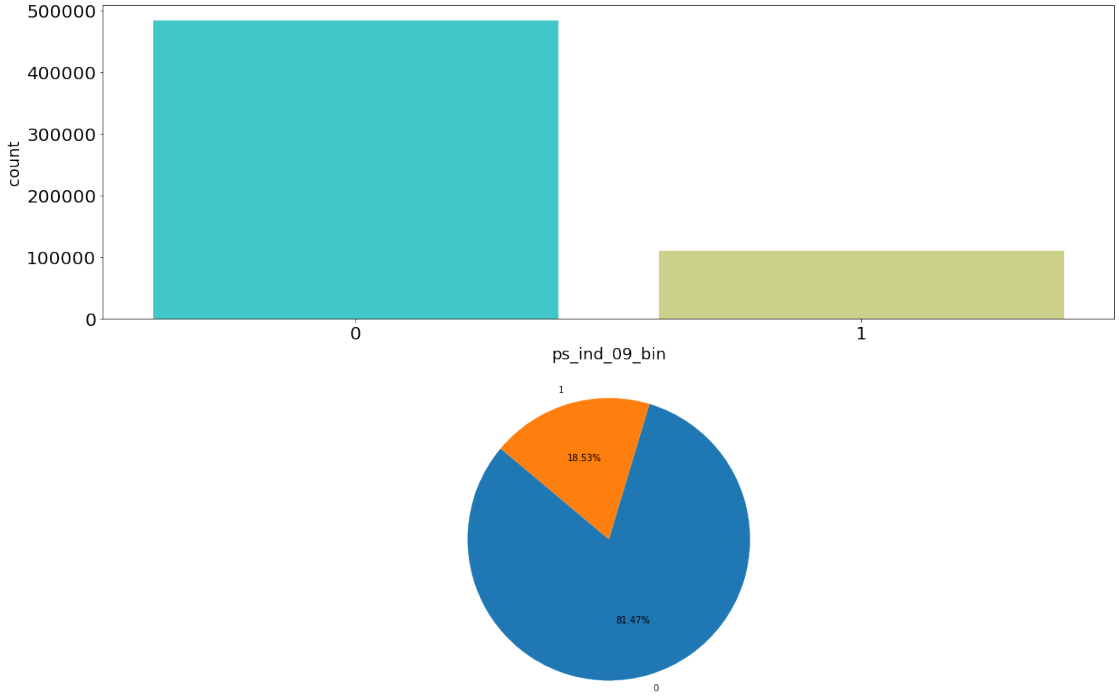

350000
300000
250000
200000
150000
100000
50000
0
count
0
1
ps_ind_06_bin

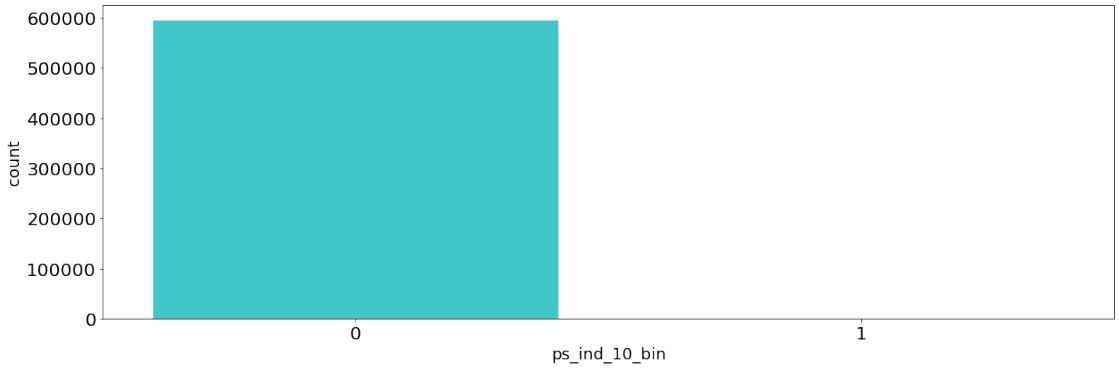

1
39.37%
60.63%
0

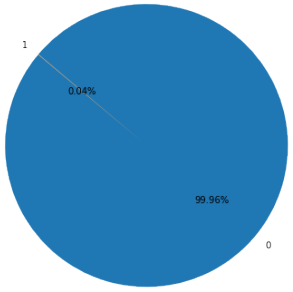

400000
300000
200000
100000
0
count
0
1
ps_ind_07_bin

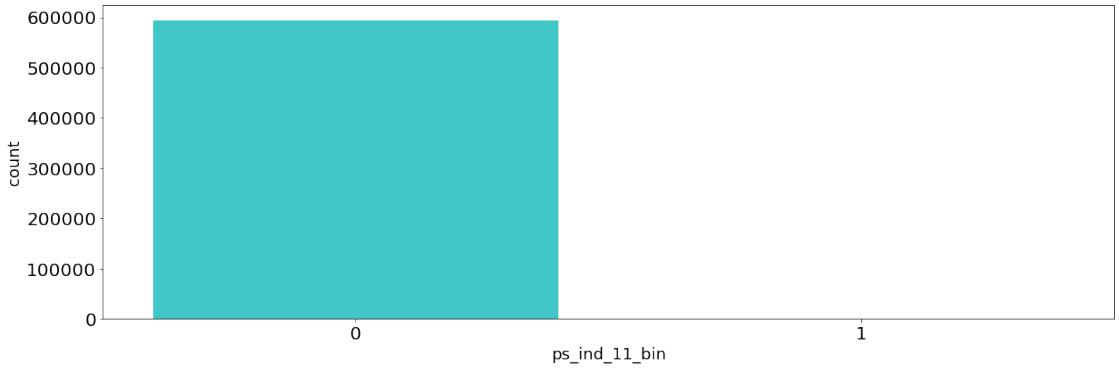

74.30%
0
25.70%
1

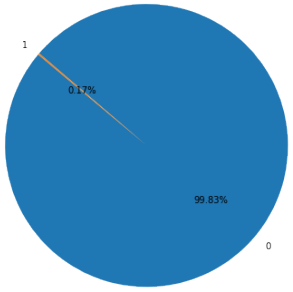

500000
400000
300000
200000
100000
0
count
0
1
ps_ind_08_bin
1
16.39%
83.61%
0

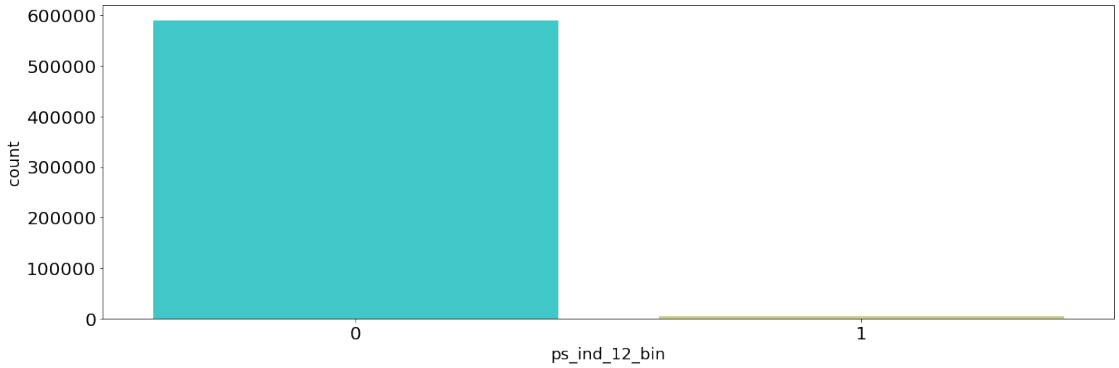

500000
400000
300000
200000
100000
0
count
0
1
ps_ind_09_bin
1
18.53%
81.47%
0

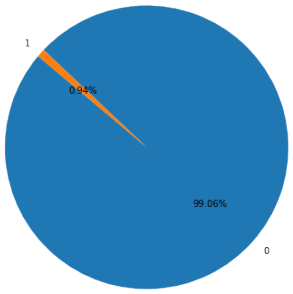

600000
500000
400000
300000
200000
100000
0
count
0
1
ps_ind_10_bin

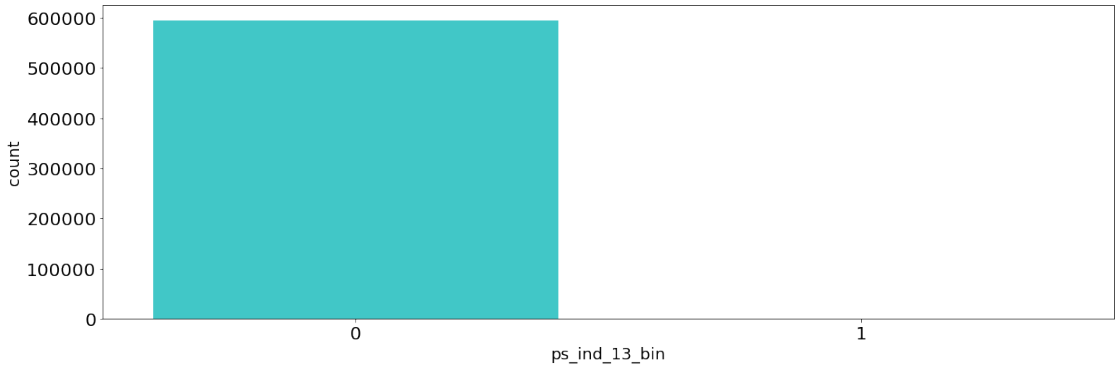

1
0.04%
99.96%
0

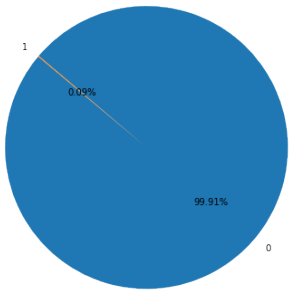

600000
500000
400000
300000
200000
100000
0
count
0
1
ps_ind_11_bin

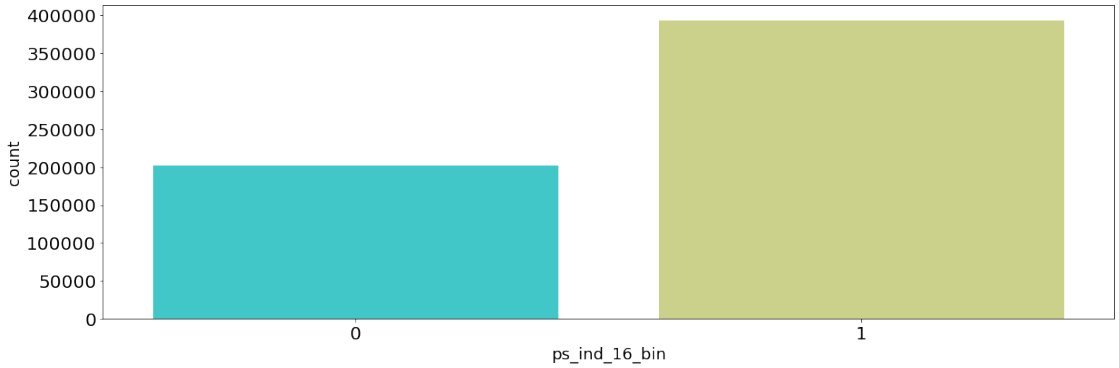

1
0.17%
99.83%
0

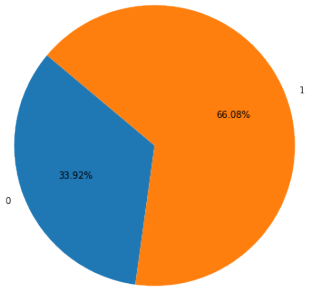

600000
500000
400000
300000
200000
100000
0
count
0
1
ps_ind_12_bin

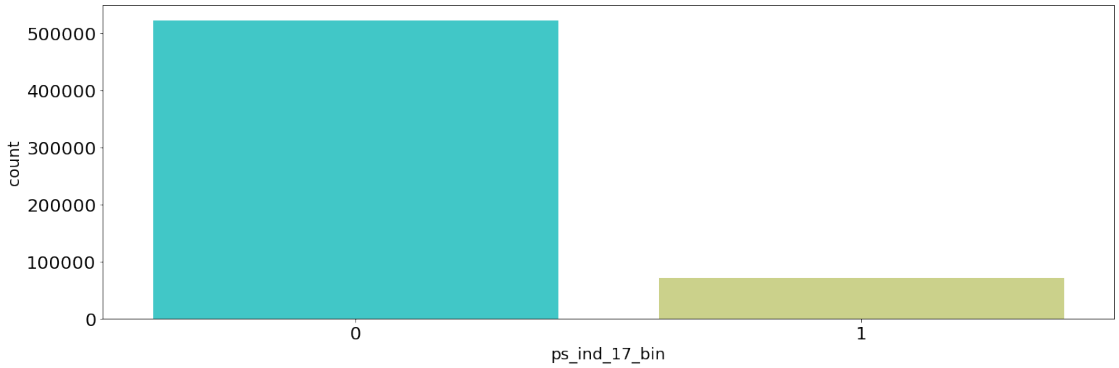

1
0.94%
99.06%
0

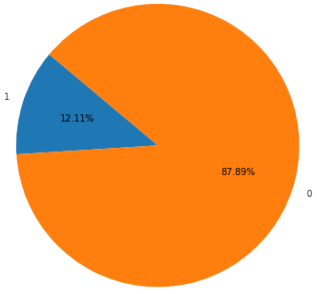

600000
500000
400000
300000
200000
100000
0
count
0
1
ps_ind_13_bin

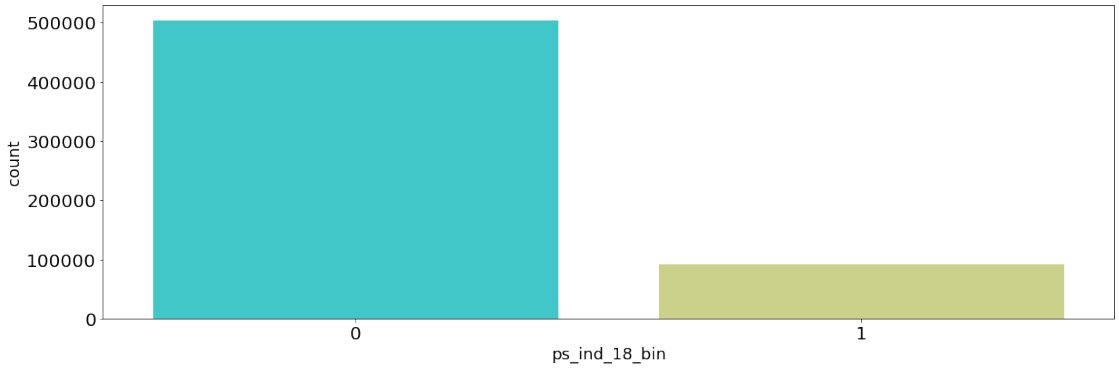

1
0.09%
99.91%
0

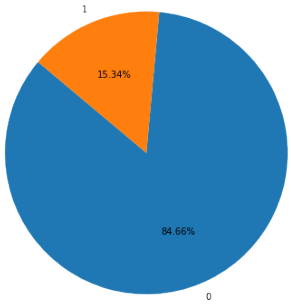

400000
350000
300000
250000
200000
150000
100000
50000
0
count
0
1
ps_ind_16_bin

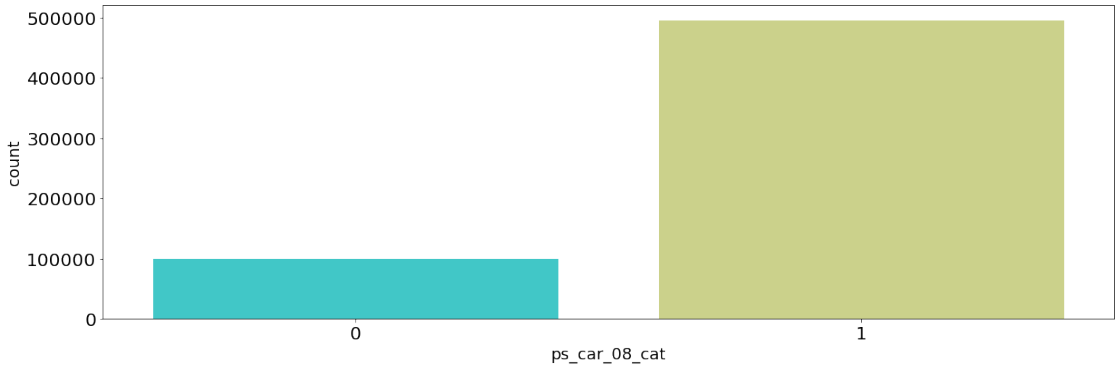

1
66.08%
33.92%
0

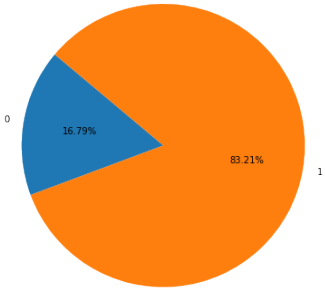

500000
400000
300000
200000
100000
0
count
0
1
ps_ind_17_bin

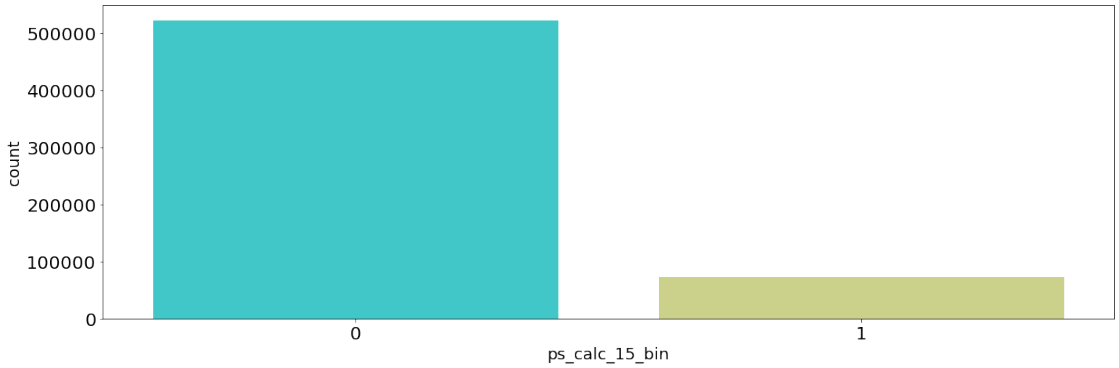

1
12.11%
87.89%
0

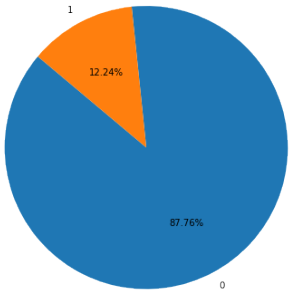

500000
400000
300000
200000
100000
0
count
0
1
ps_ind_18_bin

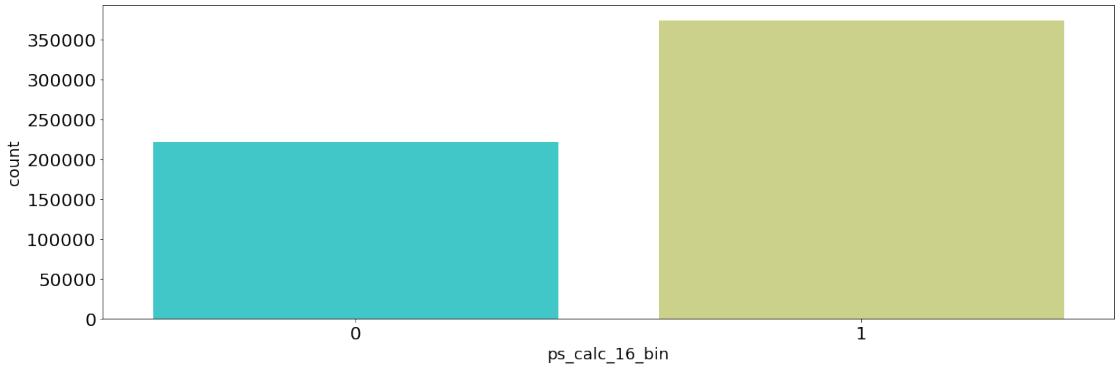

1
15.34%
84.66%
0

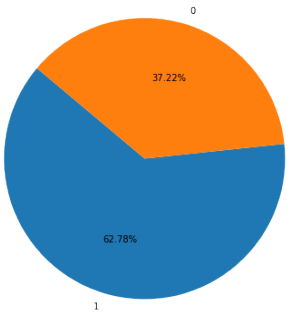

500000
400000
300000
200000
100000
0
count
0
1
ps_car_08_cat

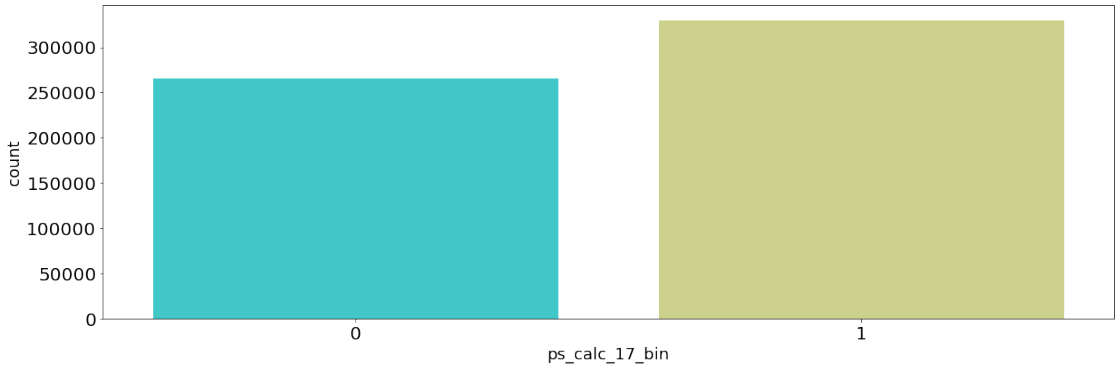

0
16.79%
83.21%
1

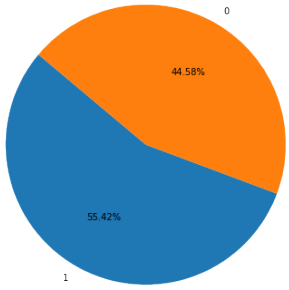

500000
400000
300000
200000
100000
0
count
0
1
ps_calc_15_bin

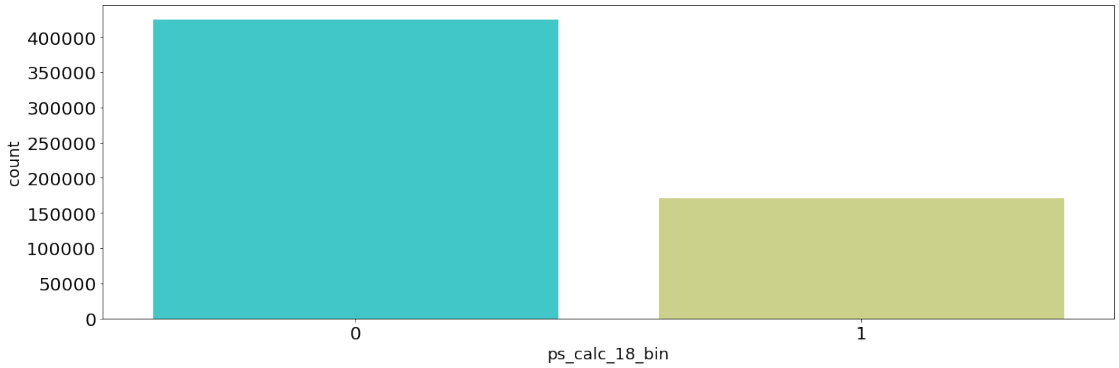

1
12.24%
87.76%
0

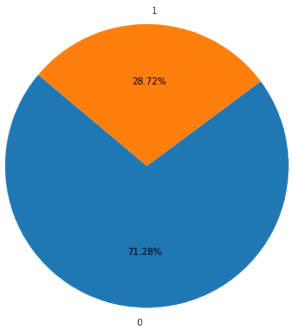

350000
300000
250000
200000
150000
100000
50000
0
count
0
1
ps_calc_16_bin

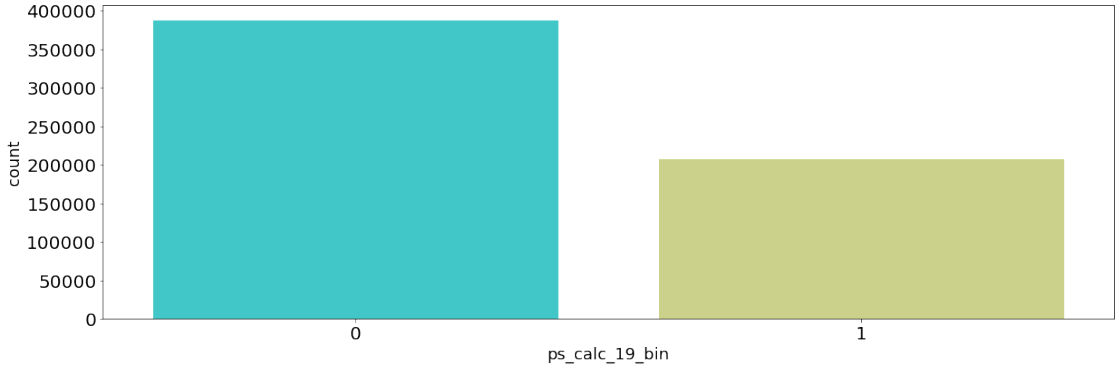

0
37.22%
62.78%
1

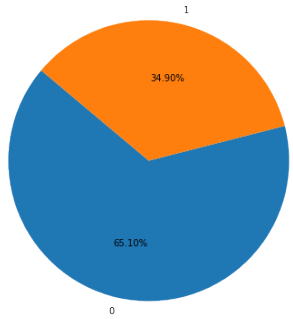

300000
250000
200000
150000
100000
50000
0
count
0
1
ps_calc_17_bin

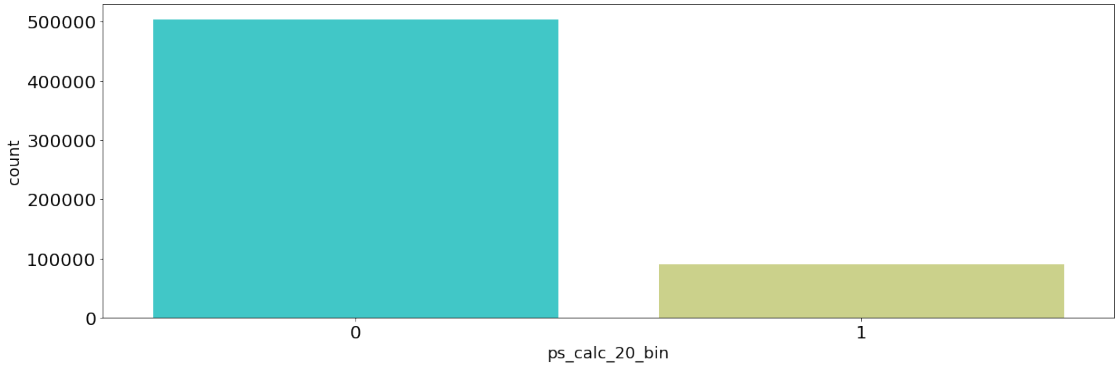

0
44.58%
55.42%
1

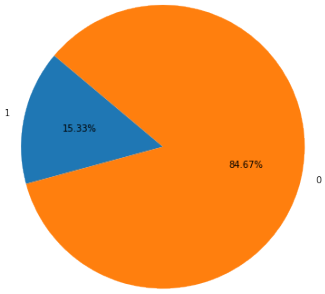

400000
350000
300000
250000
200000
150000
100000
50000
0
count
0
1
ps_calc_18_bin


1
28.72%
71.28%
0










```
In [23]: bin_col = [col for col in train.columns if '_bin' in col]
         fa.PlotBinary(train,bin_col)
```

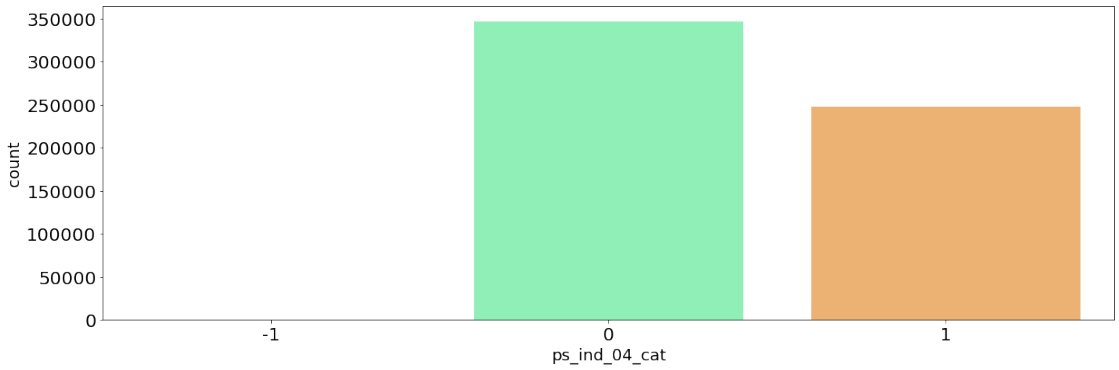
plot number of unique = 3

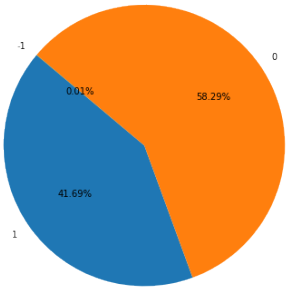
```
In [24]: feature3 = fa.Describecat(3,True)
```

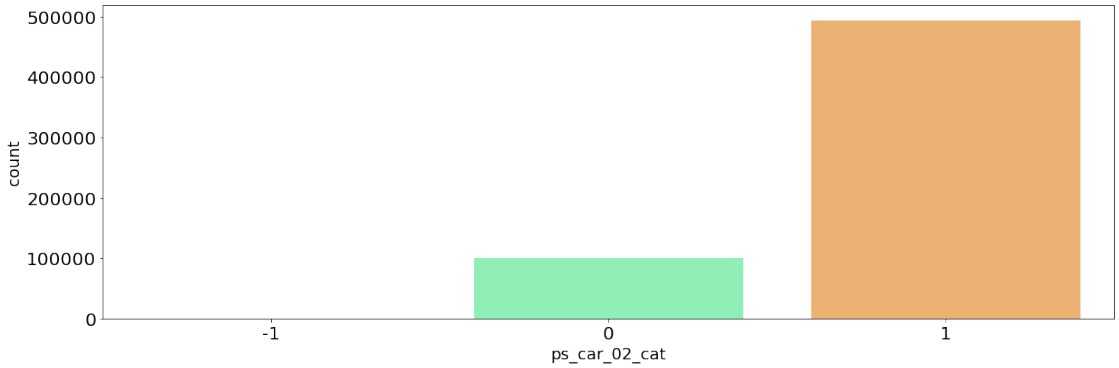
	Features	Dtype	Nunique	nduplicate	freq1	freq1_val	freq2	freq2_val	\
5	ps_ind_04_cat	int64	3	595209	0	346965	1	248164	
24	ps_car_02_cat	int64	3	595209	1	493990	0	101217	
25	ps_car_03_cat	int64	3	595209	-1	411231	1	110709	
27	ps_car_05_cat	int64	3	595209	-1	266551	1	172667	
29	ps_car_07_cat	int64	3	595209	1	553148	0	30575	
32	ps_car_10_cat	int64	3	595209	1	590179	0	4857	

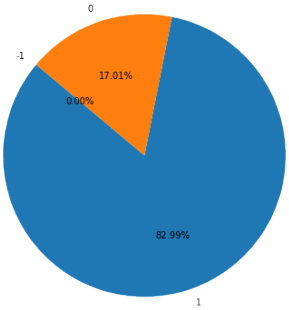
	freq3	freq3_val	mean	std	min	25%	50%	75%	max
5	-1	83	0.416794	0.493311	-1.0	0.0	0.0	1.0	1.0
24	-1	5	0.829931	0.375716	-1.0	1.0	1.0	1.0	1.0
25	0	73272	-0.504899	0.788654	-1.0	-1.0	-1.0	0.0	1.0
27	0	155994	-0.157732	0.844417	-1.0	-1.0	0.0	1.0	1.0
29	-1	11489	0.910027	0.347106	-1.0	1.0	1.0	1.0	1.0
32	2	176	0.992136	0.091619	0.0	1.0	1.0	1.0	2.0

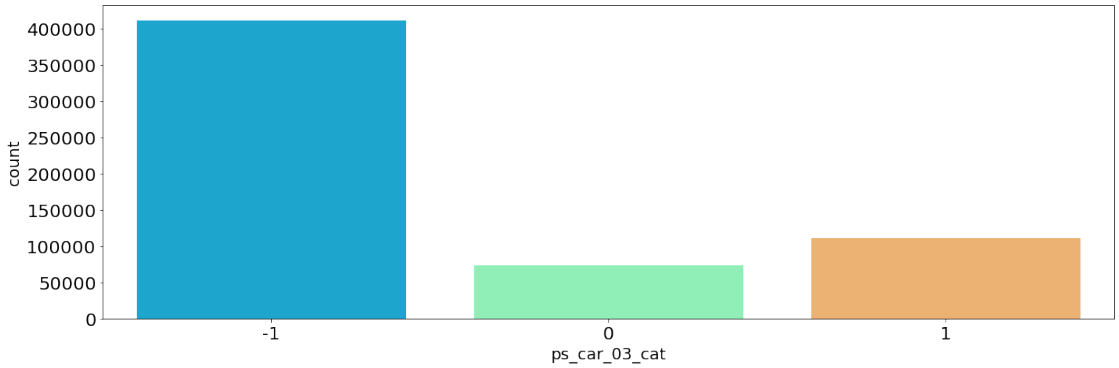
```
In [25]: fa.PlotCat(3)
```

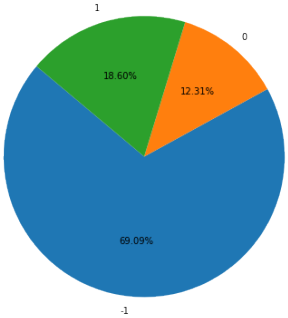


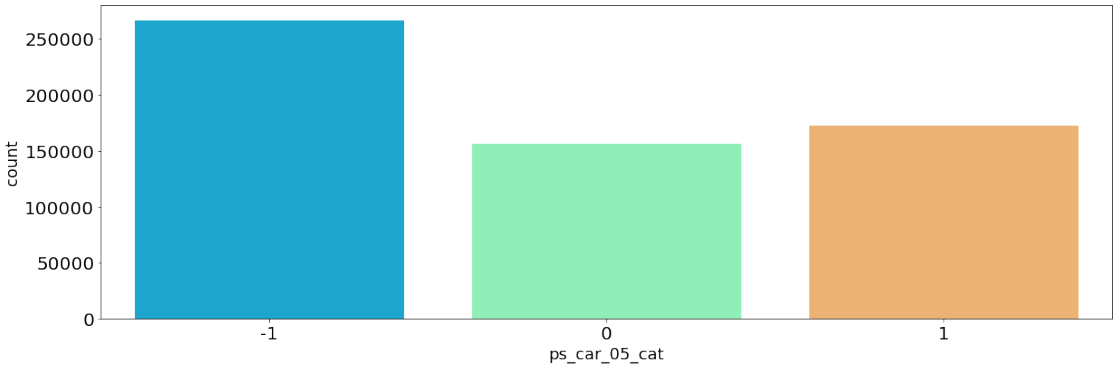


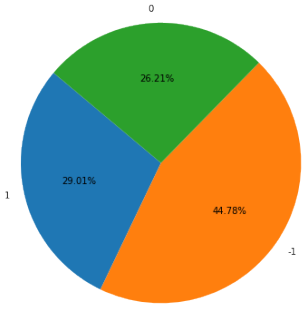

500000
400000
300000
200000
100000
0
count
-1
0
1
ps_car_02_cat

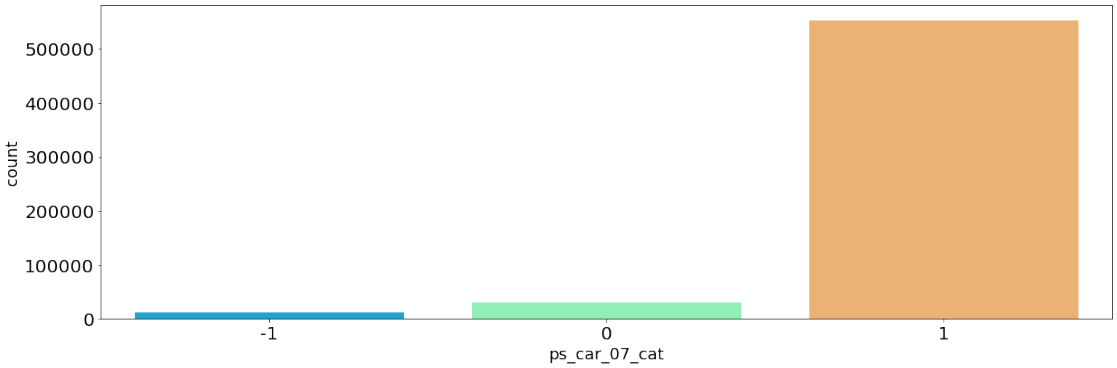

0
-1
17.01%
0.00%
82.99%
1

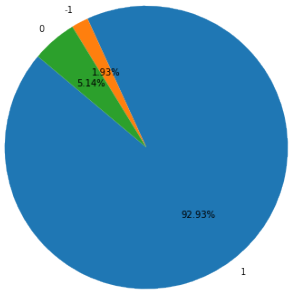

400000
350000
300000
250000
200000
150000
100000
50000
0
count
-1
0
1
ps_car_03_cat

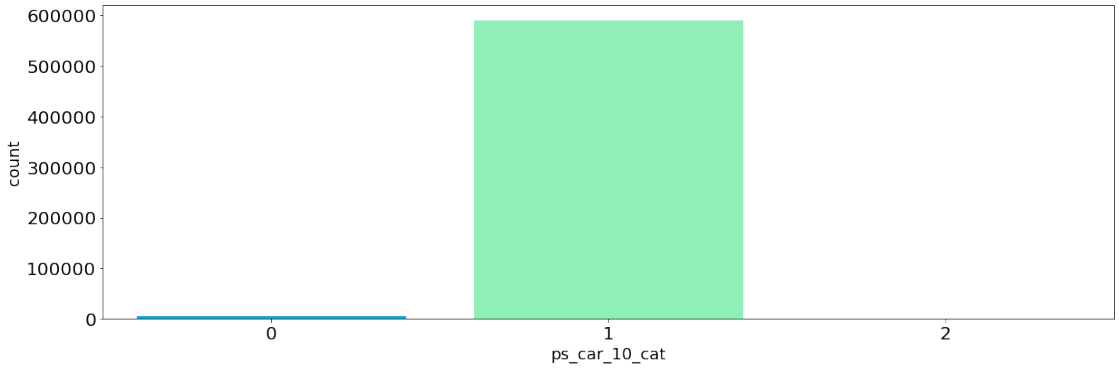

1
0
18.60%
12.31%
69.09%
-1

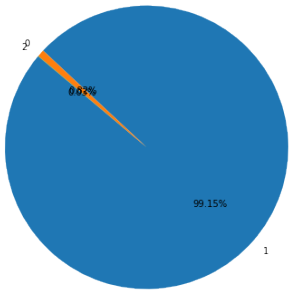

250000
200000
150000
100000
50000
0
count
-1
0
1
ps_car_05_cat


0
26.21%
29.01%
1
44.78%
-1


500000
400000
300000
200000
100000
0
count
-1
0
1
ps_car_07_cat


-1
0
1.93%
5.14%
92.93%
1



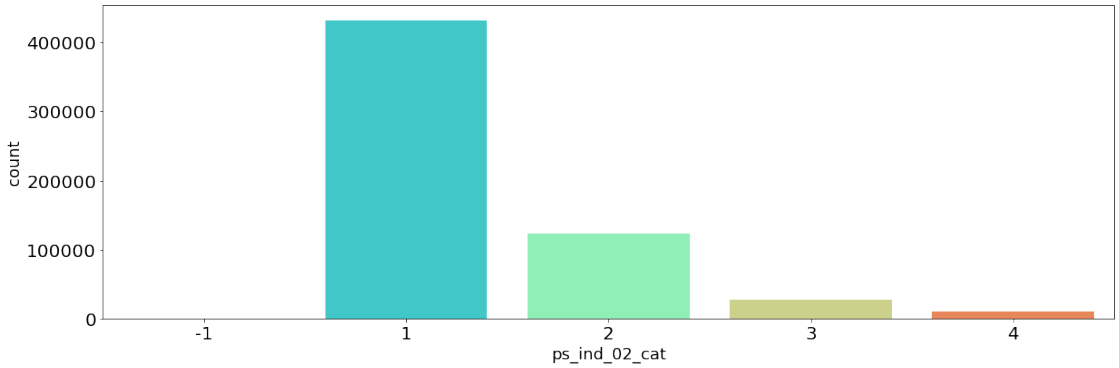


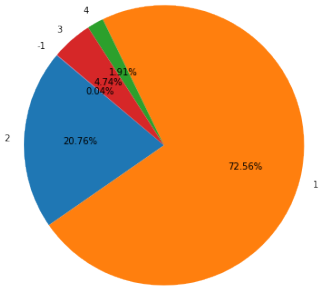
plot number of unique = 5

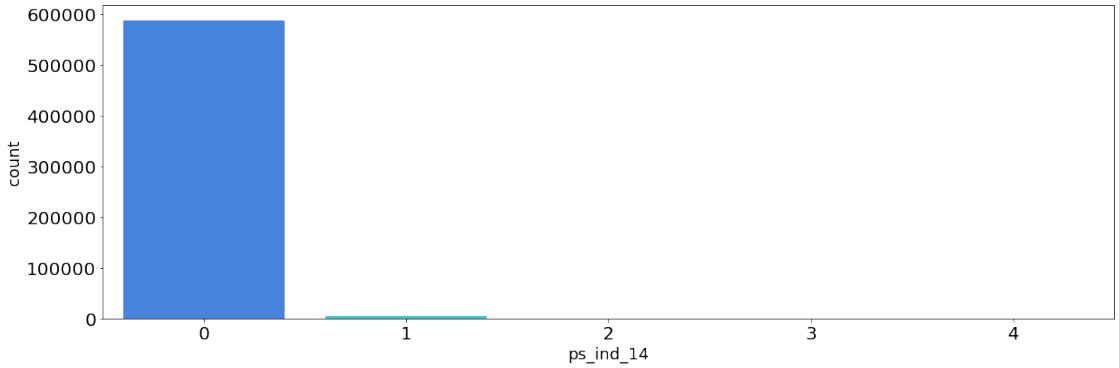
```
In [26]: feature5 = fa.Describecat(5,True)
```

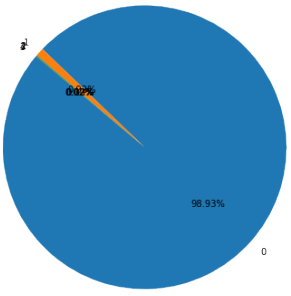
```
         Features  Dtype Nunique nduplicate freq1 freq1_val freq2 freq2_val  \
3   ps_ind_02_cat  int64       5     595207     1    431859     2    123573
15      ps_ind_14  int64       5     595207     0    588832     1      5495
34      ps_car_11  int64       5     595207     3    318919     2    189353

   freq3 freq3_val      mean       std  min  25%  50%  75%  max
3      3     28186  1.358943  0.664594 -1.0  1.0  1.0  2.0  4.0
15     2       744  0.012451  0.127545  0.0  0.0  0.0  0.0  4.0
34     1     60952  2.346072  0.832548 -1.0  2.0  3.0  3.0  3.0
```

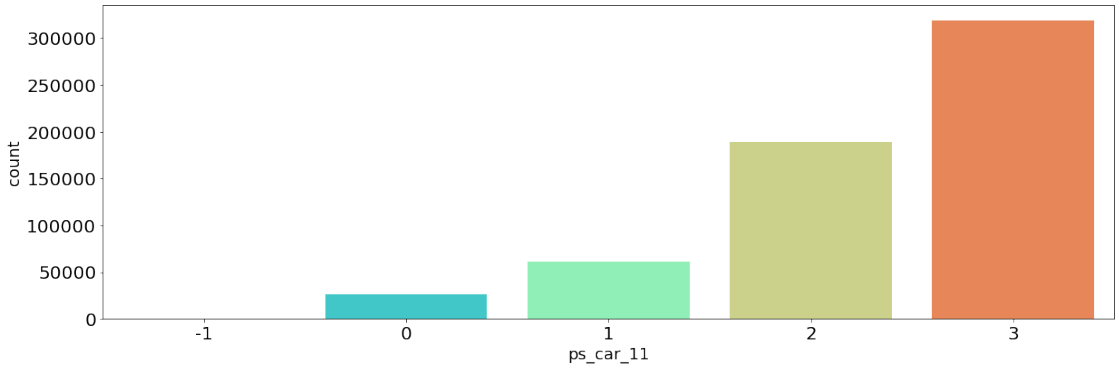
```
In [27]: fa.PlotCat(5)
```

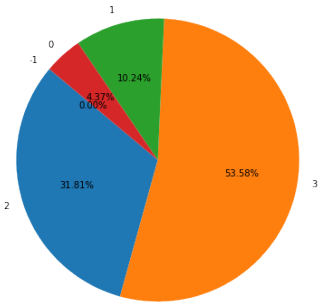

count
400000
300000
200000
100000
0
-1
1
2
3
4
ps_ind_02_cat


4
3
-1
1.91%
4.74%
0.04%
2
20.76%
72.56%
1


count
600000
500000
400000
300000
200000
100000
0
0
1
2
3
4
ps_ind_14


98.93%
0



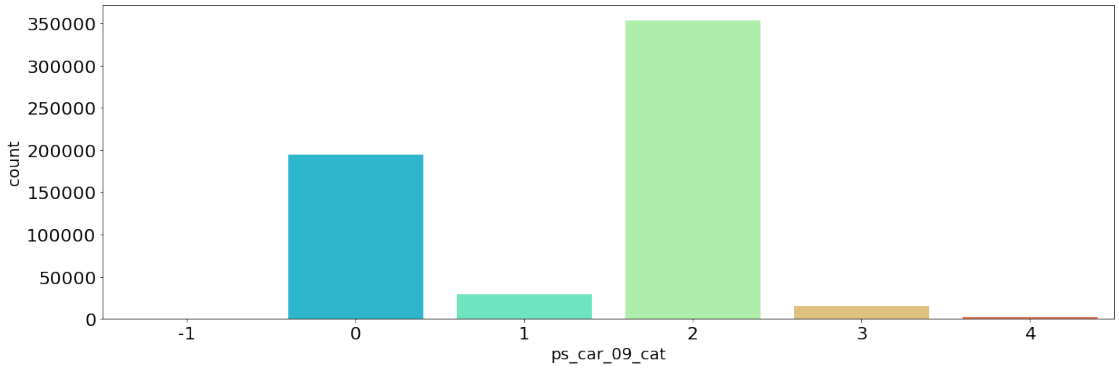


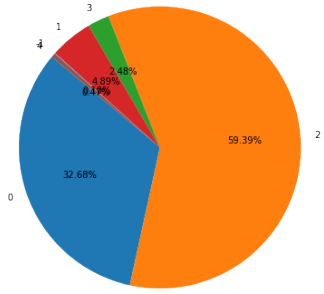
plot number of unique = 6

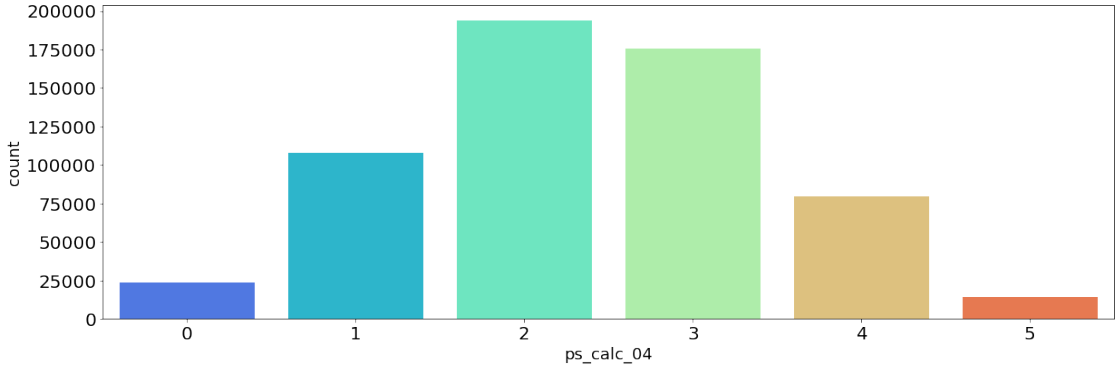
```
In [28]: feature6 = fa.Describecat(6,True)
```

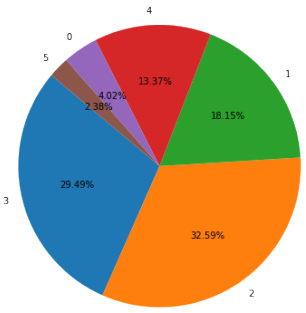
```
        Features  Dtype Nunique nduplicate freq1 freq1_val freq2 freq2_val  \
31  ps_car_09_cat  int64       6     595206     2    353482     0    194518
42     ps_calc_04  int64       6     595206     2    193977     3    175512

   freq3 freq3_val      mean       std  min  25%  50%  75%  max
31     1     29080  1.328890  0.978747 -1.0  0.0  2.0  2.0  4.0
42     1    108012  2.372081  1.117219  0.0  2.0  2.0  3.0  5.0
```

```
In [29]: fa.PlotCat(6)
```

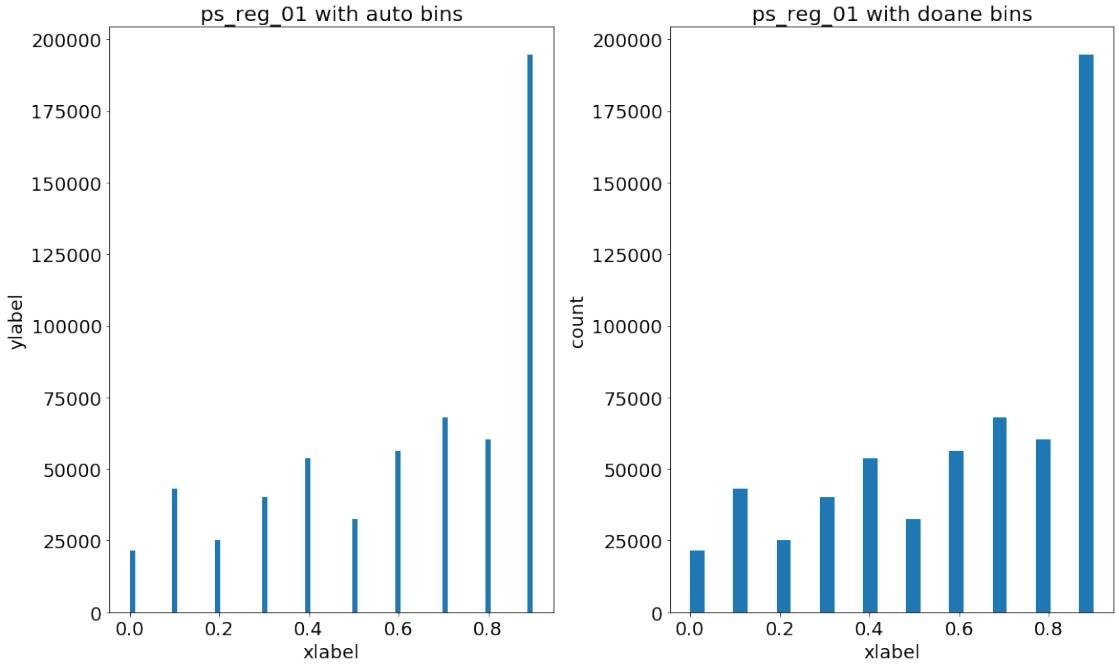




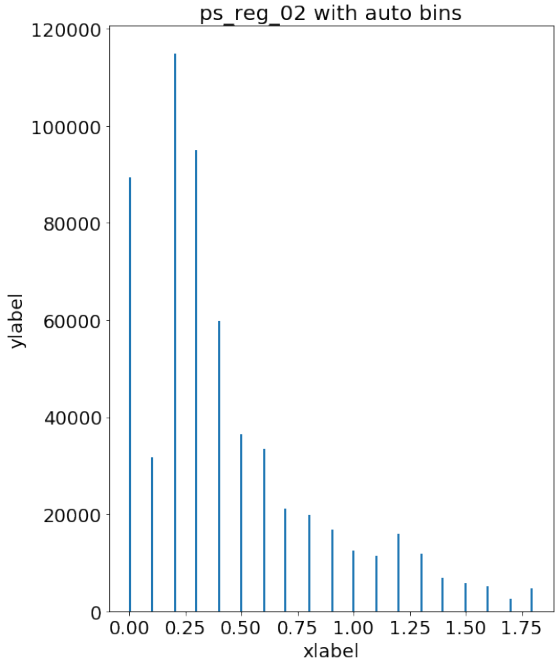


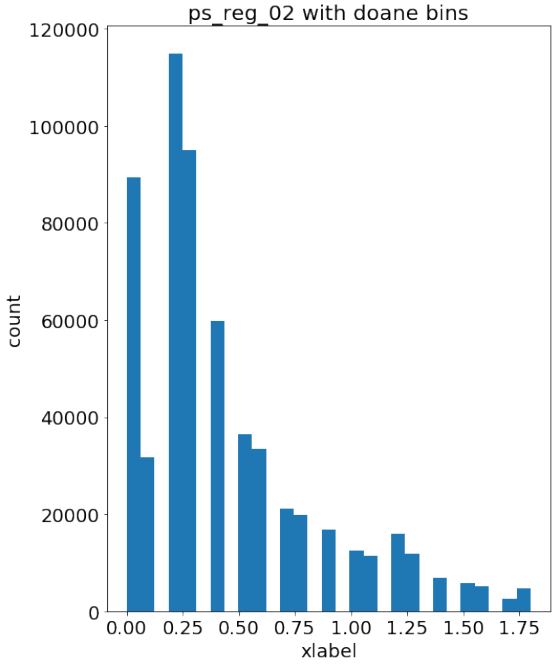


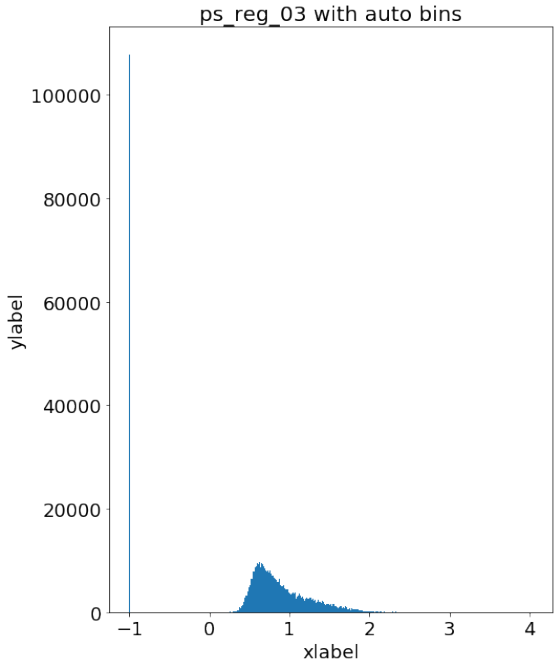
In [30]: fa.PlotNumericalBar()

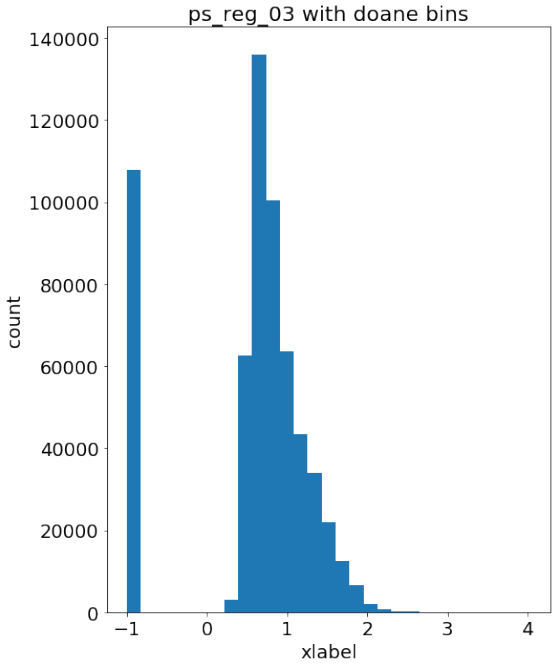

ps_reg_01 with auto bins
ylabel
xlabel
0
25000
50000
75000
100000
125000
150000
175000
200000
0.0
0.2
0.4
0.6
0.8

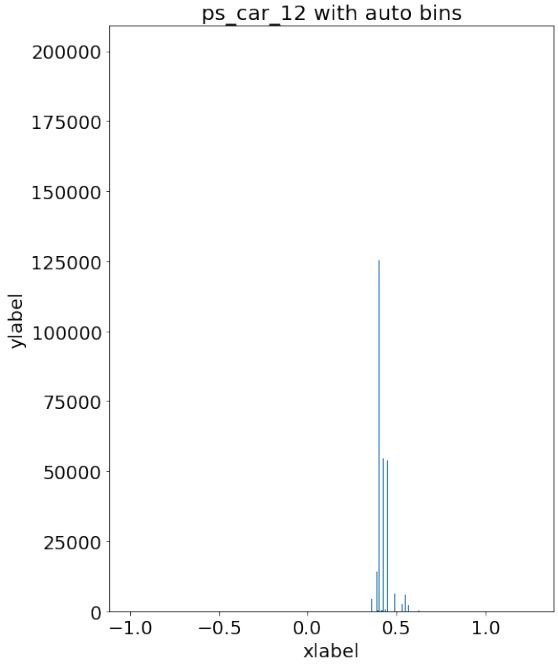
ps_reg_01 with doane bins
count
xlabel
0
25000
50000
75000
100000
125000
150000
175000
200000
0.0
0.2
0.4
0.6
0.8

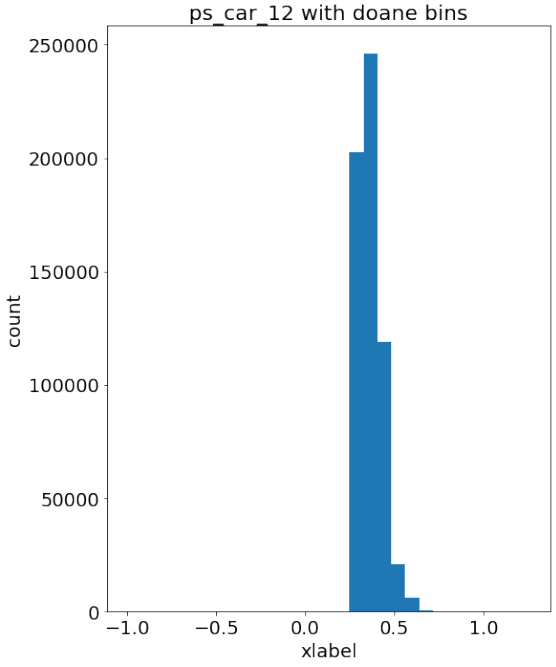

ps_reg_02 with auto bins
ylabel
xlabel
0
20000
40000
60000
80000
100000
120000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75

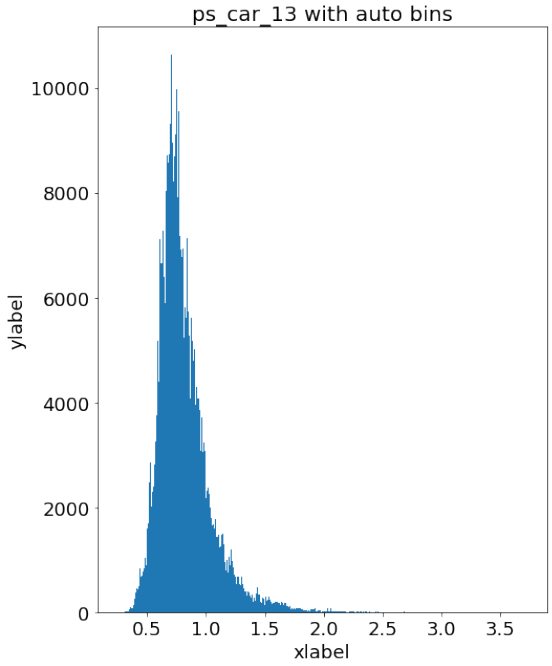

ps_reg_02 with doane bins
count
xlabel
0
20000
40000
60000
80000
100000
120000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75

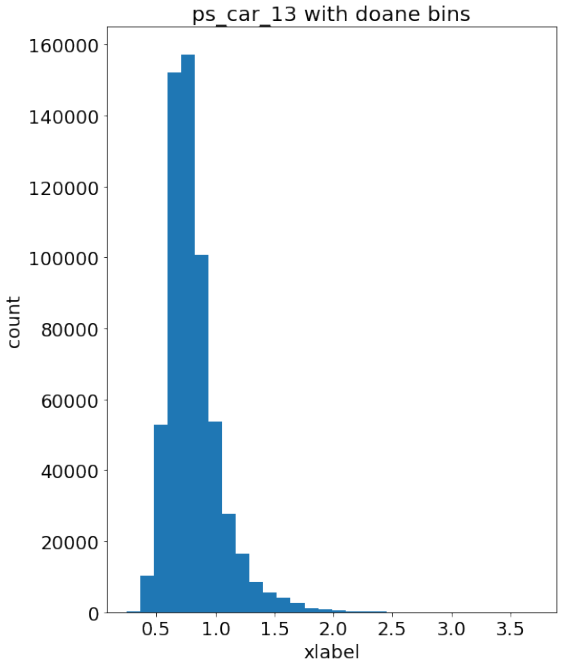

ps_reg_03 with auto bins
ylabel
xlabel
0
20000
40000
60000
80000
100000
−1
0
1
2
3
4

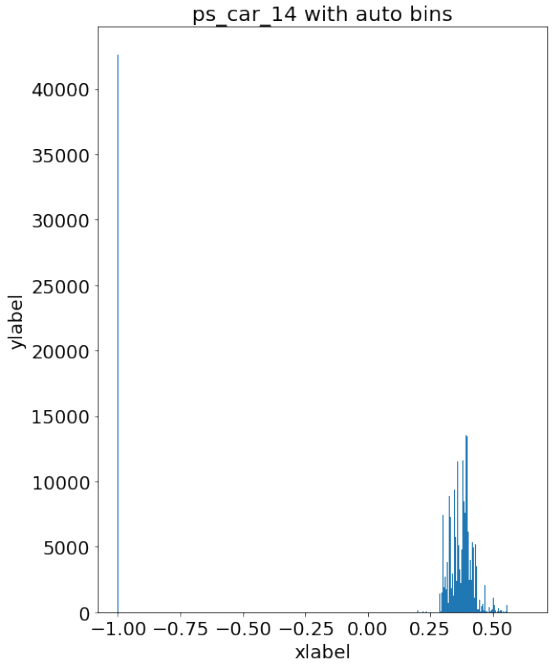

ps_reg_03 with doane bins
count
xlabel
0
20000
40000
60000
80000
100000
120000
140000
−1
0
1
2
3
4

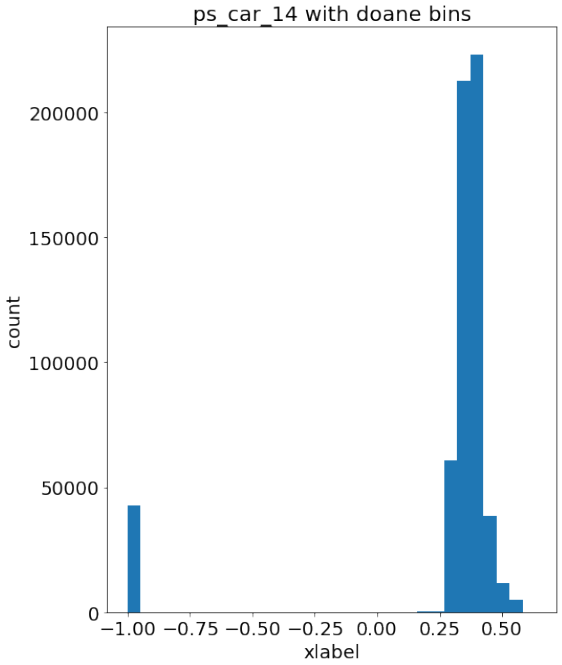

ps_car_12 with auto bins
ylabel
xlabel
0
25000
50000
75000
100000
125000
150000
175000
200000
−1.0
−0.5
0.0
0.5
1.0

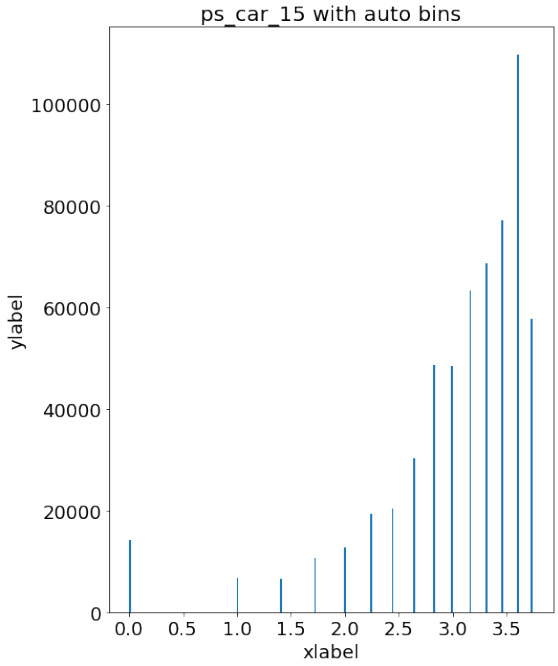

ps_car_12 with doane bins
count
xlabel
0
50000
100000
150000
200000
250000
−1.0
−0.5
0.0
0.5
1.0

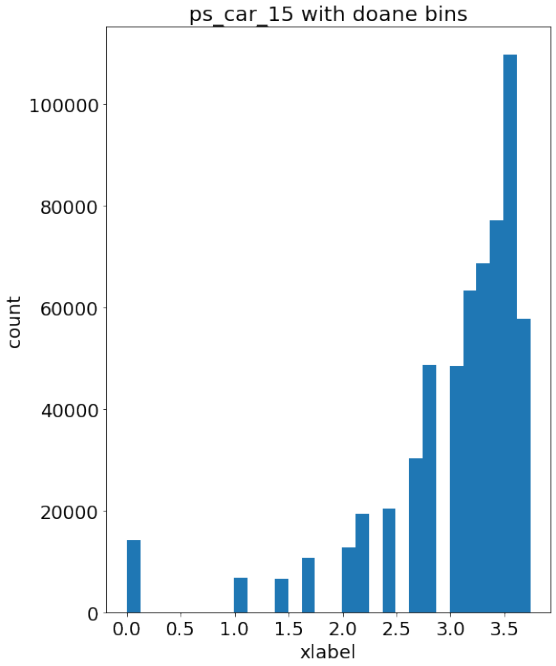

ps_car_13 with auto bins
ylabel
xlabel
0
2000
4000
6000
8000
10000
0.5
1.0
1.5
2.0
2.5
3.0
3.5

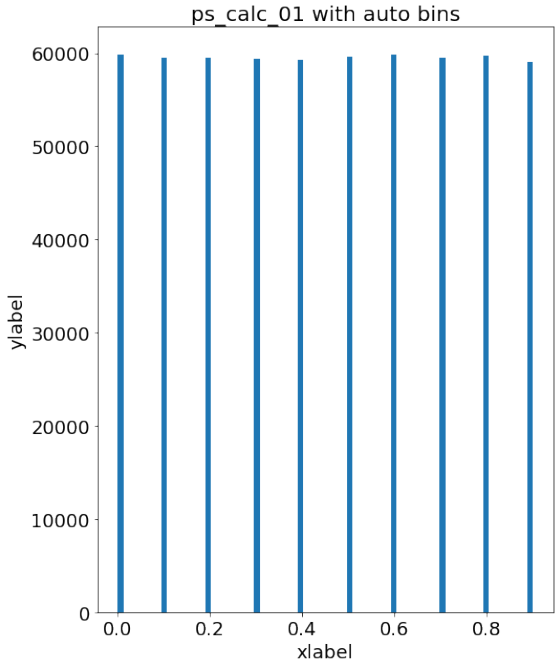

ps_car_13 with doane bins
count
xlabel
0
20000
40000
60000
80000
100000
120000
140000
160000
0.5
1.0
1.5
2.0
2.5
3.0
3.5

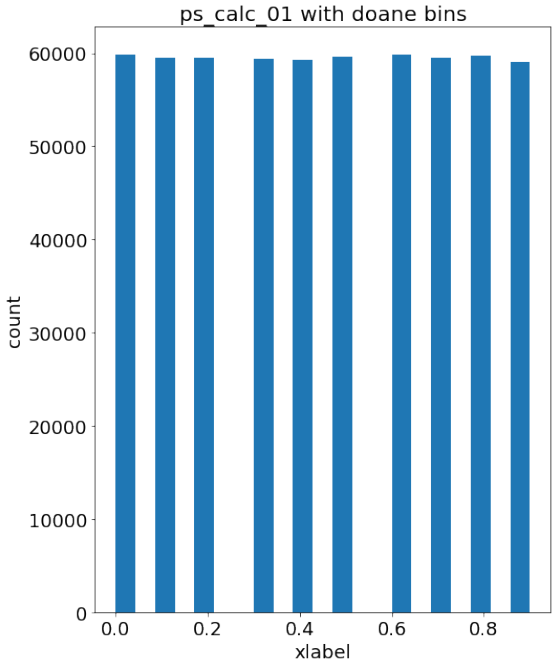

ps_car_14 with auto bins
ylabel
xlabel
0
5000
10000
15000
20000
25000
30000
35000
40000
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50

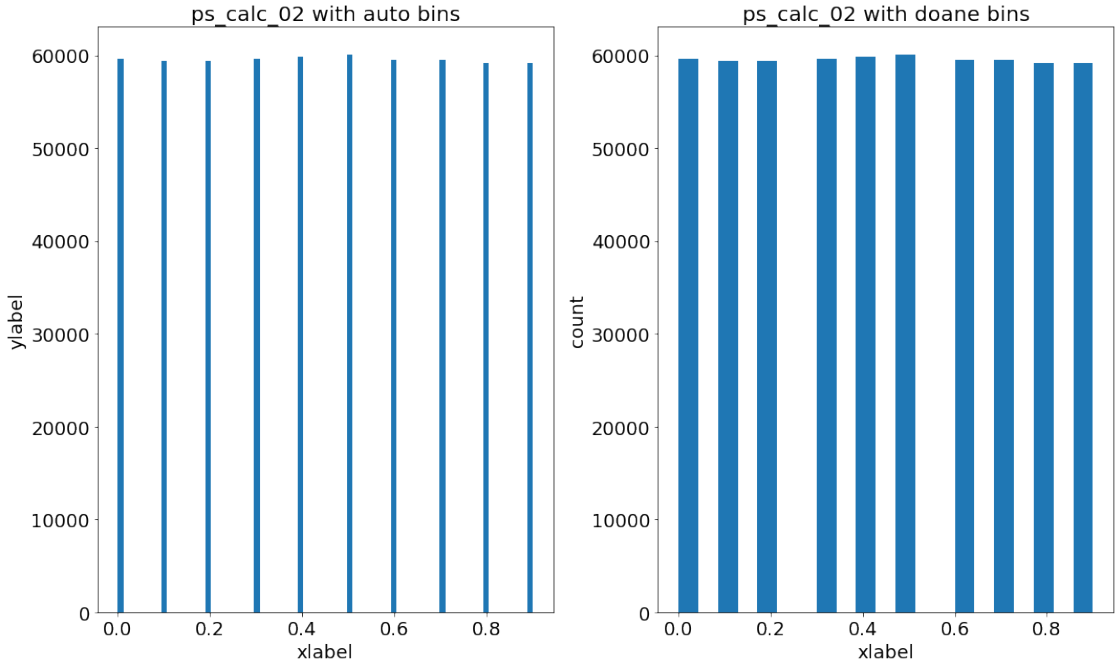

ps_car_14 with doane bins
count
xlabel
0
50000
100000
150000
200000
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50

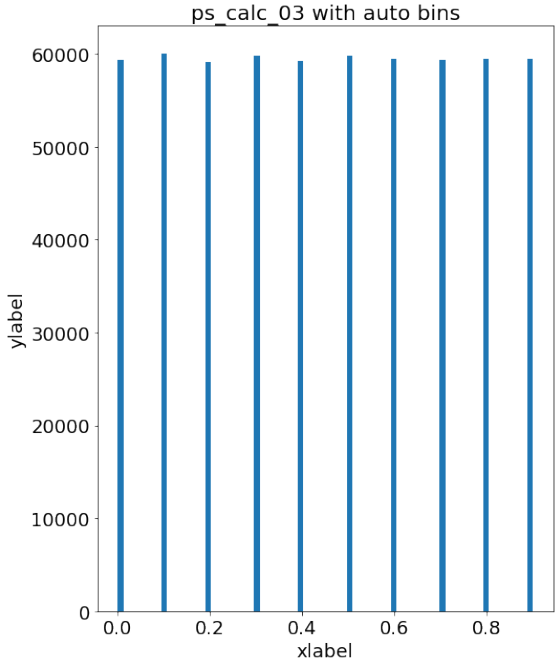

ps_car_15 with auto bins
ylabel
xlabel
0
20000
40000
60000
80000
100000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

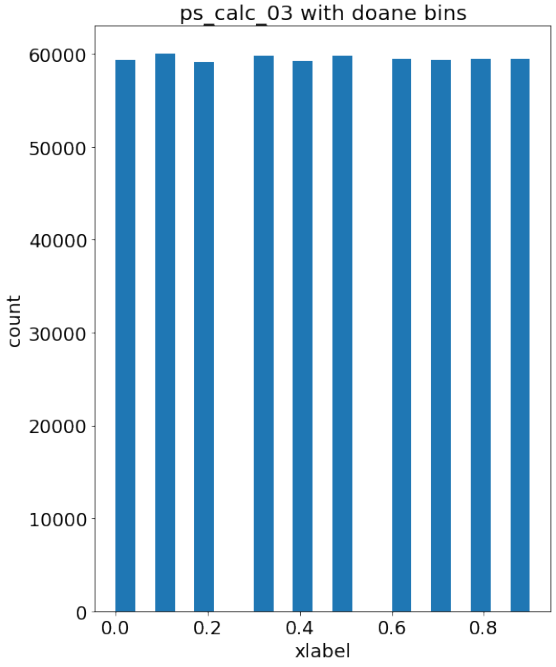

ps_car_15 with doane bins
count
xlabel
0
20000
40000
60000
80000
100000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


ps_calc_01 with auto bins
ylabel
xlabel
0
10000
20000
30000
40000
50000
60000
0.0
0.2
0.4
0.6
0.8

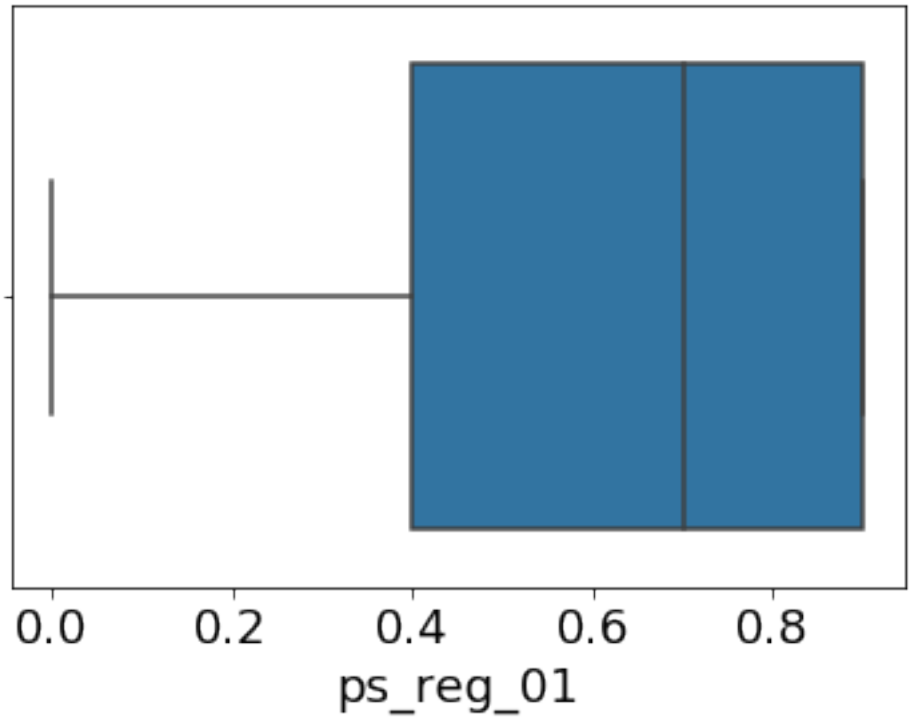

ps_calc_01 with doane bins
count
xlabel
0
10000
20000
30000
40000
50000
60000
0.0
0.2
0.4
0.6
0.8

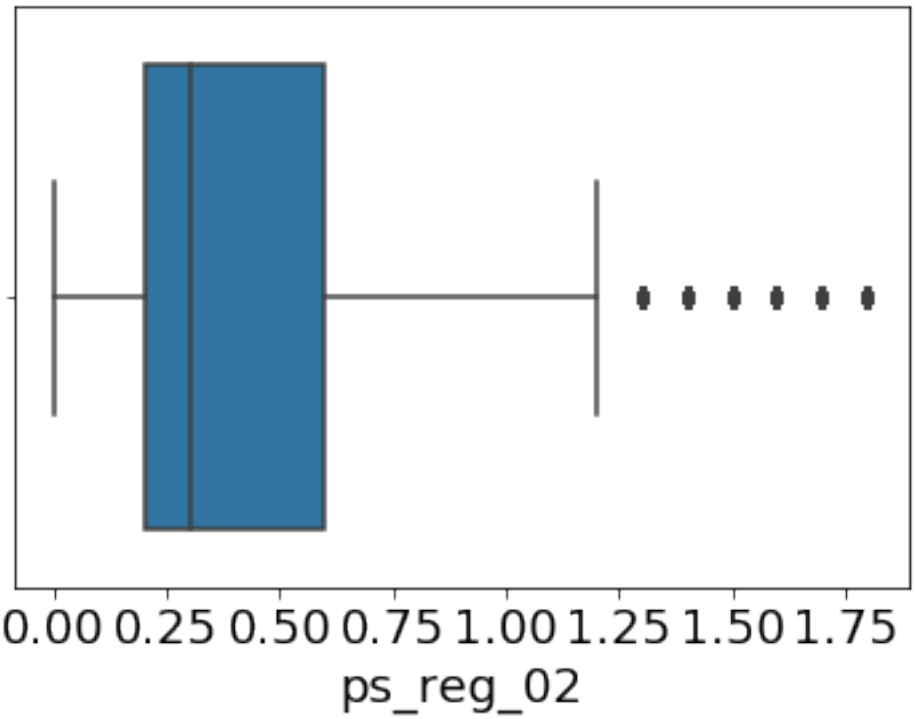


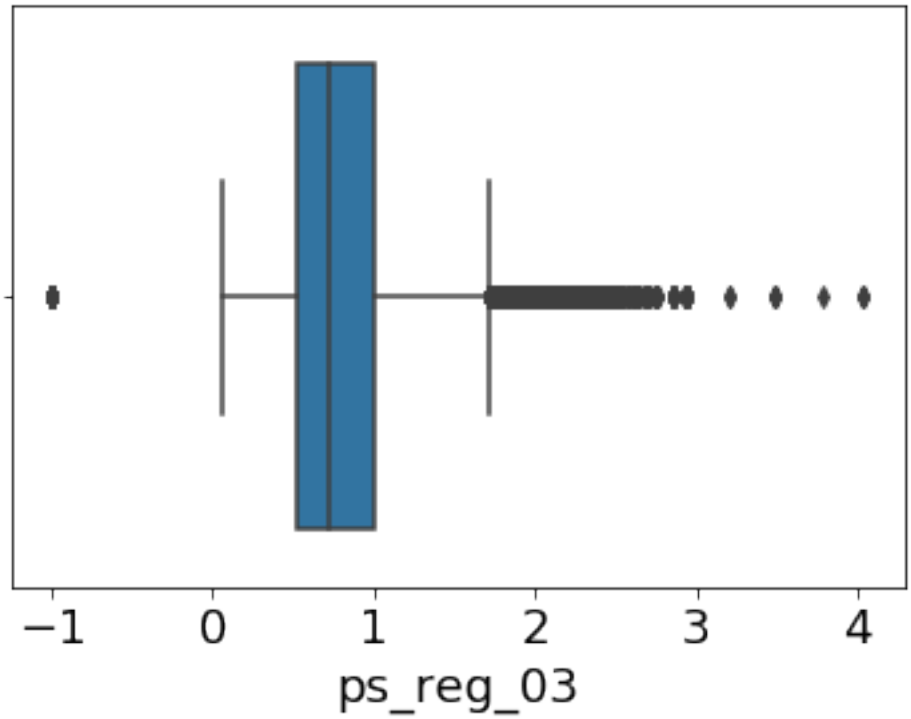


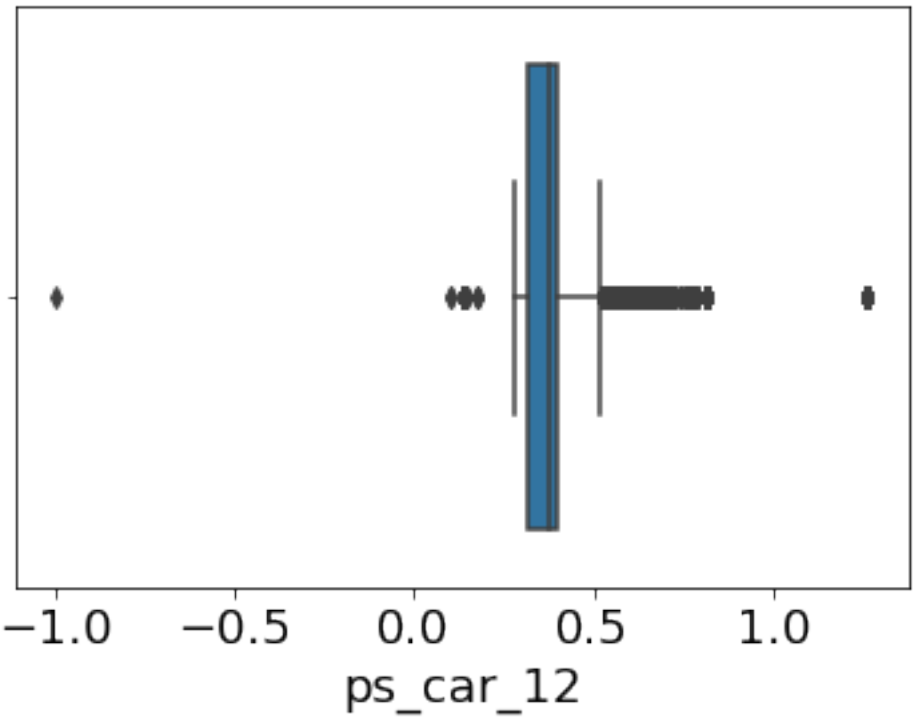


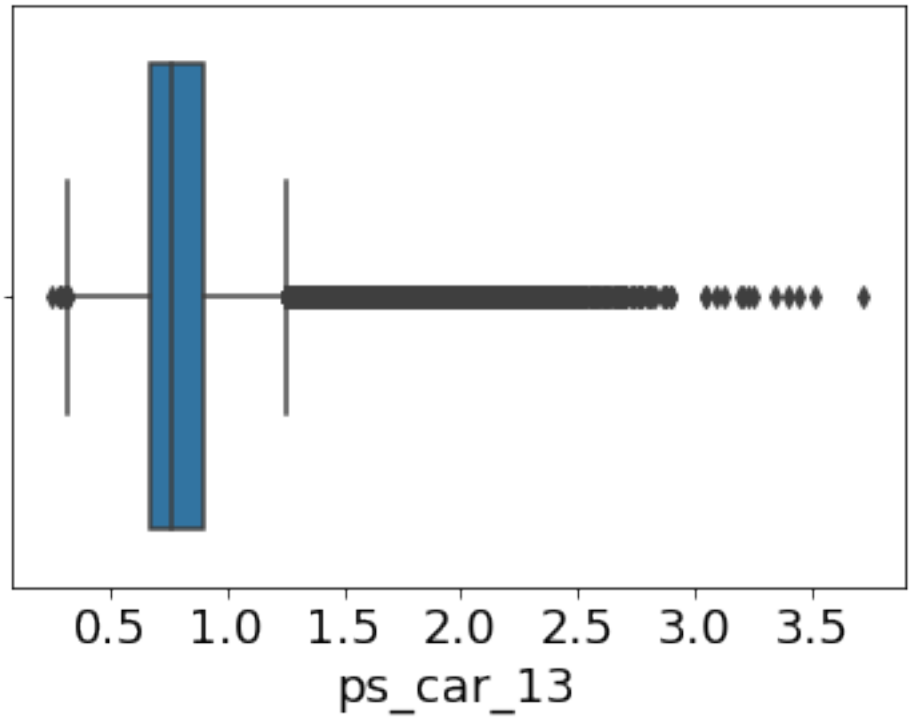
In [31]: `fa.PlotNumericalBox()`

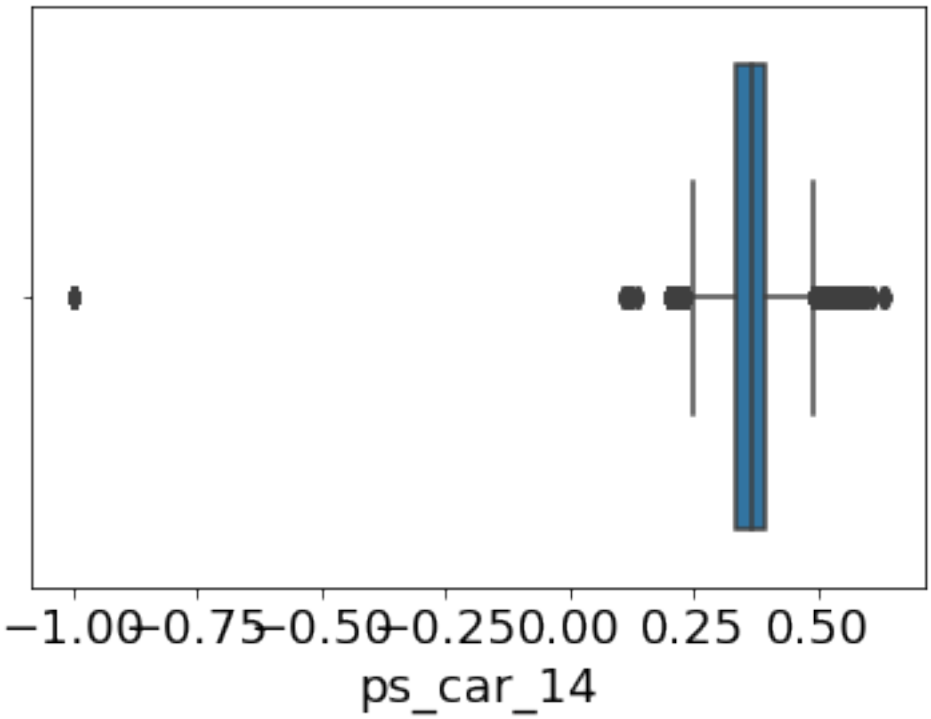

0.0
0.2
0.4
0.6
0.8
ps_reg_01

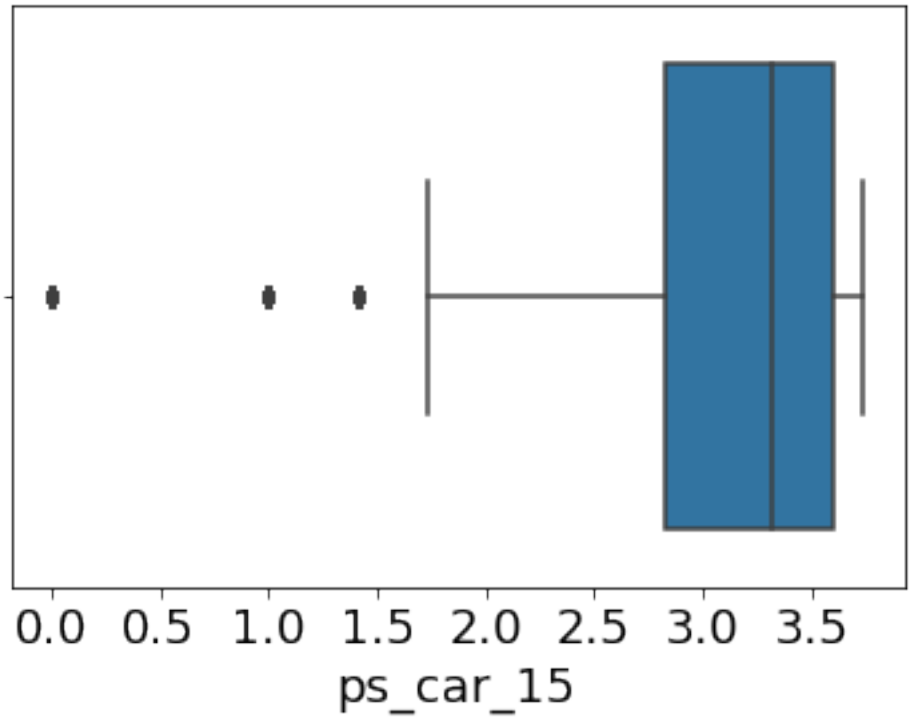

0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
ps_reg_02

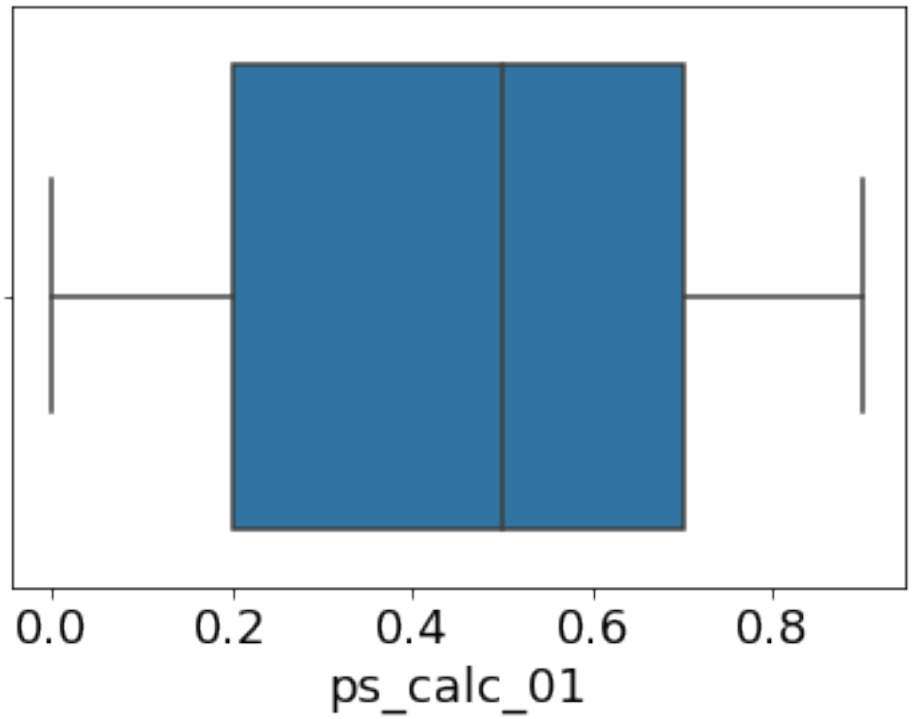

−1
0
1
2
3
4
ps_reg_03

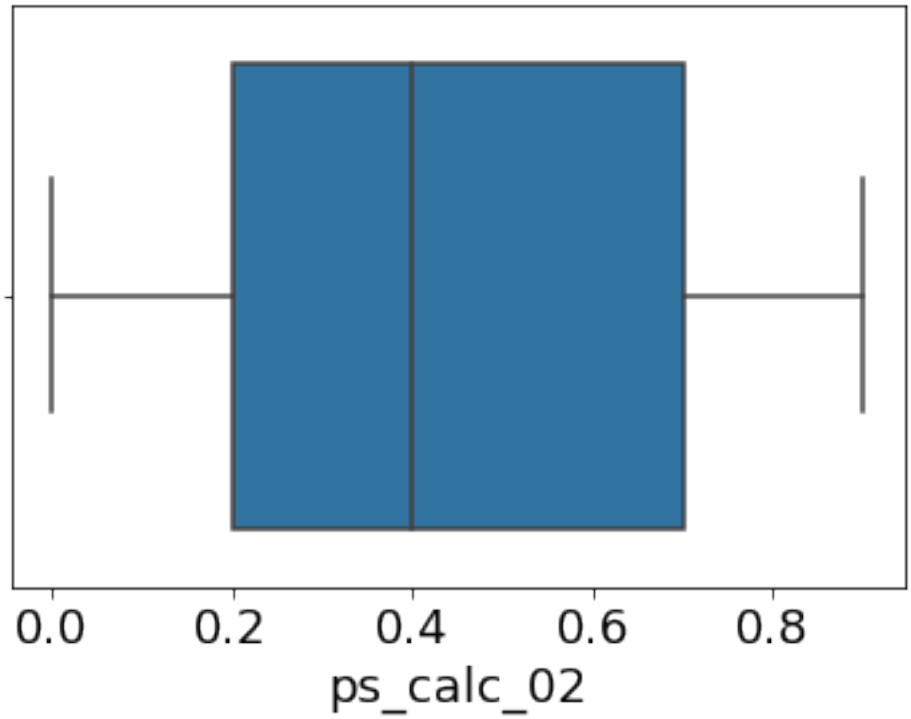

−1.0
−0.5
0.0
0.5
1.0
ps_car_12

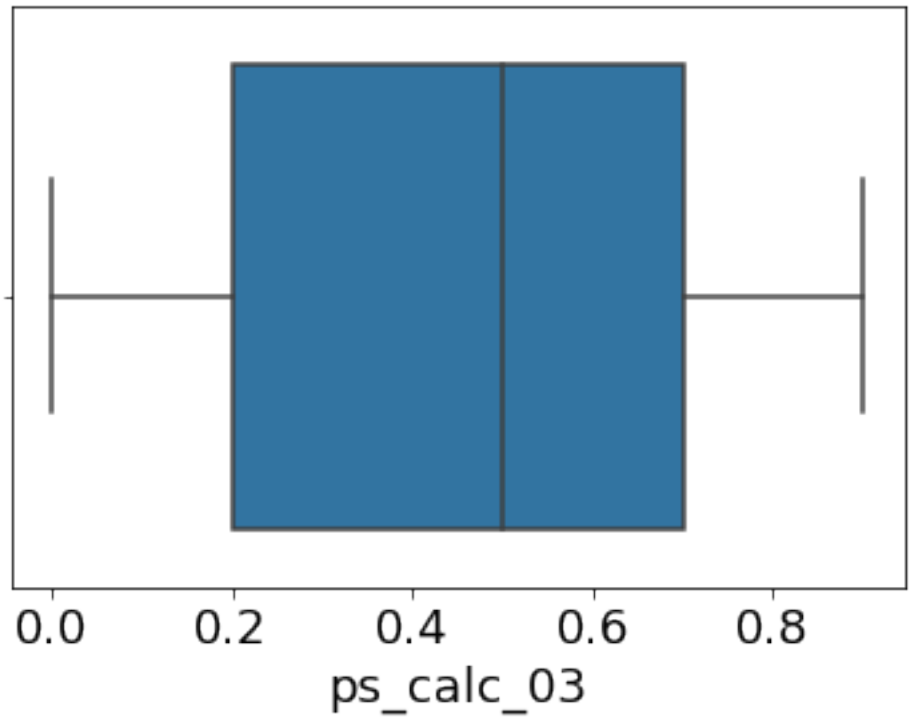

0.5
1.0
1.5
2.0
2.5
3.0
3.5
ps_car_13


−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
ps_car_14


0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
ps_car_15


0.0
0.2
0.4
0.6
0.8
ps_calc_01

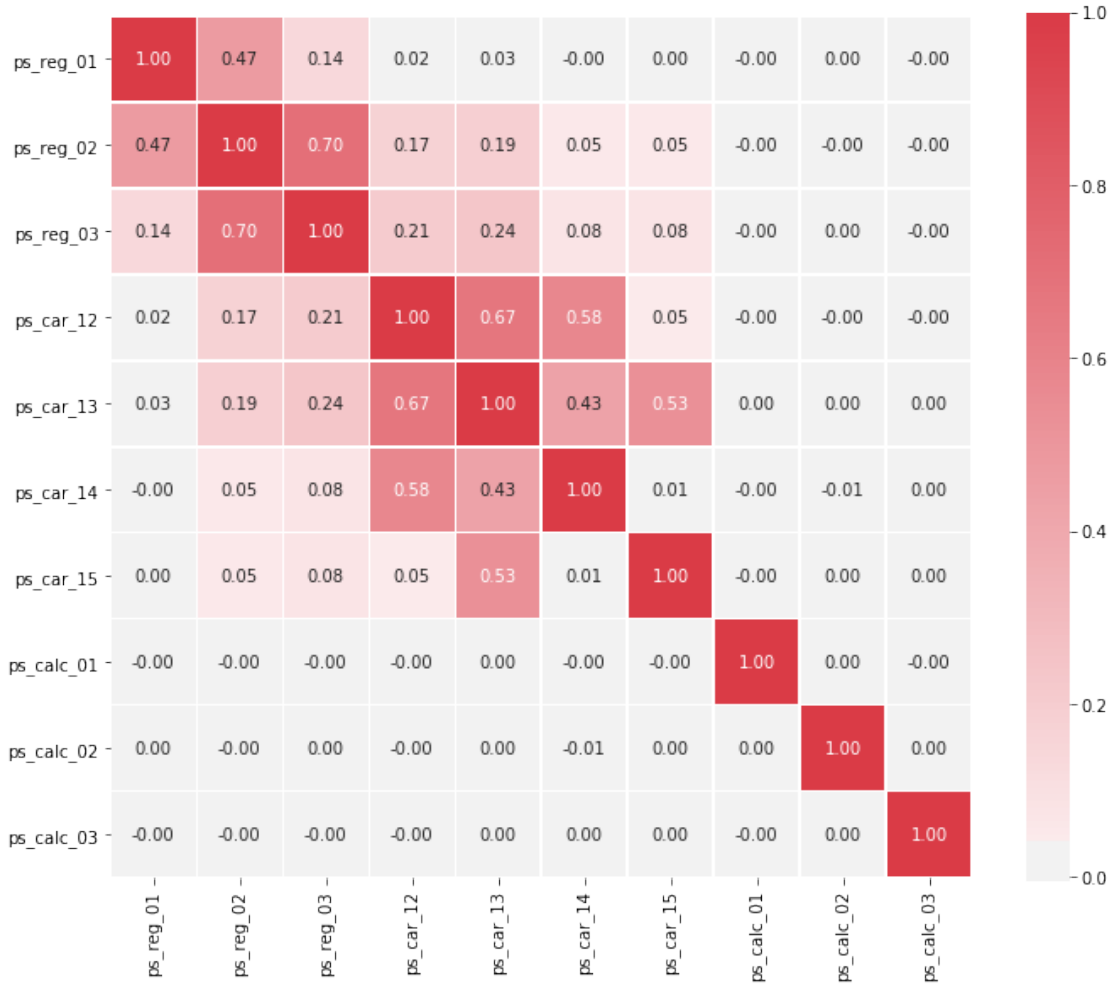

0.0
0.2
0.4
0.6
0.8
ps_calc_02


0.0
0.2
0.4
0.6
0.8
ps_calc_03

2.5 1.5. Interval variables

Checking the correlations between interval variables. A heatmap is a good way to visualize the correlation between variables.

```
In [32]: data_float = train[feature_inofrmation[feature_inofrmation['Dtype'] == 'float64'].Featu

In [33]: mc = MC.MuiltiFeatureComparisons(data_float)
         mc.HeatMap()
```

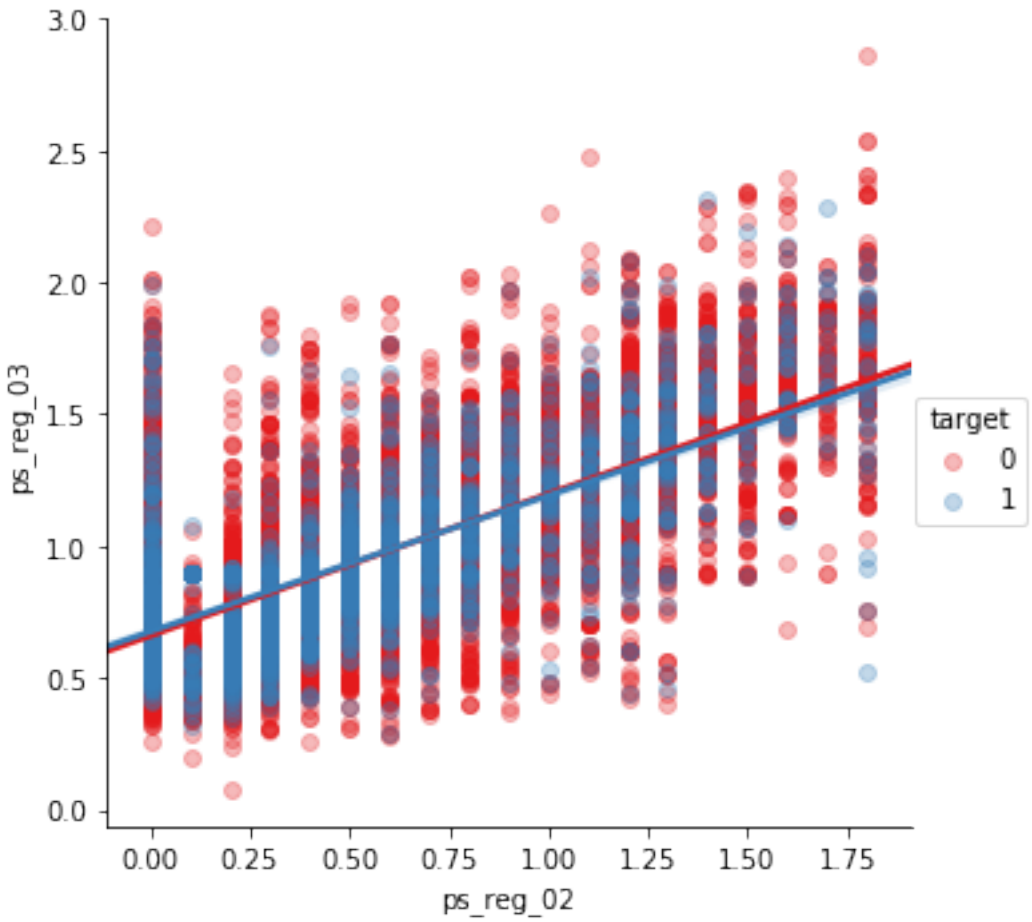



```
In [34]: s = train.sample(frac=0.1)
```

ps_reg_02 and ps_reg_03

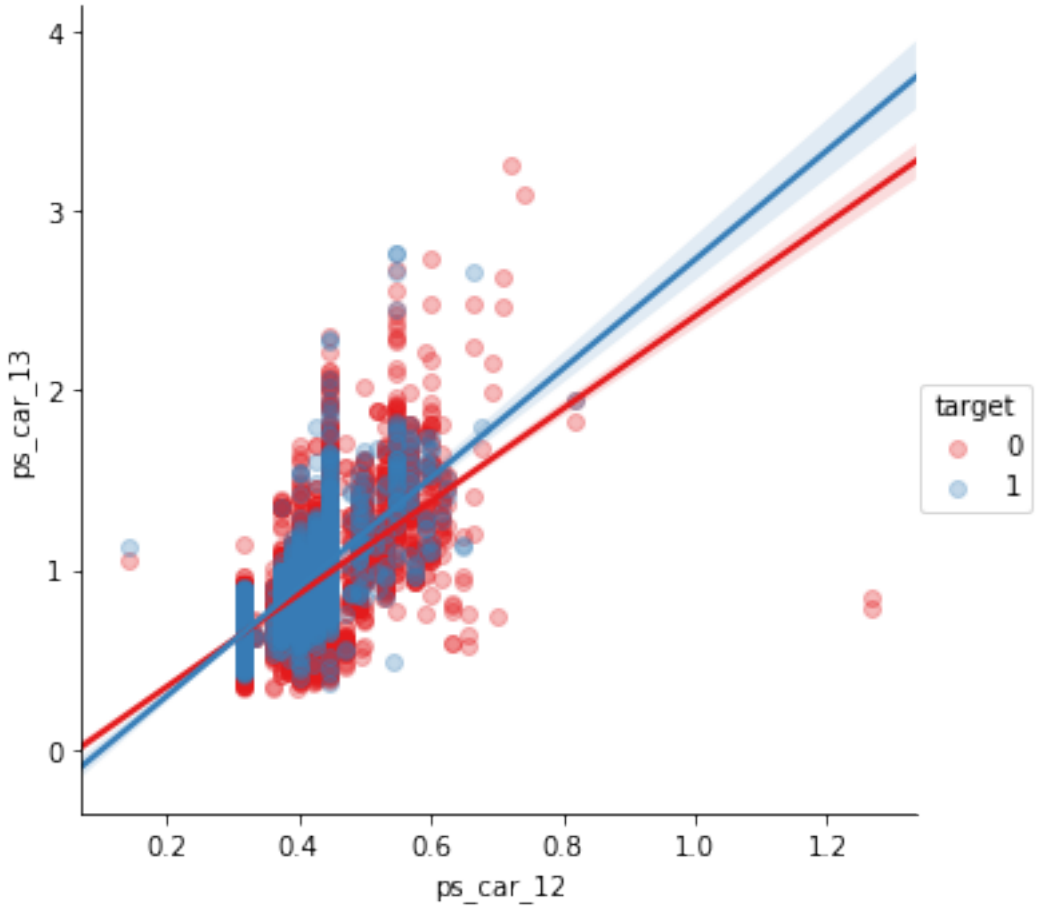
As the regression line shows, there is a linear relationship between these variables. Thanks to the hue parameter we can see that the regression lines for target=0 and target=1 are the same.

```
In [35]: sns.lmplot(x='ps_reg_02', y='ps_reg_03', data=s, hue='target', palette='Set1', scatter_
         plt.show()
```



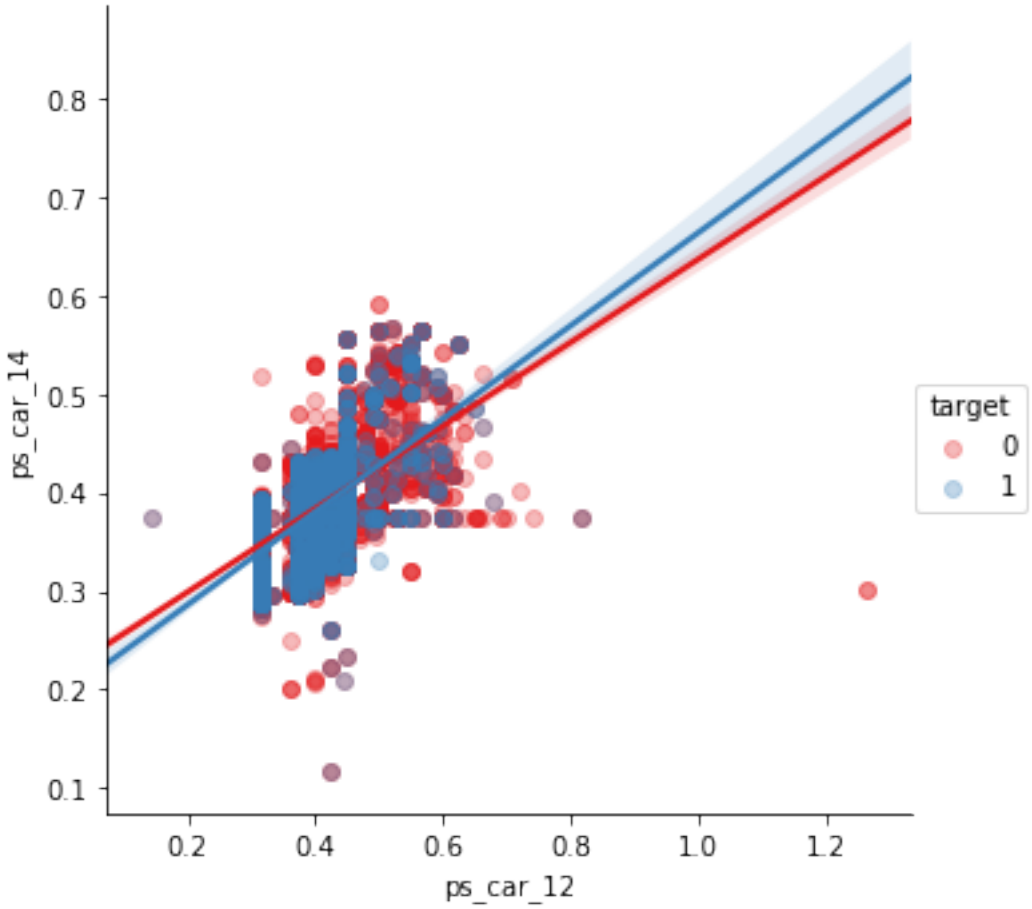
** ps_car_12 and ps_car_13 **

```
In [36]: sns.lmplot(x='ps_car_12', y='ps_car_13', data=s, hue='target', palette='Set1', scatter_
         plt.show()
```



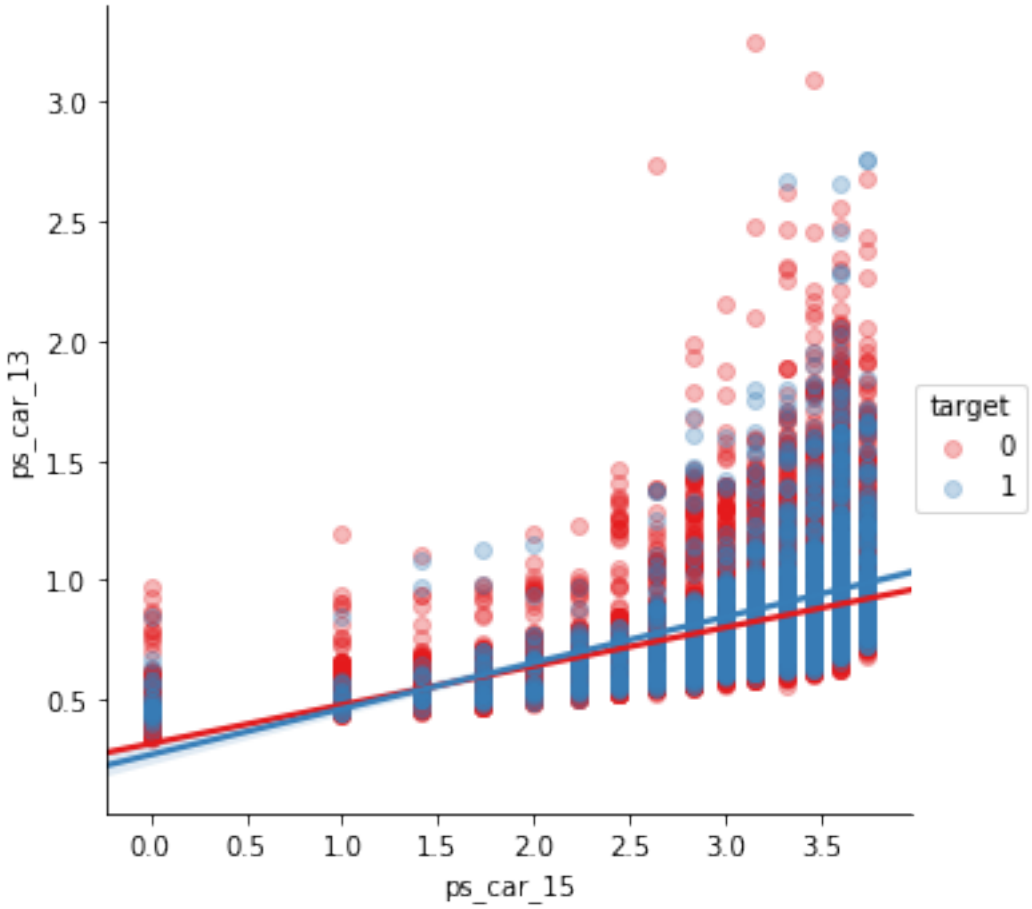
ps_car_12 and ps_car_14

```
In [37]: sns.lmplot(x='ps_car_12', y='ps_car_14', data=s, hue='target', palette='Set1', scatter_
         plt.show()
```



** ps_car_13 and ps_car_15 **

```
In [38]: sns.lmplot(x='ps_car_15', y='ps_car_13', data=s, hue='target', palette='Set1', scatter_
         plt.show()
```



3 2. Feature engineering

3.1 2.1. Creating interaction variables

```
In [39]: feng = PCAT.FeaturesEng()
         iterval_f = sample_feature_inofrmation[sample_feature_inofrmation['Dtype']=='float64'].
         train, test = feng.NumericalInteractions(train, test, iterval_f)
```

```
Before creating interactions we have 57 variables
After creating interactions we have 112 variables
Before creating interactions we have 56 variables
After creating interactions we have 111 variables
```

3.2 2.2. Frequency Encondig

```
In [40]: cat_cols=['ps_ind_02_cat','ps_car_04_cat', 'ps_car_09_cat',
                   'ps_ind_05_cat', 'ps_car_01_cat', 'ps_car_11_cat']
```

```
        # generate dataframe for frequency features for the train and test dataset
        train_freq, test_freq=feng.FreqEncoding(cat_cols, train, test)

        # merge them into the original train and test dataset
        train=pd.concat([train, train_freq], axis=1)
        test=pd.concat([test,test_freq], axis=1)
```

3.3 2.3. Binary Encoding

```
In [41]: cat_cols=['ps_ind_02_cat','ps_car_04_cat', 'ps_car_09_cat',
                   'ps_ind_05_cat', 'ps_car_01_cat']

         for w in cat_cols:
             train, test=feng.BinaryEncoding(train, test, w)

         print('train shape is: ', train.shape)
         print('test shape is: ', test.shape)
```

```
train shape is:  (216940, 135)
test shape is:  (892816, 134)
```

3.4 1.3 Features Importance

Selecting features with a Random Forest

3.5 1.3 Feature Reduction

Here we'll base feature selection on the feature importances of a random forest. With Sklearn's SelectFromModel you can then specify how many variables you want to keep. You can set a threshold on the level of feature importance manually. But we'll simply select the top 50% best variables.

```
In [42]: fsel = FSEL.FeaturesSelection()
         nnew_train, nnew_test =  fsel.MainFeatureSelection(train, ['id', 'target'], test, 'targ
```

```
 1) ps_car_13^2                  0.017398
 2) ps_car_13                    0.017363
 3) ps_car_12 ps_car_13          0.017314
 4) ps_reg_03 ps_car_13          0.017196
 5) ps_car_13 ps_car_14          0.017165
 6) ps_reg_01 ps_car_13          0.016896
 7) ps_car_13 ps_car_15          0.016713
 8) ps_reg_03 ps_car_14          0.016199
 9) ps_reg_03 ps_car_12          0.015480
10) ps_reg_03 ps_car_15          0.015103
11) ps_car_14 ps_car_15          0.015071
12) ps_reg_02 ps_car_13          0.014657
```

```
13) ps_car_13 ps_calc_01          0.014598
14) ps_reg_01 ps_reg_03           0.014592
15) ps_car_13 ps_calc_02          0.014555
16) ps_car_13 ps_calc_03          0.014553
17) ps_reg_01 ps_car_14           0.014318
18) ps_reg_03^2                   0.014111
19) ps_reg_03                     0.014105
20) ps_calc_10                    0.013795
21) ps_reg_03 ps_calc_02          0.013709
22) ps_reg_03 ps_calc_03          0.013684
23) ps_calc_14                    0.013619
24) ps_reg_03 ps_calc_01          0.013608
25) ps_car_14 ps_calc_02          0.013517
26) ps_car_14 ps_calc_01          0.013465
27) ps_car_14 ps_calc_03          0.013372
28) ps_car_12 ps_car_14           0.012813
29) ps_calc_11                    0.012723
30) ps_reg_02 ps_car_14           0.012614
31) ps_car_14^2                   0.012607
32) ps_car_14                     0.012592
33) ps_ind_03                     0.012403
34) ps_reg_02 ps_reg_03           0.012328
35) ps_ind_15                     0.012188
36) ps_car_11_cat_freq            0.011120
37) ps_car_12 ps_car_15           0.010974
38) ps_car_11_cat                 0.010817
39) ps_car_15 ps_calc_02          0.010725
40) ps_car_15 ps_calc_01          0.010708
41) ps_car_15 ps_calc_03          0.010706
42) ps_calc_13                    0.010487
43) ps_car_12 ps_calc_01          0.010303
44) ps_car_12 ps_calc_03          0.010205
45) ps_car_12 ps_calc_02          0.010168
46) ps_reg_02 ps_car_15           0.010135
47) ps_reg_01 ps_car_15           0.010128
48) ps_calc_02 ps_calc_03         0.009935
49) ps_calc_01 ps_calc_02         0.009873
50) ps_calc_01 ps_calc_03         0.009841
51) ps_calc_07                    0.009817
52) ps_calc_08                    0.009796
53) ps_reg_01 ps_car_12           0.009329
54) ps_reg_02 ps_car_12           0.009265
55) ps_reg_02 ps_calc_01          0.009165
56) ps_reg_02 ps_calc_03          0.009127
57) ps_reg_02 ps_calc_02          0.009044
58) ps_calc_06                    0.009032
59) ps_reg_01 ps_calc_03          0.008903
60) ps_reg_01 ps_calc_02          0.008875
```

```
61) ps_reg_01 ps_calc_01          0.008857
62) ps_calc_09                    0.008823
63) ps_ind_01                     0.008549
64) ps_calc_05                    0.008290
65) ps_car_06_cat                 0.008236
66) ps_calc_04                    0.008121
67) ps_calc_12                    0.008012
68) ps_reg_01 ps_reg_02           0.007916
69) ps_car_01_cat_freq            0.007149
70) ps_car_01_cat                 0.006692
71) ps_car_15                     0.005981
72) ps_car_15^2                   0.005931
73) ps_calc_03                    0.005748
74) ps_calc_03^2                  0.005744
75) ps_calc_01                    0.005734
76) ps_calc_01^2                  0.005724
77) ps_calc_02                    0.005696
78) ps_calc_02^2                  0.005688
79) ps_car_12^2                   0.005239
80) ps_car_12                     0.005184
81) ps_reg_02                     0.004845
82) ps_reg_02^2                   0.004793
83) ps_car_09_cat_freq            0.004046
84) ps_reg_01^2                   0.003994
85) ps_reg_01                     0.003983
86) ps_car_09_cat                 0.003580
87) ps_ind_02_cat_freq            0.003567
88) ps_car_11                     0.003537
89) ps_ind_02_cat                 0.003532
90) ps_ind_05_cat_freq            0.003255
91) ps_ind_05_cat                 0.003246
92) ps_ind_17_bin                 0.002858
93) ps_ind_04_cat                 0.002613
94) ps_calc_17_bin                0.002581
95) ps_calc_16_bin                0.002524
96) ps_calc_19_bin                0.002504
97) ps_calc_18_bin                0.002407
98) ps_car_07_cat                 0.002352
99) ps_ind_07_bin                 0.002335
100) ps_ind_16_bin                 0.002323
101) ps_car_04_cat_freq            0.002192
102) ps_car_04_cat                 0.002016
103) ps_calc_20_bin                0.002013
104) ps_ind_06_bin                 0.002008
105) ps_calc_15_bin                0.001962
106) ps_car_09_cat_bin_1           0.001935
107) ps_ind_08_bin                 0.001917
108) ps_ind_02_cat_bin_1           0.001876
```

```
109) ps_car_01_cat_bin_1           0.001870
110) ps_ind_02_cat_bin_2           0.001863
111) ps_car_01_cat_bin_0           0.001821
112) ps_car_01_cat_bin_2           0.001804
113) ps_car_01_cat_bin_3           0.001794
114) ps_ind_09_bin                 0.001695
115) ps_ind_18_bin                 0.001691
116) ps_car_09_cat_bin_2           0.001593
117) ps_car_02_cat                 0.001437
118) ps_ind_05_cat_bin_0           0.001427
119) ps_car_08_cat                 0.001305
120) ps_ind_05_cat_bin_1           0.001282
121) ps_car_04_cat_bin_3           0.000935
122) ps_ind_14                     0.000880
123) ps_car_04_cat_bin_2           0.000862
124) ps_ind_02_cat_bin_0           0.000854
125) ps_ind_05_cat_bin_2           0.000850
126) ps_car_10_cat                 0.000793
127) ps_ind_12_bin                 0.000701
128) ps_car_09_cat_bin_0           0.000466
129) ps_car_04_cat_bin_0           0.000431
130) ps_ind_11_bin                 0.000218
131) ps_car_04_cat_bin_1           0.000194
132) ps_ind_13_bin                 0.000147
133) ps_ind_10_bin                 0.000074
Number of features before selection: 133
Number of features after selection: 67
```

With SelectFromModel we can specify which prefit classifier to use and what the threshold is for the feature importances. With the get_support method we can then limit the number of variables in the train data.

4 3. Modeling

4.1 3.0 K-fold CV with Out-of-Fold Prediction

4.1.1 3.0.1. OOF utility functions

Convert AUC score into Gini Normalised Coeficient

4.1.2 3.0.2 Xgboost K-fold & OOF function

In this part, we are going to use the native interface of XGB and LGB, so the following functions are tailor for this. For sure it would be easiler just to call the respective sklearn api, but the native interfaces provide some nice additional capability. For instance, the 'hist' option to use fast histogram in XGB is only available via the native interface as far as I know.

Also, we need to provide the following function to convert probability into rank for these two OOF function. The needs to use normalised rank instead of predicted probabilities will become appearent later in this notebook

The k-fold function for XGB to generate OOF predictions, this function is very much similar to its sklearn counter part. The difference is that we need to use the XGB interface to facilitate the classifer, also we provide an option cover probability into rank

4.1.3 2.0.3 LigthGBM K-fold & OOF function

The same function for LGB, this one is almost identifical to the one for XGB, apart from code that call the LightGBM interface

4.2 3.1. Generate level 1 OOF Predictions

Almost there to actually generate some level OOF output! last things to do is the prepare our train and test data for our dear machine learning algorithms, and create the StratifiedKFold objec

Here, I would like remind you that for stacking, I will use consistent fold distribution at ALL level for ALL your model.

```
In [43]: from crossValidation.cvsklearn import cross_validate_sklearn
         from crossValidation.cvxgb import cross_validate_xgb
         from crossValidation.cvlgb import cross_validate_lgb

         from sklearn.metrics import roc_auc_score

         from sklearn.model_selection import train_test_split
         from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
         from sklearn.naive_bayes import BernoulliNB
         from sklearn.linear_model import LogisticRegression
         from sklearn.model_selection import StratifiedKFold

         import xgboost as xgb
         import lightgbm as lgb
         import time

         from sklearn.model_selection import StratifiedKFold
```

4.2.1 3.1.1. Level One

- Random Forest 1
- Random Forest 2
- Logistic Regression
- BernoulliNB
- XGBoost
- LightGBM

```
In [44]: # Random Forest
         rf0=RandomForestClassifier(n_estimators=200, n_jobs=6, min_samples_split=5, max_depth=7
                                    criterion='gini', random_state=0)
```

```
# Random Forest
rf1=RandomForestClassifier(n_estimators=100, n_jobs=6, min_samples_split=5, max_depth=5
                           criterion='gini', random_state=0)

# Logistic Regression
logit=LogisticRegression(random_state=0, C=0.5)

# BernouliNB()
nb=BernoulliNB()

# XGBoost
xgb_params = {
    "booster"  :  "gbtree",
    "objective"         :  "binary:logistic",
    "tree_method": "hist",
    "eval_metric": "auc",
    "eta": 0.1,
    "max_depth": 5,
    "min_child_weight": 10,
    "gamma": 0.70,
    "subsample": 0.76,

    "colsample_bytree": 0.95,
    "nthread": 6,
    "seed": 0,
    'silent': 1
}

# LightGBoost
lgb_params = {
    'task': 'train',
    'boosting_type': 'dart',
    'objective': 'binary',
    'metric': {'auc'},
    'num_leaves': 22,
    'min_sum_hessian_in_leaf': 20,
    'max_depth': 5,
    'learning_rate': 0.1,  # 0.618580
    'num_threads': 6,
    'feature_fraction': 0.6894,
    'bagging_fraction': 0.4218,
    'max_drop': 5,
    'drop_rate': 0.0123,
    'min_data_in_leaf': 10,
    'bagging_freq': 1,
    'lambda_l1': 1,
    'lambda_l2': 0.01,
```

```
        'verbose': 1
    }


    base_models = [('rf0', [rf0, False, True]),('rf1', [rf1, False, True]),('nb0',[nb,True,
                   ('logit0', [logit, True, True]),
                   ('xgb0', [xgb_params, False, False]),
                   ('lgb0', [lgb_params, False, False,True])]


    testp = ML.Ensemble(nnew_train,
                        'target',
                        ['id','target'],
                        nnew_test,
                        'id',
                        base_models,
                        5)

    lv1_train, lv1_test = testp.FitModel()
```

```
-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
sklearn: rf0
--------------------
fold cv 0 AUC score is 0.604103, Gini_Norm score is 0.208206
fold cv 1 AUC score is 0.603331, Gini_Norm score is 0.206662
fold cv 2 AUC score is 0.597920, Gini_Norm score is 0.195841
fold cv 3 AUC score is 0.601838, Gini_Norm score is 0.203676
fold cv 4 AUC score is 0.611104, Gini_Norm score is 0.222207
cv AUC score is 0.603524, Gini_Norm score is 0.207048
it takes 168.060 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
sklearn: rf1
--------------------
fold cv 0 AUC score is 0.600872, Gini_Norm score is 0.201744
fold cv 1 AUC score is 0.599315, Gini_Norm score is 0.198630
fold cv 2 AUC score is 0.594168, Gini_Norm score is 0.188336
fold cv 3 AUC score is 0.597708, Gini_Norm score is 0.195416
fold cv 4 AUC score is 0.607792, Gini_Norm score is 0.215583
cv AUC score is 0.599804, Gini_Norm score is 0.199608
it takes 69.917 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
sklearn: nb0
--------------------
fold cv 0 AUC score is 0.585148, Gini_Norm score is 0.170296
fold cv 1 AUC score is 0.584793, Gini_Norm score is 0.169586
```

```
fold cv 2 AUC score is 0.575420, Gini_Norm score is 0.150840
fold cv 3 AUC score is 0.578942, Gini_Norm score is 0.157885
fold cv 4 AUC score is 0.589557, Gini_Norm score is 0.179115
cv AUC score is 0.582661, Gini_Norm score is 0.165321
it takes 16.040 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
sklearn: logit0
--------------------
fold cv 0 AUC score is 0.599980, Gini_Norm score is 0.199960
fold cv 1 AUC score is 0.596733, Gini_Norm score is 0.193467
fold cv 2 AUC score is 0.589817, Gini_Norm score is 0.179633
fold cv 3 AUC score is 0.597290, Gini_Norm score is 0.194580
fold cv 4 AUC score is 0.604584, Gini_Norm score is 0.209169
cv AUC score is 0.597644, Gini_Norm score is 0.195288
it takes 30.088 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
xgb: xgb0
--------------------
fold cv 0 AUC score is 0.614715, Gini_Norm score is 0.229429
fold cv 1 AUC score is 0.612743, Gini_Norm score is 0.225485
fold cv 2 AUC score is 0.610731, Gini_Norm score is 0.221462
fold cv 3 AUC score is 0.612769, Gini_Norm score is 0.225538
fold cv 4 AUC score is 0.621248, Gini_Norm score is 0.242497
cv AUC score is 0.614377, Gini_Norm score is 0.228755
it takes 47.556 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
lgb: lgb0
--------------------
fold cv 0 AUC score is 0.614035, Gini_Norm score is 0.228070
fold cv 1 AUC score is 0.610871, Gini_Norm score is 0.221743
fold cv 2 AUC score is 0.607883, Gini_Norm score is 0.215765
fold cv 3 AUC score is 0.612194, Gini_Norm score is 0.224388
fold cv 4 AUC score is 0.618460, Gini_Norm score is 0.236920
cv AUC score is 0.610806, Gini_Norm score is 0.221613
it takes 29.040 seconds to perform cross validation
```

```
In [45]: lv1_train.head()
```

Out[45]:

	lgb0	logit0	nb0	rf0	rf1	xgb0
0	0.100351	0.063840	0.003708	0.082085	0.084620	0.090450
1	0.079048	0.050483	0.015696	0.071762	0.074102	0.061526
2	0.108474	0.120619	0.551665	0.111038	0.111143	0.107630
3	0.169732	0.132846	0.895994	0.117746	0.121495	0.114748
4	0.150424	0.122391	0.979595	0.120591	0.122130	0.117688

4.2.2 3.2.1. Level Two

- Random Forest 1
- Logistic Regression
- XGBoost 1
- XGBoost 2
- LightGBM

```
In [46]: base_models = [('rf0', [rf0, True, True]),
                        ('logit0', [logit, True, True]),
                        ('xgb0', [xgb_params, False, False]),
                        ('xgb1', [xgb_params, True, False]),
                        ('lgb0', [lgb_params, True, False, False])]


         testp2 = ML.Ensemble(lv1_train,
                              nnew_train['target'],
                              [],
                              lv1_test,
                              [],
                              base_models,
                              5)

         lv2_train, lv2_test = testp2.FitModel()
```

```
-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
sklearn: rf0
--------------------
fold cv 0 AUC score is 0.614193, Gini_Norm score is 0.228386
fold cv 1 AUC score is 0.613372, Gini_Norm score is 0.226744
fold cv 2 AUC score is 0.610431, Gini_Norm score is 0.220863
fold cv 3 AUC score is 0.611640, Gini_Norm score is 0.223279
fold cv 4 AUC score is 0.619883, Gini_Norm score is 0.239766
cv AUC score is 0.613172, Gini_Norm score is 0.226344
it takes 92.444 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
sklearn: logit0
--------------------
fold cv 0 AUC score is 0.615599, Gini_Norm score is 0.231199
fold cv 1 AUC score is 0.613474, Gini_Norm score is 0.226949
fold cv 2 AUC score is 0.610038, Gini_Norm score is 0.220076
fold cv 3 AUC score is 0.614249, Gini_Norm score is 0.228497
fold cv 4 AUC score is 0.621234, Gini_Norm score is 0.242468
cv AUC score is 0.614442, Gini_Norm score is 0.228883
it takes 1.900 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
```

```
xgb: xgb0
--------------------
fold cv 0 AUC score is 0.614429, Gini_Norm score is 0.228858
fold cv 1 AUC score is 0.613014, Gini_Norm score is 0.226028
fold cv 2 AUC score is 0.610764, Gini_Norm score is 0.221527
fold cv 3 AUC score is 0.613311, Gini_Norm score is 0.226621
fold cv 4 AUC score is 0.620183, Gini_Norm score is 0.240366
cv AUC score is 0.557368, Gini_Norm score is 0.114736
it takes 15.505 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
xgb: xgb1
--------------------
fold cv 0 AUC score is 0.614429, Gini_Norm score is 0.228858
fold cv 1 AUC score is 0.613014, Gini_Norm score is 0.226028
fold cv 2 AUC score is 0.610764, Gini_Norm score is 0.221527
fold cv 3 AUC score is 0.613311, Gini_Norm score is 0.226621
fold cv 4 AUC score is 0.620183, Gini_Norm score is 0.240366
cv AUC score is 0.614354, Gini_Norm score is 0.228709
it takes 15.411 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
lgb: lgb0
--------------------
fold cv 0 AUC score is 0.614540, Gini_Norm score is 0.229080
fold cv 1 AUC score is 0.613246, Gini_Norm score is 0.226491
fold cv 2 AUC score is 0.610752, Gini_Norm score is 0.221503
fold cv 3 AUC score is 0.613822, Gini_Norm score is 0.227643
fold cv 4 AUC score is 0.621225, Gini_Norm score is 0.242450
cv AUC score is 0.614715, Gini_Norm score is 0.229430
it takes 6.112 seconds to perform cross validation
```

4.2.3 3.4.1. Level One

- Logistic Regression
- XGBoost

```
In [47]: xgb_lv3_params = {
             "booster"  :  "gbtree",
             "objective"         :  "binary:logistic",
             "tree_method": "hist",
             "eval_metric": "auc",
             "eta": 0.1,
             "max_depth": 2,
             "min_child_weight": 10,
             "gamma": 0.70,
             "subsample": 0.76,
```

```
        "colsample_bytree": 0.95,
        "nthread": 6,
        "seed": 0,
        'silent': 1
    }

    base_models = [('logit0', [logit, True, True]),
                   ('xgb0', [xgb_lv3_params, False, False])]


    testp3 = ML.Ensemble(lv2_train,
                         nnew_train['target'],
                         [],
                         lv2_test,
                         [],
                         base_models,
                         5)

    lv3_train, lv3_test = testp3.FitModel()
```

```
-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
sklearn: logit0
--------------------
fold cv 0 AUC score is 0.615440, Gini_Norm score is 0.230880
fold cv 1 AUC score is 0.613716, Gini_Norm score is 0.227432
fold cv 2 AUC score is 0.611214, Gini_Norm score is 0.222429
fold cv 3 AUC score is 0.614250, Gini_Norm score is 0.228500
fold cv 4 AUC score is 0.621568, Gini_Norm score is 0.243136
cv AUC score is 0.615068, Gini_Norm score is 0.230135
it takes 1.709 seconds to perform cross validation

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
xgb: xgb0
--------------------
fold cv 0 AUC score is 0.615583, Gini_Norm score is 0.231165
fold cv 1 AUC score is 0.613595, Gini_Norm score is 0.227190
fold cv 2 AUC score is 0.611367, Gini_Norm score is 0.222733
fold cv 3 AUC score is 0.613808, Gini_Norm score is 0.227616
fold cv 4 AUC score is 0.620881, Gini_Norm score is 0.241762
cv AUC score is 0.613617, Gini_Norm score is 0.227234
it takes 13.156 seconds to perform cross validation
```

5 4. Prediction: Average Level 3 outputs

We can always still do a simple weight average, to bring the two together and see if there any extra juice to be squeezed

```
In [48]: prediction = pd.DataFrame(columns=['predition'])
         prediction['predition'] = lv3_train.mean(axis=1)
         prediction.head()
```

```
Out[48]:    predition
         0   0.084893
         1   0.061568
         2   0.111607
         3   0.136439
         4   0.123040
```

```
In [49]: def auc_to_gini_norm(auc_score):return 2*auc_score-1
         print(auc_to_gini_norm(roc_auc_score(nnew_train['target'], prediction)))
```

```
0.229112873896
```

This is the final result for and its Gini score that I reach