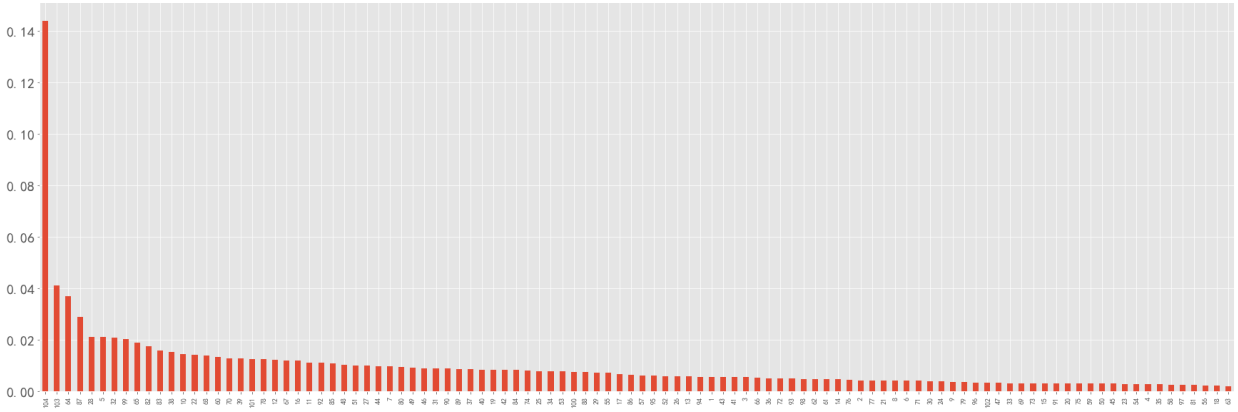
In [11]:
```
train = fullset.loc[train_idx]
test = fullset.loc[test_idx]
```

In [14]:
```
(fullset.ps_car_11_cat.value_counts()/fullset.shape[0]).plot(kind='bar',fig
plt.tick_params(axis='y', which='major', labelsize=20)
```



In [15]:
```
nominal_cols = metadata[(metadata.level == 'nominal') & (metadata.keep == T
nominal_cols
```

Out[15]:
```
['ps_ind_02_cat',
 'ps_ind_04_cat',
 'ps_ind_05_cat',
 'ps_car_01_cat',
 'ps_car_02_cat',
 'ps_car_04_cat',
 'ps_car_06_cat',
 'ps_car_07_cat',
 'ps_car_08_cat',
 'ps_car_09_cat',
 'ps_car_10_cat',
 'ps_car_11_cat']
```

In [16]:
```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown = "ignore",sparse=False)
trn_nominal_cols_enc = enc.fit_transform(train[nominal_cols])
tst_nominal_cols_enc = enc.transform(test[nominal_cols])
```

In [17]:
```
trn_nominal_cols_enc.shape
```

Out[17]:
```
(144626, 177)
```

In [18]:
```
tst_nominal_cols_enc.shape
```

Out[18]:
```
(892816, 177)
```

In [19]:
```
enc = OneHotEncoder(handle_unknown='ignore')
X = [['Male', 1], ['Female', 3], ['Female', 2]]
enc.fit(X)
```

Out[19]:
```
OneHotEncoder(handle_unknown='ignore')
```

In [20]:
```
enc.categories_
```

Out[20]:
```
[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
```

In [21]:
```
enc.transform([['Female', 1], ['Male', 4]]).toarray()
```

Out[21]:
```
array([[1., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

In [22]:
```
enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])
```

Out[22]:
```
array([['Male', 1],
       [None, 2]], dtype=object)
```

In [23]:
```
enc.get_feature_names_out(['gender', 'group'])
```

Out[23]:
```
array(['gender_Female', 'gender_Male', 'group_1', 'group_2', 'group_3'],
      dtype=object)
```

In [24]:
```
def add_noise(series, noise_level):
    return series * (1 + noise_level * np.random.randn(len(series)))

def target_encode(trn_series=None,
                  tst_series=None,
                  target=None,
                  min_samples_leaf=1,
                  smoothing=1,
                  noise_level=0):
    """
    Smoothing is computed like in the following paper by Daniele Micci-Barr
    https://kaggle2.blob.core.windows.net/forum-message-attachments/225952/
    trn_series : training categorical feature as a pd.Series
    tst_series : test categorical feature as a pd.Series
    target : target data as a pd.Series
    min_samples_leaf (int) : minimum samples to take category average into
    smoothing (int) : smoothing effect to balance categorical average vs pr
    """
    assert len(trn_series) == len(target)
    assert trn_series.name == tst_series.name
    temp = pd.concat([trn_series, target], axis=1)
    averages = temp.groupby(by=trn_series.name)[target.name].agg(["mean", "
    # 平滑
    smoothing = 1 / (1 + np.exp(-(averages["count"] - min_samples_leaf) / s
    prior = target.mean()
    averages[target.name] = prior * (1 - smoothing) + averages["mean"] * sm
    averages.drop(["mean", "count"], axis=1, inplace=True)

    ft_trn_series = pd.merge(
        trn_series.to_frame(trn_series.name),
        averages.reset_index().rename(columns={'index': target.name, target
        on=trn_series.name,
        how='left')['average'].rename(trn_series.name + '_mean').fillna(pri

    ft_trn_series.index = trn_series.index
    ft_tst_series = pd.merge(
        tst_series.to_frame(tst_series.name),
        averages.reset_index().rename(columns={'index': target.name, target
        on=tst_series.name,
        how='left')['average'].rename(trn_series.name + '_mean').fillna(pri

    ft_tst_series.index = tst_series.index
    return add_noise(ft_trn_series, noise_level), add_noise(ft_tst_series,
```

In [25]:
```
train_encoded, test_encoded = target_encode(train["ps_car_11_cat"],
                              test["ps_car_11_cat"],
                              target=train.target,
                              min_samples_leaf=100,
                              smoothing=10,
                              noise_level=0.01)
```

In [26]:
```
train["ps_car_11_cat_tar_enc"] = train_encoded.astype('float64')
test['ps_car_11_cat_tar_enc'] = test_encoded.astype('float64')
```

In [27]:
```
cols_to_drop = metadata[metadata['imputation'] == 'remove'].index.tolist()
```

In [28]:
```
cols_to_drop.append('ps_car_11_cat')
```

In [29]:
```
train_clean = train.drop(cols_to_drop,axis=1)
```

In [30]:
```
test_clean = test.drop(cols_to_drop,axis=1)
```

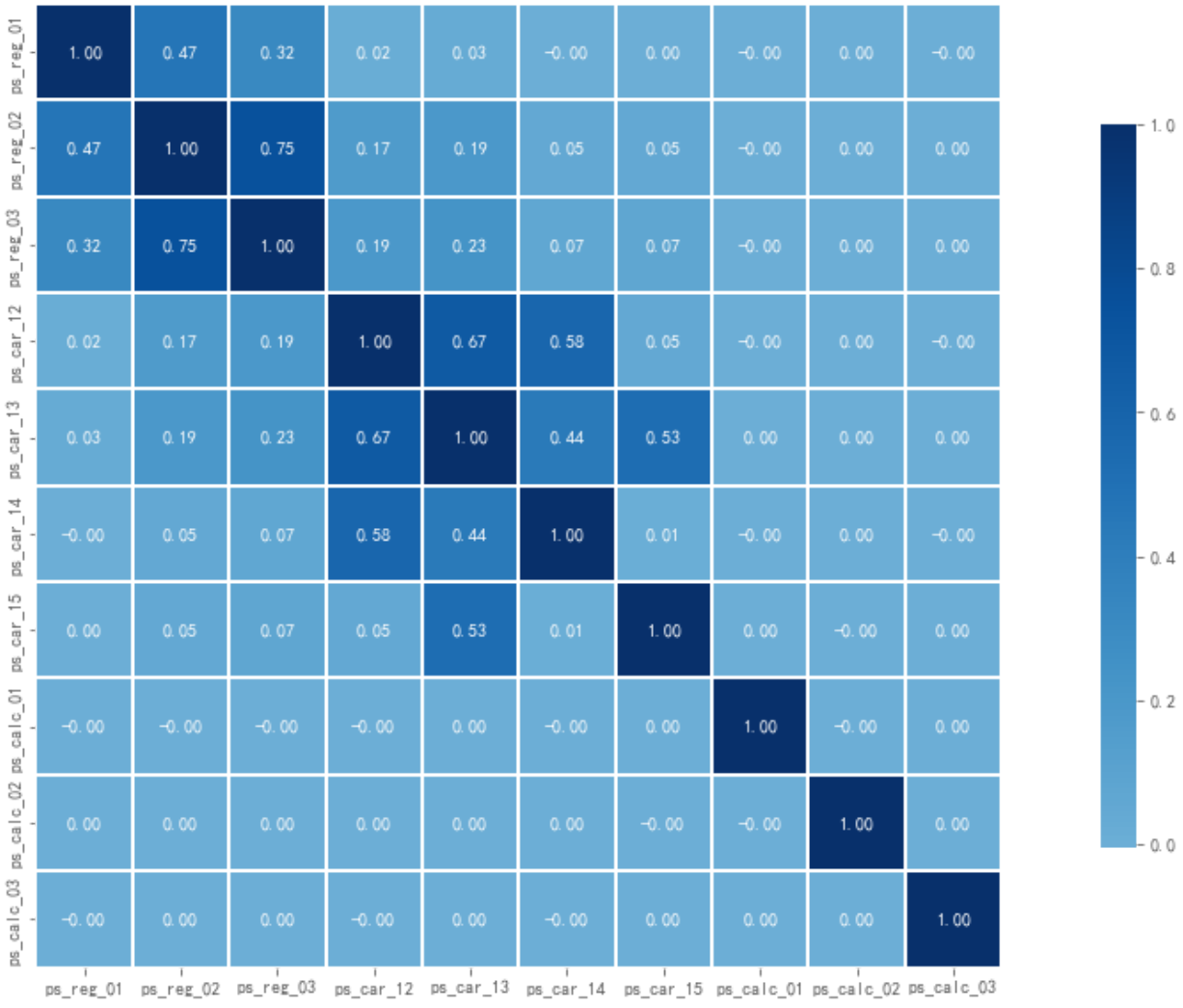
In [ ]:
```
train_clean.to_csv('train_clean.csv')
test_clean.to_csv('test_clean.csv')
```

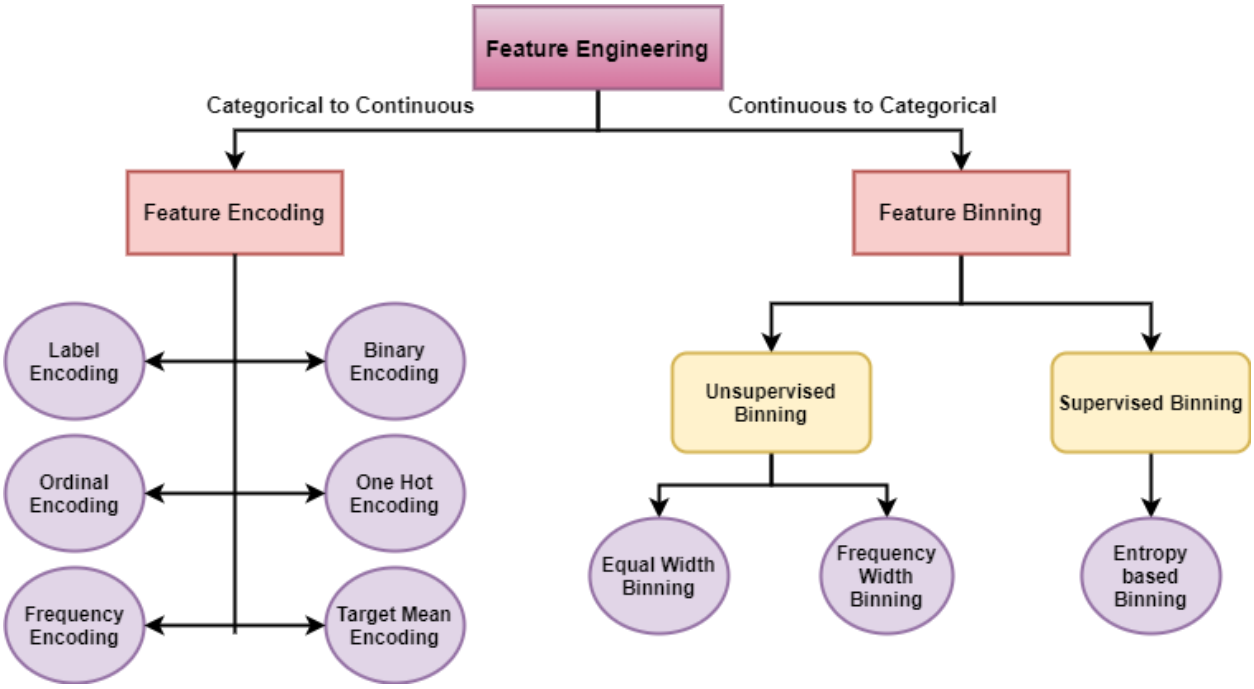
In [31]:
```
fullset = pd.concat([train_clean,test_clean],ignore_index=True)
```

In [32]:
```
continuous_cols = metadata[(metadata.level == 'interval')|(metadata.level =
```

In [33]:
```
plt.figure(figsize = [20,10])
sns.heatmap(data = fullset[continuous_cols].corr(),
            vmax=1,
            center=0,
            square=True,
           annot = True,
          fmt='.2f',
           cmap = 'Blues',
           linewidths = .3,
           cbar_kws={"shrink": .75})
```

Out[33]: <AxesSubplot:>

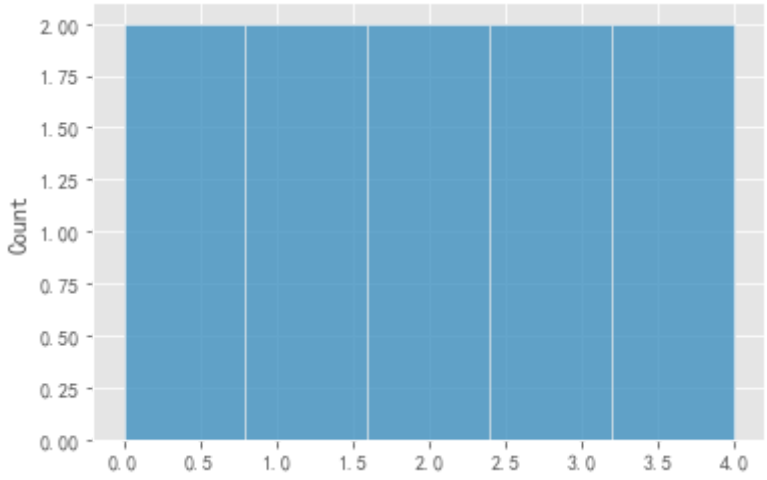




In [34]:
```
import pandas as pd
value_list = [0, 10, 20, 59, 61, 79, 80, 90, 99, 100]

value_freq_bins = pd.qcut(value_list, q=5)

value_dis_bins = pd.cut(value_list, bins=5)
```

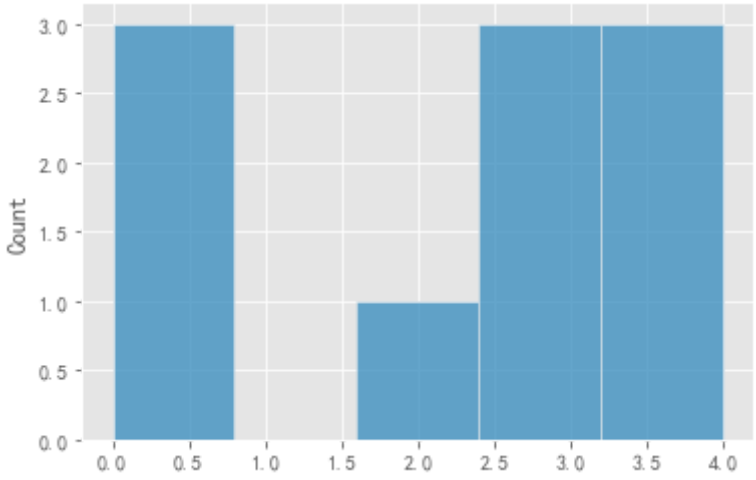
In [36]:
```
sns.histplot(value_freq_bins.codes)
```

Out[36]: <AxesSubplot:ylabel='Count'>

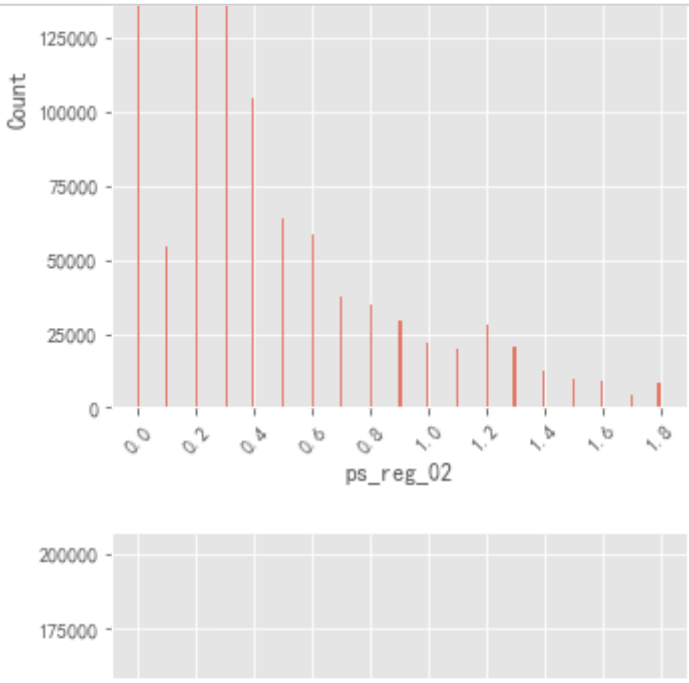


In [37]:
```
sns.histplot(value_dis_bins.codes)
```

Out[37]: <AxesSubplot:ylabel='Count'>



In [38]:
```
for col in continuous_cols:
    sns.displot(fullset[col])
    plt.xticks(ticks = np.linspace(start = fullset[col].min(),
                                   stop = fullset[col].max(),
                                   num = 10),rotation = 45)
```



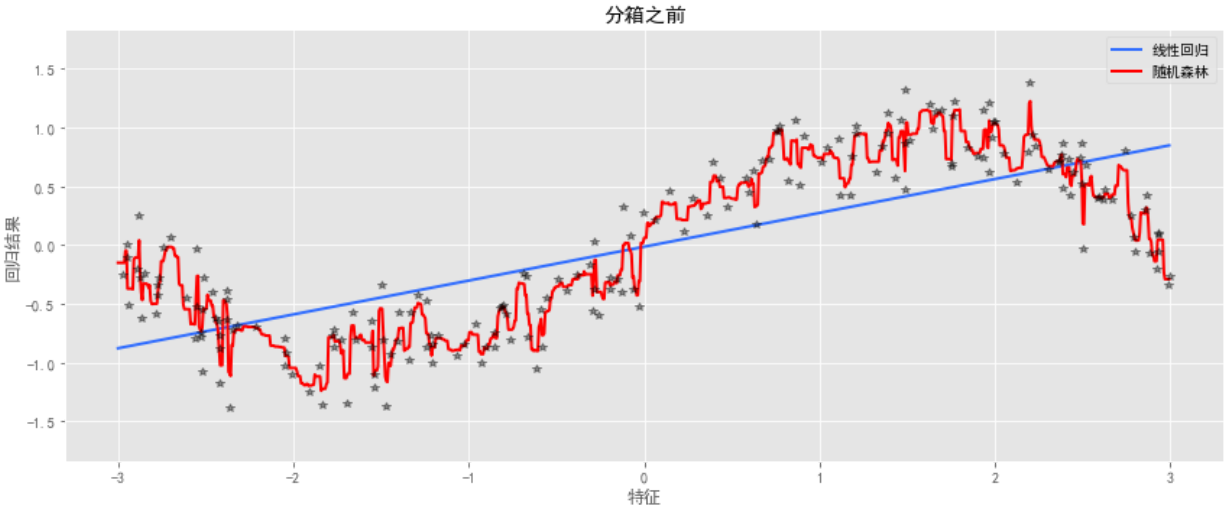
In [39]:
```
def woe_iv_encoding(data, feat, target, max_intervals, verbose = False):

    feat_bins = pd.qcut(x = data[feat], q = max_intervals, duplicates='drop
    gi = pd.crosstab(feat_bins,data[target])
    gb = pd.Series(data=data[target]).value_counts()

    bad =
    good =

    # 计算woe
    woe =

    # 计算iv
    iv =
    # 计算整个特征的iv
    f_iv = iv.sum()  # 5.2958917587905745
    if verbose == True:
        print(f"根据当前的间隔数{max_intervals}，特征{feat}所计算的总information
        print('='*80)

    # 进行映射操作
    dic = iv.to_dict()

    iv_bins = feat_bins.map(dic)

    return iv_bins.astype('float64')
```

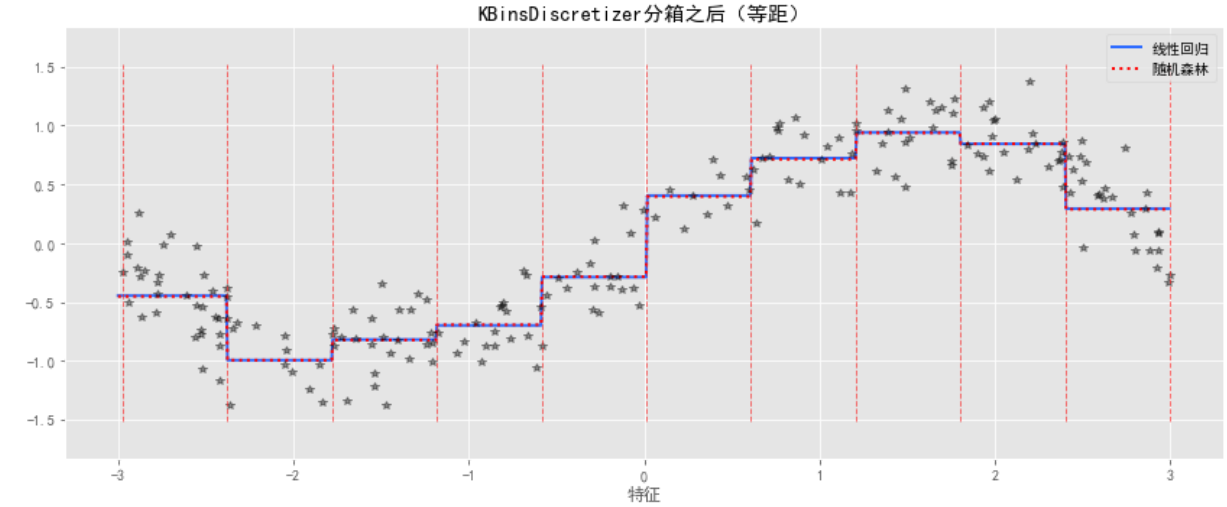
Information Value (IV)	Predictive Power
< 0.02	useless for prediction
0.02 to 0.1	weak predictor
0.1 to 0.3	medium predictor
0.3 to 0.5	strong predictor
> 0.5	suspicious or too good to be true

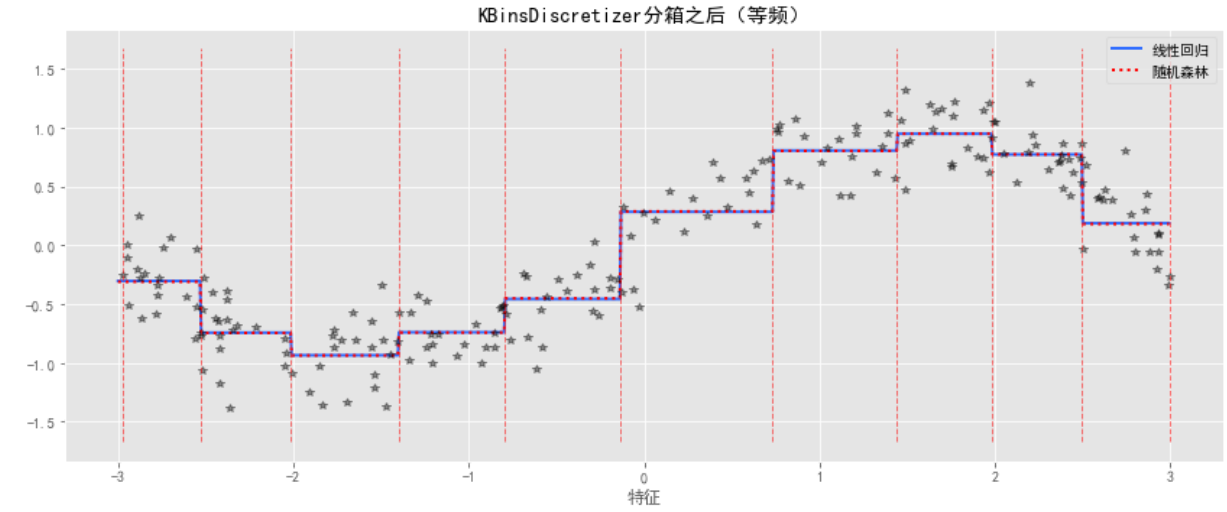
In [40]:
```
for col in continuous_cols:
    fullset[f"{col}_woe"] = woe_iv_encoding(data = fullset, feat = col, tar
    #sns.displot(fullset_copy[f"{col}_woe"])
```

In [41]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.ensemble import RandomForestRegressor

# construct the dataset
rnd = np.random.RandomState(2022)
X = rnd.uniform(-3, 3, size=200)
y = np.sin(X) + rnd.normal(size=len(X)) / 4
X = X.reshape(-1, 1)

# transform the dataset with KBinsDiscretizer
enc1 = KBinsDiscretizer(n_bins=10, encode="onehot", strategy = 'uniform')
enc2 = KBinsDiscretizer(n_bins=10, encode="onehot", strategy = 'quantile')
X_binned_unif = enc1.fit_transform(X)
X_binned_quan = enc2.fit_transform(X)

# predict with original dataset
fig, (ax1, ax2, ax3) = plt.subplots(ncols=1,nrows = 3, sharey=True, figsize
line = np.linspace(-3, 3, 1000, endpoint=False).reshape(-1, 1)
reg = LinearRegression().fit(X, y)


ax1.plot(line, reg.predict(line), linewidth=2, color="#306dff", label="线性回
reg = RandomForestRegressor(random_state=0).fit(X, y)
ax1.plot(line, reg.predict(line), linewidth=2, color="red", label="随机森林"
ax1.plot(X[:, 0], y, "*", c="k",alpha = .4)
ax1.legend(loc="best")
ax1.set_ylabel("回归结果")
ax1.set_xlabel("特征")
ax1.set_title("分箱之前")

# predict with transformed dataset
line_binned_unif = enc1.transform(line)
reg = LinearRegression().fit(X_binned_unif, y)
ax2.plot(
    line,
    reg.predict(line_binned_unif),
    linewidth=2,
    color="#306dff",
    linestyle="-",
    label="线性回归",
)

reg = RandomForestRegressor(random_state=0).fit(X_binned_unif, y)
ax2.plot(
    line,
    reg.predict(line_binned_unif),
    linewidth=2,
    color="red",
    linestyle=":",
    label="随机森林",
)
ax2.plot(X[:, 0], y, "*", c="k", alpha = .4)
```

```
ax2.vlines(enc1.bin_edges_[0], *plt.gca().get_ylim(), linewidth=1,alpha= .6
ax2.legend(loc="best")
ax2.set_xlabel("特征")
ax2.set_title("KBinsDiscretizer分箱之后（等距）")

# predict with transformed dataset
line_binned_quan = enc2.transform(line)
reg = LinearRegression().fit(X_binned_quan, y)
ax3.plot(
    line,
    reg.predict(line_binned_quan),
    linewidth=2,
    color="#306dff",
    linestyle="-",
    label="线性回归",
)

reg = RandomForestRegressor(random_state=0).fit(X_binned_quan, y)
ax3.plot(
    line,
    reg.predict(line_binned_quan),
    linewidth=2,
    color="red",
    linestyle=":",
    label="随机森林",
)
ax3.plot(X[:, 0], y, "*", c="k", alpha = .4)
ax3.vlines(enc2.bin_edges_[0], *plt.gca().get_ylim(), linewidth=1, alpha= .
ax3.legend(loc="best")
ax3.set_xlabel("特征")
ax3.set_title("KBinsDiscretizer分箱之后（等频）")

plt.tight_layout()
plt.show()
```


分箱之前
线性回归
随机森林
回归结果
特征


KBinsDiscretizer分箱之后（等距）
线性回归
随机森林
特征


KBinsDiscretizer分箱之后（等频）
线性回归
随机森林
特征

In [42]:
```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=Fal

interactions = pd.DataFrame(data=poly.fit_transform(fullset[continuous_cols
                            columns=poly.get_feature_names_out(continuous_c


interactions.drop(continuous_cols, axis=1, inplace=True)  # Remove the orig
# Concat the interaction variables to the train data
print('特征交互前，训练集有 {}个变量 '.format(fullset.shape[1]))
fullset = pd.concat([fullset, interactions], axis=1)
print('特征交互后，训练集有 {}个变量'.format(fullset.shape[1]))
```

```
特征交互前，训练集有 67个变量
特征交互后，训练集有 112个变量
```

Step 6:

- 简述【特征交互】的意义和原理
- 请叙述 PolynomialFeatures 的 degree、interaction_only、include_bias的含义

In [45]:
```
train
```

Out[45]:

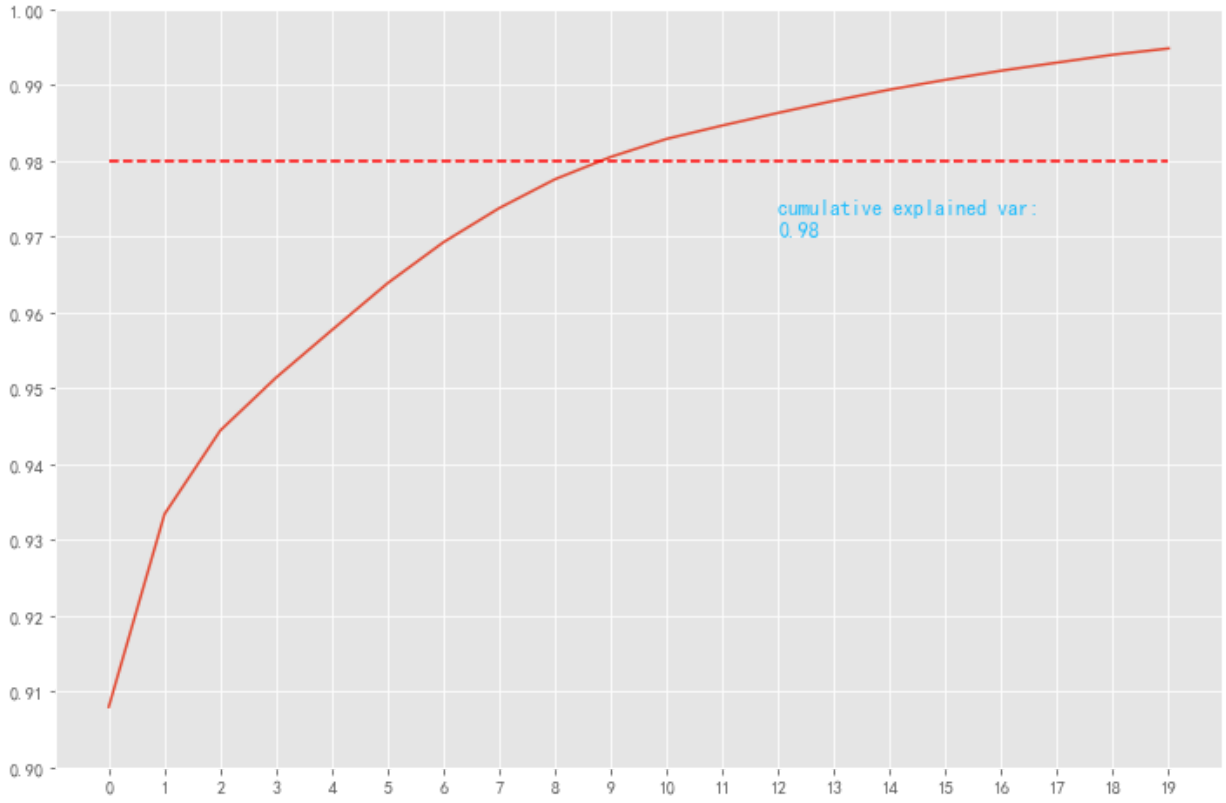
	id	target	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	ps_ind_05_cat	ps_ind_
0	744529	0.0	2	1	9	1	0	
1	673907	0.0	1	1	2	1	0	
2	730796	0.0	7	1	6	0	4	
3	306142	0.0	1	1	3	0	0	
4	1102701	0.0	6	2	7	1	0	
...	...	...	...	...	...	...	...	
144621	1487473	1.0	5	1	7	0	3	
144622	1487566	1.0	1	1	5	0	0	
144623	1487716	1.0	0	1	6	1	0	
144624	1487748	1.0	0	2	2	0	0	
144625	1487866	1.0	1	2	1	0	0	

144626 rows × 60 columns

In [46]:
```
from sklearn.decomposition import PCA

X = train.drop(['id', 'target'], axis=1)
y = train.target

n_comp = 20
print('\nPCA执行中...')
pca = PCA(n_components=n_comp, svd_solver='full', random_state=1001)
X_pca = pca.fit_transform(X)
print('Total Explained variance: %.4f' % pca.explained_variance_ratio_.sum(


plt.figure(figsize = [12,8])

pd.Series(pca.explained_variance_ratio_).cumsum().plot()

plt.plot(range(n_comp),[0.98]*20, 'r--')
plt.xticks(ticks = range(n_comp))
plt.yticks(ticks = np.linspace(0.9,1,11))
plt.text(12,0.97,'cumulative explained var: \n0.98',fontsize = 12, color =

plt.figure(figsize=(10,10))
plt.scatter(X_pca[:,0],X_pca[:,1],c=y,cmap='Set1',alpha = .7)
plt.xlabel('pc1')
plt.ylabel('pc2')
plt.title(
        "第一第二主成分散点分布图")
plt.xlabel("第一主成分解释 %.1f %% 方差" % (
        pca.explained_variance_ratio_[0] * 100.0))
plt.ylabel("第二主成分解释 %.1f %% 方差" % (
        pca.explained_variance_ratio_[1] * 100.0))
```

```
PCA执行中...
Total Explained variance: 0.9948
```

Out[46]:
```
Text(0, 0.5, '第二主成分解释 2.5 % 方差')
```


1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.90
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
cumulative explained var:
0.98

第一第二主成分散点分布图

Step 6:

- 对上面代码每一行进行comment
- 上面的方法有问题吗？如果有问题，出在哪里，怎么处理？

Step 7:

- GBDT+LR的优势有哪些？解决什么问题？

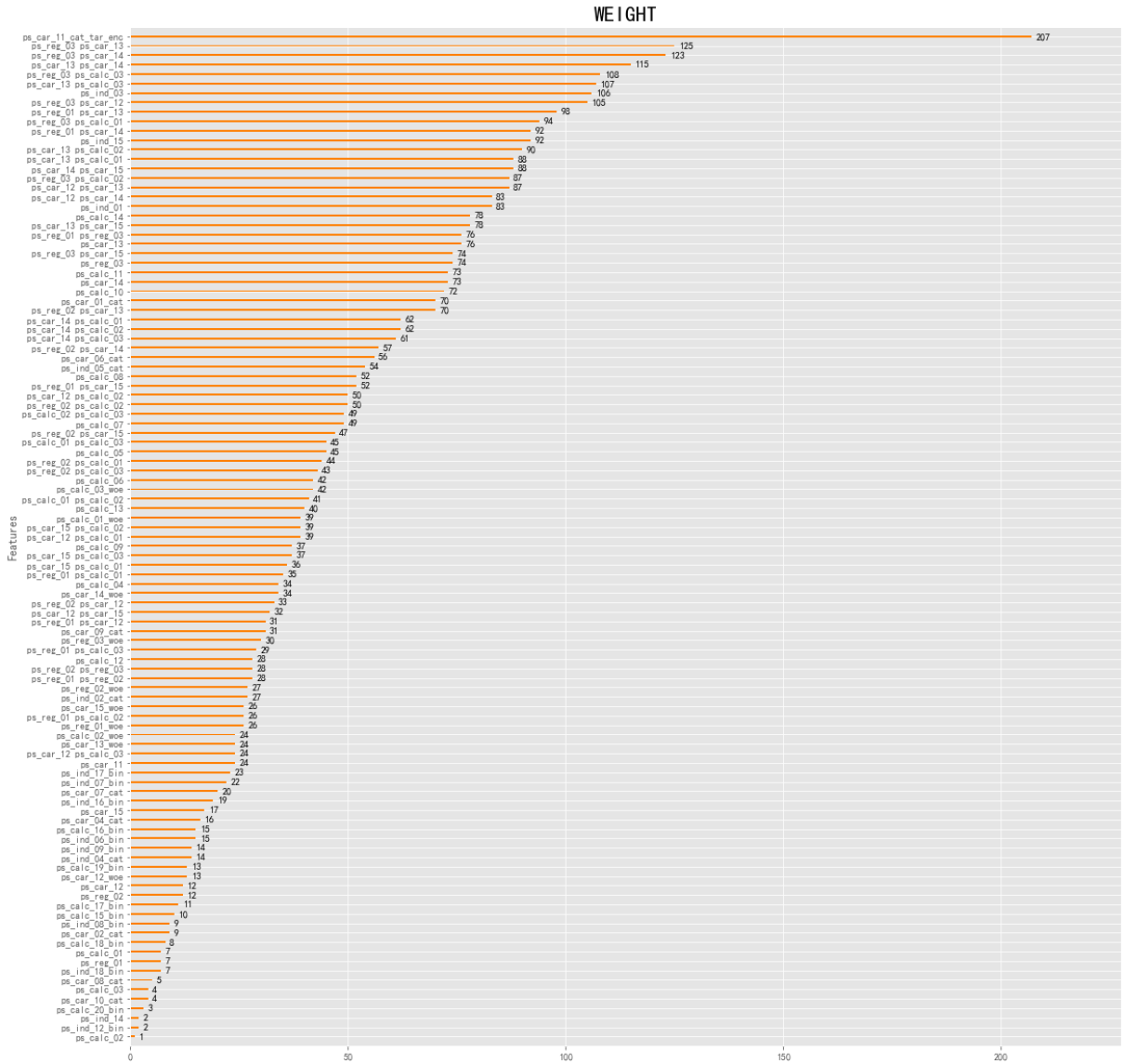
In [47]:
```
from sklearn.feature_selection import VarianceThreshold

selector = VarianceThreshold(threshold=.01)
selector.fit(fullset.drop(['id', 'target'], axis=1)) # Fit to train without

f = np.vectorize(lambda x : not x) # Function to toggle boolean array eleme

v = fullset.drop(['id', 'target'], axis=1).columns[f(selector.get_support()
print('{} variables have too low variance.'.format(len(v)))
```

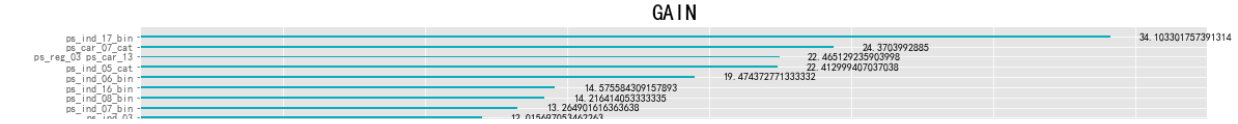
```
19 variables have too low variance.
```

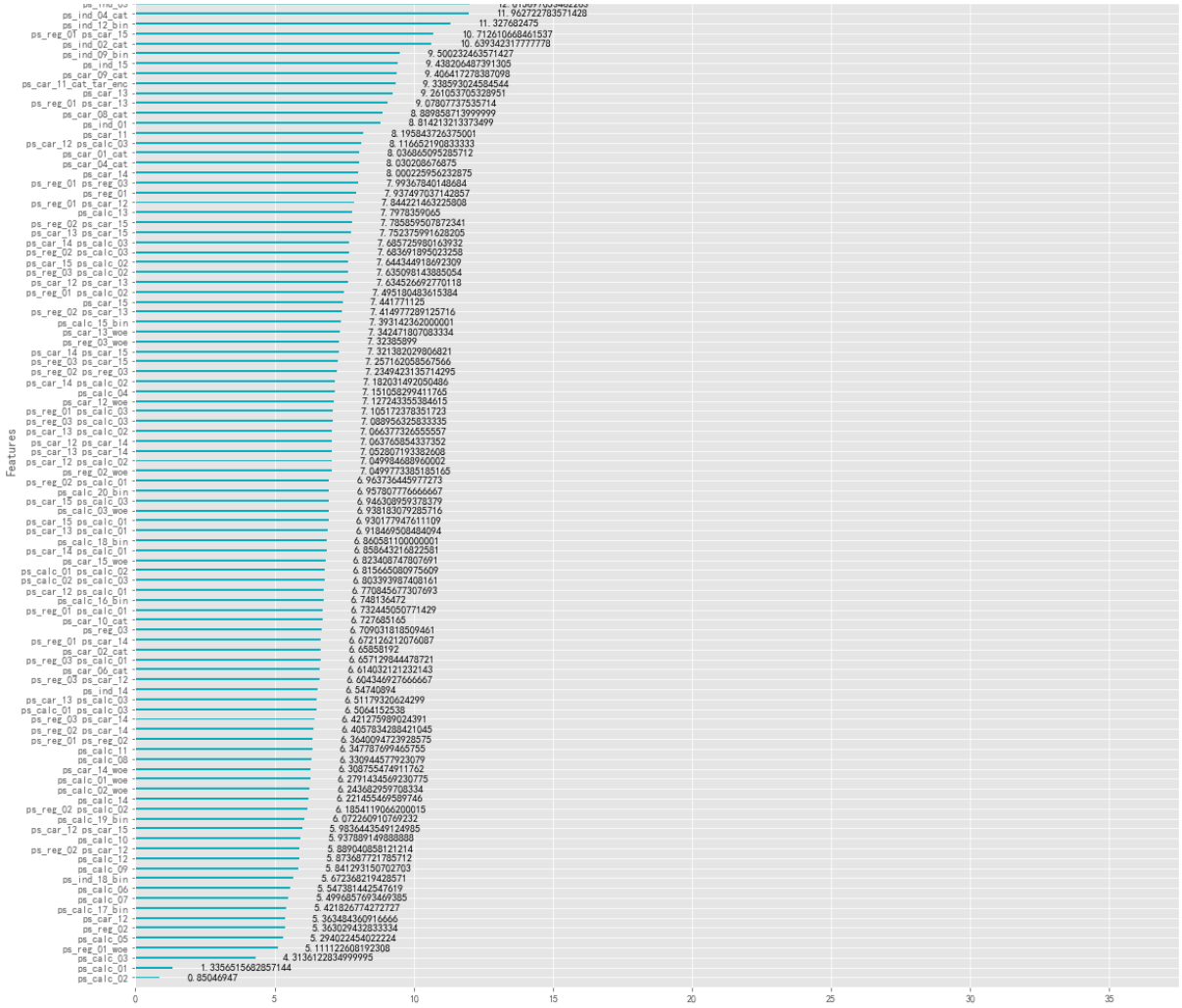
In [48]:
```
from xgboost import XGBClassifier
from xgboost import plot_importance

plt.figure(figsize = [100,20])

X = fullset.loc[train.index].drop(['id', 'target'], axis=1)
y = fullset.loc[train.index].target

model = XGBClassifier()

model.fit(X, y)
# plot feature importance
```

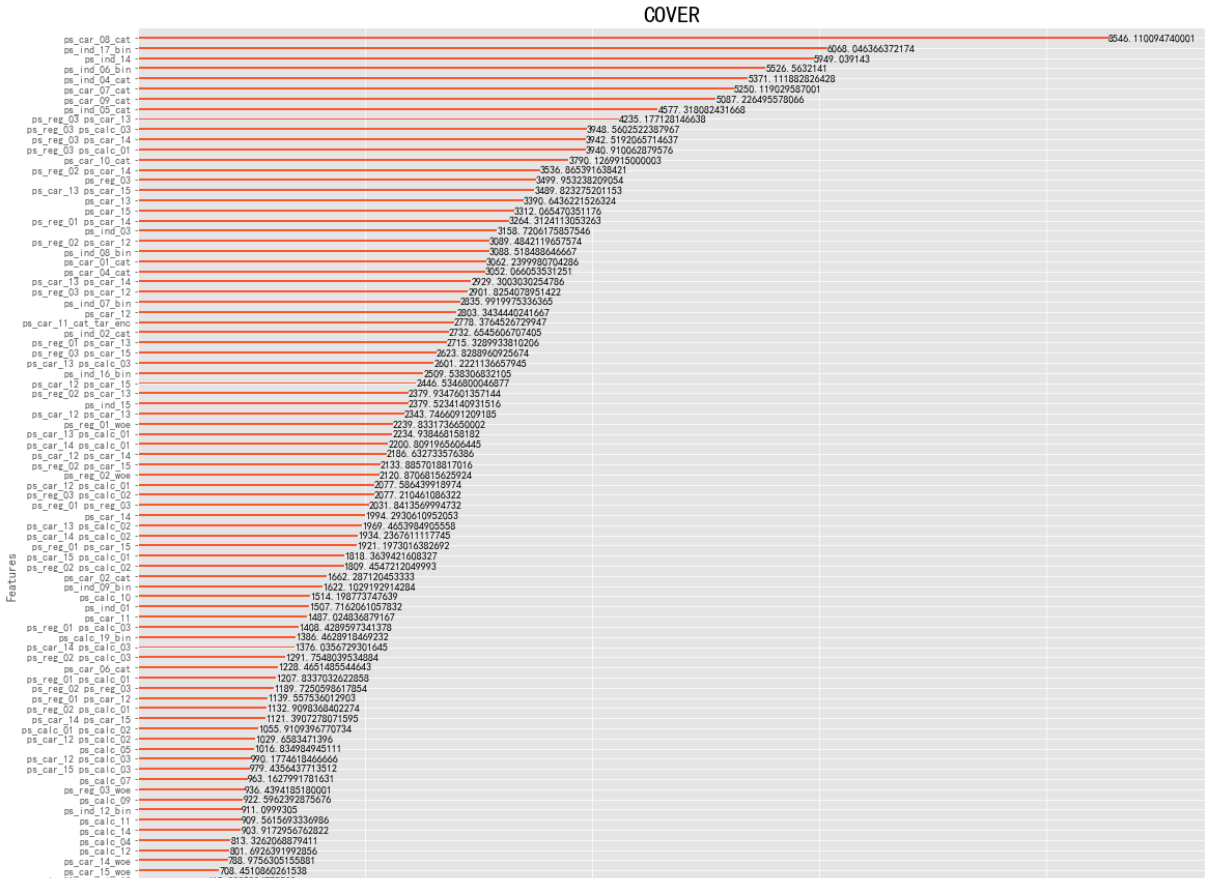
Out[48]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_sta
te=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)

<Figure size 7200x1440 with 0 Axes>
```

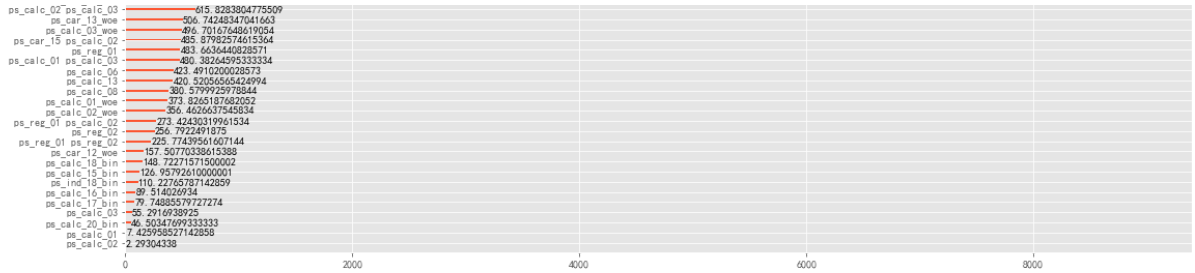
In [49]:
```
# define subplot grid
fig, axs = plt.subplots(nrows=5, ncols=1, figsize=(15, 80))
plt.tight_layout()
plt.subplots_adjust(hspace=0.1)

types = ['weight', 'gain', 'cover', 'total_gain', 'total_cover']
# loop through tickers and axes
colors = ['#ff7f01','#08aebd','#fc5531','#139948','#8950fe']
for ty, ax, color in zip(types, axs.ravel(), colors):
    # filter df for ticker and plot on specified axes
    plot_importance(ax = ax, booster = model,importance_type=ty, color = co

    # chart formatting
    ax.set_title(ty.upper(),fontsize = 22)
    ax.set_xlabel("")

plt.show()
```



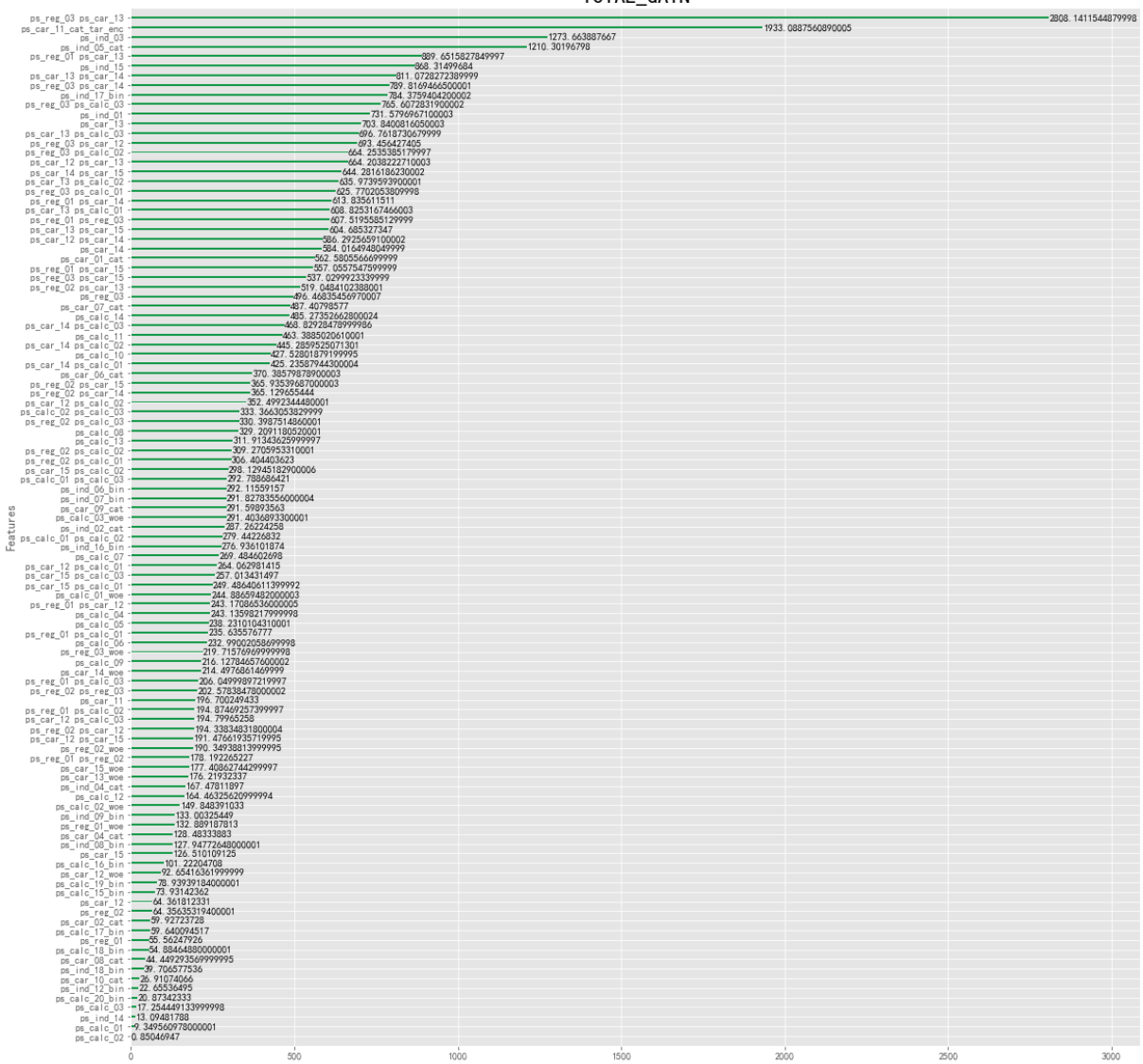



Features
ps_ind_04_cat
ps_ind_12_bin
ps_reg_01 ps_car_15
ps_ind_02_cat
ps_ind_09_bin
ps_ind_15
ps_car_09_cat
ps_car_11_cat_tar_enc
ps_car_13
ps_reg_01 ps_car_13
ps_car_08_cat
ps_ind_01
ps_car_11
ps_car_12 ps_calc_03
ps_car_01_cat
ps_car_04_cat
ps_car_14
ps_reg_01 ps_reg_03
ps_reg_01
ps_reg_01 ps_car_12
ps_car_13
ps_car_12 ps_car_13
ps_reg_01 ps_calc_02
ps_car_15
ps_reg_02 ps_car_13
ps_calc_15_bin
ps_car_13_woe
ps_reg_03_woe
ps_car_14 ps_car_15
ps_reg_03 ps_car_15
ps_reg_02 ps_reg_03
ps_car_14 ps_calc_02
ps_calc_04
ps_car_12_woe
ps_reg_01 ps_calc_03
ps_reg_03 ps_calc_03
ps_car_13 ps_calc_02
ps_car_12 ps_car_14
ps_car_13 ps_car_14
ps_car_12 ps_calc_02
ps_reg_02_woe
ps_reg_02 ps_calc_01
ps_calc_20_bin
ps_car_15 ps_calc_03
ps_calc_03_woe
ps_car_15 ps_calc_01
ps_car_13 ps_calc_01
ps_calc_18_bin
ps_car_14 ps_calc_01
ps_car_15_woe
ps_calc_01 ps_calc_02
ps_calc_02 ps_calc_03
ps_car_12 ps_calc_01
ps_calc_16_bin
ps_reg_01 ps_calc_01
ps_car_10_cat
ps_reg_03
ps_reg_01 ps_car_14
ps_car_02_cat
ps_reg_03 ps_calc_01
ps_car_06_cat
ps_reg_03 ps_car_12
ps_ind_14
ps_car_13 ps_calc_03
ps_calc_01 ps_calc_03
ps_reg_03 ps_car_14
ps_reg_02 ps_car_14
ps_reg_01 ps_reg_02
ps_calc_11
ps_calc_08
ps_car_14_woe
ps_calc_01_woe
ps_calc_02_woe
ps_calc_14
ps_reg_02 ps_calc_02
ps_calc_19_bin
ps_car_12 ps_car_15
ps_calc_10
ps_reg_02 ps_car_12
ps_calc_12
ps_calc_09
ps_ind_18_bin
ps_calc_06
ps_calc_07
ps_calc_17_bin
ps_car_12
ps_reg_02
ps_calc_05
ps_reg_01_woe
ps_calc_03
ps_calc_01
ps_calc_02
1. 3356515682857144
0. 85046947
0
5
10
15
20
25
30
35

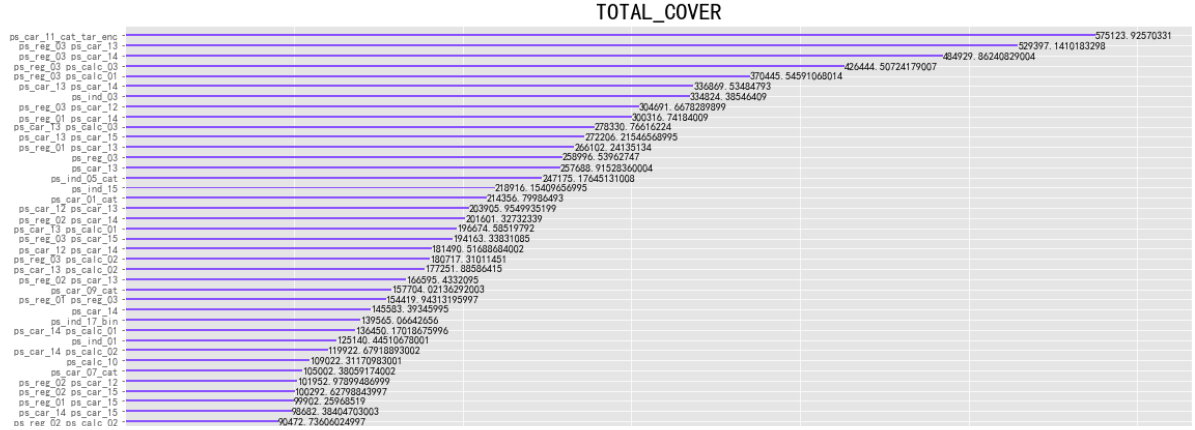

COVER
Features
ps_car_08_cat
ps_ind_17_bin
ps_ind_14
ps_ind_06_bin
ps_ind_04_cat
ps_car_07_cat
ps_car_09_cat
ps_ind_05_cat
ps_reg_03 ps_car_13
ps_reg_03 ps_calc_03
ps_reg_03 ps_car_14
ps_reg_03 ps_car_01
ps_car_10_cat
ps_reg_02 ps_car_14
ps_reg_03
ps_car_13 ps_car_15
ps_car_13
ps_car_15
ps_reg_01 ps_car_14
ps_ind_03
ps_reg_02 ps_car_12
ps_ind_08_bin
ps_car_01_cat
ps_car_04_cat
ps_car_13 ps_car_14
ps_reg_03 ps_car_12
ps_ind_07_bin
ps_car_12
ps_car_11_cat_tar_enc
ps_ind_02_cat
ps_reg_01 ps_car_13
ps_reg_03 ps_car_15
ps_car_13 ps_calc_03
ps_ind_16_bin
ps_car_12 ps_car_15
ps_reg_02 ps_car_13
ps_ind_15
ps_car_12 ps_car_13
ps_reg_01_woe
ps_car_13 ps_calc_01
ps_car_14 ps_calc_01
ps_car_12 ps_car_14
ps_reg_02 ps_car_15
ps_reg_02_woe
ps_car_12 ps_calc_01
ps_reg_03 ps_calc_02
ps_reg_01 ps_reg_03
ps_car_14
ps_car_13 ps_calc_02
ps_car_14 ps_calc_02
ps_reg_01 ps_car_15
ps_car_15 ps_calc_01
ps_reg_02 ps_calc_02
ps_car_02_cat
ps_ind_09_bin
ps_calc_10
ps_ind_01
ps_car_11
ps_reg_01 ps_calc_03
ps_calc_19_bin
ps_car_14 ps_calc_03
ps_reg_02 ps_calc_03
ps_car_06_cat
ps_reg_01 ps_calc_01
ps_reg_02 ps_reg_03
ps_reg_01 ps_car_12
ps_reg_02 ps_calc_01
ps_car_14 ps_car_15
ps_calc_01 ps_calc_02
ps_car_12 ps_calc_02
ps_calc_05
ps_car_12 ps_calc_03
ps_car_15 ps_calc_03
ps_calc_07
ps_reg_03_woe
ps_calc_09
ps_ind_12_bin
ps_calc_11
ps_calc_14
ps_calc_04
ps_calc_12
ps_car_14_woe
ps_car_15_woe
8546. 110094740001
6068. 046366372174
5949. 039143
5526. 5632141
5371. 111882826428
5250. 119029587001
5087. 226495578066
4577. 318082431668
4235. 177128146638
3948. 5602522387967
3942. 519206571 4637
3940. 910062879576
3790. 12699150000003
3536. 865391638421
3499. 953238209054
3489. 823275201153
3390. 64362215263 24
3312. 665470351176
3264. 312411305263
3158. 7206175857546
3089. 4842119657574
3088. 518488646667
3062. 23999807042 86
3052. 066053531251
2929. 3003030254786
2901. 8254078951422
2835. 99199753363 65
2803. 34344402416 67
2778. 3764526729947
2732. 6845606707405
2715. 3289933810206
2623. 8288960925674
2601. 222113666579 45
2509. 538306832105
2446. 53468000468 77
2379. 9347601357144
2379. 5234140931516
2343. 7466091209185
2239. 8331736650002
2234. 9384681581 82
2200. 8091965606445
2186. 6327335763 86
2133. 8857018817016
2120. 8706815625924
2077. 5864399187974
2077. 2104610863 22
2031. 8413569994732
1994. 2930610952053
1969. 4653984905558
1934. 2367611117745
1921. 1973016382692
1818. 3639421608327
1809. 4547212049993
1662. 2871204533 33
1622. 1029192914284
1514. 1987737476 39
1507. 7162061057832
1487. 0248360879167
1408. 4289597341378
1386. 4628918469232
1376. 0356729301645
1291. 7548039534884
1228. 4651485544643
1207. 8337032622858
1189. 7250598617854
1139. 5575360129 93
1132. 9098368402274
1121. 3907278071595
1055. 9109396770734
1029. 6583471396
1016. 8349849451 11
990. 1774618466666
979. 4354437713512
963. 1627991781631
936. 4394185180001
922. 5962392875676
911. 0999305
909. 5615693336986
903. 9172956762822
813. 3262068879411
801. 6926391992856
788. 9756305155881
708. 4510860261538



TOTAL_GAIN



TOTAL_COVER



In [51]:
```
# 特征工程的strategy作为key，对应的变量名组成的list作为value
from sklearn.feature_selection import SelectFromModel
feat_dict = {}
for thres in ['median','mean','1.25*mean']:
    model_select = SelectFromModel(model, threshold=thres, prefit=True)
    print(f'筛选前总计：{X.shape[1]}个特征')
    n_features = model_select.transform(X.values).shape[1]
    print(f'筛选后总计：  {n_features}个特征【{thres}】 ')
    print('#'*60)
    selected_vars = list(X.columns[model_select.get_support()])
    feat_dict[thres] = selected_vars
```

```
筛选前总计：110个特征
筛选后总计：  55个特征【median】
############################################################
筛选前总计：110个特征
筛选后总计：  28个特征【mean】
############################################################
筛选前总计：110个特征
筛选后总计：  13个特征【1.25*mean】
############################################################
```

In [52]:
```
for ty in types:
    feat_dict[ty] = list(model.get_booster().get_score(importance_type=ty).
```

In [54]:
```
final_train = fullset.loc[train.index][feat_dict['mean']+['target']]
final_test = fullset.loc[test.index][feat_dict['mean']]
```

In [55]:
```
final_train.to_csv("final_train.csv")
final_test.to_csv("final_test.csv")
```