前期准备

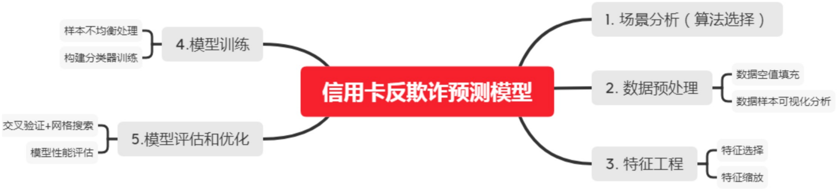
目标:通过利用信用卡的历史交易数据，进行机器学习，构建信用卡反欺诈预测模型，提前发现客户信用卡被盗刷的事件。

数据集介绍

数据集（Credit Card Fraud Detection）包含由欧洲持卡人于2013年9月使用信用卡进行交易的数据。此数据集显示两天内发生的交易，其中284,807笔交易中有492笔被盗刷。数据集非常不平衡，积极的类（被盗刷）占所有交易的0.172%。

信用卡欺诈检测问题的特点是样本的不均衡性，欺诈交易数量较少，所以可以训练一些不平衡样本的处理方式。

由于保密问题，无法提供有关数据的原始功能和更多背景信息。针对我们的目标，如果发生被盗刷，则取值1，否则为0。



场景分析

数据是持卡人两天内信用卡交易数据，要解决的问题是预测持卡人是否会发生信用卡被盗刷

判定信用卡持卡人是否会发生被盗刷是一个二元分类问题

算法选择分类算法（例如：我们选择 Logistic Regression 作为我们的baseline）

提示： 特征V1至V28是经过PCA处理，而特征Time和Amount的数据规格与其他特征差别较大，需要对其做特征缩放，尤其是对大小分布敏感的算法（如LR）一定要进行缩放处理

Amount：可以直接缩放(0,1)

Time：数据提供单位秒，可以考虑转会成小时（对应每天的时间）.

数据预处理

In [ ]:

```
# Imports
# Numpy,Pandas
import numpy as np
import pandas as pd
import datetime

# matplotlib,seaborn,pyecharts
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline

# import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler


#  忽略弹出的warnings
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.float_format', lambda x: '%.4f' % x)
```

In [ ]:

```
data_df = pd.read_csv("creditcard.csv")
print(data_df.shape)
data_df.head()
```

```
(284807, 31)
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0000	-1.3598	-0.0728	2.5363	1.3782	-0.3383	0.4624	0.2396	0.0987	0.3638	...	-0.0183	0.2778	-0.1105	0.0669	0.1285	-0.1891	0.1336	-0.0211	149.6200	0
1	0.0000	1.1919	0.2662	0.1665	0.4482	0.0600	-0.0824	-0.0788	0.0851	-0.2554	...	-0.2258	-0.6387	0.1013	-0.3398	0.1672	0.1259	-0.0090	0.0147	2.6900	0
2	1.0000	-1.3584	-1.3402	1.7732	0.3798	-0.5032	1.8005	0.7915	0.2477	-1.5147	...	0.2480	0.7717	0.9094	-0.6893	-0.3276	-0.1391	-0.0554	-0.0598	378.6600	0
3	1.0000	-0.9663	-0.1852	1.7930	-0.8633	-0.0103	1.2472	0.2376	0.3774	-1.3870	...	-0.1083	0.0053	-0.1903	-1.1756	0.6474	-0.2219	0.0627	0.0615	123.5000	0
4	2.0000	-1.1582	0.8777	1.5487	0.4030	-0.4072	0.0959	0.5929	-0.2705	0.8177	...	-0.0094	0.7983	-0.1375	0.1413	-0.2060	0.5023	0.2194	0.2152	69.9900	0

5 rows × 31 columns

In [ ]:

```
data_df.info()# 查看数据的基本信息
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
```

```
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [ ]:

```
data_df.describe().T#查看数据基本统计信息
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Time	284807.0000	94813.8596	47488.1460	0.0000	54201.5000	84692.0000	139320.5000	172792.0000
V1	284807.0000	0.0000	1.9587	-56.4075	-0.9204	0.0181	1.3156	2.4549
V2	284807.0000	0.0000	1.6513	-72.7157	-0.5985	0.0655	0.8037	22.0577
V3	284807.0000	-0.0000	1.5163	-48.3256	-0.8904	0.1798	1.0272	9.3826
V4	284807.0000	0.0000	1.4159	-5.6832	-0.8486	-0.0198	0.7433	16.8753
V5	284807.0000	0.0000	1.3802	-113.7433	-0.6916	-0.0543	0.6119	34.8017
V6	284807.0000	0.0000	1.3323	-26.1605	-0.7683	-0.2742	0.3986	73.3016
V7	284807.0000	-0.0000	1.2371	-43.5572	-0.5541	0.0401	0.5704	120.5895
V8	284807.0000	0.0000	1.1944	-73.2167	-0.2086	0.0224	0.3273	20.0072
V9	284807.0000	-0.0000	1.0986	-13.4341	-0.6431	-0.0514	0.5971	15.5950
V10	284807.0000	0.0000	1.0888	-24.5883	-0.5354	-0.0929	0.4539	23.7451
V11	284807.0000	0.0000	1.0207	-4.7975	-0.7625	-0.0328	0.7396	12.0189
V12	284807.0000	-0.0000	0.9992	-18.6837	-0.4056	0.1400	0.6182	7.8484
V13	284807.0000	0.0000	0.9953	-5.7919	-0.6485	-0.0136	0.6625	7.1269
V14	284807.0000	0.0000	0.9586	-19.2143	-0.4256	0.0506	0.4931	10.5268
V15	284807.0000	0.0000	0.9153	-4.4989	-0.5829	0.0481	0.6488	8.8777
V16	284807.0000	0.0000	0.8763	-14.1299	-0.4680	0.0664	0.5233	17.3151
V17	284807.0000	-0.0000	0.8493	-25.1628	-0.4837	-0.0657	0.3997	9.2535
V18	284807.0000	0.0000	0.8382	-9.4987	-0.4988	-0.0036	0.5008	5.0411
V19	284807.0000	0.0000	0.8140	-7.2135	-0.4563	0.0037	0.4589	5.5920
V20	284807.0000	0.0000	0.7709	-54.4977	-0.2117	-0.0625	0.1330	39.4209
V21	284807.0000	0.0000	0.7345	-34.8304	-0.2284	-0.0295	0.1864	27.2028
V22	284807.0000	-0.0000	0.7257	-10.9331	-0.5424	0.0068	0.5286	10.5031
V23	284807.0000	0.0000	0.6245	-44.8077	-0.1618	-0.0112	0.1476	22.5284
V24	284807.0000	0.0000	0.6056	-2.8366	-0.3546	0.0410	0.4395	4.5845
V25	284807.0000	0.0000	0.5213	-10.2954	-0.3171	0.0166	0.3507	7.5196
V26	284807.0000	0.0000	0.4822	-2.6046	-0.3270	-0.0521	0.2410	3.5173
V27	284807.0000	-0.0000	0.4036	-22.5657	-0.0708	0.0013	0.0910	31.6122
V28	284807.0000	-0.0000	0.3301	-15.4301	-0.0530	0.0112	0.0783	33.8478
Amount	284807.0000	88.3496	250.1201	0.0000	5.6000	22.0000	77.1650	25691.1600
Class	284807.0000	0.0017	0.0415	0.0000	0.0000	0.0000	0.0000	1.0000

In [ ]:

```
data_df['Hour'] = data_df['Time'].apply(lambda x:divmod(x,3600)[0])
data_df.sample(5)
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V22	V23	V24	V25	V26	V27	V28	Amount	Class	Hour
141625	84431.0000	-0.9712	0.2178	-0.5835	-0.8459	-3.4833	3.4689	2.9635	-2.7045	0.5741	...	-0.1583	0.1136	-0.5612	1.5789	-0.1497	0.7737	-0.1307	808.2100	0	23.0000
112827	72820.0000	1.2503	-0.1228	-0.0529	-0.7088	-0.4784	-1.1353	0.1602	-0.1927	1.2436	...	0.3372	-0.1166	0.4237	0.8241	-0.5382	0.0497	0.0099	1.0000	0	20.0000
81185	58821.0000	-0.8806	1.3339	0.5060	1.5490	-0.0688	2.6841	-2.0402	-4.8856	-1.3610	...	1.3410	-0.1978	-0.9211	0.9234	0.1549	0.1622	0.1945	122.2100	0	16.0000
2880	2434.0000	-0.5891	0.8261	0.6774	-2.3693	0.5581	-0.5497	0.7659	-1.0134	0.7274	...	0.2382	-0.2933	-0.5113	0.1661	-0.7988	0.2668	-0.0246	1.0000	0	0.0000
97872	66419.0000	1.1894	-0.2476	0.7811	0.9732	-0.5241	0.7414	-0.8110	0.4156	0.9906	...	-0.2257	-0.1400	-0.8904	0.4778	-0.3476	0.0551	0.0124	9.9900	0	18.0000

5 rows × 32 columns

In [ ]:

```
data_df.columns
```

Out[ ]:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class', 'Hour'],
      dtype='object')
```

In [ ]:

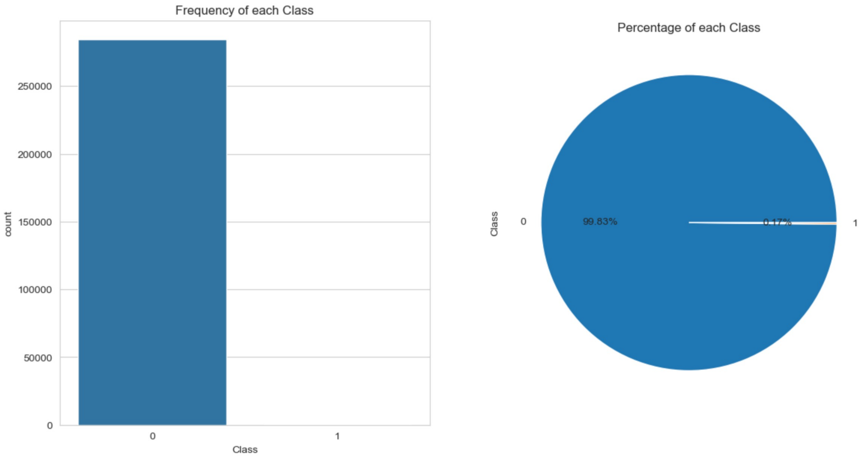
```
x_feature = ['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount','Hour']
# 构建自变量和因变量
X = data_df[x_feature]
y = data_df["Class"]
```

In [ ]:

```
data_df['Class'].value_counts()
```

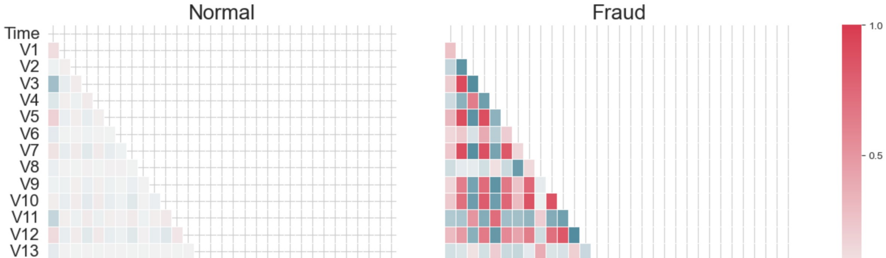
Out[ ]:
```
0    284315
1       492
Name: Class, dtype: int64
```

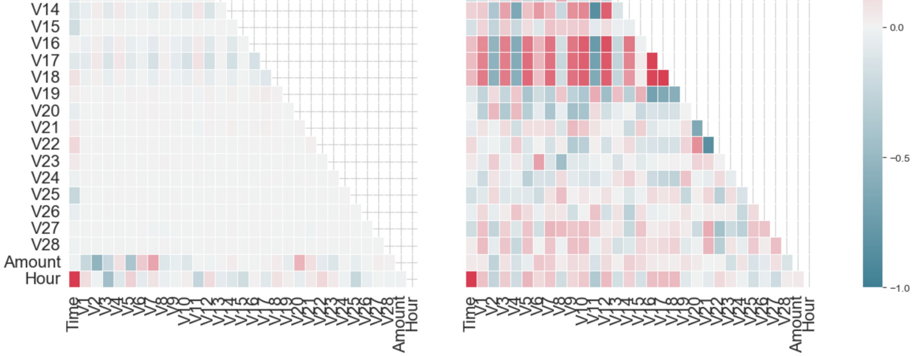
In [ ]:
```
# 目标变量分布可视化
fig, axs = plt.subplots(1,2,figsize=(14,7))
## 柱状图
sns.countplot(x='Class',data=data_df,ax=axs[0])
axs[0].set_title("Frequency of each Class")

## 圆形图
data_df['Class'].value_counts().plot(x=None,y=None, kind='pie', ax=axs[1],autopct='%1.2f%%')
axs[1].set_title("Percentage of each Class")
plt.show()
```



In [ ]:
```
# 获取数据
fraud = data_df[data_df['Class'] == 1]
nonFraud = data_df[data_df['Class'] == 0]

# 相关性计算
correlationNonFraud = nonFraud.loc[:, data_df.columns != 'Class'].corr()
correlationFraud = fraud.loc[:, data_df.columns != 'Class'].corr()

# 上三角矩阵设置
mask = np.zeros_like(correlationNonFraud)# 全部设置0
indices = np.triu_indices_from(correlationNonFraud)#返回函数的上三角矩阵
mask[indices] = True
```

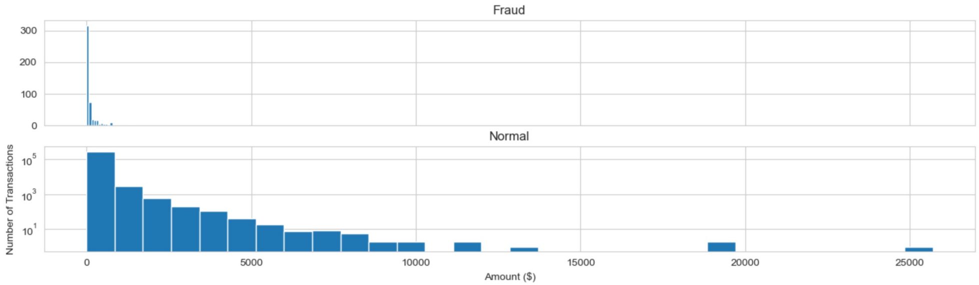
In [ ]:
```
grid_kws = {"width_ratios": (.9, .9, .05), "wspace": 0.2}
f, (ax1, ax2, cbar_ax) = plt.subplots(1, 3, gridspec_kw=grid_kws, figsize = (14, 9))

# 正常用户-特征相关性展示
cmap = sns.diverging_palette(220, 8, as_cmap=True)
ax1 =sns.heatmap(correlationNonFraud, ax = ax1, vmin = -1, vmax = 1, \
    cmap = cmap, square = False, linewidths = 0.5, mask = mask, cbar = False)
ax1.set_xticklabels(ax1.get_xticklabels(), size = 16);
ax1.set_yticklabels(ax1.get_yticklabels(), size = 16);
ax1.set_title('Normal', size = 20)

# 被欺诈的用户-特征相关性展示
ax2 = sns.heatmap(correlationFraud, vmin = -1, vmax = 1, cmap = cmap, \
ax = ax2, square = False, linewidths = 0.5, mask = mask, yticklabels = False, \
    cbar_ax = cbar_ax, cbar_kws={'orientation': 'vertical', \
                                 'ticks': [-1, -0.5, 0, 0.5, 1]})
ax2.set_xticklabels(ax2.get_xticklabels(), size = 16);
ax2.set_title('Fraud', size = 20);
```





In [ ]:

```python
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(16,4))
bins = 30
ax1.hist(data_df["Amount"][data_df["Class"]== 1], bins = bins)
ax1.set_title('Fraud')

ax2.hist(data_df["Amount"][data_df["Class"] == 0], bins = bins)
ax2.set_title('Normal')

plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.yscale('log')
plt.show()
```

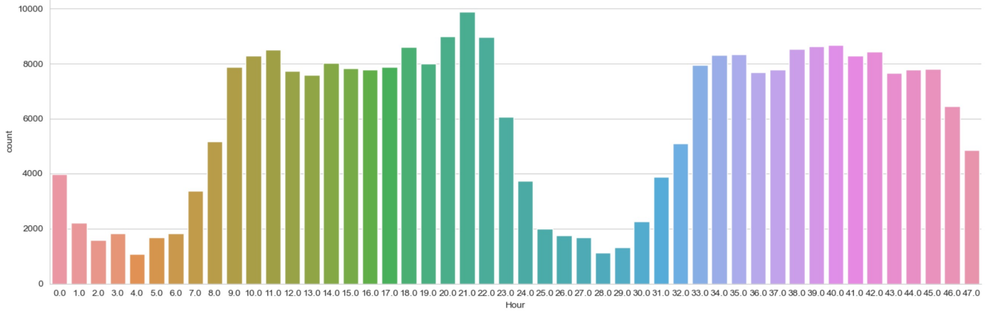


In [ ]:

```python
# 每个小时交易次数
sns.catplot(x="Hour", data=data_df, kind="count", aspect=3)
```

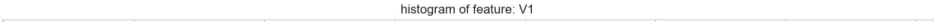
Out[ ]:

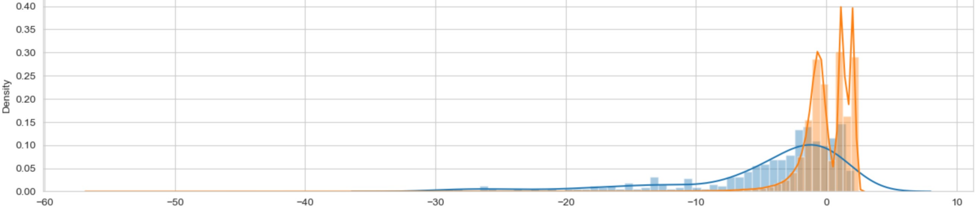
```
<seaborn.axisgrid.FacetGrid at 0x28d28bb67d0>
```



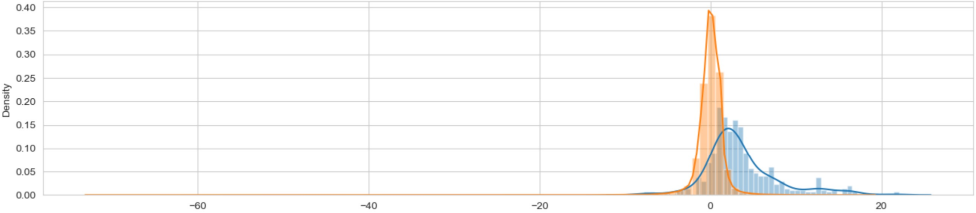
In [ ]:

```python
# 获取V1-V28 字段

v_feat_col = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15',
         'V16', 'V17', 'V18', 'V19', 'V20','V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28']
v_feat_col_size = len(v_feat_col)


plt.figure(figsize=(16,v_feat_col_size*4))
gs = gridspec.GridSpec(v_feat_col_size, 1)
for i, cn in enumerate(data_df[v_feat_col]):
    ax = plt.subplot(gs[i])
    sns.distplot(data_df[cn][data_df["Class"] == 1], bins=50)# V1 异常  绿色表示
    sns.distplot(data_df[cn][data_df["Class"] == 0], bins=100)# V1 正常  橘色表示
    ax.set_xlabel('')
    ax.set_title('histogram of feature: ' + str(cn))
```

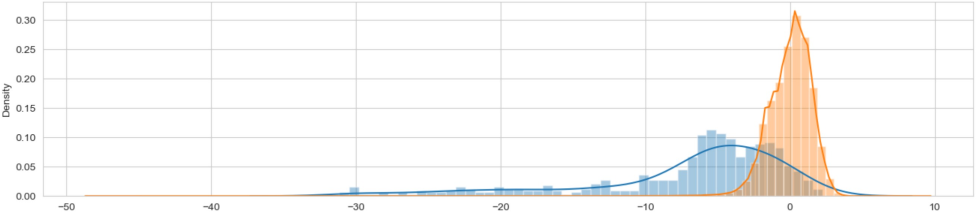




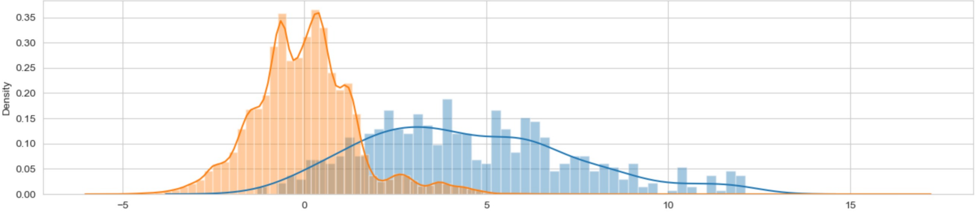
histogram of feature: V2



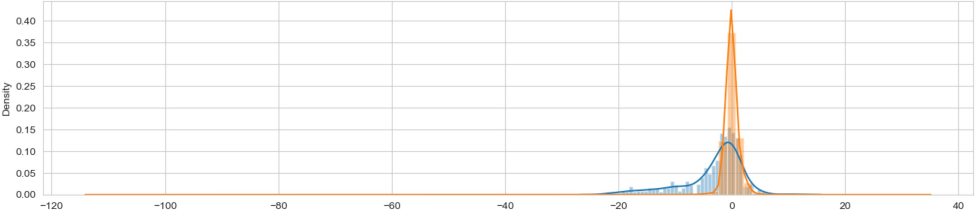
histogram of feature: V3



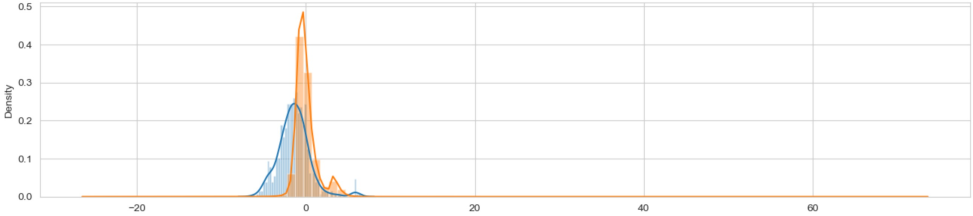
histogram of feature: V4



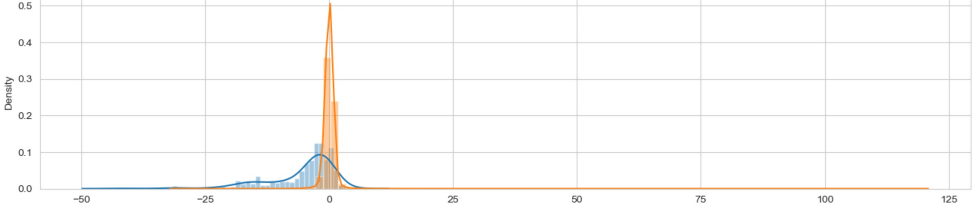
histogram of feature: V5



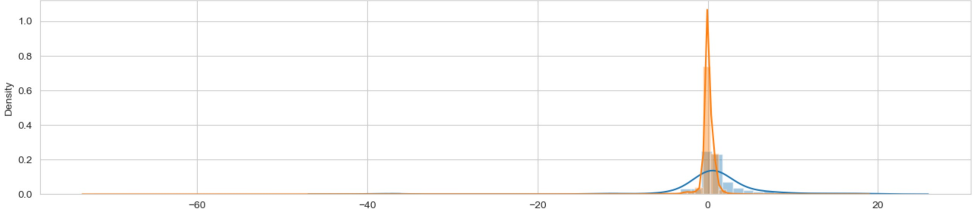
histogram of feature: V6



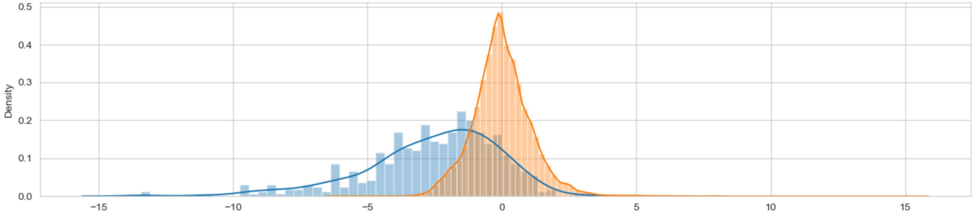
histogram of feature: V7



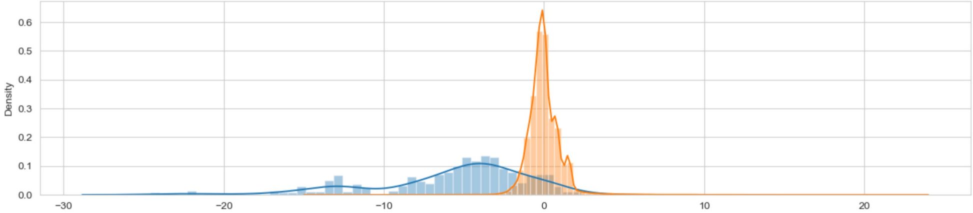
histogram of feature: V8



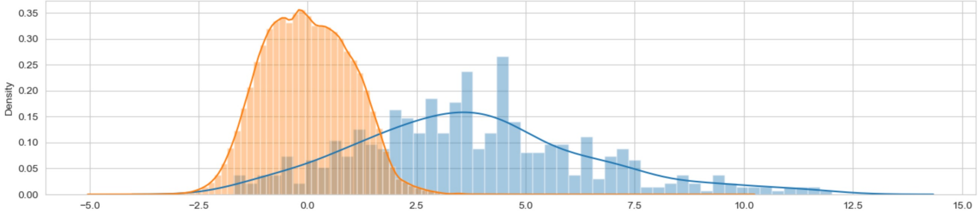
histogram of feature: V9



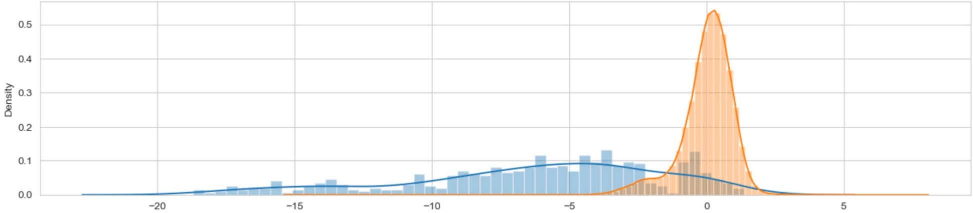
histogram of feature: V10



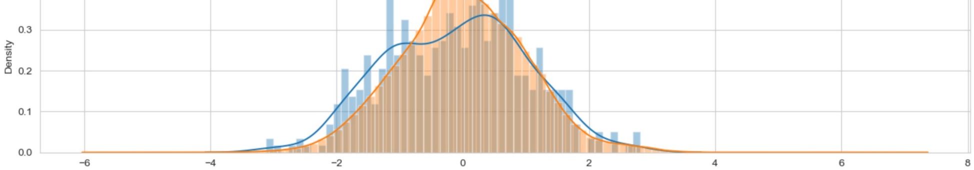
histogram of feature: V11



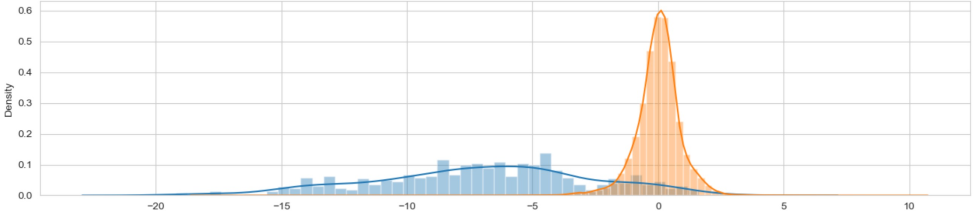
histogram of feature: V12



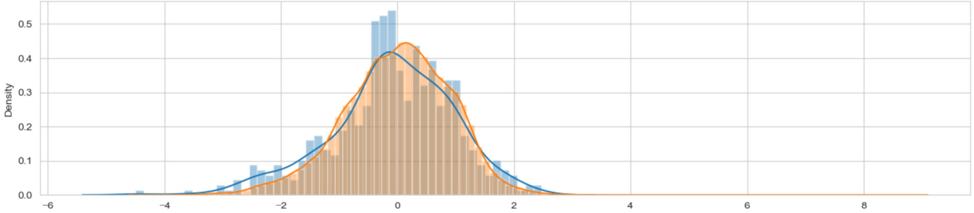
histogram of feature: V13



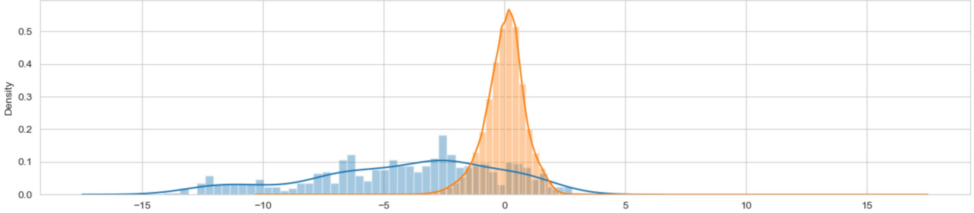
histogram of feature: V14



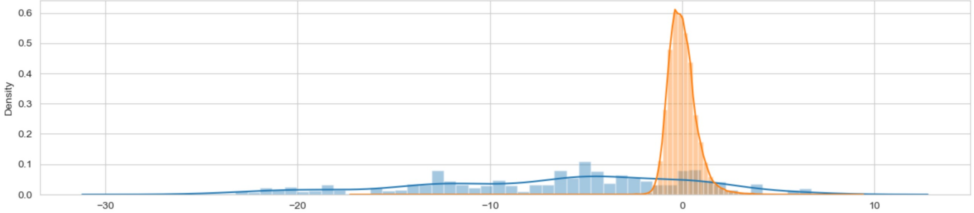
histogram of feature: V15



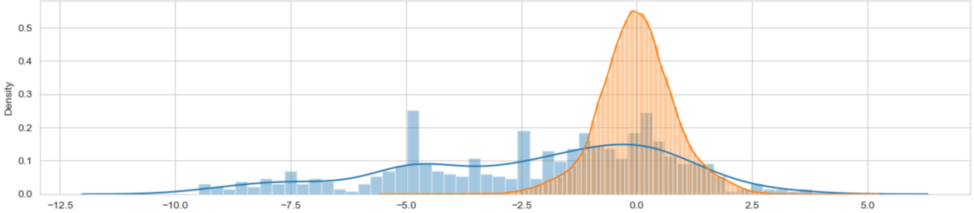
histogram of feature: V16



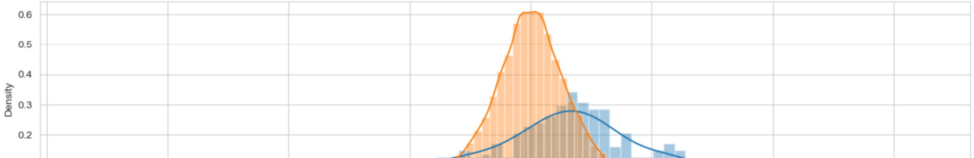
histogram of feature: V17

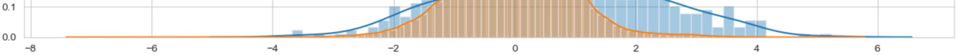


histogram of feature: V18

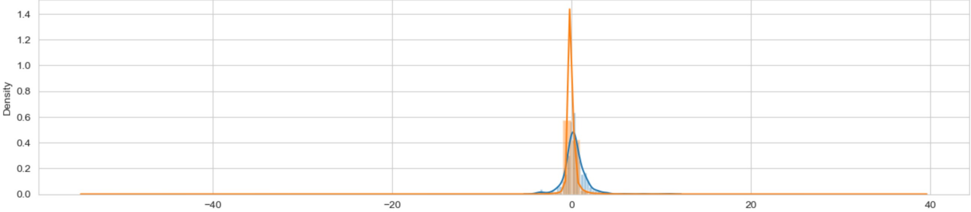


histogram of feature: V19

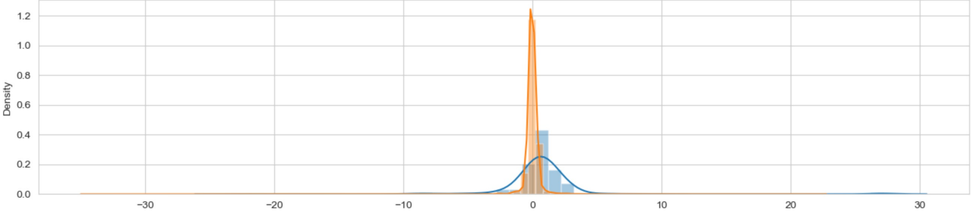




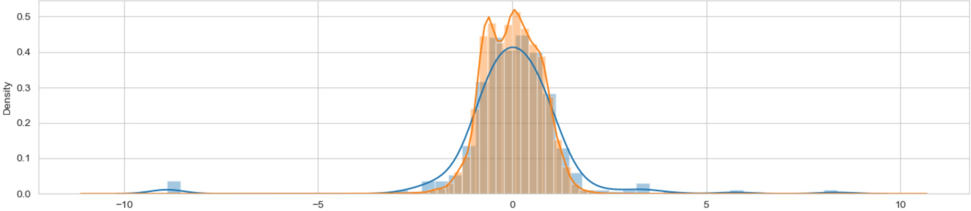
histogram of feature: V20



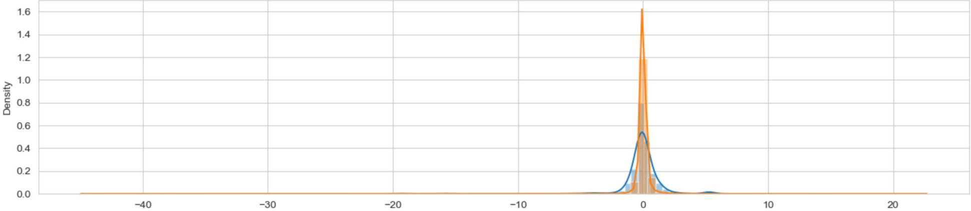
histogram of feature: V21



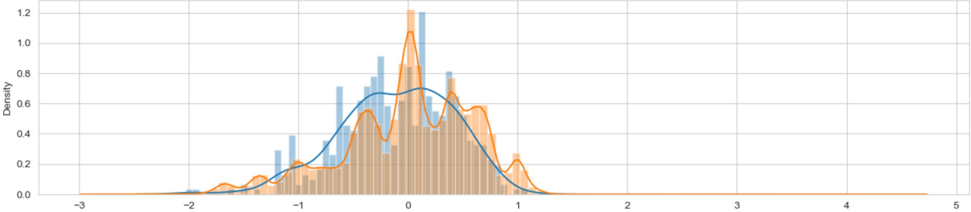
histogram of feature: V22



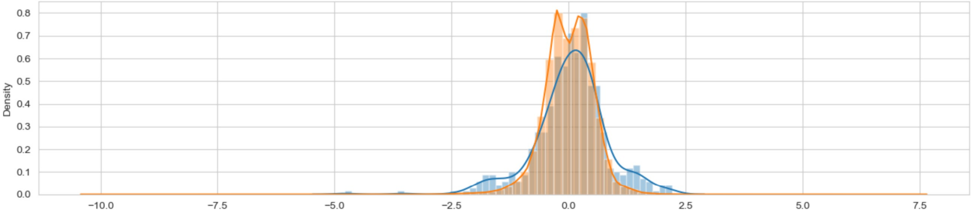
histogram of feature: V23



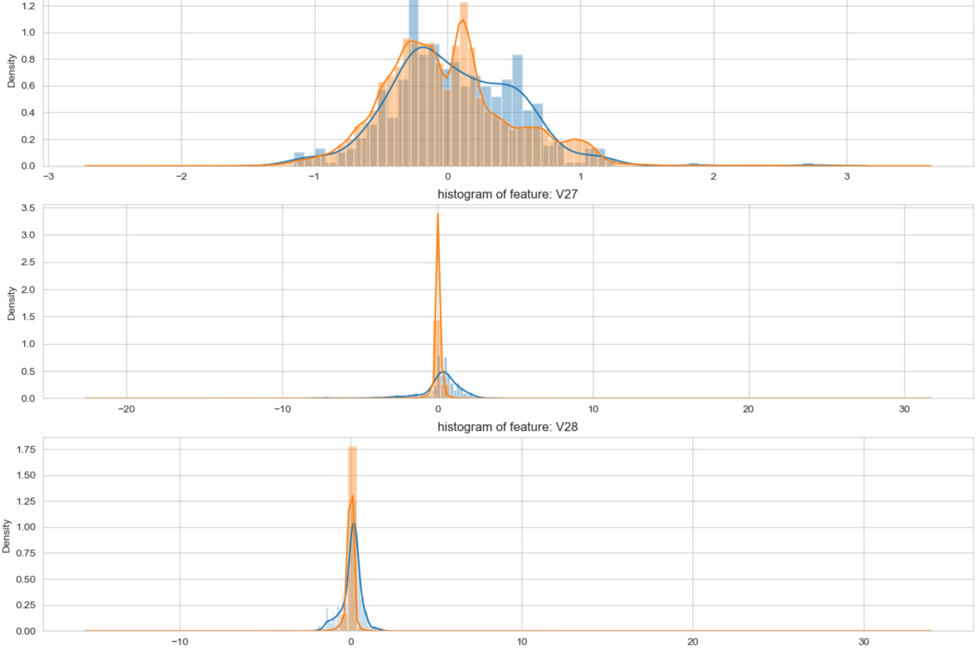
histogram of feature: V24



histogram of feature: V25



histogram of feature: V26



In [ ]:

```
data_df.head()
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V22	V23	V24	V25	V26	V27	V28	Amount	Class	Hour
0	0.0000	-1.3598	-0.0728	2.5363	1.3782	-0.3383	0.4624	0.2396	0.0987	0.3638	...	0.2778	-0.1105	0.0669	0.1285	-0.1891	0.1336	-0.0211	149.6200	0	0.0000
1	0.0000	1.1919	0.2662	0.1665	0.4482	0.0600	-0.0824	-0.0788	0.0851	-0.2554	...	-0.6387	0.1013	-0.3398	0.1672	0.1259	-0.0090	0.0147	2.6900	0	0.0000
2	1.0000	-1.3584	-1.3402	1.7732	0.3798	-0.5032	1.8005	0.7915	0.2477	-1.5147	...	0.7717	0.9094	-0.6893	-0.3276	-0.1391	-0.0554	-0.0598	378.6600	0	0.0000
3	1.0000	-0.9663	-0.1852	1.7930	-0.8633	-0.0103	1.2472	0.2376	0.3774	-1.3870	...	0.0053	-0.1903	-1.1756	0.6474	-0.2219	0.0627	0.0615	123.5000	0	0.0000
4	2.0000	-1.1582	0.8777	1.5487	0.4030	-0.4072	0.0959	0.5929	-0.2705	0.8177	...	0.7983	-0.1375	0.1413	-0.2060	0.5023	0.2194	0.2152	69.9900	0	0.0000

5 rows × 32 columns

In [ ]:

```
# 同时删除Time：保留Hour字段
droplist = ['V8', 'V13', 'V15', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28','Time']
data_df_new = data_df.drop(droplist, axis = 1)
print(data_df_new.shape) #特征从31个缩减至18个（不含目标变量）
data_df_new.tail()
```

```
(284807, 19)
```

Out[ ]:

	V1	V2	V3	V4	V5	V6	V7	V9	V10	V11	V12	V14	V16	V17	V18	V19	Amount	Class	Hour
284802	-11.8811	10.0718	-9.8348	-2.0667	-5.3645	-2.6068	-4.9182	1.9144	4.3562	-1.5931	2.7119	4.6269	1.1076	1.9917	0.5106	-0.6829	0.7700	0	47.0000
284803	-0.7328	-0.0551	2.0350	-0.7386	0.8682	1.0584	0.0243	0.5848	-0.9759	-0.1502	0.9158	-0.6751	-0.7118	-0.0257	-1.2212	-1.5456	24.7900	0	47.0000
284804	1.9196	-0.3013	-3.2496	-0.5578	2.6305	3.0313	-0.2968	0.4325	-0.4848	0.4116	0.0631	-0.5106	0.1407	0.3135	0.3957	-0.5773	67.8800	0	47.0000
284805	-0.2404	0.5305	0.7025	0.6898	-0.3780	0.6237	-0.6862	0.3921	-0.3991	-1.9338	-0.9629	0.4496	-0.6086	0.5099	1.1140	2.8978	10.0000	0	47.0000
284806	-0.5334	-0.1897	0.7033	-0.5063	-0.0125	-0.6496	1.5770	0.4862	-0.9154	-1.0405	-0.0315	-0.0843	-0.3026	-0.6604	0.1674	-0.2561	217.0000	0	47.0000

In [ ]:

```
# 对Amount和Hour 进行特征缩放
col = ['Amount','Hour']
from sklearn.preprocessing import StandardScaler # 导入模块
sc =StandardScaler() # 初始化缩放器 作用：去均值和方差归一化。且是针对每一个特征维度来做的，而不是针对样本
data_df_new[col] =sc.fit_transform(data_df_new[col])#对数据进行标准化
data_df_new.tail()
```

Out[ ]:

	V1	V2	V3	V4	V5	V6	V7	V9	V10	V11	V12	V14	V16	V17	V18	V19	Amount	Class	Hour
284802	-11.8811	10.0718	-9.8348	-2.0667	-5.3645	-2.6068	-4.9182	1.9144	4.3562	-1.5931	2.7119	4.6269	1.1076	1.9917	0.5106	-0.6829	-0.3502	0	1.6044
284803	-0.7328	-0.0551	2.0350	-0.7386	0.8682	1.0584	0.0243	0.5848	-0.9759	-0.1502	0.9158	-0.6751	-0.7118	-0.0257	-1.2212	-1.5456	-0.2541	0	1.6044
284804	1.9196	-0.3013	-3.2496	-0.5578	2.6305	3.0313	-0.2968	0.4325	-0.4848	0.4116	0.0631	-0.5106	0.1407	0.3135	0.3957	-0.5773	-0.0818	0	1.6044
284805	-0.2404	0.5305	0.7025	0.6898	-0.3780	0.6237	-0.6862	0.3921	-0.3991	-1.9338	-0.9629	0.4496	-0.6086	0.5099	1.1140	2.8978	-0.3132	0	1.6044
284806	-0.5334	-0.1897	0.7033	-0.5063	-0.0125	-0.6496	1.5770	0.4862	-0.9154	-1.0405	-0.0315	-0.0843	-0.3026	-0.6604	0.1674	-0.2561	0.5144	0	1.6044

In [ ]:

```
data_df_new.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
V1	284807.0000	0.0000	1.9587	-56.4075	-0.9204	0.0181	1.3156	2.4549
V2	284807.0000	0.0000	1.6513	-72.7157	-0.5985	0.0655	0.8037	22.0577
V3	284807.0000	-0.0000	1.5163	-48.3256	-0.8904	0.1798	1.0272	9.3826
V4	284807.0000	0.0000	1.4159	-5.6832	-0.8486	-0.0198	0.7433	16.8753
V5	284807.0000	0.0000	1.3802	-113.7433	-0.6916	-0.0543	0.6119	34.8017
V6	284807.0000	0.0000	1.3323	-26.1605	-0.7683	-0.2742	0.3986	73.3016
V7	284807.0000	-0.0000	1.2371	-43.5572	-0.5541	0.0401	0.5704	120.5895

V9	284807.0000	-0.0000	1.0986	-13.4341	-0.6431	-0.0514	0.5971	15.5950
V10	284807.0000	0.0000	1.0888	-24.5883	-0.5354	-0.0929	0.4539	23.7451
V11	284807.0000	0.0000	1.0207	-4.7975	-0.7625	-0.0328	0.7396	12.0189
V12	284807.0000	-0.0000	0.9992	-18.6837	-0.4056	0.1400	0.6182	7.8484
V14	284807.0000	0.0000	0.9586	-19.2143	-0.4256	0.0506	0.4931	10.5268
V16	284807.0000	0.0000	0.8763	-14.1299	-0.4680	0.0664	0.5233	17.3151
V17	284807.0000	-0.0000	0.8493	-25.1628	-0.4837	-0.0657	0.3997	9.2535
V18	284807.0000	0.0000	0.8382	-9.4987	-0.4988	-0.0036	0.5008	5.0411
V19	284807.0000	0.0000	0.8140	-7.2135	-0.4563	0.0037	0.4589	5.5920
Amount	284807.0000	0.0000	1.0000	-0.3532	-0.3308	-0.2653	-0.0447	102.3622
Class	284807.0000	0.0017	0.0415	0.0000	0.0000	0.0000	0.0000	1.0000
Hour	284807.0000	-0.0000	1.0000	-1.9603	-0.8226	-0.2158	0.9218	1.6044

In [ ]:

```
x_feature = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V9', 'V10', 'V11', 'V12', 'V14', 'V16', 'V17', 'V18', 'V19', 'Amount',  'Hour']
x_val = data_df_new[x_feature]
y_val = data_df_new['Class']
```

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=10,random_state=123,max_depth=4)#构建分类随机森林分类器
clf.fit(x_val, y_val) #对自变量和因变量进行拟合
```

Out[ ]:

```
▾                 RandomForestClassifier
RandomForestClassifier(max_depth=4, n_estimators=10, random_state=123)
```

In [ ]:

```
RandomForestClassifier(max_depth=4, n_estimators=10, random_state=123)
```

Out[ ]:

```
▾                 RandomForestClassifier
RandomForestClassifier(max_depth=4, n_estimators=10, random_state=123)
```

In [ ]:

```
for feature in zip(x_feature,clf.feature_importances_):
    print(feature)
```

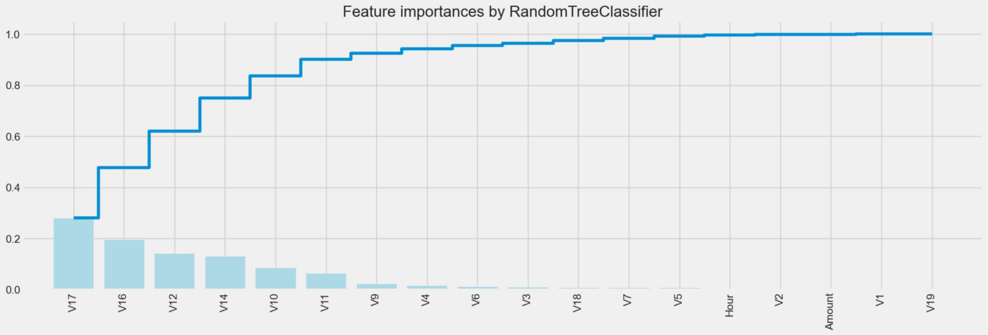
```
('V1', 0.0008826091438778425)
('V2', 0.0021058185061093608)
('V3', 0.009750867340434583)
('V4', 0.01751094043420745)
('V5', 0.008600547467227002)
('V6', 0.013298075656335426)
('V7', 0.00868358970860001)
('V9', 0.023090145788325165)
('V10', 0.0852888657921369)
('V11', 0.06537921978883558)
('V12', 0.14194613523236163)
('V14', 0.13109127164220205)
('V16', 0.19729822871872432)
('V17', 0.27966491161168533)
('V18', 0.009405287105749225)
('V19', 0.0002669771829968763)
('Amount', 0.0017493348363684953)
('Hour', 0.003987153256745854)
```

In [ ]:

```
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (12,6)

## feature importances 可视化##
importances = clf.feature_importances_
feat_names = data_df_new[x_feature].columns
indices = np.argsort(importances)[::-1]
fig = plt.figure(figsize=(20,6))
plt.title("Feature importances by RandomTreeClassifier")

x = list(range(len(indices)))

plt.bar(x, importances[indices], color='lightblue',  align="center")
plt.step(x, np.cumsum(importances[indices]), where='mid', label='Cumulative')
plt.xticks(x, feat_names[indices], rotation='vertical',fontsize=14)
plt.xlim([-1, len(indices)])
```

Out[ ]:

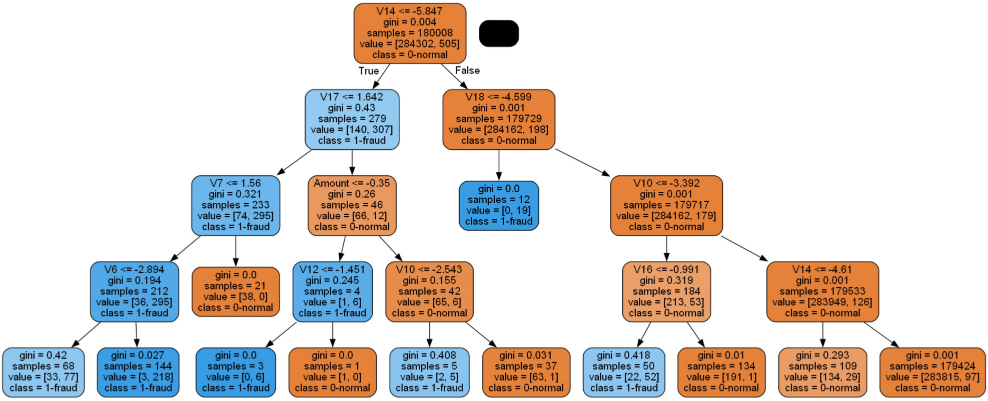
```
(-1.0, 18.0)
```



In [ ]:

```
from sklearn import tree
# 从随机森林抽取单棵树
estimator = clf.estimators_[5]

#  决策数可视化参考：https://blog.csdn.net/shenfuli/article/details/108492095
# 导入可视化工具类
import pydotplus
from IPython.display import display, Image
```

```
# 注意，根据不同系统安装Graphviz2
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin/'

dot_data = tree.export_graphviz(estimator,
                                out_file=None,
                                feature_names=x_feature,
                                class_names = ['0-normal', '1-fraud'],
                                filled = True,
                                rounded =True
                               )
graph = pydotplus.graph_from_dot_data(dot_data)
display(Image(graph.create_png()))
```


[redacted]

降维与聚类

理解t-SNE（需要掌握下面内容）

Euclidean Distance(欧式距离)

Conditional Probability（条件概率）

Normal and T-Distribution Plots （ 正态分布和T分布 ）

结论:

t-SNE算法可以很准确地将数据集中的欺诈和非欺诈案例进行聚类 虽然子样本很小，但t-SNE算法在每个场景中都能非常准确地检测到集群

这表明，进一步的预测模型在区分欺诈案件和非欺诈案件方面将表现得相当好。

In [ ]:

```
# Lets shuffle the data before creating the subsamples
df = data_df_new.sample(frac=1)
# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:492]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)
print(new_df.shape)
new_df.head()
```

```
(984, 19)
```

Out[ ]:

	V1	V2	V3	V4	V5	V6	V7	V9	V10	V11	V12	V14	V16	V17	V18	V19	Amount	Class	Hour
36198	-0.0377	0.5733	0.7040	-0.4375	0.1884	-0.1987	0.4539	-0.6237	-0.4605	1.4913	1.0553	0.6115	0.2327	-0.5630	-0.5189	-1.1367	-0.1745	0	-1.2018
105178	1.1404	1.1342	-1.4295	2.0122	0.6228	-1.1529	0.2212	0.0345	-1.8796	2.1917	-0.4878	-4.4611	1.4109	3.6193	2.2849	-0.4440	-0.3492	1	-0.5192
191297	-2.9932	-0.2240	-0.0562	-0.8877	-0.2369	-0.2840	-0.5344	-0.9847	0.6583	-0.0302	0.0462	-0.1554	-0.0084	1.1834	-1.0066	2.7423	-0.3452	0	0.6943
135718	-0.8853	1.7906	-0.9451	3.8534	-1.5435	0.1886	-2.9884	-2.2945	-1.8862	1.9043	-4.0024	-2.8019	-2.3487	-6.4267	-0.8273	1.7324	-0.3230	1	-0.2917
255556	-0.2422	4.1472	-5.6723	6.4937	1.5912	-1.6025	-0.9505	-4.1285	-3.9632	4.4695	-4.6551	-10.1498	-0.2887	1.4045	2.1321	0.7072	-0.3502	1	1.3011

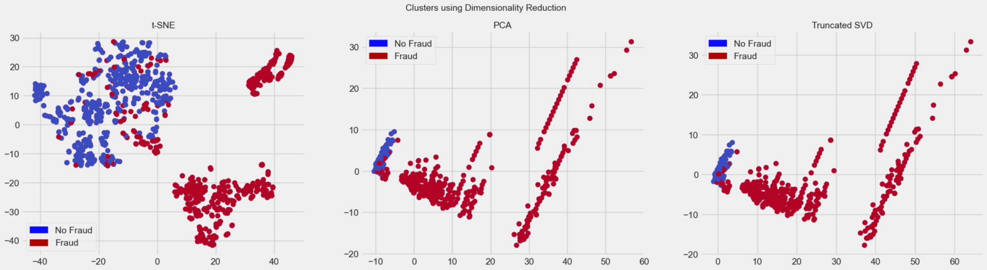
In [ ]:

```
import time
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA,TruncatedSVD

X = new_df.drop('Class', axis=1)
y = new_df['Class']

# T-SNE Implementation
t0 = time.time()
X_reduced_tsne = TSNE(n_components=2, random_state=42).fit_transform(X.values)
t1 = time.time()
print("T-SNE took {:.2} s".format(t1 - t0))

# PCA Implementation
t0 = time.time()
X_reduced_pca = PCA(n_components=2, random_state=42).fit_transform(X.values)
t1 = time.time()
print("PCA took {:.2} s".format(t1 - t0))

# TruncatedSVD
t0 = time.time()
X_reduced_svd = TruncatedSVD(n_components=2, algorithm='randomized', random_state=42).fit_transform(X.values)
t1 = time.time()
print("Truncated SVD took {:.2} s".format(t1 - t0))
```

```
T-SNE took 1.8 s
PCA took 0.005 s
Truncated SVD took 0.005 s
```

In [ ]:

```
import matplotlib.patches as mpatches

f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(24,6))
# labels = ['No Fraud', 'Fraud']
f.suptitle('Clusters using Dimensionality Reduction', fontsize=14)


blue_patch = mpatches.Patch(color='#0A0AFF', label='No Fraud')
red_patch = mpatches.Patch(color='#AF0000', label='Fraud')


# t-SNE scatter plot
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 0), cmap='coolwarm', label='No Fraud', linewidths=2)
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 1), cmap='coolwarm', label='Fraud', linewidths=2)
ax1.set_title('t-SNE', fontsize=14)

ax1.grid(True)

ax1.legend(handles=[blue_patch, red_patch])


# PCA scatter plot
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 0), cmap='coolwarm', label='No Fraud', linewidths=2)
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 1), cmap='coolwarm', label='Fraud', linewidths=2)
ax2.set_title('PCA', fontsize=14)

ax2.grid(True)

ax2.legend(handles=[blue_patch, red_patch])

# TruncatedSVD scatter plot
ax3.scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(y == 0), cmap='coolwarm', label='No Fraud', linewidths=2)
ax3.scatter(X_reduced_svd[:,0], X_reduced_svd[:,1], c=(y == 1), cmap='coolwarm', label='Fraud', linewidths=2)
ax3.set_title('Truncated SVD', fontsize=14)

ax3.grid(True)

ax3.legend(handles=[blue_patch, red_patch])

plt.show()
```



模型训练

样本不平衡解决方法 样本不平衡常用的解决方法：本项目方案（1-欺诈 0-正常）我们需要对1-欺诈数据进行过采样

过采样（oversampling），增加正样本使得正、负样本数目接近，然后再进行学习。 欠采样（undersampling），去除一些负样本使得正、负样本数目接近，然后再进行学习 过采样方法具体操作使用SMOTE（Synthetic Minority Oversampling Technique）

SMOTE的基本原理

SMOTE（Synthetic Minority Oversampling Technique）：合成少数类过采样技术。

具体可以参考： https://www.cnblogs.com/bonelee/p/8535045.html

针对python提供了SMOTE算法库（通过 pip install -U imbalanced-learn 进行算法包安装）

from imblearn.over_sampling import SMOTE # 导入SMOTE算法模块

In [ ]:

```
# 构建自变量和因变量
X = data_df[x_feature]
y = data_df["Class"]

n_sample = y.shape[0]
n_pos_sample = y[y == 1].shape[0]
n_neg_sample = y[y == 0].shape[0]
print('样本个数：{}; 正样本占{:.2%}; 负样本占{:.2%}'.format(n_sample,
                                                   n_pos_sample / n_sample,
                                                   n_neg_sample / n_sample))
print('特征维数：', X.shape[1])
```

```
样本个数：284807; 正样本占0.17%; 负样本占99.83%
特征维数： 18
```

In [ ]:

```
from imblearn.over_sampling import SMOTE # 导入SMOTE算法模块
# 处理不平衡数据
sm = SMOTE(random_state=42)    # 处理过采样的方法
X, y = sm.fit_resample(X, y)
print('通过SMOTE方法平衡正负样本后')
n_sample = y.shape[0]
n_pos_sample = y[y == 1].shape[0]
n_neg_sample = y[y == 0].shape[0]
print('样本个数：{}; 正样本占{:.2%}; 负样本占{:.2%}'.format(n_sample,
                                                   n_pos_sample / n_sample,
                                                   n_neg_sample / n_sample))
print('特征维数：', X.shape[1])
```

```
通过SMOTE方法平衡正负样本后
样本个数：568630; 正样本占50.00%; 负样本占50.00%
特征维数： 18
```

构建训练集和测试集

In [ ]:

```python
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,test_size= 0.3,random_state=42)
```

In [ ]:

```python
len(X_train),len(X_test)
```

Out[ ]:

```
(398041, 170589)
```

模型训练（baseline)

In [ ]:

```python
# 模型训练
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression() # 构建逻辑回归分类器
lr.fit(X_train, y_train)

# 测试集预测
y_pred = lr.predict(X_test)

# 模型评估
from sklearn.metrics import confusion_matrix,classification_report
print('<--------Confusion Matrix-------->\n',confusion_matrix(y_test,y_pred))
print('<--------Classification Report-------->\n',classification_report(y_test,y_pred))
```

```
<--------Confusion Matrix-------->
 [[84022  1273]
 [ 5622 79672]]
<--------Classification Report-------->
               precision    recall  f1-score   support

           0       0.94      0.99      0.96     85295
           1       0.98      0.93      0.96     85294

    accuracy                           0.96    170589
   macro avg       0.96      0.96      0.96    170589
weighted avg       0.96      0.96      0.96    170589
```

In [ ]:

```python
# 构建参数组合
param_grid = {'C': [0.1, 1, 10,100],# 一般经验10倍增加
                            'penalty': [ 'l1', 'l2']}

clf = GridSearchCV(LogisticRegression(),  param_grid, cv=5)
clf.fit(X_train, y_train)
```

Out[ ]:

```
▸         GridSearchCV
▸ estimator: LogisticRegression
      ▸ LogisticRegression
```

In [ ]:

```python
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': [0.1, 1, 10, 100], 'penalty': ['l1', 'l2']})
```

Out[ ]:

```
▸         GridSearchCV
▸ estimator: LogisticRegression
      ▸ LogisticRegression
```

In [ ]:

```python
clf.best_params_
```

Out[ ]:

```
{'C': 100, 'penalty': 'l2'}
```

In [ ]:

```python
# 测试集预测
y_pred = clf.predict(X_test)

# 模型评估
from sklearn.metrics import confusion_matrix,classification_report
print('<--------Confusion Matrix-------->\n',confusion_matrix(y_test,y_pred))
print('<--------Classification Report-------->\n',classification_report(y_test,y_pred))
```

```
<--------Confusion Matrix-------->
 [[83995  1300]
 [ 5842 79452]]
<--------Classification Report-------->
               precision    recall  f1-score   support

           0       0.93      0.98      0.96     85295
           1       0.98      0.93      0.96     85294

    accuracy                           0.96    170589
   macro avg       0.96      0.96      0.96    170589
weighted avg       0.96      0.96      0.96    170589
```

绘制学习曲线

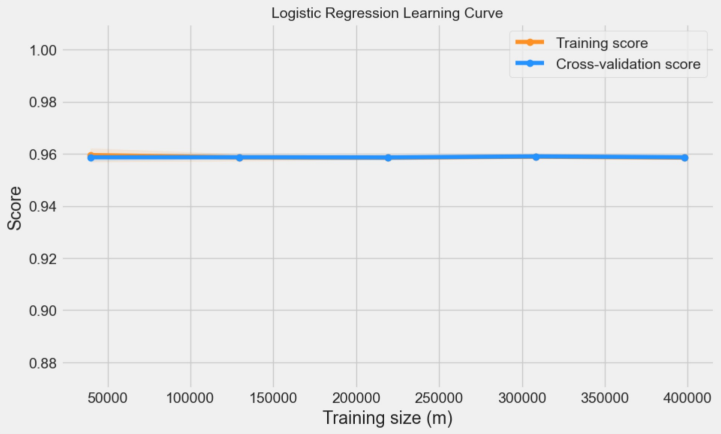
In [ ]:

```python
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import learning_curve

def plot_learning_curve(estimator, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    f, ax1 = plt.subplots(1,1, figsize=(10,6), sharey=True)
    if ylim is not None:
        plt.ylim(*ylim)
    # First Estimator
```

```
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    ax1.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="#ff9124")
    ax1.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="#2492ff")
    ax1.plot(train_sizes, train_scores_mean, 'o-', color="#ff9124",
             label="Training score")
    ax1.plot(train_sizes, test_scores_mean, 'o-', color="#2492ff",
             label="Cross-validation score")
    ax1.set_title("Logistic Regression Learning Curve", fontsize=14)
    ax1.set_xlabel('Training size (m)')
    ax1.set_ylabel('Score')
    ax1.grid(True)
    ax1.legend(loc="best")

    return plt

title = "Learning Curves (lr C:10, penalty: l2})"

estimator = LogisticRegression(penalty='l2', C=10.0)# 提供的最优参数，训练模型查看是否过拟合

cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=42)
plot_learning_curve(estimator,  X, y, (0.87, 1.01), cv=cv, n_jobs=4)
```

Out[ ]:

```
<module 'matplotlib.pyplot' from 'd:\\ProgramData\\Miniconda3\\envs\\rent\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



模型评估(混淆矩阵)

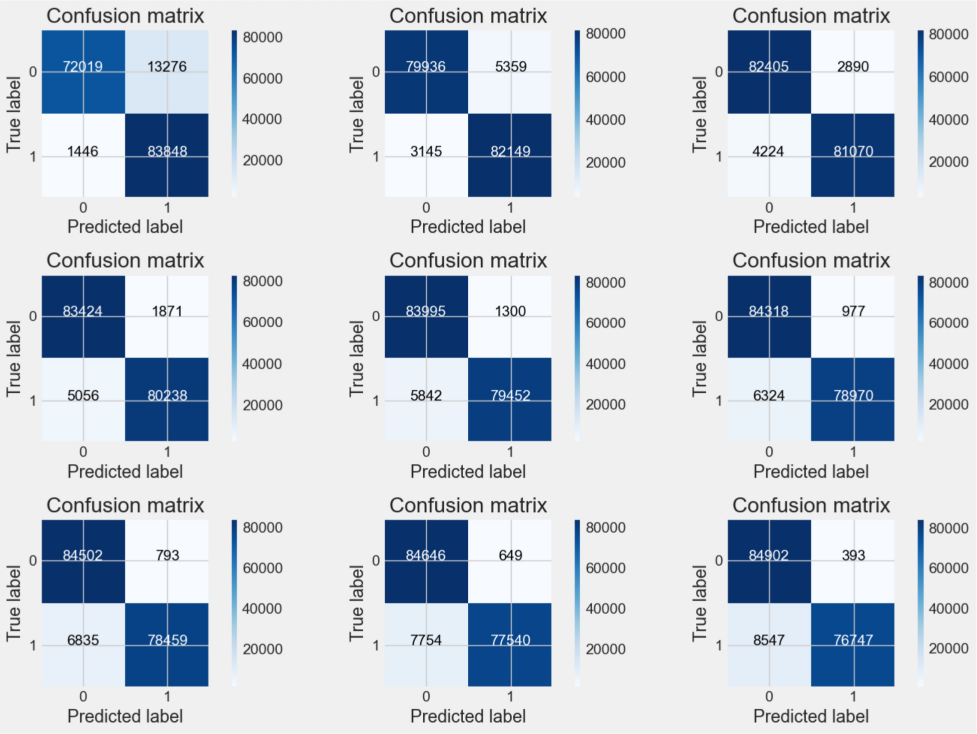
In [ ]:

```
import itertools
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=0)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [ ]:

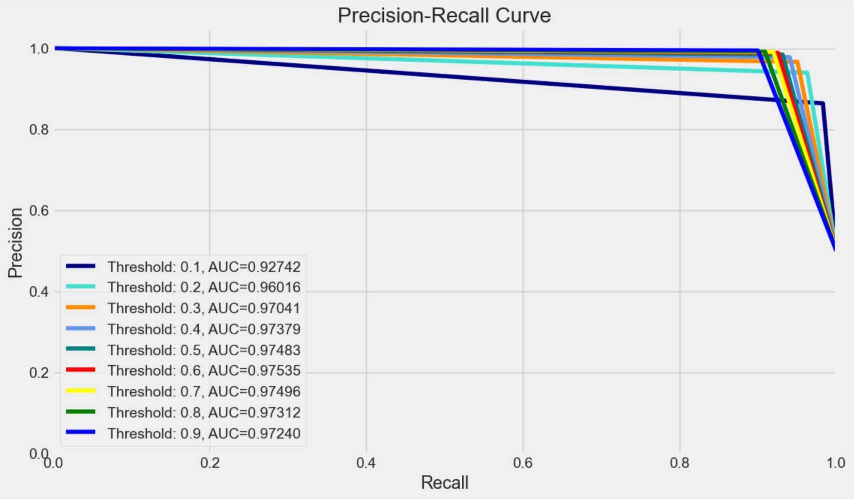
```
from sklearn.metrics import confusion_matrix


y_pred_proba = clf.predict_proba(X_test)  #predict_prob 获得一个概率值
thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]  # 设定不同阈值
plt.figure(figsize=(15,10))

j = 1
for i in thresholds:
    y_test_predictions_high_recall = y_pred_proba[:,1] > i#预测出来的概率值是否大于阈值
    plt.subplot(3,3,j)# 3 * 3 第三行和第三列的图，j表示第几个图表
    j += 1
    cnf_matrix = confusion_matrix(y_test, y_test_predictions_high_recall)
    np.set_printoptions(precision=2)

    x1 = cnf_matrix[1,1]# 正样本中预测也是正样本
    x2 = (cnf_matrix[1,0]+cnf_matrix[1,1])# 所有正样本
    print("threshold:{},Recall metric in the testing dataset {}->{}->{} ".format( i, x1/x2,x1,x2))
    # Plot non-normalized confusion matrix
    class_names = [0,1]
    plot_confusion_matrix(cnf_matrix ,classes=class_names)
```

```
threshold:0.1,Recall metric in the testing dataset 0.9830468731681009->83848->85294
threshold:0.2,Recall metric in the testing dataset 0.9631275353483246->82149->85294
threshold:0.3,Recall metric in the testing dataset 0.9504771730719629->81070->85294
threshold:0.4,Recall metric in the testing dataset 0.9407226768588647->80238->85294
threshold:0.5,Recall metric in the testing dataset 0.9315074917344713->79452->85294
threshold:0.6,Recall metric in the testing dataset 0.9258564494571717->78970->85294
threshold:0.7,Recall metric in the testing dataset 0.9198654067109058->78459->85294
threshold:0.8,Recall metric in the testing dataset 0.9090909090909091->77540->85294
threshold:0.9,Recall metric in the testing dataset 0.8997936548877998->76747->85294
```



In [ ]:

```
from itertools import cycle

thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue', 'teal', 'red', 'yellow', 'green', 'blue','black'])

plt.figure(figsize=(12,7))

j = 1
for i,color in zip(thresholds,colors):
    y_test_predictions_prob = y_pred_proba[:,1] > i #预测出来的概率值是否大于阈值

    precision, recall, thresholds = precision_recall_curve(y_test, y_test_predictions_prob)
    area = auc(recall, precision)# recall ,precision 组成的面积

    # Plot Precision-Recall curve
    plt.plot(recall, precision, color=color,
                 label='Threshold: %s, AUC=%0.5f' %(i , area))
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('Precision-Recall Curve')
    plt.legend(loc="lower left")
```


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Threshold: 0.1, AUC=0.92742
Threshold: 0.2, AUC=0.96016
Threshold: 0.3, AUC=0.97041
Threshold: 0.4, AUC=0.97379
Threshold: 0.5, AUC=0.97483
Threshold: 0.6, AUC=0.97535
Threshold: 0.7, AUC=0.97496
Threshold: 0.8, AUC=0.97312
Threshold: 0.9, AUC=0.97240