In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from sklearn import metrics
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
data = pd.read_csv("TRAIN.csv")
```

In [3]:

```
data.shape
```

Out[3]:

```
(499120, 59)
```

In [4]:

```
data.head()
```

Out[4]:

	id	target	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	ps_ind_05_cat	ps_ind_06_bin	ps_ind_0
0	7	0	2	2	5	1	0	0	
1	9	0	1	1	7	0	0	0	
2	13	0	5	4	9	1	0	0	
3	16	0	0	1	2	0	0	1	
4	17	0	0	2	0	1	0	1	

5 rows × 59 columns

In [5]:

```
#check for null values
np.where(data.isnull())
```

Out[5]:

```
(array([], dtype=int64), array([], dtype=int64))
```

In [6]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499120 entries, 0 to 499119
Data columns (total 59 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   id              499120 non-null  int64  
 1   target          499120 non-null  int64  
 2   ps_ind_01       499120 non-null  int64  
 3   ps_ind_02_cat   499120 non-null  int64  
 4   ps_ind_03       499120 non-null  int64  
 5   ps_ind_04_cat   499120 non-null  int64  
 6   ps_ind_05_cat   499120 non-null  int64  
 7   ps_ind_06_bin   499120 non-null  int64  
 8   ps_ind_07_bin   499120 non-null  int64  
 9   ps_ind_08_bin   499120 non-null  int64  
 10  ps_ind_09_bin   499120 non-null  int64  
 11  ps_ind_10_bin   499120 non-null  int64  
 12  ps_ind_11_bin   499120 non-null  int64  
 13  ps_ind_12_bin   499120 non-null  int64  
 14  ps_ind_13_bin   499120 non-null  int64  
 15  ps_ind_14       499120 non-null  int64  
 16  ps_ind_15       499120 non-null  int64  
 17  ps_ind_16_bin   499120 non-null  int64  
 18  ps_ind_17_bin   499120 non-null  int64  
 19  ps_ind_18_bin   499120 non-null  int64  
 20  ps_reg_01       499120 non-null  float64
 21  ps_reg_02       499120 non-null  float64
 22  ps_reg_03       499120 non-null  float64
 23  ps_car_01_cat   499120 non-null  int64  
 24  ps_car_02_cat   499120 non-null  int64  
 25  ps_car_03_cat   499120 non-null  int64  
 26  ps_car_04_cat   499120 non-null  int64  
 27  ps_car_05_cat   499120 non-null  int64  
 28  ps_car_06_cat   499120 non-null  int64  
 29  ps_car_07_cat   499120 non-null  int64  
 30  ps_car_08_cat   499120 non-null  int64  
 31  ps_car_09_cat   499120 non-null  int64  
 32  ps_car_10_cat   499120 non-null  int64  
 33  ps_car_11_cat   499120 non-null  int64  
 34  ps_car_11       499120 non-null  int64  
 35  ps_car_12       499120 non-null  float64
 36  ps_car_13       499120 non-null  float64
 37  ps_car_14       499120 non-null  float64
 38  ps_car_15       499120 non-null  float64
 39  ps_calc_01      499120 non-null  float64
 40  ps_calc_02      499120 non-null  float64
 41  ps_calc_03      499120 non-null  float64
 42  ps_calc_04      499120 non-null  int64  
 43  ps_calc_05      499120 non-null  int64  
 44  ps_calc_06      499120 non-null  int64  
 45  ps_calc_07      499120 non-null  int64  
 46  ps_calc_08      499120 non-null  int64  
 47  ps_calc_09      499120 non-null  int64  
 48  ps_calc_10      499120 non-null  int64  
 49  ps_calc_11      499120 non-null  int64  
 50  ps_calc_12      499120 non-null  int64  
 51  ps_calc_13      499120 non-null  int64  
 52  ps_calc_14      499120 non-null  int64  
 53  ps_calc_15_bin  499120 non-null  int64  
 54  ps_calc_16_bin  499120 non-null  int64
 55  ps_calc_17_bin  499120 non-null  int64
 56  ps_calc_18_bin  499120 non-null  int64
 57  ps_calc_19_bin  499120 non-null  int64
 58  ps_calc_20_bin  499120 non-null  int64
dtypes: float64(10), int64(49)
memory usage: 224.7 MB
```

In [7]:

```
data.loc[data['target']==0].shape,data.loc[data['target']==1].shape
```

Out[7]:

```
((480895, 59), (18225, 59))
```

In [8]:

```
data.describe()
```

Out[8]:

	id	target	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	ps_ir
count	4.991200e+05	499120.000000	499120.000000	499120.000000	499120.000000	499120.000000	49912
mean	7.438299e+05	0.036514	1.901270	1.359304	4.421079	0.417220	
std	4.292680e+05	0.187566	1.983708	0.664709	2.699942	0.493389	
min	7.000000e+00	0.000000	0.000000	-1.000000	0.000000	-1.000000	-
25%	3.722780e+05	0.000000	0.000000	1.000000	2.000000	0.000000	
50%	7.436865e+05	0.000000	1.000000	1.000000	4.000000	0.000000	
75%	1.115324e+06	0.000000	3.000000	2.000000	6.000000	1.000000	
max	1.488027e+06	1.000000	7.000000	4.000000	11.000000	1.000000	

8 rows × 59 columns

In [9]:

```
y = data['target']
X = data.drop('target',axis = 1)
```

In [10]:

```
#oversampling on minority
smote = SMOTE()
X, y = smote.fit_resample(X, y)
```

In [11]:

```
#train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1,stratify = y)
```

In [12]:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[12]:

```
((865611, 58), (96179, 58), (865611,), (96179,))
```

In [13]:

```
X_train_id = X_train['id']
X_train = X_train.drop("id",axis=1)
```

```
X_test_id = X_test['id']
X_test = X_test.drop("id",axis=1)
```

Logistic Regression

In [14]:

```
parameters = {
    'C': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
}
clf = LogisticRegression()
gscv = GridSearchCV(clf, param_grid = parameters, scoring='roc_auc', cv=10)
```

In [15]:

```
gscv.fit(X_train,y_train)
```

Out[15]:

```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
                               1000, 10000]},
             scoring='roc_auc')
```

In [16]:

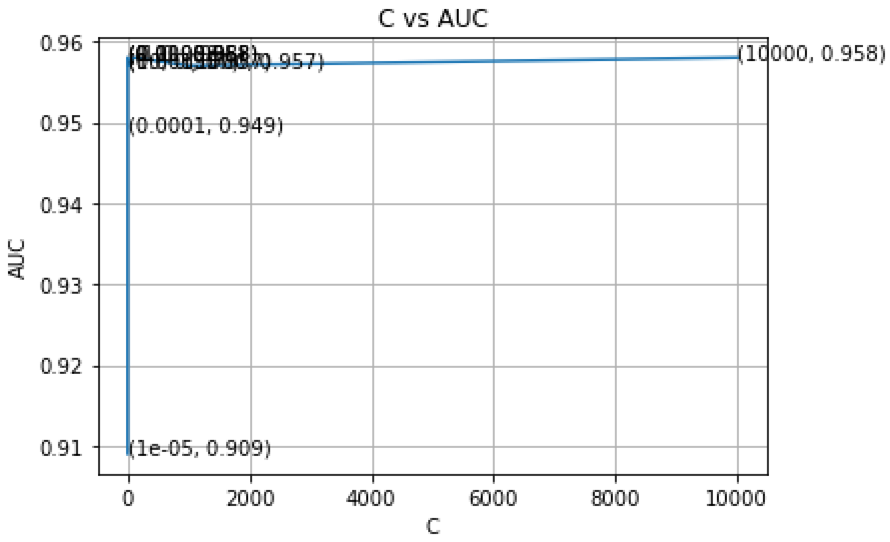
```
result=pd.DataFrame(gscv.cv_results_)
display(result)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	spl
0	9.009678	3.951964	0.056649	0.032646	0.00001	{'C': 1e-05}	0.899379	
1	7.278608	0.343338	0.045086	0.001564	0.0001	{'C': 0.0001}	0.946825	
2	7.542290	0.265399	0.045220	0.002442	0.001	{'C': 0.001}	0.955158	
3	7.769092	0.416858	0.047615	0.006377	0.01	{'C': 0.01}	0.958640	
4	7.796304	0.420956	0.048022	0.004345	0.1	{'C': 0.1}	0.960694	
5	7.960440	0.342552	0.049008	0.006694	1	{'C': 1}	0.958686	
6	7.930875	0.211714	0.048654	0.004804	10	{'C': 10}	0.958591	
7	7.991924	0.366423	0.046891	0.004362	100	{'C': 100}	0.960006	
8	8.023064	0.407536	0.047649	0.003574	1000	{'C': 1000}	0.958624	
9	7.544941	0.101527	0.045450	0.002535	10000	{'C': 10000}	0.958002	

In [17]:

```
C = result['param_C'].values
avgaccscore = np.round(result['mean_test_score'].values,3)
fig, ax = plt.subplots()
sns.lineplot(x= C, y = avgaccscore)
for i, txt in enumerate(avgaccscore):
    ax.annotate((C[i],avgaccscore[i]), (C[i],avgaccscore[i]))
plt.grid()
plt.title("C vs AUC")
plt.xlabel("C")
plt.ylabel("AUC")
plt.show()

print("best hyperparameters :", gscv.best_params_)
print("Best AUC :",gscv.best_score_)
```



```
best hyperparameters : {'C': 1}
Best AUC : 0.9583216348124612
```

RandomForest Classifier

In [18]:

```
parameters ={'max_depth': [1, 5],
             'min_samples_split':[5, 10]}

clf = RandomForestClassifier()
gscv = GridSearchCV(clf, param_grid = parameters, scoring='roc_auc', cv=10)
```

In [19]:

```
gscv.fit(X_train,y_train)
```

Out[19]:

```
GridSearchCV(cv=10, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 5], 'min_samples_split': [5, 10]},
             scoring='roc_auc')
```

In [20]:

```
result=pd.DataFrame(gscv.cv_results_)
display(result)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_
0	17.864940	0.807276	0.314597	0.012424	1	
1	18.386464	0.993228	0.330494	0.028556	1	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_
2	60.943874	0.548633	0.507395	0.008325	5	
3	62.731793	1.914924	0.528651	0.021751	5	

In [21]:

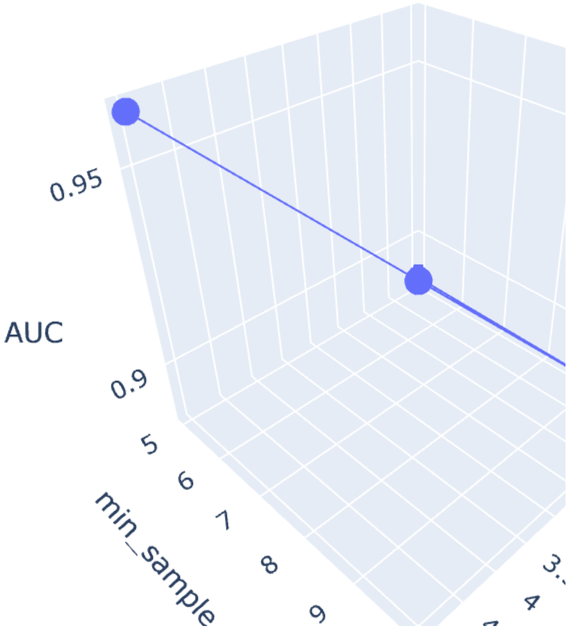
```
md = result['param_max_depth'].values
mss= result['param_min_samples_split'].values
avgaucscore = np.round(result['mean_test_score'].values,3)

trace1 = go.Scatter3d(x=md,y=mss,z=avgaucscore, name = 'Cross validation')

tc = [trace1]
layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth'),
        yaxis = dict(title='min_sample_split'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=tc, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')

print("best hyperparameters :", gscv.best_params_)
print("Best AUC :",gscv.best_score_)
```



```
best hyperparameters : {'max_depth': 5, 'min_samples_split': 10}
Best AUC : 0.9623506480350908
```

Since Random Forest classifier is performing better than Logistic Regression, we can train the final model using best parameter from trained GridSearchCV on Random Forest Classifier.

In [22]:

```
clf = RandomForestClassifier(max_depth = gscv.best_params_['max_depth'], min_samples_sp
clf.fit(X_train,y_train)
```

Out[22]:

```
RandomForestClassifier(max_depth=5, min_samples_split=10)
```

In [23]:

```
#performance on test data
y_test_pred = clf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = metrics.roc_curve(y_test.values, y_test_pred)
auc_score = metrics.auc(fpr, tpr)
print(auc_score)
```

```
0.9587476283890529
```

In [24]:

```
score_data = pd.read_csv("SCORE.csv")
```

In [25]:

```
score_data.head()
```

Out[25]:

	id	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	ps_ind_05_cat	ps_ind_06_bin	ps_ind_07_
0	1249566	0	1	9	0	0	1	
1	1249567	1	1	8	1	0	1	
2	1249568	0	1	7	0	4	0	
3	1249569	3	1	3	1	0	0	
4	1249570	4	1	10	0	0	0	

5 rows × 58 columns

In [26]:

```
score_result = pd.DataFrame()
score_result['id'] = score_data['id']
score_data = score_data.drop('id',axis = 1)
```

In [27]:

```
score_result['Probability'] = clf.predict_proba(score_data)[:,1]
score_result['PRED_Target'] = clf.predict(score_data)
```

In [28]:

```
score_result.to_csv("result.csv")
```

```
In [ ]:
```