Credit card fraud - imbalance dataset with SMOTE

References:

- SMOTE documentation
- Metrics specific to imbalanced learning
- How to Effortlessly Handle Class Imbalance with Python and SMOTE*
- SMOTE using Python
- SMOTE for Imbalanced Classification with Python
- Overcoming Class Imbalance using SMOTE Techniques**
- dataset - credit card fraud

In [36]:
```
import sys
sys.path.append('./')
sys.path.append('../')

import warnings
warnings.filterwarnings('ignore',  category=FutureWarning)
warnings.filterwarnings('ignore',  category=UserWarning)
```

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from fxClassification import *
from fxMachineLearning import *
from fxEDA import *
```

In [3]:
```
df = pd.read_csv('./datasets/credit_dataset.csv')
df.head()
```

Out[3]:

	Unnamed: 0	ID	GENDER	CAR	REALITY	NO_OF_CHILD	INCOME	INCOME_TYPE	EDUCATION_TYPE	FAMILY_TYPE	HOUSE_TY
0	0	5008806	M	Y	Y	0	112500.0	Working	Secondary / secondary special	Married	Hous apartm
1	1	5008808	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	Hous apartm
2	2	5008809	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	Hous apartm
3	3	5008810	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	Hous apartm
4	4	5008811	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	Hous apartm

Data descriptions:

- BEGIN_MONTH - The month of the extracted data is the starting point, backwards, 0 is the current month, -1 is the previous month, and so on
- REALITY - Is there a property

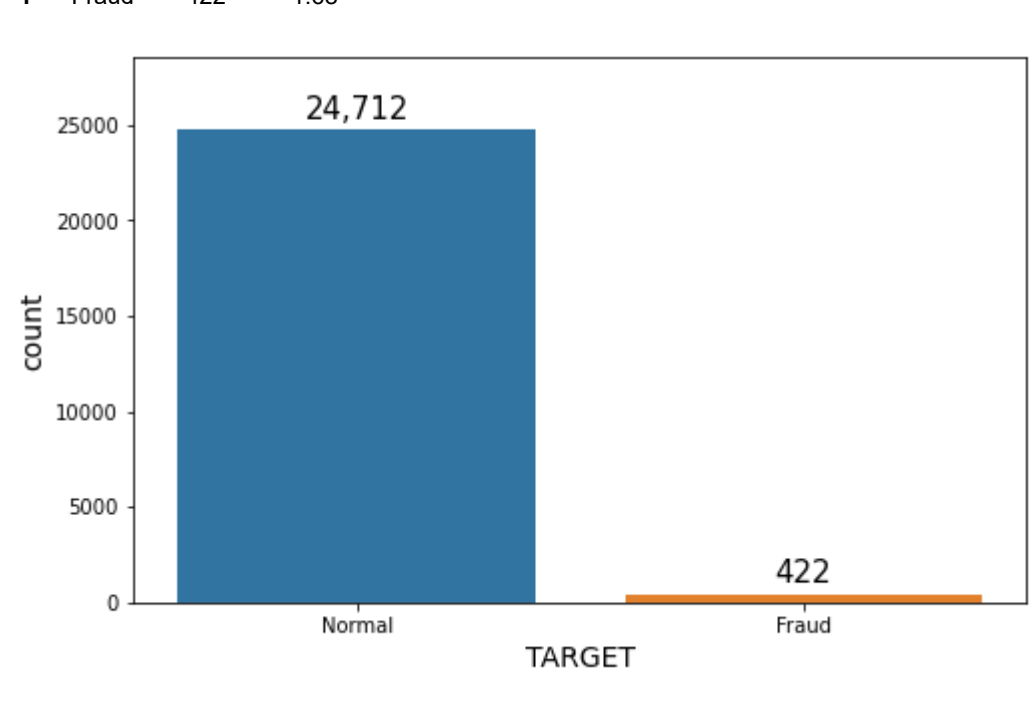
In [4]:
```
df.columns
```

Out[4]:
```
Index(['Unnamed: 0', 'ID', 'GENDER', 'CAR', 'REALITY', 'NO_OF_CHILD', 'INCOME',
       'INCOME_TYPE', 'EDUCATION_TYPE', 'FAMILY_TYPE', 'HOUSE_TYPE',
       'FLAG_MOBIL', 'WORK_PHONE', 'PHONE', 'E_MAIL', 'FAMILY SIZE',
       'BEGIN_MONTH', 'AGE', 'YEARS_EMPLOYED', 'TARGET'],
      dtype='object')
```

In [5]:
```
fxCountPlot(df, 'TARGET', targetlabel={0: 'Normal', 1:'Fraud'}, plotitem='count')
```

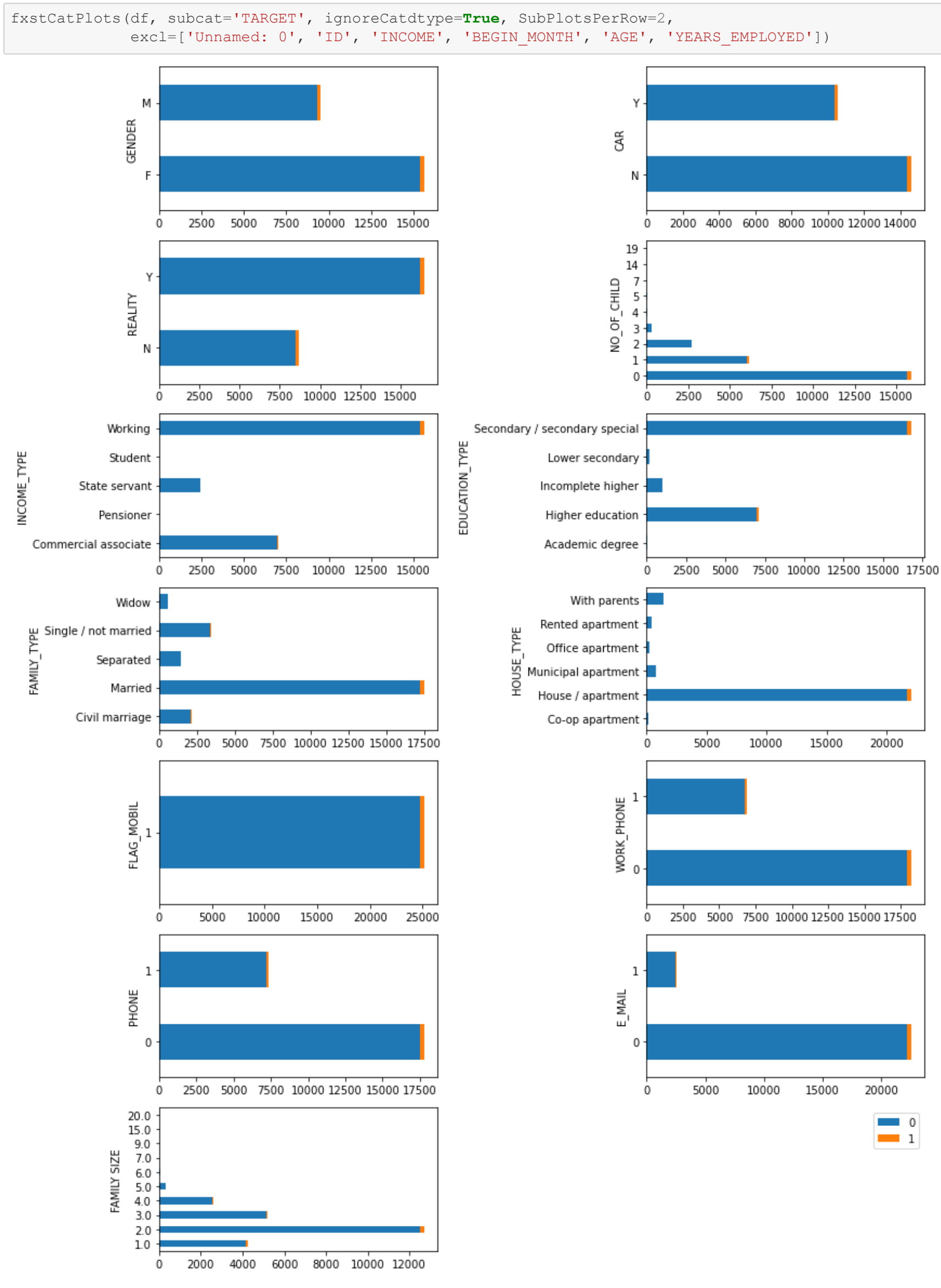
Out[5]:

	target	count	percent
0	Normal	24712	98.32
1	Fraud	422	1.68



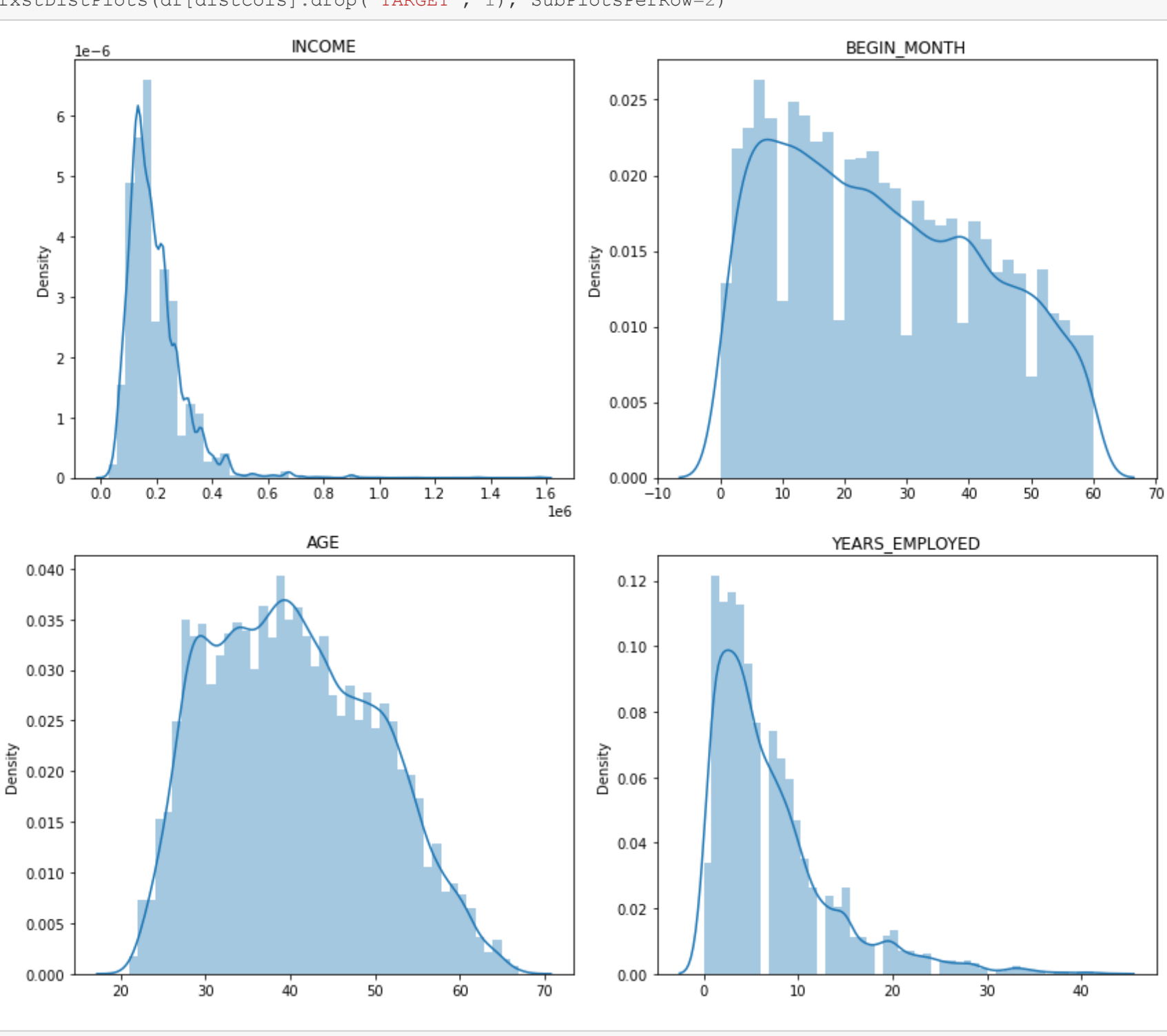
Overly unbalance dataset, most likely would make high-accuracy but low-recall models

In [6]:
```
fxstCatPlots(df, subcat='TARGET', ignoreCatdtype=True, SubPlotsPerRow=2,
             excl=['Unnamed: 0', 'ID', 'INCOME', 'BEGIN_MONTH', 'AGE', 'YEARS_EMPLOYED'])
```

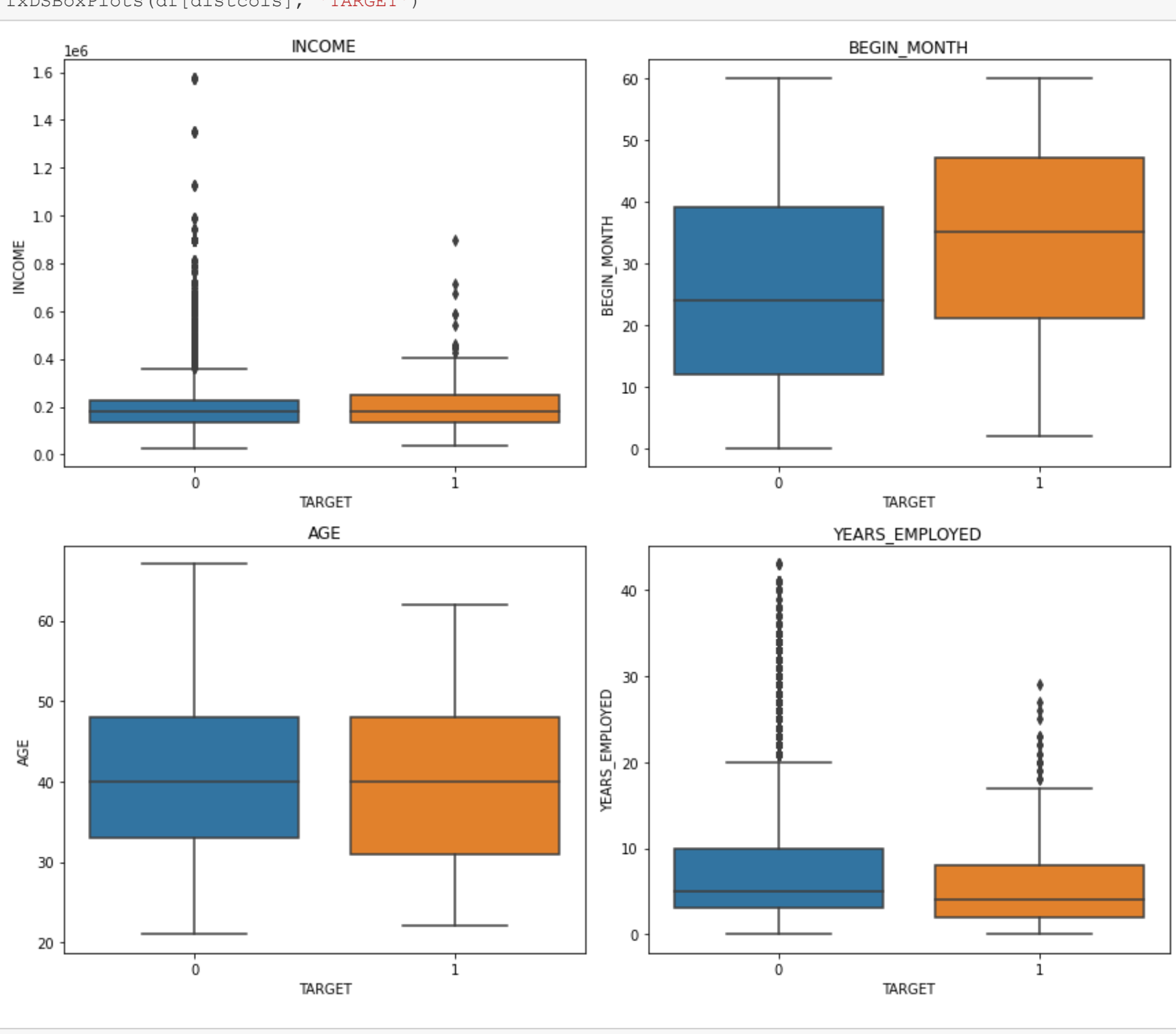


In [7]:
```
distcols = ['INCOME', 'BEGIN_MONTH', 'AGE', 'YEARS_EMPLOYED', 'TARGET']
```

In [8]:
```
fxstDistPlots(df[distcols].drop('TARGET', 1), SubPlotsPerRow=2)
```



In [9]:
```
fxDSBoxPlots(df[distcols], 'TARGET')
```



In [ ]:

Data preperation for machine learning

In [10]:
```
dropitems = ['Unnamed: 0', 'ID', 'FLAG_MOBIL', 'HOUSE_TYPE']
df.drop(dropitems, axis=1, inplace=True)
df.head()
```

Out[10]:

	GENDER	CAR	REALITY	NO_OF_CHILD	INCOME	INCOME_TYPE	EDUCATION_TYPE	FAMILY_TYPE	WORK_PHONE	PHONE	E_MAI
0	M	Y	Y	0	112500.0	Working	Secondary / secondary special	Married	0	0	
1	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	0	1	
2	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	0	1	
3	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	0	1	
4	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	0	1	

In [11]:
```
incometypedict = {'Student': 0,'Pensioner': 1, 'State servant': 2,
                  'Commercial associate': 3, 'Working': 4}
edutypedict = {'Lower secondary': 0, 'Secondary / secondary special': 1,
               'Incomplete higher': 2, 'Higher education': 3, 'Academic degree': 4}
familytypedict = {'Widow': 0, 'Separated': 1, 'Single / not married': 2,
                  'Civil marriage': 3, 'Married': 4}

df.GENDER = df.GENDER.map({'F': 0, 'M': 1})
df.CAR = df.CAR.map({'N': 0, 'Y': 1})
df.REALITY = df.REALITY.map({'N': 0, 'Y': 1})
df.INCOME_TYPE = df.INCOME_TYPE.map(incometypedict)
df.EDUCATION_TYPE = df.EDUCATION_TYPE.map(edutypedict)
df.FAMILY_TYPE = df.FAMILY_TYPE.map(familytypedict)
```

In [12]:
```
df_train, df_test = fxTrainTestSplit(df, 'TARGET')
```

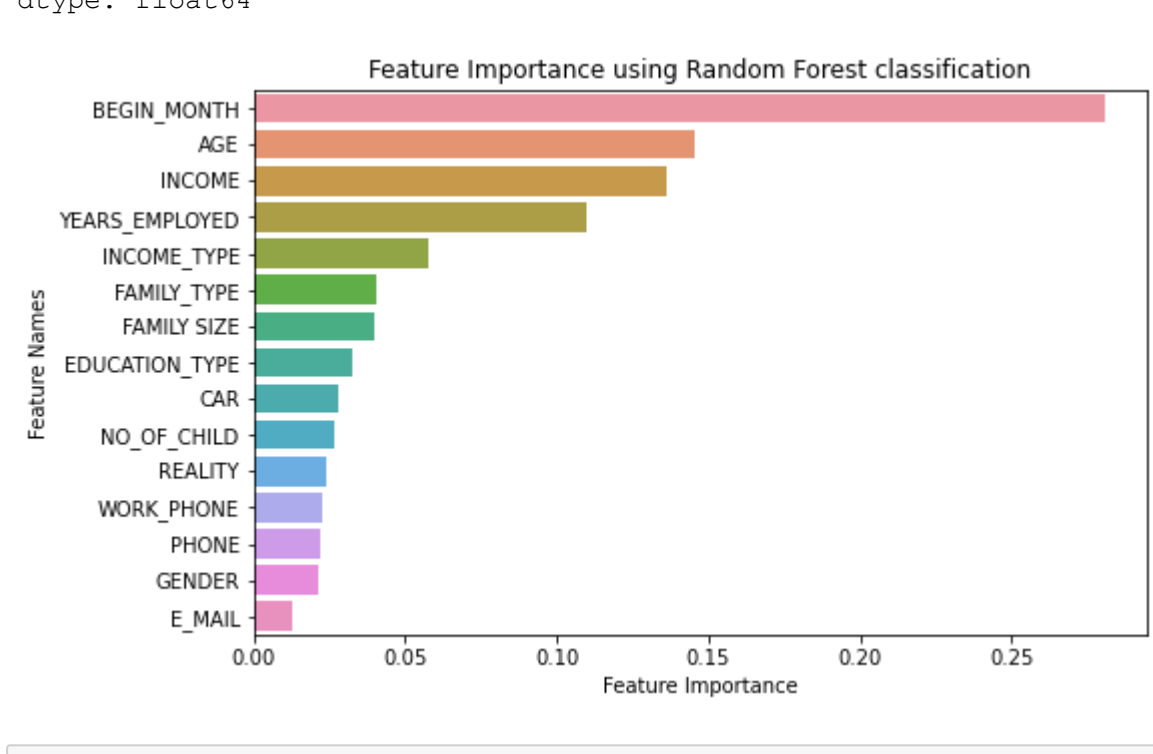
```
Train set size:  20107
Test set size:  5027
Train set target classes:  Counter({0: 19771, 1: 336})
Test set target classes:  Counter({0: 4941, 1: 86})
Return: df_train, df_test
```

Get features importances

In [13]:
```
fxFeatImportances(df, 'TARGET')
```

```
Set the 'classifier='Gradient Boosting, Ada Boosting, Decision Tree, Light GBM for feature importance
s
--- 3.96 seconds ---
```

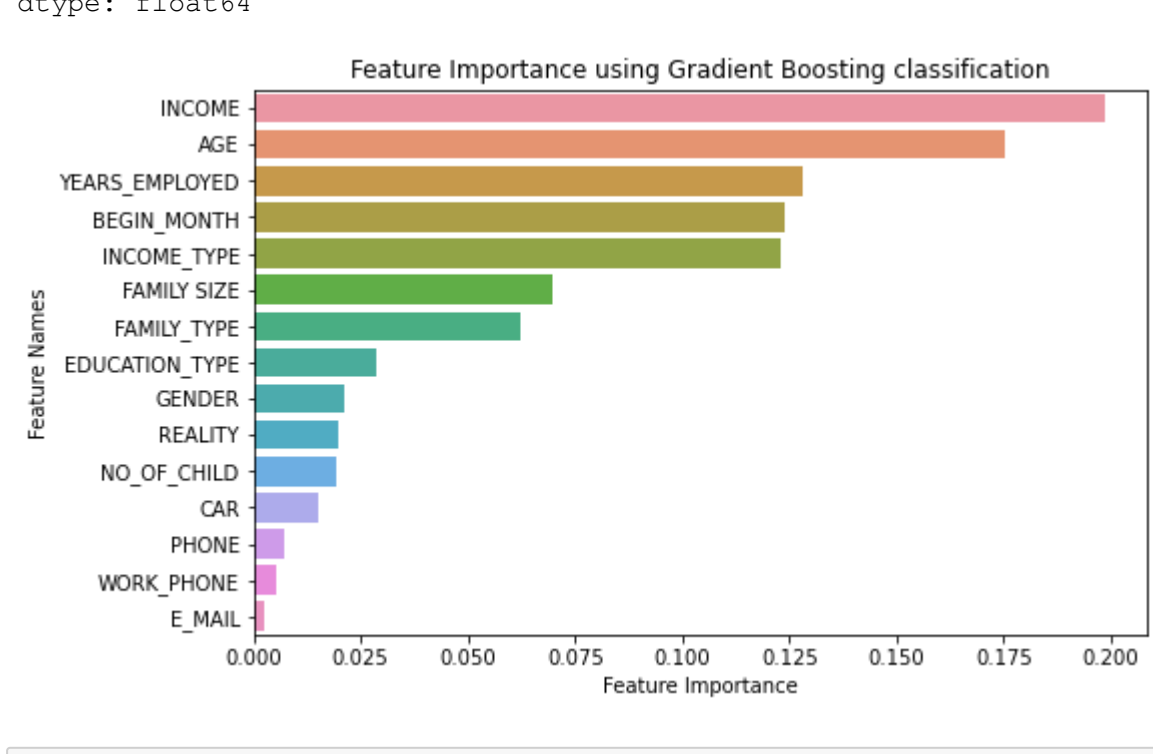
Out[13]:
```
BEGIN_MONTH       0.280783
AGE               0.145362
INCOME            0.136524
YEARS_EMPLOYED    0.109916
INCOME_TYPE       0.057978
FAMILY_TYPE       0.040493
FAMILY SIZE       0.039643
EDUCATION_TYPE    0.032339
CAR               0.027726
NO_OF_CHILD       0.026789
REALITY           0.023866
WORK_PHONE        0.022584
PHONE             0.021680
GENDER            0.021269
E_MAIL            0.013049
dtype: float64
```



In [14]:
```
fxFeatImportances(df, 'TARGET', classifier='Gradient Boosting')
```

```
Set the 'classifier='Random Forest, Ada Boosting, Decision Tree, Light GBM for feature importances
--- 1.84 seconds ---
```

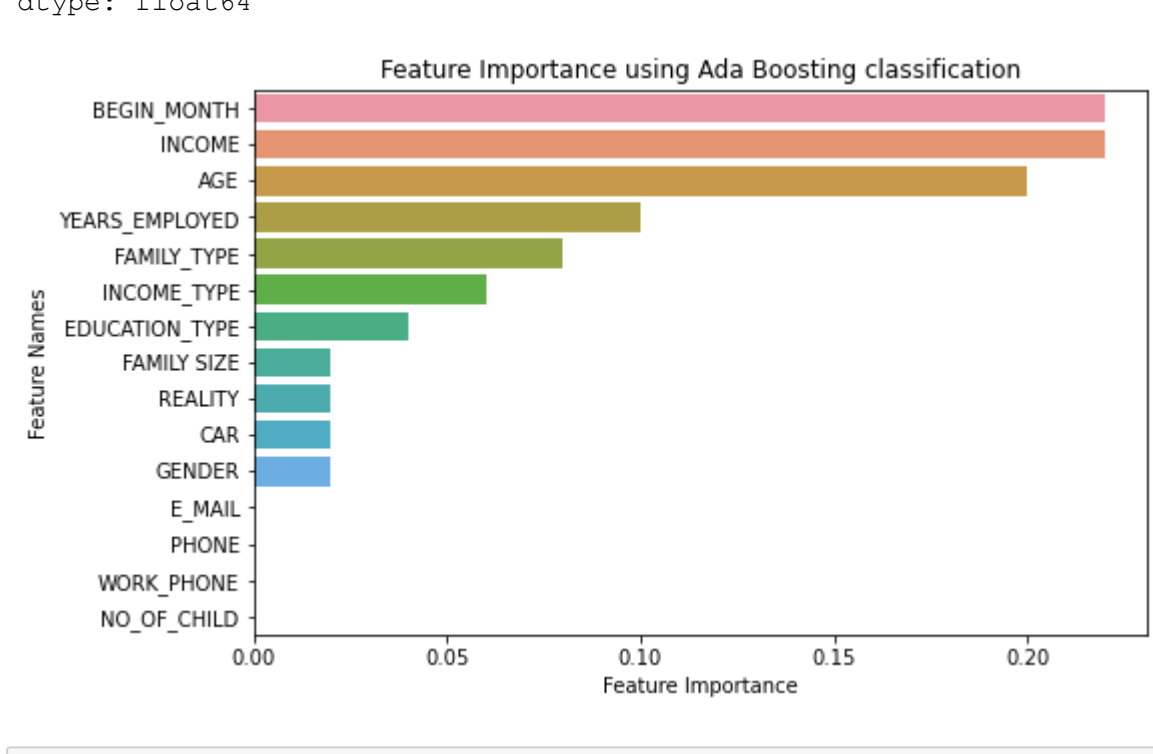
Out[14]:
```
INCOME            0.198577
AGE               0.175313
YEARS_EMPLOYED    0.128227
BEGIN_MONTH       0.123903
INCOME_TYPE       0.123125
FAMILY SIZE       0.069858
FAMILY_TYPE       0.062119
EDUCATION_TYPE    0.028425
GENDER            0.021286
REALITY           0.019820
NO_OF_CHILD       0.019160
CAR               0.015170
PHONE             0.007215
WORK_PHONE        0.005456
E_MAIL            0.002345
dtype: float64
```



In [15]:
```
fxFeatImportances(df, 'TARGET', classifier='Ada Boosting')
```

```
Set the 'classifier='Random Forest, Gradient Boosting, Decision Tree, Light GBM for feature importanc
es
--- 0.76 seconds ---
```

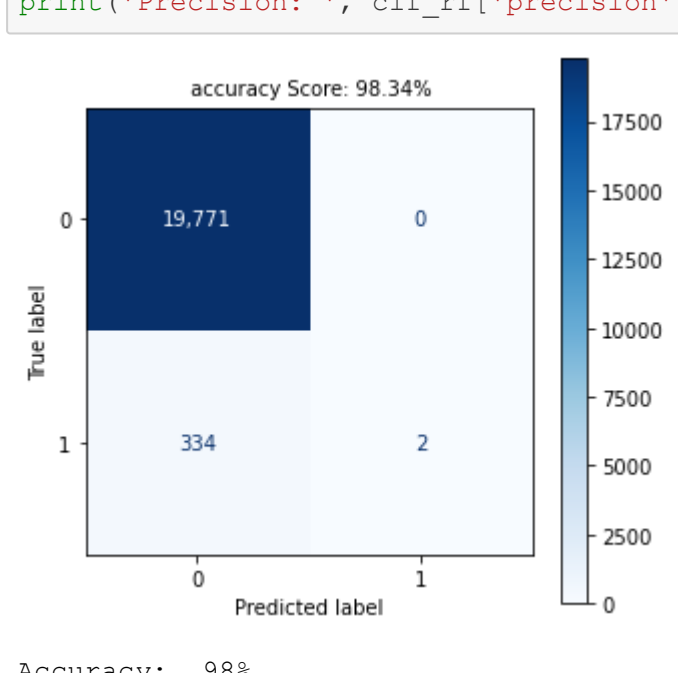
Out[15]:
```
BEGIN_MONTH       0.22
INCOME            0.22
AGE               0.20
YEARS_EMPLOYED    0.10
FAMILY_TYPE       0.08
INCOME_TYPE       0.06
EDUCATION_TYPE    0.04
FAMILY SIZE       0.02
REALITY           0.02
CAR               0.02
GENDER            0.02
E_MAIL            0.00
PHONE             0.00
WORK_PHONE        0.00
NO_OF_CHILD       0.00
dtype: float64
```



In [16]:
```
# drop less important features
df.drop(['CAR', 'EDUCATION_TYPE', 'GENDER', 'NO_OF_CHILD', 'REALITY',
         'E_MAIL', 'PHONE', 'WORK_PHONE'], axis=1, inplace=True)
```

Feed into Machine Learning model (without SMOTE)

In [37]:
```
clf_rf = clfRandomForest(df_train, 'TARGET')
print('Recall:', clf_rf['recall'])
print('Precision: ', clf_rf['precision'])
```

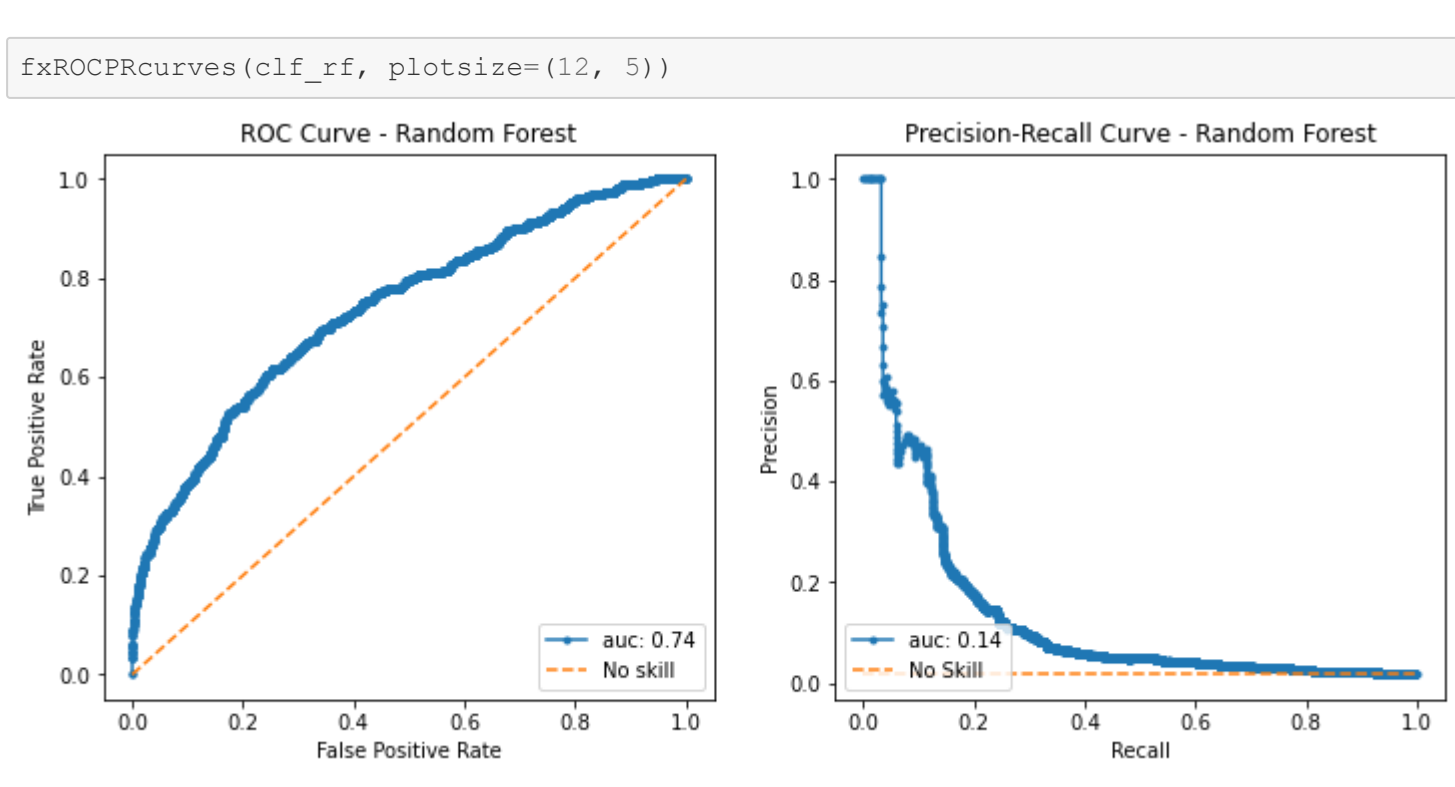



```
Accuracy:  98%
Recall:  0.6%
Precision:  100.0%
F1:  1.18%
dictionary keys:
 ['type', 'prediction', 'accuracy', 'recall', 'precision', 'f1', 'df_proba', 'params', 'importances',
'metrics_report', 'confusionmatrix', 'scale', 'classifier', 'targetlabel', 'y_pred_prob', 'ROC_PR']
Recall: 0.6%
Precision:  100.0%
```

Observations:

- All are being predicted as 0, the machine learning basically does not learn anything

In [18]:
```
fxROCPRcurves(clf_rf, plotsize=(12, 5))
```



Using SMOTE to balance the dataset

Synthetic Minority Oversampling Technique

Over vs. under sampling

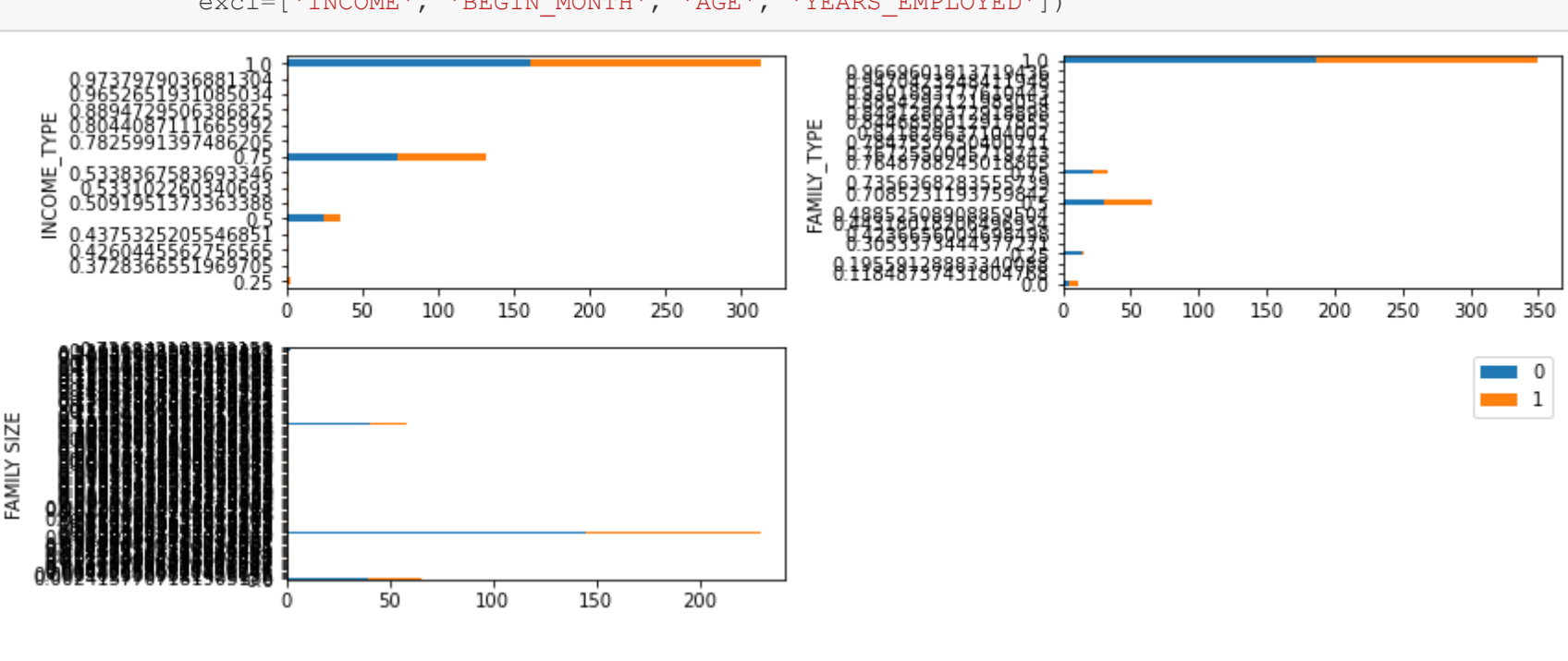
SMOTE

In [19]:
```
dfsm = fxSMOTE(df, 'TARGET', engine='smote')
```

```
Smote engine options: 'smote', 'adasyn', 'smotetomek', 'smoteenn'
Target count before smote: Counter({0: 24712, 1: 422})
Target count after smote: Counter({0: 24712, 1: 24712})
```

In [20]:
```
dfsm_frac = dfsm.sample(frac=0.01) # tamke 1% of the transform data
```

In [21]:
```
fxstCatPlots(dfsm_frac, subcat='TARGET', ignoreCatdtype=True, SubPlotsPerRow=2,
             excl=['INCOME', 'BEGIN_MONTH', 'AGE', 'YEARS_EMPLOYED'])
```

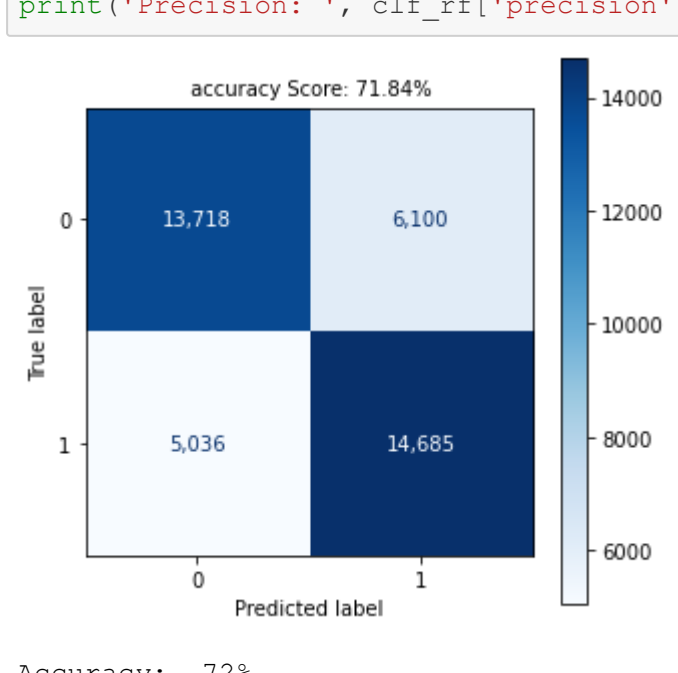


Retrain the model with transformed data

In [22]:
```
dfsm_train, dfsm_test = fxTrainTestSplit(dfsm, 'TARGET')
```

```
Train set size:  39539
Test set size:  9885
Train set target classes:  Counter({0: 19818, 1: 19721})
Test set target classes:  Counter({1: 4991, 0: 4894})
Return: df_train, df_test
```

In [23]:
```
clf_rf = clfRandomForest(dfsm_train, 'TARGET')
print('Recall:', clf_rf['recall'])
print('Precision: ', clf_rf['precision'])
```




```
Accuracy:  72%
Recall:  74.46%
Precision:  70.65%
F1:  72.51%
dictionary keys:
 ['type', 'prediction', 'accuracy', 'recall', 'precision', 'f1', 'df_proba', 'params', 'importances',
'metrics_report', 'confusionmatrix', 'scale', 'classifier', 'targetlabel', 'y_pred_prob', 'ROC_PR']
Recall: 74.46%
Precision:  70.65%
```

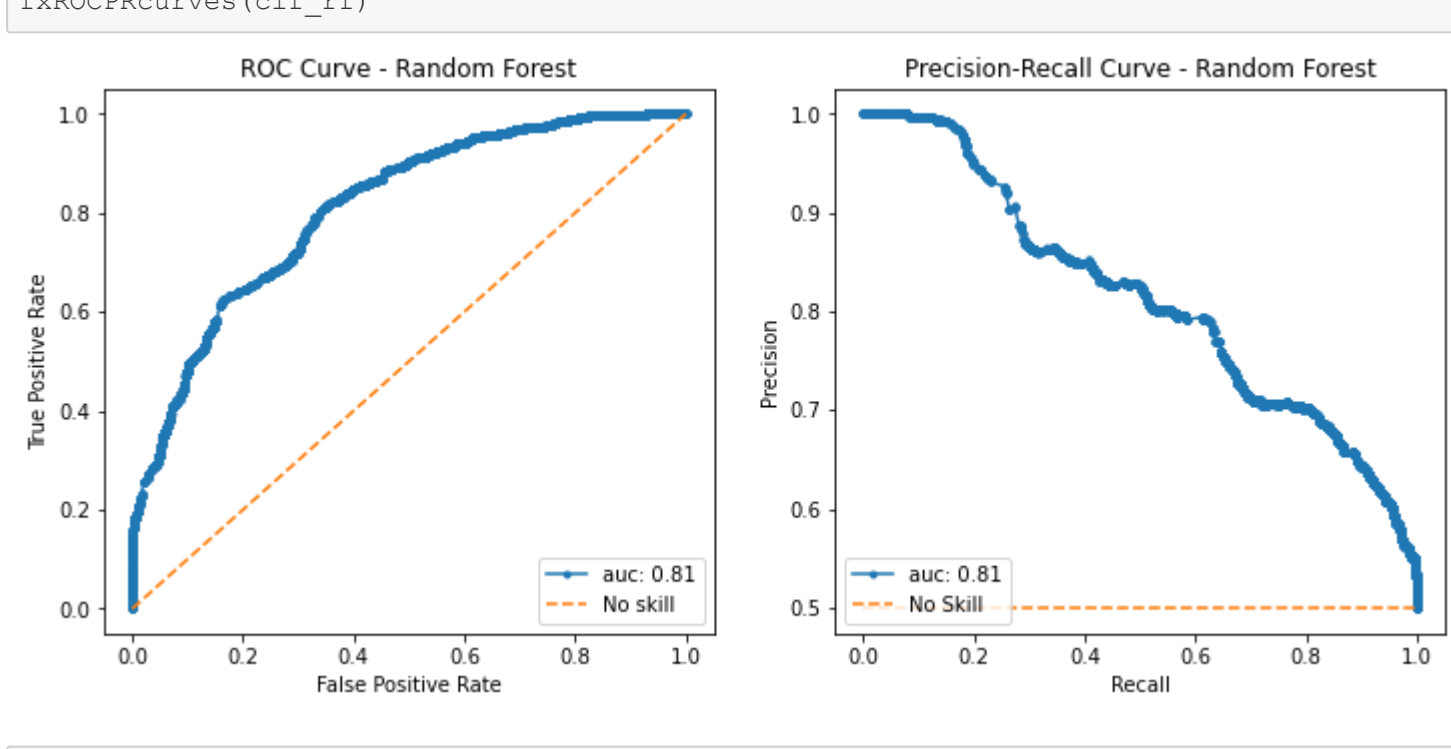
In [24]:
```
print(clf_rf['metrics_report'])
```

```
              precision    recall  f1-score   support

           0       0.73      0.69      0.71     19818
           1       0.71      0.74      0.73     19721

    accuracy                           0.72     39539
   macro avg       0.72      0.72      0.72     39539
weighted avg       0.72      0.72      0.72     39539
```

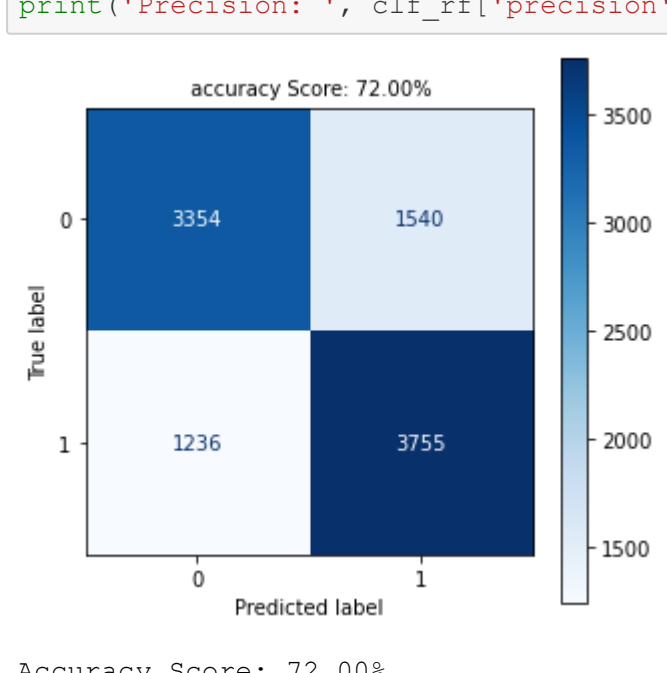
In [25]:
```
fxROCPRcurves(clf_rf)
```



In [26]:
```
# fxLearningCurve(dfsm_train, 'TARGET', clf_rf['classifier']) # take long time
```

Perform on test data

In [27]:
```
perf_smrf_test = clfTestdsPerformance(dfsm_test, 'TARGET', clf_rf['classifier'])
print('Recall:', clf_rf['recall'])
print('Precision: ', clf_rf['precision'])
```




```
Accuracy Score: 72.00%
Recall Score: 75.24%
Precision Score: 70.92%
F1 Score: 73.01%

              precision    recall  f1-score   support

           0       0.73      0.69      0.71      4894
           1       0.71      0.75      0.73      4991

    accuracy                           0.72      9885
   macro avg       0.72      0.72      0.72      9885
weighted avg       0.72      0.72      0.72      9885

dictionary keys:
 ['report', 'accuracy', 'recall', 'precision', 'f1', 'confusionmatrix', 'prediction']
Recall: 74.46%
Precision:  70.65%
```

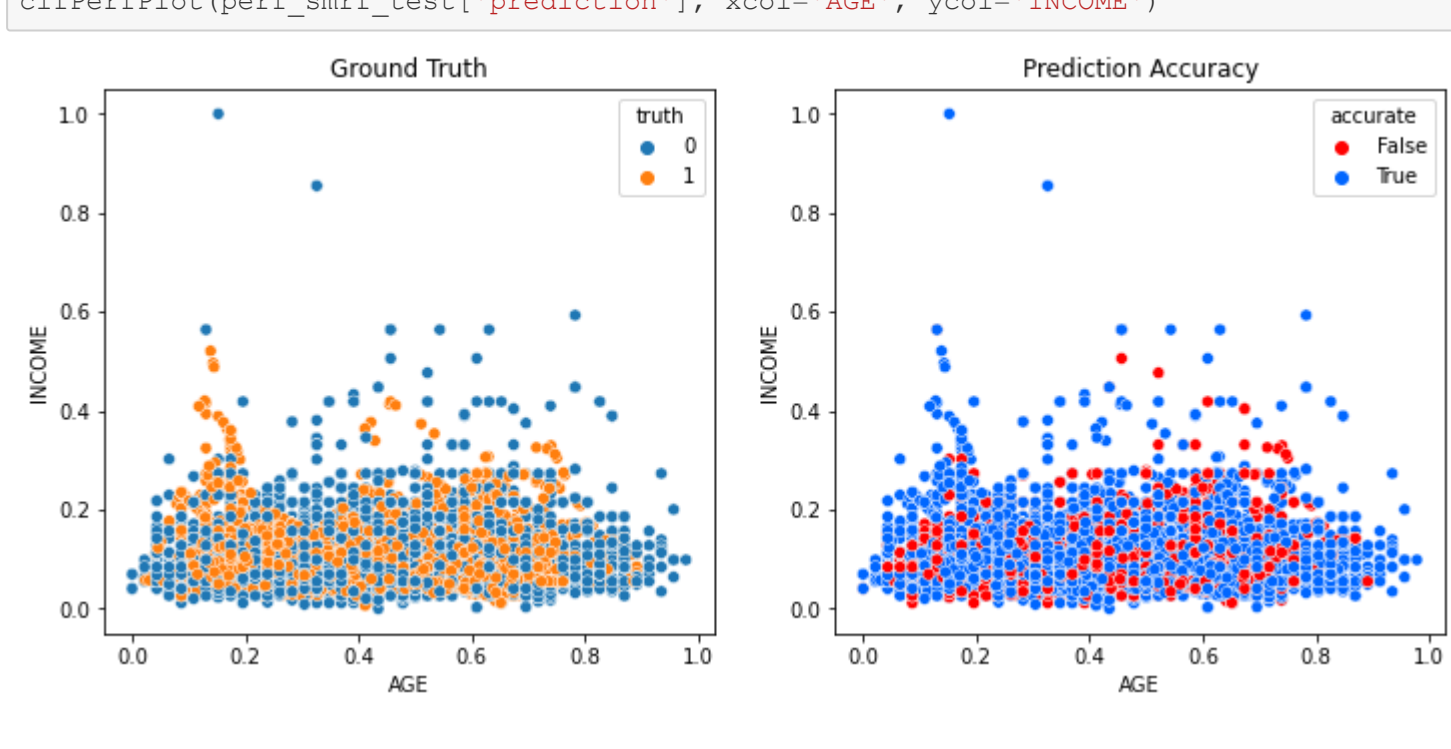
In [28]:
```
perf_smrf_test['prediction']
```

Out[28]:

	INCOME	INCOME_TYPE	FAMILY_TYPE	FAMILY SIZE	BEGIN_MONTH	AGE	YEARS_EMPLOYED	truth	prediction	accurate	likel
31542	0.142442	0.750000	1.0	0.052632	0.757860	0.456522	0.046512	1	1	True	0.6
25501	0.049771	1.000000	0.5	0.040419	0.814271	0.640523	0.071820	1	1	True	0.6
1536	0.063953	1.000000	1.0	0.157895	0.200000	0.282609	0.302326	0	0	True	0.3
13373	0.098837	1.000000	0.0	0.000000	0.200000	0.717391	0.116279	0	0	True	0.4
19409	0.098837	1.000000	0.5	0.000000	0.516667	0.217391	0.232558	0	0	True	0.4
...	...	...	...	...	...	...	...	...	...	...	...
32887	0.185510	0.677990	1.0	0.105263	0.209601	0.586957	0.126138	1	0	False	0.4
41419	0.123155	1.000000	1.0	0.052632	0.695194	0.601589	0.201700	1	1	True	0.5
22922	0.055233	1.000000	1.0	0.105263	0.100000	0.500000	0.069767	0	0	True	0.2
21167	0.215116	1.000000	1.0	0.052632	0.050000	0.652174	0.348837	0	0	True	0.2
26915	0.074116	0.425199	1.0	0.089516	0.474934	0.145496	0.027833	1	1	True	0.7

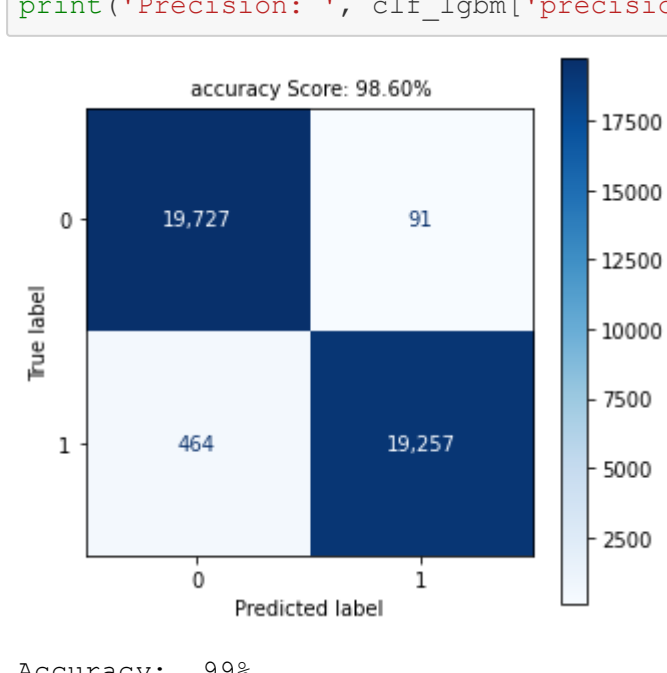
9885 rows × 11 columns

In [29]:
```
clfPerfPlot(perf_smrf_test['prediction'], xcol='AGE', ycol='INCOME')
```



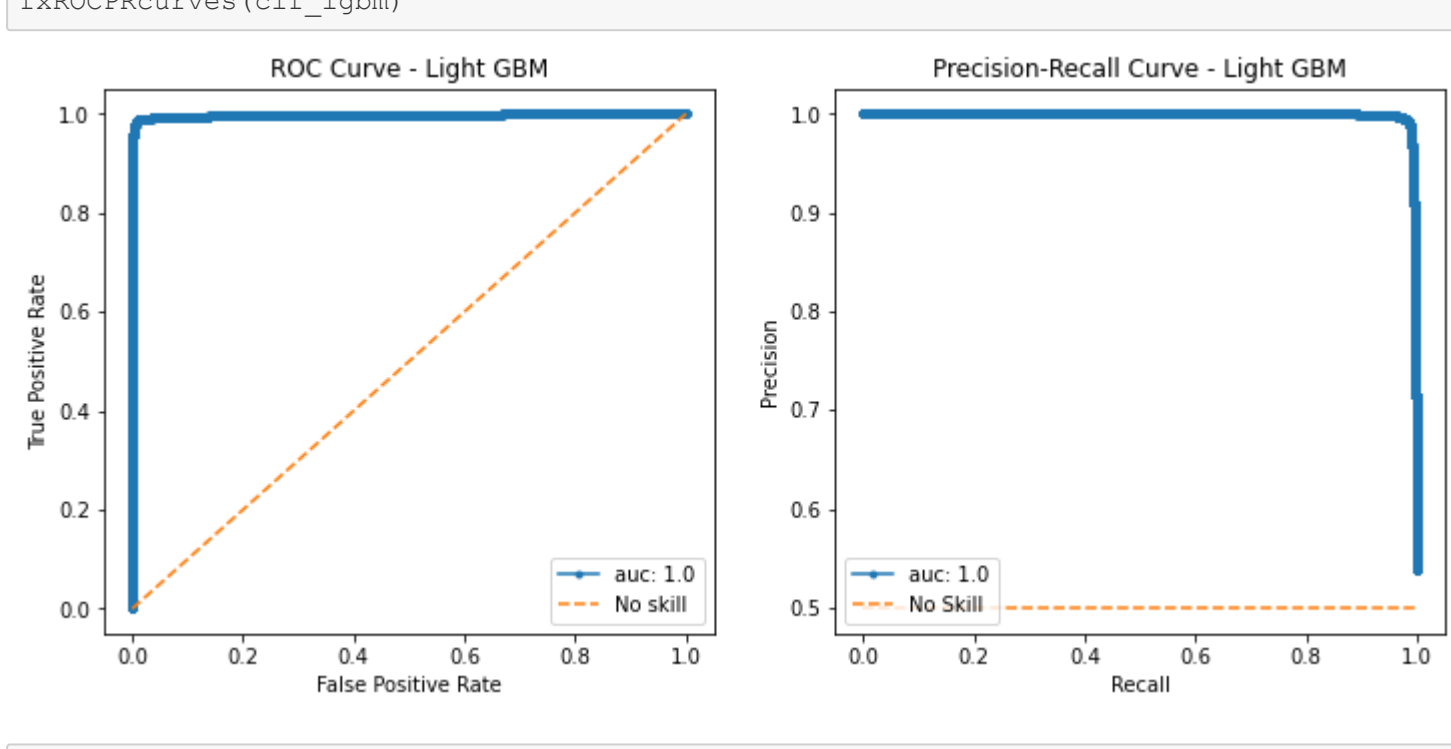
Using Light GBM classifier

In [30]:
```
clf_lgbm = clfLightGBM(dfsm_train, 'TARGET')
print('Recall:', clf_lgbm['recall'])
print('Precision: ', clf_lgbm['precision'])
```

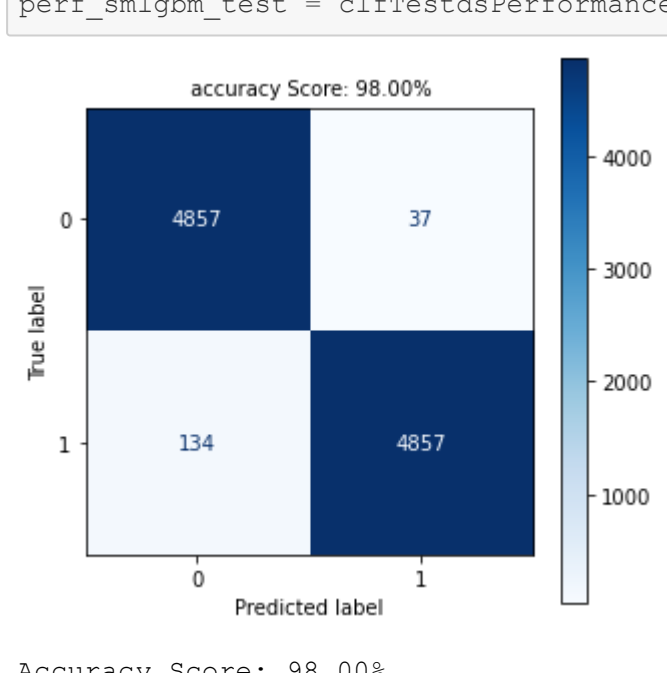



```
Accuracy:  99%
Recall:  97.65%
Precision:  99.53%
F1:  98.58%
dictionary keys:
 ['type', 'prediction', 'accuracy', 'recall', 'precision', 'f1', 'df_proba', 'importances', 'params',
'metrics_report', 'confusionmatrix', 'scale', 'classifier', 'targetlabel', 'y_pred_prob', 'ROC_PR']
Recall: 97.65%
Precision:  99.53%
```

In [31]:
```
fxROCPRcurves(clf_lgbm)
```



In [32]:
```
perf_smlgbm_test = clfTestdsPerformance(dfsm_test, 'TARGET', clf_lgbm['classifier'])
```




```
Accuracy Score: 98.00%
Recall Score: 97.32%
Precision Score: 99.24%
F1 Score: 98.27%

              precision    recall  f1-score   support

           0       0.97      0.99      0.98      4894
           1       0.99      0.97      0.98      4991

    accuracy                           0.98      9885
   macro avg       0.98      0.98      0.98      9885
weighted avg       0.98      0.98      0.98      9885

dictionary keys:
 ['report', 'accuracy', 'recall', 'precision', 'f1', 'confusionmatrix', 'prediction']
```

In [33]:
```
clfPerfPlot(perf_smlgbm_test['prediction'], xcol='AGE', ycol='INCOME')
```