In [8]:
```
import numpy as np
import pandas as pd

import plotly as py
import plotly_express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score as f1
from sklearn.metrics import confusion_matrix
#import scikitplot as skplt
```

In [9]:
```
df = pd.read_csv("BankChurners.csv")
df.head()
```

Out[9]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	I
0	768805383	Existing Customer	45	M	3	High School	
1	818770008	Existing Customer	49	F	5	Graduate	
2	713982108	Existing Customer	51	M	3	Graduate	
3	769911858	Existing Customer	40	F	4	High School	
4	709106358	Existing Customer	40	M	3	Uneducated	

5 rows × 23 columns

数据基本信息

In [10]:
```
df.shape

# 结果
(10127, 23)
```

Out[10]:
```
(10127, 23)
```

In [11]:
```
# 全部字段
columns = df.columns
columns
```

Out[11]:
```
Index(['CLIENTNUM', 'Attrition_Flag', 'Customer_Age', 'Gender',
       'Dependent_count', 'Education_Level', 'Marital_Status',
       'Income_Category', 'Card_Category', 'Months_on_book',
       'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',
       'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio',
       'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1',
       'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2'],
      dtype='object')
```

字段解释为：

CLIENTNUM：Client number - Unique identifier for the customer holding the account

Attrition_Flag：Flag indicative of account closure in next 6 months (between Jan to Jun 2013)

Customer_Age：Age of the account holder

Gender：Gender of the account holder

Dependent_count：Number of people financially dependent on the account holder

Education_Level：Educational qualification of account holder (ex - high school, college grad etc.)

Marital_Status：Marital status of account holder (Single, Married, Divorced, Unknown)

Income_Category：Annual income category of the account holder

Card_Category：Card type depicting the variants of the cards by value proposition (Blue, Silver and Platinum)

Months_on_book：Number of months since the account holder opened an an account with the lender

Total_Relationship_Count：Total number of products held by the customer. Total number of relationships the account holder has with the bank (example - retail bank, mortgage, wealth management etc.)

Months_Inactive_12_mon：Total number of months inactive in last 12 months

Contacts_Count_12_mon：Number of Contacts in the last 12 months. No. of times the

account holder called to the call center in the past 12 months

Credit_Limit：Credit limit

Total_Revolving_Bal：Total amount as revolving balance

Avg_Open_To_Buy：Open to Buy Credit Line (Average of last 12 months)

Total_Amt_Chng_Q4_Q1：Change in Transaction Amount (Q4 over Q1)

Total_Trans_Amt：Total Transaction Amount (Last 12 months)

Total_Trans_Ct：Total Transaction Count (Last 12 months)

Total_Ct_Chng_Q4_Q1：Change in Transaction Count (Q4 over Q1)

Avg_Utilization_Ratio：Average Card Utilization Ratio

Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependen

Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependen

In [12]:
```
df.dtypes   # 字段类型；部分截图
```

Out[12]:
```
CLIENTNUM
int64
Attrition_Flag
object
Customer_Age
int64
Gender
object
Dependent_count
int64
Education_Level
object
Marital_Status
object
Income_Category
object
Card_Category
object
Months_on_book
int64
Total_Relationship_Count
int64
Months_Inactive_12_mon
int64
Contacts_Count_12_mon
int64
Credit_Limit
float64
Total_Revolving_Bal
int64
Avg_Open_To_Buy
float64
Total_Amt_Chng_Q4_Q1
float64
Total_Trans_Amt
int64
Total_Trans_Ct
int64
Total_Ct_Chng_Q4_Q1
float64
Avg_Utilization_Ratio
float64
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_De
pendent_count_Education_Level_Months_Inactive_12_mon_1    float64
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_De
pendent_count_Education_Level_Months_Inactive_12_mon_2    float64
dtype: object
```

In [14]:
```
# 不同字段类型的统计
#通过下面的代码能够统计不同类型下的字段数量:
pd.value_counts(df.dtypes)
```

Out[14]:
```
int64      10
float64     7
object      6
dtype: int64
```

In [15]:
```
df.describe().style.background_gradient(cmap="ocean_r")  # 表格美化输出
```

Out[15]:

	CLIENTNUM	Customer_Age	Dependent_count	Months_on_book	Total_Relatio
count	10127.000000	10127.000000	10127.000000	10127.000000	1
mean	739177606.333663	46.325960	2.346203	35.928409	
std	36903783.450231	8.016814	1.298908	7.986416	
min	708082083.000000	26.000000	0.000000	13.000000	
25%	713036770.500000	41.000000	1.000000	31.000000	
50%	717926358.000000	46.000000	2.000000	36.000000	
75%	773143533.000000	52.000000	3.000000	40.000000	
max	828343083.000000	73.000000	5.000000	56.000000	

缺失值

In [19]:
```
# 缺失值比例：数据中没有缺失值
total = df.isnull().sum().sort_values(ascending=False)
Percentage = total / len(df)

# 每个字段的缺失值统计
df.isnull().sum()
```

Out[19]:
```
CLIENTNUM
0
Attrition_Flag
0
Customer_Age
0
Gender
0
Dependent_count
0
Education_Level
0
Marital_Status
0
Income_Category
0
Card_Category
0
Months_on_book
0
Total_Relationship_Count
0
Months_Inactive_12_mon
0
Contacts_Count_12_mon
0
Credit_Limit
0
Total_Revolving_Bal
0
Avg_Open_To_Buy
0
Total_Amt_Chng_Q4_Q1
0
Total_Trans_Amt
0
Total_Trans_Ct
0
Total_Ct_Chng_Q4_Q1
0
Avg_Utilization_Ratio
0
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_De
pendent_count_Education_Level_Months_Inactive_12_mon_1    0
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_De
pendent_count_Education_Level_Months_Inactive_12_mon_2    0
dtype: int64
```

根据值的降序排列，第一个是0，结果表明数据本身是没有缺失值的

删除无关字段

In [21]:
```
no_use = np.arange(21, df.shape[1])  # 最后两个字段
no_use
```

Out[21]:
```
array([21, 22])
```

In [22]:
```
# 1、删除多个字段
df.drop(df.columns[no_use], axis=1, inplace=True)
```

In [23]:
```
#CLIENTNUM表示的客户编号的信息，对建模无用直接删除：
# 2、删除单个字段
df.drop("CLIENTNUM", axis=1, inplace=True)
```

In [24]:
```
#新生成的df的字段（删除了无效字段之后）：
df.columns
```

Out[24]:
```
Index(['Attrition_Flag', 'Customer_Age', 'Gender', 'Dependent_count',
       'Education_Level', 'Marital_Status', 'Income_Category', 'Card_Categor
y',
       'Months_on_book', 'Total_Relationship_Count', 'Months_Inactive_12_mon
',
       'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',
       'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio'],
      dtype='object')
```

In [26]:
```
#再次查看数据的描述统计信息：
df.describe().style.background_gradient(cmap="ocean_r")
```

Out[26]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months
count	10127.000000	10127.000000	10127.000000	10127.000000	
mean	46.325960	2.346203	35.928409	3.812580	
std	8.016814	1.298908	7.986416	1.554408	
min	26.000000	0.000000	13.000000	1.000000	
25%	41.000000	1.000000	31.000000	3.000000	
50%	46.000000	2.000000	36.000000	4.000000	
75%	52.000000	3.000000	40.000000	5.000000	
max	73.000000	5.000000	56.000000	6.000000	

EDA-Exploratory Data Analysis

In [27]:
```
#基于使用频率和数值特征
#取出和用户的数值型字段信息：
# df_frequency = df[["Customer_Age","Total_Trans_Ct","Total_Trans_Amt","Mont

df_frequency = pd.concat([df['Customer_Age'],
                          df['Total_Trans_Ct'],
                          df['Total_Trans_Amt'],
                          df['Months_Inactive_12_mon'],
                          df['Credit_Limit'],
                          df['Attrition_Flag']],
                         axis=1)

df_frequency.head()
```

Out[27]:

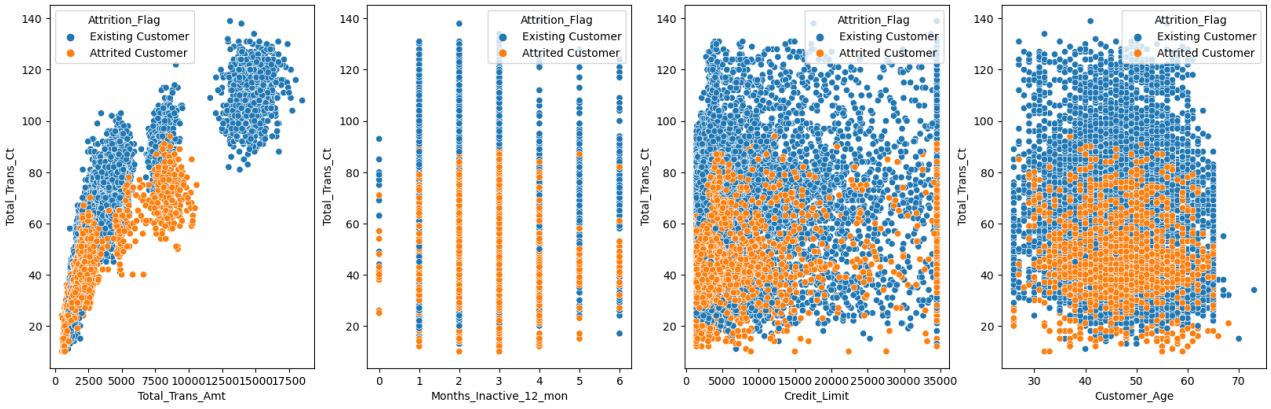
	Customer_Age	Total_Trans_Ct	Total_Trans_Amt	Months_Inactive_12_mon	Credit_Limit
0	45	42	1144	1	12691.0
1	49	33	1291	1	8256.0
2	51	20	1887	1	3418.0
3	40	20	1171	4	3313.0
4	40	28	816	1	4716.0

In [29]:
```
#探索在不同的Attrition_Flag下，两两字段之间的关系：
df["Attrition_Flag"].value_counts()
```

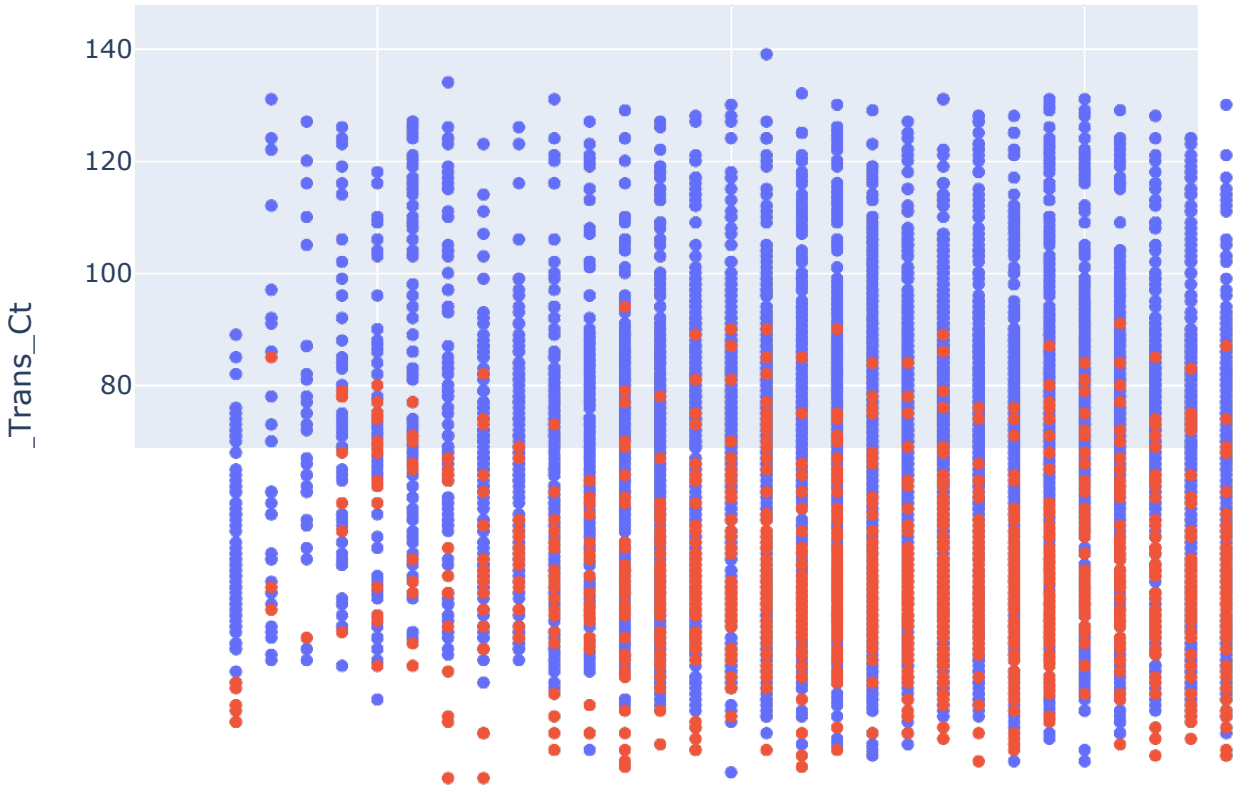
Out[29]:
```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
```

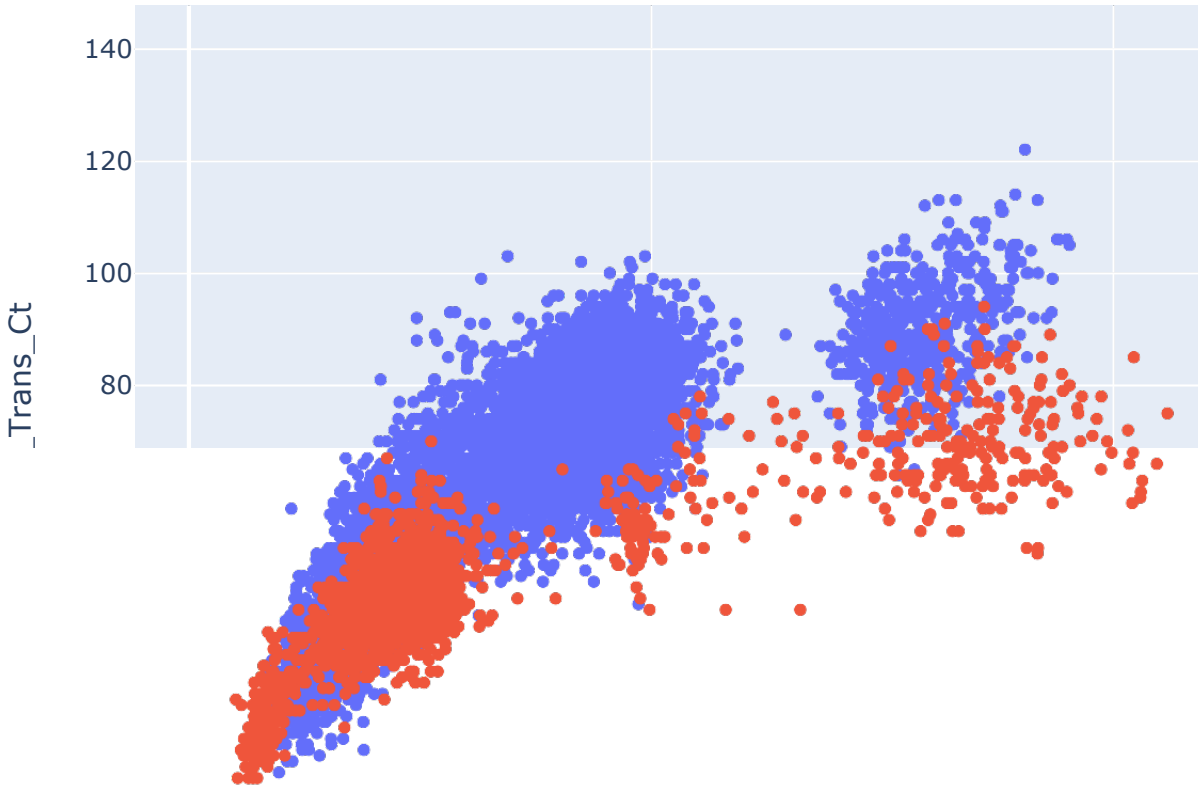
结果表明：现有顾客为8500，流失客户为1627

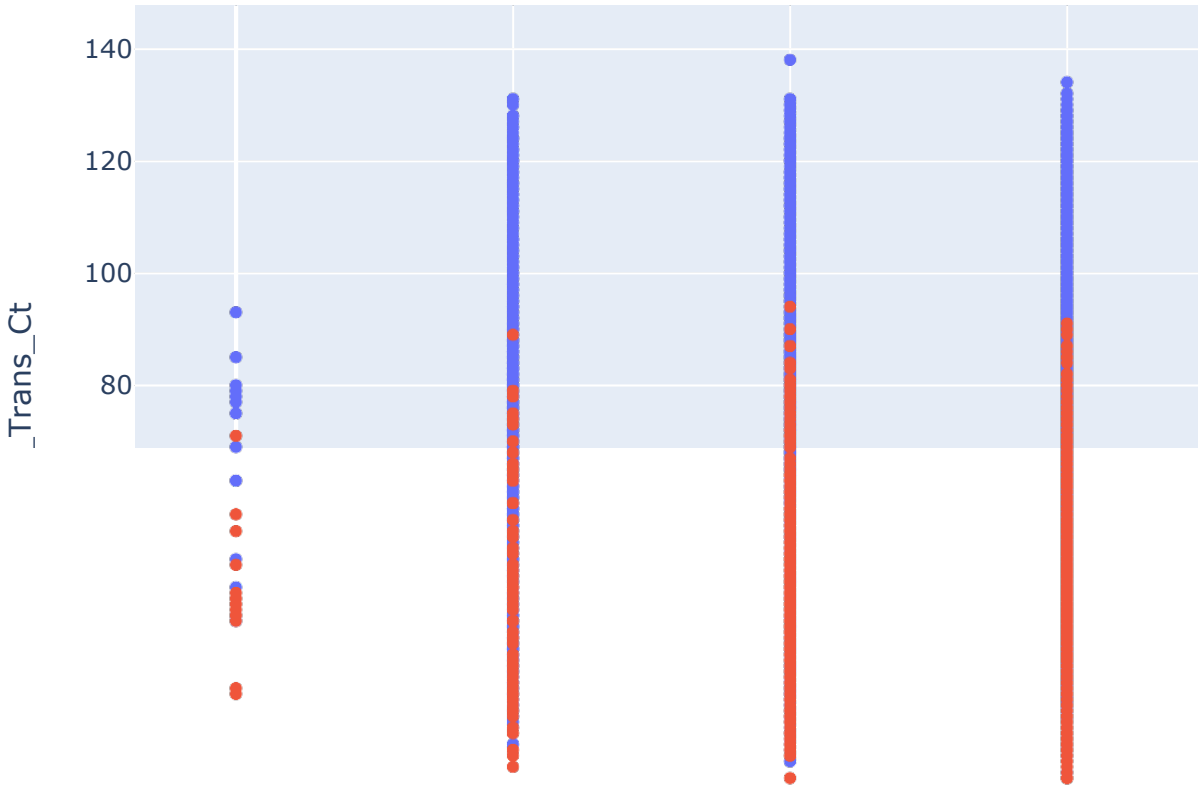
In [31]:
```
# 定义画布大小

fig, ax = plt.subplots(ncols=4, figsize=(20,6))

sns.scatterplot(data=df_frequency,
                x="Total_Trans_Amt",
                y="Total_Trans_Ct",
                hue="Attrition_Flag",
                ax=ax[0])

sns.scatterplot(data=df_frequency,
                x="Months_Inactive_12_mon",
                y="Total_Trans_Ct",
                hue="Attrition_Flag",
                ax=ax[1])

sns.scatterplot(data=df_frequency,
                x="Credit_Limit",
                y="Total_Trans_Ct",
                hue="Attrition_Flag",
                ax=ax[2])

sns.scatterplot(data=df_frequency,
              x="Customer_Age",
              y="Total_Trans_Ct",
              hue="Attrition_Flag",
              ax=ax[3])

plt.show()
```



In [58]:
```
#基于plotly的实现:
for col in ["Customer_Age","Total_Trans_Amt","Months_Inactive_12_mon","Credi
    fig = px.scatter(df_frequency,
                     x=col,
                     y="Total_Trans_Ct",
                     color="Attrition_Flag")
    fig.show()
```


140
120
100
80
_Trans_Ct

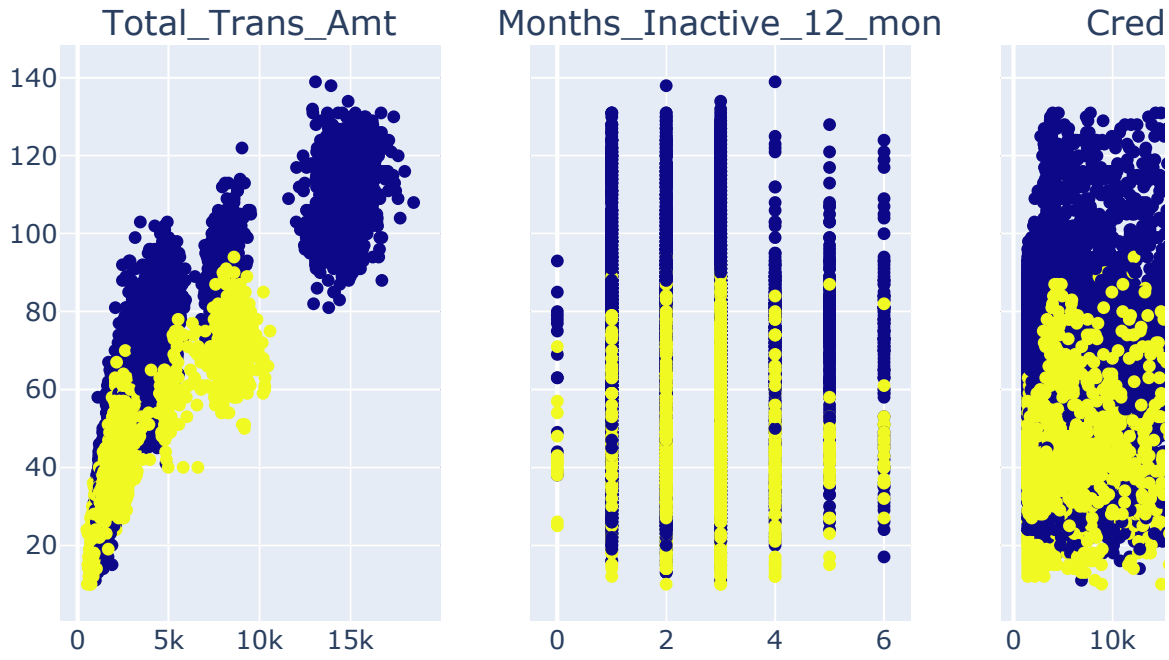

140
120
100
80
_Trans_Ct


140
120
100
80
_Trans_Ct


140
120
100
80
_Trans_Ct

In [60]:

```
#上面展示的一个字段和Total_Trans_Ct的关系。下面是基于go.Scatter实现：

# 生成一个副本

df_frequency_copy = df_frequency.copy()
df_frequency_copy["Attrition_Flag_number"] = df_frequency_copy["Attrition_Fl

# 两个基本参数：设置行、列

four_columns = ["Total_Trans_Amt","Months_Inactive_12_mon","Credit_Limit","C

fig = make_subplots(rows=1,
                    cols=4,
                    start_cell="top-left",
                    shared_yaxes=True,
                    subplot_titles=four_columns  # 子图
                   )

for i, v in enumerate(four_columns):
    r = i // 4 + 1  # 行
    c = (i + 1) % 4  # 列-余数

    if c == 0:
        fig.add_trace(go.Scatter(x=df_frequency_copy[v].tolist(),
                                 y=df_frequency_copy["Total_Trans_Ct"].tolist(),
                                 mode='markers',
                                 marker=dict(color=df_frequency_copy.Attrition_F
                  row=r, col=4)

    else:
        fig.add_trace(go.Scatter(x=df_frequency_copy[v].tolist(),
                                 y=df_frequency_copy["Total_Trans_Ct"].tolist(),
                                 mode='markers',
                                 marker=dict(color=df_frequency_copy.Attrition_F

                  row=r, col=c)

fig.update_layout(width=1000, height=450, showlegend=False)

fig.show()
```



蓝色：现有客户；黄色：流失客户

我们得到如下的几点结论：

图1：用户每年花费的金额越高，越可能留下来（非流失）

2-3个月不进行互动，用户流失的可能性较高

用户的信用额度越高，留下来的可能性越大

从图3中观察到：流失客户的信用卡使用次数大部分低于100次

从第4个图中观察到，用户年龄分布不是重要因素

基于用户人口统计信息

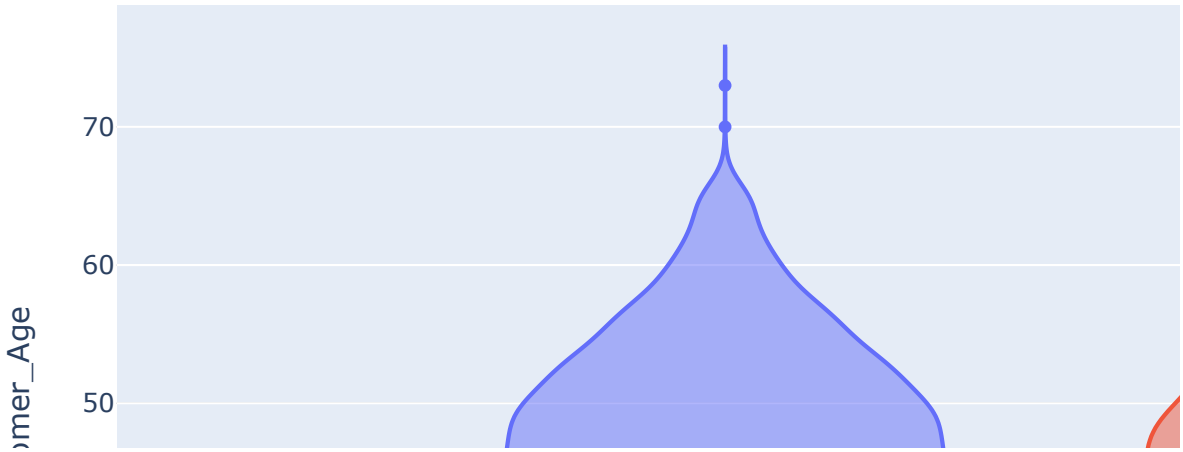
用户的人口统计信息主要是包含：用户年龄、性别、受教育程度、状态（单身、已婚等）、收入水平等信息

In [32]:
```
#取出相关的字段进行分析：

df_demographic=df[['Customer_Age',
                   'Gender',
                   'Education_Level',
                   'Marital_Status',
                   'Income_Category',
                   'Attrition_Flag']]

df_demographic.head()
```

Out[32]:

	Customer_Age	Gender	Education_Level	Marital_Status	Income_Category	Attrition_Flag
0	45	M	High School	Married	$60K-80K	Existing Customer
1	49	F	Graduate	Single	Less than $40K	Existing Customer
2	51	M	Graduate	Married	$80K-120K	Existing Customer
3	40	F	High School	Unknown	Less than $40K	Existing Customer
4	40	M	Uneducated	Married	$60K-80K	Existing Customer

In [61]:
```
#不同类型顾客的年龄分布
px.violin(df_demographic,
          y="Customer_Age",
          color="Attrition_Flag")
```



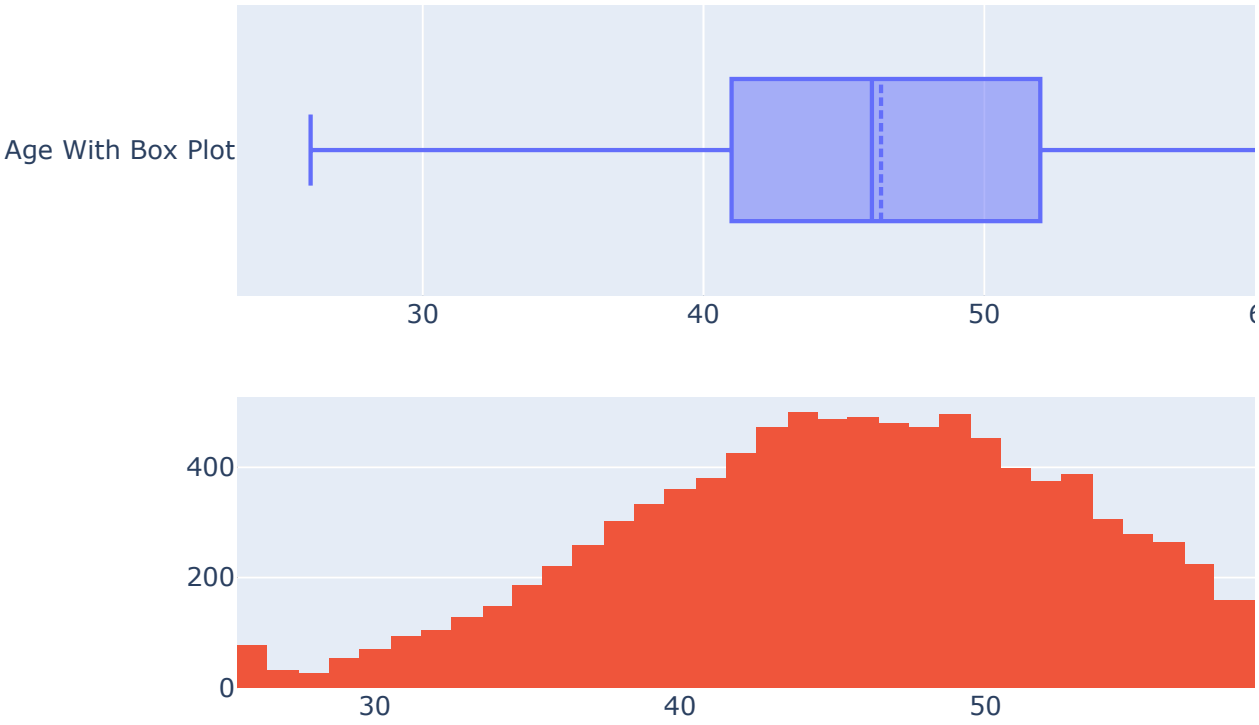
从上面的小提琴图看出来，不同类型的用户在年龄上的分布是类似的。

结论：年龄并不是用户是否流失的关键因素

In [33]:

```
#顾客的年龄分布
fig = make_subplots(rows=2, cols=1)

trace1=go.Box(x=df['Customer_Age'],name='Age With Box Plot',boxmean=True)
trace2=go.Histogram(x=df['Customer_Age'],name='Age With Histogram')

fig.add_trace(trace1, row=1,col=1)
fig.add_trace(trace2, row=2,col=1)

fig.update_layout(height=500, width=1000, title_text="用户年龄分布")
fig.show()
```

用户年龄分布

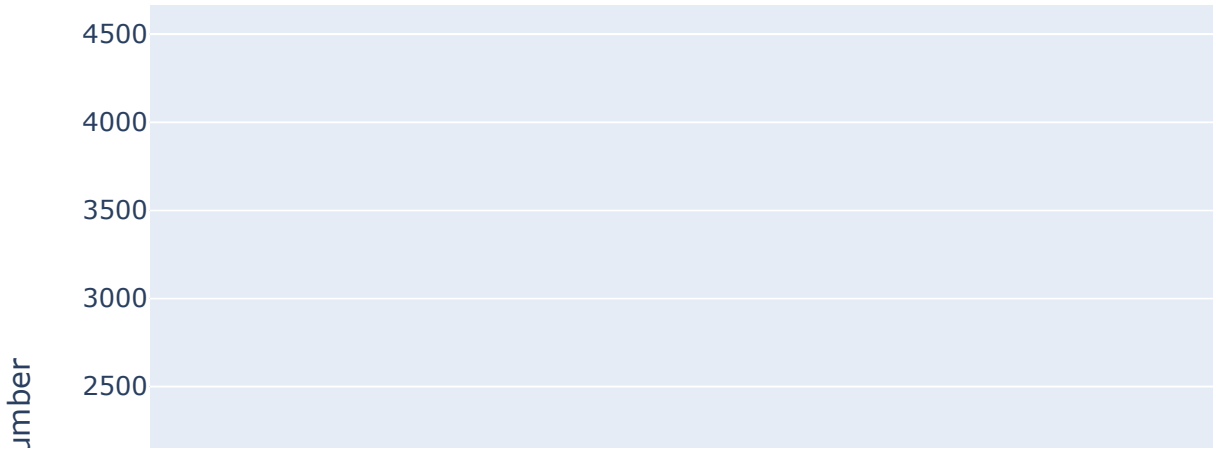


In [62]:
```
#不同类型下不同性别顾客统计
flag_gender = df.groupby(["Attrition_Flag","Gender"]).size().reset_index().r
flag_gender
```

Out[62]:

	Attrition_Flag	Gender	number
0	Attrited Customer	F	930
1	Attrited Customer	M	697
2	Existing Customer	F	4428
3	Existing Customer	M	4072

In [63]:
```
fig = px.bar(flag_gender,
             x="Attrition_Flag",
             y="number",
             color="Gender",
             barmode="group",
             text="number")

fig.show()
```



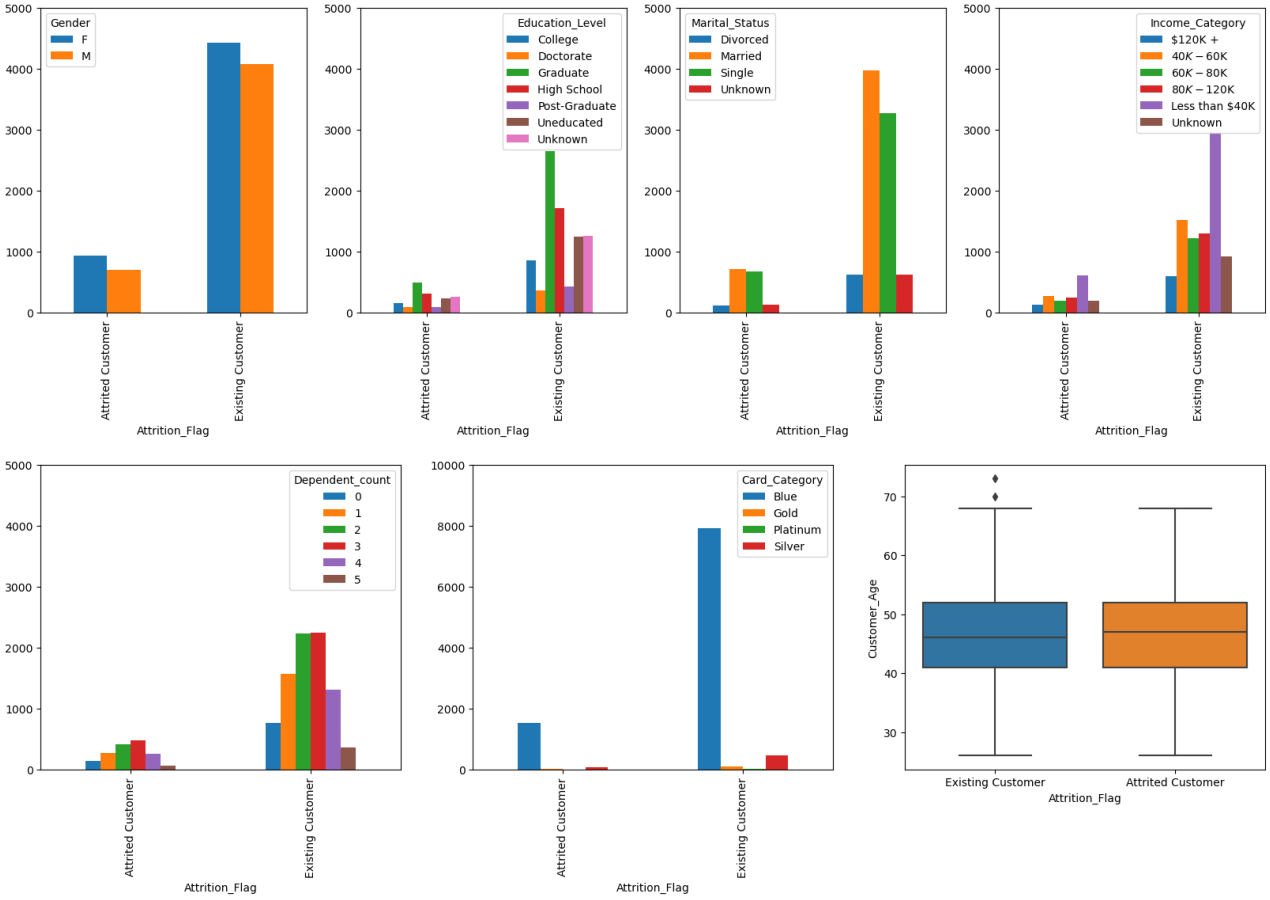
从上面的柱状图中看出来：

女性在本次数据中高于男性；在两种不同类型的客户中女性也是高于男性

数据不平衡：现有客户和流失客户是不平衡的，大约是8400:1600

交叉表统计分析

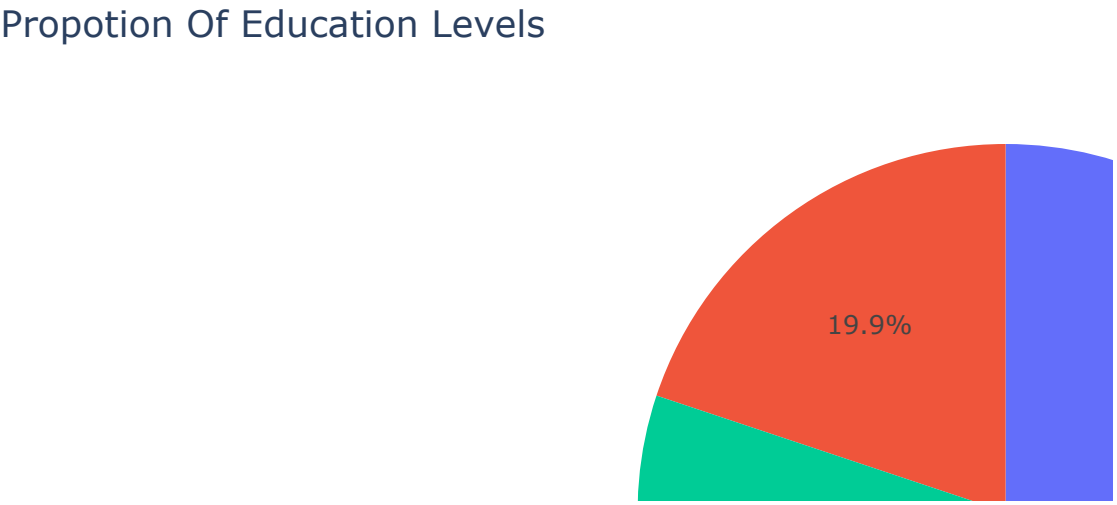
In [64]:
```
#基于pandas中交叉表的数据统计分析。解释交叉表很好的文章：https://pbpython.com/pandas

fig, (ax1,ax2,ax3,ax4) = plt.subplots(ncols=4, figsize=(20,5))

pd.crosstab(df["Attrition_Flag"],df["Gender"]).plot(kind="bar", ax=ax1, ylim
pd.crosstab(df["Attrition_Flag"],df["Education_Level"]).plot(kind="bar", ax=
pd.crosstab(df["Attrition_Flag"],df["Marital_Status"]).plot(kind="bar", ax=a
pd.crosstab(df["Attrition_Flag"],df["Income_Category"]).plot(kind="bar", ax=


fig, (ax1,ax2,ax3) = plt.subplots(ncols=3, figsize=(20,5))
pd.crosstab(df['Attrition_Flag'],df['Dependent_count']).plot(kind='bar',ax=a
pd.crosstab(df['Attrition_Flag'],df['Card_Category']).plot(kind='bar',ax=ax2

_box = sns.boxplot(data=df_demographic,x='Attrition_Flag',y='Customer_Age',

plt.show()
```



可以观察到：在两种客户中，不同的教育水平和个人状态的分布是类似的。
这个结论也验证了：年龄并不是影响现有或者流失客户的因素。

受教育程度

In [65]:
```
fig = px.pie(df,names='Education_Level',title='Propotion Of Education Levels
fig.show()
```



对比两种客户数量

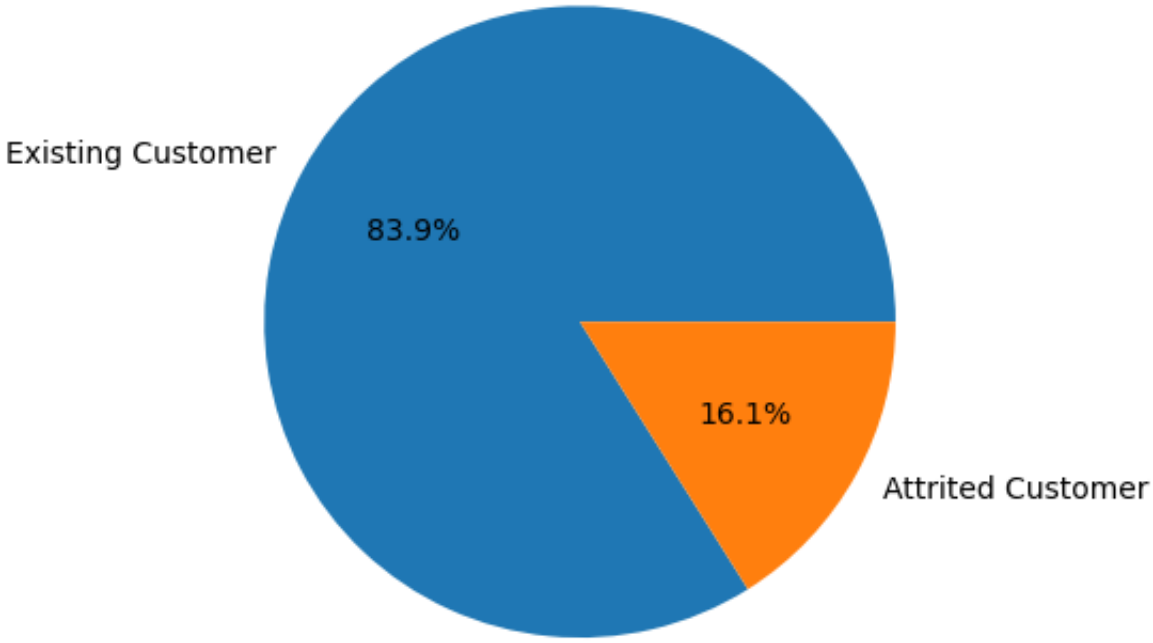
In [66]:
```
churn = df["Attrition_Flag"].value_counts()
churn
```

Out[66]:
```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
```

In [67]:
```
churn.keys()
```

Out[67]:
```
Index(['Existing Customer', 'Attrited Customer'], dtype='object')
```

In [68]:
```
plt.pie(x=churn, labels=churn.keys(),autopct="%.1f%%")

plt.show()
```



上面的饼图表明：

现有客户还是占据了绝大部分 后面将通过采样的方式使得两种类型的客户数量保持平衡。

相关性

现有数据中的字段涉及到分类型和数值型，采取不同的分析和编码方式

数值型变量：使用相关系数Pearson

分类型变量：使用Cramer's V ； 克莱姆相关系数，常用于分析双变量之间的关系

参考内容：https://blog.csdn.net/deecheanW/article/details/120474864

In [69]:
```
# 字符型字段
# 相同效果：df.select_dtypes(include="O")
df_categorical=df.loc[:,df.dtypes==np.object_]
df_categorical.head()

# 数值型字段
df_number = df.select_dtypes(exclude="O")
df_number.head()
```

Out[69]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Ina
0	45	3	39	5	
1	49	5	44	6	
2	51	3	36	4	
3	40	4	34	3	
4	40	3	21	5	

对Attrition_Flag字段执行独热码编码操作：

In [70]:
```
# 先保留原信息
df_number["Attrition_Flag"] = df.loc[:, "Attrition_Flag"]
```

类型编码

In [71]:
```
from sklearn import preprocessing

label = preprocessing.LabelEncoder()
df_categorical_encoded = pd.DataFrame()

# 对分类型的字段进行类型编码
for i in df_categorical.columns:
    df_categorical_encoded[i] = label.fit_transform(df_categorical[i])
```

In [72]:
```
### 计算克莱姆系数-cramers_V
```

In [73]:
```
from scipy.stats import chi2_contingency

# 定义计算克莱姆系数的函数
def cal_cramers_v(v1,v2):
    crosstab = np.array(pd.crosstab(v1,v2,rownames=None,colnames=None))
    stat = chi2_contingency(crosstab)[0]

    obs = np.sum(crosstab)
    mini = min(crosstab.shape) - 1

    return stat / (obs * mini)
```

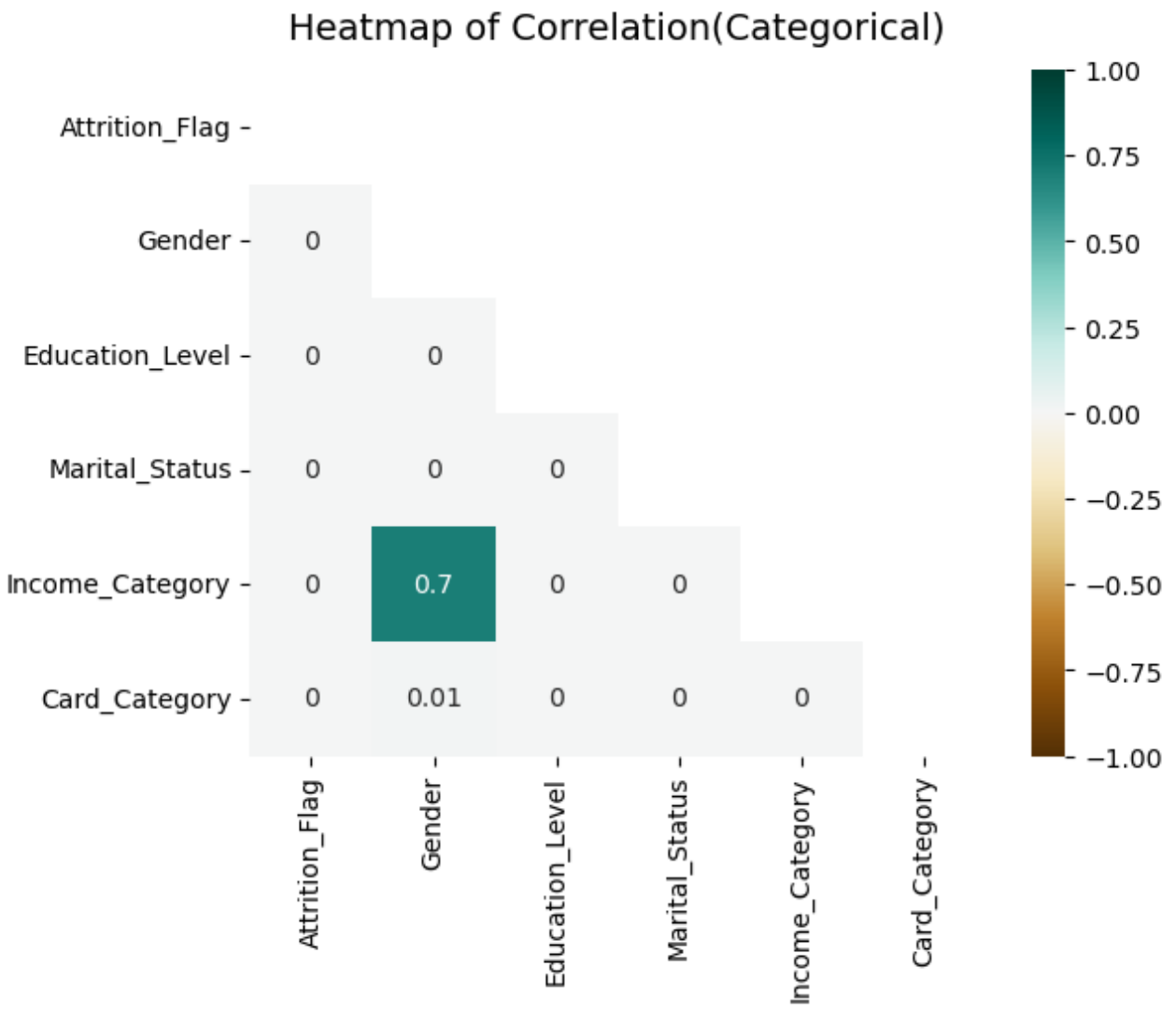
In [74]:
```
rows = []
for v1 in df_categorical_encoded:
    col = []
    for v2 in df_categorical_encoded:
        # 计算克莱姆系数
        cramers = cal_cramers_v(df_categorical_encoded[v1],df_categorical_en
        col.append(round(cramers, 2))
    rows.append(col)
```

In [75]:
```
# 克莱姆系数下的热力图

cramers_results = np.array(rows)

cramerv_matrix = pd.DataFrame(cramers_results,
                              columns=df_categorical_encoded.columns,
                              index=df_categorical_encoded.columns)
cramerv_matrix.head()
```

Out[75]:

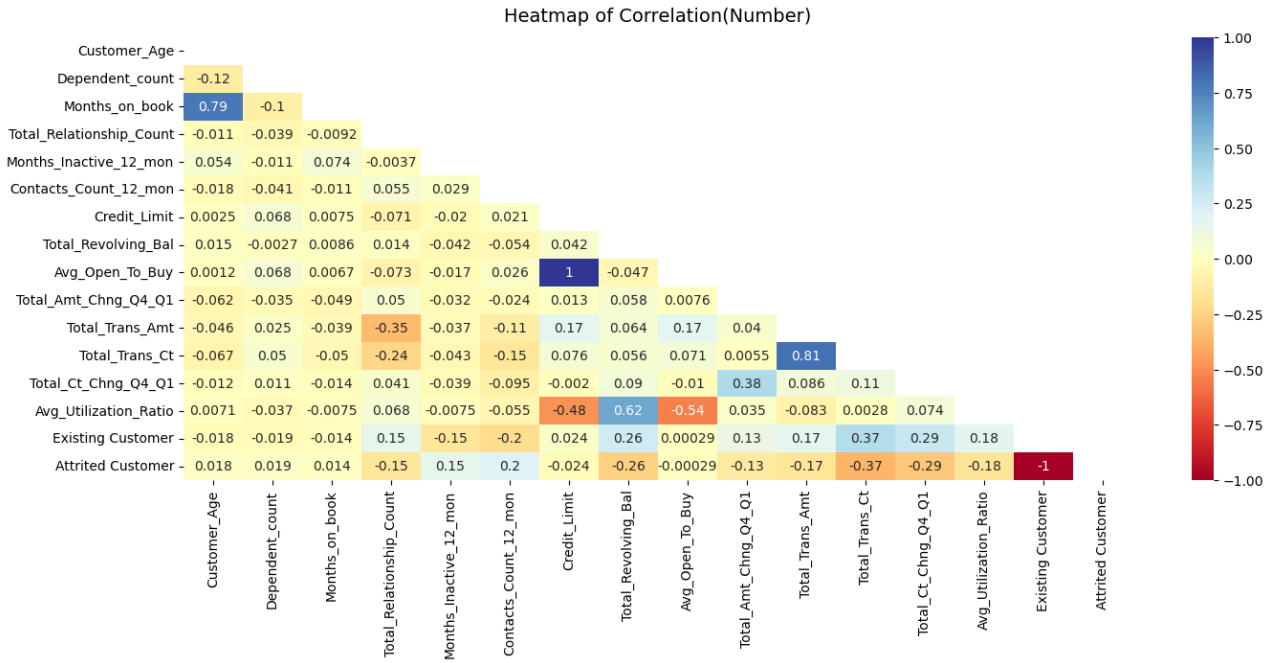
	Attrition_Flag	Gender	Education_Level	Marital_Status	Income_Category
Attrition_Flag	1.0	0.0	0.0	0.0	0.0
Gender	0.0	1.0	0.0	0.0	0.7
Education_Level	0.0	0.0	1.0	0.0	0.0
Marital_Status	0.0	0.0	0.0	1.0	0.0
Income_Category	0.0	0.7	0.0	0.0	1.0

In [76]:
```
#绘制相关的热力图:
mask = np.triu(np.ones_like(cramerv_matrix, dtype=np.bool_))
cat_heatmap = sns.heatmap(cramerv_matrix, # 系数矩阵
                          mask=mask,
                          vmin=-1,
                          vmax=1,
                          annot=True,
                          cmap="BrBG")

cat_heatmap.set_title("Heatmap of Correlation(Categorical)", fontdict={"font

plt.show()
```



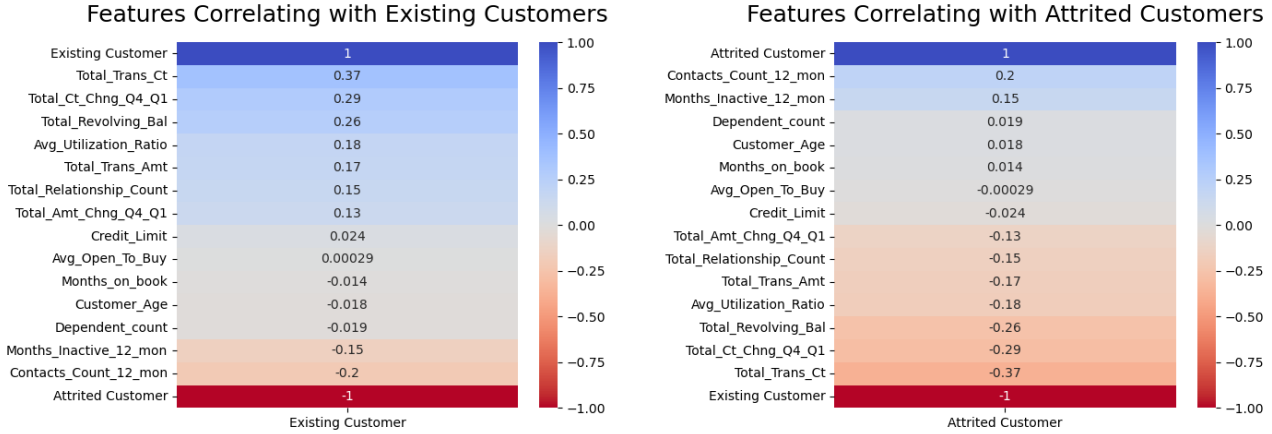
In [86]:
```
df_number['Existing Customer']=df_number['Attrition_Flag'].apply(lambda x: 1
df_number['Attrited Customer']=df_number['Attrition_Flag'].apply(lambda x: 0
```

In [87]:
```
# 基于数值型字段的相关系数

from scipy import stats

num_corr = df_number.corr()  # 相关系数
plt.figure(figsize = (16,6))

mask = np.triu(np.ones_like(num_corr, dtype=np.bool_))
heatmap_number = sns.heatmap(num_corr, mask=mask,
                             vmin=-1, vmax=1,
                             annot=True, cmap="RdYlBu")

heatmap_number.set_title("Heatmap of Correlation(Number)", fontdict={"fontsi

plt.show()
```



In [90]:

```
fig, ax = plt.subplots(ncols=2, figsize=(15,6))

heatmap = sns.heatmap(num_corr[["Existing Customer"]].sort_values(by="Existi
                      ax=ax[0],
                      vmin=-1,
                      vmax=1,
                      annot=True,
                      cmap="coolwarm_r")
heatmap.set_title("Features Correlating with Existing Customers",fontdict={"

heatmap = sns.heatmap(num_corr[["Attrited Customer"]].sort_values(by="Attrit
                      ax=ax[1],
                      vmin=-1,
                      vmax=1,
                      annot=True,
                      cmap="coolwarm_r")
heatmap.set_title("Features Correlating with Attrited Customers",fontdict={"

fig.tight_layout(pad=5)

plt.show()
```



小结：从上面右侧的热力图中能看到下面的字段和流失类型客户是无相关的。相关系数的值在正负0.1之间（右图）

Credit Limit

Average Open To Buy

Months On Book

Age

Dependent Count

现在我们考虑将上面的字段进行删除：

In [91]:
```
df_model = df.copy()

df_model = df_model.drop(['Credit_Limit','Customer_Age','Avg_Open_To_Buy','M
```

用户标识编码

In [92]:
```
df_model['Attrition_Flag'] = df_model['Attrition_Flag'].map({'Existing Custo
```

In [93]:
```
#剩余字段的独热码:
df_model=pd.get_dummies(df_model)
```

建模

切分数据

在之前已经验证过现有客户和流失客户的数量是不均衡的，我们使用SMOTE(Synthetic Minority Oversampling Technique，通过上采样合成少量的数据)采样来平衡数据。

In [96]:
```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
```

In [97]:
```
# 特征和目标变量

# X = df_model.drop("Attrition_Flag", axis=1, inplace=True)
X = df_model.loc[:, df_model.columns != "Attrition_Flag"]
y = df_model["Attrition_Flag"]

# 分割数据
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, ra
```

SMOTE采样

In [98]:
```
sm = SMOTE(sampling_strategy="minority", k_neighbors=20, random_state=42)

# 实施采样过程
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
```

3种模型

In [99]:
```
# 1、随机森林

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(X_train_res, y_train_res)
```

Out[99]:
```
RandomForestClassifier()
```

一般在使用树模型建模的时候数据不需要归一化。但是在使用支持向量机的时候需要：

In [100…
```
# 2、支持向量机

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

# 使用支持向量机数据需要归一化

svm = make_pipeline(StandardScaler(), SVC(gamma='auto'))
svm.fit(X_train_res, y_train_res)
```

Out[100]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(gamma='auto'))])
```

In [101…
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(gamma='auto'))])
```

Out[101]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('svc', SVC(gamma='auto'))])
```

In [102…
```
# 3、提升树

from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(n_estimators=100,  # tree的个数
                                learning_rate=1.0,  # 学习率
                                max_depth=1,   # 叶子的最大深度
                                random_state=42)

gb.fit(X_train_res, y_train_res)
```

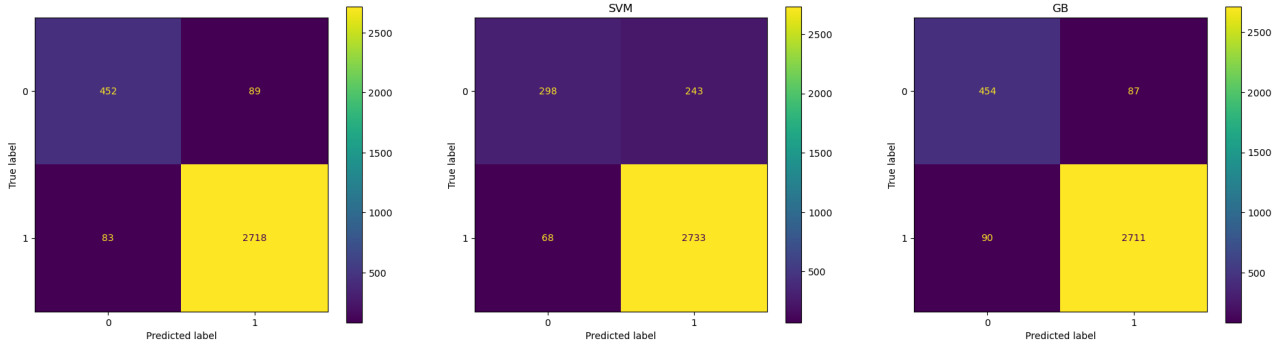
Out[102]:
```
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, random_state=42)
```

模型预测

In [104…
```
y_rf = rf.predict(X_test)
y_svm = svm.predict(X_test)
y_gb = gb.predict(X_test)
```

混淆矩阵

In [111…
```
from sklearn.metrics import ConfusionMatrixDisplay

fig,ax=plt.subplots(ncols=3, figsize=(20,6))

ConfusionMatrixDisplay.from_estimator(rf, X_test, y_test, ax=ax[0])
#x[0].title.set_text('RF')

ConfusionMatrixDisplay.from_estimator(svm, X_test, y_test, ax=ax[1])
ax[1].title.set_text('SVM')

ConfusionMatrixDisplay.from_estimator(gb, X_test, y_test, ax=ax[2])
ax[2].title.set_text('GB')
fig.tight_layout(pad=5)

plt.show()
```



分类模型得分

In [112…
```
# classification_report, recall_score, precision_score, f1_score

from sklearn.metrics import classification_report, recall_score, precision_s

print('Random Forest Classifier')
print(classification_report(y_test, y_rf))

print('------------------------')
print('Support Vector Machine')
print(classification_report(y_test, y_svm))

print('------------------------')
print('Gradient Boosting')
print(classification_report(y_test, y_gb))
```

```
Random Forest Classifier
              precision    recall  f1-score   support

           0       0.84      0.84      0.84       541
           1       0.97      0.97      0.97      2801

    accuracy                           0.95      3342
   macro avg       0.91      0.90      0.90      3342
weighted avg       0.95      0.95      0.95      3342

------------------------
Support Vector Machine
              precision    recall  f1-score   support

           0       0.81      0.55      0.66       541
           1       0.92      0.98      0.95      2801

    accuracy                           0.91      3342
   macro avg       0.87      0.76      0.80      3342
weighted avg       0.90      0.91      0.90      3342

------------------------
Gradient Boosting
              precision    recall  f1-score   support

           0       0.83      0.84      0.84       541
           1       0.97      0.97      0.97      2801

    accuracy                           0.95      3342
   macro avg       0.90      0.90      0.90      3342
weighted avg       0.95      0.95      0.95      3342
```

从3种模型的混淆矩阵和分类模型的相关评价指标来看：可以看到随机森林和提升树的结果都是优于支持向量机的

模型调参优化

针对随机森林和提升树模型采用两种不同的调参优化方法：

随机森林：随机搜索调参

梯度提升树：网格搜索调参

随机搜索调参-随机森林模型

In [113…
```
from sklearn.model_selection import RandomizedSearchCV
```

In [114…
```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num =
# n_estimators  # 随机森林中树的个数

max_features = ['auto', 'sqrt']
```

In [115…
```
# 每个tree的最大叶子数
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)

max_depth
```

Out[115]:
```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None]
```

In [117…
```
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]
```

随机搜索参数

In [118…
```
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

In [120…
```
rf_random = RandomizedSearchCV(
    estimator = rf,  # rf模型
    param_distributions=random_grid, # 搜索参数
    n_iter=30,
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1)

rf_random.fit(X_train_res, y_train_res)
print(rf_random.best_params_)
```

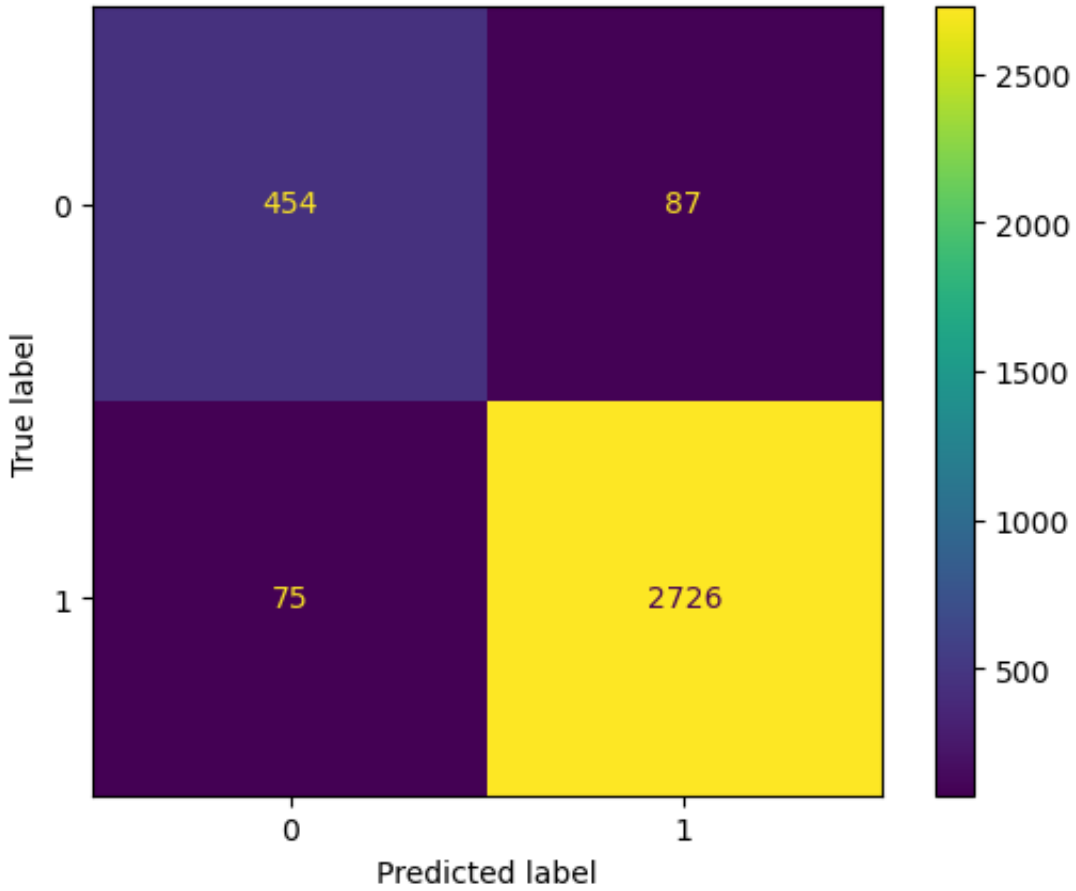
```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
{'n_estimators': 2000, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_f
eatures': 'auto', 'max_depth': 20, 'bootstrap': True}
```

使用搜索参数建模 使用上面搜索之后的参数再次建模：

In [122…

```
rf_clf_search= RandomForestClassifier(n_estimators=1400,
                                      min_samples_split=2,
                                      min_samples_leaf=1,
                                      max_features='auto',
                                      max_depth=110,
                                      bootstrap=True)

rf_clf_search.fit(X_train_res,y_train_res)
y_rf_opt=rf_clf_search.predict(X_test)

print('Random Forest Classifier (Optimized)')

print(classification_report(y_test, y_rf_opt))

_rf_opt=ConfusionMatrixDisplay.from_estimator(rf_clf_search, X_test, y_test)
```

```
Random Forest Classifier (Optimized)
              precision    recall  f1-score   support

           0       0.86      0.84      0.85       541
           1       0.97      0.97      0.97      2801

    accuracy                           0.95      3342
   macro avg       0.91      0.91      0.91      3342
weighted avg       0.95      0.95      0.95      3342
```



调参后的混淆矩阵：左上角的449变成452，说明分类的更加准确了

网格搜索调参-提升树模型

网格搜索参数

In [123…
```
from sklearn.model_selection import GridSearchCV

param_test1 = {'n_estimators':range(20,100,10)}
param_test1
```

Out[123]:
```
{'n_estimators': range(20, 100, 10)}
```

In [124…
```
# 实施搜索

grid_search1 = GridSearchCV(estimator = GradientBoostingClassifier(learning_
                                                                   min_samples_s
                                                                   min_samples_l
                                                                   max_depth=8,
                                                                   max_features=
                                                                   subsample=0.8
                                                                   random_state=
                            param_grid = param_test1, # 搜索参数
                            scoring='roc_auc',
                            n_jobs=4,
                            cv=5)

grid_search1.fit(X_train_res,y_train_res)

grid_search1.best_params_
```

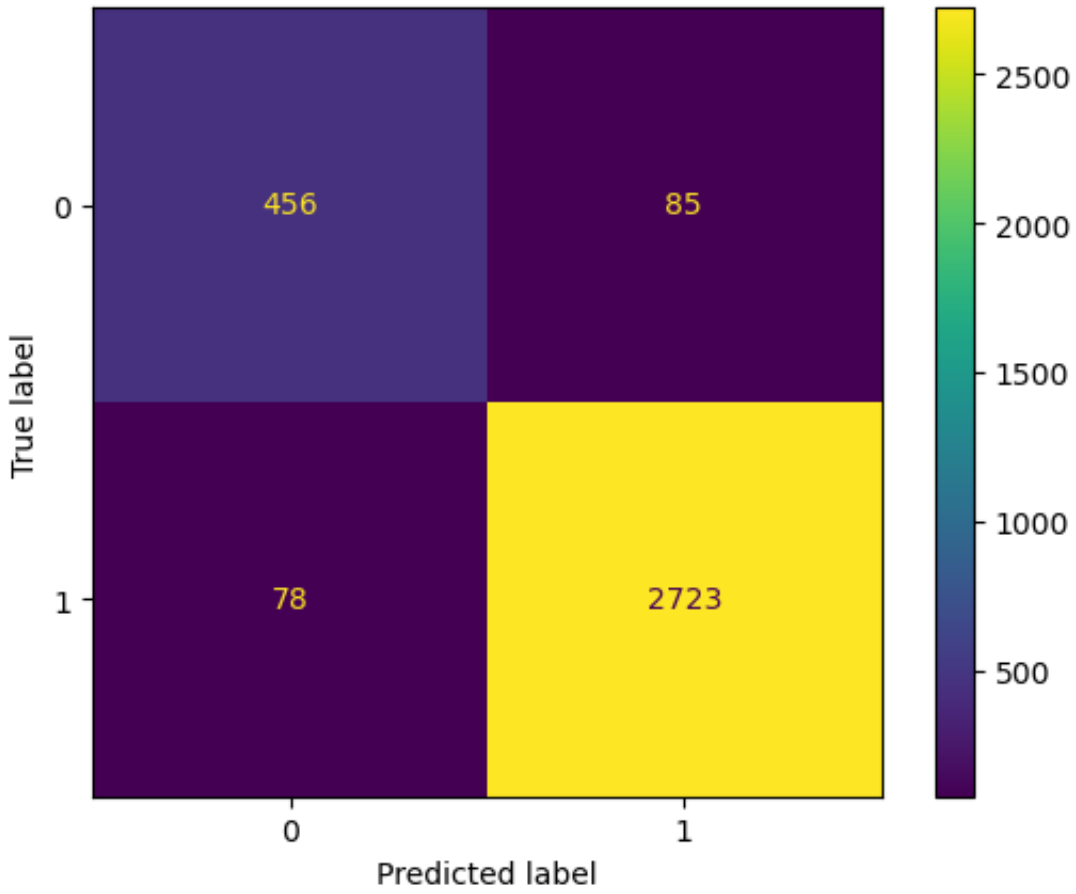
Out[124]:
```
{'n_estimators': 90}
```

使用搜索参数建模

In [126…
```
gb_clf_opt=GradientBoostingClassifier(n_estimators=90,  # 搜索到的参数90
                                      learning_rate=1.0,
                                      min_samples_split=500,
                                      min_samples_leaf=50,
                                      max_depth=8,
                                      max_features='sqrt',
                                      subsample=0.8,
                                      random_state=10)
# 再次拟合
gb_clf_opt.fit(X_train_res,y_train_res)

y_gb_opt=gb_clf_opt.predict(X_test)
print('Gradient Boosting (Optimized)')
print(classification_report(y_test, y_gb_opt))

print(recall_score(y_test,y_gb_opt,pos_label=0))
_gbopt=ConfusionMatrixDisplay.from_estimator(gb_clf_opt, X_test, y_test)
_gbopt
```

```
Gradient Boosting (Optimized)
              precision    recall  f1-score   support

           0       0.85      0.84      0.85       541
           1       0.97      0.97      0.97      2801

    accuracy                           0.95      3342
   macro avg       0.91      0.91      0.91      3342
weighted avg       0.95      0.95      0.95      3342

0.8428835489833642
```

Out[126]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1fb003a9dc0>



左上角的分类数目从454提升到456，也有一定的提升，但是效果并不是很明显

本notebook从一份用户相关的数据出发，从数据预处理、特征工程和编码，到建模分析和调参优化，完成了整个用户流失预警的全流程分析。整体模型的结果准确率达到了95%，召回率也达到了84.2%

In [ ]: