Background & Context¶

The Thera bank recently saw a steep decline in the number of users of their credit card, credit cards are a good source of income for banks because of different kinds of fees charged by the banks like annual fees, balance transfer fees, and cash advance fees, late payment fees, foreign transaction fees, and others. Some fees are charged on every user irrespective of usage, while others are charged under specified circumstances.
Customers' leaving credit cards services would lead bank to loss, so the bank wants to analyze the data of customers' and identify the customers who will leave their credit card services and reason for same – so that bank could improve upon those areas
You as a Data scientist at Thera bank need to come up with a classification model that will help bank improve their services so that customers do not renounce their credit cards

Objective¶

Explore and visualize the dataset. Build a classification model to predict if the customer is going to churn or not Optimize the model using appropriate techniques Generate a set of insights and recommendations that will help the bank

Data Dictionary:¶

- CLIENTNUM: Client number. Unique identifier for the customer holding the account
- Attrition_Flag: Internal event (customer activity) variable - if the account is closed then 1 else 0
- Customer_Age: Age in Years
- Gender: Gender of the account holder
- Dependent_count: Number of dependents
- Education_Level: Educational Qualification of the account holder
- Marital_Status: Marital Status of the account holder
- Income_Category: Annual Income Category of the account holder
- Card_Category: Type of Card
- Months_on_book: Period of relationship with the bank
- Total_Relationship_Count: Total no. of products held by the customer
- Months_Inactive_12_mon: No. of months inactive in the last 12 months
- Contacts_Count_12_mon: No. of Contacts in the last 12 months
- Credit_Limit: Credit Limit on the Credit Card
- Total_Revolving_Bal: Total Revolving Balance on the Credit Card
- 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

In [265]:

```
# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd
# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns
# Libraries to tune model, get different metric scores, and split data
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.metrics import accuracy_score,precision_score,recall_score
```

```
from sklearn import metrics
# Library to impute missing values
from sklearn.impute import KNNImputer
# Library to build a logistic regression model
from sklearn.linear_model import LogisticRegression
# Library to supress the warning
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
```

In [266]:

```
pd.set_option('display.max_columns', None)
```

In [342]:

```
data = pd.read_csv('BankChurners.csv')
```

In [343]:

```
df= data.copy()
```

In [344]:

```
df.head()
```

Out[344]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Incom ateg
0	768805383	Existing Customer	45	M	3	High School	Married	$60K $80K
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less t $40K
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K $120I
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less t $40K
4	709106358	Existing Customer	40	M	3	Uneducated	Married	$60K $80K

In [345]:

```
np.unique(df["Attrition_Flag"])
```

Out[345]:

```
array(['Attrited Customer', 'Existing Customer'], dtype=object)
```

1.Explore the dataset and extract insights using Exploratory Data Analysis¶

In [346]:

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
```

```
---  ------                    --------------  -----
 0   CLIENTNUM                 10127 non-null  int64
 1   Attrition_Flag            10127 non-null  object
 2   Customer_Age              10127 non-null  int64
 3   Gender                    10127 non-null  object
 4   Dependent_count           10127 non-null  int64
 5   Education_Level           10127 non-null  object
 6   Marital_Status            10127 non-null  object
 7   Income_Category           10127 non-null  object
 8   Card_Category             10127 non-null  object
 9   Months_on_book            10127 non-null  int64
 10  Total_Relationship_Count  10127 non-null  int64
 11  Months_Inactive_12_mon    10127 non-null  int64
 12  Contacts_Count_12_mon     10127 non-null  int64
 13  Credit_Limit              10127 non-null  float64
 14  Total_Revolving_Bal       10127 non-null  int64
 15  Avg_Open_To_Buy           10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 17  Total_Trans_Amt           10127 non-null  int64
 18  Total_Trans_Ct            10127 non-null  int64
 19  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 20  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(5), int64(10), object(6)
memory usage: 1.6+ MB
```

- There are no missing values

In [347]:

```
df.shape
```

Out[347]:

```
(10127, 21)
```

In [348]:

```
df.describe().T
```

Out[348]:

	count	mean	std	min	25%	50%	
CLIENTNUM	10127.0	7.391776e+08	3.690378e+07	708082083.0	7.130368e+08	7.179264e+08	7 0
Customer_Age	10127.0	4.632596e+01	8.016814e+00	26.0	4.100000e+01	4.600000e+01	5 0
Dependent_count	10127.0	2.346203e+00	1.298908e+00	0.0	1.000000e+00	2.000000e+00	3 0
Months_on_book	10127.0	3.592841e+01	7.986416e+00	13.0	3.100000e+01	3.600000e+01	4 0
Total_Relationship_Count	10127.0	3.812580e+00	1.554408e+00	1.0	3.000000e+00	4.000000e+00	5 0
Months_Inact	10127.0	2.341167e+	1.010622e+	0.0	2.000000e+	2.000000e+	3

	count	mean	std	min	25%	50%	
ive_12_mon		00	00		00	00	0
Contacts_Count_12_mon	10127.0	2.455317e+00	1.106225e+00	0.0	2.000000e+00	2.000000e+00	3 0
Credit_Limit	10127.0	8.631954e+03	9.088777e+03	1438.3	2.555000e+03	4.549000e+03	1 0
Total_Revolving_Bal	10127.0	1.162814e+03	8.149873e+02	0.0	3.590000e+02	1.276000e+03	1 0
Avg_Open_To_Buy	10127.0	7.469140e+03	9.090685e+03	3.0	1.324500e+03	3.474000e+03	9 0
Total_Amt_Chng_Q4_Q1	10127.0	7.599407e-01	2.192068e-01	0.0	6.310000e-01	7.360000e-01	8 0
Total_Trans_Amt	10127.0	4.404086e+03	3.397129e+03	510.0	2.155500e+03	3.899000e+03	4 0
Total_Trans_Ct	10127.0	6.485869e+01	2.347257e+01	10.0	4.500000e+01	6.700000e+01	8 0
Total_Ct_Chng_Q4_Q1	10127.0	7.122224e-01	2.380861e-01	0.0	5.820000e-01	7.020000e-01	8 0
Avg_Utilization_Ratio	10127.0	2.748936e-01	2.756915e-01	0.0	2.300000e-02	1.760000e-01	5 0

- Clientnum is a unique identifer and can be dropped.

In [349]:
```
df.drop(['CLIENTNUM'],axis=1,inplace=True)
```
In [350]:
```
df.describe(include=['object']).T
```
Out[350]:

	count	unique	top	freq
Attrition_Flag	10127	2	Existing Customer	8500
Gender	10127	2	F	5358
Education_Level	10127	7	Graduate	3128
Marital_Status	10127	4	Married	4687
Income_Category	10127	6	Less than $40K	3561

	count	unique	top	freq
Card_Category	10127	4	Blue	9436

Observations-

- Most of the Customers surveid are current customers and have an open account.
- Most of the customers completed graduate school.
- Most of the customers are Married.
- Most of the customers make less than $40K a year. - This seems weird since they have graduate degrees.
- Most customers are blue card members.

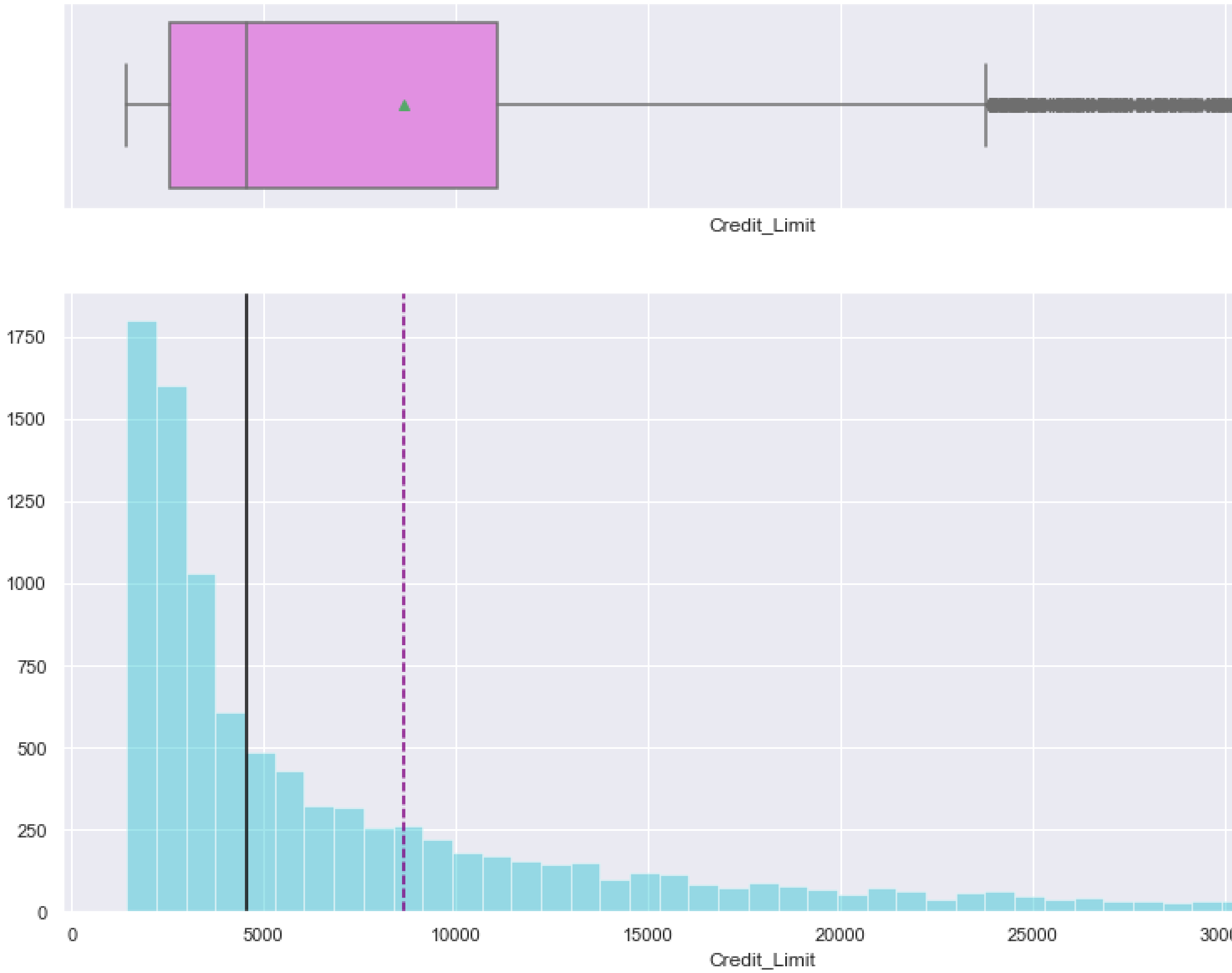
In [351]:

```
# While doing uni-variate analysis of numerical variables we want to study their
central tendency
# and dispersion.
# Let us write a function that will help us create boxplot and histogram for any input
numerical
# variable.
# This function takes the numerical column as the input and returns the boxplots
# and histograms for the variable.
# Let us see if this help us write faster and cleaner code.
def histogram_boxplot(feature, figsize=(15,10), bins = None):
    """ Boxplot and histogram combined
    feature: 1-d feature array
    figsize: size of fig (default (9,8))
    bins: number of bins (default None / auto)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(nrows = 2, # Number of rows of the subplot
grid= 2
                                           sharex = True, # x-axis will be shared among
all subplots
                                           gridspec_kw = {"height_ratios": (.25, .75)},
                                           figsize = figsize
                                           ) # creating the 2 subplots
    sns.boxplot(feature, ax=ax_box2, showmeans=True, color='violet') # boxplot will be
created and a star will indicate the mean value of the column
    sns.distplot(feature, kde=F, ax=ax_hist2, bins=bins,color = 'orange') if bins else
sns.distplot(feature, kde=False, ax=ax_hist2,color='tab:cyan') # For histogram
    ax_hist2.axvline(np.mean(feature), color='purple', linestyle='--') # Add mean to
the histogram
    ax_hist2.axvline(np.median(feature), color='black', linestyle='-') # Add median to
the histogram
```

Observation on Credit Limit¶

In [352]:

```
histogram_boxplot(data['Credit_Limit'])
```

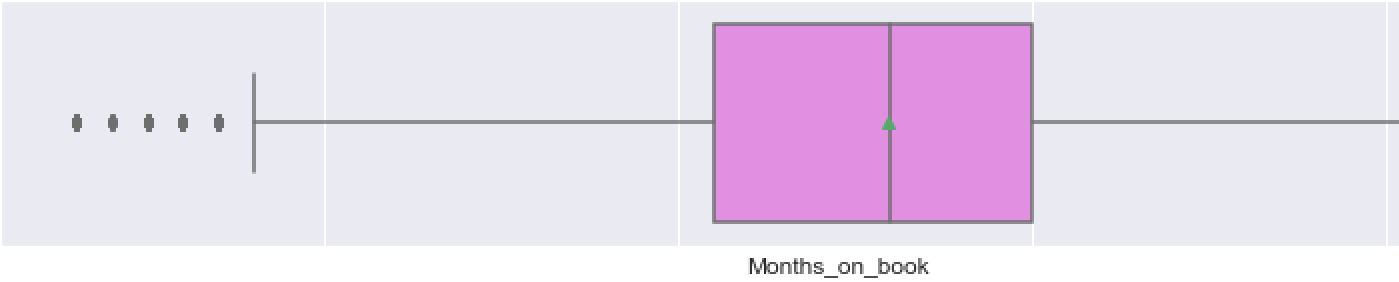


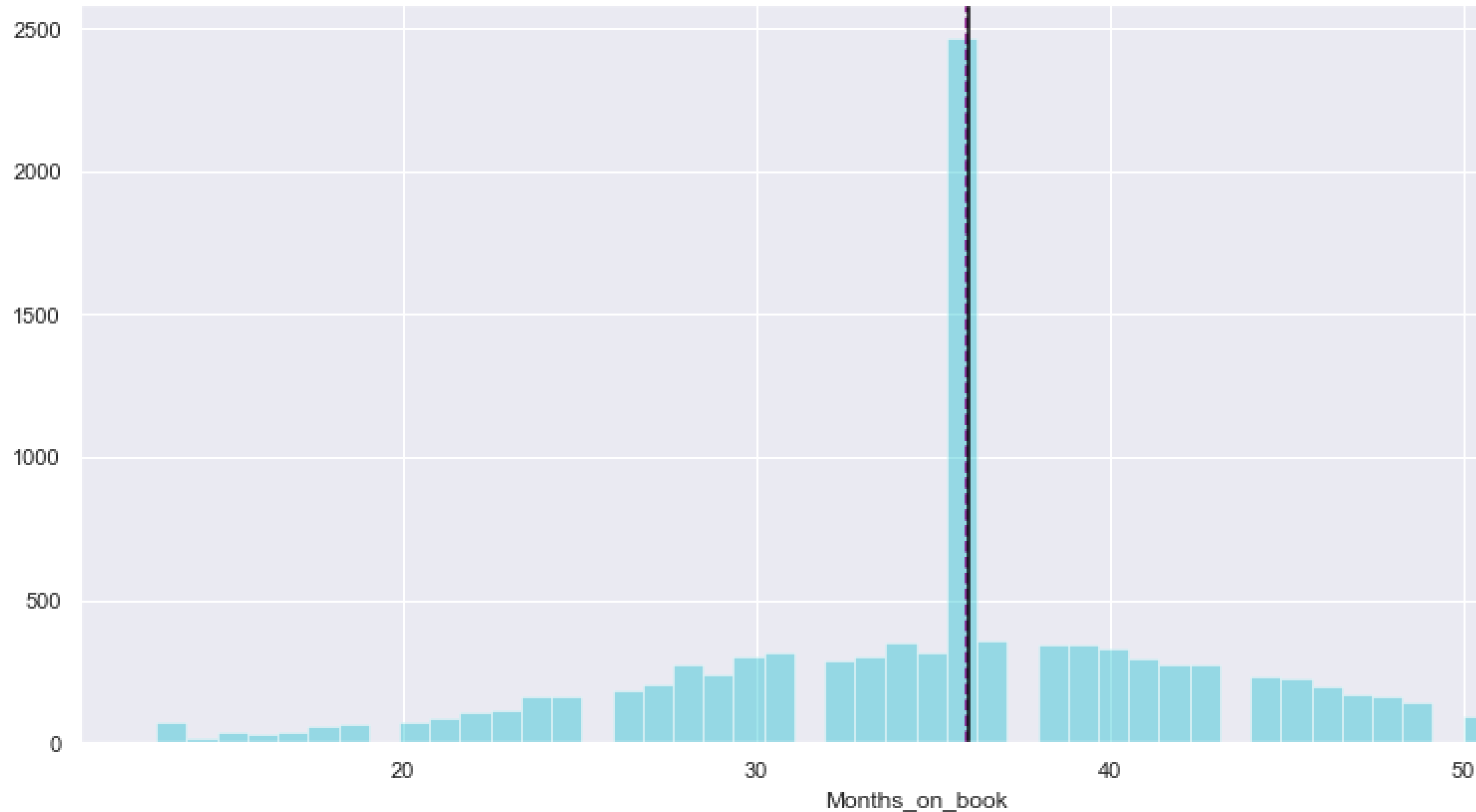
- Average Crdit limit of a customer is around $8,500
- Credit limit is right skewed.

Observations on Months on Book¶

In [353]:

```
histogram_boxplot(data['Months_on_book'])
```





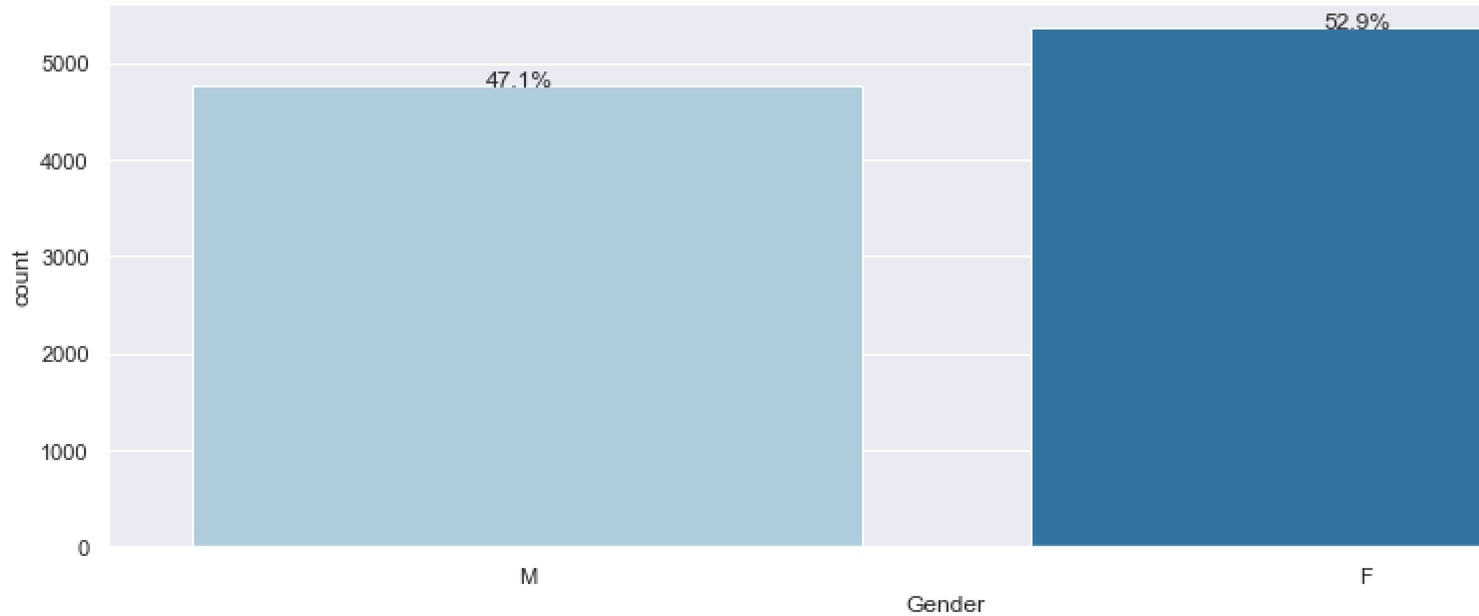
- Average months a customer holds a card is 35 months.
- Ohterwise it is distributed well over the other months.

In [354]:
```
# Function to create barplots that indicate percentage for each category.

def perc_on_bar(z):
    '''
    plot
    feature: categorical feature
    the function won't work if a column is passed in hue parameter
    '''

    total = len(data[z]) # length of the column
    plt.figure(figsize=(15,5))
    ax = sns.countplot(data[z],palette='Paired')
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total) # percentage of each
class of the category
        x = p.get_x() + p.get_width() / 2 - 0.05 # width of the plot
```

```
        y = p.get_y() + p.get_height()           # hieght of the plot

        ax.annotate(percentage, (x, y), size = 12) # annotate the percantage
    plt.show() # show the plot
```

In [355]:

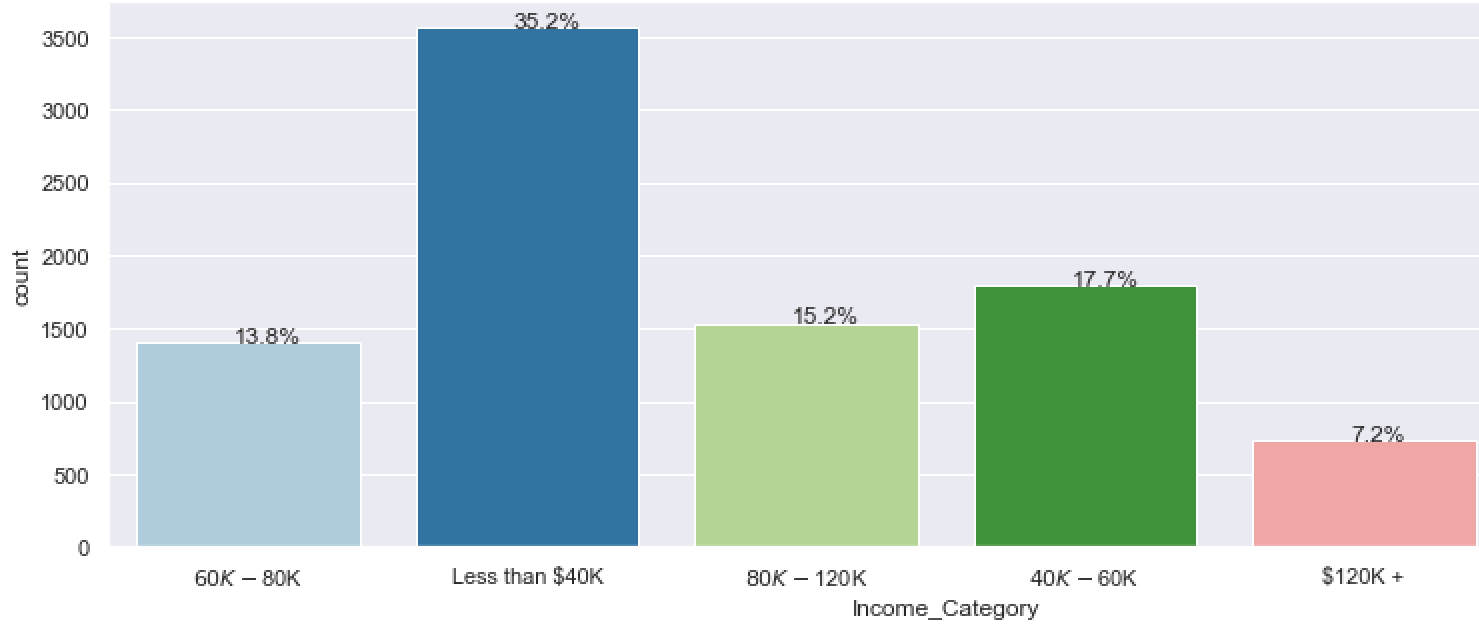
```
perc_on_bar('Gender')
```



- more Female than male, but very close

In [356]:

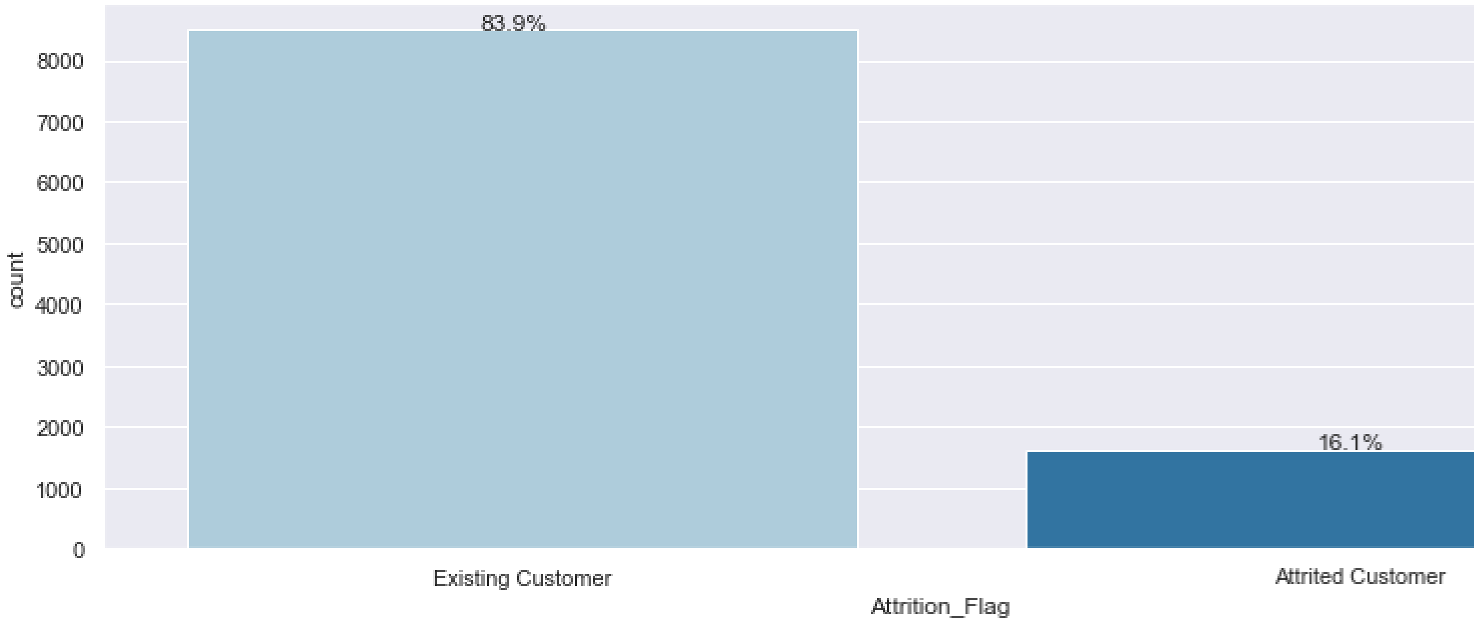
```
perc_on_bar('Income_Category')
```



Most customers make less than $40K a year, with second going to 40-60K

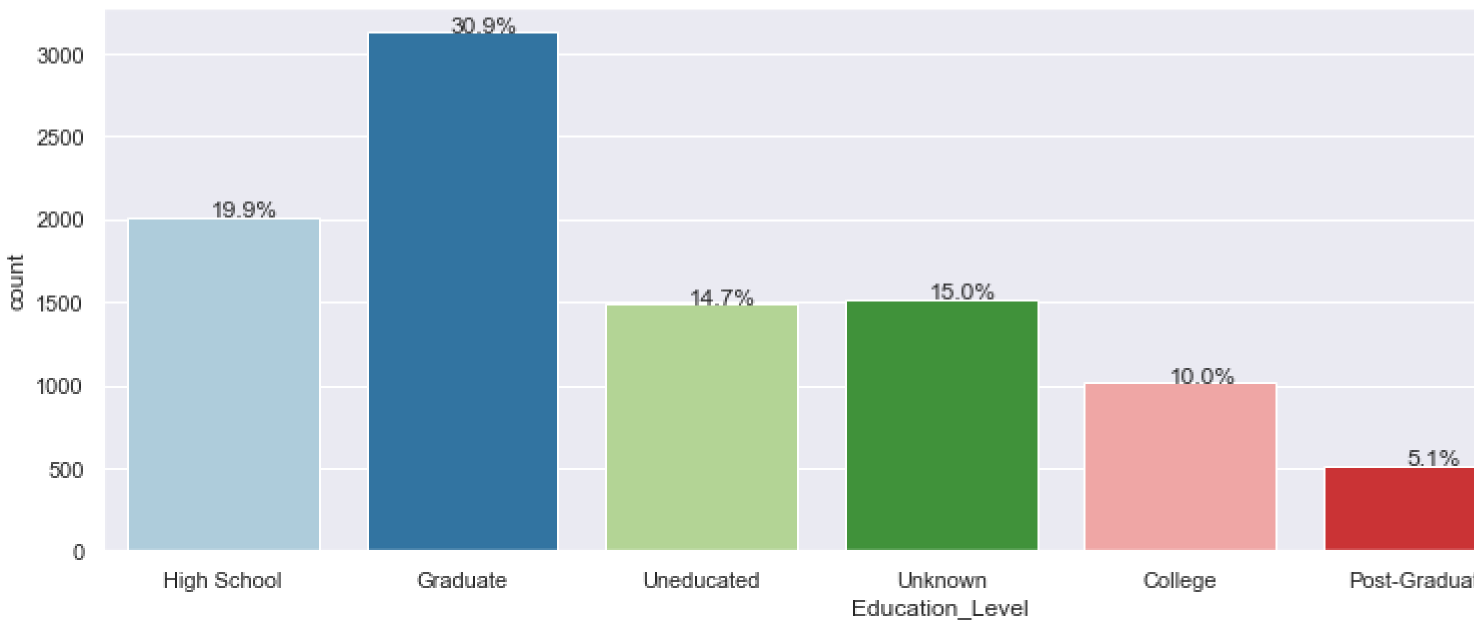
In [357]:

```
perc_on_bar('Attrition_Flag')
```



- 84% are existing customers

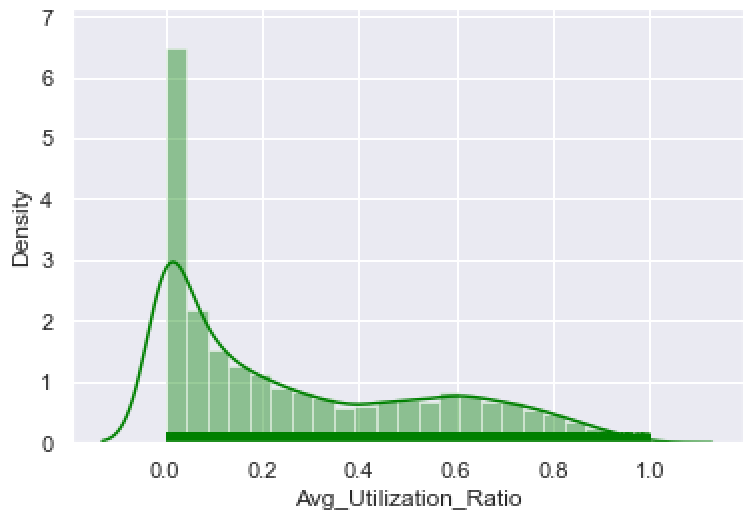
In [358]:
```
perc_on_bar('Education_Level')
```



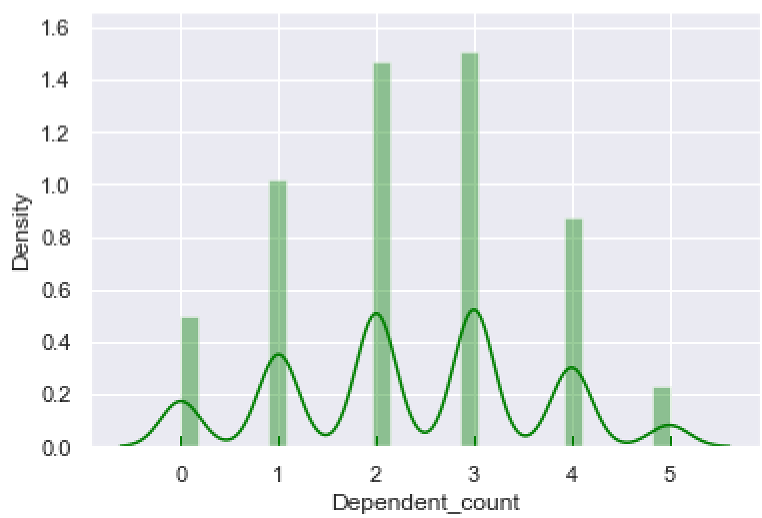
- 31% of customers hold graduate degrees, second high school only.
- This is a strange insight to me as a person with a graduate degree, would make more that $40K a year.

In [359]:
```
sns.distplot(df['Avg_Utilization_Ratio'], kde=True, rug=True, color='green')
```

Out[359]:
```
<AxesSubplot:xlabel='Avg_Utilization_Ratio', ylabel='Density'>
```



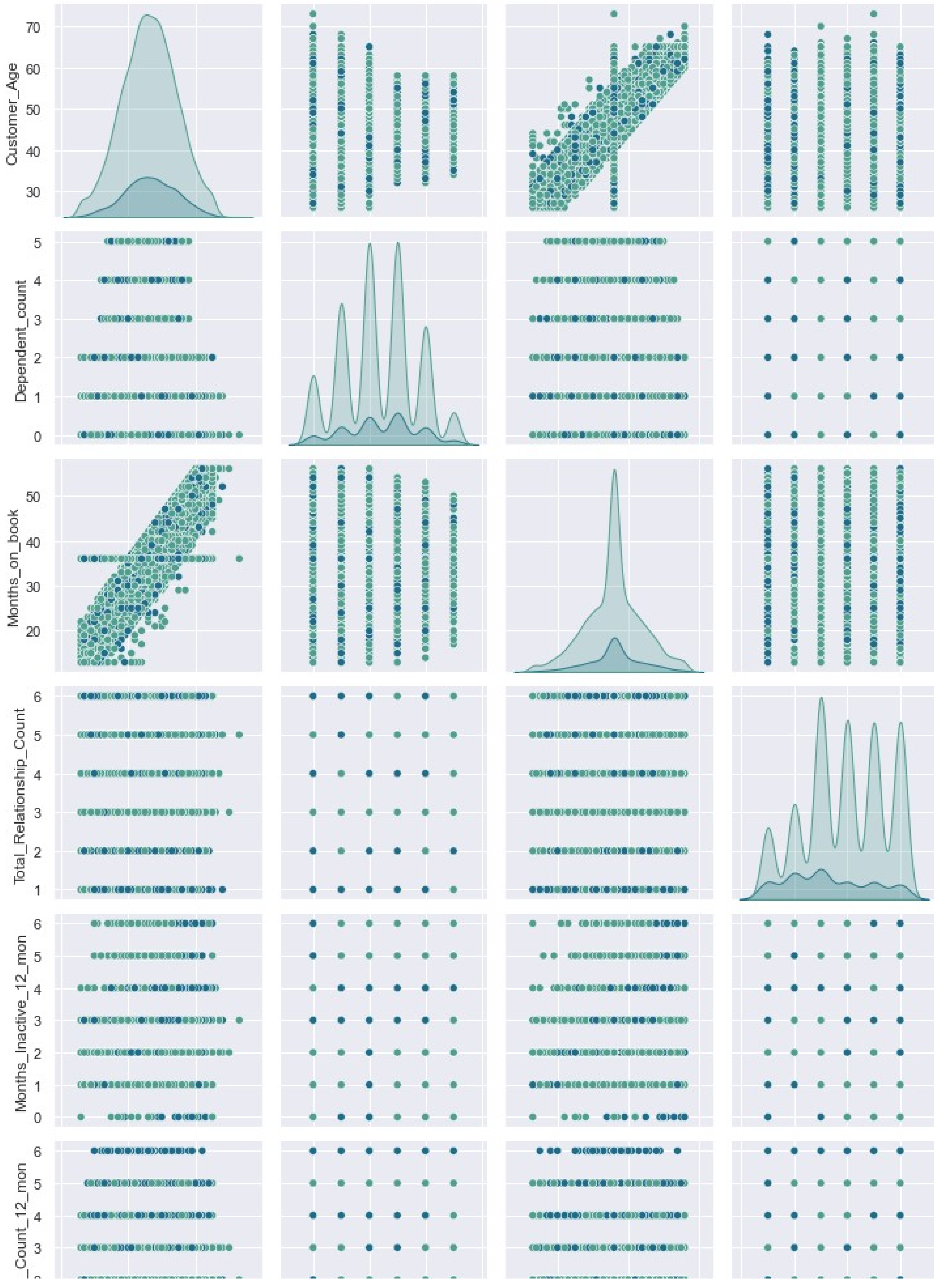
- Utilization is right skewed, with a large amount of customers hardly using thier credit, if at all.

In [360]:
```
sns.distplot(df['Dependent_count'], kde=True, rug=True,color='green')
```
Out[360]:
```
<AxesSubplot:xlabel='Dependent_count', ylabel='Density'>
```


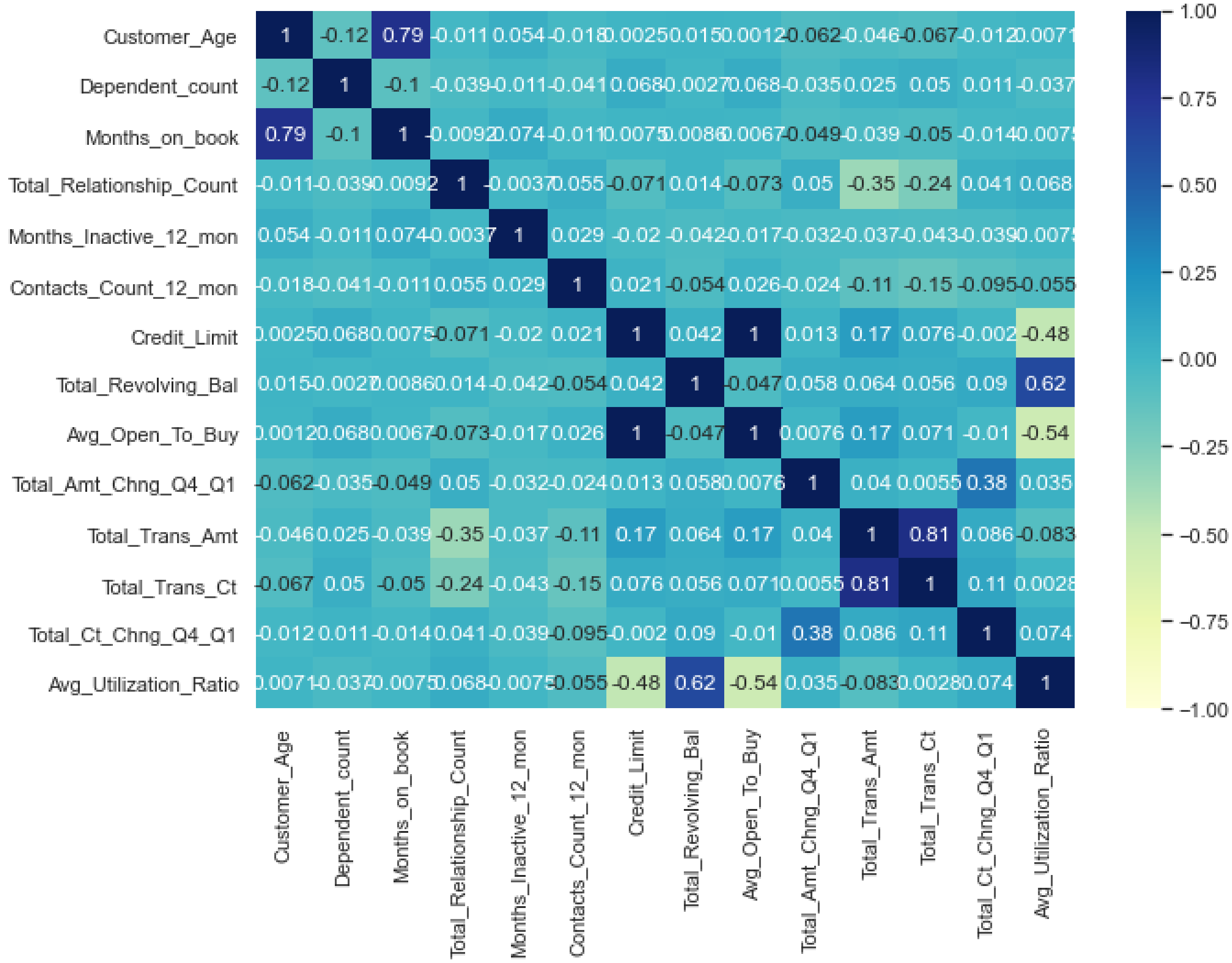
- Customers have 2 to 3 dependents

Bivariate Analysis¶

In [361]:
```
sns.pairplot(df,hue='Attrition_Flag',palette='crest')
```
Out[361]:
```
<seaborn.axisgrid.PairGrid at 0x1f8329cf400>
```


Customer_Age
Dependent_count
Months_on_book
Total_Relationship_Count
Months_Inactive_12_mon
_Count_12_mon

In [362]:
```
plt.figure(figsize=(10,7))
sns.heatmap(df.corr(),annot=True,vmin=-1,vmax=1,fmt='.2g', cmap="YlGnBu")
plt.show()
```



- There is a 1:1 ration for Average Open and Credit Limit, I will drop one of those.
- Months on Books is highly corrolated with Age.
- Transaction amount and transaction count are also correlated.

In [363]:
```
df.drop(['Avg_Open_To_Buy'],axis=1,inplace=True)
```

In [364]:
```
# creating instance of labelencoder
labelencoder = LabelEncoder()
# Assigning numerical values and storing in another column
encodingList = ['Attrition_Flag',
'Gender','Education_Level','Marital_Status','Income_Category','Card_Category']
for i in encodingList:
    df[i] = labelencoder.fit_transform(df[i])
```

In [365]:
```
df['Attrition_Flag'].replace(1,'Existing Customer',inplace=True)
df['Attrition_Flag'].replace(0,'Attrited Customer',inplace=True)
```

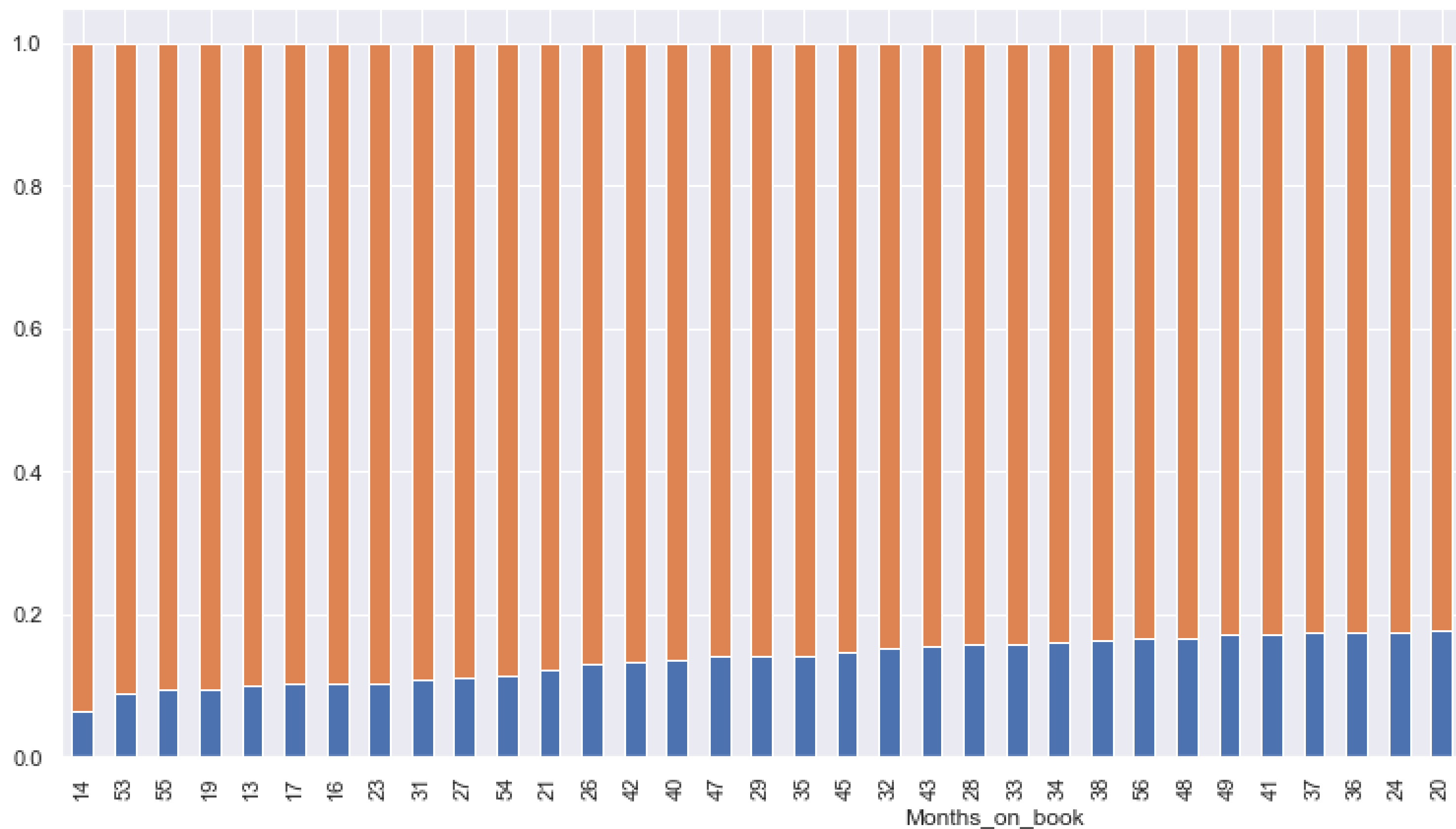
In [366]:
```
### Function to plot stacked bar charts for categorical columns
```

```
def stacked_plot(x):

    sns.set()
    ## crosstab
    tab1 = pd.crosstab(x,df['Attrition_Flag'],margins=True).sort_values(by='Existing
Customer',ascending=False)
    print(tab1)
    print('-'*120)
    ## visualising the cross tab
    tab =
pd.crosstab(x,df['Attrition_Flag'],normalize='index').sort_values(by='Existing
Customer',ascending=False)
    tab.plot(kind='bar',stacked=True,figsize=(17,7))
    plt.legend(loc='lower left', frameon=False)
    plt.legend(loc="upper left", bbox_to_anchor=(1,1))
    plt.show()
```

In [367]:

```
stacked_plot(df['Months_on_book'])
Attrition_Flag  Attrited Customer  Existing Customer    All
Months_on_book
All                          1627               8500  10127
36                            430               2033   2463
37                             62                296    358
34                             57                296    353
38                             57                290    347
40                             45                288    333
31                             34                284    318
39                             64                277    341
35                             45                272    317
33                             48                257    305
41                             51                246    297
32                             44                245    289
30                             58                242    300
42                             36                235    271
28                             43                232    275
43                             42                231    273
29                             34                207    241
45                             33                194    227
44                             42                188    230
27                             23                183    206
26                             24                162    186
46                             36                161    197
47                             24                147    171
48                             27                135    162
25                             31                134    165
24                             28                132    160
49                             24                117    141
23                             12                104    116
56                             17                 86    103
22                             20                 85    105
21                             10                 73     83
50                             25                 71     96
```

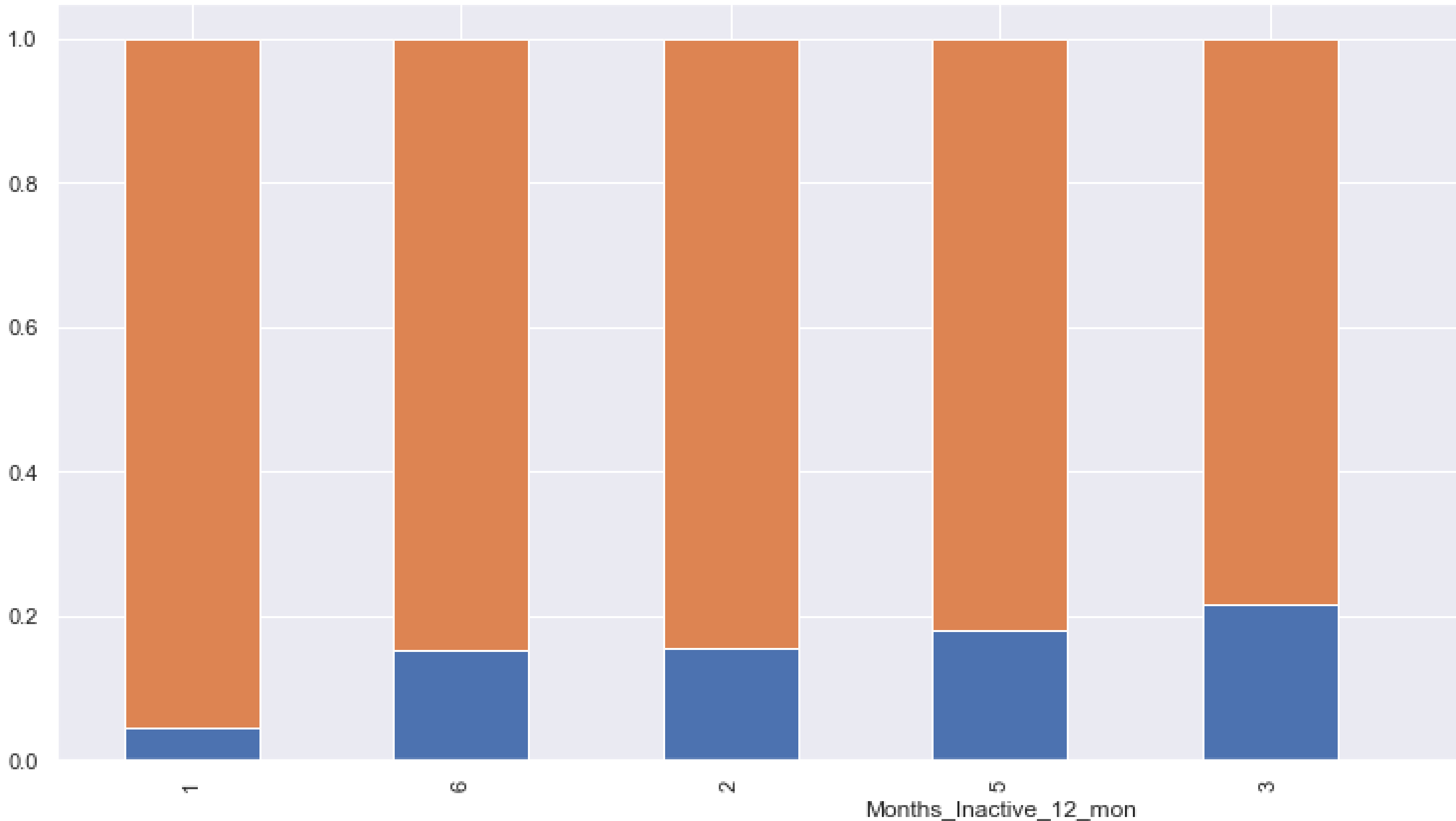
```
53                               7                71     78
51                              16                64     80
13                               7                63     70
20                              13                61     74
19                               6                57     63
52                              12                50     62
54                               6                47     53
18                              13                45     58
55                               4                38     42
17                               4                35     39
16                               3                26     29
15                               9                25     34
14                               1                15     16
------------------------------------------------------------------------------------------
----------------------------------
```

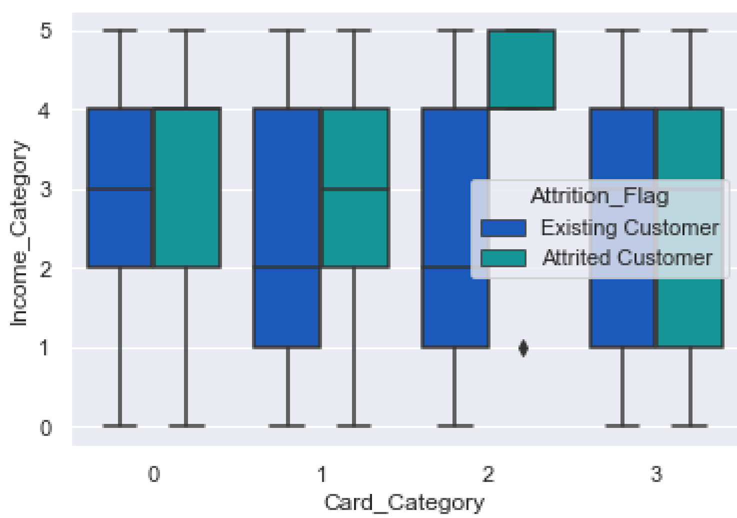


In [368]:

```
stacked_plot(df['Months_Inactive_12_mon'])
```

```
Attrition_Flag          Attrited Customer  Existing Customer    All
Months_Inactive_12_mon
All                                  1627               8500  10127
3                                     826               3020   3846
2                                     505               2777   3282
1                                     100               2133   2233
4                                     130                305    435
5                                      32                146    178
6                                      19                105    124
0                                      15                 14     29
------------------------------------------------------------------------------------------
----------------------------------
```



In [369]:
```
sns.boxplot(x='Card_Category', y='Income_Category', hue='Attrition_Flag', data = df,
palette='winter')
```
Out[369]:
```
<AxesSubplot:xlabel='Card_Category', ylabel='Income_Category'>
```


In [370]:
```
df.groupby(df['Attrition_Flag']).mean()
```
Out[370]:

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Ca egory
Attrition_Flag							
Attrited Customer	46.659496	0.428396	2.402581	3.119852	1.494776	2.924401	0.17025
Existing Customer	46.262118	0.479059	2.335412	3.092118	1.457412	2.852353	0.18164

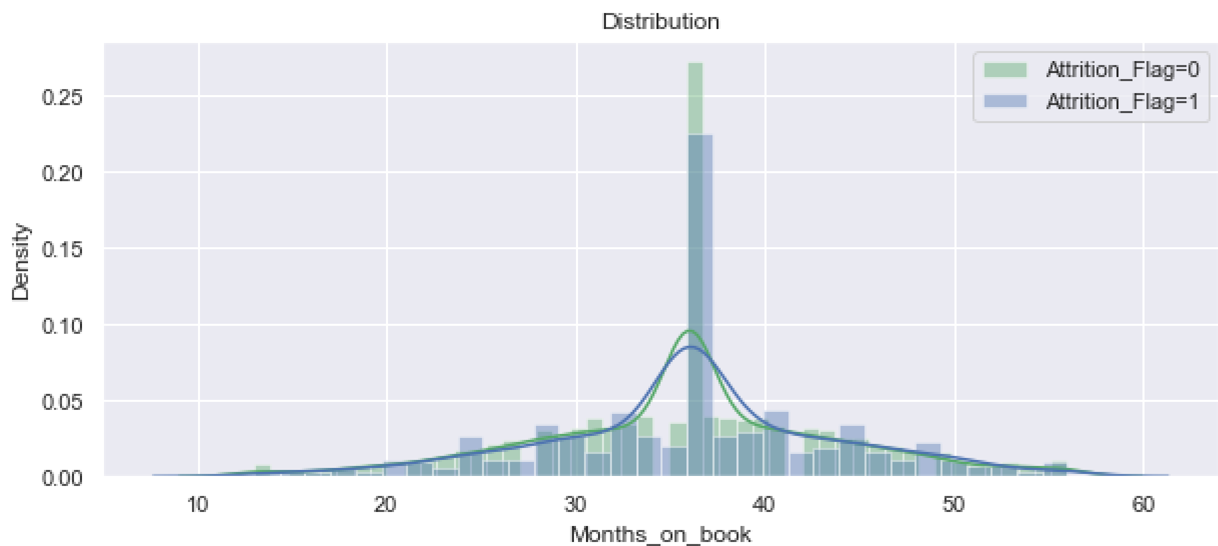
- People who spend more, have a larger revolving balance, and have more transactions tend to keep thier account.

In [375]:

```
plt.figure(figsize=(10,4))
sns.distplot(df[df["Attrition_Flag"] == 'Existing Customer']['Months_on_book'], color = 'g',label='Attrition_Flag=0')
sns.distplot(df[df["Attrition_Flag"] == 'Attrited Customer']['Months_on_book'], color = 'b',label='Attrition_Flag=1')
plt.legend()
plt.title("Distribution")
```

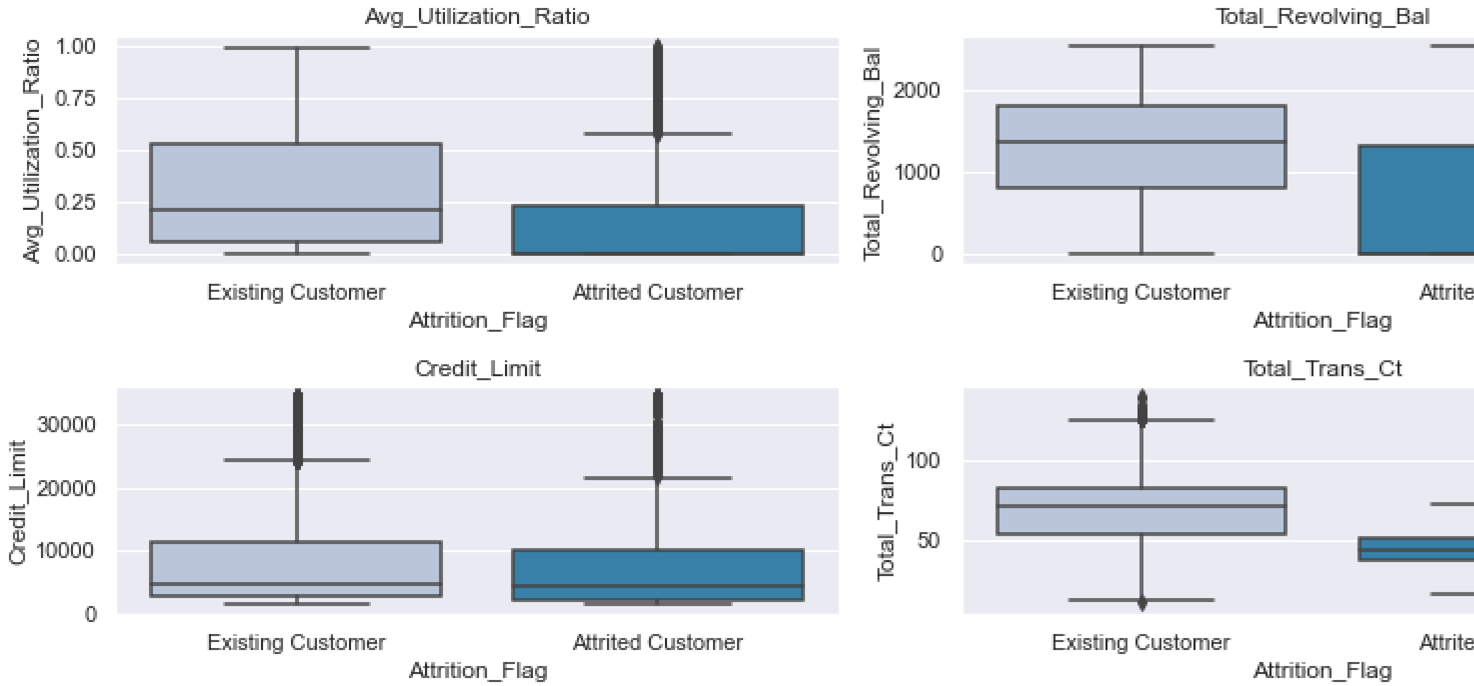
Out[375]:

```
Text(0.5, 1.0, 'Distribution')
```



In [378]:

```
cols = data[['Avg_Utilization_Ratio','Total_Revolving_Bal','Credit_Limit', 'Total_Trans_Ct']].columns.tolist()
plt.figure(figsize=(12,7))

for i, variable in enumerate(cols):
                     plt.subplot(3,2,i+1)
                     sns.boxplot(data["Attrition_Flag"],data[variable],palette="PuBu")
                     plt.tight_layout()
                     plt.title(variable)
plt.show()
```



In [379]:

```
# creating instance of labelencoder
labelencoder = LabelEncoder()
# Assigning numerical values and storing in another column
encodingList = ['Attrition_Flag']
for i in encodingList:
    df[i] = labelencoder.fit_transform(df[i])
```

In [380]:

```
columns = df.columns
```

In [381]:

```
len(df.columns)
```

Out[381]:

```
19
```

In [382]:

```
df
```

Out[382]:

	Attrition_ Flag	Customer_ Age	Gender	Dependent_ count	Education_ Level	Marital_ Status	Income_C ategory	Ca e
0	1	45	1	3	3	1	2	0
1	1	49	0	5	2	2	4	0
2	1	51	1	3	2	1	3	0
3	1	40	0	4	3	3	4	0
4	1	40	1	3	5	1	2	0

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Ca e
...	...	...	...	...	...	...	...	...
10122	1	50	1	2	2	2	1	0
10123	0	41	1	2	6	0	1	0
10124	0	44	0	1	3	1	4	0
10125	0	30	1	2	2	3	1	0
10126	0	43	0	2	2	1	4	3

10127 rows × 19 columns

In [383]:

```
# Create correlation matrix
corr_matrix = df.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Find features with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]

# Drop features
df.drop(to_drop, axis=1, inplace=True)
```

In [384]:

```
df
```

Out[384]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Ca e
0	1	45	1	3	3	1	2	0
1	1	49	0	5	2	2	4	0
2	1	51	1	3	2	1	3	0
3	1	40	0	4	3	3	4	0
4	1	40	1	3	5	1	2	0

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Ca e
...	...	...	...	...	...	...	...	...
10122	1	50	1	2	2	2	1	0
10123	0	41	1	2	6	0	1	0
10124	0	44	0	1	3	1	4	0
10125	0	30	1	2	2	3	1	0
10126	0	43	0	2	2	1	4	3

10127 rows × 19 columns

Removing Outliers¶

In [385]:

```
from scipy import stats
df = df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]
```

In [386]:

```
df
```

Out[386]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Ca e
5	1	44	1	2	2	1	1	0
10	1	42	1	5	5	3	0	0
14	1	57	0	2	2	1	4	0
19	1	45	0	2	2	1	5	0
20	1	47	1	1	1	0	2	0
...	...	...	...	...	...	...	...	...
10118	0	50	1	1	6	3	3	0

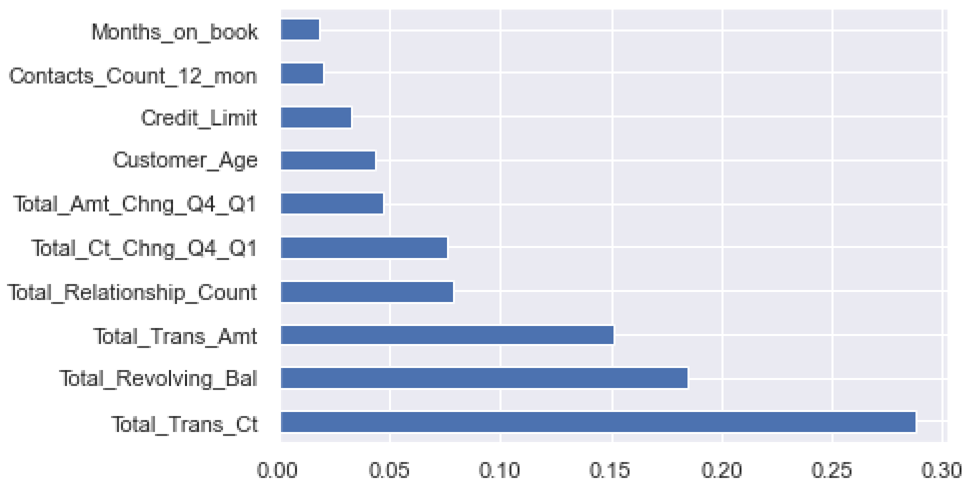
	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Ca e
10119	0	55	0	3	5	2	5	0
10123	0	41	1	2	6	0	1	0
10124	0	44	0	1	3	1	4	0
10125	0	30	1	2	2	3	1	0

8842 rows × 19 columns

In [387]:
```
from sklearn.model_selection import train_test_split
```
In [388]:
```
columns
```
Out[388]:
```
Index(['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',
       'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt', 'Total_Trans_Ct',
       'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio'],
      dtype='object')
```
In [389]:
```
X = df[['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',
       'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt', 'Total_Trans_Ct',
       'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio']]
y = df['Attrition_Flag']
```
In [390]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)
```
Logistic Regression¶

In [391]:
```
from sklearn.linear_model import LogisticRegression
```
In [392]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```
Out[392]:
```
LogisticRegression()
```
Predictions and Evaluations¶

In [393]:
```
predictions = logmodel.predict(X_test)
```
In [394]:
```
from sklearn.metrics import accuracy_score
```

In [395]:
```
print(accuracy_score(y_test,predictions))
```
```
0.8810829335161069
```
Decision Tree¶
In [396]:
```
from sklearn.tree import DecisionTreeClassifier
```
In [397]:
```
dtree = DecisionTreeClassifier()
```
In [398]:
```
dtree.fit(X_train,y_train)
```
Out[398]:
```
DecisionTreeClassifier()
```
Predictions and Evaluations¶
In [399]:
```
predictions = dtree.predict(X_test)
```
In [400]:
```
print(accuracy_score(y_test,predictions))
```
```
0.9372858122001371
```
In [401]:
```
feat_importances = pd.Series(dtree.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
```
Out[401]:
```
<AxesSubplot:>
```


Random Forest¶
In [402]:
```
from sklearn.ensemble import RandomForestClassifier
```
In [403]:
```
rfc = RandomForestClassifier(n_estimators=600)
```
In [404]:
```
rfc.fit(X_train,y_train)
```
Out[404]:
```
RandomForestClassifier(n_estimators=600)
```
Predictions and Evaluations¶
In [405]:
```
predictions = rfc.predict(X_test)
```
In [406]:
```
print(accuracy_score(y_test,predictions))
```

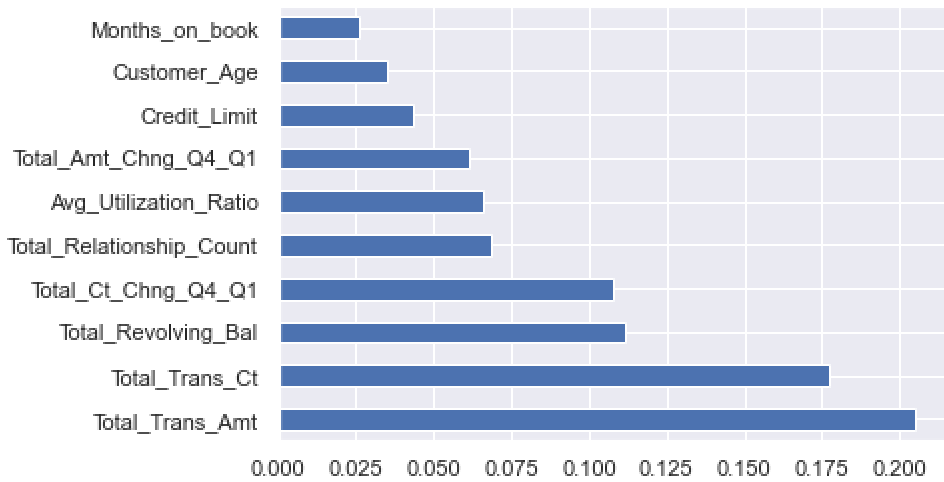
```
0.9561343385880741
```

In [407]:

```
feat_importances = pd.Series(rfc.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
```

Out[407]:

```
<AxesSubplot:>
```



Bagging Classifier¶

In [409]:

```
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
import pandas as pd

seed = 8
kfold = model_selection.KFold(n_splits = 5)
```

In [410]:

```
# initialize the base classifier
base_cls = DecisionTreeClassifier()

# no. of base classifier
num_trees = 100
```

In [411]:

```
# bagging classifier
model = BaggingClassifier(base_estimator = base_cls,
                          n_estimators = num_trees,
                          random_state = seed)
```

In [412]:

```
results = model_selection.cross_val_score(model, X, y, cv = kfold)
print("accuracy :")
print(results.mean())
accuracy :
0.9131444254877235
```

Gradient Boosting Classifier¶

In [413]:

```
# Import models and utility functions
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import mean_squared_error as MSE
from sklearn import datasets

# Setting SEED for reproducibility
seed = 1

# Instantiate Gradient Boosting Regressor
gbr = GradientBoostingClassifier(n_estimators = 200, max_depth = 1, random_state =
seed)

# Fit to training set
gbr.fit(X_train, y_train)

# Predict on test set
predictions = gbr.predict(X_test)

# test set RMSE
test_rmse = MSE(y_test, predictions) ** (1 / 2)

# Print rmse
print('RMSE test set: {:.2f}'.format(test_rmse))
RMSE test set: 0.26
```

In [414]:

```
print(accuracy_score(y_test,predictions))
0.933173406442769
```

XGBoost Classifier¶

In [415]:

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [416]:

```
encoder = LabelEncoder()
binary_encoded_y = pd.Series(encoder.fit_transform(y))
```

In [417]:

```
X_train, X_test, y_train, y_test = train_test_split(X, binary_encoded_y,
random_state=42)
```

In [418]:

```
classifier = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1),
    n_estimators=200
)
classifier.fit(X_train,y_train)
```

Out[418]:

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   n_estimators=200)
```

In [419]:

```
predictions = classifier.predict(X_test)
```

In [420]:

```
print(accuracy_score(y_test,predictions))
0.95838986883763
```

Grid Search¶

In [421]:

```
#Grid Search
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
clf = LogisticRegression()
grid_values = {'penalty': ['l1', 'l2'],'C':[0.001,.009,0.01,.09,1,5,10,25]}
grid_clf_acc = GridSearchCV(clf, param_grid = grid_values,scoring = 'recall')
grid_clf_acc.fit(X_train, y_train)

#Predict values based on new parameters
predictions = grid_clf_acc.predict(X_test)

# New Model Evaluation metrics
print('Accuracy Score : ' + str(accuracy_score(y_test,predictions)))
```

```
Accuracy Score : 0.8819538670284939
```

In [425]:

```
pip install -U imbalanced-learn
```

```
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.8.0-py3-none-any.whl (206 kB)
Requirement already satisfied: joblib>=0.11 in c:\users\slhil\anaconda3\lib\site-packages (from imbalanced-learn) (1.0.1)
Requirement already satisfied: numpy>=1.13.3 in c:\users\slhil\anaconda3\lib\site-packages (from imbalanced-learn) (1.19.2)
Requirement already satisfied: scikit-learn>=0.24 in c:\users\slhil\anaconda3\lib\site-packages (from imbalanced-learn) (0.24.1)
Requirement already satisfied: scipy>=0.19.1 in c:\users\slhil\anaconda3\lib\site-packages (from imbalanced-learn) (1.6.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\slhil\anaconda3\lib\site-packages (from scikit-learn>=0.24->imbalanced-learn) (2.1.0)
Installing collected packages: imbalanced-learn
Successfully installed imbalanced-learn-0.8.0
Note: you may need to restart the kernel to use updated packages.
```

SMOTE¶

In [426]:

```
from imblearn.over_sampling import SMOTE
```

In [427]:

```
print("Before UpSampling, counts of label 'Yes': {}".format(sum(y_train==1)))
print("Before UpSampling, counts of label 'No': {} \n".format(sum(y_train==0)))

sm = SMOTE(sampling_strategy = 1 ,k_neighbors = 5, random_state=1)   #Synthetic Minority Over Sampling Technique
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)


print("After UpSampling, counts of label 'Yes': {}".format(sum(y_train_over==1)))
print("After UpSampling, counts of label 'No': {} \n".format(sum(y_train_over==0)))


print('After UpSampling, the shape of train_X: {}'.format(X_train_over.shape))
```

```
print('After UpSampling, the shape of train_y: {} \n'.format(y_train_over.shape))
Before UpSampling, counts of label 'Yes': 5548
Before UpSampling, counts of label 'No': 1083

After UpSampling, counts of label 'Yes': 5548
After UpSampling, counts of label 'No': 5548

After UpSampling, the shape of train_X: (11096, 18)
After UpSampling, the shape of train_y: (11096,)
```

Logistic Regression on SMOTE over sampled data¶

In [428]:

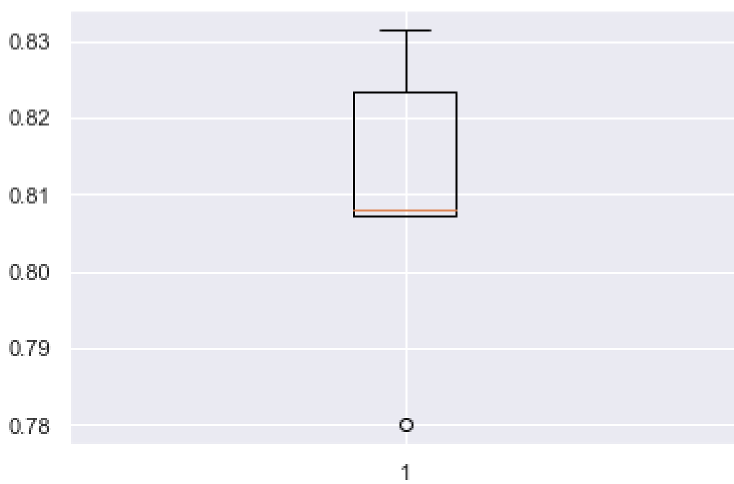
```
log_reg_over = LogisticRegression(random_state = 1)

# Training the basic logistic regression model with training set
log_reg_over.fit(X_train_over,y_train_over)
```

Out[428]:

```
LogisticRegression(random_state=1)
```

In [429]:

```
scoring='recall'
kfold=StratifiedKFold(n_splits=5,shuffle=True,random_state=1)     #Setting number of splits equal to 5
cv_result_over=cross_val_score(estimator=log_reg_over, X=X_train_over, y=y_train_over, scoring=scoring, cv=kfold)
#Plotting boxplots for CV scores of model defined above
plt.boxplot(cv_result_over)
plt.show()
```



- Performance of model on training set varies between 0.80 to 0.83, which is not an improvement from the previous model
- Let's check the performance on the test set.

In [431]:

```
##  Function to calculate different metric scores of the model - Accuracy, Recall and Precision
def get_metrics_score(model,train,test,train_y,test_y,flag=True):
    '''
    model : classifier to predict values of X
```

```
    '''
    # defining an empty list to store train and test results
    score_list=[]

    pred_train = model.predict(train)
    pred_test = model.predict(test)

    train_acc = model.score(train,train_y)
    test_acc = model.score(test,test_y)

    train_recall = metrics.recall_score(train_y,pred_train)
    test_recall = metrics.recall_score(test_y,pred_test)

    train_precision = metrics.precision_score(train_y,pred_train)
    test_precision = metrics.precision_score(test_y,pred_test)


score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,test_pre
cision))

    # If the flag is set to True then only the following print statements will be
dispayed. The default value is set to True.
    if flag == True:
        print("Accuracy on training set : ",model.score(train,train_y))
        print("Accuracy on test set : ",model.score(test,test_y))
        print("Recall on training set : ",metrics.recall_score(train_y,pred_train))
        print("Recall on test set : ",metrics.recall_score(test_y,pred_test))
        print("Precision on training set :
",metrics.precision_score(train_y,pred_train))
        print("Precision on test set : ",metrics.precision_score(test_y,pred_test))

    return score_list # returning the list with train and test sco
```

In [433]:

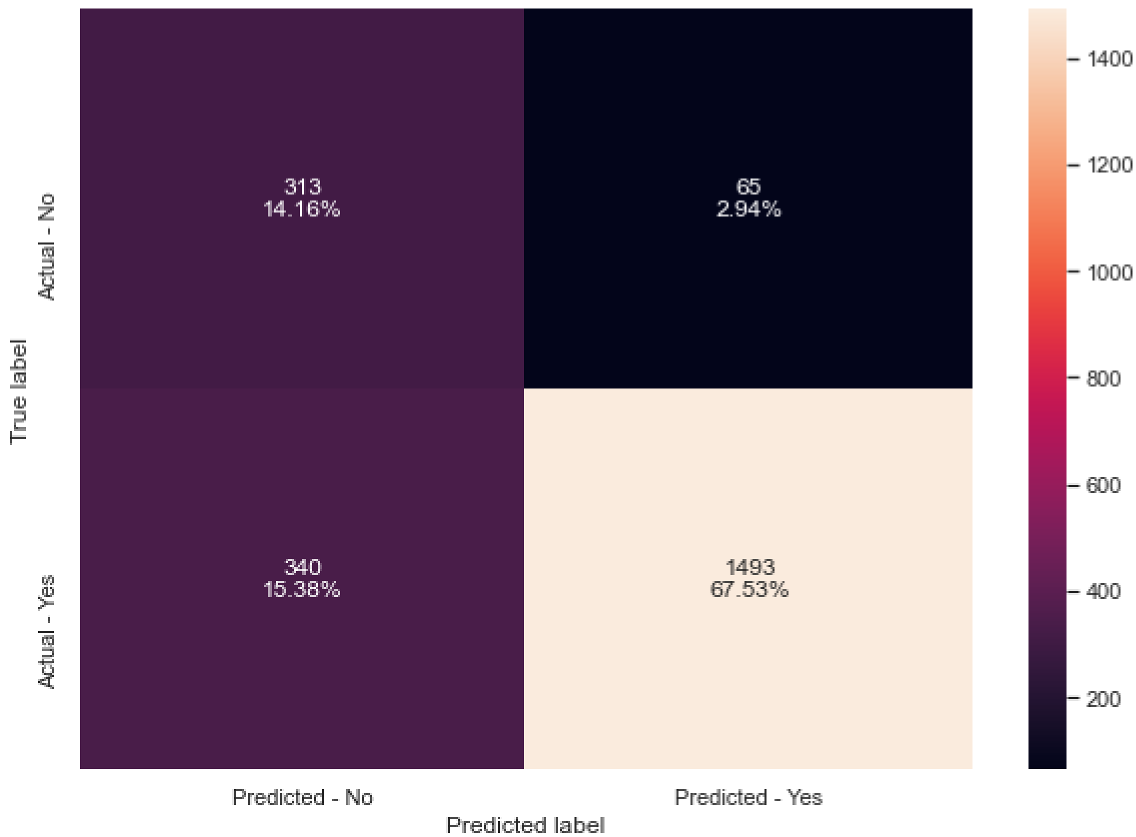
```
def make_confusion_matrix(model,y_actual,labels=[1, 0]):
    '''
    model : classifier to predict values of X
    y_actual : ground truth

    '''
    y_predict = model.predict(X_test)
    cm=metrics.confusion_matrix( y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Actual - No","Actual - Yes"]],
                  columns = [i for i in ['Predicted - No','Predicted - Yes']])
    group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
```

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

In [434]:

```
#Calculating different metrics
get_metrics_score(log_reg_over,X_train_over,X_test,y_train_over,y_test)

# creating confusion matrix
make_confusion_matrix(log_reg_over,y_test)
```

```
Accuracy on training set :  0.8166906993511175
Accuracy on test set :  0.8168249660786974
Recall on training set :  0.8190338860850757
Recall on test set :  0.8145117294053464
Precision on training set :  0.8152134912091855
Precision on test set :  0.9582798459563543
```



Logistic Regression on undersampled data¶

In [435]:

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state = 1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

In [436]:

```
print("Before Under Sampling, counts of label 'Yes': {}".format(sum(y_train==1)))
print("Before Under Sampling, counts of label 'No': {} \n".format(sum(y_train==0)))

print("After Under Sampling, counts of label 'Yes': {}".format(sum(y_train_un==1)))
print("After Under Sampling, counts of label 'No': {} \n".format(sum(y_train_un==0)))

print('After Under Sampling, the shape of train_X: {}'.format(X_train_un.shape))
print('After Under Sampling, the shape of train_y: {} \n'.format(y_train_un.shape))
```

```
Before Under Sampling, counts of label 'Yes': 5548
Before Under Sampling, counts of label 'No': 1083

After Under Sampling, counts of label 'Yes': 1083
After Under Sampling, counts of label 'No': 1083

After Under Sampling, the shape of train_X: (2166, 18)
After Under Sampling, the shape of train_y: (2166,)
```

In [437]:

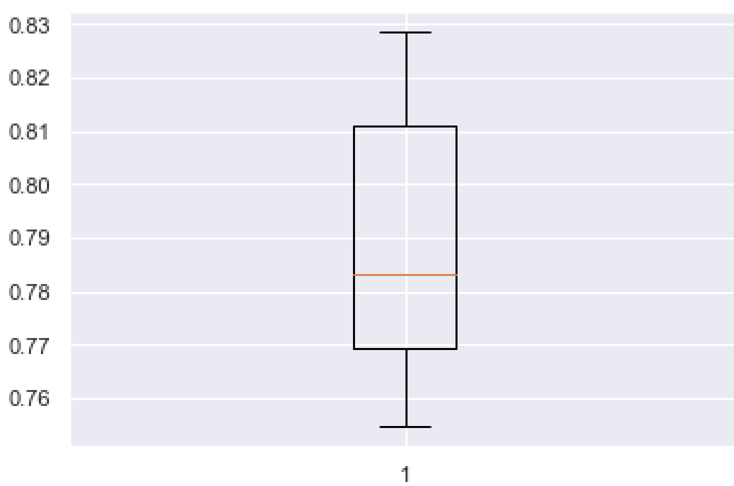
```
log_reg_under = LogisticRegression(random_state = 1)
log_reg_under.fit(X_train_un,y_train_un )
```

Out[437]:

```
LogisticRegression(random_state=1)
```

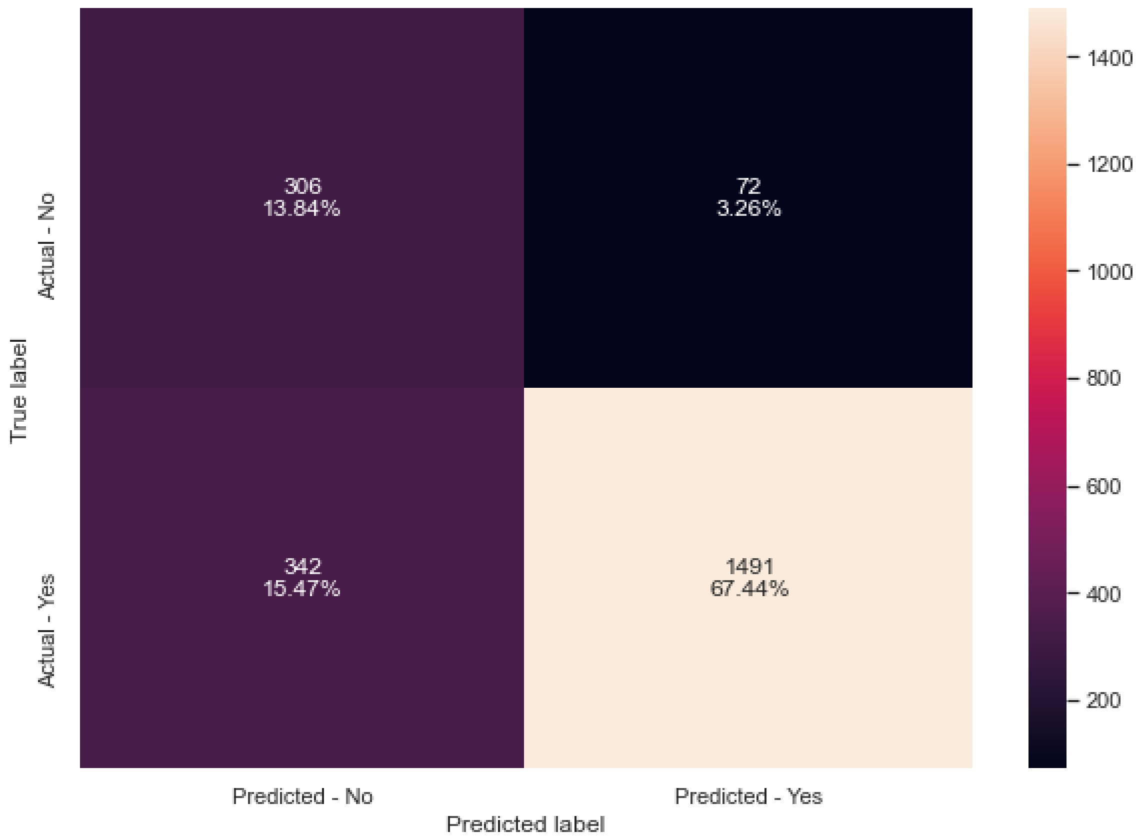
In [438]:

```
scoring='recall'
kfold=StratifiedKFold(n_splits=5,shuffle=True,random_state=1)     #Setting number of splits equal to 5
cv_result_under=cross_val_score(estimator=log_reg_under, X=X_train_un, y=y_train_un, scoring=scoring, cv=kfold)
#Plotting boxplots for CV scores of model defined above
plt.boxplot(cv_result_under)
plt.show()
```



In [439]:

```
#Calculating different metrics
get_metrics_score(log_reg_under,X_train_un,X_test,y_train_un,y_test)


# creating confusion matrix
make_confusion_matrix(log_reg_under,y_test)
Accuracy on training set :  0.8051708217913204
Accuracy on test set :  0.8127544097693351
Recall on training set :  0.7848568790397045
Recall on test set :  0.8134206219312602
Precision on training set :  0.8180943214629451
Precision on test set :  0.9539347408829175
```



In [441]:
```
lr = LogisticRegression(random_state=1)
lr.fit(X_train,y_train)
```
Out[441]:
```
LogisticRegression(random_state=1)
```
In [444]:
```
# Choose the type of classifier.
lr_estimator = LogisticRegression(random_state=1,solver='saga')

# Grid of parameters to choose from
parameters = {'C': np.arange(0.1,1.1,0.1)}

# Run the grid search
grid_obj = GridSearchCV(lr_estimator, parameters, scoring='recall')
grid_obj = grid_obj.fit(X_train_over, y_train_over)

# Set the clf to the best combination of parameters
lr_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
lr_estimator.fit(X_train_over, y_train_over)
```
Out[444]:
```
LogisticRegression(C=0.1, random_state=1, solver='saga')
```
In [445]:
```
# defining list of model
models = [lr]

# defining empty lists to add train and test results
acc_train = []
```

```
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []

# looping through all the models to get the metrics score - Accuracy, Recall and 
Precision
for model in models:

    j = get_metrics_score(model,X_train,X_test,y_train,y_test,False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
```

In [446]:

```
# defining list of models
models = [log_reg_over, lr_estimator]

# looping through all the models to get the metrics score - Accuracy, Recall and 
Precision
for model in models:

    j = get_metrics_score(model,X_train_over,X_test,y_train_over,y_test,False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
```

In [447]:

```
# defining list of model
models = [log_reg_under]

# looping through all the models to get the metrics score - Accuracy, Recall and 
Precision
for model in models:

    j = get_metrics_score(model,X_train_un,X_test,y_train_un,y_test,False)
    acc_train.append(j[0])
    acc_test.append(j[1])
    recall_train.append(j[2])
    recall_test.append(j[3])
    precision_train.append(j[4])
    precision_test.append(j[5])
```

In [448]:

```
comparison_frame = pd.DataFrame({'Model':['Logistic Regression','Logistic Regression on 
Oversampled data',
                                          'Logistic Regression-Regularized (Oversampled 
data)','Logistic Regression on Undersampled data'],
```

```
                                          'Train_Accuracy': acc_train,'Test_Accuracy': acc_test,

'Train_Recall':recall_train,'Test_Recall':recall_test,

'Train_Precision':precision_train,'Test_Precision':precision_test})

#Sorting models in decreasing order of test recall
comparison_frame
```

Out[448]:

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision
0	Logistic Regression	0.888855	0.880597	0.963590	0.960720	0.909029	0.901690
1	Logistic Regression on Oversampled data	0.816691	0.816825	0.819034	0.814512	0.815213	0.958280
2	Logistic Regression-Regularized (Oversampled d...	0.702145	0.770240	0.811644	0.809602	0.665829	0.903226
3	Logistic Regression on Undersampled data	0.805171	0.812754	0.784857	0.813421	0.818094	0.953935

Finding Coefficents¶

In [449]:

```
log_odds = log_reg_under.coef_[0]
pd.DataFrame(log_odds, X_train_un.columns, columns=['coef']).T
```

Out[449]:

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Mo
coef	-0.084107	0.024118	-0.290675	-0.218574	-0.145912	-0.218576	-0.004135	0.0

Converting coefficents to Odds¶

In [450]:

```
odds = np.exp(np.abs(log_reg_under.coef_[0]))-1
pd.set_option('display.max_rows',None)
pd.DataFrame(odds, X_train_un.columns, columns=['Change in odds']).T
```

Out[450]:

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category
Change in odds	0.087745	0.024411	0.33733	0.244301	0.157095	0.244304	0.004144

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category

Conclusions and Business Insights¶

- Logistic Regression on oversampled data did the best in the test data.
- Top three factors that effect credit card attrition: Total Transaction Amount, Total Transaction Counts, and Total Revolving Balance. So, in short, those who keep thier account, use thier credit cards a lot.
- Business should focus on getting customers to use the credit they have more often. This seems to be the best predictor of keeping a customer.
- Over 30% of customers have graduate degrees and make less than $40K a year

Attrited Client Profile:¶

- Clients who are contacted a lot are 52% more likly to leave.
- Clients who are inactive during a 12 month period are 49% more likely to leave.
- Clients who have a smaller credit limit.
- Clients who have a 50% smaller revolving balance.

In [ ]:

In [ ]: