Credit Card Churn Model

Data dictionary

- CLIENTNUM - ID of the customer holding the credit card.
- Customer_Age - Age of the customer.
- Gender - Sex of the customer.
- Dependent_count - Number of dependents of the customer.
- Education_Level - Educational qualification of the customer.
- Marital_Status - Civil status of the customer.
- Income_Category - Annual income range of the customer.
- Card_Category - Type of card owned by the customer.
- Months_on_book - Number of months elapsed since the account opening.
- Total_Relationship_Count - Total number of products held by the customer.
- Months_Inactive_12_mon - Number of months with no transactions in the last year.
- Contacts_Count_12_mon - Number of contacts with the bank in the last year.
- Credit_Limit - Credit limit on the credit card.
- Total_Revolving_Bal - Total revolving balance on the credit card.
- Avg_Open_To_Buy - Average card "Open To Buy" (=credit limit - account balance) in the last year.
- Total_Amt_Chng_Q4_Q1 - Change in transaction amount over the last year (Q4 over Q1).
- Total_Trans_Amt - Total amount of transactions made in the last year.
- Total_Trans_Ct - Number of transactions made in the last year.
- Total_Ct_Chng_Q4_Q1 - Change in transaction number over the last year (Q4 over Q1).
- Avg_Utilization_Ratio - Average card "Utilization ratio" (=account balance / credit limit) in the last year.
- Attrition_Flag - Target variable. "Attrited Customer" if the customer closed their account, otherwise "Existing Customer".

Import library

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings("ignore")

        %matplotlib inline
```

Read data

```
In [2]: ds = pd.read_csv('BankChurners.csv')
        ds.head()
```

Out[2]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_
0	768805383	Existing Customer	45	M	3	High School	Married	$60K-$80K	Blue	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K-$120K	Blue	
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	
4	709106358	Existing Customer	40	M	3	Uneducated	Married	$60K-$80K	Blue	

5 rows × 23 columns

Data cleaning

In [3]:
```
ds = ds.iloc[:,1:-2]
ds
```

Out[3]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book
0	Existing Customer	45	M	3	High School	Married	$60K-$80K	Blue	39
1	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	44
2	Existing Customer	51	M	3	Graduate	Married	$80K-$120K	Blue	36
3	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	34
4	Existing Customer	40	M	3	Uneducated	Married	$60K-$80K	Blue	21
...	...	...	...	...	...	...	...	...	...
10122	Existing Customer	50	M	2	Graduate	Single	$40K-$60K	Blue	40
10123	Attrited Customer	41	M	2	Unknown	Divorced	$40K-$60K	Blue	25
10124	Attrited Customer	44	F	1	High School	Married	Less than $40K	Blue	36
10125	Attrited Customer	30	M	2	Graduate	Unknown	$40K-$60K	Blue	36
10126	Attrited Customer	43	F	2	Graduate	Married	Less than $40K	Silver	25

10127 rows × 20 columns

In [4]:
```python
ds.isnull().sum()
```

Out[4]:
```
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
dtype: int64
```

EDA

Seperate numerical and categorical data

In [5]:
```python
ds_num = ds[['Customer_Age', 'Dependent_count', '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']]
ds_cat = ds[['Attrition_Flag', 'Gender', 'Education_Level', 'Marital_Status', 'Income_Category', 'Card_Category']]
```

In [6]:
```
ds_cat.nunique()
```

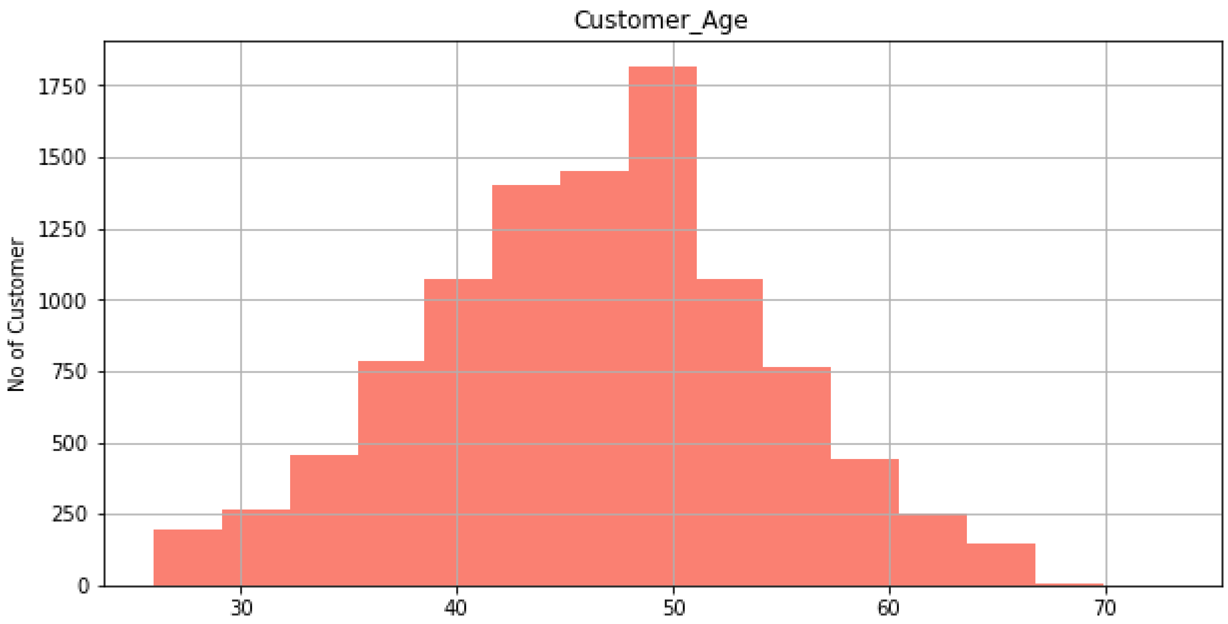
Out[6]:
```
Attrition_Flag     2
Gender             2
Education_Level    7
Marital_Status     4
Income_Category    6
Card_Category      4
dtype: int64
```

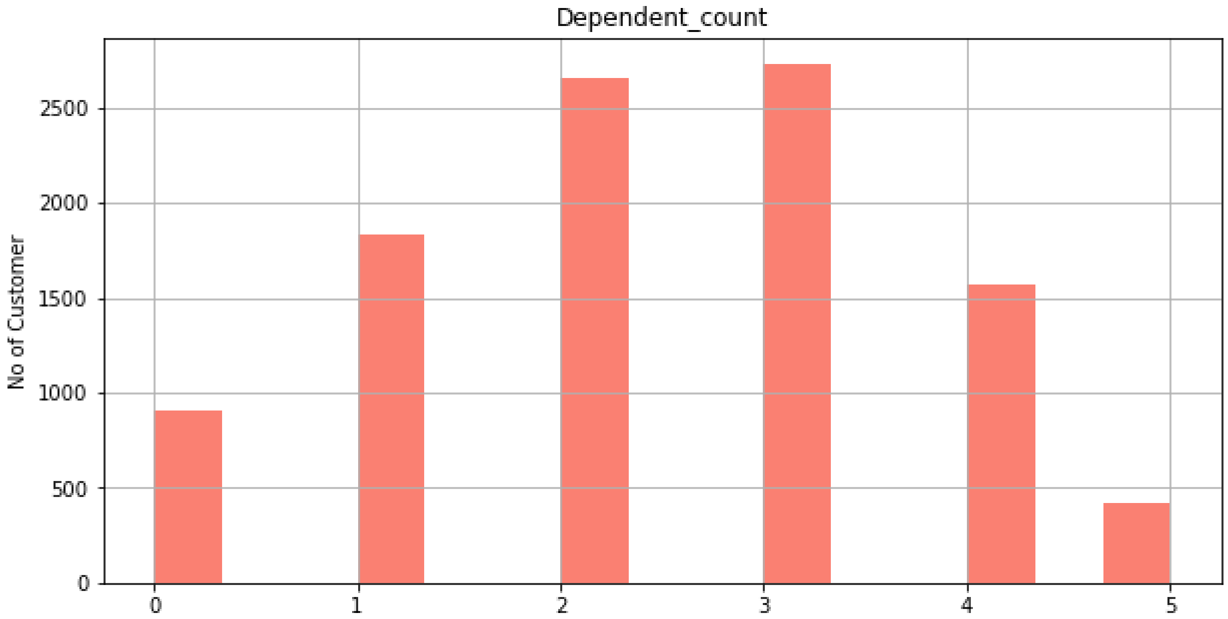
In [7]:
```
for i in ds_cat.columns:
    print(f'{i}: {pd.unique(ds_cat[i])}') #unique values for categorical data
```

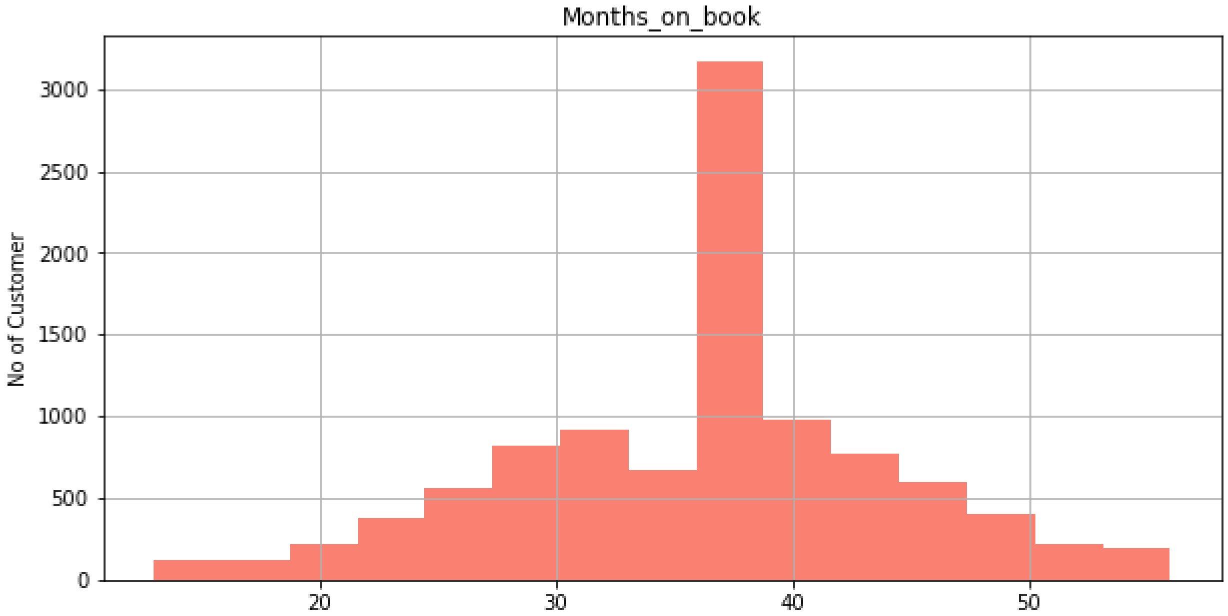
```
Attrition_Flag: ['Existing Customer' 'Attrited Customer']
Gender: ['M' 'F']
Education_Level: ['High School' 'Graduate' 'Uneducated' 'Unknown' 'College' 'Post-Graduate'
 'Doctorate']
Marital_Status: ['Married' 'Single' 'Unknown' 'Divorced']
Income_Category: ['$60K - $80K' 'Less than $40K' '$80K - $120K' '$40K - $60K' '$120K +'
 'Unknown']
Card_Category: ['Blue' 'Gold' 'Silver' 'Platinum']
```

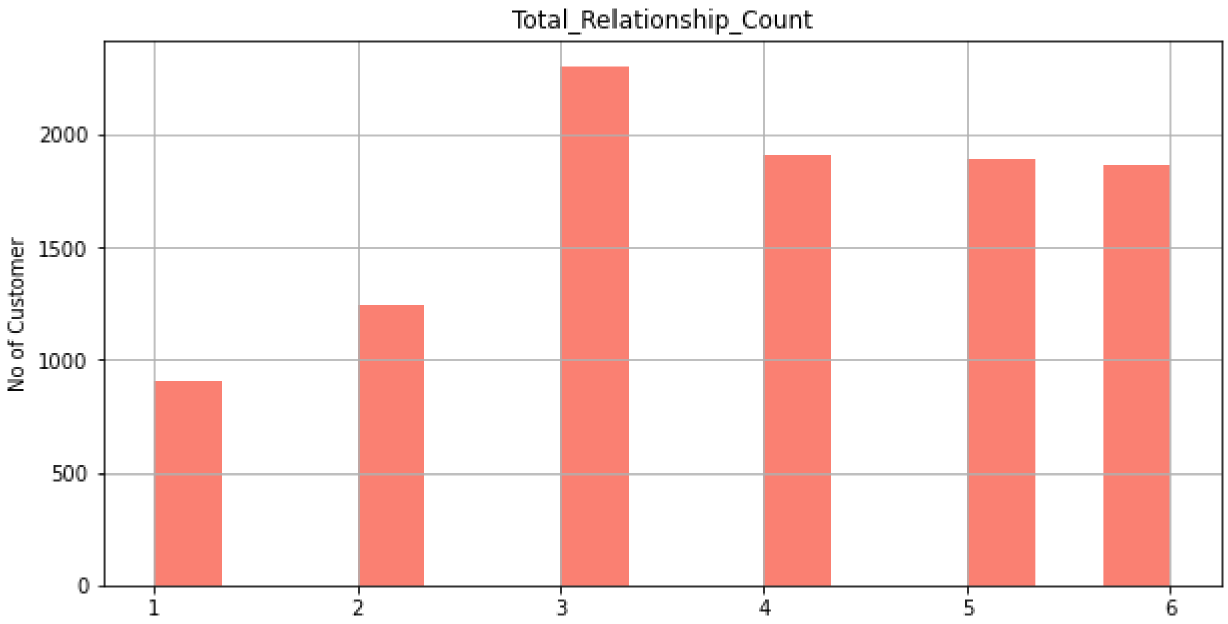
Visualize numerical data

In [68]:
```
for i in ds_num.columns:
    plt.subplots(figsize = (10,5))
    plt.hist(ds_num[i], color = 'salmon', bins = 15)
    plt.title(i)
    plt.grid()
    plt.ylabel('No of Customer')
    plt.show()
```

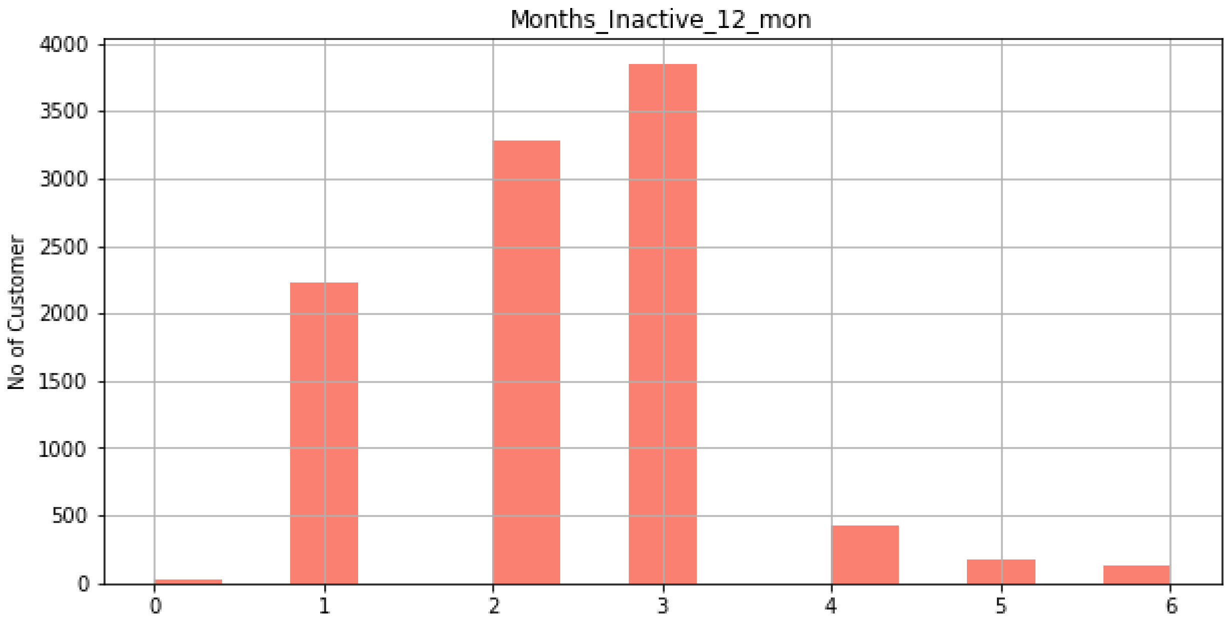



Dependent_count
No of Customer
0
500
1000
1500
2000
2500
0
1
2
3
4
5

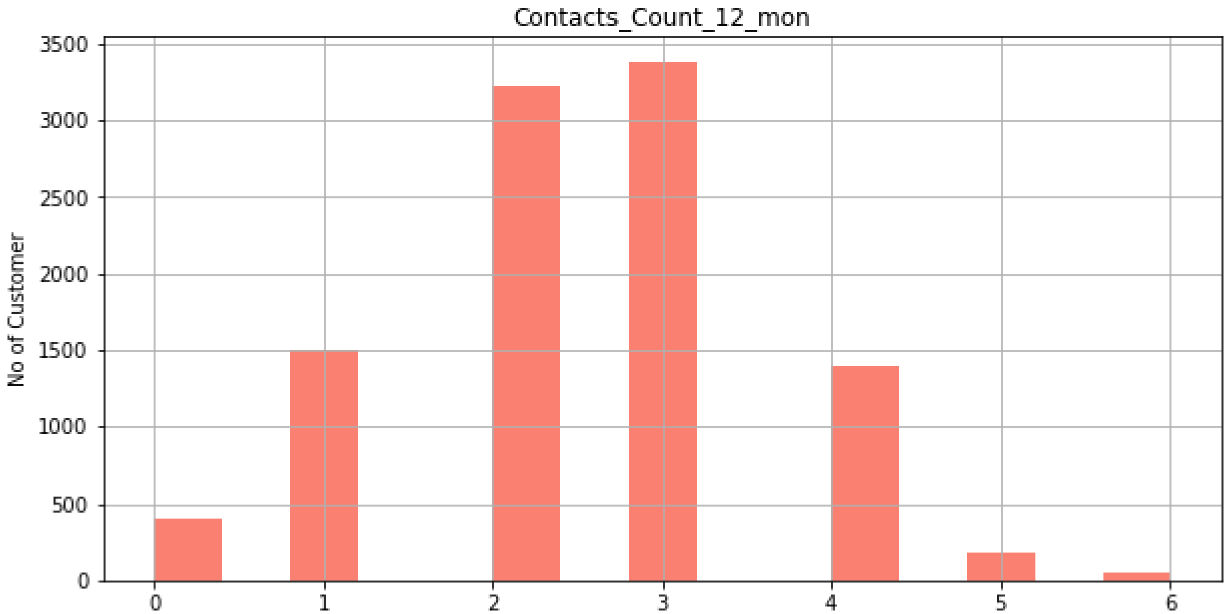

Months_on_book
No of Customer
3000
2500
2000
1500
1000
500
0
20
30
40
50

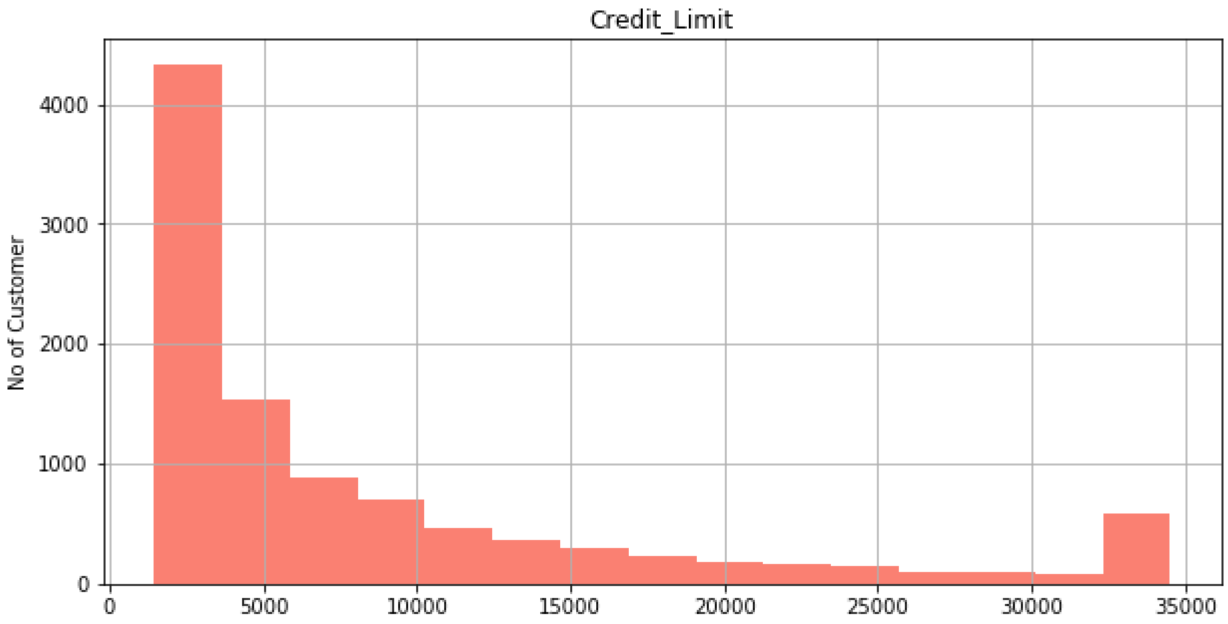


Total_Relationship_Count
No of Customer
0
500
1000
1500
2000
1
2
3
4
5
6

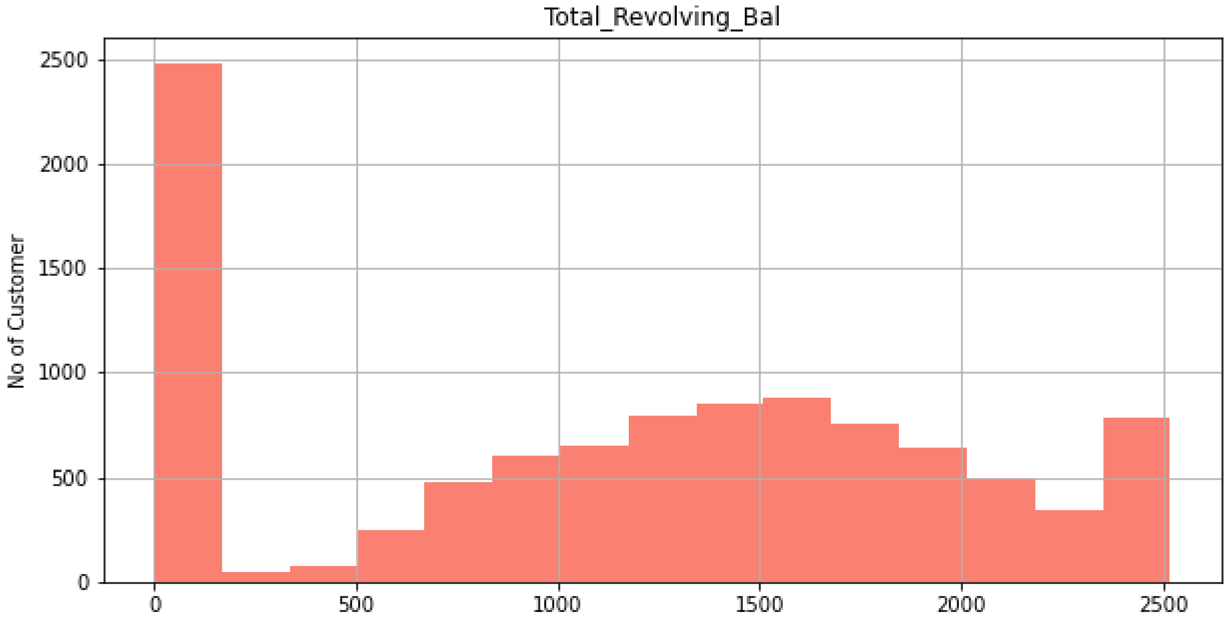


Months_Inactive_12_mon
No of Customer
0
500
1000
1500
2000
2500
3000
3500
4000
0
1
2
3
4
5
6

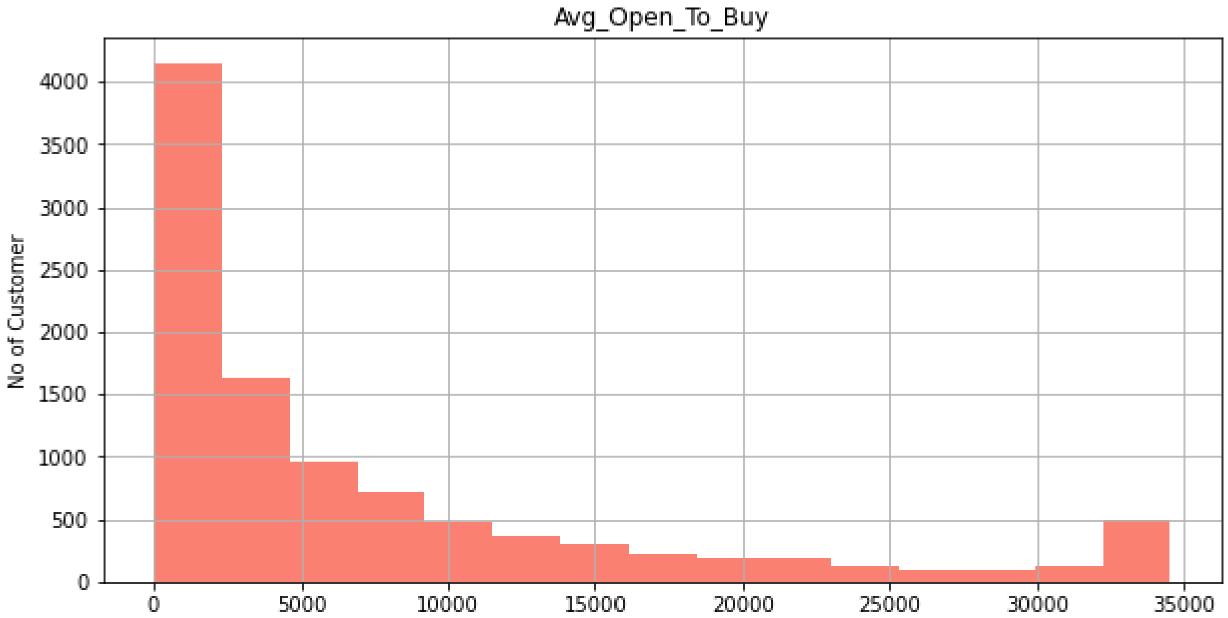

Contacts_Count_12_mon
No of Customer
0
500
1000
1500
2000
2500
3000
3500
0
1
2
3
4
5
6



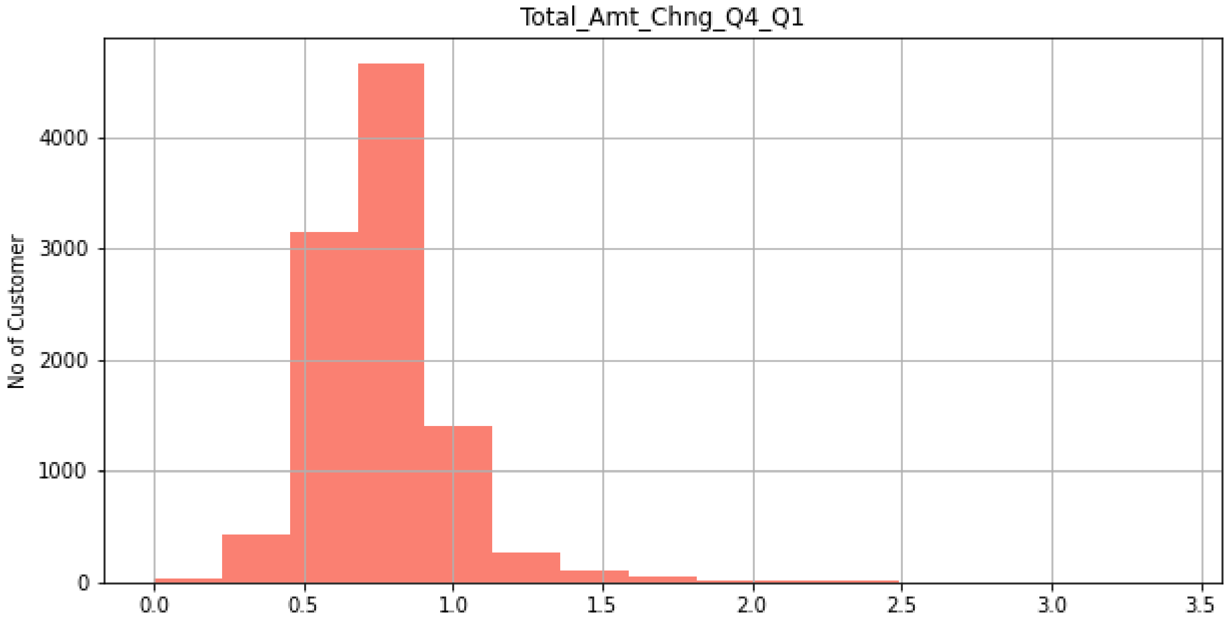
Credit_Limit
No of Customer
0
1000
2000
3000
4000
0
5000
10000
15000
20000
25000
30000
35000



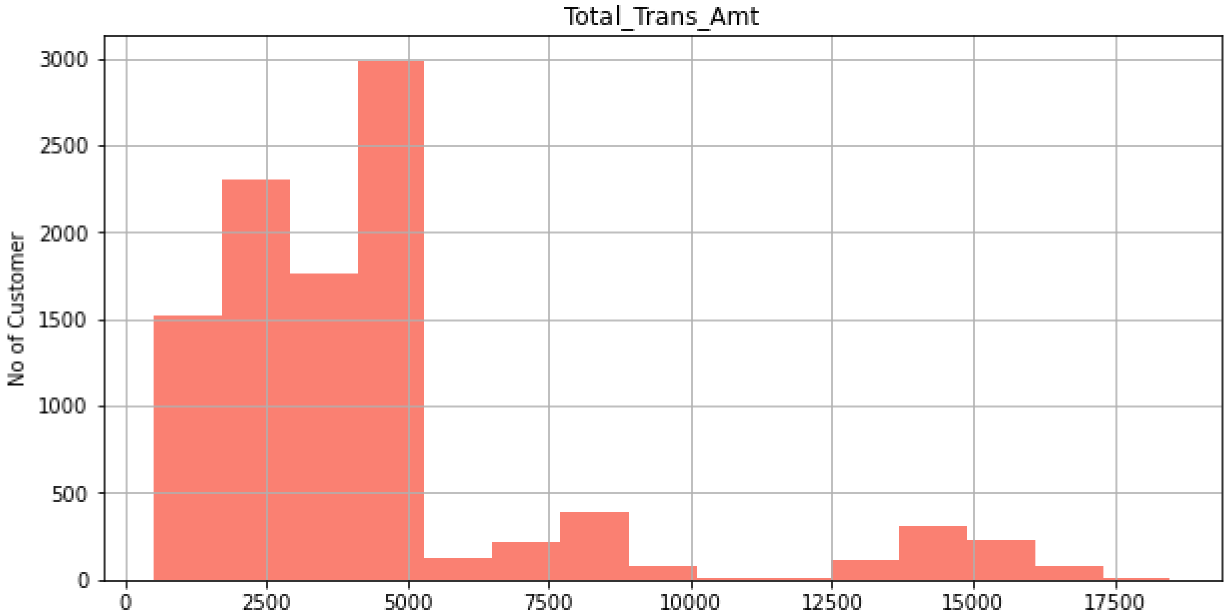
Total_Revolving_Bal
No of Customer
2500
2000
1500
1000
500
0
0
500
1000
1500
2000
2500

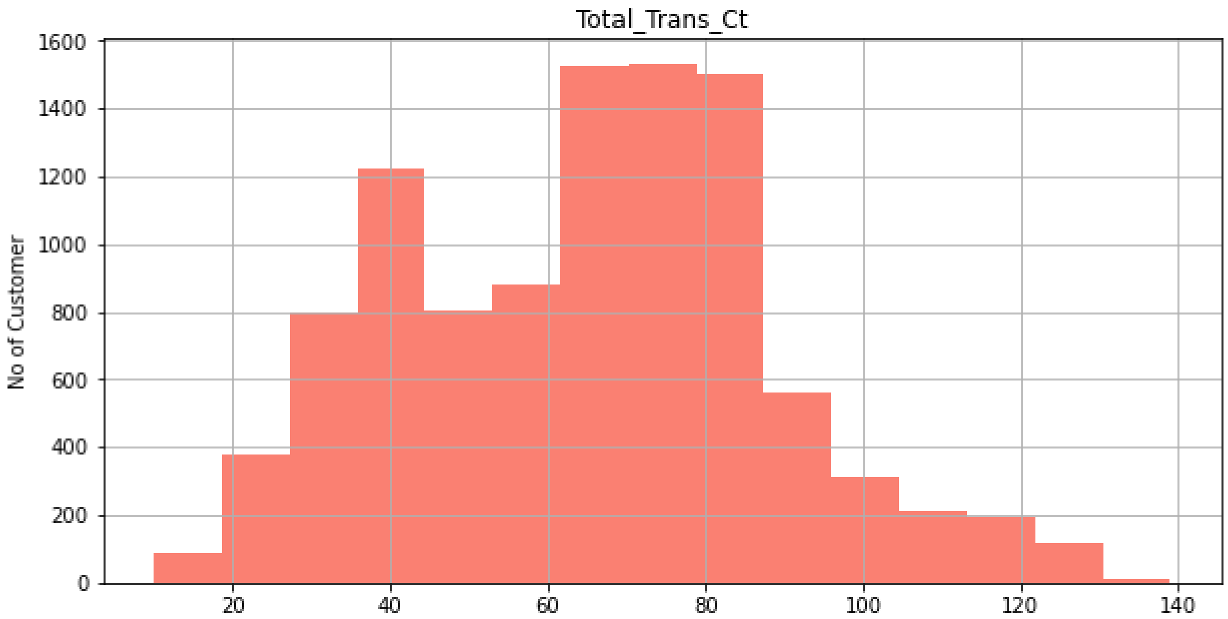


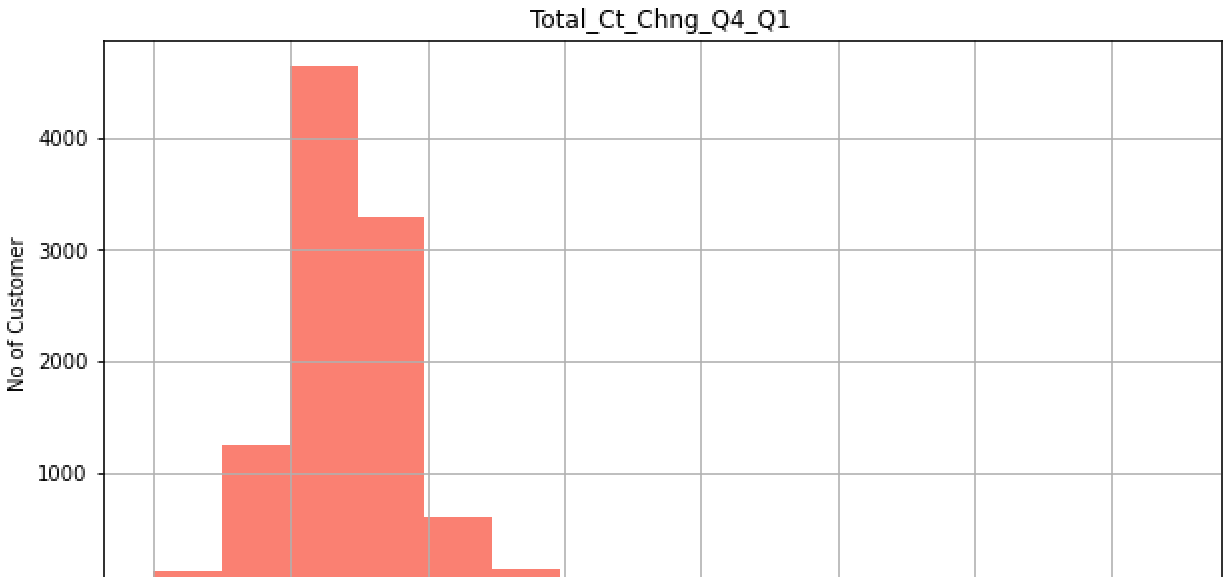
Avg_Open_To_Buy
No of Customer
0
500
1000
1500
2000
2500
3000
3500
4000
0
5000
10000
15000
20000
25000
30000
35000

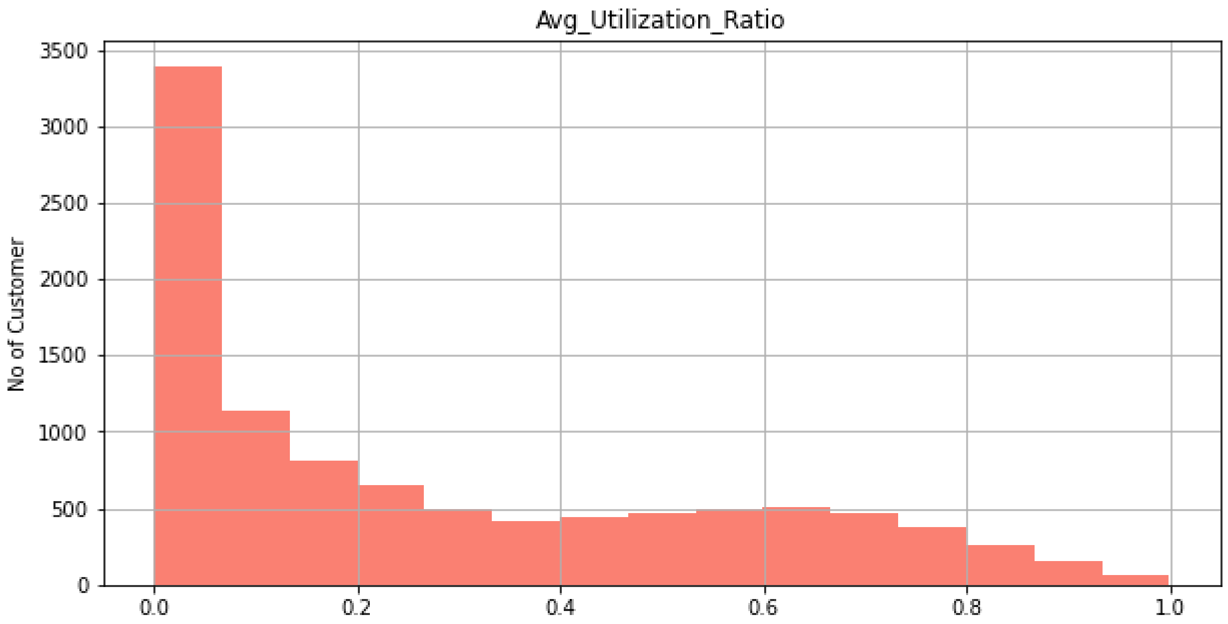


Total_Amt_Chng_Q4_Q1
No of Customer
0
1000
2000
3000
4000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Total_Trans_Amt
No of Customer
3000
2500
2000
1500
1000
500
0
0
2500
5000
7500
10000
12500
15000
17500


Total_Trans_Ct
No of Customer
0
200
400
600
800
1000
1200
1400
1600
20
40
60
80
100
120
140


Total_Ct_Chng_Q4_Q1
No of Customer
4000
3000
2000
1000


Avg_Utilization_Ratio
No of Customer
3500
3000
2500
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0

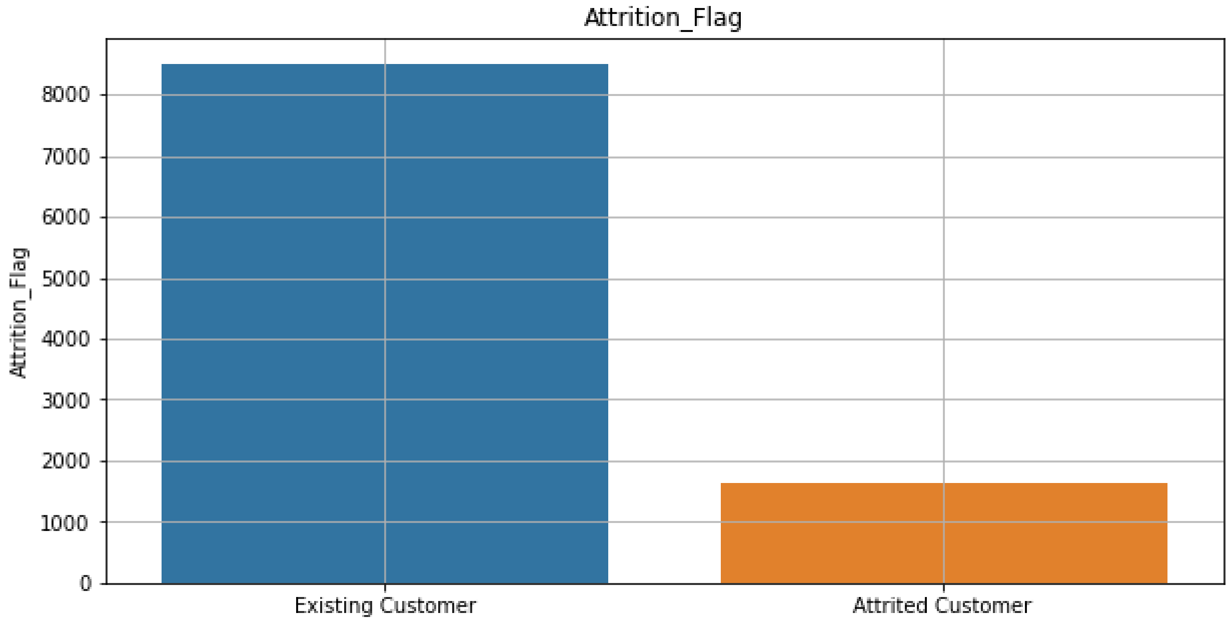
In [79]:
```
print('Oldest Customer: ', ds.Customer_Age.max())
print('Youngest Customer: ', ds.Customer_Age.min())
print('Lowest Months On Book: ', ds.Months_on_book.min())
print('Longest Months On Book: ', ds.Months_on_book.max())
```

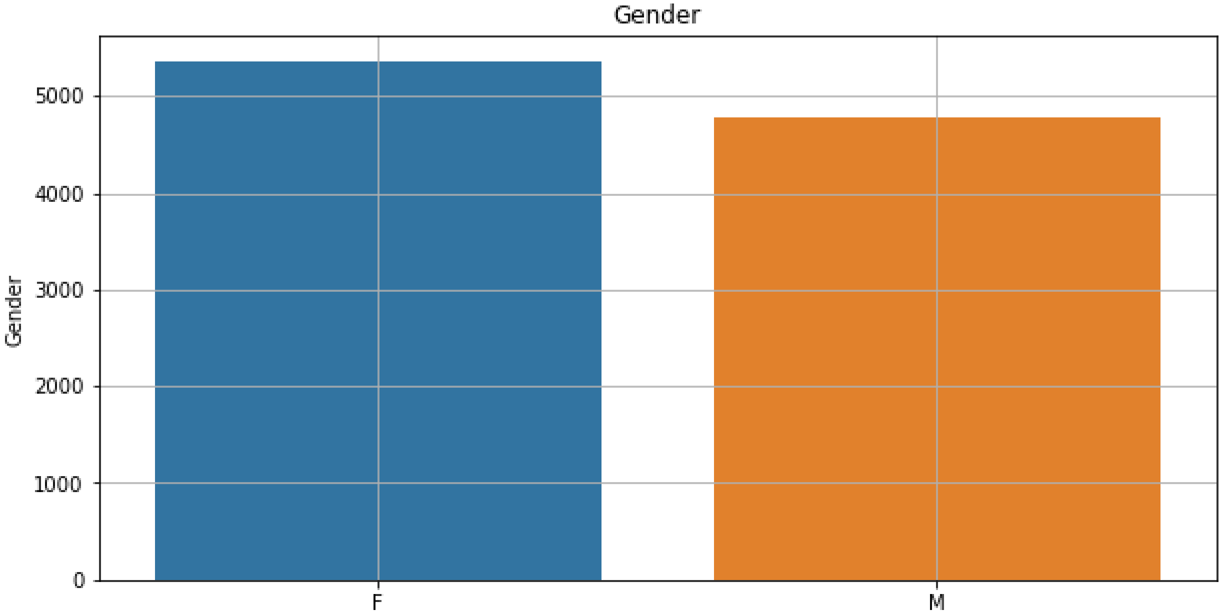
```
Oldest Customer:  73
Youngest Customer:  26
Lowest Months On Book:  13
Longest Months On Book:  56
```

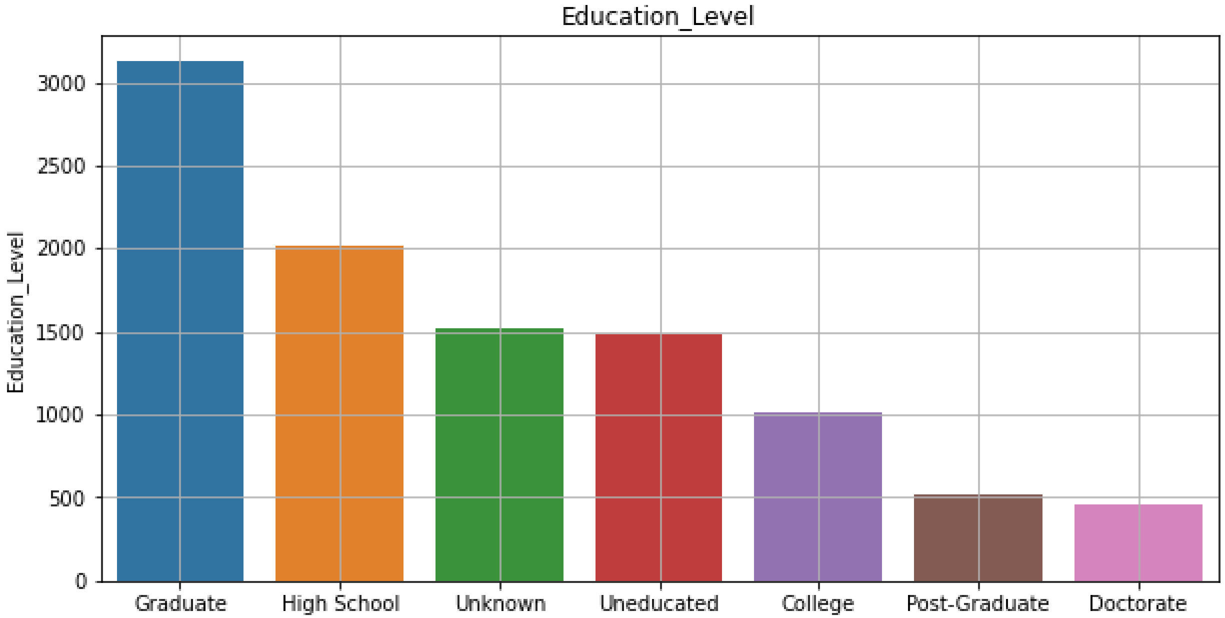
- The customer age histogram is symmetrical and unimodal. The peak of the histogram lies at 50 years which means most of the customer age is around 50 years old. The youngest customer age is 26 years old while the oldest customer age is 73 years old.
- The months on book histogram issymmetrical and unimodal. The peak of the histogram lies at 36 months which means most of the customer in on book for 36 months. The shortest months customer on book is 13 months while the longest months customer on book is 56 months.

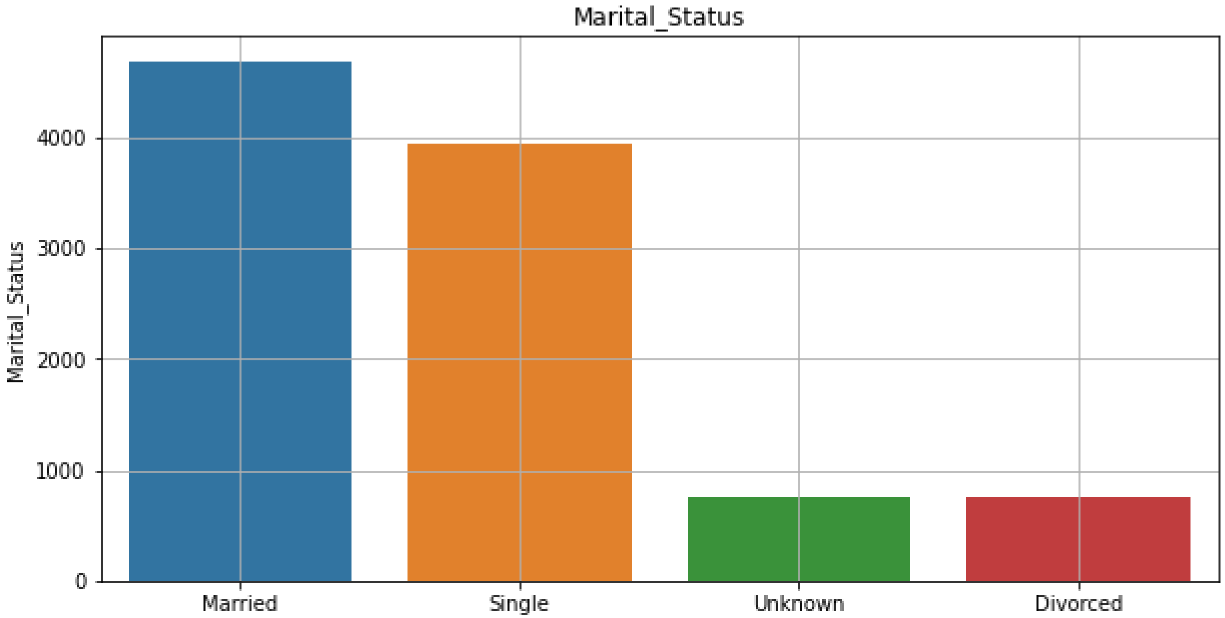
Visualize categorical data

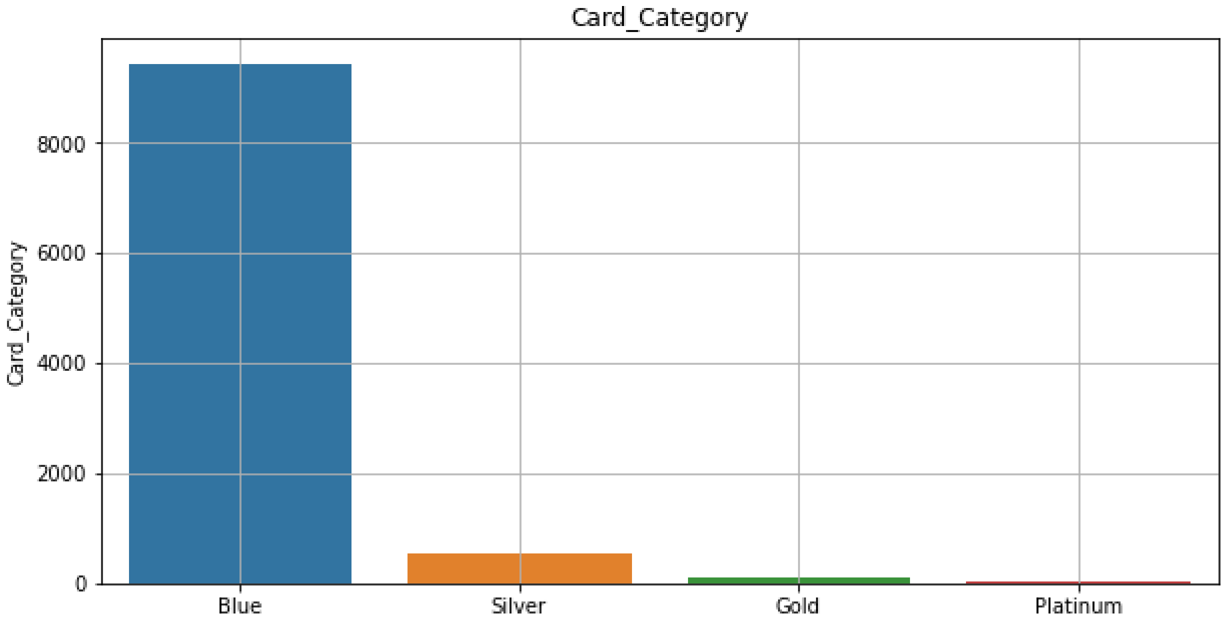
In [80]:
```
for i in ds_cat.columns:
    plt.subplots(figsize = (10,5))
    sns.barplot(ds_cat[i].value_counts().index, ds_cat[i].value_counts()).set_title (i)
    plt.grid()
    plt.show()
```




Gender
Gender
0
1000
2000
3000
4000
5000
F
M


Education_Level
Education_Level
0
500
1000
1500
2000
2500
3000
Graduate
High School
Unknown
Uneducated
College
Post-Graduate
Doctorate


Marital_Status
Marital_Status
4000
3000
2000
1000
0
Married
Single
Unknown
Divorced

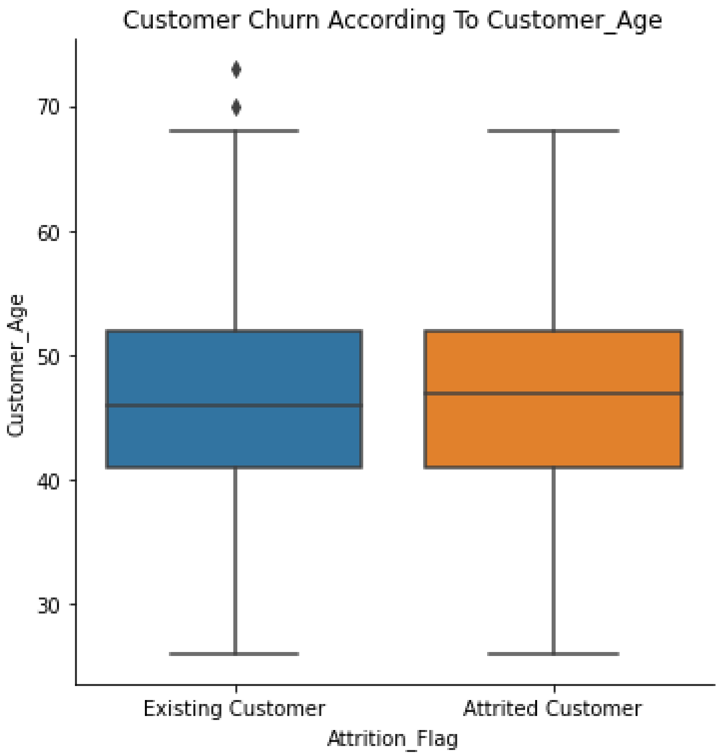


- Most of the credit card customer is a graduated education level.

- Most of the credit card customer has income less than 40k usd

Visualizing numerical data with output

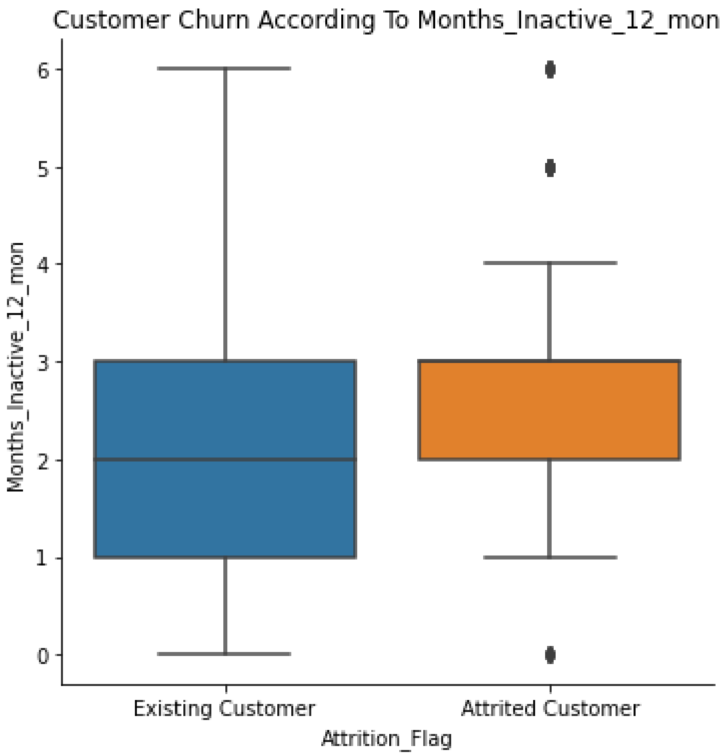
In [10]:
```
for i in ds_num.columns:
    sns.catplot(x='Attrition_Flag', y =ds_num[i], data = ds, kind = 'box')
    plt.title(f'Customer Churn According To {i}')
```



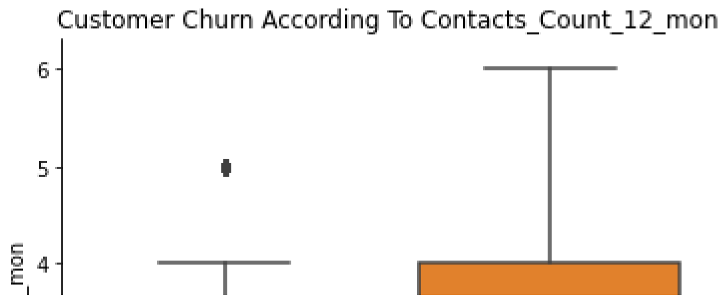

Customer Churn According To Dependent_count
Dependent_count
5
4
3
2
1
0
Existing Customer
Attrited Customer
Attrition_Flag

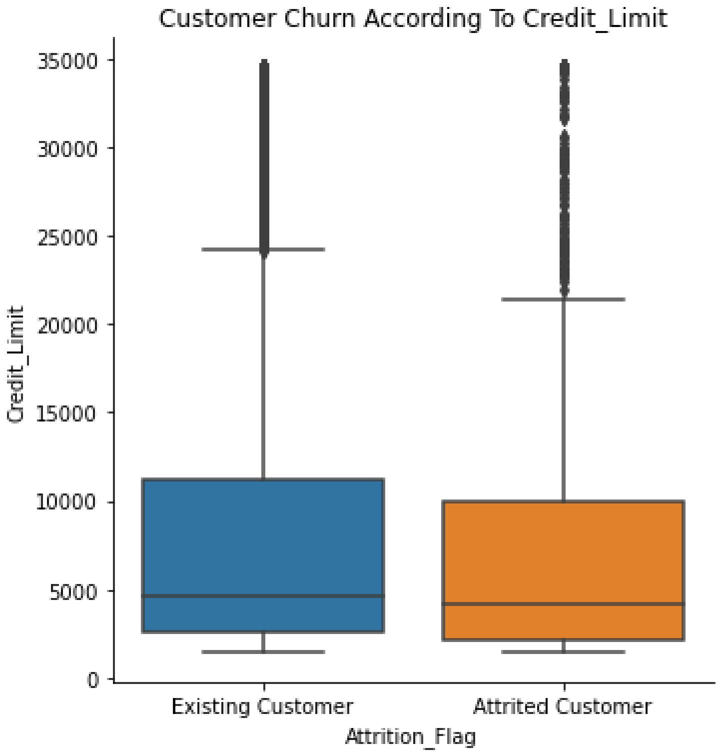

Customer Churn According To Months_on_book

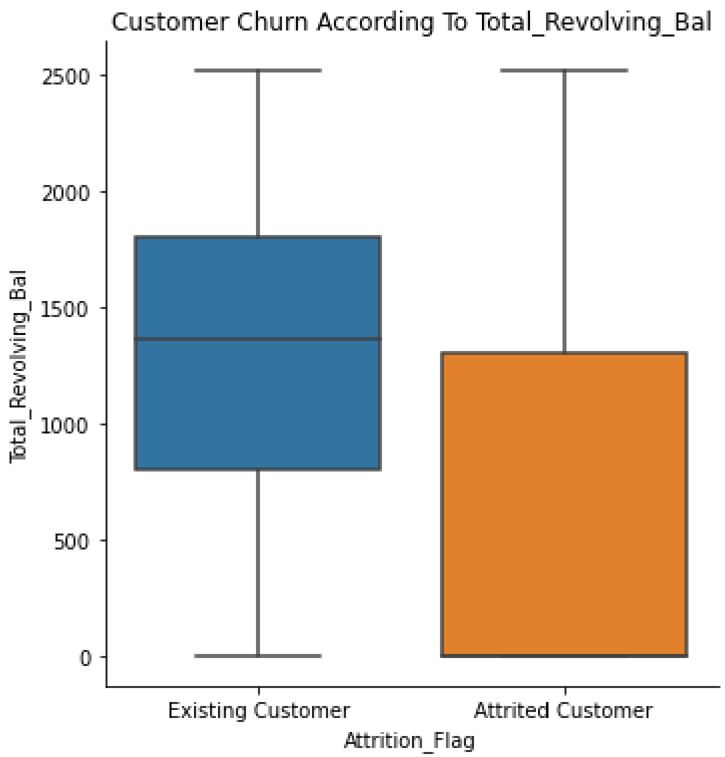

Customer Churn According To Total_Relationship_Count
Total_Relationship_Count
6
5
4
3
2
1
Existing Customer
Attrited Customer
Attrition_Flag

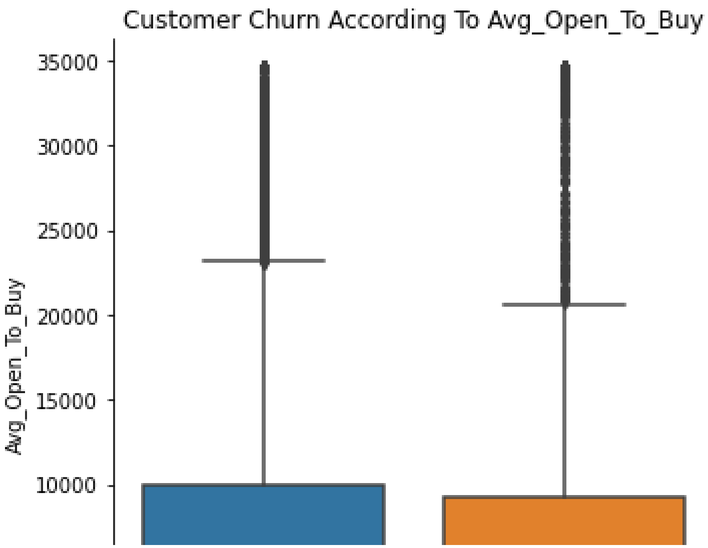


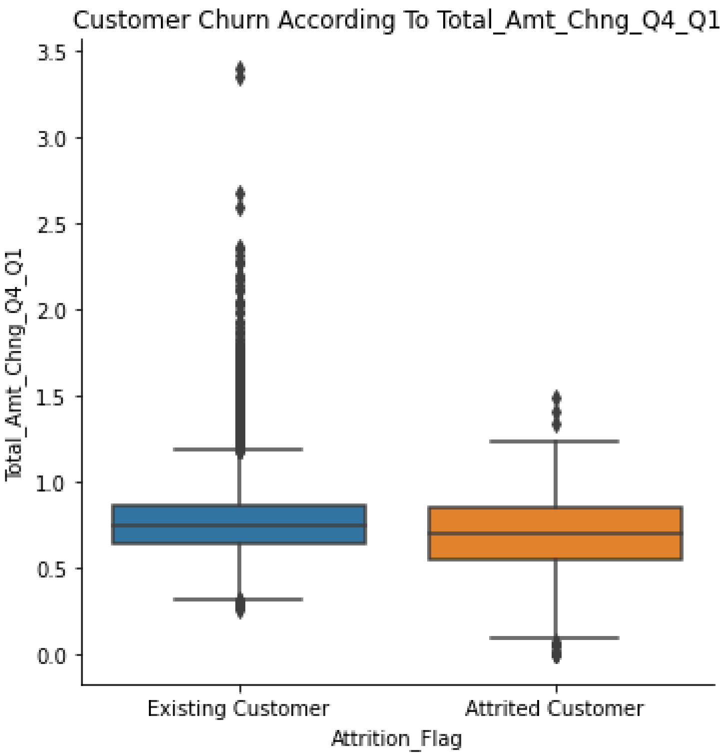
Customer Churn According To Months_Inactive_12_mon
Months_Inactive_12_mon
6
5
4
3
2
1
0
Existing Customer
Attrited Customer
Attrition_Flag

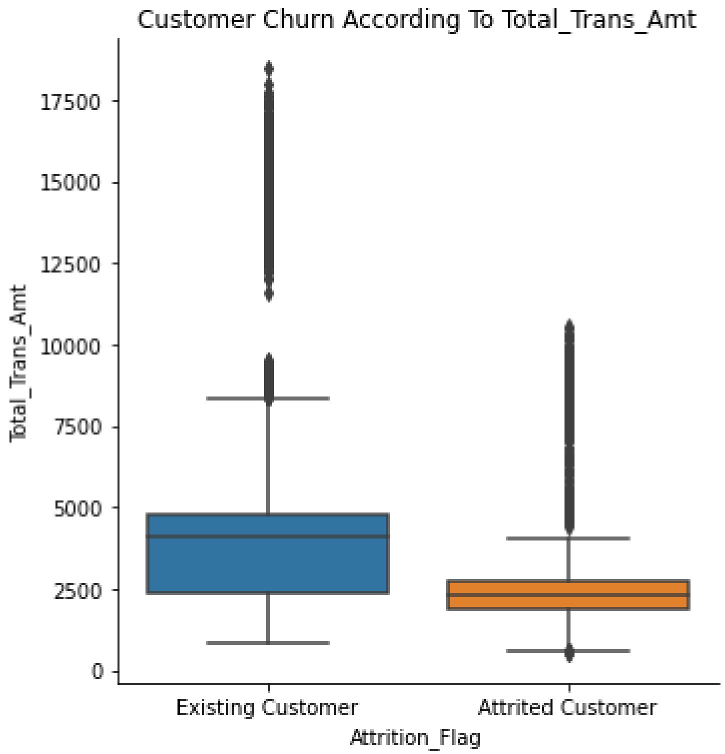

Customer Churn According To Contacts_Count_12_mon
6
5
4
_mon

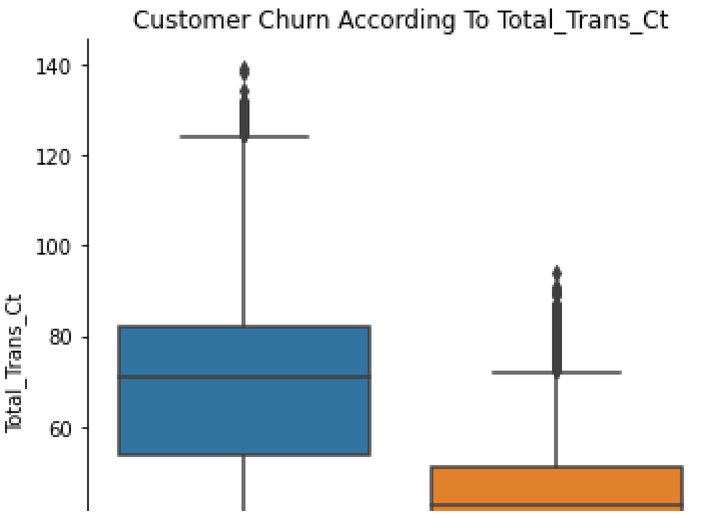

Customer Churn According To Credit_Limit
35000
30000
25000
20000
15000
10000
5000
0
Credit_Limit
Existing Customer
Attrited Customer
Attrition_Flag

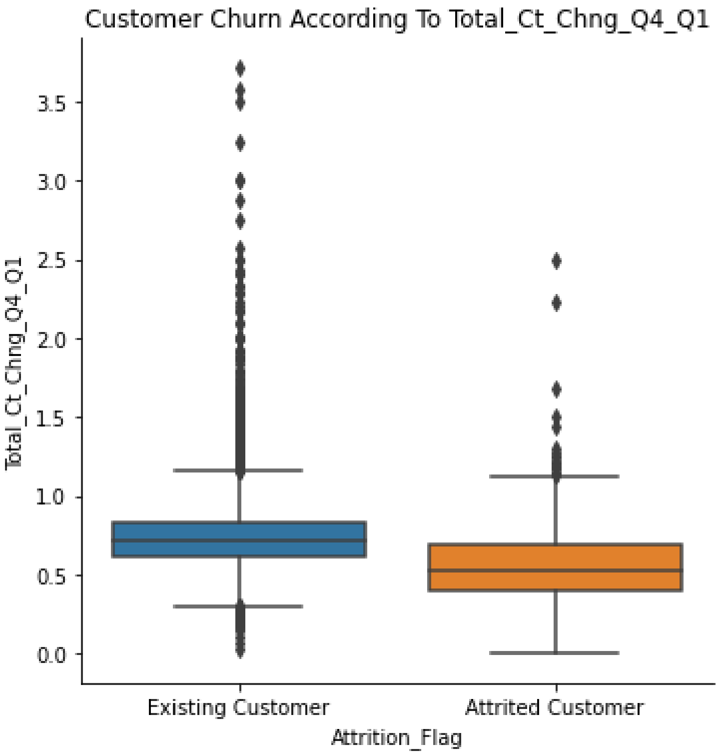

Customer Churn According To Total_Revolving_Bal
2500
2000
1500
1000
500
0
Total_Revolving_Bal
Existing Customer
Attrited Customer
Attrition_Flag

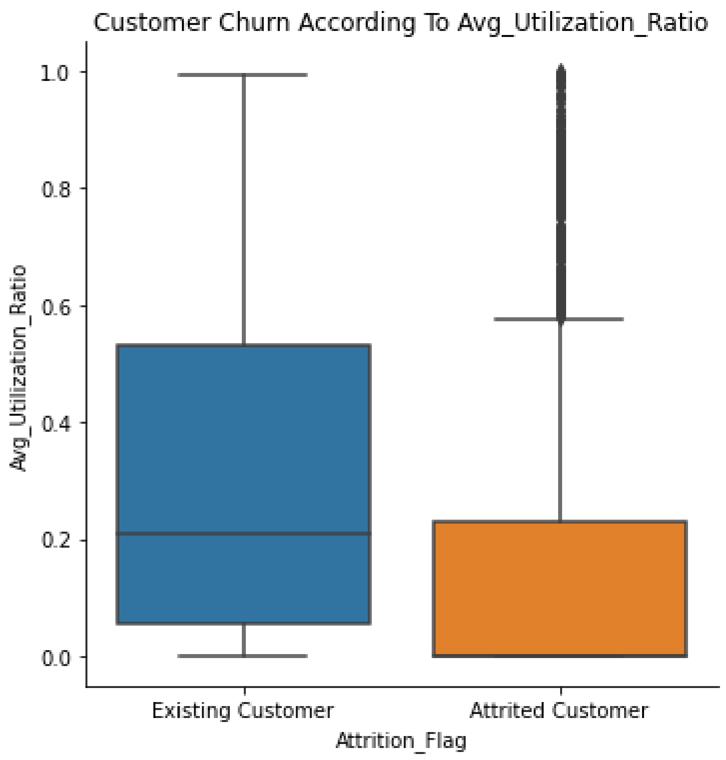

Customer Churn According To Avg_Open_To_Buy
Avg_Open_To_Buy
35000
30000
25000
20000
15000
10000


Customer Churn According To Total_Amt_Chng_Q4_Q1
Total_Amt_Chng_Q4_Q1
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Existing Customer
Attrited Customer
Attrition_Flag


Customer Churn According To Total_Trans_Amt
Total_Trans_Amt
17500
15000
12500
10000
7500
5000
2500
0
Existing Customer
Attrited Customer
Attrition_Flag


Customer Churn According To Total_Trans_Ct
Total_Trans_Ct
140
120
100
80
60


Customer Churn According To Total_Ct_Chng_Q4_Q1
Total_Ct_Chng_Q4_Q1
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Existing Customer
Attrited Customer
Attrition_Flag

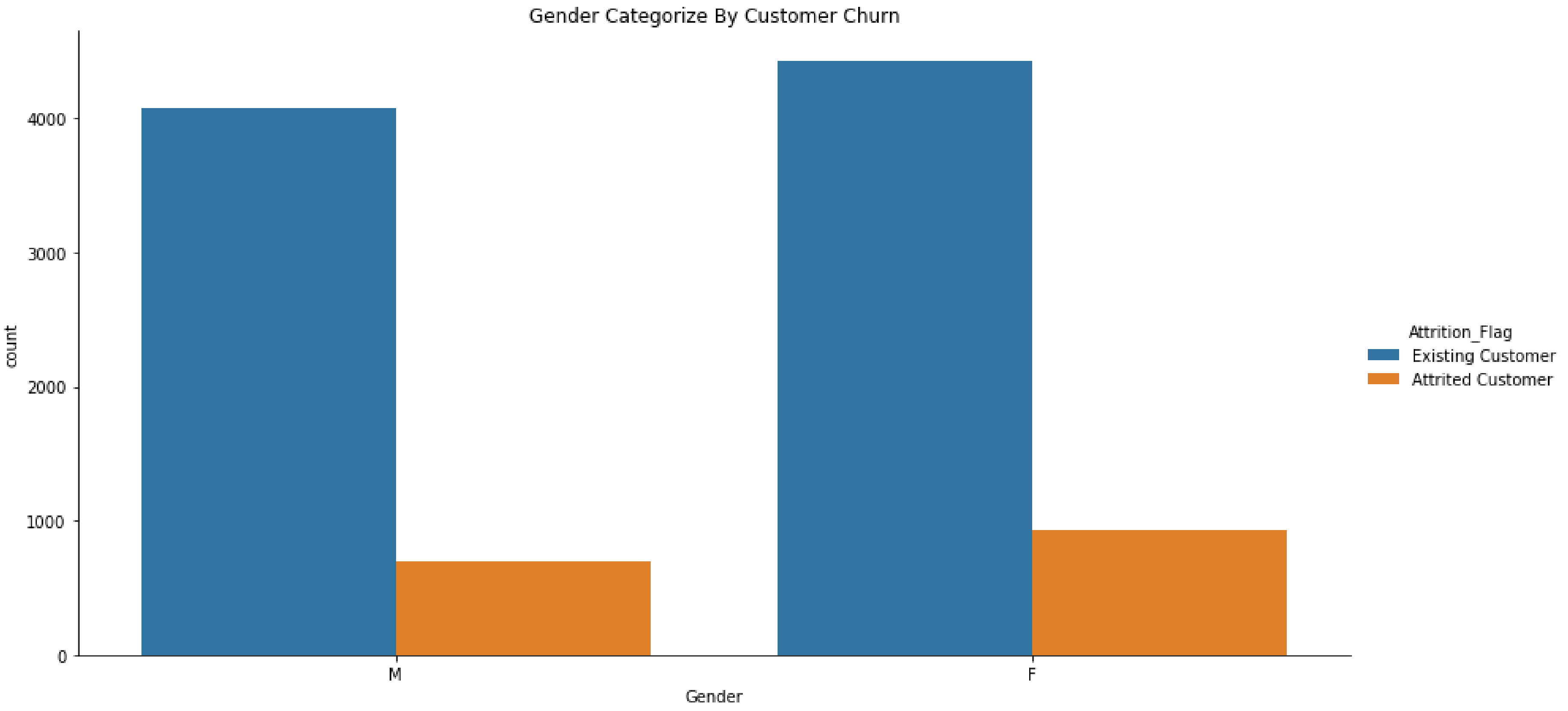


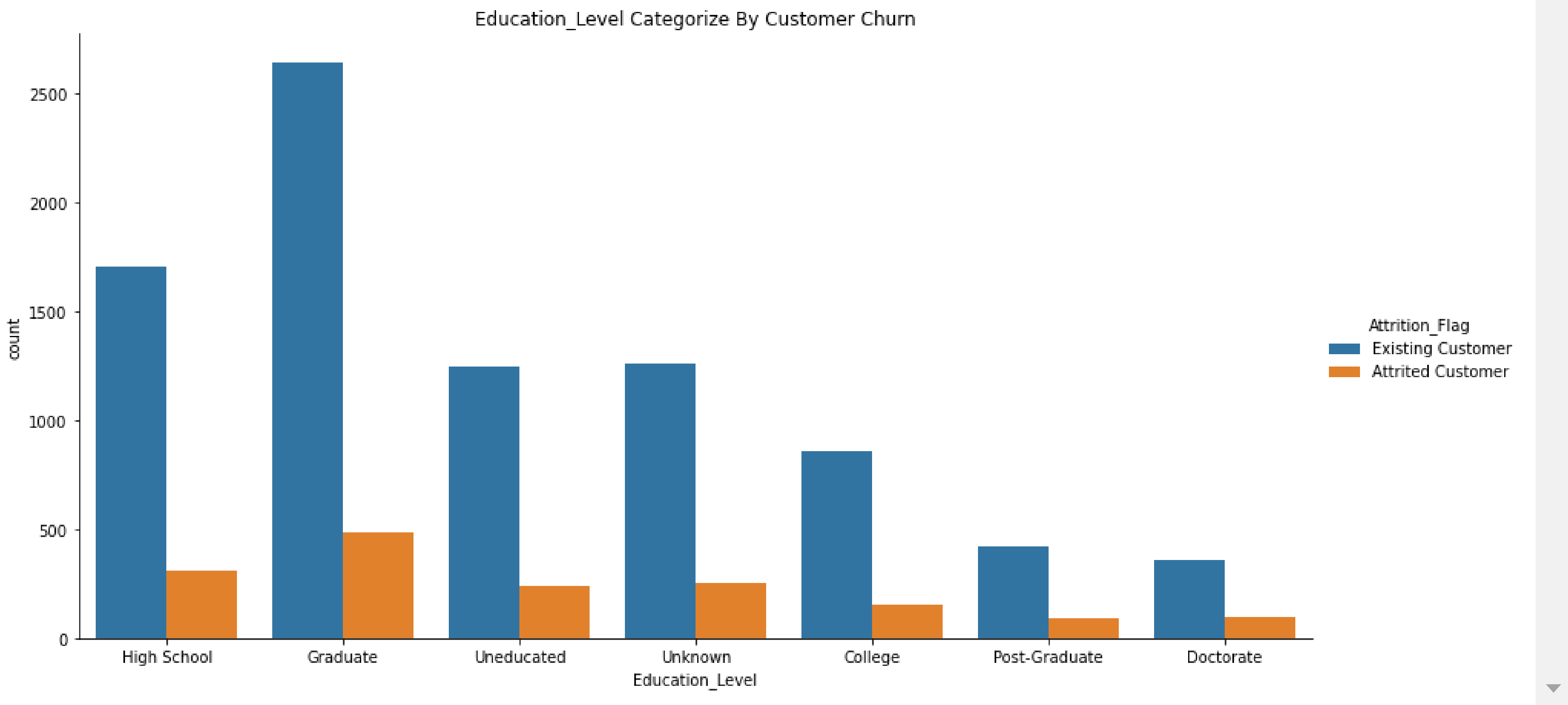
- We notice a major difference when analyzing the revolving balance of the customers credit lines. More than half of the churned customers have paid off all their debt. It is unclear whether this measure for the revolving balance is conducted before or after the termination of the credit line

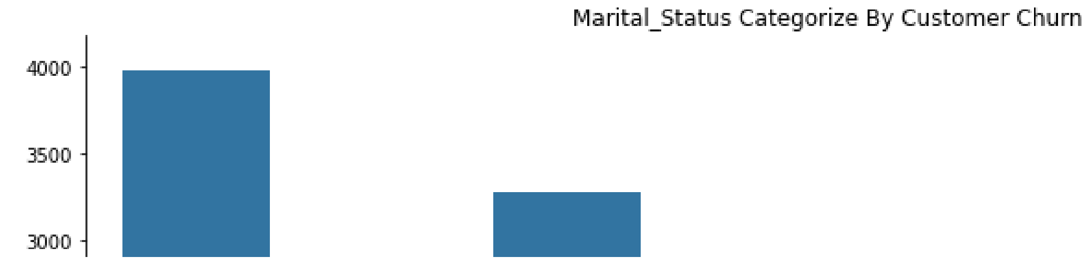
and whether customers who cancel their credit cards are asked to pay off all their standing debt.
- Most of credit card customer that has total amount of transactions made in the last year higher than 2500 usd and the number of transactions made in the last year higher than 50 transactions has lower probability to churn. Churned customers used their credit card less often and spent significantly less money.

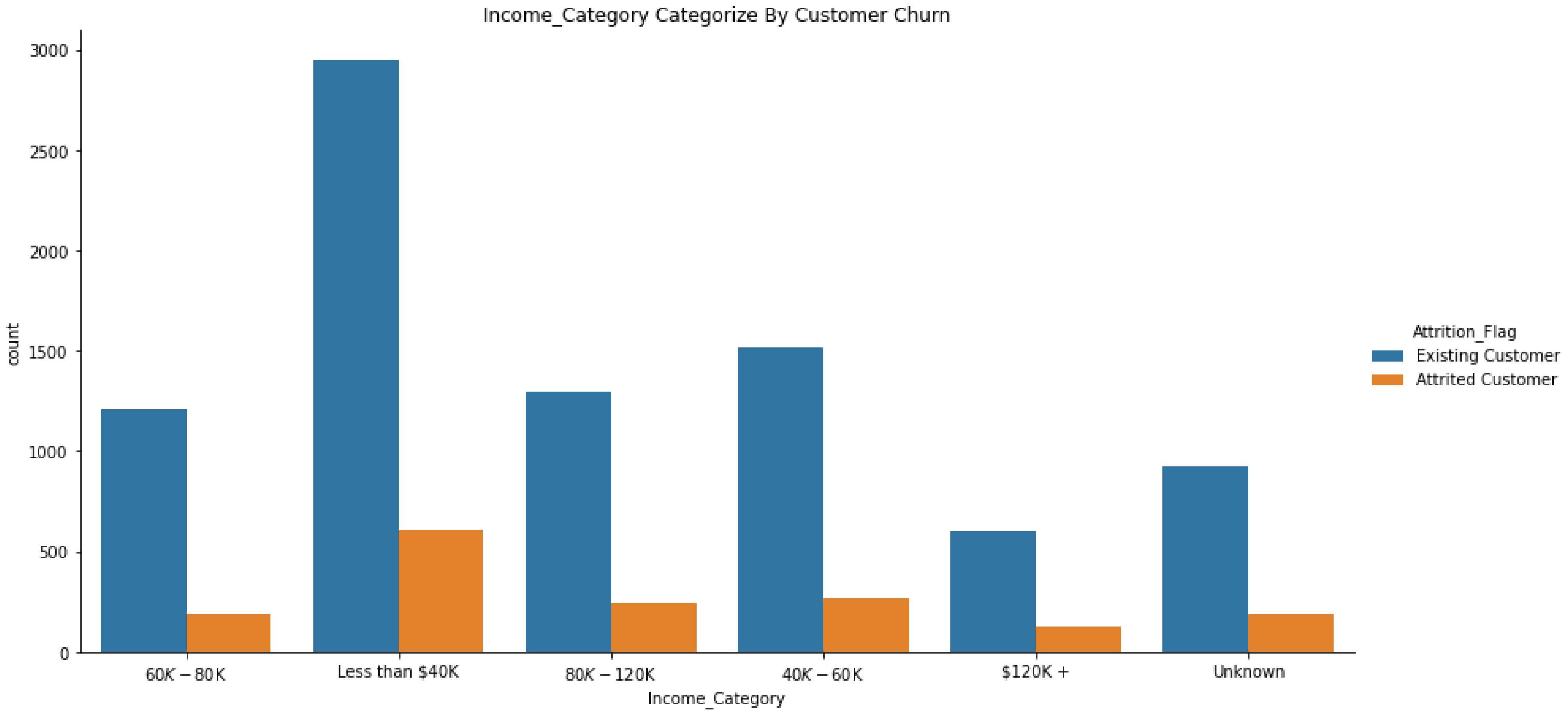
Visualizing categorical data with output

```
In [11]: for i in ds_cat.columns[1:]:
             sns.catplot(x = i, data = ds, kind = 'count', hue = 'Attrition_Flag', aspect = 2, height = 6)
             plt.title(f'{i} Categorize By Customer Churn')
```




Education_Level Categorize By Customer Churn
count
2500
2000
1500
1000
500
0
High School
Graduate
Uneducated
Unknown
College
Post-Graduate
Doctorate
Education_Level
Attrition_Flag
Existing Customer
Attrited Customer


Marital_Status Categorize By Customer Churn
4000
3500
3000


Income_Category Categorize By Customer Churn
count
3000
2500
2000
1500
1000
500
0
60K − 80K
Less than $40K
80K − 120K
40K − 60K
$120K +
Unknown
Income_Category
Attrition_Flag
Existing Customer
Attrited Customer

Preparing data for machine learning

In [13]:
```
X = ds.iloc[:,1:].values
y = ds.iloc[:,0].values
```

In [14]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
le = LabelEncoder()
y = le.fit_transform(y)
X[:,1] = le.fit_transform(X[:,1])
```

In [15]:
```
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3, 4, 5, 6])], remainder = 'passthrough')
X = ct.fit_transform(X)
```

In [16]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [17]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Machine learning

Naive Bayes

In [17]:
```
from sklearn.naive_bayes import GaussianNB

classifierNB = GaussianNB()
classifierNB = classifierNB.fit(X_train,y_train)
y_pred = classifierNB.predict(X_test)
```

In [18]:
```
from sklearn.metrics import confusion_matrix

cmNB = confusion_matrix(y_test, y_pred)
print(cmNB)
```

```
[[ 198  111]
 [ 144 1573]]
```

In [19]:
```
from sklearn.metrics import accuracy_score
ACC_NB = accuracy_score(y_test, y_pred)*100
print('The accurancy score is ', ACC_NB)
model_acc={}
model_acc['Naive Bayes'] = ACC_NB
```

```
The accurancy score is  87.41362290227048
```

Logistic regression

In [20]:
```
from sklearn.linear_model import LogisticRegression

classifierLR = LogisticRegression(random_state=0)
classifierLR.fit(X_train,y_train)
y_pred = classifierLR.predict(X_test)
```

In [21]:
```
cmLR = confusion_matrix(y_test, y_pred)
print(cmLR)
```

```
[[ 185  124]
 [  71 1646]]
```

In [22]:
```
ACC_LR = accuracy_score(y_test, y_pred)*100
print('The accurancy score is ', ACC_LR)
model_acc['Logistic Regression'] = ACC_LR
```

```
The accurancy score is  90.37512339585389
```

K Nearest Neighbours

In [23]:
```
from sklearn.neighbors import KNeighborsClassifier

classifierKNN = KNeighborsClassifier(n_neighbors=5, metric='minkowski',p=2)
classifierKNN.fit(X_train,y_train)
y_pred=classifierKNN.predict(X_test)
```

In [24]:
```
cmKNN = confusion_matrix(y_test,y_pred)
print(cmKNN)
```

```
[[  93  216]
 [  38 1679]]
```

In [25]:
```
ACC_KNN = accuracy_score(y_test, y_pred)*100
print('The accurancy score is ', ACC_KNN)
model_acc['K Nearest Neighbour'] = ACC_KNN
```

```
The accurancy score is  87.4629812438302
```

Support Vector Machine

In [26]:
```
from sklearn.svm import SVC

classifierSVC = SVC(kernel='linear', random_state=0) #kernel can be changed to increase accurancy
classifierSVC.fit(X_train,y_train)
y_pred = classifierSVC.predict(X_test)
```

In [27]:
```
cmSVC = confusion_matrix(y_test,y_pred)
print(cmSVC)
```

```
[[ 185  124]
 [  67 1650]]
```

In [28]:
```
ACC_SVC = accuracy_score(y_test, y_pred)*100
print('The accuracy score is ', ACC_SVC)
model_acc['Support Vector Machine'] = ACC_SVC
```

```
The accuracy score is  90.57255676209279
```

Desicion Tree

In [29]:
```
from sklearn.tree import DecisionTreeClassifier

classifierDT = DecisionTreeClassifier(criterion='entropy', random_state=0)
classifierDT.fit(X_train, y_train)
y_pred = classifierDT.predict(X_test)
```

In [30]:
```
cmDT = confusion_matrix(y_test, y_pred)
print(cmDT)
```

```
[[ 241   68]
 [  54 1663]]
```

In [31]:
```
ACC_DT = accuracy_score(y_test, y_pred)*100
print('The accurancy score is ', ACC_DT)
model_acc['Decision Tree'] = ACC_DT
```

```
The accurancy score is  93.97828232971372
```

Random Forest

In [32]:
```
from sklearn.ensemble import RandomForestClassifier

classifierRFC = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', random_state=0)
classifierRFC.fit(X_train, y_train)
y_pred = classifierRFC.predict(X_test)
```

In [33]:
```
cmRFC = confusion_matrix(y_test,y_pred)
print(cmRFC)
```

```
[[ 236   73]
 [  27 1690]]
```

In [34]:
```
ACC_RFC = accuracy_score(y_test, y_pred)*100
print('The accurancy score is ', ACC_RFC)
model_acc['Random Forest Classification'] = ACC_RFC
```

```
The accurancy score is  95.06416584402764
```

XGBoost

In [35]:
```
import xgboost as xgb

clf= xgb.XGBClassifier(max_depth=7, n_estimators=100, colsample_bytree=0.8, subsample=0.8, learning_rate=0.09)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
[20:25:21] WARNING: ..\src\learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the obje
ctive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

In [36]:
```
cmXG = confusion_matrix(y_test,y_pred)
print(cmXG)
```

```
[[ 270   39]
 [  14 1703]]
```

In [37]:
```
ACC_XG = accuracy_score(y_test, y_pred)*100
print('The accurancy score is ', ACC_XG)
model_acc['XGBoost'] = ACC_XG
```

```
The accurancy score is  97.38400789733464
```

Artificial Neural Network

In [60]:
```python
import tensorflow as tf

ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(units=5, activation='relu')) #input layer and first hidden layer
ann.add(tf.keras.layers.Dense(units=5, activation='relu')) #second hidden layer
ann.add(tf.keras.layers.Dense(units=5, activation='relu')) #third hidden layer
ann.add(tf.keras.layers.Dense(units=5, activation='relu')) #fourth hidden layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid')) #output layer
```

In [61]:
```python
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 7/100
254/254 [==============================] - 1s 5ms/step - loss: 0.2492 - accuracy: 0.9026
Epoch 8/100
254/254 [==============================] - 1s 5ms/step - loss: 0.2425 - accuracy: 0.9031
Epoch 9/100
254/254 [==============================] - 1s 5ms/step - loss: 0.2362 - accuracy: 0.9046
Epoch 10/100
254/254 [==============================] - 1s 5ms/step - loss: 0.2310 - accuracy: 0.9080
Epoch 11/100
254/254 [==============================] - 1s 5ms/step - loss: 0.2261 - accuracy: 0.9098
Epoch 12/100
254/254 [==============================] - 1s 5ms/step - loss: 0.2212 - accuracy: 0.9101
Epoch 13/100
254/254 [==============================] - 1s 5ms/step - loss: 0.2157 - accuracy: 0.9104
Epoch 14/100

254/254 [==============================] - 1s 5ms/step - loss: 0.2112 - accuracy: 0.9119
Epoch 15/100
254/254 [==============================] - 1s 5ms/step - loss: 0.2067 - accuracy: 0.9129
Epoch 16/100
```

In [62]:
```python
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)
```

Out[62]:
```
array([[1, 1],
       [0, 0],
       [1, 1],
       ...,
       [1, 0],
       [1, 1],
       [1, 1]])
```

In [63]:
```python
cmAnn = confusion_matrix(y_test, y_pred)
print(cmAnn)
```

```
[[ 239   70]
 [  66 1651]]
```

In [64]:
```python
ACC_Ann = accuracy_score(y_test, y_pred)*100
print('The accurancy score is ', ACC_Ann)
model_acc['Artificial Neural Network'] = ACC_Ann
```

```
The accurancy score is  93.2872655478776
```

Accurancy summary of all model

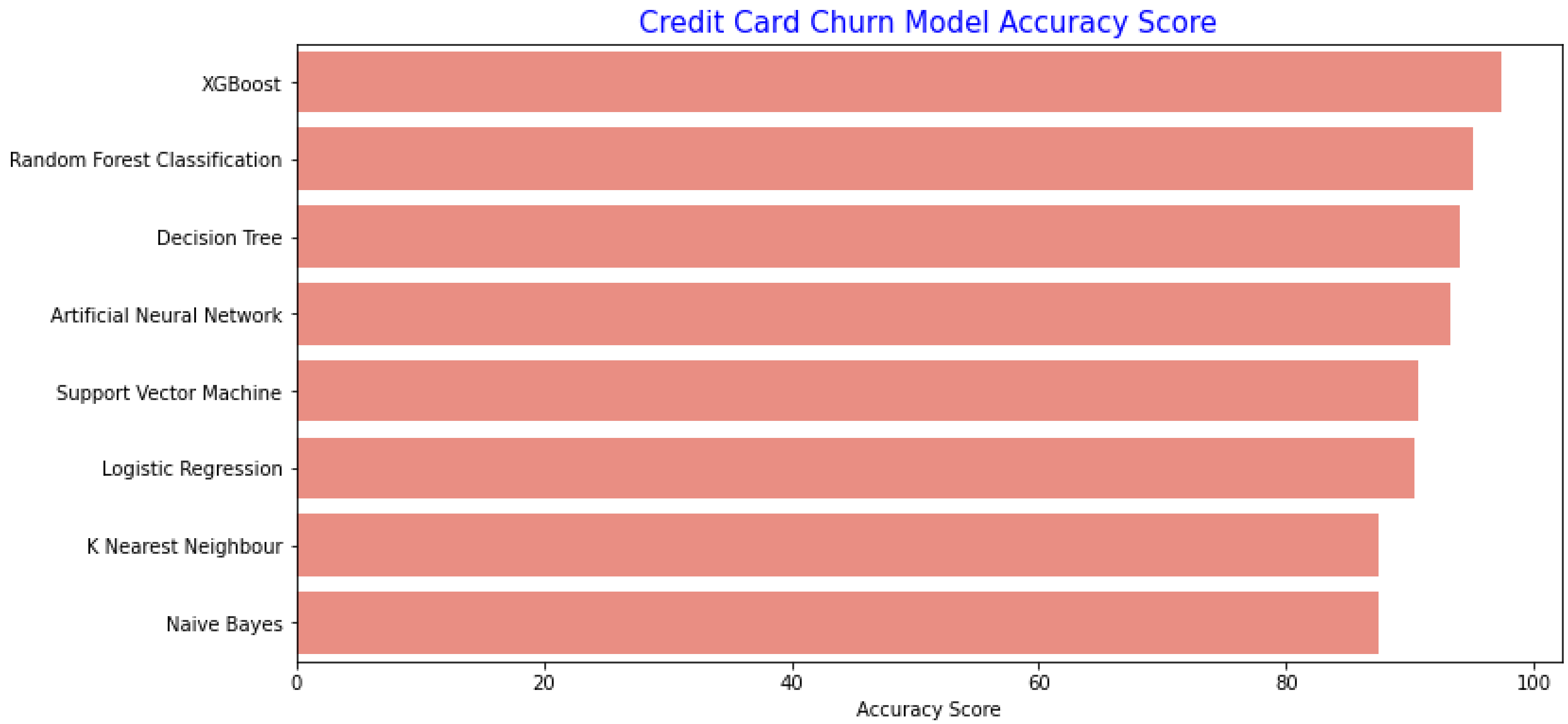
In [65]:
```
model_ds = pd.DataFrame.from_dict(model_acc,orient = 'index',columns = ['Accuracy Score'])
model_ds = model_ds.sort_values(by ='Accuracy Score',ascending = False)
model_ds
```

Out[65]:

	Accuracy Score
XGBoost	97.384008
Random Forest Classification	95.064166
Decision Tree	93.978282
Artificial Neural Network	93.287266
Support Vector Machine	90.572557
Logistic Regression	90.375123
K Nearest Neighbour	87.462981
Naive Bayes	87.413623

In [66]:
```python
gig,ax = plt.subplots(figsize = (12,6))
sns.barplot(x="Accuracy Score", y=model_ds.index, data=model_ds,color = 'salmon')
plt.title('Credit Card Churn Model Accuracy Score', fontsize=15, color="blue")
```

Out[66]: `Text(0.5, 1.0, 'Credit Card Churn Model Accuracy Score')`



For conclusion, XGBoost model have the highest accuracy than other model which is 97.384%. Other model approach does not perform as good as XGB which is the second highest accuracy is Random Forest with 95%. But from the confusion matrix score, the data have high bias towards existing customer. We need more data from customer that has churn to get the complexity of the model.

```
In [ ]:
```