Credit Card Customers: Predict Churning Customers (Alternate Project Option #2)

Group: Anthony Yun, Danny Sov

The URL for the data

https://www.kaggle.com/sakshigoyal7/credit-card-customers

The cleaned dataset contains the following fields:

- **Income_Sorted***: Income_Category was sorted in numerical descending order:
  - (4 = $120K +, 3 = $80K - $120K, 2 = $60K - $80K, 1 = $40K - $60K, 0 = Less than $40K)
- **Attrition_Flag**: Internal event (customer activity) variable - if the account is closed then 1 else 0
- **Age**: Demographic variable - Customer's Age in Years
- **Gender**: Demographic variable - M=Male, F=Female
- **Education_Level**: Demographic variable - Educational Qualification of the account holder (example: high school, college graduate, etc.)
- **Marital_Status**: Demographic variable - Married, Single, Divorced, Unknown
- **Income_Category**: Demographic variable - Annual Income Category of the account holder:
  - (< $40K, $40K - 60K, $60K - $80K, $80K-$120K, > $120K, Unknown)
- **Length_Of_Relationship**: Period of relationship with bank
- **Total_Relationship_Count**: Total no. of products held by the customer
- **Customer_Contacts**: 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
- **Available_Credit**: Open to Buy Credit Line (Average of last 12 months)
- **Total_Trans_Amt**: Total Transaction Amount (Last 12 months)
- **Total_Trans_Ct**: Total Transaction Count (Last 12 months)
- **Avg_Trans_Amt***: Total Transaction Amount divided by Total Transaction Count (Last 12 months)
- **Relationship_Category***: How many years of relationship the customer has:
  - Diamond VIP >= 50 yrs,
  - Platinum VIP = 40-49 yrs,
  - Gold VIP = 30-39 yrs,
  - Silver VIP = 20-29 yrs,
  - Bronze VIP = 10-19 yrs,
  - Valued Customer = under 10 yrs

※ Questions

1. What are the demographics of the customers?
2. Does education level impact the credit line of the customers?
3. How do female and male customers compare in reported income and are customers with higher income given higher credit limits?
4. Is it possible that the highest credit limits being given to mostly males is due to household income reported for the husband's income in married couples?
5. Is it less likely for customers to leave if they have multiple products with one financial institution?
6. How does customer's balance and usage affect attrition?

※ Import Libraries and Set-Up for Visualizations & Statistics

In [6]:
```
print('Done by Anthony')
# filter warnings from displaying in Jupyter Notebook
import warnings;
warnings.filterwarnings('ignore')
```

```
Done by Anthony
```

In [7]:
```
print('Done by Anthony')
# import required libraries of pandas, numpy
import pandas as pd
import numpy as np
# import visualization libraries of pyplot and seaborn
import matplotlib.pyplot as plt
import seaborn as sns
# set default matplotlib style to 'ggplot'
plt.style.use('ggplot')

# set default seaborn theme, scaling, and color palette
sns.set()

# import statistics package
from scipy import stats
from scipy.stats import normaltest
# allow matplot plots to show in jupyter notebook
%matplotlib inline
```

```
Done by Anthony
```

※ Import Data

In [9]:
```
print('Done by Anthony')
# load dataset BankChurners.csv into a dataframe and read it into DataFrame:
df = pd.read_csv('BankChurners.csv')
```

```
Done by Anthony
```

※ Inspect Data

In [11]:
```
print('Done by Anthony')
# inspect the top 5 rows of the dataframe using the head method
df.head()
```

```
Done by Anthony
```

Out[11]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_L
0	768805383	Existing Customer	45	M	3	High Sc
1	818770008	Existing Customer	49	F	5	Grac
2	713982108	Existing Customer	51	M	3	Grac
3	769911858	Existing Customer	40	F	4	High Sc
4	709106358	Existing Customer	40	M	3	Uneduc

5 rows × 23 columns

In [12]:
```
print('Done by Anthony')
# inspect the bottom 5 rows of the dataframe using the tail method
df.tail()
```

```
Done by Anthony
```

Out[12]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Educati
10122	772366833	Existing Customer	50	M	2	
10123	710638233	Attrited Customer	41	M	2	
10124	716506083	Attrited Customer	44	F	1	Hi
10125	717406983	Attrited Customer	30	M	2	
10126	714337233	Attrited Customer	43	F	2	

5 rows × 23 columns

In [13]:
```
print('Done by Anthony')
# call the shape attribute to look at the number of rows and columns in the
df.shape
```

```
Done by Anthony
```

Out[13]: (10127, 23)

※ Cleaning Data

In [15]:
```
print('Done by Anthony')
# drop all duplicate rows and cast it to a new DataFrame df_Customers so tha
df_Customers = df.drop_duplicates()
```

```
Done by Anthony
```

In [16]:
```
print('Done by Anthony')
# call the columns attribute and iterate the columns to check if there any e
for col in df_Customers.columns:
    print(col)
```

```
Done by Anthony
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_De
pendent_count_Education_Level_Months_Inactive_12_mon_1
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_De
pendent_count_Education_Level_Months_Inactive_12_mon_2
```

In [17]:

```
print('Done by Anthony')
# drop columns of clientnum, dependent count, cardcategory, monthsinactive12
df_Customers.drop(['CLIENTNUM',
                   'Dependent_count',
                   'Card_Category',
                   'Months_Inactive_12_mon',
                   'Total_Amt_Chng_Q4_Q1',
                   'Total_Ct_Chng_Q4_Q1',
                   'Avg_Utilization_Ratio',
                   'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Cont
                   'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Cont
                  ]
                  ,axis=1, inplace=True)
```

```
Done by Anthony
```

In [18]:

```
print('Done by Anthony')
# rename columns customer age to age, monthsonbook to length of relationship
df_Customers.rename(columns={'Customer_Age': 'Age', 'Months_on_book': 'Lengt

# inspect the top 5 rows of the new DataFrame df_Customers
df_Customers.head()
```

```
Done by Anthony
```

Out[18]:

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

In [19]:

```
print('Done by Anthony')
# uniformly capitalize the first letters of the column names
df_Customers.columns = [col.capitalize() for col in df_Customers.columns]
# convert column names to string dtype
df_Customers.columns = df_Customers.columns.astype(str)

# convert strings into titlecase: first character of each word to uppercase
df_Customers.columns = df_Customers.columns.str.title()
```

```
Done by Anthony
```

In [20]:

```
print('Done by Anthony')
# call the info method to confirm titlecase and
# get additional information about our dataframe including the datatype of t
# how many missing values exist in each column
df_Customers.info()

## confirm there are no null values in the dataset
```

```
Done by Anthony
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Attrition_Flag            10127 non-null  object 
 1   Age                       10127 non-null  int64  
 2   Gender                    10127 non-null  object 
 3   Education_Level           10127 non-null  object 
 4   Marital_Status            10127 non-null  object 
 5   Income_Category           10127 non-null  object 
 6   Length_Of_Relationship    10127 non-null  int64  
 7   Total_Relationship_Count  10127 non-null  int64  
 8   Customer_Contacts         10127 non-null  int64  
 9   Credit_Limit              10127 non-null  float64
 10  Total_Revolving_Bal       10127 non-null  int64  
 11  Available_Credit          10127 non-null  float64
 12  Total_Trans_Amt           10127 non-null  int64  
 13  Total_Trans_Ct            10127 non-null  int64  
dtypes: float64(2), int64(7), object(5)
memory usage: 1.1+ MB
```

In [21]:
```
print('Done by Anthony')
# although there were no 'NaN' values, notice the 'Unknown' value in the Mar
# check Education_Level column to see if there are possibly more in other co
# return the unique values in the Education_Level column
df_Customers['Education_Level'].unique()
```

```
Done by Anthony
```

Out[21]:
```
array(['High School', 'Graduate', 'Uneducated', 'Unknown', 'College',
       'Post-Graduate', 'Doctorate'], dtype=object)
```

In [22]:
```
print('Done by Anthony')
# drop all rows with 'Unknown' values in the dataset
df_Customers = df_Customers.replace('Unknown', np.nan).dropna()
```

```
Done by Anthony
```

In [23]:
```
print('Done by Anthony')
# call the shape attribute to look at the revised number of rows and columns
# our original DataFrame df had 10,127 rows,
# by dropping 'Unknown' values, 3046 rows (about 30%) of the dataset has bee
# in addition, 9 columns have been deemed irrelevent and removed
df_Customers.shape
```

```
Done by Anthony
```

Out[23]:
```
(7081, 14)
```

In [24]:
```
print('Done by Anthony')
# write a function that takes two inputs, Total_Trans_Amt and Total_Trans_Ct
# and calculates the Avg_Trans_Amt and add new column to the DataFrame df_Cu
def calculate_Avg_Trans_Amt(df_Customers):
    df_Customers['Avg_Trans_Amt'] = df_Customers['Total_Trans_Amt']/df_Custo
    return df_Customers
    # returns the average between total and count
```

```
df_Customers = calculate_Avg_Trans_Amt(df_Customers)


# call the datatype for the df_Customers DataFrame
df_Customers.dtypes
```

```
Done by Anthony
```

Out[24]:

```
Attrition_Flag               object
Age                           int64
Gender                       object
Education_Level              object
Marital_Status               object
Income_Category              object
Length_Of_Relationship        int64
Total_Relationship_Count      int64
Customer_Contacts             int64
Credit_Limit                float64
Total_Revolving_Bal           int64
Available_Credit            float64
Total_Trans_Amt               int64
Total_Trans_Ct                int64
Avg_Trans_Amt               float64
dtype: object
```

In [25]:

```
print('Done by Anthony')
# pass in 'category' or 'float64' parameters to astype method to convert dat
conversion_dict = {'Gender': 'category',
                   'Education_Level': 'category',
                   'Marital_Status': 'category',
                   'Income_Category' : 'category',
                   'Total_Revolving_Bal': np.float64,
                   'Total_Trans_Amt': np.float64
                  }
df_Customers = df_Customers.astype(conversion_dict)
# check the datatype again to confirm
df_Customers.dtypes
```

```
Done by Anthony
```

Out[25]:

```
Attrition_Flag                object
Age                            int64
Gender                      category
Education_Level             category
Marital_Status              category
Income_Category             category
Length_Of_Relationship         int64
Total_Relationship_Count       int64
Customer_Contacts              int64
Credit_Limit                 float64
Total_Revolving_Bal          float64
Available_Credit             float64
Total_Trans_Amt              float64
Total_Trans_Ct                 int64
Avg_Trans_Amt                float64
dtype: object
```

In [26]:
```
print('Done by Anthony')
# define function vip to create a Relationship_Category years as a parameter

#   if years >= 50:return 'Diamond VIP'
#    elif 40 <= years < 50: return 'Platinum VIP'
#    elif 30 <= years < 40: return 'Gold VIP'
#    elif 20 <= years < 30:return 'Silver VIP'
#    elif 10 <= years < 20:return 'Bronze VIP'
#    else:return 'Valued Customer'

def VIP(years):
    if years >= 50:
        return 'Diamond VIP'
    elif 40 <= years < 50:
        return 'Platinum VIP'
    elif 30 <= years < 40:
        return 'Gold VIP'
    elif 20 <= years < 30:
        return 'Silver VIP'
    elif 10 <= years < 20:
        return 'Bronze VIP'
    else:
        return 'Valued Customer'

df_Customers['Relationship_Category'] = df_Customers['Length_Of_Relationship

df_Customers.head()
```

Done by Anthony

Out[26]:

	Attrition_Flag	Age	Gender	Education_Level	Marital_Status	Income_Category	Le
0	Existing Customer	45	M	High School	Married	$60K - $80K	
1	Existing Customer	49	F	Graduate	Single	Less than $40K	
2	Existing Customer	51	M	Graduate	Married	$80K - $120K	
4	Existing Customer	40	M	Uneducated	Married	$60K - $80K	
5	Existing Customer	44	M	Graduate	Married	$40K - $60K	

In [27]:
```
print('Done by Danny')
# create a mapping DataFrame to represent a custom sort
sort_mapping = {
    "Less than $40K": 0,
    "$40K - $60K": 1,
    "$60K - $80K": 2,
    "$80K - $120K": 3,
    "$120K +": 4
}
```

```
df_mapping = pd.DataFrame(list(sort_mapping.items()), columns=["Income", "So

df_mapping = df_mapping.set_index("Income")

df_mapping = df_mapping.rename(columns={"Sort_Order": "index"})

df_mapping
```

Done by Danny

Out[27]:

	index
Income	
Less than $40K	0
$40K - $60K	1
$60K - $80K	2
$80K - $120K	3
$120K +	4

In [28]:
```
print('Done by Danny')
# create a new column 'Income_Sorted' with mapped value from sort_mapping
df_Customers["Income_Sorted"] = df_Customers["Income_Category"].map(sort_map

# check the dtype of new added column 'Income_Sorted'
df_Customers['Income_Sorted'].dtypes
```

Done by Danny

Out[28]:
```
CategoricalDtype(categories=[4, 1, 2, 3, 0], ordered=False, categories_dtyp
e=int64)
```

In [29]:
```
print('Done by Danny')
# pass in 'int' parameters to astype method to convert 'Income_Sorted' datat
df_Customers['Income_Sorted'] = df_Customers['Income_Sorted'].astype(int)
```

Done by Danny

In [30]:
```
print('Done by Danny')
# reorder rows based on new column Income_Sorted in descending order
df_Customers = df_Customers.sort_values(by='Income_Sorted', ascending=False)

df_Customers = df_Customers.reset_index(drop=True)

df_Customers
```

Done by Danny

Out[30]:

	Attrition_Flag	Age	Gender	Education_Level	Marital_Status	Income_Category
0	Existing Customer	56	M	College	Married	$120K +
1	Existing Customer	57	M	Post-Graduate	Married	$120K +
2	Existing Customer	33	M	Post-Graduate	Married	$120K +
3	Attrited Customer	52	M	College	Single	$120K +
4	Existing Customer	48	M	Graduate	Married	$120K +
...	...	...	...	...	...	...
7076	Existing Customer	26	M	Graduate	Married	Less than $40K
7077	Existing Customer	39	F	Graduate	Married	Less than $40K
7078	Attrited Customer	51	F	Doctorate	Single	Less than $40K
7079	Existing Customer	37	F	Uneducated	Married	Less than $40K
7080	Attrited Customer	43	F	Graduate	Married	Less than $40K

7081 rows × 17 columns

In [31]:

```
print('Done by Danny')
# the set_index method has been called to place Income_Sorted column first a

df_Customers = df_Customers.set_index('Income_Sorted')

df_Customers
```

```
Done by Danny
```

Out[31]:

	Attrition_Flag	Age	Gender	Education_Level	Marital_Status	Income_
Income_Sorted						
4	Existing Customer	56	M	College	Married	
4	Existing Customer	57	M	Post-Graduate	Married	
4	Existing Customer	33	M	Post-Graduate	Married	
4	Attrited Customer	52	M	College	Single	
4	Existing Customer	48	M	Graduate	Married	
...	...	...	...	...	...	
0	Existing Customer	26	M	Graduate	Married	Less
0	Existing Customer	39	F	Graduate	Married	Less
0	Attrited Customer	51	F	Doctorate	Single	Less
0	Existing Customer	37	F	Uneducated	Married	Less
0	Attrited Customer	43	F	Graduate	Married	Less

7081 rows × 16 columns

In [32]:

```
print('Done by Anthony')
# pivot table to get a horizontal view of the data column
#  output mean of the values as age columns as gender
pivot_table = df_Customers.pivot_table(values='Age', columns='Gender', aggfu

pivot_table
```

Done by Anthony

Out[32]:

Gender	F	M
Age	46.436741	46.266595

In [33]:

```
print('Done by Danny')
# another pivot table to get a horizontal view of the data column to see the
#values length of relationship, columns attrition flag, aggfunc median
pivot_table_median = df_Customers.pivot_table(values='Length_Of_Relationship
                                              columns='Attrition_Flag',
                                              aggfunc='median')
```

```
print(pivot_table_median)
df_Customers.shape
```

```
Done by Danny
Attrition_Flag          Attrited Customer  Existing Customer
Length_Of_Relationship               36.0               36.0
```

Out[33]: (7081, 16)

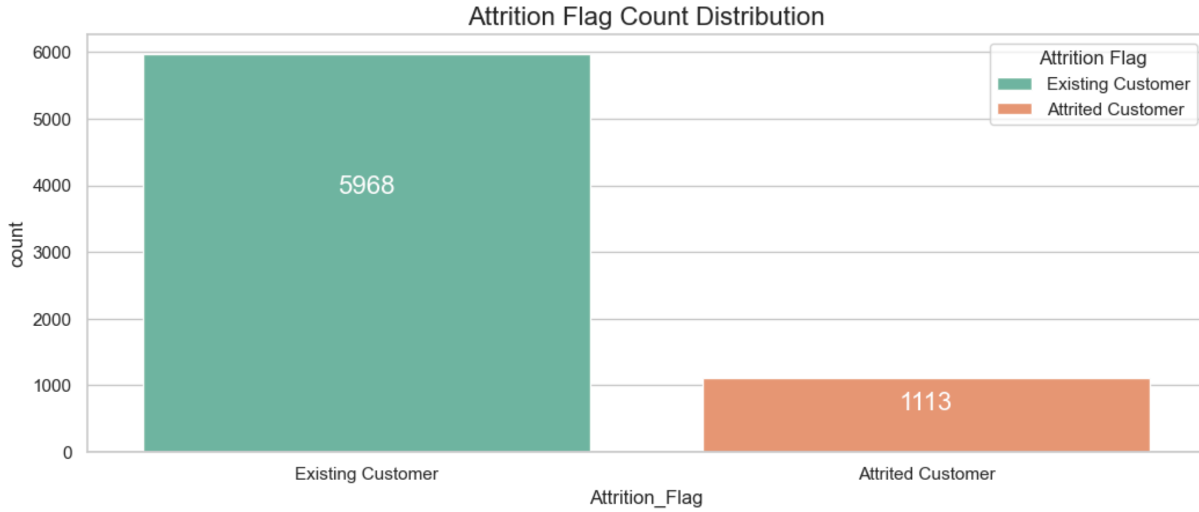
📝 Note that at the end of all the data cleaning, we now have 7081 rows x 17 columns.

※ Data Visualization

In [36]:

```
print('Done by Danny')
# visualize the difference in the number of the 'Attrition_Flag' data by cal
# set a seaborn style
sns.set(style="whitegrid")

# set the figure size
plt.figure(figsize=(12, 6))

# plot with hue and palette details x as attrition flag, hue as attrition fl
ax = sns.countplot(x='Attrition_Flag', data=df_Customers, hue='Attrition_Fla


# annotate each countplot bar with its value count
for p in ax.patches:
    count = int(p.get_height())
    y_position = p.get_height() * 2 / 3
    ax.annotate(f'{count}',
                (p.get_x() + p.get_width() / 2., y_position),
                ha='center',
                va='center',
                fontsize=16,
                color='white',
                xytext=(0, 0),
                textcoords='offset points')

plt.title("Attrition Flag Count Distribution", fontsize=16)

plt.legend(title='Attrition Flag', loc='upper right', labels=['Existing Cust

plt.tight_layout(pad=5.0)

plt.show()
```

```
Done by Danny
```



✅ Conclusion

The countplot shows us visually that the number of attrited customers is approximately 1/6 (16.67%) of the existing customers.

As noted above, there are 7081 rows of values in each column. When we divide 1113 (Attrited Customers) by 7081 (total count), we get 15.72%, which is very close to our visual estimation! 😀

❓ Question

1. What are the demographics of the customers?

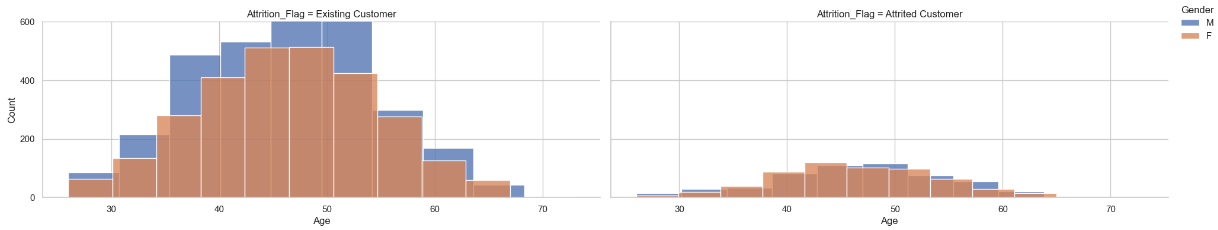
In [39]:
```
print('Done by Danny')
# call the value_counts function on the column Series 'Gender' to find the c
gender_counts = df_Customers['Gender'].value_counts()
print(gender_counts)

## the male and female counts are fairly equal
```

```
Done by Danny
Gender
M    3706
F    3375
Name: count, dtype: int64
```

In [40]:
```
print('Done by Danny')
# set a seaborn style
sns.set(style="whitegrid")

# plot a histogram FacetGrid to show the age of customers separately by genc
#col as attrition flag, hue as gender, hue order 'M','F'
g = sns.FacetGrid(df_Customers, col="Attrition_Flag", hue="Gender", hue_orde

g.map(sns.histplot, "Age", kde=False, bins=10, multiple="dodge")

g.set(ylim=(0, 600))
plt.yticks(range(0, 601, 200))
```

```
g.add_legend(title="Gender", label_order=["M", "F"], loc='upper right')

plt.show()
```

```
Done by Danny
```



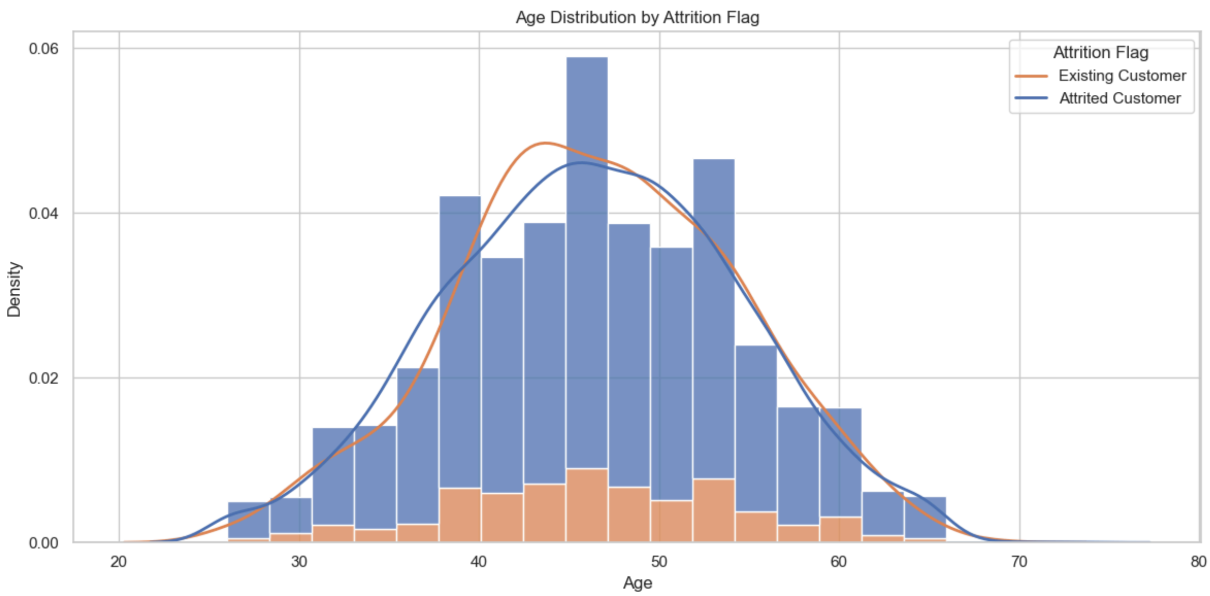
✅ Conclusion

The histogram shows the customers' age and gender graphed separately by two separate plots: Existing Customer and Attrited Customer.

In [42]:

```
print('Done by Danny')
# set seaborn style
sns.set(style="whitegrid")

plt.figure(figsize=(12, 6))

# form a facetgrid using  with a 'Atttitionrflag' hue
sns.histplot(df_Customers, x="Age", hue="Attrition_Flag", stat="density", bi

# map the above form facetgrid with 'Age' attributes
sns.kdeplot(data=df_Customers, x="Age", hue="Attrition_Flag", common_norm=Fa

plt.xlabel("Age")
plt.ylabel("Density")
plt.title("Age Distribution by Attrition Flag")

plt.yticks([i * 0.02 for i in range(4)])

plt.legend(title="Attrition Flag", loc="upper right", labels=['Existing Cust

plt.tight_layout()
plt.show()

### what are the number on the y-axis?
```

```
Done by Danny
```



✅ Conclusion

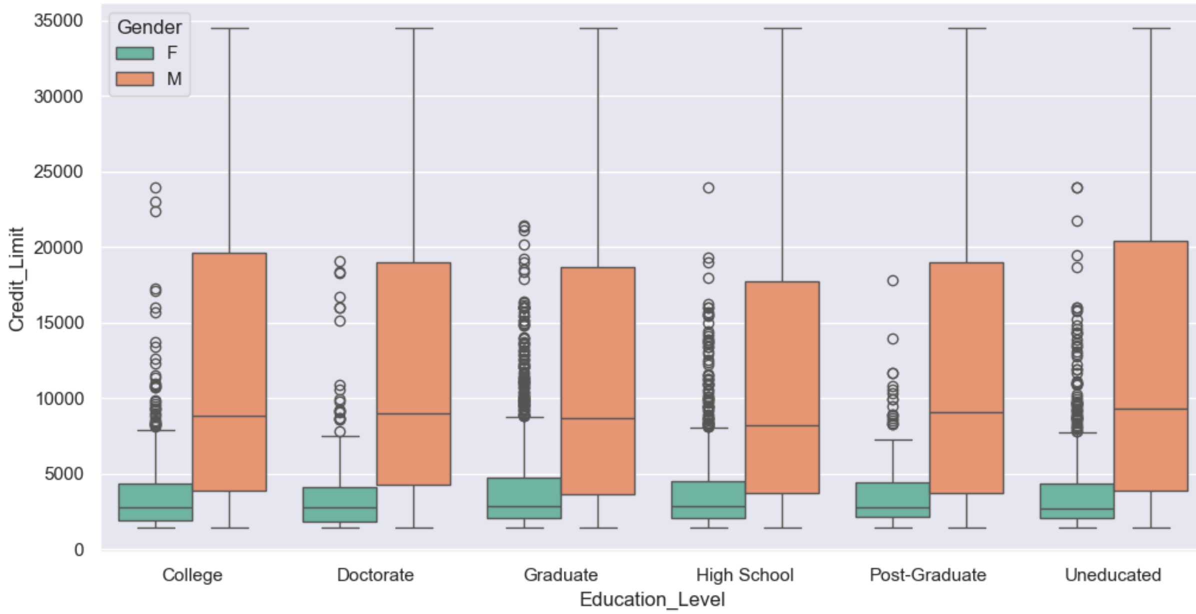
The distribution plot shows that the ages of existing and attrited customers are very similar. Majority of the customers are between 35 to 55 years old.

❓ Question

2. Does education level impact the credit line of the customers?

In [45]:
```
print('Done by Anthony')
# reset default seaborn theme, scaling, and color palette
sns.set()

# set a seaborn style
sns.set(style='whitegrid')

# you set the context to a notebook
sns.set(context='notebook')

# set the figure size
plt.figure(figsize=(12, 6))

# call the boxplot with our DataFrame as data. Give it x-axis, y-axis, and a
#x education level, y credit limit, hue gender
sns.boxplot(
    data=df_Customers,
    x='Education_Level',
    y='Credit_Limit',
    hue='Gender',
    palette='Set2'
)
plt.show()
```

```
Done by Anthony
```



✅ Conclusion

*This boxplot shows the shocking disparity of the credit limits assigned to the female and male customers. Across the board, the female customers were given far less credit limits than male customers, no matter how educated they were.

Despite the gender differences, the credit limit was very similar with no regard to the education levels of the customers .

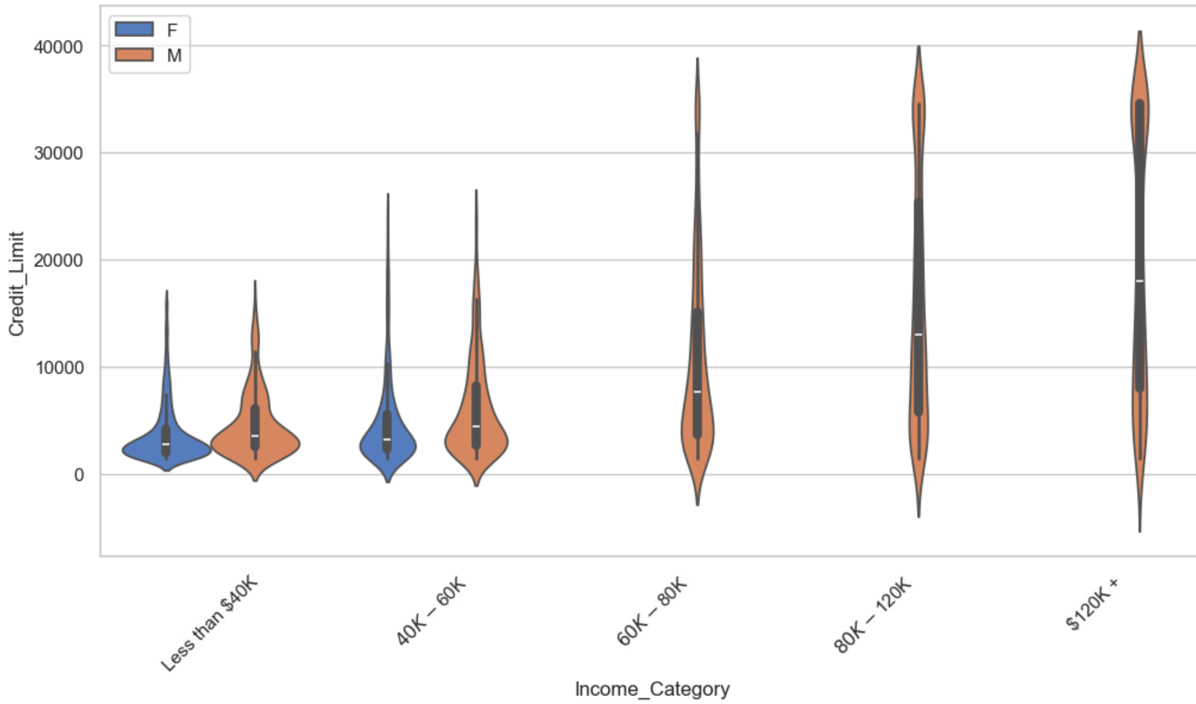
❓ Question

3. How do female and male customers compare in reported income and are customers with higher income given higher credit limits?

In [48]:
```
print('Done by Anthony')
# reset default seaborn theme, scaling, and color palette
sns.set()

# set style to plot conditional relationships
sns.set_style("whitegrid")

# set the figure size
plt.figure(figsize=(12, 6))

# call the violinplot with df_Customers as data. Give it x-axis, y-axis, and
sns.violinplot(
    data=df_Customers,
    x='Income_Category',
    y='Credit_Limit',
    hue='Gender',
    palette='muted',
    order=['Less than $40K', '$40K - $60K', '$60K - $80K', '$80K - $120K', '
)
#x income category, y credit limit, hue gender

                 # sort the order of the Income_Category categorical string
```

```
# call the set_rotation method to rotate x-axis labels to avoid overlapping
plt.xticks(rotation=45)
plt.legend(loc='upper left')
plt.tight_layout
plt.show()
```

```
Done by Anthony
```



✅ Conclusion

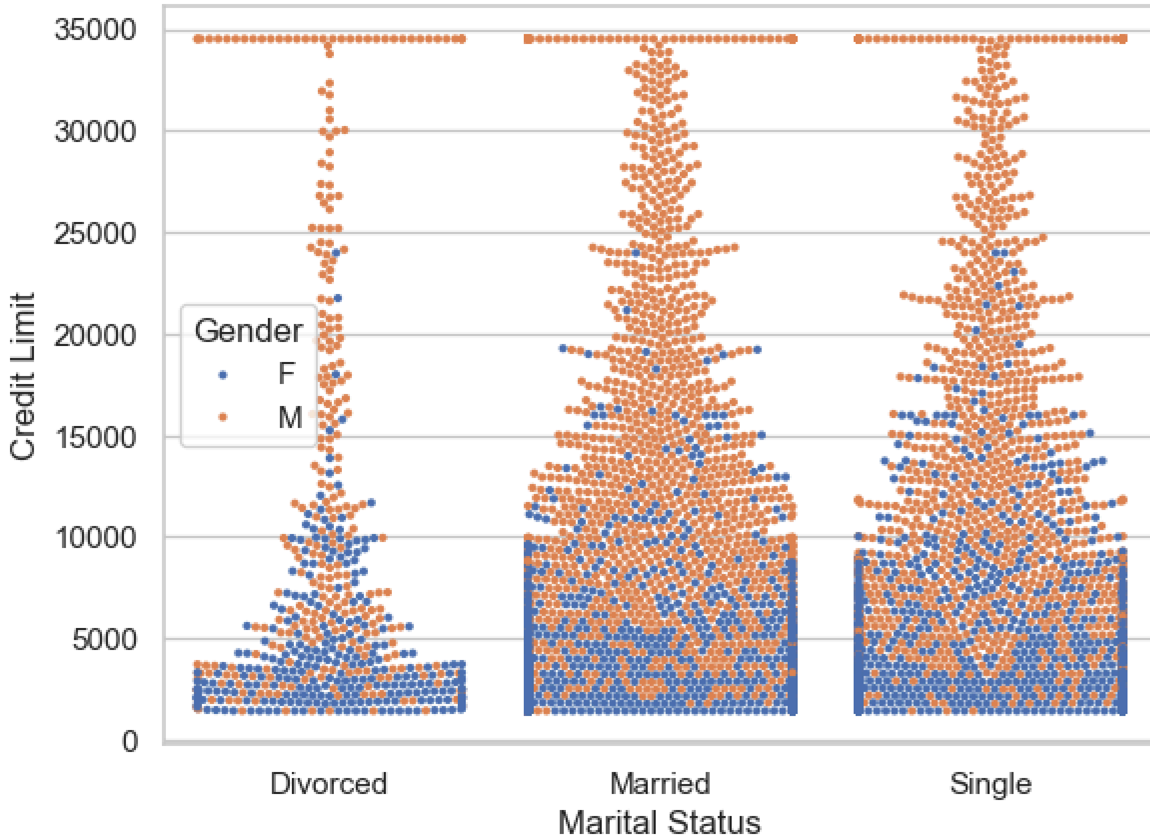
As expected the income reported by the male and female customers were consistent to their income. The customers with the higher income were given high credit limits. 😱

❓ Question

4. Is it possible that the highest credit limits being given to mostly males is due to household income reported for the husband's income in married couples?

In [51]:
```
print('Done by Anthony')
# call the swarmplot to visualize the credit limits for the different marita
# set the figure size

sns.swarmplot(data=df_Customers, x='Marital_Status', y='Credit_Limit', hue='
plt.xlabel('Marital Status')
plt.ylabel('Credit Limit')
plt.legend(title='Gender')
plt.figure(figsize=(12, 6))
plt.show()
```

```
Done by Anthony
```




```
<Figure size 1200x600 with 0 Axes>
```

✅ Conclusion

It is sad to say the huge disparity between the income reported by the male and female customers was NOT due to household income reported for the husband's income for married customers. Males were given 2-3 times higher credit limits no matter their marital status! 😤

※ Descriptive Statistics

❓ Question

5. Is it less likely for customers to leave if they have multiple products with one financial institution?

In [55]:
```
print('Done by Anthony')
# return the unique values in the Education_Level column
df_Customers['Education_Level'].unique()
```

```
Done by Anthony
```

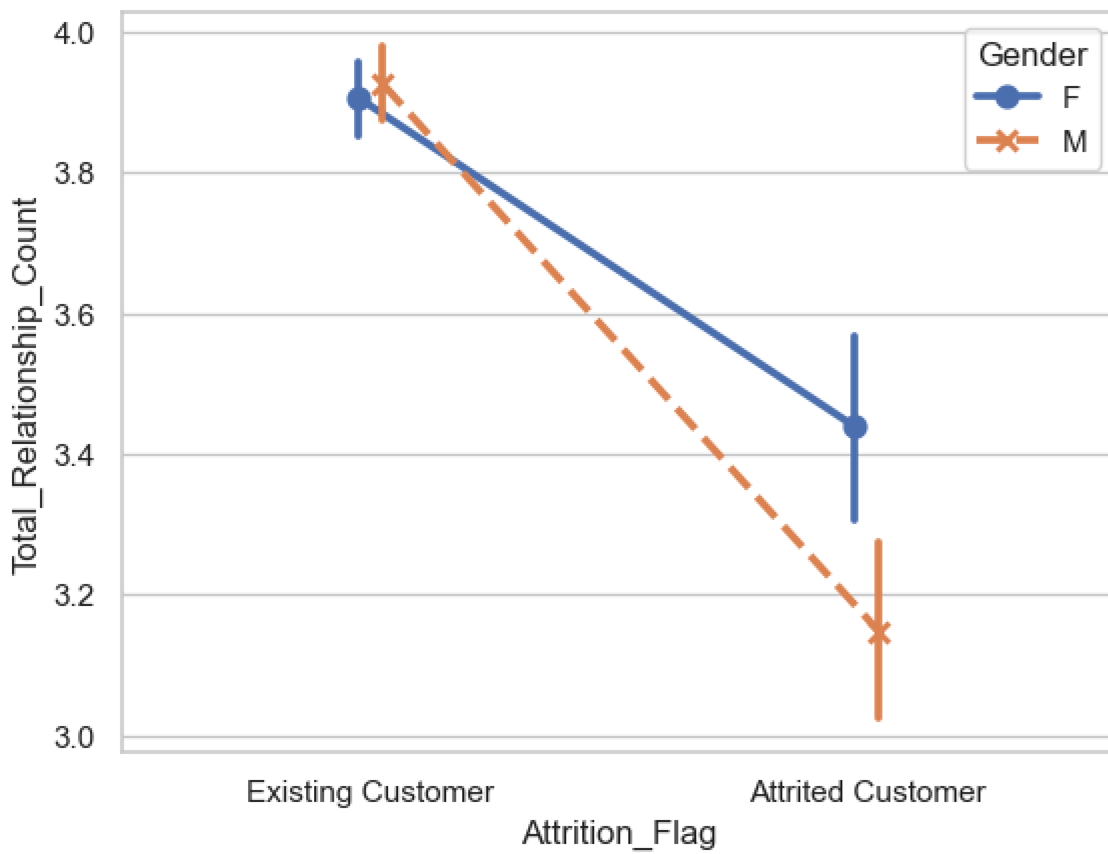
Out[55]:
```
['College', 'Post-Graduate', 'Graduate', 'High School', 'Uneducated', 'Doct
orate']
Categories (6, object): ['College', 'Doctorate', 'Graduate', 'High School',
'Post-Graduate', 'Uneducated']
```

In [56]:
```
print('Done by Anthony')
# plot the point plot to show estimate and confidence interval, connecting p
```

```
sns.pointplot(
    data=df_Customers,
    x='Attrition_Flag',
    y='Total_Relationship_Count',
    hue='Gender',
    markers=["o", "x"],
    linestyles=["-", "--"],
    errorbar='ci',
    dodge=True
)
plt.figure(figsize=(5, 5))
#x as attrition flag, y as totalrelationship count,

            # Separate the points for different hue levels along the categor
```

```
Done by Anthony
```

Out[56]: `<Figure size 500x500 with 0 Axes>`




```
<Figure size 500x500 with 0 Axes>
```

✅ Conclusion

*Males with approximately less than 3.3 and females with approximately less than 3.6 multiple products show the greater numbers of customer churning. It seems more attrited male customers had less multi-products than females, we can also see that more males had slightly higher number of multi-products.

❓ Question

6. How does customer's balance and usage affect attrition?

In [59]:
```
print('Done by Anthony')
# create a new DataFrame of just the columns of customer's credit card balar
df_Credit = df_Customers[['Total_Trans_Ct','Total_Trans_Amt','Avg_Trans_Amt'

#  let's get some statistics in the new DataFrame use describe
df_Credit.describe()
```

Done by Anthony

Out[59]:

	Total_Trans_Ct	Total_Trans_Amt	Avg_Trans_Amt	Total_Revolving_Bal
count	7081.000000	7081.000000	7081.000000	7081.000000
mean	64.503319	4394.299816	62.543610	1167.501624
std	23.809330	3468.461606	26.923696	812.315606
min	10.000000	510.000000	19.137931	0.000000
25%	44.000000	2089.000000	46.807229	463.000000
50%	67.000000	3831.000000	55.530864	1282.000000
75%	80.000000	4740.000000	65.486842	1781.000000
max	134.000000	17995.000000	190.193182	2517.000000

In [ ]:
```
print('Done by Anthony')
# pairplots are used to plot multiple bivariate distributions in a dataset
# showing relationship for combination of variables in a DataFrame as a matr
# and the diagonal plots are the univariate plots.

# create a pairplot of the customer's credit card balance and usage

               # as the upper triangle plots mirror the lower triangle ones,
sns.pairplot(
    data=df_Customers,
    vars=['Total_Trans_Ct', 'Total_Trans_Amt', 'Avg_Trans_Amt','Total_Revolv
    hue='Attrition_Flag',
    corner=True,
    diag_kind='kde'
    )
```

Done by Anthony

Out[ ]: <seaborn.axisgrid.PairGrid at 0x16a989a30>

✅ Conclusion

*By visualizing multiple plots, we can make multiple conclusions with one plot:

- Attrited customers had lower total transaction count, average transaction, total transaction amount, and total balance compared to the existing customers

*In comparing credit card customer's actitivites and balance, we can easily compare the customer credit card balance and usage in customer attrition. I can conclude that attrited customers overall used their credit card less, spent less money per transaction and total transaction, and carried a lower balance.

In [ ]:
```
print('Done by Anthony')
# generate a new DataFrame for Attrited Customer
df_Attrited_Customers = df_Customers[df_Customers['Attrition_Flag'] == 'Attr

# use only the Credit_Limit column
df_Credit_Limit = df_Attrited_Customers[['Credit_Limit']]

# pass it to another data frame
df_Credit_Limit.head()
```

In [ ]:
```
print('Done by Anthony')
# ECDF (Emperical Cumulative Distribution Function) is a step function used
# The advantage of using an ECDF plot over histograms is that ECDF are immur
# ECDF can also answer what percentage of the data is under the specific val

# create our ECDF function
def ecdf(data):
    """Compute ECDF for a one-dimensional array of measurements."""
    # sort data from the lowest value to the highest value
    # Number of data points: n
    n = len(data)

    # x-data for the ECDF: x
    x = np.sort(data)

    # the y-axis is set to evenly spaced data points generated by using the
    # with a maximum of one (which is 100%) and then divding by the total nu
    y = np.arange(1, n+1) / n
    return x, y


# create the x and y axis for the column 'Credit_Limit'
x, y = ecdf(df_Credit_Limit['Credit_Limit'])

# set seaborn style
sns.set(style='whitegrid')

# set the figure size
plt.figure(figsize=(10, 6))

# set the ecdf
plt.plot(x, y, marker=".", linestyle="none")

# use the percentile method to get the array of 2.5, 25, 50, 75, 97.5 percer
percentiles = [2.5, 25, 50, 75, 97.5]
perc_values = np.percentile(df_Credit_Limit, percentiles)

# overlay percentiles as red diamonds
plt.plot(perc_values, [0.025, 0.25, 0.5, 0.75, 0.975], 'rD', label='Percenti

# give it a title and add the labels to x and y axes
plt.title('ECDF of Credit Limits of Attrited Customers')
plt.xlabel('Credit Limit (USD)')
plt.ylabel('ECDF')
```

```
# use the plt.margins function to make sure none of the data points run over
# choosing a value of point .02 gives a 2% buffer all around the plot
plt.margins(0.02)

# show the plot
plt.show()
```

✅ Conclusion

Some information we can get from the ECDF are:

- Approximately 75% of the attrited customers have a credit limit less than 10,000 USD.
- Top 25% percent have the greatest spread of credit limits, between 10,000 USD to 35,000 USD.
- Majority of the attrited customer has a credit limit of less than 5000 USD.

In [ ]:
```
print('Done by Anthony')
# compute the mean with np.mean method and pass in the array element Attrite
np.mean(df_Credit_Limit)
```

In [ ]:
```
print('Done by Anthony')
# compute the median with np.median method and pass in the array element Att
np.median(df_Credit_Limit)
```

In [ ]:
```
print('Done by Anthony')
# compute the variance with np.var method and pass in the array element Attr
np.var(df_Credit_Limit)
```

In [ ]:
```
print('Done by Anthony')
# compute the standard deviation with np.std method and pass in the array el
np.std(df_Credit_Limit)
```

※ Quantitative Data Exploratory Descriptive

In [ ]:
```
print('Done by Danny')
# Correlation is the concept of linear relationship between two variables.
sns.set(style="whitegrid")
# use the lmplot to plot data and linear regression between two variables
lmplot = sns.lmplot(
    data=df_Customers,
    x="Total_Revolving_Bal",
    y="Total_Trans_Amt",
    hue="Attrition_Flag",
    palette={"Existing Customer": "blue", "Attrited Customer": "green"},
    markers=["o", "s"],
    height=6,
    aspect=1.5,
    scatter_kws={"s": 50, "alpha": 0.6},
```

```
    line_kws={"linewidth": 2},
)

lmplot.set_axis_labels("Total_Revolving_Bal", "Total_Trans_Amt")
plt.title("Relationship Between Total Revolving Balance and Total Transactio
plt.legend(title="Attrition_Flag", loc="upper right")
```

In [ ]:
```
print('Done by Danny')
# use the correlation method to calculate the correlation between 'Total_Rev
columns_of_interest = ['Total_Revolving_Bal', 'Total_Trans_Amt']

correlation_matrix = df_Customers[columns_of_interest].corr()

print(correlation_matrix)
```

✅ Conclusion

*The horizontal regression line for both existing and attrited customers indicates Total_Revolving_Bal does not seem to be a good predictor of the Total_Trans_Amt. Furthermore, the data points are grouped into 3 sections but very scatteredaway from the fitted line. This shows lots of variability and refutes the reliability of the variable.

*The correlation between 'Total_Trans_Amt' and 'Total_Revolving_Bal' is approximately 0.063782. After reviewing the lmplot and the correlation method, we can say that they have a weak or no linear relationship.

In [ ]:
```
print('Done by Danny')
# regplot performs a simple linear regression model fit and plots the data
# jointplot can use regplot to show linear regression fit on the joint axes
# use the correlation method to calculate the correlation between 'Total_Rev
sns.set(style="whitegrid")

plot = sns.jointplot(
    df_Customers,
    x="Avg_Trans_Amt",
    y="Total_Trans_Ct",
    kind="reg",
    height=8,
    color="magenta",
    scatter_kws={'alpha': 0.3},
)

plt.show()
```

In [ ]:
```
print('Done by Danny')
# use the correlation method to calculate the correlation between 'Avg_Trans
columns_of_interest = ['Avg_Trans_Amt', 'Total_Trans_Ct']

correlation_matrix = df_Customers[columns_of_interest].corr()

print(correlation_matrix)
```

✅ Conclusion

The rising regression line for the credit card customers indicates Avg_Trans_Amt is positively correlated to the Total_Trans_Ct. The Avg_Trans_Amt may be a pretty good predictor of Total_Trans_Ct since the regression line is almost diagonal. Furthermore, the correlation between them is approximately 0.812263.

After reviewing the regplot and the correlation method, we can say that 'Avg_Trans_Amt' and 'Total_Trans_Ct' have a strong positive correlation.

In [ ]:
```
print('Done by Danny')
# use the regplot to plot data and linear regression between two variables x
plt.figure(figsize=(10, 6))

sns.regplot(
    x='Total_Relationship_Count',
    y='Total_Trans_Amt',
    data=df_Customers,
    scatter_kws={'s':10},
    line_kws={'color':'yellow'}
)

plt.title('Linear Regression between Total_Relationship_Count and Total_Trar

plt.show()
```

In [ ]:
```
print('Done by Danny')
# use the correlation method to calculate the correlation between 'Total_Rel
columns_of_interest = ['Total_Relationship_Count', 'Total_Trans_Amt']

correlation_matrix = df_Customers[columns_of_interest].corr()

print(correlation_matrix)
```

✅ Conclusion

The declining regression line for the credit card customers indicates Total_Relationship_Count is negatively correlated to the Total_Trans_Amt. The Total_Relationship_Count is likely not a good predictor of Total_Trans_Amt. Furthermore, the correlation between them is approximately 0.0.348024.

After reviewing the regplot and the correlation method, we can say that 'Total_Relationship_Count' and 'Total_Trans_Amt' have a slightly negative correlation.

※ Testing Hypothesis

In [ ]:
```
print('Done by Danny')
# Covariance is a measure of how two variables are related to each other; hc
#do this for avg trans amt and total trans amt
cov_matrix = df_Customers[['Avg_Trans_Amt', 'Total_Trans_Amt']].cov()

print(cov_matrix)
```

✅ Conclusion

The covariance of Avg_Trans_Amt and Total_Trans_Amt are positively correlated because 8.53431928e+04 is greater than 0.

💠 Normal Test 1

- **H_0** : Distribution is Normal
- **H_1** : Distribution is not Normal

In [ ]:
```
print('Done by Danny')
# call the normaltest in the statistics package to test if the Avg_Trans_Amt
stat, p_value = normaltest(df_Customers['Avg_Trans_Amt'])

# indicate the significance level (denoted as α or alpha)
alpha = 0.05

# print results
print("Normal Test Results:")
print(f"    statistics = {stat}")
print(f"    p_value = {p_value}")

# null hypothesis: x comes from a normal distribution
if p_value < alpha:
    print("The null hypothesis can be rejected: The distribution is not norm
else:
    print("The null hypothesis cannot be rejected: The distribution is norma
```

In [ ]:
```
print('Done by Danny')
# utilize plt.rcParams to help me to set the figure width width of 10 and he
plt.rcParams['figure.figsize'] = (10, 6)

# set style
sns.set(style="whitegrid")

# set context, font scale and font size to enhance plot
sns.set_context("notebook", font_scale=1.2)

# plot distplot to show the distribution
# fit will plot a normal curve to the distribution histogram
# set kde to false because kde is set by default
sns.histplot(df_Customers['Avg_Trans_Amt'], kde=True, stat="density", bins=3

# add title and xlabel to the plot
plt.title("Distribution of Average Transaction Amount", fontsize=16)
plt.xlabel("Average Transaction Amount", fontsize=14)

plt.show()
```

✅ Conclusion

A small p-value (typically ≤ 0.05) indicatges strong evidence against the H_0 , so you reject the null hypothesis.

Therefore, the histogram clearly shows the Avg_Trans_Amt is right-skewed. As the p-value (0) is less than 0.05 (the significance level), the null hypothesis is false and can be rejected.

💠 Normal Test 2

In [ ]:
```
print('Done by Danny')
# call the normaltest in the statistics package to test if the Customer_Cont
stat, p_value = stats.normaltest(df_Customers['Customer_Contacts'])

# indicate the significance level (denoted as α or alpha)
alpha = 0.05

# print results
print(f"Normal Test Results:\n    Statistics = {stat}\n    p-value = {p_valu

 # null hypothesis: x comes from a normal distribution
if p_value < alpha:
    print("The null hypothesis can be rejected: The data does not come from 
else:
    print("The null hypothesis cannot be rejected: The data may come from a 
```

In [ ]:
```
print('Done by Anthony')
# utilize plt.rcParams to help me to set the figure width width of 10 and he
plt.rcParams['figure.figsize'] = [10, 6]

# set style
sns.set_style("whitegrid")

# set context, font scale and font size to enhance plot
sns.set_context("notebook", font_scale=1.2)

# plot distplot to show the distribution
# fit will plot a normal curve to the distribution histogram
# set kde to false because kde is set by default
sns.histplot(df_Customers['Customer_Contacts'], kde=True, stat="density",bir

# add title and xlabel to the plot
plt.title("Histogram Distribution of Customer Contacts")
plt.xlabel("Customer Contacts")

plt.show()
```

✅ Conclusion

A large p-value (> 0.05) indicates weak evidence against the H_0 , so you fail to reject the null hypothesis.

Therefore, the histogram shows a fairly normal distribution of Customer_Contacts. As the p-value (0.6621785833391884) is greater than 0.05 (the significance level), the null hypothesis is true.

💠 Chi-Squared Test

In [ ]:
```
print('Done by Anthony')
# test whether two categorical variables are related or independent.
from scipy.stats import chi2_contingency
# Example of the Chi-Squared Test
```

```
stat, p, dof, expected = chi2_contingency(df_Customers['Avg_Trans_Amt'], df_
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
   print('Probably independent')
else:
  print('Probably dependent')
# print
```

Pearson's Correlation Test

In [ ]:
```
print('Done by Anthony')
from scipy.stats import pearsonr
# test whether two samples have a linear relationship.
stat, p = pearsonr(df_Customers['Avg_Trans_Amt'], df_Customers['Total_Trans_
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
  print('Probably independent')
else:
 print('Probably dependent')
```

ANOVA

In [ ]:
```
print('Done by Danny')
#extract the Length_Of_Relationship and Avg_Trans_Amt data
data = df_Customers[['Length_Of_Relationship', 'Avg_Trans_Amt', 'Income_Cate

#group the data by Income_Category
grouped_data = [group['Avg_Trans_Amt'].values for name, group in data.groupb

#use the f_oneway method, built-in function of the scipy package, to perform
#pass in the price data of the two car make groups that we want to #compare,
anova_result = stats.f_oneway(*grouped_data)

print(f"ANOVA results: F={anova_result}")
if p < 0.05:
    print("Reject null hypothesis: There is a significant difference betweer
else:
    print("Fail to reject null hypothesis: No significant difference betweer
```

✅ Conclusion

The average transaction amount between the highest (120K USD+) and the lowest (Less than 40K USD) income categories were not significantly different, as the F-test score is very low and p-value is larger than 0.05.

Z Test

Since my dataset is larger than 30, z-test will be used over the t-test, to test a hypothesis about the population mean. By the virtue of limit central theorem, one sample z-test to do the test.

Testing the hypothesis that the mean is 88 against the alternative that it is not

$\begin{align} & {{H}_{0}}:\,\mu =\,{{\mu }_{0}} \\ & {{H}_{1}}:\,\mu \ne {{\mu }_{0}} \\ \end{align}$

In [ ]:
```
print('Done by Danny')
data = df_Customers['Avg_Trans_Amt']

mu = 88

sample_mean = np.mean(data)
sample_std = np.std(data, ddof=1)  # Use ddof=1 for sample standard deviatic
sample_size = len(data)

t_statistic, p_value = stats.ttest_1samp(data, mu)

print(f"The test statistic is: {t_statistic:.5f}")
print(f"The p-value is: {p_value:.5f}")

if p_value < 0.05:
    print("The null hypothesis is rejected, meaning the mean is significantl
else:
    print("The null hypothesis cannot be rejected, meaning the mean is not s
```

✅ Conclusion

The p-value is 0. At alpha = 0.05 level of significance, we can reject the null hypothesis and conclude that the mean Avg_Trans_Amt is not 88.

Testing the hypothesis that the mean is 88 against the alternative that it is GREATER

$\begin{align} & {{H}_{0}}:\mu <={{\mu }_{0}} \\ & {{H}_{1}}:\mu >{{\mu }_{0}} \\ \end{align}$

In [ ]:
```
print('Done by Danny')
data = df_Customers['Avg_Trans_Amt']

mu = 88

t_statistic, p_value = stats.ttest_1samp(data, mu)

if t_statistic > 0:
    p_value = p_value / 2

print(f"The test statistic is: {t_statistic:.5f}")
print(f"The p-value is: {p_value:.5f}")

if p_value < 0.05:
    print("The null hypothesis is rejected, meaning the mean is significantl
else:
    print("The null hypothesis cannot be rejected, meaning the mean is not s
```

Testing the hypothesis that the mean is 88 against the alternative that it is SMALLER

$\begin{align} & {{H}_{0}}:\mu >={{\mu }_{0}} \\ & {{H}_{1}}:\mu < {{\mu }_{0}} \\ \end{align}$

In [ ]:
```
print('Done by Danny')
data = df_Customers['Avg_Trans_Amt']

mu = 88

t_statistic, p_value = stats.ttest_1samp(data, mu)

if t_statistic > 0:
    p_value = p_value / 2

print(f"The test statistic is: {t_statistic:.5f}")
print(f"The p-value is: {p_value:.5f}")

if p_value < 0.05:
    print("The null hypothesis is rejected, meaning the mean is significantl
else:
    print("The null hypothesis cannot be rejected, meaning the mean is not s
```

※ Summary and Conclusion

**This dataset of credit card customers was really insightful. There were so many demographic and customer activity information. It was interesting how some information I thought would be a good correlation had no effect on each other. For example, I thought the total transaction count and total revolving balance would be related, but surprised to find they were independent of each other.

**It would be very difficult to predict churning customers without doing more research and finding out what else the customers want or need. The bank may run some promotion to retain their customers. It would be great to set-up some promotion and see if that helps to increase more cross-selling opportunities, as it seems customers that have more products tend to continue their relationship with the bank.

**Finally, the most surprising aspect of this analysis is how low the income has been reported for females and how few females have accounts at this bank. I think they really should focus on taking advantage of tapping into this market.