In [24]:

```python
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score

from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
import time
```

LOAD DATA

In [25]:

```python
start_time = time.time()
bank = pd.read_csv("BankChurners.csv")
bank.head()
```

Out[25]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category
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

Goal of the project: to train a classifier that can predict the *Attrition_Flag* column based on the other columns

Catergories of dataset:

1. **CLIENTNUM**: Unique identifier for each customer. (Integer)
2. **Attrition_Flag**: Flag indicating whether or not the customer has churned out. (Boolean)
3. **Customer_Age**: Age of customer. (Integer)
4. **Gender**: Gender of customer. (String)
5. **Dependent_count**: Number of dependents that customer has. (Integer)
6. **Education_Level**: Education level of customer. (String)
7. **Marital_Status**: Marital status of customer. (String)
8. **Income_Category**: Income category of customer. (String)
9. **Card_Category**: Type of card held by customer. (String)
10. **Months_on_book**: How long customer has been on the books. (Integer)
11. **Total_Relationship_Count**: Total number of relationships customer has with the credit card provider. (Integer)
12. **Months_Inactive_12_mon**: Number of months customer has been inactive in the last twelve months. (Integer)
13. **Contacts_Count_12_mon**: Number of contacts customer has had in the last twelve months. (Integer)
14. **Credit_Limit**: Credit limit of customer. (Integer)
15. **Total_Revolving_Bal**: Total revolving balance of customer. (Integer)
16. **Avg_Open_To_Buy**: Average open to buy ratio of customer. (Integer)
17. **Total_Amt_Chng_Q4_Q1**: Total amount changed from quarter 4 to quarter 1. (Integer)
18. **Total_Trans_Amt**: Total transaction amount. (Integer)
19. **Total_Trans_Ct**: Total transaction count. (Integer)
20. **Total_Ct_Chng_Q4_Q1**: Total count changed from quarter 4 to quarter 1. (Integer)
21. **Avg_Utilization_Ratio**: Average utilization ratio of customer. (Integer)
22. **Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months**
    Naive Bayes classifier for predicting whether or not someone will churn based on characteristics such
23. **Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months**
    Naive Bayes Classifier Attrition Flag Card Category Contacts Count 12 Mon Dependent Count Education Level Months Inactive 12 Mon 2

EXPLORATORY PROCESS

In this section, we will check on the quality of dataset to see whether it has missing values or any categorical variables that need to be turned to numeric ones.

In [26]:
```
unique_values_data = []

for col in bank.columns:
    # Get the unique values in the column
    num_unique_vals = bank[col].nunique()
    unique_values_data.append([col, num_unique_vals])

# Create a DataFrame with the column names and unique value counts
unique_values_df = pd.DataFrame(unique_values_data, columns=['Column', 'Number of Unique Values'])
unique_values_df
```

Out[26]:

	Column	Number of Unique Values
0	CLIENTNUM	10127
1	Attrition_Flag	2
2	Customer_Age	45
3	Gender	2
4	Dependent_count	6
5	Education_Level	7
6	Marital_Status	4
7	Income_Category	6
8	Card_Category	4
9	Months_on_book	44
10	Total_Relationship_Count	6
11	Months_Inactive_12_mon	7
12	Contacts_Count_12_mon	7
13	Credit_Limit	6205
14	Total_Revolving_Bal	1974
15	Avg_Open_To_Buy	6813
16	Total_Amt_Chng_Q4_Q1	1158
17	Total_Trans_Amt	5033
18	Total_Trans_Ct	126
19	Total_Ct_Chng_Q4_Q1	830
20	Avg_Utilization_Ratio	964
21	Naive_Bayes_Classifier_Attrition_Flag_Card_Cat...	1704
22	Naive_Bayes_Classifier_Attrition_Flag_Card_Cat...	640

In [27]:
```
# Check for duplicate row
bank[bank.duplicated(keep=False)]
```

Out[27]:

CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	M

0 rows × 23 columns

Comment: Luckily we don't have any duplicate row

In [28]:
```
# Check for duplicate ID
bank.duplicated(subset=['CLIENTNUM']).unique()
```

Out[28]:
```
array([False])
```

Comment: There is no duplicate ID too

In [29]:
```
# Drop the last 2 columns
bank = bank.drop(['Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Ed
        'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Le
bank.head()
```

Out[29]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category
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 × 21 columns

In [30]:

```python
# Get more info to see how much data is missing
bank.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
```

Comment: It seems like there is no missing value in this dataset. However, the missing values in this case are labeled *Unknown*. We will find in all columns.

In [31]:

```python
# Count "Unknown" values in all columns
bank[bank == 'Unknown'].count()
```

Out[31]:

```
CLIENTNUM                      0
Attrition_Flag                 0
Customer_Age                   0
Gender                         0
Dependent_count                0
Education_Level             1519
Marital_Status               749
Income_Category             1112
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
```

Comment:

- When we changed to look for *Unknown* value, we can see there are 3 columns having it, accounting for over 10%. Our next action is to deal with the unknown values.
- Understanding the drawback of removing all missing values which are loss of information, bias, and impact on relationships between variables, we decided to replace all missing values with the most requent value in mentioned columns

In [32]:
```
for col in ['Education_Level', 'Marital_Status', 'Income_Category']:
    mode = bank[col].mode()[0]
    bank[col] = bank[col].replace('Unknown', mode)

# Count "Unknown" values in all columns again
bank[bank == 'Unknown'].count()
```

Out[32]:
```
CLIENTNUM                   0
Attrition_Flag              0
Customer_Age                0
Gender                      0
Dependent_count             0
Education_Level             0
Marital_Status              0
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
```

Comment: In bankchurners dataset, we have 'Gender', 'Education_Level', 'Marital_Status', 'Income_Category', and 'Card_Category' are the categorical columns with string values and 'Attrition_Flag' is also a categorical column, but it can be handled separately as a binary variable. For the others, we will encode these categorical columns using label encoding.

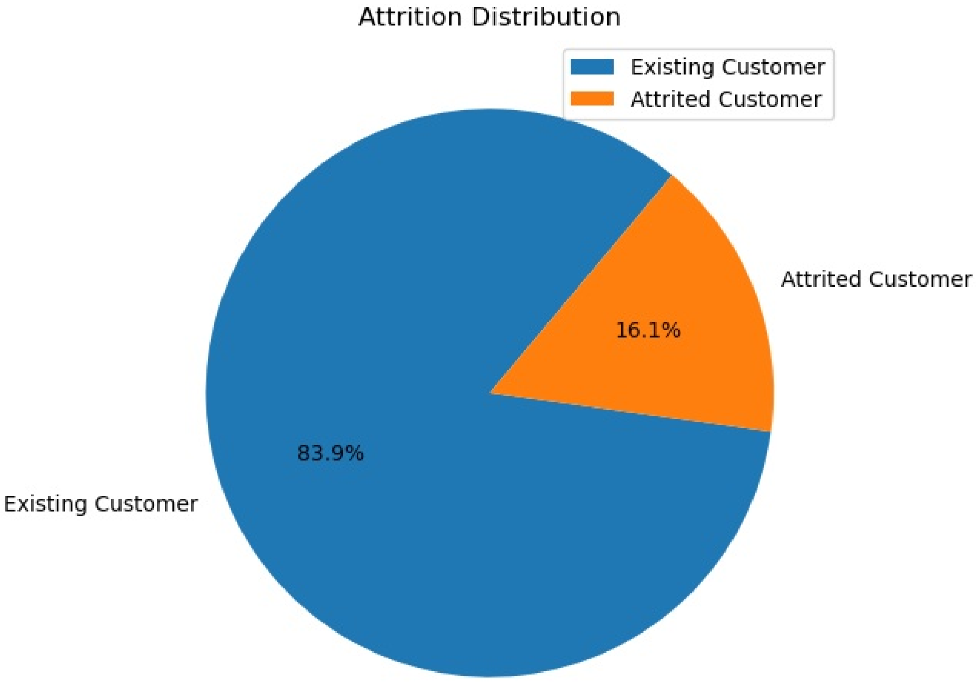
DATASET VISUALIZATION

Before encoding those 5 columns, we will visualize the data to understand more about data.

In [33]:
```
# Display the distribution of Attrition_Flag column
attrition_counts = bank['Attrition_Flag'].value_counts()
print("Attrition Distribution:")
print(attrition_counts)

# Visualize the distribution using a pie chart
plt.figure(figsize=(8, 6))
attrition_counts.plot.pie(autopct='%1.1f%%', startangle=50, legend=True)
plt.title("Attrition Distribution")
plt.ylabel('')
plt.show()
```

```
Attrition Distribution:
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
```



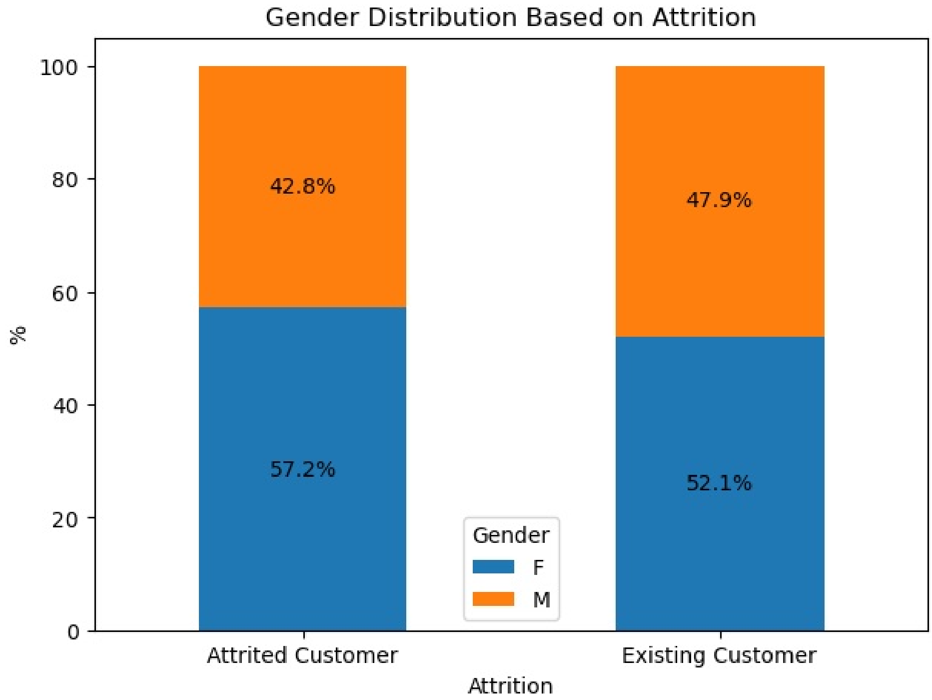
In [34]:
```
# Display gender distribution based on attrition
```

```
gender_counts = bank.groupby(['Attrition_Flag', 'Gender']).size().unstack()
gender_percentages = gender_counts.div(gender_counts.sum(axis=1), axis=0) * 100

# Visualize the distribution using a stacked bar chart
ax = gender_percentages.plot(kind='bar', stacked=True, figsize=(7, 5))
plt.title("Gender Distribution Based on Attrition")
plt.xlabel("Attrition")
plt.ylabel("%")
plt.legend(title="Gender")
plt.xticks(rotation=0)

# Add percentage annotations to the bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.1f}%', (x + width / 2, y + height / 2), ha='center', va='center')

plt.show()
```



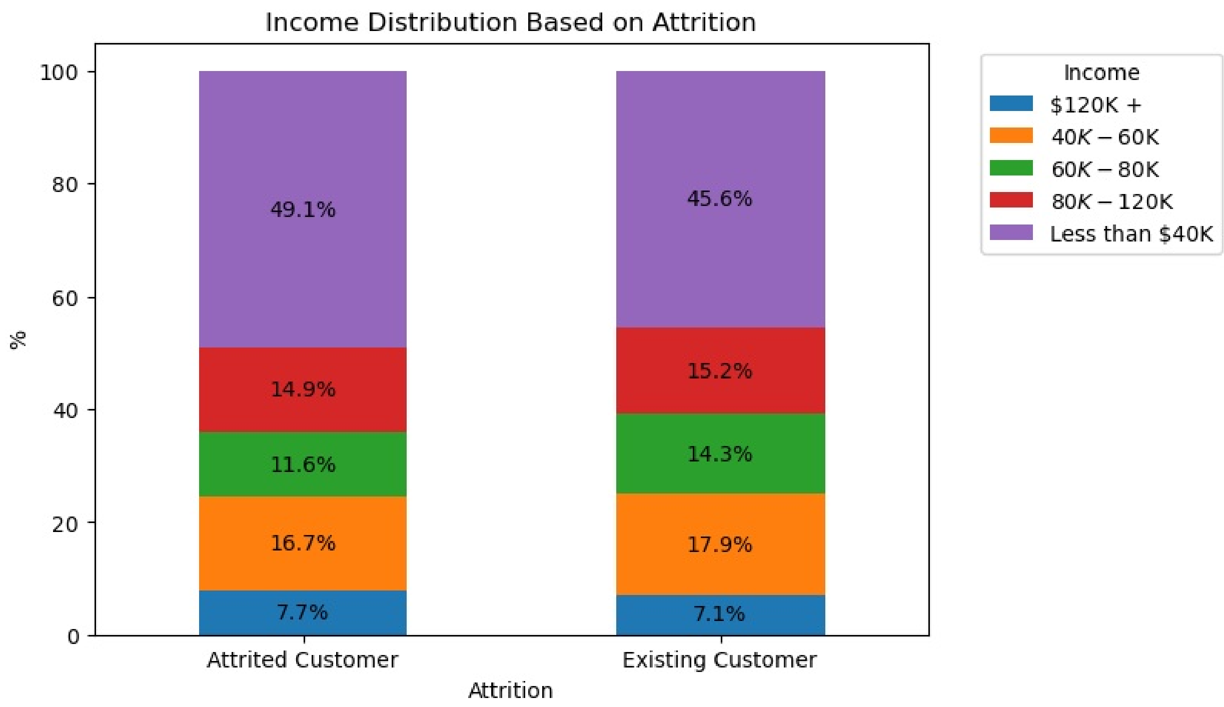
In [35]:

```
# Display gender distribution based on attrition
income_counts = bank.groupby(['Attrition_Flag', 'Income_Category']).size().unstack()
income_percentages = income_counts.div(income_counts.sum(axis=1), axis=0) * 100

# Visualize the distribution using a stacked bar chart
ax = income_percentages.plot(kind='bar', stacked=True, figsize=(7, 5))
plt.title("Income Distribution Based on Attrition")
plt.xlabel("Attrition")
plt.ylabel("%")
plt.legend(title="Income", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xticks(rotation=0)

# Add percentage annotations to the bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate(f'{height:.1f}%', (x + width / 2, y + height / 2), ha='center', va='center')

plt.show()
```



ENCODING CATEGORICAL VARIABLES

In [36]:

```
# Create a label encoder object
le = LabelEncoder()

# Encode the 'Gender' column (binary variable)
bank['Gender'] = le.fit_transform(bank['Gender'])

# Encode the 'Attrition_Flag' column (binary variable)
bank['Attrition_Flag'] = le.fit_transform(bank['Attrition_Flag'])

# List of other categorical columns to encode
categorical_columns = ['Education_Level', 'Marital_Status', 'Income_Category', 'Card_Category']

# Encode the other categorical columns using label encoding
for col in categorical_columns:
    bank[col] = le.fit_transform(bank[col])

bank.head()
```

Out[36]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category
0	768805383	1	45	1	3	3	1	2	0
1	818770008	1	49	0	5	2	2	4	0
2	713982108	1	51	1	3	2	1	3	0
3	769911858	1	40	0	4	3	1	4	0
4	709106358	1	40	1	3	5	1	2	0

5 rows × 21 columns

PREDICTION MODEL CONSTRUCTION

In this section, we will choose two models for this classifier data (KNN & Forest Decision Tree). After comparing the accuracy score, we will fit the best model to predict which customer will churn out.

In [37]:

```
# We choose Attrition_flag as Target to predict and drop it from original table
X = bank.drop('Attrition_Flag', axis=1)
Y = bank['Attrition_Flag']

# Split dataset into train and test set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Scale feature with KNN model
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Fit KNN model to esimate the accuracy and precision score
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train_scaled, Y_train)
Y_pred_knn = knn.predict(X_test_scaled)
knn_accuracy = accuracy_score(Y_test, Y_pred_knn)
```

```
knn_precision = precision_score(Y_test, Y_pred_knn)

# Fit forest model to esimate the accuracy and precision score
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, Y_train)
Y_pred_rf = rf.predict(X_test)
rf_accuracy = accuracy_score(Y_test, Y_pred_rf)
rf_precision = precision_score(Y_test, Y_pred_rf)
```

```
C:\Users\hoang\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike othe
r reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the
`axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set
`keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [38]:

```
# Print out the table with model name and accuracy
model_accuracies = pd.DataFrame({"Model": ["KNN", "Random Forest"],
                                 "Accuracy": [knn_accuracy, rf_accuracy],
                                "Precision": [knn_precision, rf_precision]})
print(model_accuracies)
```

```
           Model  Accuracy  Precision
0            KNN  0.903258   0.908646
1  Random Forest  0.956565   0.962665
```

Comment: According to the table, we can see the accuracy of Random Forest model (95.7%) is higher than that of KNN, which means Random Forest model can predict more correctly than KNN does. Moreover, the Precision score of Random Forest is higher too, so ratio of giving false positive outcomes is lower. Therefore, among 2 models, Random Forest will be a better choice.

In [39]:

```
optimal_model = rf
X_scaled = X
Y_pred_bank = optimal_model.predict(X_scaled)

# Predicted Table for Entire Dataset
client_nums = bank['CLIENTNUM']
df = pd.DataFrame({"CLIENTNUM": client_nums,"Actual": Y, "Predicted": Y_pred_bank})
print(df.head(10))

# Export to csv file
df.to_csv("predictions.csv", index=False)

end_time = time.time()
execution_time = end_time - start_time
print("Execution time: {:.6f} seconds".format(execution_time))
```

```
   CLIENTNUM  Actual  Predicted
0  768805383       1          1
1  818770008       1          1
2  713982108       1          1
3  769911858       1          1
4  709106358       1          1
5  713061558       1          1
6  810347208       1          1
7  818906208       1          1
8  710930508       1          1
9  719661558       1          1
Execution time: 5.722677 seconds
```

Processing math: 100%