Final Project Report

Group members

Jefford Secondes, C0865112

Maricris Resma, C0872252

Jovi Fez Bartolata, C0869701

Luz Zapanta, C0879190

Keyvan Amini, C0866360

Lambton College

2023F-T3 BDM 3603 – Big Data Framework

Prof. Manasa Ram

December 12, 2023

Table of Contents **Pages**

1. Introduction

A.) Problem statement

In the fast-changing world of financial services, credit card companies need to understand and manage their relationships with customers well to be successful. Credit card companies must figure out how to find and serve different groups of customers with different products and services. This is one of their biggest problems. The goal of this project is to use predictive modeling to sort and divide credit card customers into groups based on their demographics, transaction history, and behavior.

B.) Need of the study/project

- To understand the factors that are responsible for customer churn.
- To propose commercial actions aimed at maintaining clients that are showing signs of churn and offer them customized offers.
- To develop a churn predictions model for this company.
- Provide business recommendations on the campaign.

C.) Understanding Business

The average number of customers who leave is 16%. Through this chance, we can find the leaks and learn more about why customers leave, which will cut the costs of losing customers by a large amount. The data shows that keeping current customers is just as important as getting new ones, if not more so.

D.) Project description

This project aims to develop a model that effectively predicts customer attrition for a credit card company. Moreover, this activity aims to identify key factors that contribute to attrition and deliver actionable insights to minimize client loss. The scope of work will include data collection, cleaning, analysis, feature selection, and model creation utilizing machine learning methods.

2. Understanding the dataset

A.) Dataset

This dataset has a lot of information about customers that was gathered from a portfolio of consumer credit cards so that analysts can figure out which customers are likely to leave. It has a lot of information about the person, like their age, gender, marital status, and level of income. It also has information about their relationship with the credit card company, like the type of card they have, how many months they've had it open, and when they haven't used it. It also has important information about how customers spend their money that helps you figure out if they are going to leave, like the total revolving balance, credit limit, and average open to buy rate. You can also look at metrics like the total amount of change from quarter 4 to quarter 1, the average utilization ratio, and the Naive Bayes classifier attrition flag (card category is combined with contacts count in 12 months, dependent count, education level, and months inactive). With this set of useful predicted data points across multiple variables, we can get up-to-date

information that can tell us if an account will stay stable over the long term or if a customer is about to leave. This gives us the tools we need to manage a portfolio or serve individual customers.

Dataset: https://www.kaggle.com/datasets/thedevastator/predicting-credit-card-customer-attrition-with-m/data

B.) Why we choose this dataset?

For predicting credit card customer segmentation, picking a dataset that is relevant to the real world and useful is a good idea. Credit card customer segmentation divides customers into groups based on things like how much they spend and how they use their credit. This lets businesses target them more effectively and learn more about their risk. This dataset usually has a lot of different information in it, like transaction history, demographics, and credit usage, which makes it good for looking into complicated relationships. Using this dataset for predictive modeling can help financial institutions make better decisions, learn more about how customers behave, and improve the customer experience. This shows how useful the dataset is for solving real business problems in the financial sector.

3. Data Exploration | EDA

A.) Data Handling

Importing libraries

SparkSession: It sets up and initializes the entry point to Apache Spark from **pyspark.sql**, which lets you make distributed data processing apps.

The **BinaryClassificationEvaluator** and **MulticlassClassificationEvaluator** classes, which are both **from pyspark.ml.evaluation**, are used to rate how well binary and multiclass classification models work. They figure out things like accuracy and the area under **the ROC curve**.

These are feature engineering tools from **pyspark.ml.feature**. They are called **VectorAssembler** and **StandardScaler**. **VectorAssembler** is used to put together feature vectors, and **StandardScaler** is used to scale and normalize numerical features.

DecisionTreeClassifier: This classification algorithm, which can be found in **pyspark.ml.classification**, uses decision trees, which are common for tasks that need to sort things into two or more groups.

StringIndexer is another part of **pyspark.ml.feature** that turns categorical labels into numbers, which is a step that many machine learning algorithms need to take.

Pipeline: This feature from pyspark.ml makes it easier to create and run machine learning workflows by letting you combine multiple stages in a certain order.

If you import functions and **integerType** from **pyspark.sql**, you can use them to do different things with data in PySpark **DataFrames**, like specifying data types.

Outside Libraries:

This **is LogisticRegression** from scikit-learn: From **sklearn.linear_model**, this is a well-known binary classification algorithm. It's used with PySpark classifiers to do comparative analysis or when **PySpark's logistic regression implementation doesn't work**.

You can use **seaborn (sns), pandas (pd),** and **matplotlib.Use pyplot as plt**: A lot of people use these outside libraries to see and analyze data**. Seaborn** is a high-level tool for making visually appealing statistical graphics. It is based on **Matplotlib. Pandas** is a flexible library for working with data, and **Matplotlib** is a complete Python library for making 2D plots. They are often used together to make visualizations, especially when looking at data or showing results in places other than PySpark.

DATA HANDLING

```
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.classification import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
from pyspark.sql.functions import *
from pyspark.sql.types import IntegerType
import seaborn as sns
from pyspark.ml.evaluation import *
import pandas as pd
import matplotlib.pyplot as plt
```

Creating a Spark session

```
# Create a Spark session
spark = SparkSession.builder.appName("CreditCardAttrition").getOrCreate()
```

✓ 4.2s

Reading the data file

```
df = spark.read.csv("creditcardattritiondata.csv", header=True, inferSchema=True)
```

✓ 5.7s

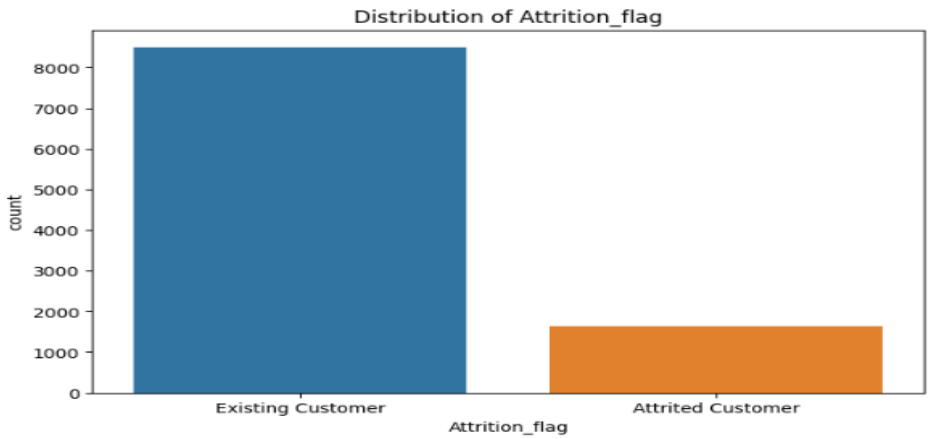
This is how a PySpark DataFrame is made from a CSV file called "creditcardattritiondata.csv." Spark's read.csv method reads the contents of the CSV file and loads them into a DataFrame called df. If you set header=True, the header will be in the first row of the CSV file. The header is what names the columns in the DataFrame. This helps make things clearer and easier to find again. When inferSchema=True, PySpark is told to automatically figure out the column data types from the CSV file's content. This makes sure that the DataFrame has the right schema. Overall, this line of code is very important for creating a PySpark DataFrame from outside data so that it can be used for data exploration, analysis, and machine learning tasks in the Spark environment.

Plot the data using MATPLOTLIB before preprocessing step

```
import matplotlib.pyplot as plt

# Assuming your target variable is called 'Attrition' (change it accordingly)
target_variable = 'Attrition_flag'

# Check the distribution of the target variable
df.groupBy(target_variable).count().show()

# Convert the Spark DataFrame to a Pandas DataFrame for easier plotting
pd_df = df.select(target_variable).toPandas()

# Plot the distribution of the target variable using matplotlib
plt.figure(figsize=(8, 5))
sns.countplot(x=target_variable, data=pd_df)
plt.title('Distribution of {}'.format(target_variable))
plt.show()
```

1.1s

```
+------------------+-----+
|   Attrition_flag|count|
+------------------+-----+
|Existing Customer| 8500|
|Attrited Customer| 1627|
+------------------+-----+
```



The bar chart showing the distribution of Attrition_flag to Existing Customers and Attrited Customers

Visual inspection of data

Define the schema printSchema()

printSchema() method in PySpark is used to display the schema of a DataFrame. The schema describes the structure of the DataFrame, including the names of the columns and their corresponding data types.
When you run printSchema(), the output will include details about each column in the DataFrame. It shows the column name, its data type, and whether it allows null values.

Schema indicates that the DataFrame has number of columns and if they are String, integer and null values

```
1  print("DataFrame Schema:")
2  df.printSchema()
```

✓ 0.0s Python

```
DataFrame Schema:
root
 |-- CLIENTNUM: integer (nullable = true)
 |-- Attrition_Flag: string (nullable = true)
 |-- Customer_Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Dependent_count: integer (nullable = true)
 |-- Education_Level: string (nullable = true)
 |-- Marital_Status: string (nullable = true)
 |-- Income_Category: string (nullable = true)
 |-- Card_Category: string (nullable = true)
 |-- Months_on_book: integer (nullable = true)
 |-- Total_Relationship_Count: integer (nullable = true)
 |-- Months_Inactive_12_mon: integer (nullable = true)
 |-- Contacts_Count_12_mon: integer (nullable = true)
 |-- Credit_Limit: double (nullable = true)
 |-- Total_Revolving_Bal: integer (nullable = true)
 |-- Avg_Open_To_Buy: double (nullable = true)
 |-- Total_Amt_Chng_Q4_Q1: double (nullable = true)
 |-- Total_Trans_Amt: integer (nullable = true)
 |-- Total_Trans_Ct: integer (nullable = true)
 |-- Total_Ct_Chng_Q4_Q1: double (nullable = true)
 |-- Avg_Utilization_Ratio: double (nullable = true)
 |-- Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1: double (nullable = true)
 |-- Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2: double (nullable = true)
```

Top 5 rows

```
+---------+-----------------+------------+------+---------------+---------------+--------------+---------------+-------------+--------------+------------------------+
|CLIENTNUM|   Attrition_Flag|Customer_Age|Gender|Dependent_count|Education_Level|Marital_Status|Income_Category|Card_Category|Months_on_book|Total_Relationship_Count|
+---------+-----------------+------------+------+---------------+---------------+--------------+---------------+-------------+--------------+------------------------+
|789124683|Existing Customer|          54|     M|              2|        Unknown|       Married|   $80K - $120K|         Blue|            42|                       4|
|717296808|Existing Customer|          67|     F|              1|       Graduate|       Married| Less than $40K|         Blue|            56|                       4|
|711551958|Attrited Customer|          59|     M|              2|  Post-Graduate|       Married|    $60K - $80K|         Blue|            46|                       3|
|713441958|Existing Customer|          46|     M|              2|    High School|       Married|        $120K +|         Blue|            36|                       5|
|789513858|Attrited Customer|          43|     M|              3|        Unknown|       Married|    $40K - $60K|         Blue|            35|                       2|
+---------+-----------------+------------+------+---------------+---------------+--------------+---------------+-------------+--------------+------------------------+
only showing top 5 rows
```

Displaying the last 5 rows

```
last_5_rows = df.toPandas().tail(5)

# Display the last 5 rows
print(last_5_rows)
```

✓ 1.3s

```
       CLIENTNUM     Attrition_Flag  Customer_Age Gender  Dependent_count  \
10122  711339258  Existing Customer            49      F                3
10123  711465333  Existing Customer            35      M                4
10124  712438308  Existing Customer            27      F                1
10125  717774333  Attrited Customer            48      M                3
10126  778745058  Existing Customer            50      M                3

      Education_Level Marital_Status Income_Category Card_Category  \
10122        Graduate         Single     $40K - $60K        Silver
10123     High School         Single    $80K - $120K          Blue
10124     High School        Married         Unknown          Blue
10125     High School        Married     $60K - $80K          Blue
10126     High School         Single    $80K - $120K        Silver

       Months_on_book  ...  Credit_Limit  Total_Revolving_Bal  \
10122              36  ...       20176.0                 2219
10123              26  ...       24250.0                    0
10124              15  ...        7469.0                    0
10125              36  ...        8431.0                    0
10126              41  ...       34516.0                    0

       Avg_Open_To_Buy  Total_Amt_Chng_Q4_Q1  Total_Trans_Amt  Total_Trans_Ct  \
10122          17957.0                 0.693             8255              86
10123          24250.0                 0.753            12593              99
10124           7469.0                 0.688            16730             122
...
10125                                          0.001751
10126                                          0.999940
```

Variable Overview

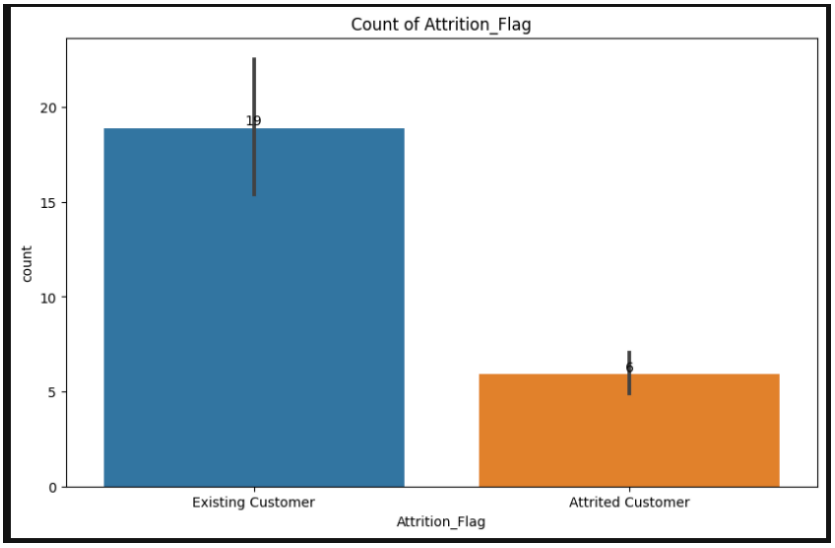
This dataset provides customer information to predict customer attrition for a consumer credit card portfolio.

This dataset provides customer information to predict customer attrition for a consumer credit card portfolio.

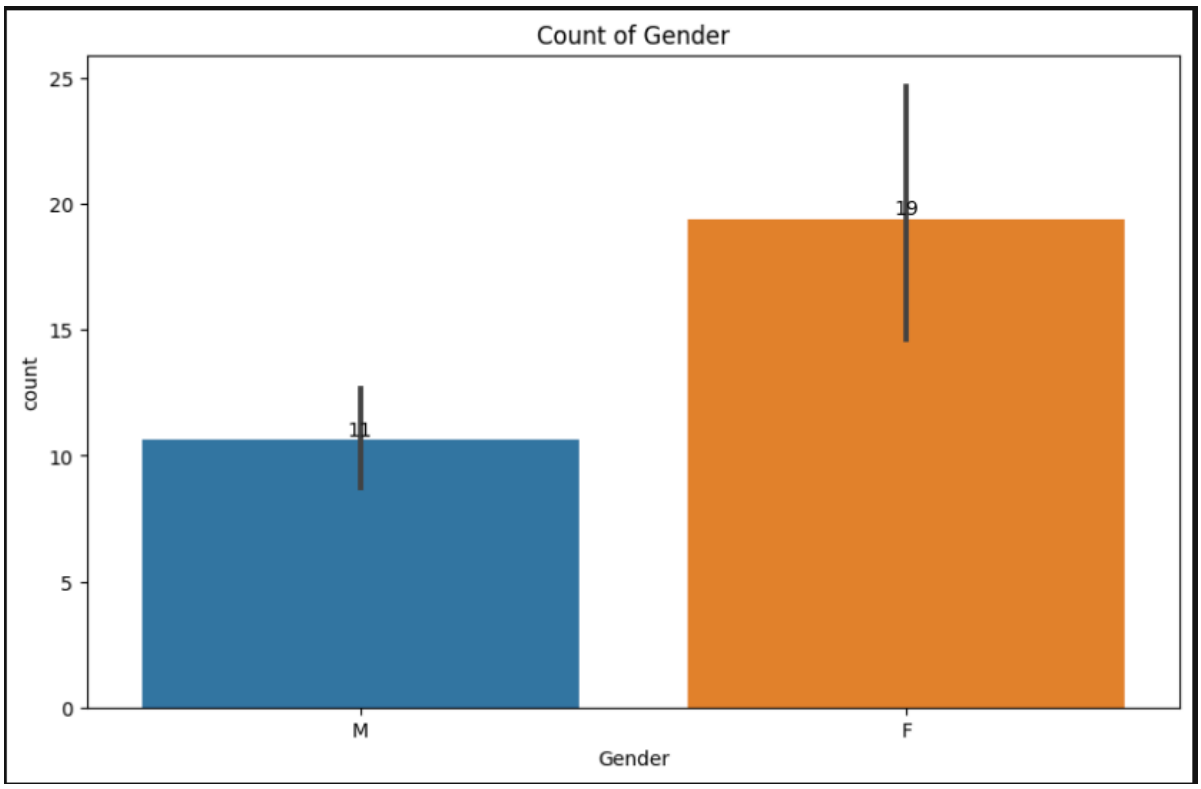
- Attrition_Flag: Flag indicating whether or not the customer has churned out. (Boolean)
- Age: Age of customer. (Integer)
- Gender: Gender of customer. (String)
- Dependents: Number of dependents that customer has. (Integer)
- Education: Education level of customer. (String)
- Marital_Status: Marital status of customer. (String)
- Income_Category: Income category of customer. (String)
- Card_Category: Type of card held by customer. (String)
- Months: How long customer has been on the books. (Integer)
- Relations: Total number of relationships customer has with the credit card provider. (Integer)
- Inactive_Months: Number of months customer has been inactive in the last twelve months. (Integer)
- Contacts_Count: Number of contacts customer has had in the last twelve months. (Integer)
- Credit_Limit: Credit limit of customer. (Integer)
- Revolving_balance: Total revolving balance of customer. (Integer)
- Open_Buy_Ratio:Average open to buy ratio of customer. (Integer)
- Q4_Q1_Amt_Change: Total amount changed from quarter 4 to quarter 1. (Integer)
- Trans_Amount: Total transaction amount. (Integer)
- Trans_Count: Total transaction count. (Integer)
- Q4_Q1_Ct_Change: Total count changed from quarter 4 to quarter 1. (Integer)
- Uti_Ratio: Average utilization ratio of customer. (Integer)
- Churn_Prob: Naive Bayes classifier for predicting whether or not someone will churn based on characteristics such (Integer)

Univariate & Multivariate Analysis

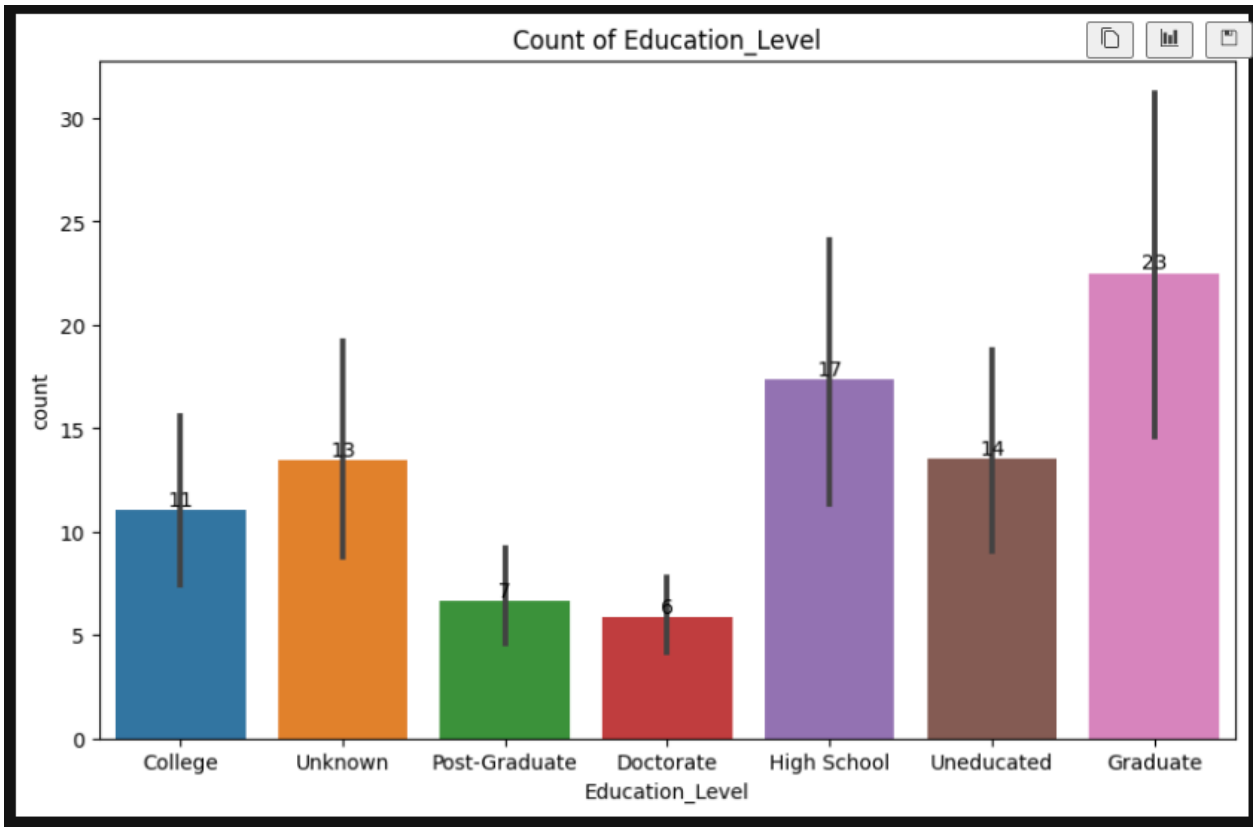
Count plot – Attrition_flag



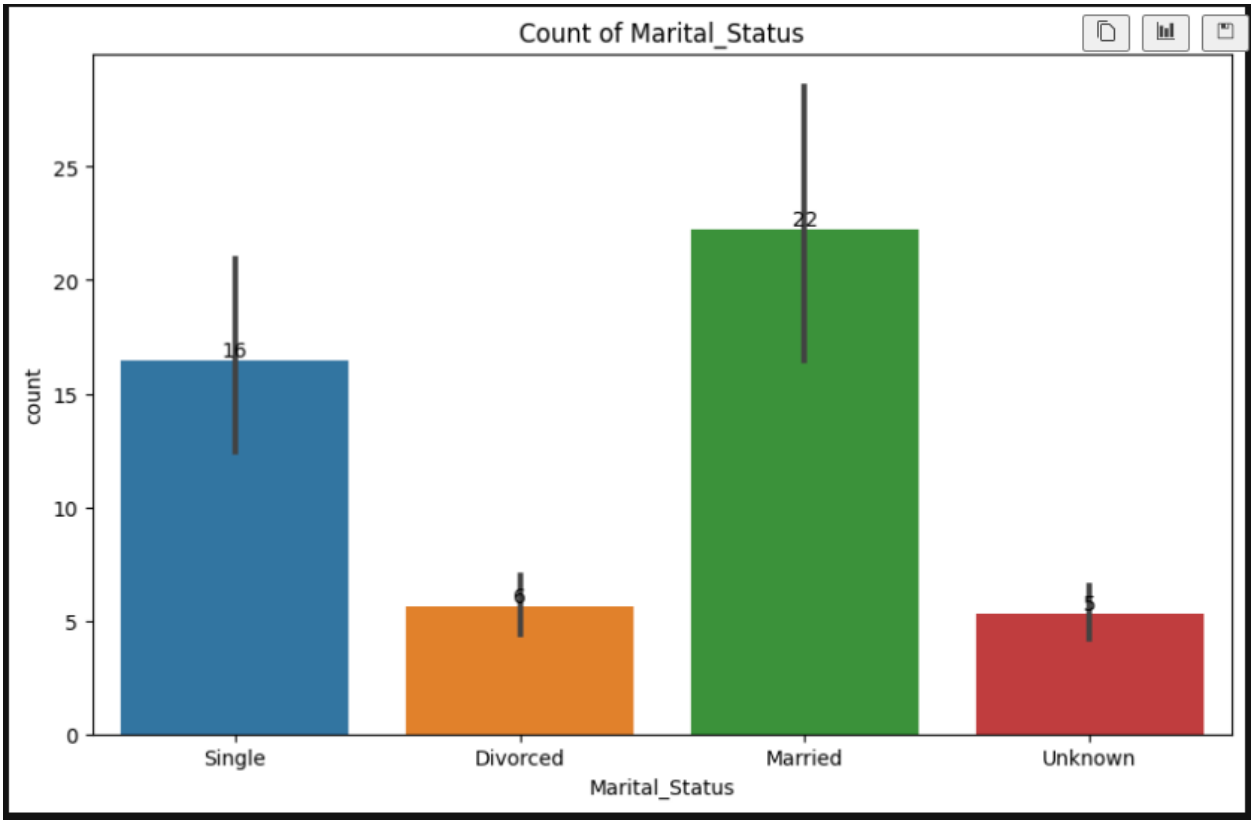
Count of Gender



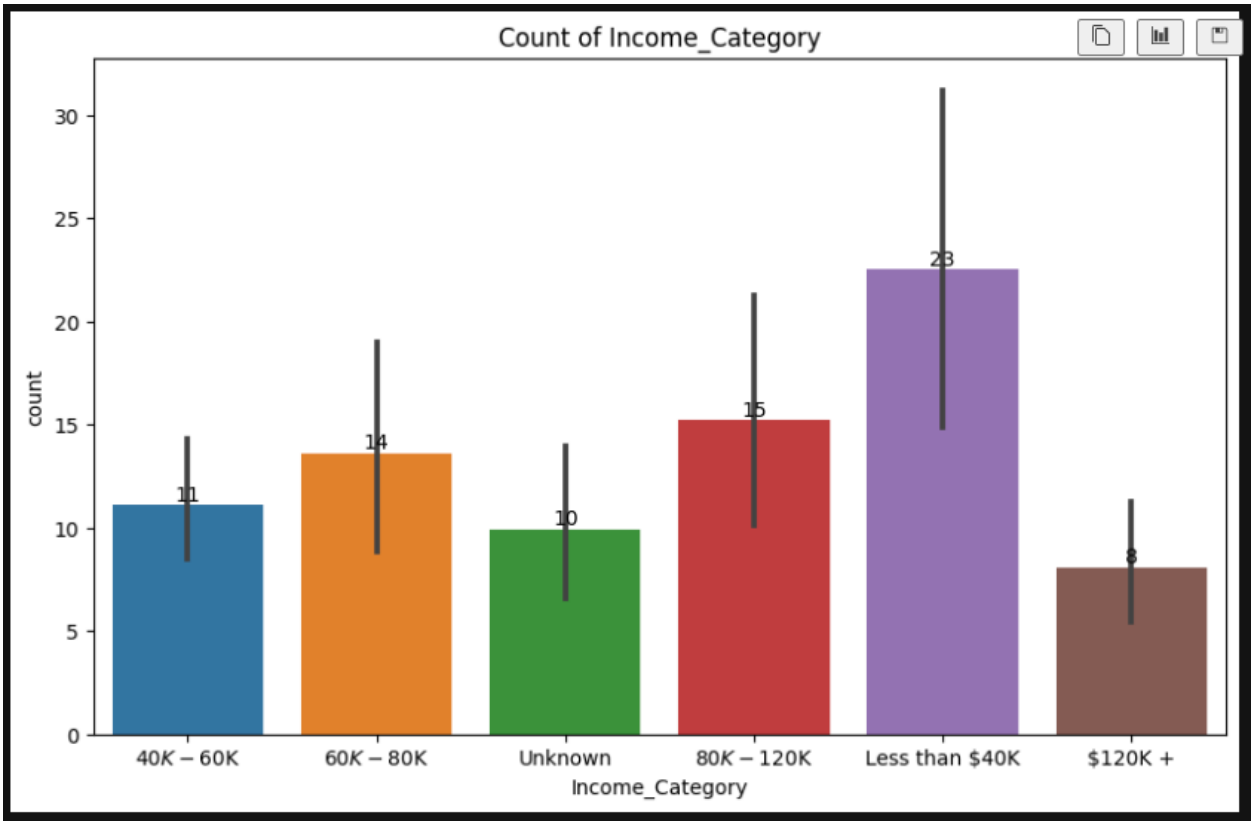
Count of Education_Level



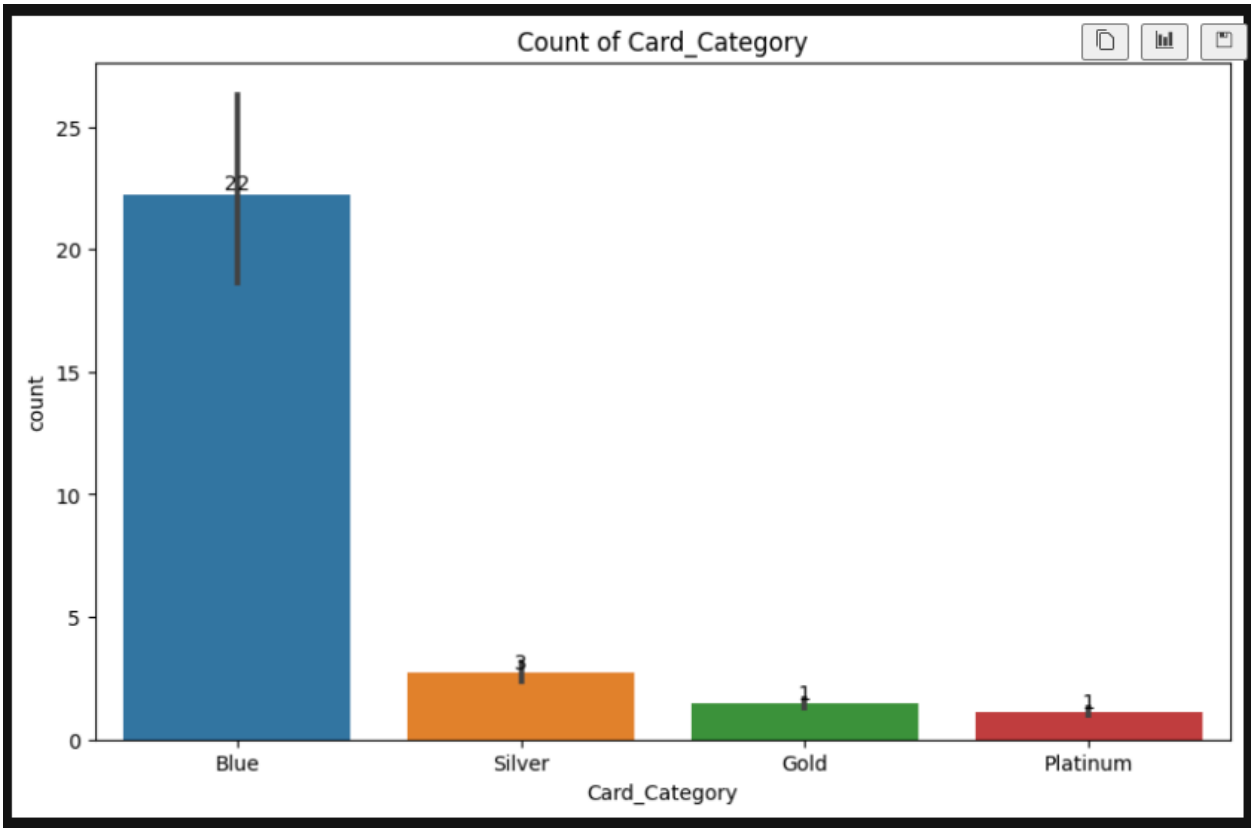
Count plot Marital_status



Count plot Income_Category



Count of plot - Card_Category



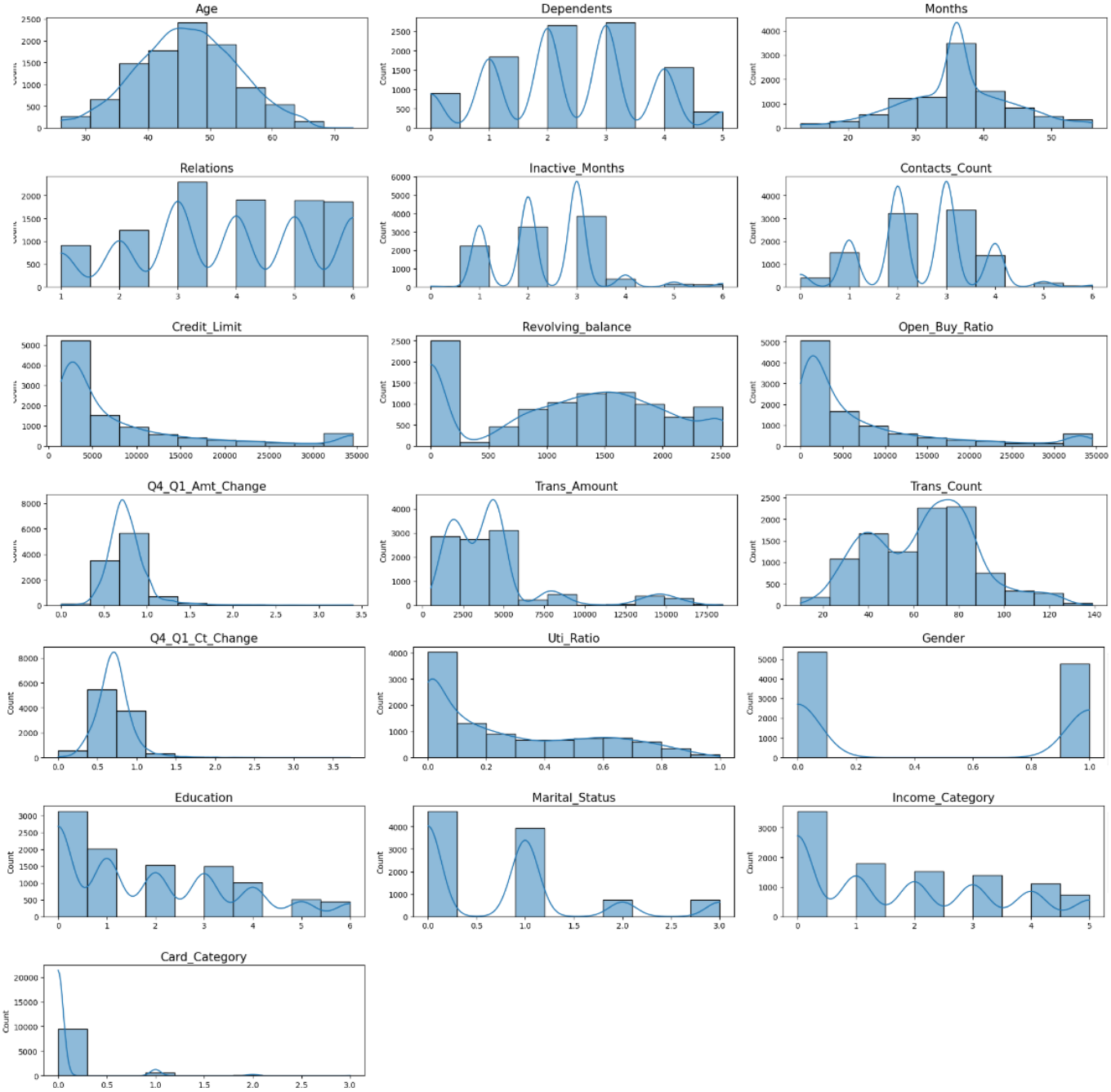
All the above shown charts are the count plots of the Categorical variables of the data set. The insights are as follows:

- According to the count plots, highest count is for the following cases.
  - Attrition_flag – Existing Customer
  - Gender – Female
  - Education_level – Graduate
  - Marital Status – Married
  - Income_Category – Less than $40K
  - Card_Category – Blue

- Categorical Features : ['Gender', 'Education_Level', 'Marital_Status', 'Income_Category', 'Card_Category']
- Numerical Features: ['Attrition_Flag', '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']

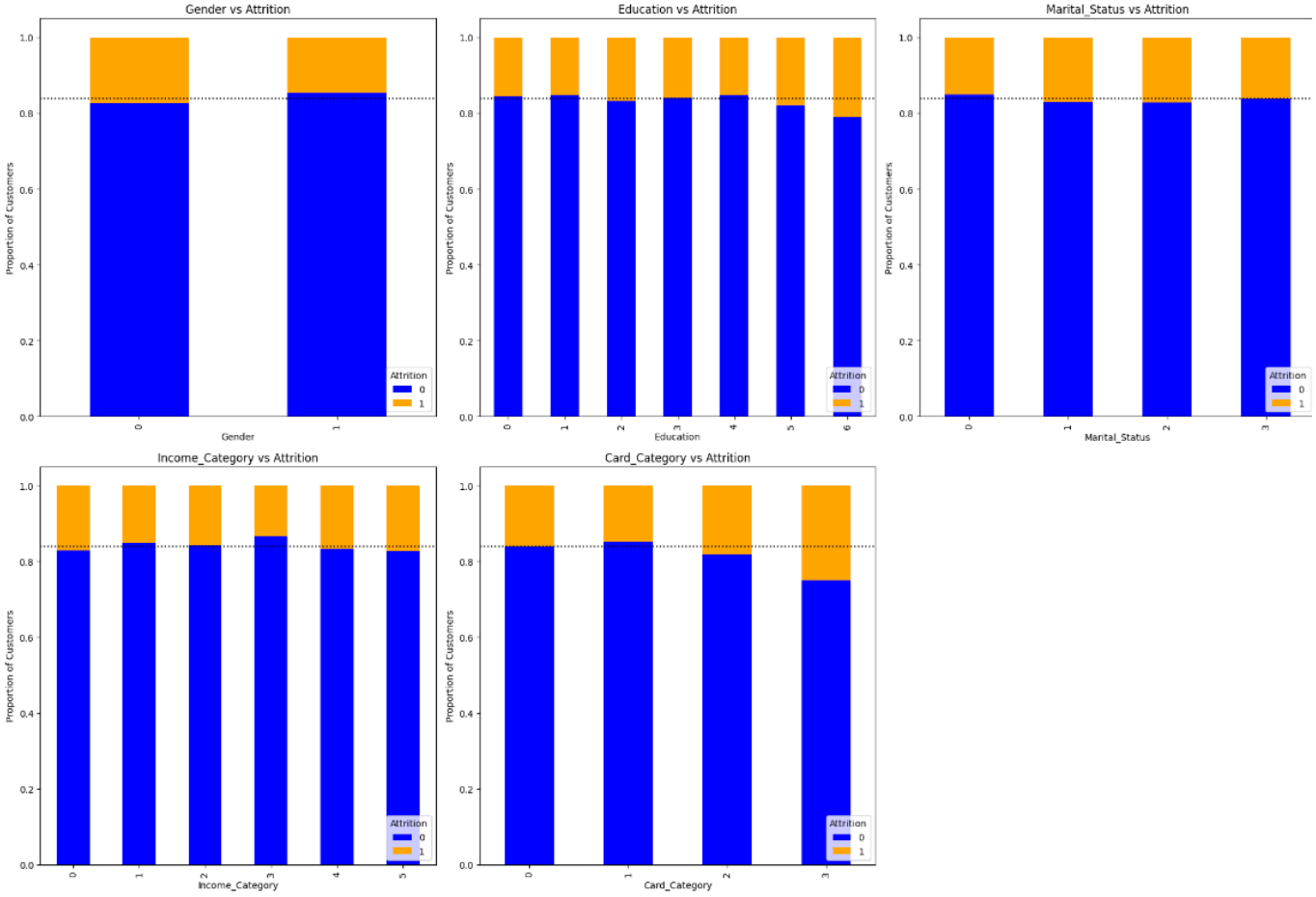
Extract numerical features

This valuable insights into the characteristics of the data, inform decisions regarding data preprocessing, and contribute to the overall understanding of the dataset.



Extract categorical features

This provides a clear visualization of the frequency distribution of categories within each categorical feature. Understanding the distribution helps to identify the prevalence of different categories and their relative frequencies in the dataset.



4.) Data Preprocessing and Data Cleaning

Dimensions of data

- The data is in a shape of 10127 rows and 23 columns
- There are 3 data types in the data set – dtypes: float64(7), int64(10), object(6)
- No feature has null or nan values

Describe the data

You can use the describe() function on a DataFrame (df) to get basic statistics for the numerical columns. This function gives you an overview of the central tendency, dispersion, and shape of the distribution of the numerical columns in a DataFrame.

df.describe().show()

DATA PREPROCESSING

Describe the data

```
#You can use the describe() function on a DataFrame (df) to get basic statistics for the numerical columns. This function gives you an overview of the cent
df.describe().show()
```

✓ 1.5s Python

```
+-------+------------------+--------------------+------------------+----------+------------------+---------------+--------------+--------------+-----------------+------------------+
|summary|         CLIENTNUM|      Attrition_Flag|      Customer_Age|    Gender|   Dependent_count|Education_Level|Marital_Status|Income_Category|    Card_Category|         Months_
+-------+------------------+--------------------+------------------+----------+------------------+---------------+--------------+--------------+-----------------+------------------+
|  count|             10127|               10127|             10127|     10127|             10127|          10127|         10127|         10127|            10127|
|   mean|7.391776063336625E8|                null|46.32596030413745|      null|2.3462032191172115|           null|          null|          null|             null|35.928409
| stddev|3.690378345023116E7|                null|8.016814032549046|      null| 1.29890834890379|           null|          null|          null|             null| 7.986416
|    min|         708082083|Attrited Customer|                26|         F|                 0|        College|      Divorced|      $120K +|             Blue|
|    max|         828343083|Existing Customer|                73|         M|                 5|        Unknown|       Unknown|       Unknown|           Silver|
+-------+------------------+--------------------+------------------+----------+------------------+---------------+--------------+--------------+-----------------+------------------+
```

Observations:

- **The raw data has 10,127 observations/rows and 23 variables/columns.**
- **Y Variable / Dependent Variable is Attrition_Flag**
- **X Variable / Independent Variables will be taken from the other 22 variables**

Number of columns and rows
Dimensions of data
Handle any missing or duplicate data

Observations:

- No feature has null or nan values
- No duplicate records
- Education_Level, Marital_Status and Income_Category have 'Unknown' data as value

```
# Handling missing data: Drop rows with any missing values
df = df.dropna()

# Handling duplicate data: Drop duplicate rows
df = df.dropDuplicates()

# Show the cleaned DataFrame
df.show(5)

column_count = len(df.columns) ## count the number of columns
row_count = df.count()  ## count the number of rows
print (f"Number of columns: {column_count}")
print(f"Number of rows: {row_count}")


```

✓ 2.7s Python

```
+---------+-----------------+------------+------+---------------+---------------+--------------+---------------+-------------+--------------+--------------------+
|CLIENTNUM|   Attrition_Flag|Customer_Age|Gender|Dependent_count|Education_Level|Marital_Status|Income_Category|Card_Category|Months_on_book|Total_Relationship_C
+---------+-----------------+------------+------+---------------+---------------+--------------+---------------+-------------+--------------+--------------------+
|789124683|Existing Customer|          54|     M|              2|        Unknown|       Married|    $80K - $120K|         Blue|            42|
|717296808|Existing Customer|          67|     F|              1|       Graduate|       Married| Less than $40K|         Blue|            56|
|711551958|Attrited Customer|          59|     M|              2|  Post-Graduate|       Married|     $60K - $80K|         Blue|            46|
|713441958|Existing Customer|          46|     M|              2|    High School|       Married|        $120K +|         Blue|            36|
|789513858|Attrited Customer|          43|     M|              3|        Unknown|       Married|    $40K - $60K|         Blue|            35|
+---------+-----------------+------------+------+---------------+---------------+--------------+---------------+-------------+--------------+--------------------+
only showing top 5 rows

Number of columns: 23
Number of rows: 10127
```

Count of Unique Values

Unique values

select('Attrition_Flag') method is used to extract only the 'Attrition_Flag' column from the DataFrame. The distinct() method is then applied to retain only the unique values in that column. Finally, the collect() method is used to gather the unique values into a list of Row objects.
The code iterates through each Row in the unique_values list and prints the value of the first (and only) column in each row. In PySpark, the values in a Row object can be accessed using indexing, starting from 0. Since we are dealing with a single-column DataFrame ('Attrition_Flag'), row[0] retrieves the unique value in each row.

```python
# Get unique values in the 'Attrition_Flag' column
unique_values = df.select('Attrition_Flag').distinct().collect()

# Print the unique values
for row in unique_values:
    print(row[0])
```

0.2s Python

```
Existing Customer
Attrited Customer
```

Get count of 'Attrition_Flag' column for Existing Customer and Attrited Customer

```python
# Get of 'Attrition_Flag' row
df.groupby('Attrition_flag').count().show()
```

1.3s Python

```
+-----------------+-----+
|   Attrition_flag|count|
+-----------------+-----+
|Existing Customer| 8500|
|Attrited Customer| 1627|
+-----------------+-----+
```

Encode Categorical variables (String Indexer/label encoding/onehot encoding) Gender_Index

```python
encoder = StringIndexer(inputCol='Gender', outputCol='Gender_Index')
df = encoder.fit(df).transform(df)

# Cast 'Gender_Index' to IntegerType
df = df.withColumn('Gender_Index', df['Gender_Index'].cast(IntegerType()))

# Get unique values in the encoded 'Gender' column
unique_values = df.select('Gender_Index').distinct().collect()

# Extract and print the unique values
unique_values = [row['Gender_Index'] for row in unique_values][::-1]
print(unique_values)
```

29] ✓ 2.4s

```
[0, 1]
```

Drop unnecessary features

Features that are not relevant to the target variable or may not carry meaningful information for the prediction task. In such cases, including irrelevant features may introduce noise into the model and reduce its overall performance.

CLIENTNUM,
'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1',
'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2'

5.) Encode Categorical Variables

Feature transformer (String indexer/ label encoding /onehot encoding)

StringIndexer is a feature transformer that is used to rank categorical columns in a DataFrame that are strings. It gives each unique string value in a certain column a unique numerical index. People often call this process label encoding or index encoding. Most of the time, the indexed values are fed into machine learning algorithms, since many of *them* need numbers as input. *We convert the categorical value to numerical value.*

Encode Categorical variables (String Indexer/label encoding/onehot encoding) Gender_Index

```
1  encoder = StringIndexer(inputCol='Gender', outputCol='Gender_Index')
2  df = encoder.fit(df).transform(df)
3
4  # Cast 'Gender_Index' to IntegerType
5  df = df.withColumn('Gender_Index', df['Gender_Index'].cast(IntegerType()))
6
7  # Get unique values in the encoded 'Gender' column
8  unique_values = df.select('Gender_Index').distinct().collect()
9
10 # Extract and print the unique values
11 unique_values = [row['Gender_Index'] for row in unique_values][::-1]
12 print(unique_values)
```

29] ✓ 2.4s

```
[0, 1]
```

Attrition_Flag	Age	Gender	Dependents	Education	Marital_Status	Income_Category	Card_Category	Months	Relations	Inactive_Months	Contacts_Count	Credit_Limit
Existing Customer	54	M	2	Unknown	Married	$80K - $120K	Blue	42	4	2	3	12217.0
Existing Customer	67	F	1	Graduate	Married	Less than $40K	Blue	56	4	3	2	3006.0
Attrited Customer	59	M	2	Post-Graduate	Married	$60K - $80K	Blue	46	3	3	3	1438.3
Existing Customer	46	M	2	High School	Married	$120K +	Blue	36	5	3	2	19727.0
Attrited Customer	43	M	3	Unknown	Married	$40K - $60K	Blue	35	2	2	3	1438.3

only showing top 5 rows

VectorAssembler is a feature transformer that Spark provides. VectorAssembler is a useful tool for putting together raw feature columns into a single feature vector column in Spark's machine learning library (MLlib)

```
# feature column named 'features' and a target column named 'Attrition_Flag'
feature_columns = ['Gender', 'Education', 'Marital_Status', 'Income_Category', 'Card_Category']
```

✓ 0.0s

```

# Assemble features into a vector column
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
```

✓ 0.0s

6.) ***Data Transformation***

Standard Scaler

StandardScaler

```
# StandardScaler
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withMean=True, withStd=True)
```

73] ✓ 0.0s

A method used in machine learning that makes features more consistent by changing their mean to 0 and their standard deviation to 1. It works on every feature separately, making sure that they are all about the same level so that no one feature takes over. Normalization is very important for algorithms that depend on feature size, like gradient-based optimization methods. It is common to use Standard Scaler to improve model performance, especially in algorithms like Support Vector Machines, k-means clustering, and Principal Component Analysis. Consistent feature scaling helps make training and evaluating models more accurate and efficient.

7.) Model building

Supervised learning is a type of data analysis where the desired outcome is known or labeled, like whether the customer(s) who left did not. When the goal is to put them into groups based on why each one churned, however, it becomes unsupervised. This could be done to find out how customers feel about your business and why they leave.

Classification and Regression are both types of supervised learning. However, classification is used when the outcome has a limited range of possible values, while regression is used when the outcome has an infinite range of possible values. A classification algorithm uses training data to figure out what kind of thing new observations belong to. When you give a program a dataset of observations, it learns from them and then puts new observations into several classes or groups. You can refer to classes as labels, targets, or groups.

The main goal of a classification algorithm is to figure out what kind of data it is given. These algorithms are mostly used to guess what will happen with categorical data. There are two more groups that classification algorithms can be put into: There are linear models and non-linear models.

Model choices and why?

Since, this project is focused on a Classification problem. The following models are *built with* the following model procedures to *analyze* and review the dataset and get the performance and importance of the features available on the dataset which can *gather* more information about the subjects. Selecting Logistic Regression and Decision Trees for Predicting Credit Card Customer Segmentation offers a balanced approach. Logistic Regression suits this task due to its suitability for binary classification, providing interpretable probabilities for customer attrition. It efficiently handles linear relationships in demographic and transactional data. Meanwhile, Decision Trees excel in capturing non-linear patterns, useful for uncovering complex interactions among diverse features in credit behavior. Their interpretability aids in identifying influential factors impacting segmentation. The combined use of both models leverages Logistic Regression's simplicity for straightforward insights and Decision Trees' capacity to reveal intricate relationships, enhancing the overall predictive capability in a nuanced domain like credit card customer segmentation.

Logistic Regression

One of the "white box" algorithms that helps us figure out the probability values and the cut-offs that go with them is logistic regression. Logistic regression is used to solve this kind of problem because it gives us the probability outputs we need. From there, we can choose the right cut-off points to get the target class outputs. With this model, no assumptions need to be made, and it's faster to put things into groups. This is a practical way to figure out how to divide credit card customers into groups. It is very good at binary classification tasks, which makes it perfect for figuring out how customers will act in situations like customer loss or segmentation. Its simplicity makes it easy to understand the coefficients, which gives us information about the things that affect how credit card customers behave. Logistic Regression is a model that lets you easily see how linear relationships work between things like transaction history and demographics. It is also very fast to

use. In credit card segmentation, it's important to tell the difference between groups of customers. Logistic Regression gives a clear and understandable framework for predictive modeling, which helps financial institutions make strategic decisions.

Logistic Regression

```
# Logistic Regression classifier
lr_classifier = LogisticRegression(featuresCol='scaled_features', labelCol='Attrition_Flag', predictionCol='prediction')
```
✓ 0.0s

```
# Create a pipeline with assembler, scaler, and classifier stages
pipeline = Pipeline(stages=[assembler, scaler, lr_classifier])
```
✓ 0.0s

```
# Train/test split
train_data, test_data = df.randomSplit([0.8, 0.2], seed=42)
```
✓ 0.0s

```
# Fit the pipeline model on the training data
model = pipeline.fit(train_data)
```
✓ 8.0s

```
# Make predictions on the test data
predictions = model.transform(test_data)
```
✓ 0.0s

b.) Decision Tree

Decision trees can be used to predict credit card customer segmentation because they are easy to understand, can capture non-linear relationships in customer behavior, and can work with a variety of data types. They give you information about how important features are, which is necessary to understand the factors that affect segmentation. Decision trees use little computer power, can be expanded, and allow for quick prototyping. Decision trees can be changed to work with datasets that aren't balanced, which can happen in credit card analysis. They can also be used as parts of ensemble methods to improve the accuracy of predictions. Overall, decision trees are a clear and useful way to understand and guess complicated patterns in how credit card customers are divided.

Decision Tree classifier

+ Code + Markdown

```
# Decision Tree classifier
dt_classifier = DecisionTreeClassifier(featuresCol='scaled_features', labelCol='Attrition_Flag', predictionCol='prediction')

# Create a pipeline with assembler, scaler, and classifier stages
pipeline = Pipeline(stages=[assembler, scaler, dt_classifier])

# Train/test split
train_data, test_data = df.randomSplit([0.8, 0.2], seed=42)

# Fit the pipeline model on the training data
model = pipeline.fit(train_data)

# Make predictions on the test data
predictions = model.transform(test_data)
```

8.) Model Evaluation | Evaluation Metrics

Logistic Regression

```
# Performance Evaluation Metrics
binary_evaluator = BinaryClassificationEvaluator(labelCol='Attrition_Flag')
multi_evaluator = MulticlassClassificationEvaluator(labelCol='Attrition_Flag', metricName='f1')
```

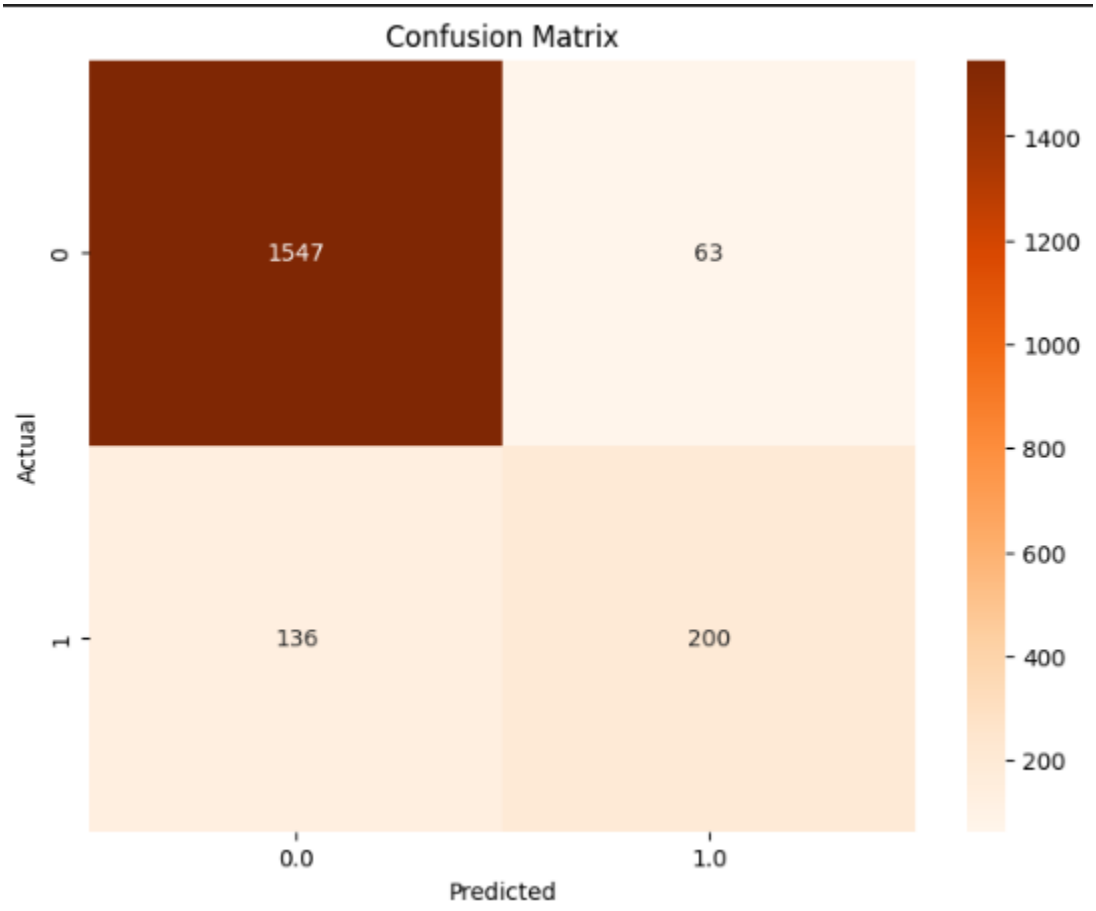
0.0s

```
# ROC AUC
roc_auc = binary_evaluator.evaluate(predictions, {binary_evaluator.metricName: 'areaUnderROC'})
print(f"ROC AUC: {roc_auc}")
# Testing Accuracy
accuracy = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: 'accuracy'})
print(f"Testing Accuracy: {accuracy}")

# Precision
precision = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: 'weightedPrecision'})
print(f"Precision: {precision}")

# Recall
recall = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: 'weightedRecall'})
print(f"Recall: {recall}")

# F1-score
f1_score = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: 'f1'})
print(f"F1-score: {f1_score}")
```

10.4s

ROC AUC: 0.9150399290150913
Testing Accuracy: 0.8977389516957862
Precision: 0.8917843257367837
Recall: 0.8977389516957863
F1-score: 0.8926411593225264

The logistic regression model does well on a number of different evaluation criteria. Receiver Operating Characteristic Area Under the Curve (ROC AUC) reaches a commendable value of 0.915, which shows that the model is very good at telling the difference between good and bad situations. The testing accuracy is 89.8%, which shows that the model is generally right at classifying observations. Precision, which shows how accurate positive predictions are, is very high at 89.2%, which means there aren't many false positives. At the same time, recall, which shows how sensitive the model is to real positive cases, is also high at 89.8%, showing that it can catch most positive cases. The F1-score, which measures how well precision and recall work together, is 0.893, which shows that the model is good at finding a good balance between these two important performance metrics. Overall, these results show that the logistic regression model is a good choice for figuring out how to divide credit card customers into groups because it has a good balance of accuracy, precision, and recall.



Finding out how well the logistic regression model does at classifying and predicting credit card customer segmentation can be done by looking at its confusion matrix. With 1,547 true positives (TP), the model was able to correctly identify customers in the given segment. Nevertheless, there were 63 false negatives (FN), which means that the model missed customers who were actually in the segment. The fact that there were 136 false positives (FP) means that the model thought some customers were in the segment when they weren't. On the bright side, 200 true negatives (TN) show that the predictions about customers who don't belong to the segment were correct. The precision, which was found to be 0.919, is the percentage of correctly predicted positive cases out of all predicted positive cases. A recall of about 0.961 means that the model can correctly identify a large number of real positive cases. These metrics show that the model is good at finding positive cases, but it also shows where it needs to improve, especially when it comes to lowering the number of false negatives and false positives so that credit card customer segmentation is more accurate.

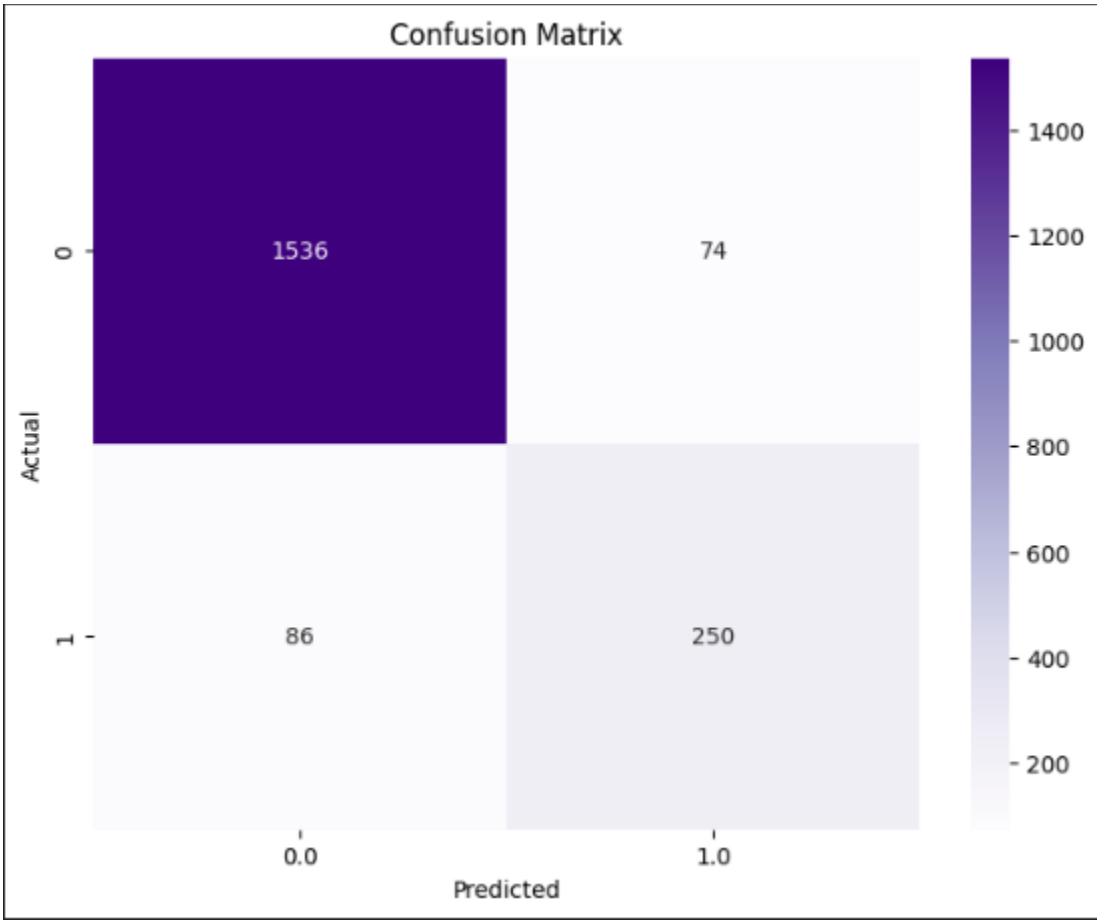
Decision Tree

ROC AUC: 0.8909087548062705

```
# Performance Evaluation Metrics
binary_evaluator = BinaryClassificationEvaluator(labelCol='Attrition_Flag')
multi_evaluator = MulticlassClassificationEvaluator(labelCol='Attrition_Flag', metricName='f1')

# ROC AUC
roc_auc = binary_evaluator.evaluate(predictions, {binary_evaluator.metricName: 'areaUnderROC'})
print(f"ROC AUC: {roc_auc}")

# Testing Accuracy
accuracy = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: 'accuracy'})
print(f"Testing Accuracy: {accuracy}")

# Precision
precision = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: 'weightedPrecision'})
print(f"Precision: {precision}")

# Recall
recall = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: 'weightedRecall'})
print(f"Recall: {recall}")

# F1-score
f1_score = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: 'f1'})
print(f"F1-score: {f1_score}")
```

Testing Accuracy: 0.9177800616649537
Precision: 0.9166986180886775
Recall: 0.9177800616649537
F1-score: 0.9171852437042265

The logistic regression model does well on a number of different evaluation criteria. With a ROC AUC score of 0.891, the model seems to be good at telling the difference between good and bad situations. With a test accuracy of 91.8%, it's clear that the observations were correctly classified overall. Precision, which measures how accurate positive predictions are, is very high at 91.7%, which means there aren't many false positives. At the same time, recall, which shows how sensitive the model is to real positive cases, is the same as accuracy at 91.8%, showing that it can catch most positive cases. The model's balanced performance is shown by its F1-score of 91.7%, which is a harmonic mean of precision and recall. All of these results show that the logistic regression model is a good fit for the job. It has a good balance between precision and recall, and it is very good at predicting the credit card customer segmentation.



The logistic regression model's confusion matrix shows how well it sorts credit card customers into groups. With 1,536 true positives (TP), the model was able to correctly identify customers in the given segment. That being said, it also showed 74 false negatives (FN), which means that it missed customers who were actually in the segment. The fact that there were 86 false positives (FP) means that the model thought some customers belonged to the segment when they actually didn't. On the plus side, 250 true negatives (TN) show correct predictions of customers who don't belong to the segment. The precision, which was found to be 0.947, is the percentage of correctly predicted positive cases out of all predicted positive cases. The model's ability to catch a lot of real positive cases is shown by the recall, which is about 0.954. Collectively, these metrics show how good the model is at finding good examples while also showing where it goes wrong and how accurate it is overall at segmenting credit card customers.

Model evaluation table

The code compares the performance of a Logistic Regression model and a Decision Tree model on a binary classification task using a dataset with various features. The code begins by preparing the data, including assembling the feature vector. It then splits the data into training and testing sets. Subsequently, two machine learning pipelines are constructed—one for Logistic Regression and another for Decision Tree each comprising a feature assembler and the respective classifier. The models are trained on the training data, and predictions are generated on the test data.

For the evaluation, the code employs various metrics, including ROC AUC, Testing Accuracy, Precision, Recall, and F1-Score. These metrics provide a comprehensive assessment of the models' performance. **BinaryClassificationEvaluato**r is used for ROC AUC, while **MulticlassClassificationEvaluator i**s employed for the remaining metrics. The evaluations are stored in a Pandas DataFrame for convenient comparison. The resulting table summarizes the performance of both models across the selected metrics, offering insights into their effectiveness in predicting the binary target variable (Attrition_Flag). The user can customize the feature columns, models, and evaluation metrics based on their specific use case and requirements.

	Model	ROC AUC	Testing Accuracy	Precision	Recall	F1-Score
0	Logistic Regression	0.915040	0.897739	0.891784	0.897739	0.892641
1	Decision Tree	0.890909	0.917780	0.916699	0.917780	0.917185

```
# Logistic Regression
lr_assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
lr_scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withMean=True, withStd=True)
lr_model = LogisticRegression(featuresCol='scaled_features', labelCol='Attrition_Flag', predictionCol='prediction')
lr_pipeline = Pipeline(stages=[lr_assembler, lr_scaler, lr_model])
lr_model = lr_pipeline.fit(train_data)
lr_predictions = lr_model.transform(test_data)

# Decision Tree
dt_assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
dt_scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withMean=True, withStd=True)
dt_model = DecisionTreeClassifier(featuresCol='scaled_features', labelCol='Attrition_Flag', predictionCol='prediction')
dt_pipeline = Pipeline(stages=[dt_assembler, dt_scaler, dt_model])
dt_model = dt_pipeline.fit(train_data)
dt_predictions = dt_model.transform(test_data)

# Performance Evaluation Metrics
binary_evaluator = BinaryClassificationEvaluator(labelCol='Attrition_Flag')
multi_evaluator = MulticlassClassificationEvaluator(labelCol='Attrition_Flag', metricName='f1')

# Create a dictionary to store the results
results = {
    'Model': ['Logistic Regression', 'Decision Tree'],
    'ROC AUC': [
        binary_evaluator.evaluate(lr_predictions, {binary_evaluator.metricName: 'areaUnderROC'}),
        binary_evaluator.evaluate(dt_predictions, {binary_evaluator.metricName: 'areaUnderROC'})
    ],
    'Testing Accuracy': [
        multi_evaluator.evaluate(lr_predictions, {multi_evaluator.metricName: 'accuracy'}),
        multi_evaluator.evaluate(dt_predictions, {multi_evaluator.metricName: 'accuracy'})
    ],
    'Precision': [
        multi_evaluator.evaluate(lr_predictions, {multi_evaluator.metricName: 'weightedPrecision'}),
        multi_evaluator.evaluate(dt_predictions, {multi_evaluator.metricName: 'weightedPrecision'})
    ],
    'Recall': [
        multi_evaluator.evaluate(lr_predictions, {multi_evaluator.metricName: 'weightedRecall'}),
        multi_evaluator.evaluate(dt_predictions, {multi_evaluator.metricName: 'weightedRecall'})
    ],
    'F1-Score': [
        multi_evaluator.evaluate(lr_predictions, {multi_evaluator.metricName: 'f1'}),
        multi_evaluator.evaluate(dt_predictions, {multi_evaluator.metricName: 'f1'})
    ]
}

# Create a Pandas DataFrame
results_df = pd.DataFrame(results)

# Display the results
print(results_df)
```

9.) Evalution metrics use

(ROC AUC, Testing Accuracy, Precision, Recall and F1-Score on Logistic regression and Decision tree)

Rules for judging how well machine learning models work are very important for figuring out how well they can make predictions. There are different metrics that are used for different things in logistic regression and decision tree models. The Receiver Operating Characteristic Area Under the Curve (ROC AUC) is useful for both models because it compares the sensitivity and specificity and gives a full picture of how well the classification worked. One common metric is testing accuracy, which shows how accurate predictions are overall but might not be enough when datasets aren't balanced.

When there are differences between classes, precision, recall, and F1-score become very important. For logistic regression, which works well for binary classification, precision measures how accurate positive predictions are, recall measures how well actual positive instances are captured, and the F1-score combines both metrics to stress the importance of finding a balance between precision and recall. In situations where the cost of false positives and false negatives is different, these metrics are very important. On the other hand, decision trees can handle multi-class classification and get better accuracy, recall, and F1-score when used with more than one class. They are models that can be understood, and knowing these metrics helps improve the structure of the tree for better prediction in certain situations. In short, the model, the type of data, and the specific goals of the machine learning task all affect the choice of evaluation metrics.

10.) Conclusion (Effectiveness of the model choose)

We choose Decision Tree model over Logistic regression.

Overall, both models do well across all metrics, but the Decision Tree model has slightly better Testing Accuracy, Precision, Recall, and F1-Score than the Logistic Regression model. The Logistic Regression model, on the other hand, has a slightly higher ROC AUC.

You should choose between the two models based on the application's goals and needs. The Decision Tree model is the best choice if we want to make predictions that are accurate overall and have a good balance between precision and recall. But if getting a large area under the ROC curve is very important, the Logistic Regression model might be better.

When choosing the best fit for the dataset, it's best to think about the application's specifics, how important false positives and false negatives are, and how easy it is to understand the models. It is also possible to do more research or experiments to improve the models and see how stable they are.

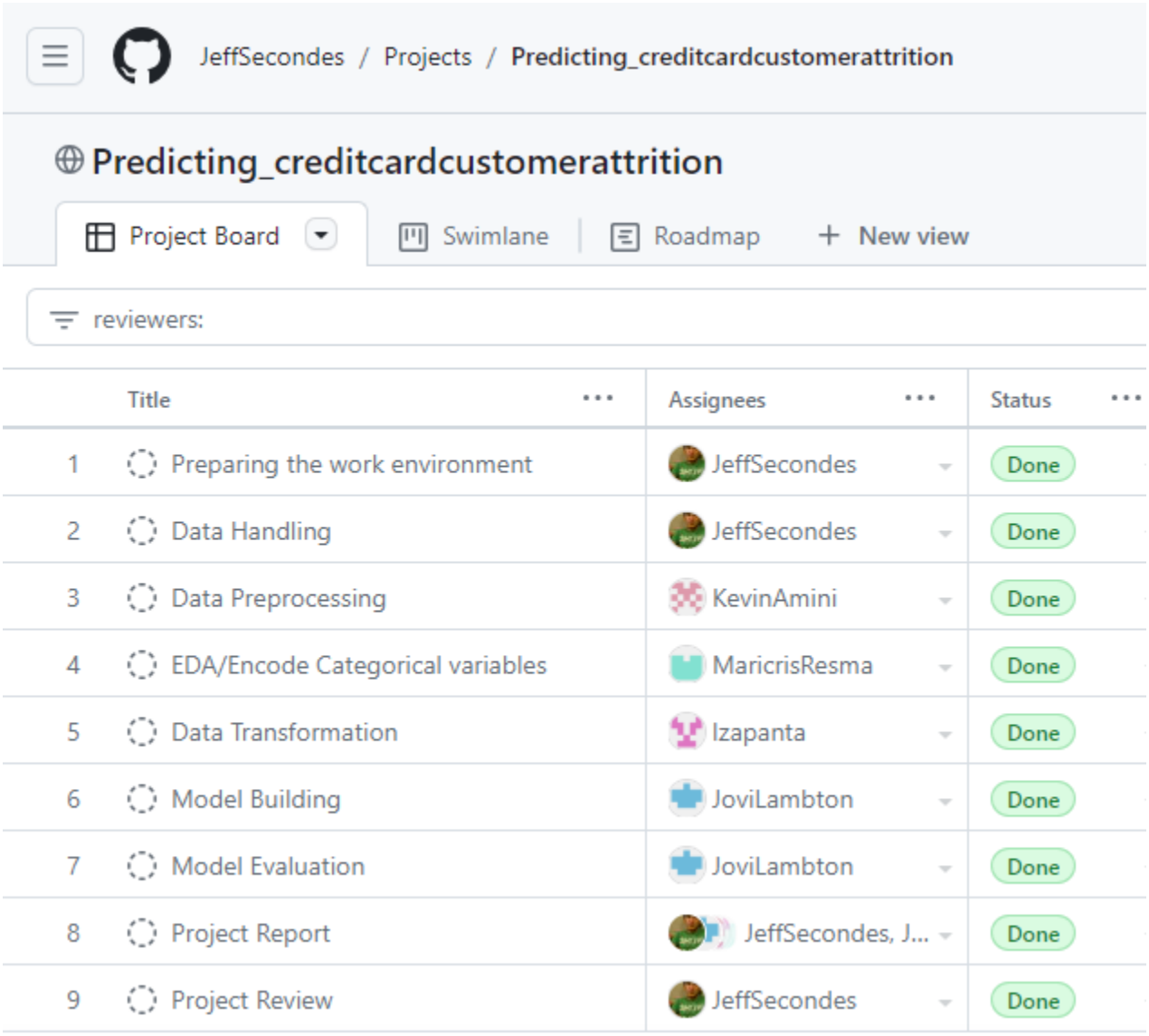
The Decision Tree model is efficient at predicting credit card customer segmentation because it is naturally good at working with and making sense of the dataset's characteristics. Decision trees show the decision-making process in a clear and easy-to-understand way. This makes them especially useful when it's important to understand how to divide customers into groups. The model's ability to rank features by how important they are helps us learn more about which customer attributes have a big effect on the segmentation results.

Decision trees are great at figuring out complex interactions and relationships between features that don't follow a straight line. This is a useful skill in situations where linear models might not work well. This adaptability is very important for figuring out how to divide credit card customers into groups, since different factors may not interact in a straight line. Also, Decision Trees can handle categorical variables without a lot of extra work, so they can handle the fact that some customer attributes, like gender or card category, are categorical.

The model's ability to work with datasets that aren't balanced is helpful for credit card customer segmentation, where the number of customers in different groups may vary. Tuning hyperparameters like tree depth and minimum samples per leaf is easy, so it's easy to find the right balance between model complexity and generalization. Decision Trees can also be used with ensemble methods like Random Forests to improve prediction even more by combining the best features of several trees and reducing the risk of overfitting.

Even though Decision Trees work, it's important to know their flaws, like how they can be affected by noisy data. The Decision Tree model was chosen because of the unique features of the credit card customer segmentation dataset. Its usefulness has been proven through extensive testing, model evaluation, and a thorough comprehension of the business context and objectives.

Project task contribution



	Title	Assignees	Status
1	Preparing the work environment	JeffSecondes	Done
2	Data Handling	JeffSecondes	Done
3	Data Preprocessing	KevinAmini	Done
4	EDA/Encode Categorical variables	MaricrisResma	Done
5	Data Transformation	Izapanta	Done
6	Model Building	JoviLambton	Done
7	Model Evaluation	JoviLambton	Done
8	Project Report	JeffSecondes, J...	Done
9	Project Review	JeffSecondes	Done

END