CHURN PREDICTION IN BANKING USING MACHINE LEARNING

ECE 552 – BIG DATA TECHNOLOGIES

FINAL PROJECT – GROUP 12

GEORGE MASON UNIVERSITY

Team Members:

Anil Kumar Mureboina – G01372036

Viswa Sujit Reddy Challa – G01379526

Aakash Reddy Tummala – G01357366

CHURN PREDICTION IN BANKING USING MACHINE LEARNING

Anil Kumar Mureboina, Vishwa Sujit Reddy Challa, Aakash Reddy Tummala

Group 12, GMU ECE 552 Big Data Technologies, Spring 2023

Abstract - This paper analyzed the BankChurner dataset in the banking industry, employing classification machine learning models and PySpark. The results showed that these models had exceptional accuracy, precision, recall, and F1 score. Visualization techniques were used to gain insights from the data, with String Indexer used for categorical feature encoding and Vector Assembler creating feature vectors. Parquet format was used for efficient data storage and retrieval. This work demonstrated the power of classification and regression modeling in understanding and predicting customer churn in the banking sector.

Keywords: Visualization, Bank, Modeling, PySpark, Classification, Regression, parquet, String Indexer, Vector Assembler.

I. INTRODUCTION

Customer attrition, also known as churn, is a major challenge for financial institutions in the competitive banking market. To anticipate and comprehend customer churn, banks must use regression analysis techniques on large datasets like the BankChurner dataset. The BankChurner dataset is an important tool for performing regression analysis with the goal of finding the key drivers of customer turnover. Banks can quantify each factor's effect on churn, gauge its importance, and determine its direction using regression analysis. This research seeks to arm banking institutions with the information and resources required to actively retain their important customers and create long-term partnerships.

II. HARDWARE USED

The entire project has been performed and executed in an Azure Virtual Machine, whose specifications and configurations are as listed below:

Specification	**Value**
Operating System	Windows 10
Number of CPU Cores	4
RAM	16 GB

Table 1 Hardware used by VM.

III. SOFTWARE USED

Using PySpark technology, the entire project was carried out in Jupyter Notebook. In addition, MongoDb Comapss was used to import the data utilized for the analysis. Most of the data analysis and modeling within Jupyter Notebook has been performed with PySpark. As PySpark does not support complex visualizations due to which the Pandas, Matplotlib and few other libraries have been used exclusively for visualizations.

IV. DATASET [9]

This dataset was sourced from Kaggle which consists of data of around ten thousand customers. The BankChurner dataset provides a comprehensive collection of customer data, such as age, gender, education level, marital status, income category, and various banking-related attributes. Its primary objective is to facilitate the analysis and forecasting of customer churn or attrition in a banking context. This data set is useful for comprehending the factors that contribute to customer churn and developing effective strategies to reduce it. Financial institutions can gain insight into the causes of customer churn by analyzing the correlation between customer characteristics, banking behaviors, and churn outcomes. This dataset is useful for conducting regression analysis and other statistical modeling techniques to identify significant predictors of churn, enabling banks to make data-driven decisions and implement proactive retention strategies to retain valuable customers and maximize customer lifetime value.

Information on the individual attributes can be found in the table below:

Attribute Name	**Description**
CLIENTNUM	Identification number of a client.
Attrition_Flag	This column represents whether a customer has attired or not.
Customer_Age	It denotes the age of the customer.
Gender	This column indicates the gender of the customer.
Dependent_count	It represents the number of dependents the customer has.
Education_Level	This column represents the education level of the customer.
Marital_Status	It denotes the marital status of the customer.
Income_Category	This column categorizes the customer's income level.
Card_Category	It represents the category of the credit card the customer holds.
Months_on_book	It denotes the number of months the customer has been a customer of the bank.
Total_Relationship_Count	This column represents the total number of products held by the customer.
Months_Inactive_12_mon	It denotes the number of months the customer was inactive in the last 12 months.
Contacts_Count_12_mon	It denotes the number of months in the last 12 months during which the customer was inactive.
Credit_Limit	It denotes the credit limit of the customer.
Total_Revolving_Bal	This column represents the total revolving balance on the customer's credit card.
Avg_Open_To_Buy	It denotes the average unused credit limit of the customer.
Total_Amt_Chng_Q4_Q1	This column represents the total change in transaction amount from the fourth quarter to the first quarter.
Total_Trans_Amt	It denotes the total transaction amount made by the customer.
Total_Trans_Ct	This column represents the total number of transactions made by the customer.
Total_Ct_Chng_Q4_Q1	It denotes the total change in transaction count from the fourth quarter to the first quarter.
Avg_Utilization_Ratio	This column represents the average utilization ratio of the customer.

Table 2 Attributes in the dataset.

V. DATAFLOW AND ARCHITECTURE

The flow of the data that has been employed in this project can be viewed in the dataflow diagram below.

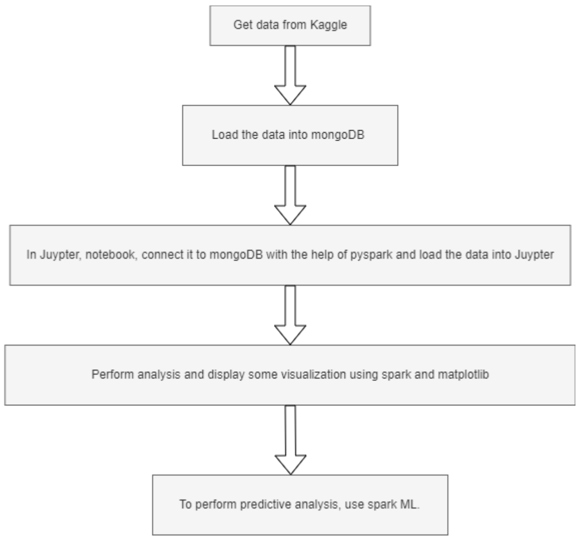

Get data from Kaggle
Load the data into mongoDB
In Juypter, notebook, connect it to mongoDB with the help of pyspark and load the data into Juypter
Perform analysis and display some visualization using spark and matplotlib
To perform predictive analysis, use spark ML.

Figure 1 Flow diagram of the project

The data was initially downloaded in csv format from Kaggle, started the mongodb server using docker and loaded dataset into a mongodb container. The container's data was then loaded into Jupyter Notebook using a mongodb and pyspark connector. The data were then converted to parquet format for analysis purposes. PySpark and matplotlib were utilized in tandem to perform data analysis and visualization. Using PySpark Machine Learning algorithms, a predictive data analysis was conducted. [5]

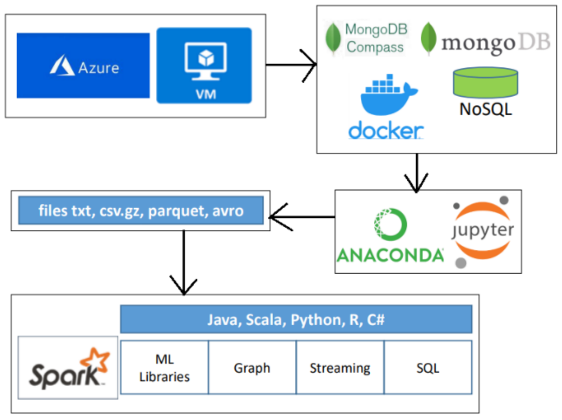
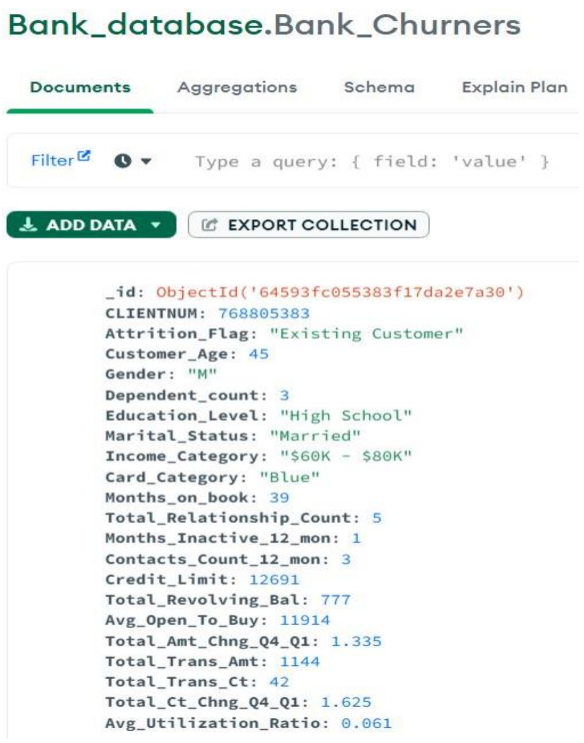


Figure 2 Software architecture of the project.

VI. LOADING THE DATA

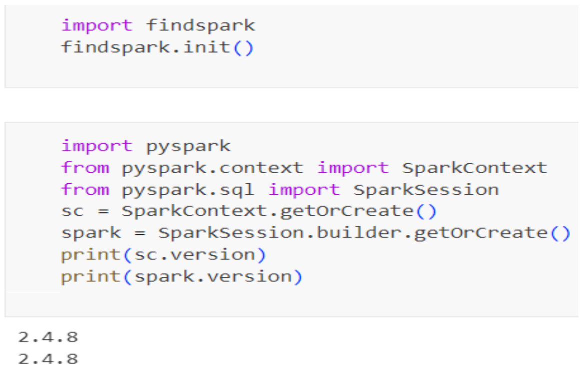
As the initial step in loading the csv dataset, a database with the name Bank_database was created in MongoDB, followed by the creation of a container with the same name. As shown below, the CSV data has been loaded into the container. We have created a container of a database in MongoDB compass and loaded.



```
_id: ObjectId('64593fc055383f17da2e7a30')
CLIENTNUM: 768805383
Attrition_Flag: "Existing Customer"
Customer_Age: 45
Gender: "M"
Dependent_count: 3
Education_Level: "High School"
Marital_Status: "Married"
Income_Category: "$60K - $80K"
Card_Category: "Blue"
Months_on_book: 39
Total_Relationship_Count: 5
Months_Inactive_12_mon: 1
Contacts_Count_12_mon: 3
Credit_Limit: 12691
Total_Revolving_Bal: 777
Avg_Open_To_Buy: 11914
Total_Amt_Chng_Q4_Q1: 1.335
Total_Trans_Amt: 1144
Total_Trans_Ct: 42
Total_Ct_Chng_Q4_Q1: 1.625
Avg_Utilization_Ratio: 0.061
```

Figure 3 Bank churners data in MongoDB.

The required libraries have been imported into Jupyter Notebook to execute the project as shown below:



```
import findspark
findspark.init()
```

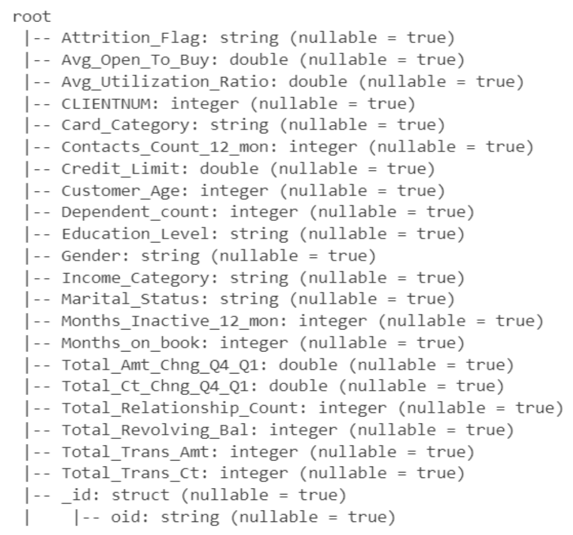
```
import pyspark
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
sc = SparkContext.getOrCreate()
spark = SparkSession.builder.getOrCreate()
print(sc.version)
print(spark.version)
```

```
2.4.8
2.4.8
```

Figure 4 Libraries imported for this project.

After the above step, a connection is established between mongoDb and Juypter notebook, then the data is loaded into Juypter notebook:

Displaying the schema of the dataset:



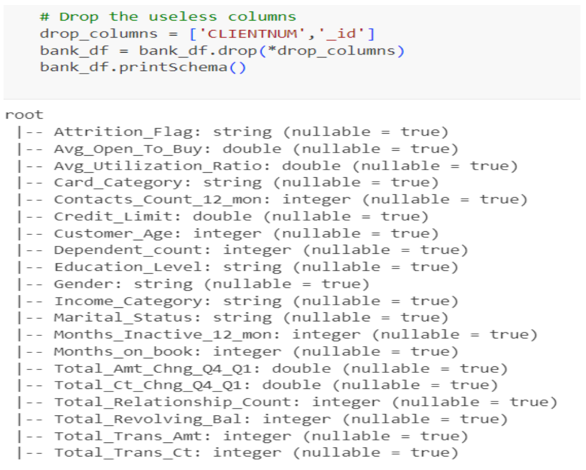
```
root
 |-- Attrition_Flag: string (nullable = true)
 |-- Avg_Open_To_Buy: double (nullable = true)
 |-- Avg_Utilization_Ratio: double (nullable = true)
 |-- CLIENTNUM: integer (nullable = true)
 |-- Card_Category: string (nullable = true)
 |-- Contacts_Count_12_mon: integer (nullable = true)
 |--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

Figure 5 Schema of the dataset.

VII. DATA PREPROCESSING

It is essential to analyze the available data and preprocess it in order to make it suitable for analysis. As part of the data preprocessing steps, the following operations have been performed on this dataset.

A. Only the columns that are significant for analysis have been retained, while clientnum and _id have been removed using the drop() function, as demonstrated below:



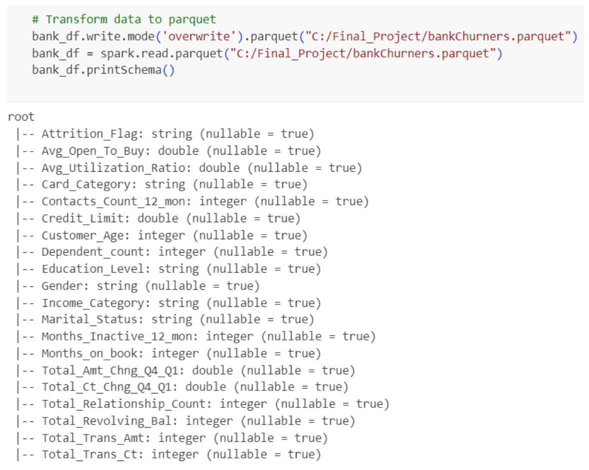
```
# Drop the useless columns
drop_columns = ['CLIENTNUM','_id']
bank_df = bank_df.drop(*drop_columns)
bank_df.printSchema()
```

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

Figure 6 Dropped unnecessary columns.

B. The rows with null values have been dropped to aid with analysis because there is plenty of data and there is no need for filling methods to be used.

C. There will be far less data to scan, which will need less memory, hence parquet conversion is advised. [3]



```
# Transform data to parquet
bank_df.write.mode('overwrite').parquet("C:/Final_Project/bankChurners.parquet")
bank_df = spark.read.parquet("C:/Final_Project/bankChurners.parquet")
bank_df.printSchema()
```

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

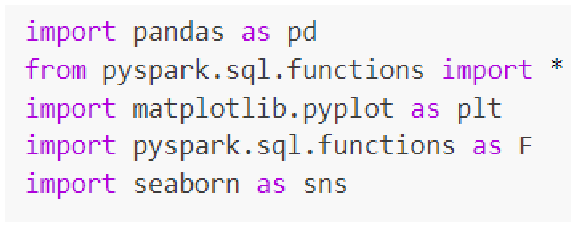
Figure 7 Schema after converting the data into parquet.

VIII. DATA EXPLORATION

The data converted to parquet will been explored using pySpark's functions like groupby(), count() and agg() to find patterns between different variables and define the direction of analysis and answer the following questions:

- Is there any feasible relationship between Marital status and Average utilization Ratio?
- What factors is the credit limit dependent on?
- Can existing customer attrition be anticipated using current existing customers and attritted customers?

We have imported a few additional libraries for data exploration. [8]



```
import pandas as pd
from pyspark.sql.functions import *
import matplotlib.pyplot as plt
import pyspark.sql.functions as F
import seaborn as sns
```

Figure 8 Importing libraries for Data exploration.

The analysis of the categorical variables in the dataset gives us a general idea of what the distribution is like, so each variable is plotted with respect to our response variable. The distribution of income category with respect to whether the customer is attrited or not. Based on the data and the graph we can see that most of the customer base is from less than 40K income category. The attrition is maximum from this category. There are a significant number of unknowns, which can lead to a potential breakthrough. There should be new methods employed to tackle unknowns.

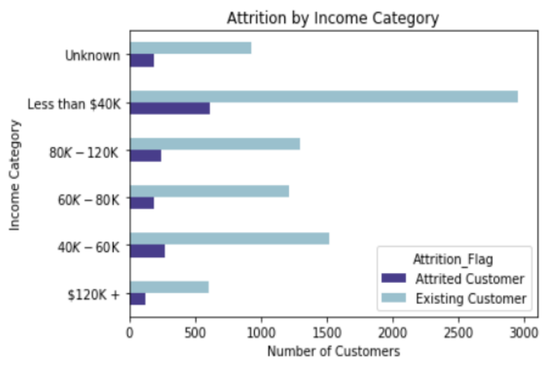


Figure 9 Attrition of customer by their income level.

The distribution of education level customer attrition, when visualized, shows that the customer base consists of people from seven different categories from which graduate category has the most customers and attrition and there are a significant number of unknowns that are equal to the number of uneducated customers.

Attrition_Flag	Attrited Customer	Existing Customer
Education_Level		
College	154	859
Doctorate	95	356
Graduate	487	2641
High School	306	1707
Post-Graduate	92	424
Uneducated	237	1250
Unknown	256	1263

Table 3 Customer attrition for their education level.

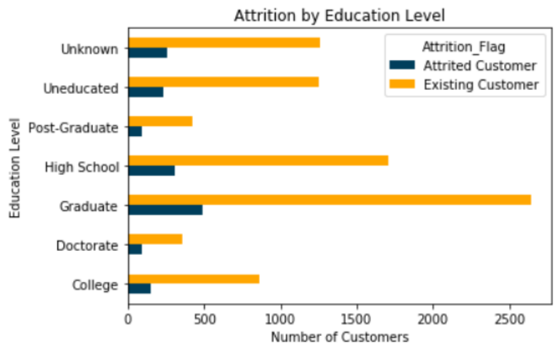


Figure 10 Attrition on customers by their education level.

The gender doesn't seem to be an influencing factor as both the genders seem to have a similar attrition ratio.

Attrition_Flag	Attrited Customer	Existing Customer
Gender		
F	930	4428
M	697	4072

Table 4 Customer attrition based on their gender.

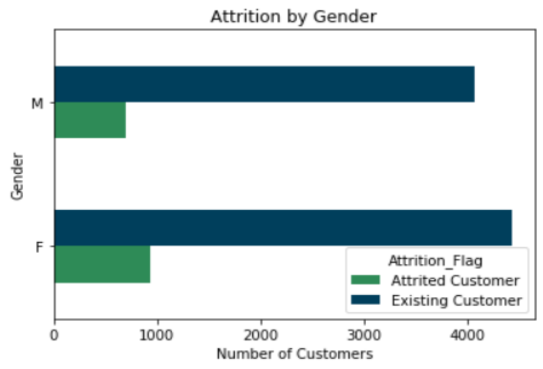


Figure 11 Customer attrition by their gender.

Most of the users have a blue card, hence the highest attrition and total users count belongs to the blue card category. We can see that platinum has the lowest user base.

Attrition_Flag	Attrited Customer	Existing Customer
Card_Category		
Blue	1519	7917
Gold	21	95
Platinum	5	15
Silver	82	473

Table 5 Customer attrition based on their card category.

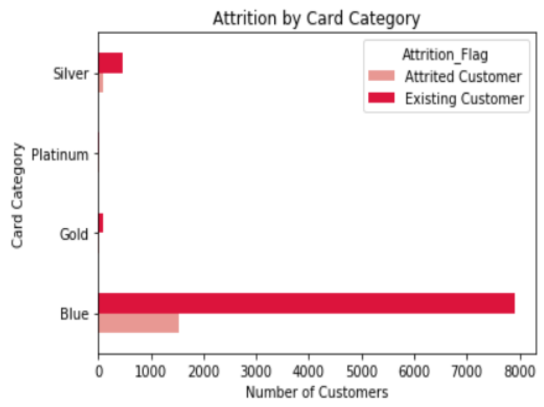
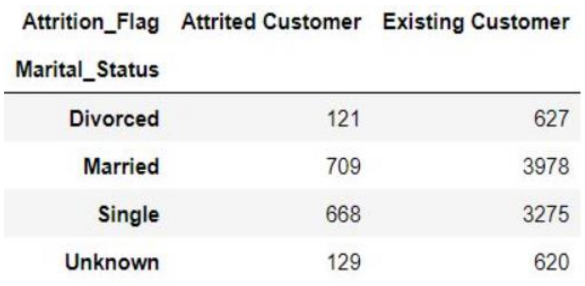


Figure 12 Customer attrition based on their car category.

In the marital status distribution, married and single statuses have similar attrition while married has a higher existing customer base making single a higher attrition ratio category.



Attrition_Flag	Attrited Customer	Existing Customer
Marital_Status		
Divorced	121	627
Married	709	3978
Single	668	3275
Unknown	129	620

Table 6 Customer attrition based on their marital status.

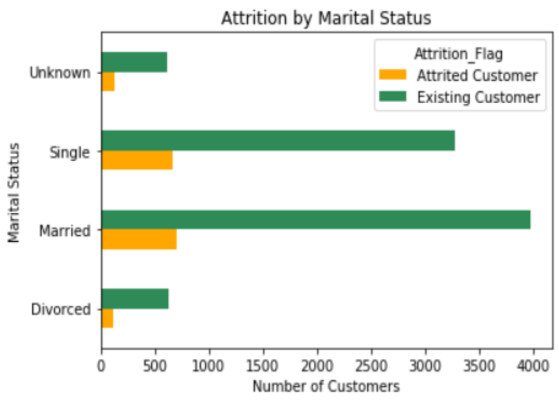


Figure 13 Customer attrition and their marital status.

After all the categorical variables, the numerical variables are also analyzed. The below graphs provide general statistics for the customers.

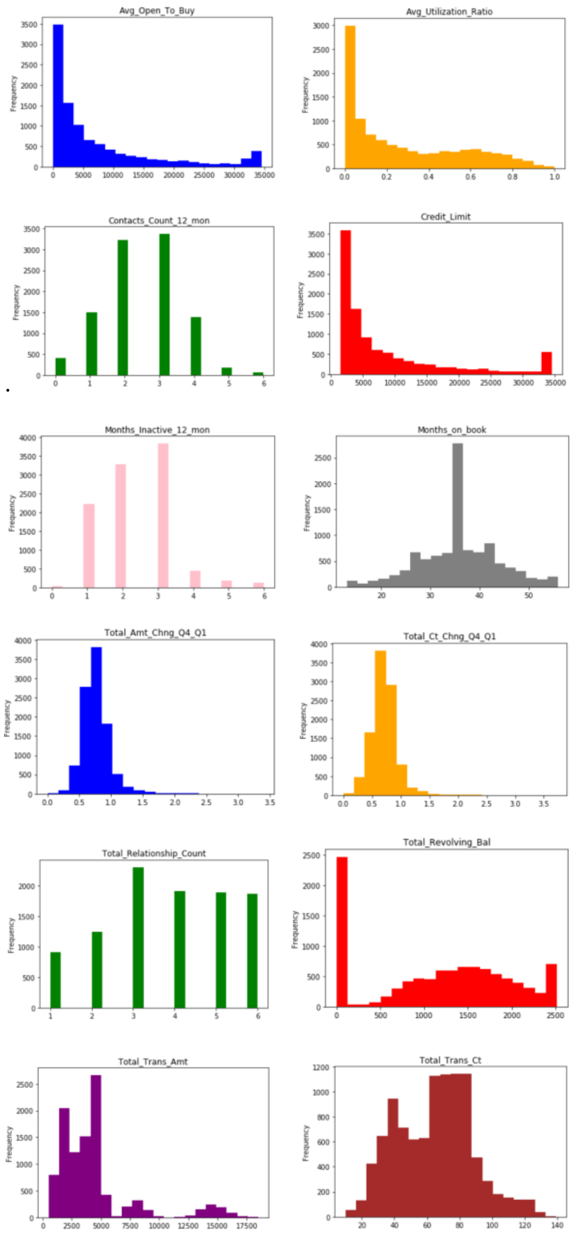
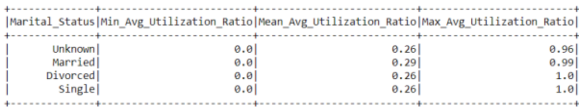


Figure 14 Histograms of the numerical variables

Answering the question, **Is there any feasible relationship between Marital status and Average utilization Ratio?** Initial analysis reveals that divorced customers do not have 100% utilization and can have utilization as low as zero percent.



```
+--------------+--------------------------+---------------------------+--------------------------+
|Marital_Status|Min_Avg_Utilization_Ratio|Mean_Avg_Utilization_Ratio|Max_Avg_Utilization_Ratio|
+--------------+--------------------------+---------------------------+--------------------------+
|       Unknown|                       0.0|                       0.26|                      0.96|
|       Married|                       0.0|                       0.29|                      0.99|
|      Divorced|                       0.0|                       0.26|                       1.0|
|        Single|                       0.0|                       0.26|                       1.0|
+--------------+--------------------------+---------------------------+--------------------------+
```

Table 7 Marital status & utilization ratio of the customers.

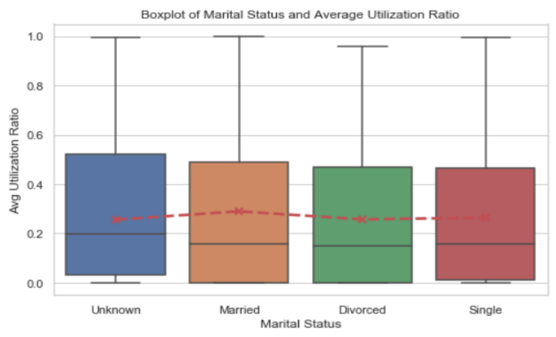


Figure 6 Marital status & average utilization ratio.

Married users have a wide range of utilization ratios including zero and hundred percent and also have the highest mean among all the other categories while sharing the median with single category which is the second followed by divorced and unknown. Married people, in general, have more people dependent on them and singles have a higher maintenance lifestyle so this might be a reasonable assumption towards a pattern to be checked for.

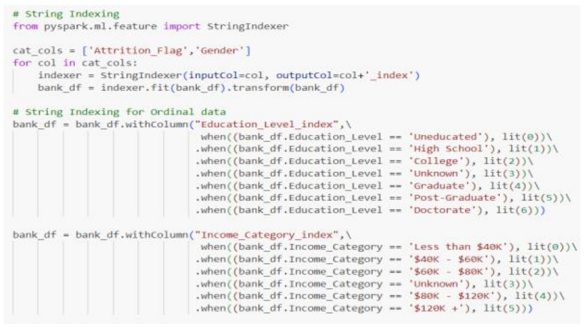
IX. DATA PREPARATION

The further question requires regression and correlation analysis hence the dataset needs to be further cleaned and arranged in a way that is suitable so that it can be fed to the algorithms for results.

The dataset preparation consists of the following:

A. String Indexing

String Indexing is the feature transformation process that helps in encoding the column of string labels into a column of corresponding label indices. A demonstration of this process would be the transformation of attrition flag, gender, education level and income category columns. [2]

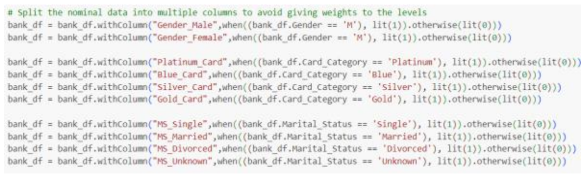


```
# String Indexing
from pyspark.ml.feature import StringIndexer

cat_cols = ['Attrition_Flag','Gender']
for col in cat_cols:
    indexer = StringIndexer(inputCol=col, outputCol=col+'_index')
    bank_df = indexer.fit(bank_df).transform(bank_df)

# String Indexing for Ordinal data
bank_df = bank_df.withColumn("Education_Level_index",\
                              when((bank_df.Education_Level == 'Uneducated'), lit(0))\
                             .when((bank_df.Education_Level == 'High School'), lit(1))\
                             .when((bank_df.Education_Level == 'College'), lit(2))\
                             .when((bank_df.Education_Level == 'Unknown'), lit(3))\
                             .when((bank_df.Education_Level == 'Graduate'), lit(4))\
                             .when((bank_df.Education_Level == 'Post-Graduate'), lit(5))\
                             .when((bank_df.Education_Level == 'Doctorate'), lit(6)))

bank_df = bank_df.withColumn("Income_Category_index",\
                              when((bank_df.Income_Category == 'Less than $40K'), lit(0))\
                             .when((bank_df.Income_Category == '$40K - $60K'), lit(1))\
                             .when((bank_df.Income_Category == '$60K - $80K'), lit(2))\
                             .when((bank_df.Income_Category == 'Unknown'), lit(3))\
                             .when((bank_df.Income_Category == '$80K - $120K'), lit(4))\
                             .when((bank_df.Income_Category == '$120K +'), lit(5)))
```

Figure 7 Code executed for string indexing.

B. Splitting the nominal data into multiple columns to avoid giving weight to them.

The dataset contains columns that have multiple levels that do not have any order. So encoding these columns would only result in a false outlook of the data. For example, gender, marital status, card category are such columns where each level in the columns any levels do not have indicating greatness with respect to other levels. Gender has male and female do not have an order etc.

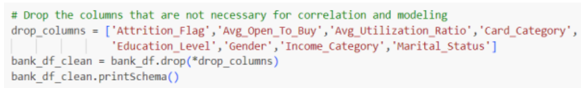


```
# Split the nominal data into multiple columns to avoid giving weights to the levels
bank_df = bank_df.withColumn("Gender_Male",when((bank_df.Gender == 'M'), lit(1)).otherwise(lit(0)))
bank_df = bank_df.withColumn("Gender_Female",when((bank_df.Gender == 'M'), lit(1)).otherwise(lit(0)))

bank_df = bank_df.withColumn("Platinum_Card",when((bank_df.Card_Category == 'Platinum'), lit(1)).otherwise(lit(0)))
bank_df = bank_df.withColumn("Blue_Card",when((bank_df.Card_Category == 'Blue'), lit(1)).otherwise(lit(0)))
bank_df = bank_df.withColumn("Silver_Card",when((bank_df.Card_Category == 'Silver'), lit(1)).otherwise(lit(0)))
bank_df = bank_df.withColumn("Gold_Card",when((bank_df.Card_Category == 'Gold'), lit(1)).otherwise(lit(0)))

bank_df = bank_df.withColumn("MS_Single",when((bank_df.Marital_Status == 'Single'), lit(1)).otherwise(lit(0)))
bank_df = bank_df.withColumn("MS_Married",when((bank_df.Marital_Status == 'Married'), lit(1)).otherwise(lit(0)))
bank_df = bank_df.withColumn("MS_Divorced",when((bank_df.Marital_Status == 'Divorced'), lit(1)).otherwise(lit(0)))
bank_df = bank_df.withColumn("MS_Unknown",when((bank_df.Marital_Status == 'Unknown'), lit(1)).otherwise(lit(0)))
```

Figure 8 Code executed for splitting nominal data.

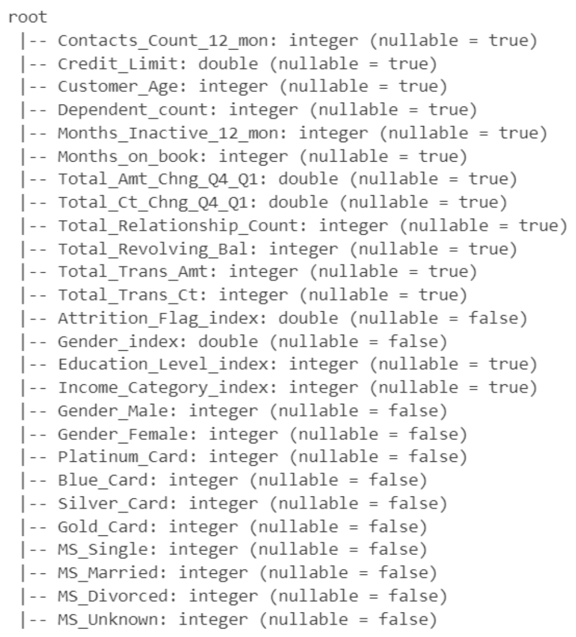
C. Dropping the unnecessary columns.

There are few columns which won't be necessary for answering the rest of the questions, thus their columns have been removed for simplification of analysis.



```
# Drop the columns that are not necessary for correlation and modeling
drop_columns = ['Attrition_Flag','Avg_Open_To_Buy','Avg_Utilization_Ratio','Card_Category',
                'Education_Level','Gender','Income_Category','Marital_Status']
bank_df_clean = bank_df.drop(*drop_columns)
bank_df_clean.printSchema()
```

Figure 9 Code for dropping the unwanted columns.

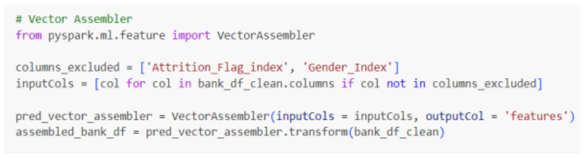


```
root
 |-- Contacts_Count_12_mon: integer (nullable = true)
 |-- Credit_Limit: double (nullable = true)
 |-- Customer_Age: integer (nullable = true)
 |-- Dependent_count: integer (nullable = true)
 |-- Months_Inactive_12_mon: integer (nullable = true)
 |-- Months_on_book: integer (nullable = true)
 |-- Total_Amt_Chng_Q4_Q1: double (nullable = true)
 |-- Total_Ct_Chng_Q4_Q1: double (nullable = true)
 |-- Total_Relationship_Count: integer (nullable = true)
 |-- Total_Revolving_Bal: integer (nullable = true)
 |-- Total_Trans_Amt: integer (nullable = true)
 |-- Total_Trans_Ct: integer (nullable = true)
 |-- Attrition_Flag_index: double (nullable = false)
 |-- Gender_index: double (nullable = false)
 |-- Education_Level_index: integer (nullable = true)
 |-- Income_Category_index: integer (nullable = true)
 |-- Gender_Male: integer (nullable = false)
 |-- Gender_Female: integer (nullable = false)
 |-- Platinum_Card: integer (nullable = false)
 |-- Blue_Card: integer (nullable = false)
 |-- Silver_Card: integer (nullable = false)
 |-- Gold_Card: integer (nullable = false)
 |-- MS_Single: integer (nullable = false)
 |-- MS_Married: integer (nullable = false)
 |-- MS_Divorced: integer (nullable = false)
 |-- MS_Unknown: integer (nullable = false)
```

Figure 10 Dataset schema after cleaning.

D. Vector Assembling

One of Apache Spark's feature transformers is the VectorAssembler. A group of input columns are combined into a single vector column for use as an input by machine learning algorithms. It creates a new column from a list of input column names that includes a vector of those column's values by assembling all the pertinent features into a single column, when working with machine learning algorithms that need input characteristics to be expressed as a single vector. [4]

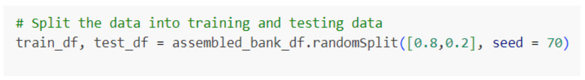


```
# Vector Assembler
from pyspark.ml.feature import VectorAssembler

columns_excluded = ['Attrition_Flag_index', 'Gender_Index']
inputCols = [col for col in bank_df_clean.columns if col not in columns_excluded]

pred_vector_assembler = VectorAssembler(inputCols = inputCols, outputCol = 'features')
assembled_bank_df = pred_vector_assembler.transform(bank_df_clean)
```

Figure 11 Code for vector assembler.

E. Splitting data for training and testing.

The amount of data, even though limited, is sufficient for the model to be trained and have some data left over with negligible improvement after a certain point. So rather than using all the data to train the model, we can split it into two parts and use one smaller part to evaluate the results and see if the model is accurately predicting the results. Here the training is randomly collected and has 80 percent of the values of the dataset and the testing dataset has the remaining 20 percent of values.



```
# Split the data into training and testing data
train_df, test_df = assembled_bank_df.randomSplit([0.8,0.2], seed = 70)
```

Figure 12 Code to split dataset for training and testing.

X. CORRELATION ANALYSIS

The question, **what factors is the credit limit dependent on?** can be answered by finding out the correlation between the credit limit and all the other columns in the dataset and the highly correlated values might induce a change in credit limit and higher the correlation, greater the change in credit limit when one of these factors are changed.

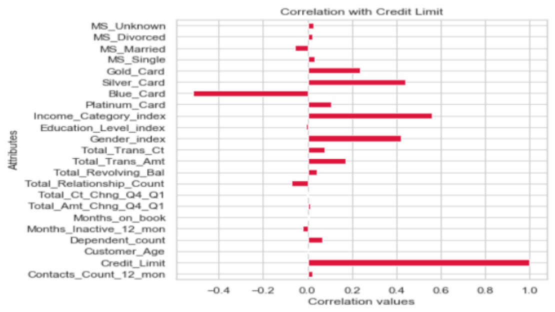


Figure 13 Correlation of attributes with credit limit.

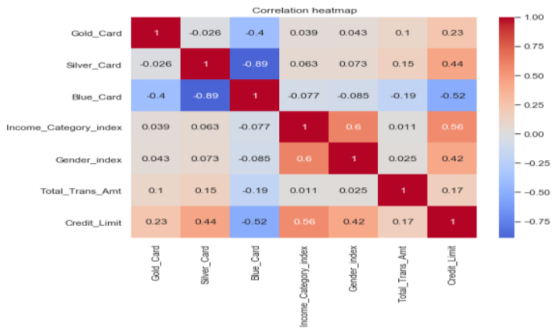


Figure 14 Heatmap of the credit limit correlation.

In the primary analysis of correlation, the findings show that the card category and gender have moderate correlation between 0.4-0.6 which is not indicative of a very high correlation but also is not low enough to ignore its effects of the credit limit.

So, a combination of factors among which card category, income category and gender play a greater role are crucial while deciding the credit limit of an applied customer.

XI. PREDICTIVE MODELING

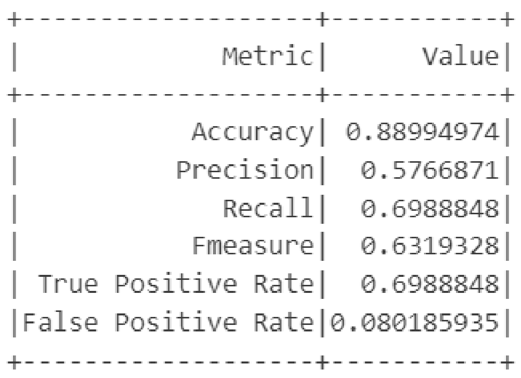
The previous steps ensure that the data is ready to be fed into the machine learning algorithms to train and get the prediction regarding the attrition of a certain customer given the details required significantly are provided. The question, **can existing customer attrition be anticipated using current existing customers and attritted customers?**

It can be answered using regression. Two types of regression models have been used and the one with higher accuracy will be chosen at the end.

1. Logistic Regression

The logistic regression statistical model is primarily used for binary classification which is exactly the type of classification of existing or attrited customers.
The model fits the training data, and the following are a few of the prediction values compared to the actual values in the test set. [6]

The data here shows that the model is reliable in classifying the attrition of a customer. [7]



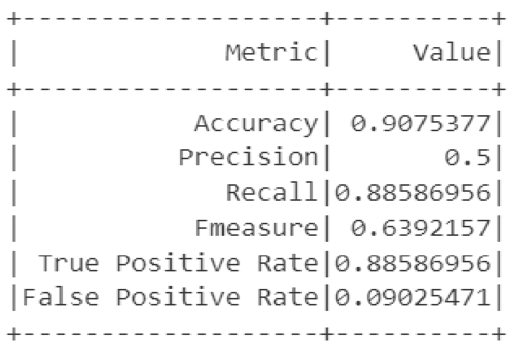
```
+------------------+----------+
|            Metric|     Value|
+------------------+----------+
|          Accuracy|0.88994974|
|         Precision| 0.5766871|
|            Recall| 0.6988848|
|          Fmeasure| 0.6319328|
| True Positive Rate| 0.6988848|
|False Positive Rate|0.080185935|
+------------------+----------+
```

Table 8 Evaluation metrics of logistic regression.

2. Random Forest Regression

The random forest regression model can be used for both classification and regression, which uses multiple decision trees to learn and predict the required.

The data here shows that the model is reliable in classifying the attrition of a customer. [7]



```
+------------------+----------+
|            Metric|     Value|
+------------------+----------+
|          Accuracy| 0.9075377|
|         Precision|       0.5|
|            Recall|0.88586956|
|          Fmeasure| 0.6392157|
| True Positive Rate|0.88586956|
|False Positive Rate|0.09025471|
+------------------+----------+
```

Table 9 Evaluation metrics of the random forest tree.

XII. EVALUATION METRICS

The MulticlassMetrics() function, has been used to evaluate the metrics of the model. The features that the functions displays are Accuracy, Precision, Recall, Fmeasure, True Positive Rate, False Positive Rate [1]

Accuracy: The total accuracy of the model's predictions is measured by accuracy. It determines what percentage of all cases (including true positives and true negatives) were correctly categorised. Accuracy is determined by:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Where, TP represents true positives, TN represents true negatives, FP represents false positives, and FN represents false negatives.

Precision: Precision measures the model's accuracy in distinguishing true positive cases from anticipated positive ones. It is the proportion of real positives to all anticipated positive events, including both real and false positives. The formula for precision is:

Precision = TP / (TP + FP)

Recall/ True Positive Rate: The model's capacity to recognize positive cases among the real positive examples is measured by recall. It is the proportion of genuine positive cases (including true positives and false negatives) to the total number of true positives. Recall is determined by:

Recall = TP / (TP + FN)

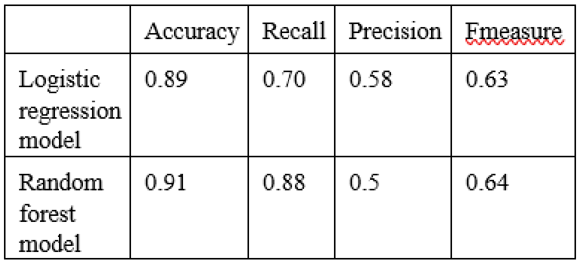
Fmeasure: Precision and recall are combined into a single statistic called the F-measure, which balances both measurements. It is the precision and recall harmonic mean. The F-measure is determined as follows:

F-measure = (2 * Precision * Recall) / (Precision + Recall)

False Positive Rate: The percentage of negative events that are mistakenly categorized as positive is measured by the false positive rate. It measures the proportion of false positives to all truly negative events.

False Positive Rate = FP / (FP + TN)

The accuracies and metrics of all the models can be seen below:



	Accuracy	Recall	Precision	Fmeasure
Logistic regression model	0.89	0.70	0.58	0.63
Random forest model	0.91	0.88	0.5	0.64

Table 10 Accuracies of Predictive Models.

The slightly higher accuracy and recall of the random forest regression makes it the better model. Even though consideration of these statistics is not universal they make a good case for this problem.

XIII. CONCLUSION

In conclusion, this paper analyzed the Bank Churners dataset using classification machine learning models to understand and predict customer churn in the banking industry. Several classification algorithms, such as logistic regression, random forest, and support vector machines, were employed to build predictive models.

The findings of this study demonstrated that machine learning models can effectively predict customer churn in the banking sector. The models achieved notable performance metrics, including high accuracy, precision, recall, and F1 score. These results indicate the models' ability to identify potential churners and assist banks in taking proactive measures to retain customers. This knowledge can help banks create focused client retention strategies and raise satisfaction levels.

Overall, this study adds to the research on churn prediction and demonstrates the usefulness of categorization machine learning methods in the banking sector. Future studies should investigate the incorporation of more sophisticated algorithms or data sources to further improve accuracy.

References:

[1] Medium. (2018, March 15). Accuracy, Precision, Recall or F1? Retrieved from https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

[2] Apache Spark. (n. d.). Extracting, transforming and selecting features. Retrieved from https://spark.apache.org/docs/latest/ml-features.html#stringindexer

[3] SparkByExample. (2023 February 03). PySpark Read and Write Parquet File. Retrieved from https://sparkbyexamples.com/pyspark/pyspark-read-and-write-parquet-file/

[4] Apache Spark. (n. d.). VectorIndexer. Retrieved from https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.VectorIndexer.html

[5] MongoDB. (n. d.). Spark Connector Python Guide. Retrieved from https://www.mongodb.com/docs/spark-connector/current/python-api/

[6] Medium. (2020, May 04). Logistic Regression with PySpark. Retrieved from https://medium.com/swlh/logistic-regression-with-pyspark-60295d41221

[7] Sparkitecture. (n. d.). Model Evaluation. Retrieved from https://www.sparkitecture.io/machine-learning/model-evaluation

[8] Pandas. (n. d.). pandas.DataFrame.pivot. Retrieved from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html

[9] “Churning customers 98.95% detected,” kaggle.com. https://www.kaggle.com/code/winternguyen/churning-customers-98-95-detected/input