FACULTY OF ENGINEERING, ENVIRONMENT & COMPUTING

(7082CEM)

Coursework

Big Data Analytics and Visualization Using PySpark

MODULE LEADER: Dr. Marwan Fuad

Student Name: Abhijith Shaji

SID: 10915804

CREDIT CARD CUSTOMER CHURN PREDICTION USING LOGISTICAL REGRESSION AND RANDOM FOREST

I can confirm that all work submitted is my own: Yes

1. INTRODUCTION

Artificial Intelligence plays a vital role in the field of information technology, particularly in machine learning. In this article, I will discuss and show how PySpark is used for the project using tableau machine learning to create visualisations. We begin with a comprehensive description of the packages that will be utilised and how they may be installed during the whole research. Anaconda navigator, Python, Spark and Tableau are the stacks that I have utilised. I also described in a Jupyter notebook how to build a new python file.

In this course work we are dealing with the dataset of credit card customers of bank. We know that customers are every important for financial institutions. Losing customers will always affect the Bank business. If we can predict the situation of a customer ending one service, we can avoid and help the bank to retain the customer. To achieve this, we have created two PySpark ML model using Logistical regression and Random Forest algorithm. In this course work, we cover through all the procedure including the setp up Pyspark for Jupyter notebook, Data pre-processing, EDA, ML model creation and implementation.

In each step I have tried to cover all possible scenarios that we can have while using the PySpark. Here I have used accuracy and area under the ROC curve to choose the best model among these two. I have implemented every step using PySpark libraries only.

The aims of this course study are extensive reading and research. New technologies like PySpark have been intriguing. PySpark modelling and prediction machine learning is straightforward and understandable. This study helps to better analyse and understand data using different images and display tables. Tableau was highly intriguing for the display component.

2. ENVIRONMENT SETUP

Here we are discussing the implementation of our environment for setting and execution of our project. In this section, we will explicitly discuss the set-up of PySpark and discuss in-depth coming section.

2.1. BACKGROUND STUDY

SPARK

Apache Spark has been widely used open-source software, in Bigdata analytics for processing large scale data. Initially in 2009 it was just research project. Spark's objective was to create a new framework, optimized for fast-track processing such as machine learning and interactive data analysis while maintaining Hadoop MapReduce's scalability and fault tolerance.

Spark is tightly integrated into the Hadoop ecosystem, allowing it to interact seamlessly with Hadoop Distributed File System (HDFS), Apache Hive, and other Hadoop components. When the size of the data is too large for Spark to handle in-memory, Hadoop's HDFS functionality can assist in overcoming that barrier. Spark makes use of the best parts of Hadoop, including HDFS for data reading and storage, MapReduce for optional processing, and YARN for resource allocation. Figure 1 describes the framework of spark

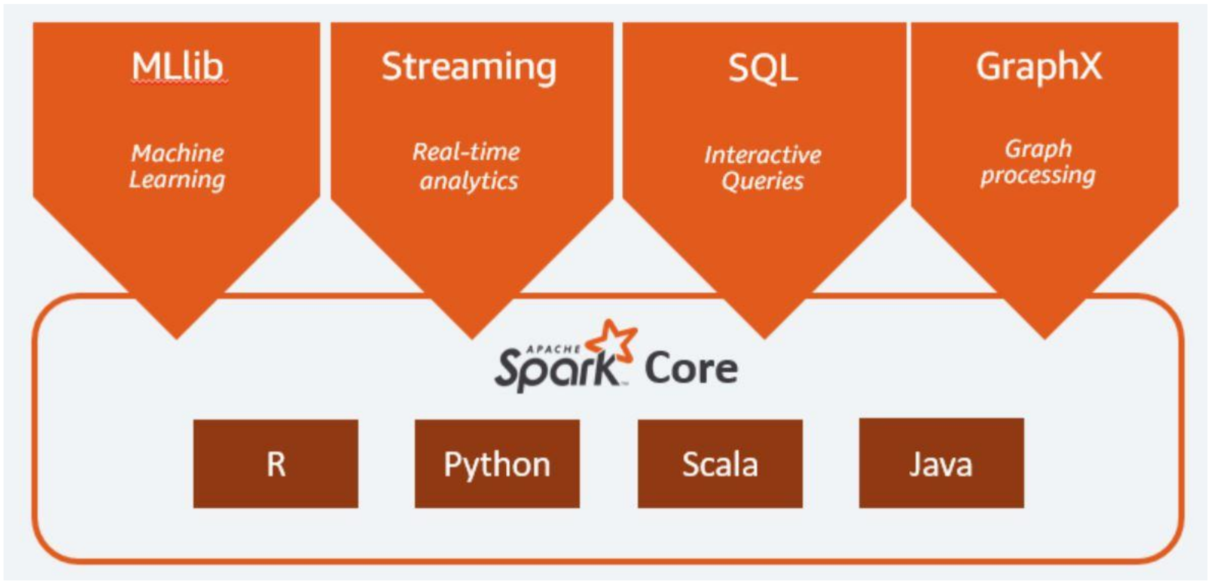


Figure 1 Spark framework

The SQL, machine learning, and streaming are incorporated into the Spark to improve the large data analysis. The Spark is supplied with Programming APIs for R, Scala, Python, and Java. Python is the unique language that makes the process of the real-time screen and machine education faster for a broad spectrum of libraries. I have chosen python to show a solid understanding of the given semi structured data set after considering all of the above-mentioned advantages in this study. I've discussed the combination of Python with Spark in the following parts.

SPARK CONTEXT

SparkContext is an entry point for any function of Spark. When any Spark application is executed, a driver program starts with the main function and is then launched. The driver program then works on working nodes within the executors. SparkContext uses Py4J to initiate a javaspark before a JavaSparkContext is created. However, PySpark has SparkContext as 'sc' by default, thus creating a new SparkContext will never work [1].

SPARK RDD

PySpark's RDD (Resilient Distributed Dataset) is a fault-tolerant, immutable distributed collection of objects. An RDD is immutable, which means it cannot be changed once it is created. RDD divides each record into logical divisions, which can be calculated on various cluster nodes. In other words, RDDs are a collection of objects comparable to a Python list, with the exception that RDDs are calculated over many processes spread across numerous physical servers, also known as nodes in a cluster, whereas a Python collection lives and processes in just one process.

Furthermore, RDDs provide data abstraction of data division and distribution suited to conduct computations in parallel on multiple nodes; we don't have to worry about parallelism while executing transformations on RDDs because PySpark provides it by default [2].

SPARK DataFrame

DataFrame is a series of rows ordered in order with named columns. Examples include a table in an Excel database or a column header. It has a similarity to RDD as well. It is easier and more efficient to investigate exploration and provide aggregated statistics on huge data sets.

PySpark

The Apache Spark in Python's PySpark is an interface. Not only can Spark applications be created with Python APIs but also PySpark shell can be used to interactively scan your data for a distributed environment. Most of Spark's capabilities including Spark SQL, DataFrame, streaming, MLlib, and Spark Core are supported by PySpark. PySpark also enables you to connect with the Apache Spark and Python programming language utilizing Resilient Distributed Datasets (RDDs). The benefit of the Py4j library has been obtained. Py4J is a popular PySpark-integrated module that permits dynamical interaction between Python and JVM objects. A number of libraries for creating efficient applications are available in PySpark. In addition, some external libraries are compatible as well [3]. Figure 2 shows the feature of PySpark.

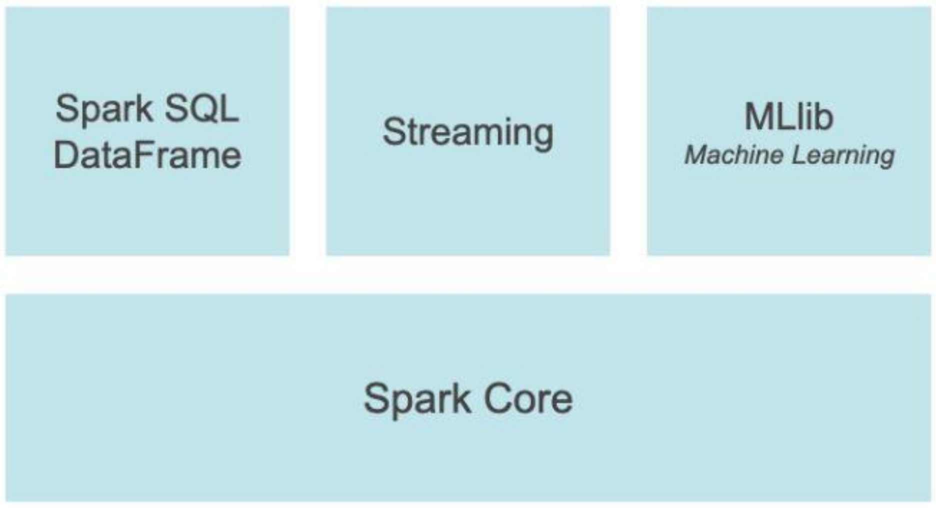


Figure 2 Features of PySpark

A. Spark SQL and DataFrame

Spark SQL is a structured data processing Spark module. It provides an abstracted programming called DataFrame as well as a SQL Query Engine provided.

B. Streaming

Significant interactive and analytically based applications covering both streaming and historical data while keeping Spark's simplicity in usage and fault tolerance allow Apache Spark's streaming component on top of Spark.

C. MLlib

MLlib is a scalable machine learning library developed on top of Spark. It provides a consistent collection of high-level APIs for building and tuning realistic machine learning pipelines.

D. Spark Core

Spark Core is the Spark platform's general execution engine. The Spark Core is the foundation for all other Spark features. It has RDD and in-memory computing features.

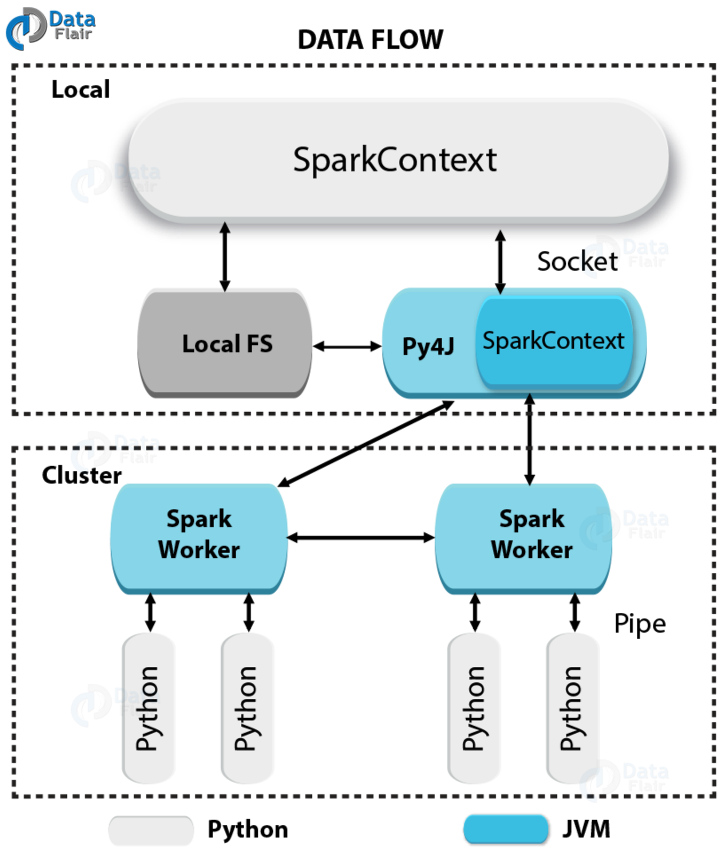


Figure 3 Data flow diagram of PySpark

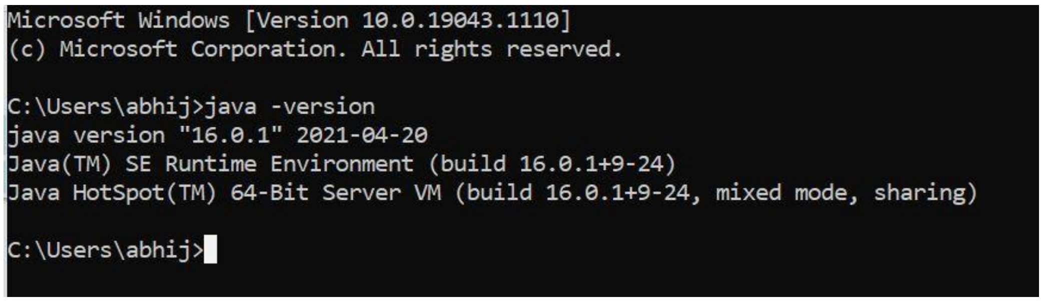
2.2.INSTALLATION MODULES AND SETUP

Machine that we have used here to carry out our experiment have 16GB RAM, 1TB HDD, 256GB SSD and Windows 10 Home operating system. Here we are discussing the detailed steps of installation and setup.

Step 1: As we discussed in earlier section for PySpark we need JAVA in our environment so first we are trying to add JAVA to our environment. For that we need to check that JAVA already installed in our environment or not. To check that we can use below command in Command Prompt.

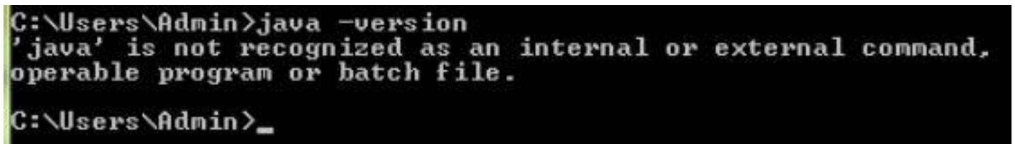
```
java -version
```

If Java is already installed in our environment it will display the version details like in the figure 4. Otherwise, we will get an error message as in the figure 5.



```
Microsoft Windows [Version 10.0.19043.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\abhij>java -version
java version "16.0.1" 2021-04-20
Java(TM) SE Runtime Environment (build 16.0.1+9-24)
Java HotSpot(TM) 64-Bit Server VM (build 16.0.1+9-24, mixed mode, sharing)

C:\Users\abhij>
```

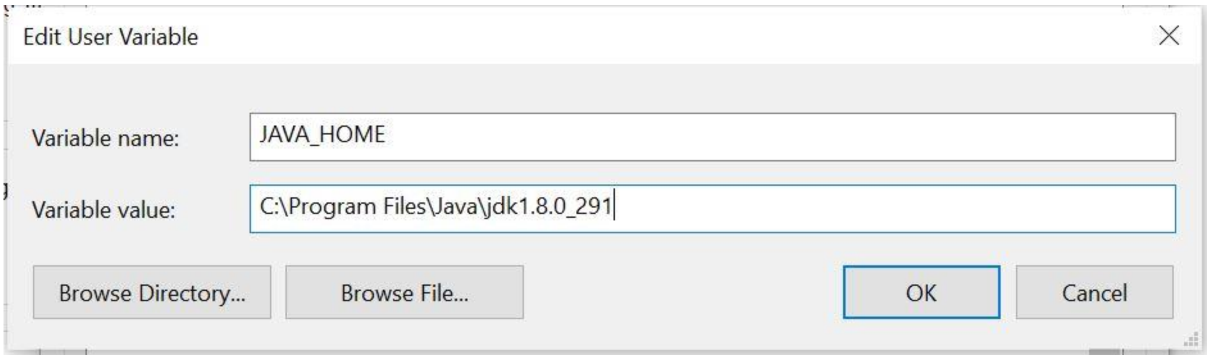
Figure 4 JAVA version result



```
C:\Users\Admin>java -version
'java' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Admin>_
```

Figure 5 JAVA not installed message

Once we install JAVA in our environment, we must add set environment variable JAVA_HOME and add it in the path variable. Figure 6 shows setting the variable. Otherwise, the system won't be able access it.



Edit User Variable

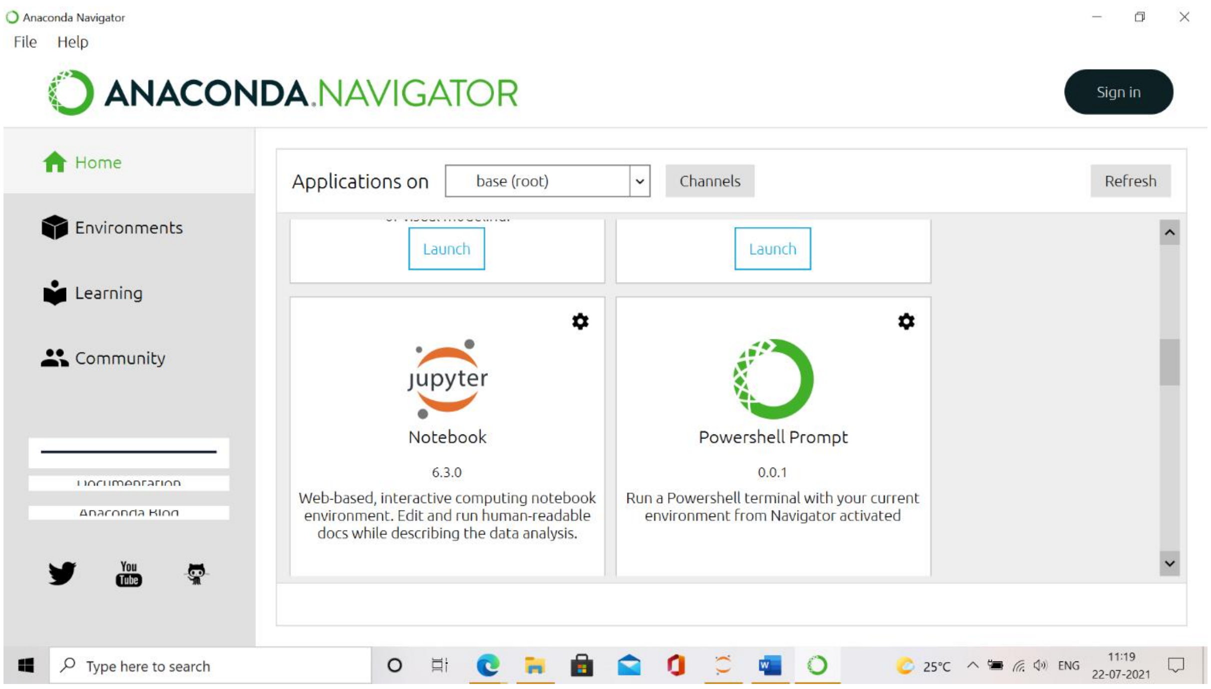
Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk1.8.0_291

Figure 6 JAVA_HOME environment variable

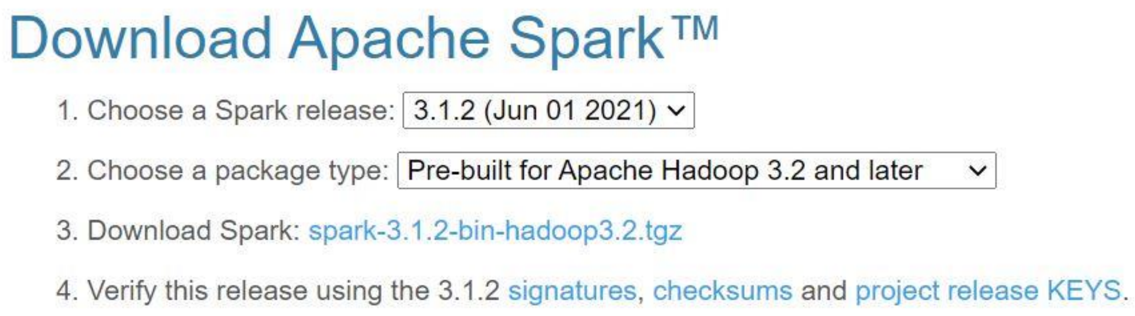
Step 2: Installing Anaconda, it is a Python distribution with pre-built packages that are extensively used in data research. The Anaconda installation covers the administration of the Conda package besides the pre-configured Python packages and other utilities. Configure Python environments and instal additional packages with the standard Anaconda installation using the command line Conda Package

Manager. Anaconda Navigator is an Anaconda distribution graphical user interface (GUI) that facilitates the setup, installation and utilisation of products such as Jupyter Notebook. For this we can directly download and install anaconda any system.



2 Anaconda Navigator

Step 3: Download and Install Spark. To install spark, we download latest version 3.1.2 from apache foundation. From below we can see the version and package details of the spark we have used in this experiment.



Download Apache Spark™

1. Choose a Spark release: 3.1.2 (Jun 01 2021)
2. Choose a package type: Pre-built for Apache Hadoop 3.2 and later
3. Download Spark: spark-3.1.2-bin-hadoop3.2.tgz
4. Verify this release using the 3.1.2 signatures, checksums and project release KEYS.

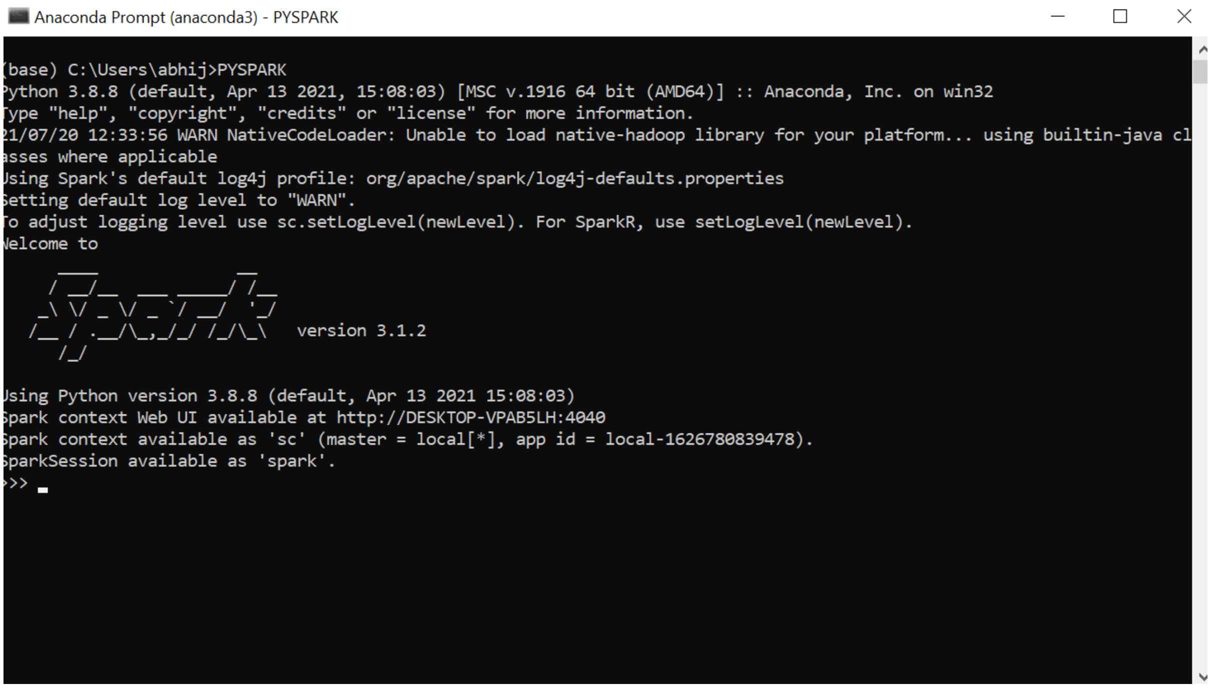
Once we download the .tgz file we extract the file and place it in C:\Spark then we set the environment variables as below. And also update the PATH variable with spark bin older location.

```
SPARK_HOME      = C:\Spark\spark-3.1.2-bin-hadoop3.2
HADOOP_HOME     = C:\Spark\spark-3.1.2-bin-hadoop3.2
```

```
C:\Spark\spark-3.1.2-bin-hadoop3.2\bin
```

Step 4: Download and setup winutils.exe. For this we can directly download from the GitHub, once download the file copy and paste in the bin folder of spark

Step 5: Now we can check the PySpark installation in Anaconda prompt typing the pyspark command. If the installation is successful, we get the result in the prompt as below.


Anaconda Prompt (anaconda3) - PYSPARK

```
(base) C:\Users\abhij>PYSPARK
ython 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
ype "help", "copyright", "credits" or "license" for more information.
21/07/20 12:33:56 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
etting default log level to "WARN".
o adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
elcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/

Using Python version 3.8.8 (default, Apr 13 2021 15:08:03)
park context Web UI available at http://DESKTOP-VPAB5LH:4040
park context available as 'sc' (master = local[*], app id = local-1626780839478).
parkSession available as 'spark'.
>>>
```

PySpark inside Jupyter notebook:

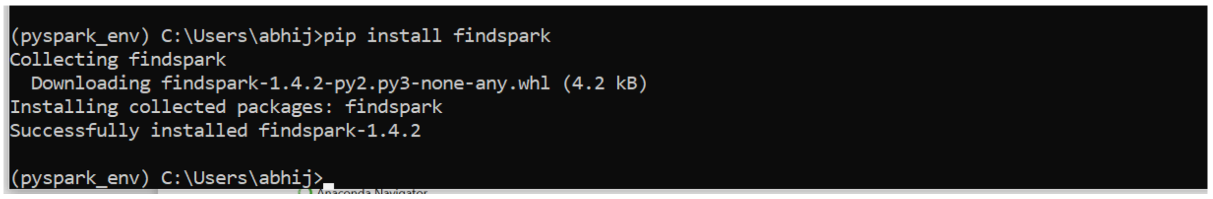
The web-based program Jupyter Notebook is used extensively to edit, operate, and share Python code. It gives you a lot of freedom in changing and executing parts of your code. This study used Jupyter Notebook as a code editor. It is also supplied with the Jupyter notebook Anaconda package.

In order to access PySpark via jupyter, we must install a python module named findspark. The figure shows the result of the execution of the command to install the command below.

conda install -c conda-forge findspark

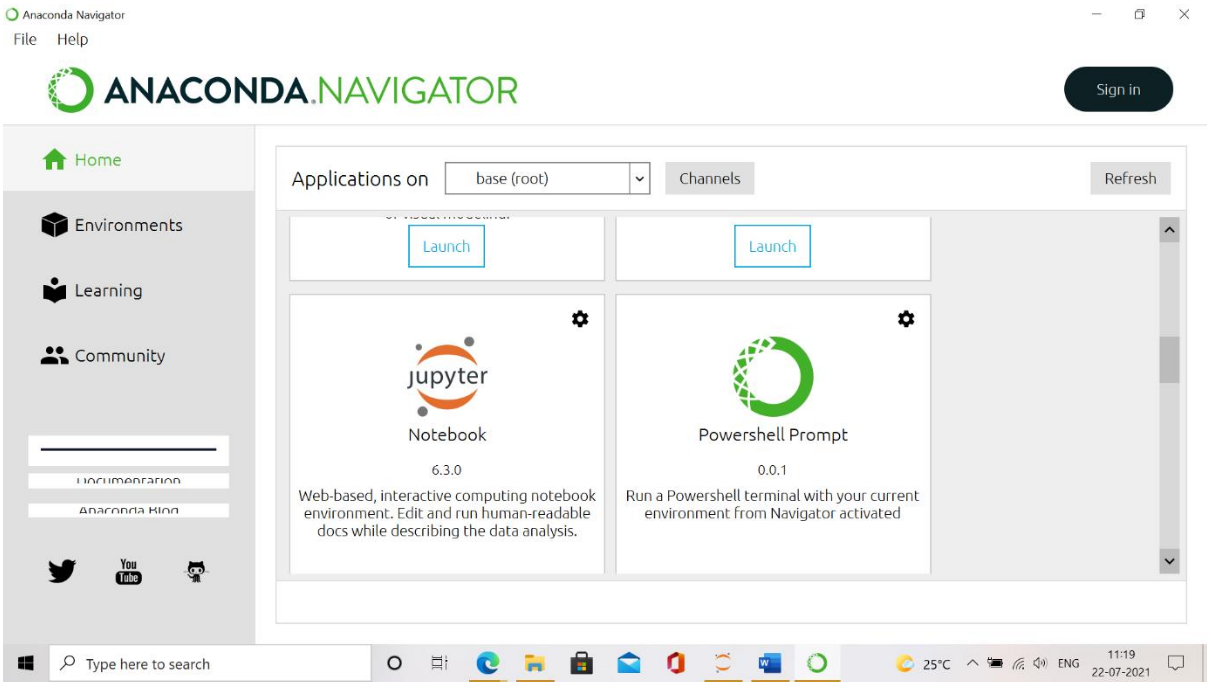
We can also install by below command in pyspark environment

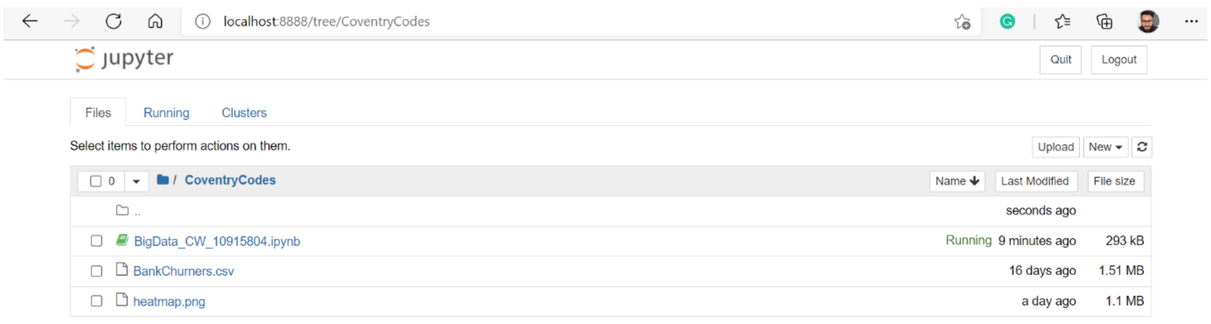
pip install findspark



```
(pyspark_env) C:\Users\abhij>pip install findspark
Collecting findspark
  Downloading findspark-1.4.2-py2.py3-none-any.whl (4.2 kB)
Installing collected packages: findspark
Successfully installed findspark-1.4.2

(pyspark_env) C:\Users\abhij>
```

Then we need to launch our jupyter notebook from anaconda navigator then we can access jupyter notebook can be accessed through web browser here am using edge that is my default web browser.


Anaconda Navigator
File Help
ANACONDA.NAVIGATOR
Sign in
Home
Environments
Learning
Community
Applications on base (root)
Channels
Refresh
Launch
Launch
Notebook
6.3.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Type here to search
25°C
ENG
11:19
22-07-2021



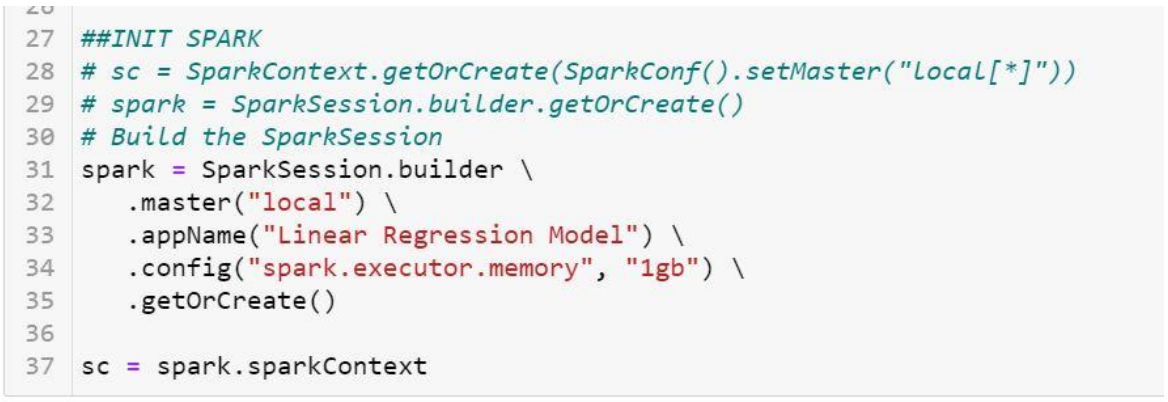
Select items to perform actions on them.

/ CoventryCodes		Last Modified	File size
..		seconds ago	
BigData_CW_10915804.ipynb	Running	9 minutes ago	293 kB
BankChurners.csv		16 days ago	1.51 MB
heatmap.png		a day ago	1.1 MB

Before we start with jupyter notebook we need to make sure that we can access pyspark in jupyter notebook below code can help in that.

```
In [6]:  import findspark
         findspark.init()
         findspark.find()

Out[6]: 'D:\\Software Centre\\spark-3.1.2-bin-hadoop3.2'
```

Once we have found that we can access spark we need to create a spark session and spark context ‘sc’ to proceed the experiment using below code.



```
##INIT SPARK
# sc = SparkContext.getOrCreate(SparkConf().setMaster("local[*]"))
# spark = SparkSession.builder.getOrCreate()
# Build the SparkSession
spark = SparkSession.builder \
    .master("local") \
    .appName("Linear Regression Model") \
    .config("spark.executor.memory", "1gb") \
    .getOrCreate()

sc = spark.sparkContext
```

3. DATASET

The bank always worried by the increasing number of clients who leave their credit card services. we should really like to know who would be churned so that you may proactively go to the consumer to give better services and shift the customers' choices in the opposite direction. In this dataset we have 23 attributes including the target attribute there are 22 input attributes most of them are might not necessary for our experiment. During the data pre-processing we will handle these scenarios. In our dataset 10127 records. Each o these records contains the actual customer data with out any personal information. From the dataset we can see that only 16.07 percent of our clients have churned. As a result, training our algorithm to anticipate churning consumers can be challenging. From below table we can see the attributes and its details.

Attribute	Detailes
CLIENTNUM	Client number. Unique identifier for the customer holding the account
Customer_Age	Demographic variable - Customer's Age in Years
Gender	Demographic variable - M=Male, F=Female
Dependent_count	Demographic variable - Number of dependents
Education_Level	Demographic variable - Educational Qualification of the account holder (example high school, college graduate, etc.)
Marital_Status	Demographic variable - Married, Single, Divorced, Unknown
Income_Category	Demographic variable - Annual Income Category of the account holder (< 40K, 40K - 60K, 60K− 80K, 80K− 120K, > $120K, Unknown)
Card_Category	Product Variable - Type of Card (Blue, Silver, Gold, Platinum)
Months_on_book	Period of relationship with bank
Total_Relationship_Count	Total no. of products held by the customer
Months_Inactive_12_mon	No. of months inactive in the last 12 months
Contacts_Count_12_mon	No. of Contacts in the last 12 months
Credit_Limit	Credit Limit on the Credit Card
Total_Revolving_Bal	Total Revolving Balance on the Credit Card
Avg_Open_To_Buy	Open to Buy Credit Line (Average of last 12 months)
Total_Amt_Chng_Q4_Q1	Change in Transaction Amount (Q4 over Q1)
Total_Trans_Amt	Total Transaction Amount (Last 12 months)
Total_Trans_Ct	Total Transaction Count (Last 12 months)
Total_Ct_Chng_Q4_Q1	Change in Transaction Count (Q4 over Q1)
Avg_Utilization_Ratio	Average Card Utilization Ratio
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1	Naïve bayes classsifier result
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2	Naïve bayes classsifier result

4. DATA PREPROCSSING

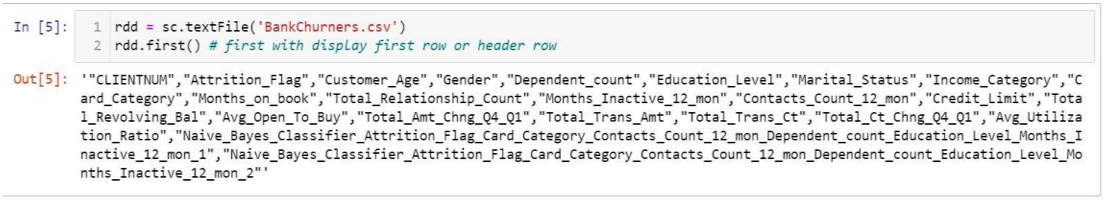
Data pre-processing is one of the vital step in machine learning project. Chances of getting noise, unwanted and errors in dataset is higher than a processed data. It is critical that we do data processing for the improved performance of our machine learning model there by we bring more meaning for our data. Or this we have several steps.

I. Loading the raw data into the environment.

For loading the csv data we have 2 different methods in PySpark.

Method 1: Load data using RDD

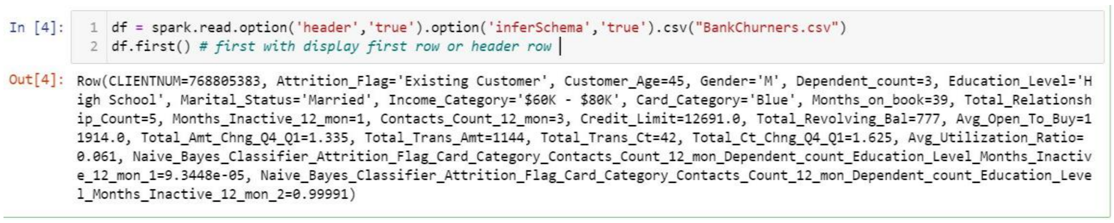
Using textFile() function of Spark Context we can load data through RDD. Below code snippet shows how to load data in python through RDD.



```
In [5]:  rdd = sc.textFile('BankChurners.csv')
         rdd.first() # first with display first row or header row

Out[5]: '"CLIENTNUM","Attrition_Flag","Customer_Age","Gender","Dependent_count","Education_Level","Marital_Status","Income_Category","C
        ard_Category","Months_on_book","Total_Relationship_Count","Months_Inactive_12_mon","Contacts_Count_12_mon","Credit_Limit","Tota
        l_Revolving_Bal","Avg_Open_To_Buy","Total_Amt_Chng_Q4_Q1","Total_Trans_Amt","Total_Trans_Ct","Total_Ct_Chng_Q4_Q1","Avg_Utiliza
        tion_Ratio","Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_I
        nactive_12_mon_1","Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Mo
        nths_Inactive_12_mon_2"'
```

Method 2: Load through dataframe

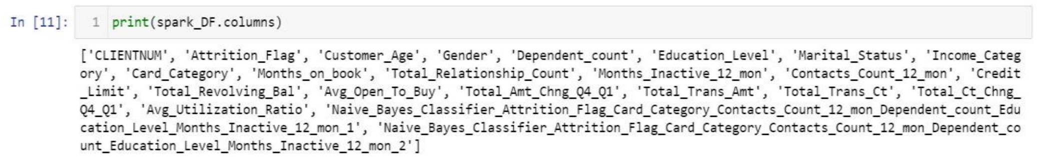
In PySpark we can load data in spark dataframe which is slight different from pandas dataframe. Below mentioned code snippent shows how to load data in spark dataframe.



```
In [4]:  df = spark.read.option('header','true').option('inferSchema','true').csv("BankChurners.csv")
         df.first() # first with display first row or header row

Out[4]: Row(CLIENTNUM=768805383, Attrition_Flag='Existing Customer', Customer_Age=45, Gender='M', Dependent_count=3, Education_Level='H
        igh School', Marital_Status='Married', Income_Category='$60K - $80K', Card_Category='Blue', Months_on_book=39, Total_Relationsh
        ip_Count=5, Months_Inactive_12_mon=1, Contacts_Count_12_mon=3, Credit_Limit=12691.0, Total_Revolving_Bal=777, Avg_Open_To_Buy=1
        1914.0, 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, Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactiv
        e_12_mon_1=9.3448e-05, Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Leve
        l_Months_Inactive_12_mon_2=0.99991)
```

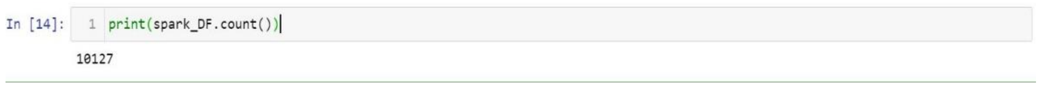
II. Fetch useful information from data.

1. Display all attributes of data



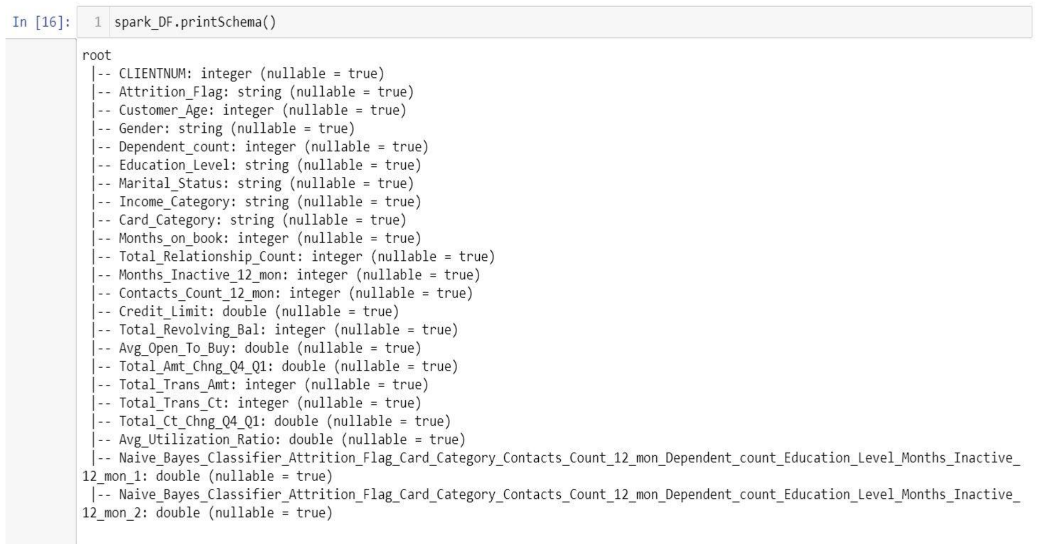
```
In [11]:  print(spark_DF.columns)

['CLIENTNUM', 'Attrition_Flag', 'Customer_Age', 'Gender', 'Dependent_count', 'Education_Level', 'Marital_Status', 'Income_Categ
ory', 'Card_Category', 'Months_on_book', 'Total_Relationship_Count', 'Months_Inactive_12_mon', 'Contacts_Count_12_mon', 'Credit
_Limit', 'Total_Revolving_Bal', 'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt', 'Total_Trans_Ct', 'Total_Ct_Chng_
Q4_Q1', 'Avg_Utilization_Ratio', 'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Edu
cation_Level_Months_Inactive_12_mon_1', 'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_co
unt_Education_Level_Months_Inactive_12_mon_2']
```

2. Fetch records count



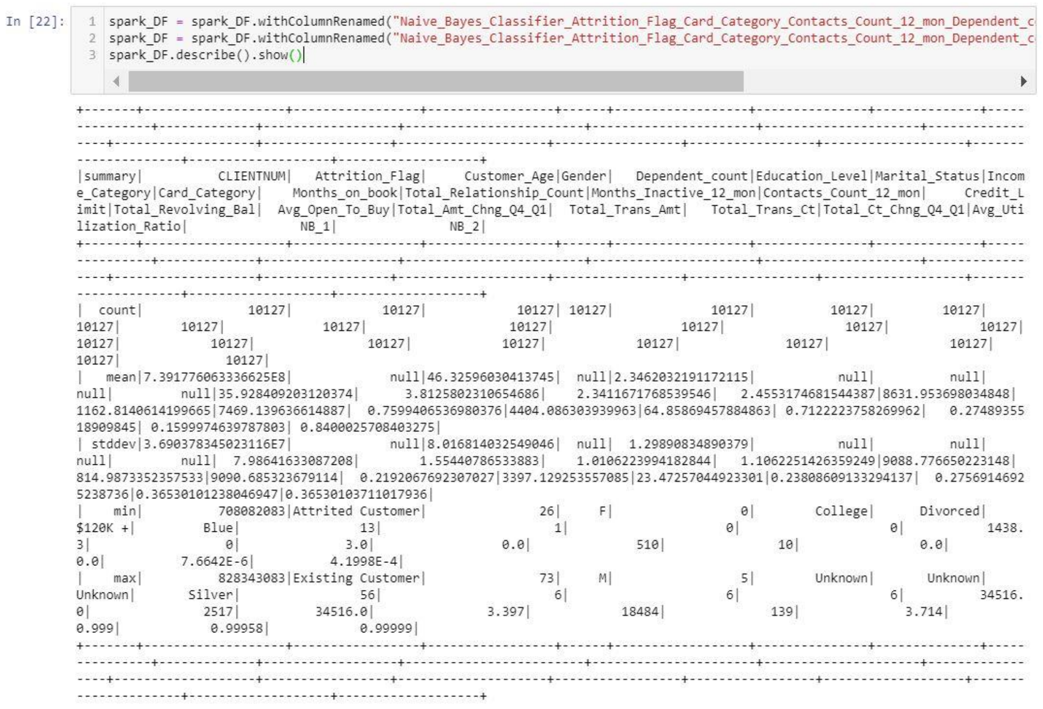
```
In [14]:  print(spark_DF.count())

10127
```

3. Display the structure of data using schema function



```
In [16]:  spark_DF.printSchema()

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)
```

4. Fetch statistical data such as mean, median, count, standard deviation, minimum and maximum values for each attribute from the data using describe function.

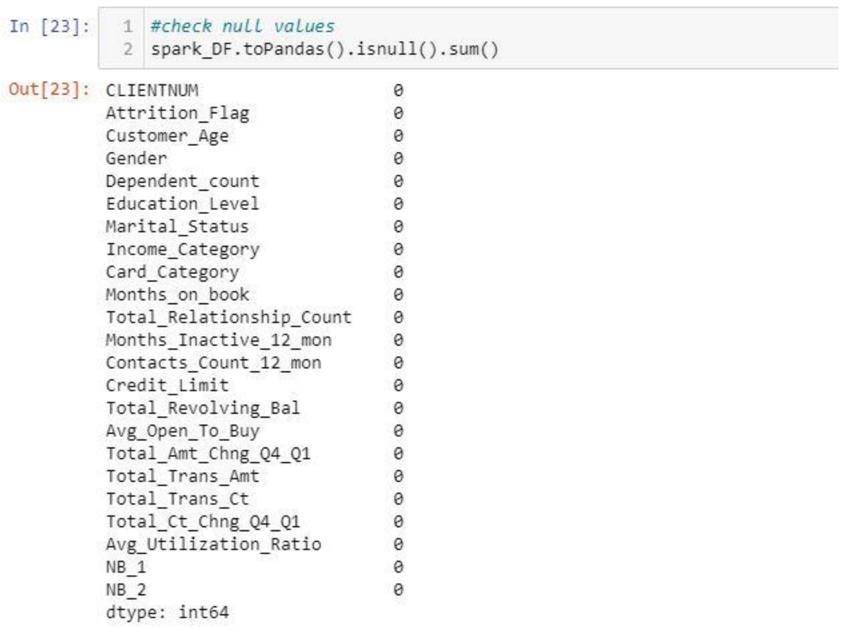


```
In [22]:
spark_DF = spark_DF.withColumnRenamed("Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_c
spark_DF = spark_DF.withColumnRenamed("Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_c
spark_DF.describe().show()
```

```
+-------+------------------+----------------+------------------+------+------------------+---------------+--------------+-----
----------+-------------+-----------------+------------------------+----------------------+---------------------+------------
----+------------------+-----------------+--------------------+------------------+------------------+-------------------+-------
-------------+------------------+------------------+
|summary|         CLIENTNUM|  Attrition_Flag|      Customer_Age|Gender|   Dependent_count|Education_Level|Marital_Status|Incom
e_Category|Card_Category|   Months_on_book|Total_Relationship_Count|Months_Inactive_12_mon|Contacts_Count_12_mon|      Credit_L
imit|Total_Revolving_Bal|  Avg_Open_To_Buy|Total_Amt_Chng_Q4_Q1|   Total_Trans_Amt|    Total_Trans_Ct|Total_Ct_Chng_Q4_Q1|Avg_Uti
lization_Ratio|              NB_1|              NB_2|
+-------+------------------+----------------+------------------+------+------------------+---------------+--------------+-----
----------+-------------+-----------------+------------------------+----------------------+---------------------+------------
----+------------------+-----------------+--------------------+------------------+------------------+-------------------+-------
-------------+------------------+------------------+
|  count|             10127|           10127|             10127| 10127|             10127|          10127|         10127|
10127|        10127|            10127|                   10127|                 10127|                10127|             10127|
10127|             10127|            10127|               10127|             10127|             10127|              10127|
10127|             10127|
|   mean|7.391776063336625E8|            null|46.32596030413745|  null|2.3462032191172115|           null|          null|
null|         null|35.928409203120374|      3.8125802310654686|    2.3411671768539546|   2.4553174681544387|8631.953698034848|
1162.8140614199665|7469.139636614887|  0.7599406536980376|4404.086303939963|64.85869457884863| 0.7122223758269962|   0.27489355
18909845| 0.1599974639787803| 0.8400025708403275|
| stddev|3.690378345023116E7|            null|8.016814032549046|  null|  1.29890834890379|           null|          null|
null|         null|  7.98641633087208|        1.55440786533883|    1.0106223994182844|   1.1062251426359249|9088.77665023148|
814.9873352357533|9090.685323679114|  0.2192067692307027|3397.129253557085|23.47257044923301|0.23808609133294137|  0.2756914692
5238736|0.36530101238046947|0.36530103711017936|
|    min|          708082083|Attrited Customer|               26|     F|                 0|        College|      Divorced|
$120K +|         Blue|               13|                       1|                     0|                    0|            1438.
3|                0|              3.0|                 0.0|               510|                10|                0.0|
0.0|         7.6642E-6|          4.1998E-4|
|    max|          828343083|Existing Customer|               73|     M|                 5|        Unknown|       Unknown|
Unknown|       Silver|               56|                       6|                     6|                    6|           34516.
0|             2517|          34516.0|               3.397|             18484|               139|              3.714|
0.999|           0.99958|           0.99999|
+-------+------------------+----------------+------------------+------+------------------+---------------+--------------+-----
----------+-------------+-----------------+------------------------+----------------------+---------------------+------------
----+------------------+-----------------+--------------------+------------------+------------------+-------------------+-------
-------------+------------------+------------------+
```

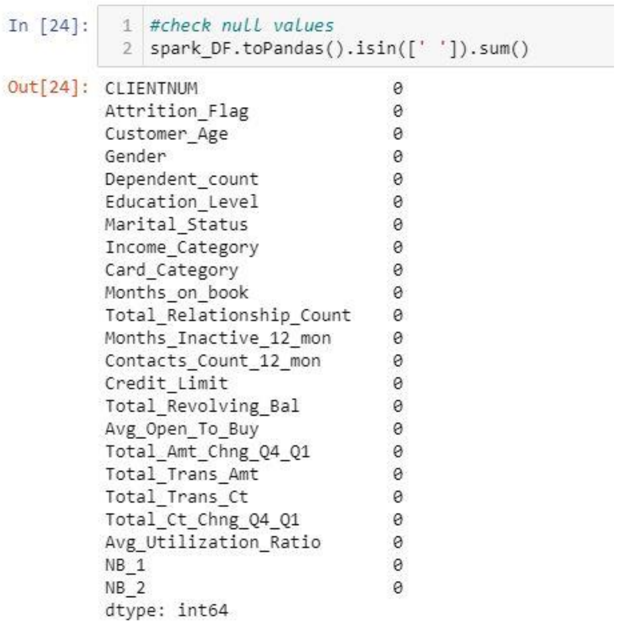
III. Handle missing and Null values in data

Since we are fetching these data rom real-world there can be missing and null values in the dataset. This will affect the performance of our model. To handle this we can use we can replace those values with mean or mean of that attribute, or by removing that entire row from the data.



```
In [23]:
#check null values
spark_DF.toPandas().isnull().sum()
```

```
Out[23]: 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
         NB_1                        0
         NB_2                        0
         dtype: int64
```

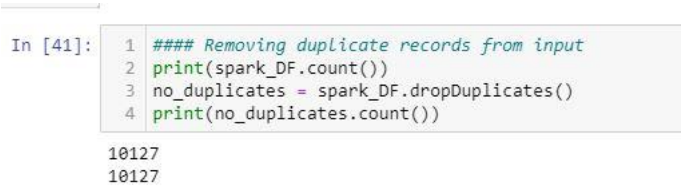


```
In [24]:  #check null values
          spark_DF.toPandas().isin([' ']).sum()

Out[24]:  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
          NB_1                        0
          NB_2                        0
          dtype: int64
```

From the code result we can see there is no missing or null values in our data

IV. Removing duplicate from the dataset

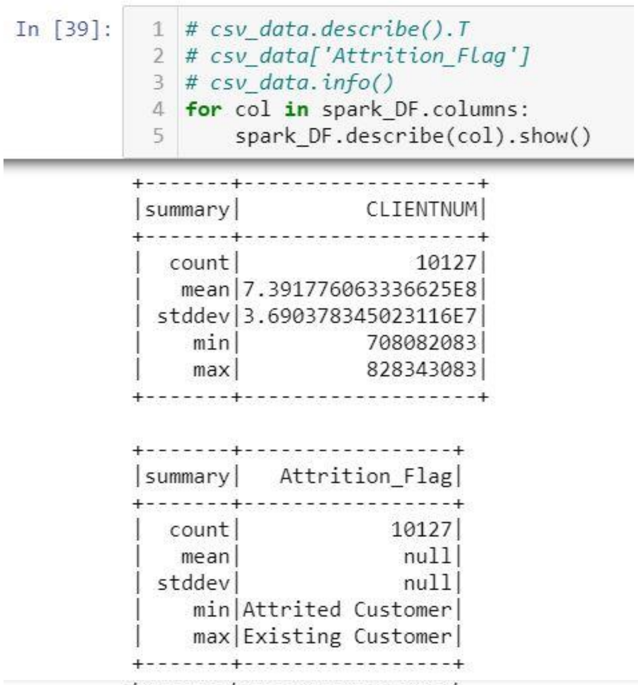
Using dropDuplicates() method we can remove all duplicate records from the data.



```
In [41]:  #### Removing duplicate records from input
          print(spark_DF.count())
          no_duplicates = spark_DF.dropDuplicates()
          print(no_duplicates.count())

          10127
          10127
```

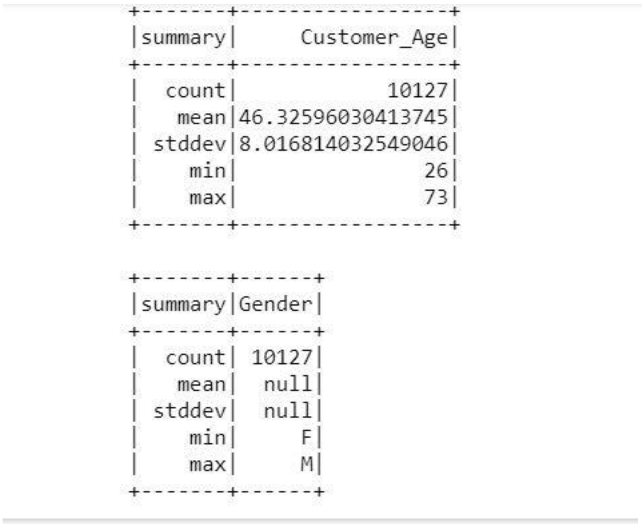
V. Summary of each attribute

Using describe method we are checking the details of each attribute. Which includes mean, median, count, standard deviation etc.

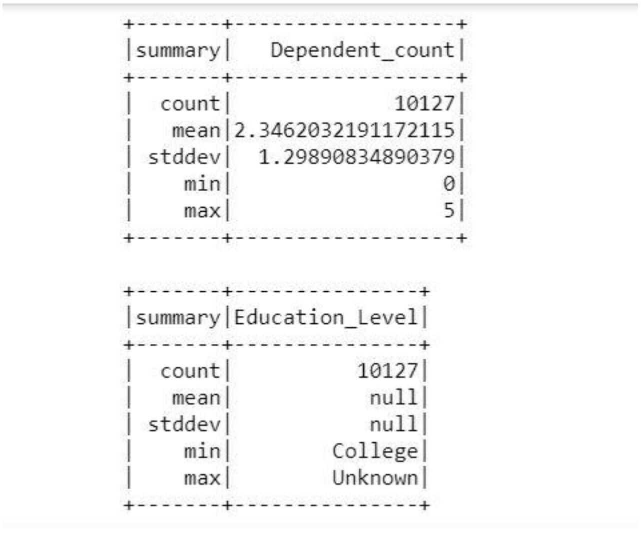


```
In [39]:
# csv_data.describe().T
# csv_data['Attrition_Flag']
# csv_data.info()
for col in spark_DF.columns:
    spark_DF.describe(col).show()
```

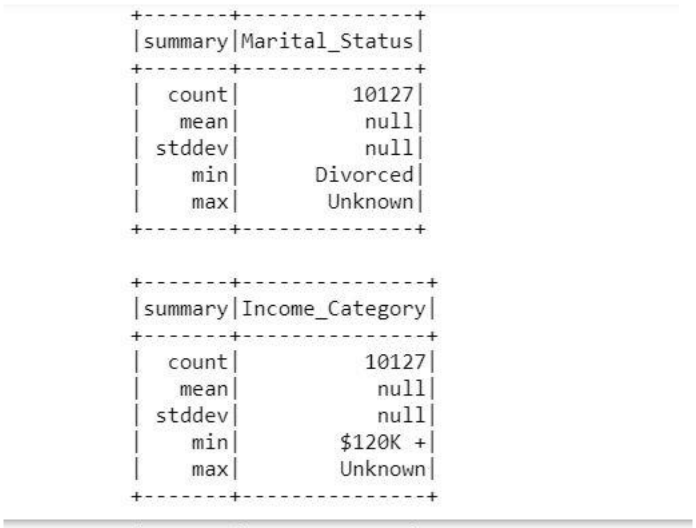
```
+-------+-------------------+
|summary|          CLIENTNUM|
+-------+-------------------+
|  count|              10127|
|   mean|7.391776063336625E8|
| stddev|3.690378345023116E7|
|    min|          708082083|
|    max|          828343083|
+-------+-------------------+

+-------+-----------------+
|summary|   Attrition_Flag|
+-------+-----------------+
|  count|            10127|
|   mean|             null|
| stddev|             null|
|    min|Attrited Customer|
|    max|Existing Customer|
+-------+-----------------+
```



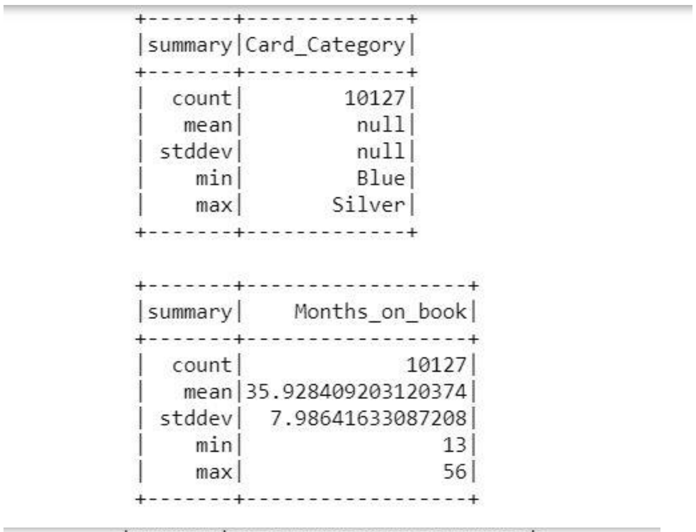
```
+-------+-----------------+
|summary|     Customer_Age|
+-------+-----------------+
|  count|            10127|
|   mean|46.32596030413745|
| stddev|8.016814032549046|
|    min|               26|
|    max|               73|
+-------+-----------------+

+-------+------+
|summary|Gender|
+-------+------+
|  count| 10127|
|   mean|  null|
| stddev|  null|
|    min|     F|
|    max|     M|
+-------+------+
```



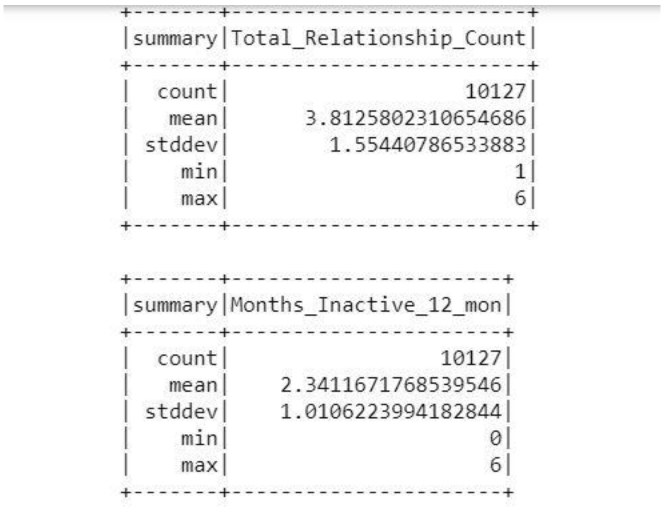
```
+-------+------------------+
|summary|   Dependent_count|
+-------+------------------+
|  count|             10127|
|   mean|2.3462032191172115|
| stddev|  1.29890834890379|
|    min|                 0|
|    max|                 5|
+-------+------------------+

+-------+---------------+
|summary|Education_Level|
+-------+---------------+
|  count|          10127|
|   mean|           null|
| stddev|           null|
|    min|        College|
|    max|        Unknown|
+-------+---------------+
```



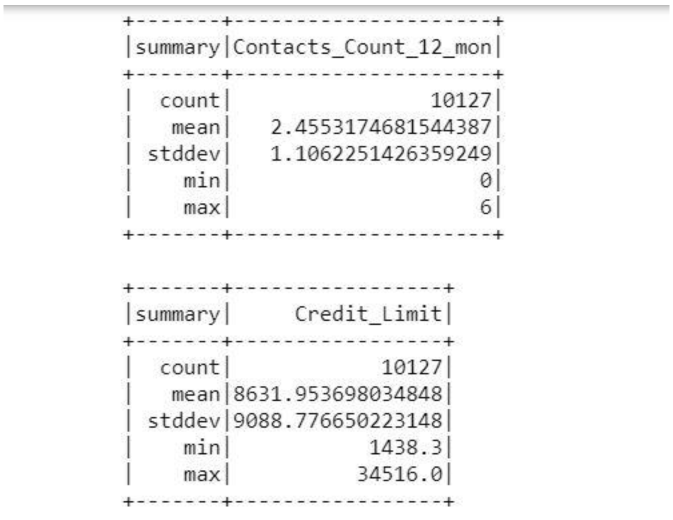
```
+-------+--------------+
|summary|Marital_Status|
+-------+--------------+
|  count|         10127|
|   mean|          null|
| stddev|          null|
|    min|      Divorced|
|    max|       Unknown|
+-------+--------------+

+-------+---------------+
|summary|Income_Category|
+-------+---------------+
|  count|          10127|
|   mean|           null|
| stddev|           null|
|    min|         $120K +|
|    max|        Unknown|
+-------+---------------+
```



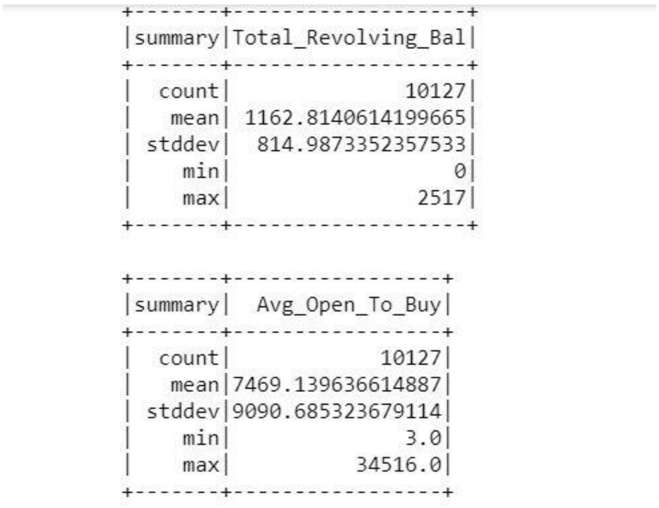
```
+-------+-------------+
|summary|Card_Category|
+-------+-------------+
|  count|        10127|
|   mean|         null|
| stddev|         null|
|    min|         Blue|
|    max|       Silver|
+-------+-------------+

+-------+-----------------+
|summary|   Months_on_book|
+-------+-----------------+
|  count|            10127|
|   mean|35.928409203120374|
| stddev| 7.98641633087208|
|    min|               13|
|    max|               56|
+-------+-----------------+
```



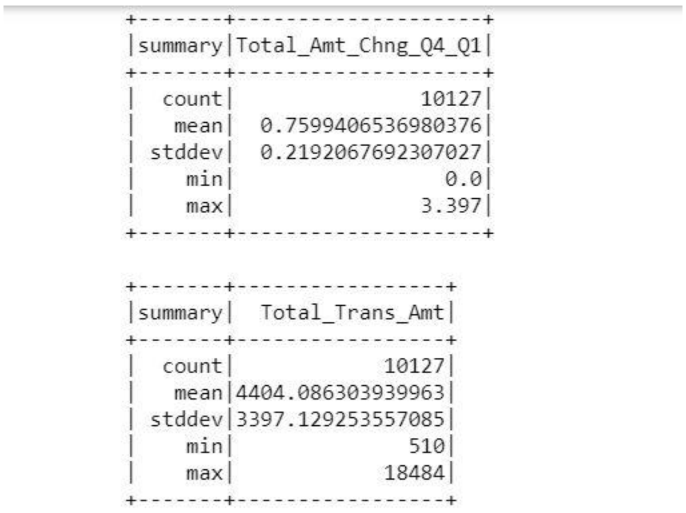
```
+-------+------------------------+
|summary|Total_Relationship_Count|
+-------+------------------------+
|  count|                   10127|
|   mean|      3.8125802310654686|
| stddev|        1.55440786533883|
|    min|                       1|
|    max|                       6|
+-------+------------------------+

+-------+----------------------+
|summary|Months_Inactive_12_mon|
+-------+----------------------+
|  count|                 10127|
|   mean|    2.3411671768539546|
| stddev|    1.0106223994182844|
|    min|                     0|
|    max|                     6|
+-------+----------------------+
```



```
+-------+--------------------+
|summary|Contacts_Count_12_mon|
+-------+--------------------+
|  count|               10127|
|   mean|  2.4553174681544387|
| stddev|  1.1062251426359249|
|    min|                   0|
|    max|                   6|
+-------+--------------------+

+-------+----------------+
|summary|    Credit_Limit|
+-------+----------------+
|  count|           10127|
|   mean|8631.953698034848|
| stddev|9088.776650223148|
|    min|          1438.3|
|    max|         34516.0|
+-------+----------------+
```



```
+-------+------------------+
|summary|Total_Revolving_Bal|
+-------+------------------+
|  count|             10127|
|   mean| 1162.8140614199665|
| stddev|  814.9873352357533|
|    min|                 0|
|    max|              2517|
+-------+------------------+

+-------+----------------+
|summary|  Avg_Open_To_Buy|
+-------+----------------+
|  count|           10127|
|   mean|7469.139636614887|
| stddev|9090.685323679114|
|    min|             3.0|
|    max|         34516.0|
+-------+----------------+
```



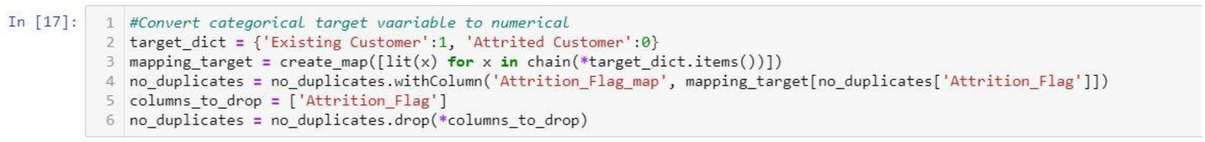
```
+-------+-------------------+
|summary|Total_Amt_Chng_Q4_Q1|
+-------+-------------------+
|  count|              10127|
|   mean|  0.7599406536980376|
| stddev|  0.2192067692307027|
|    min|                0.0|
|    max|              3.397|
+-------+-------------------+

+-------+----------------+
|summary|  Total_Trans_Amt|
+-------+----------------+
|  count|           10127|
|   mean|4404.086303939963|
| stddev|3397.129253557085|
|    min|             510|
|    max|           18484|
+-------+----------------+
```

5. EDA

Exploratory data analysis refers to the essential process of early data inquiry so that patterns can be identified, anomalies can be identified, hypothesis is assessed, and assumptions are examined using summary statistical data and visual data. Using graphical representation, we can easily identify the details of each attribute. For that I have used both tableau and python.

A. EDA using Python

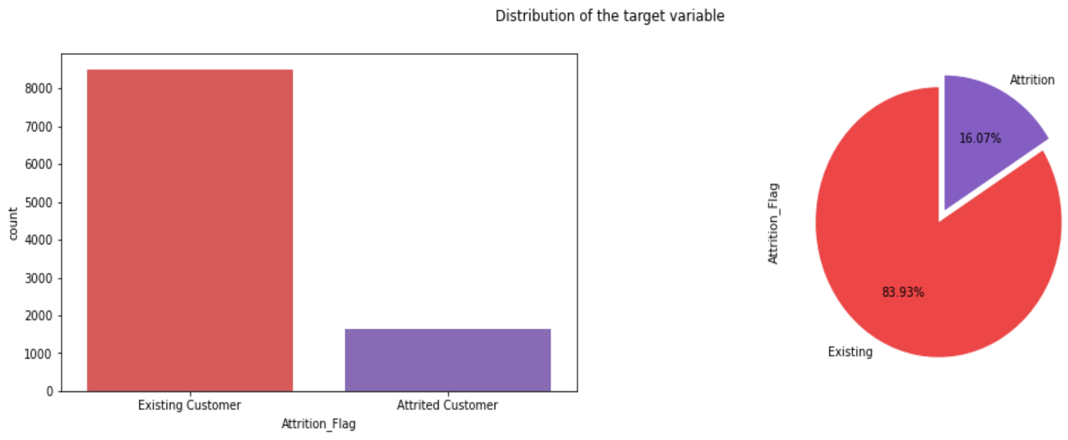
Here I have used pyplot and seaborn libraries to perform exploratory data analysis. Before starting with graphing, we have to convert the target attribute from categorical to numeric type we have used below method to convert it. To display correlation heat map, we need numerical data.



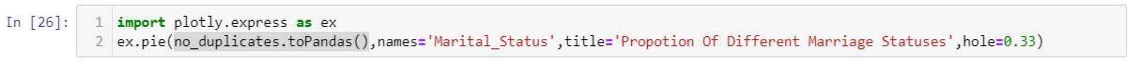
```
In [17]:
#Convert categorical target vaariable to numerical
target_dict = {'Existing Customer':1, 'Attrited Customer':0}
mapping_target = create_map([lit(x) for x in chain(*target_dict.items())])
no_duplicates = no_duplicates.withColumn('Attrition_Flag_map', mapping_target[no_duplicates['Attrition_Flag']])
columns_to_drop = ['Attrition_Flag']
no_duplicates = no_duplicates.drop(*columns_to_drop)
```

a. Churn ratio.

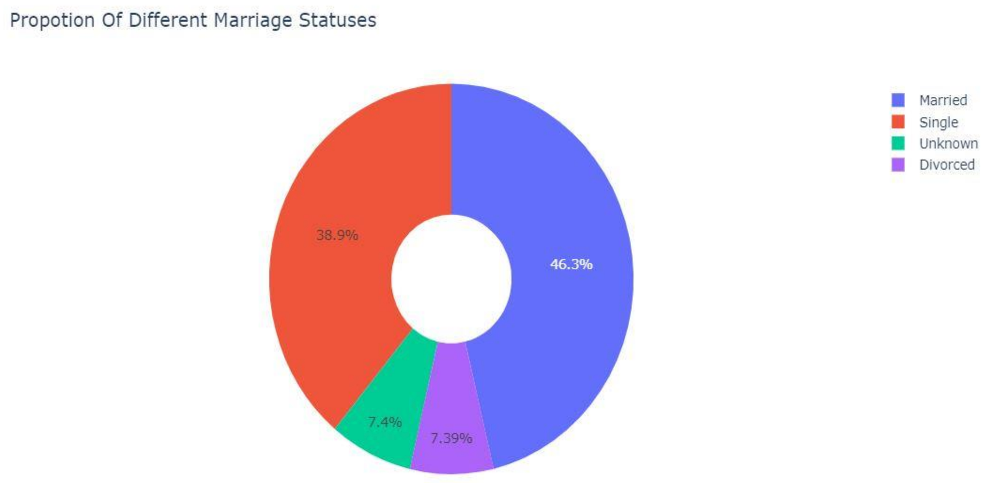
Displays ratio of target variable. From the graph we can see that only 16.07% churned customers are here in our dataset. So it might be difficult train our model



b. Marital status exploring



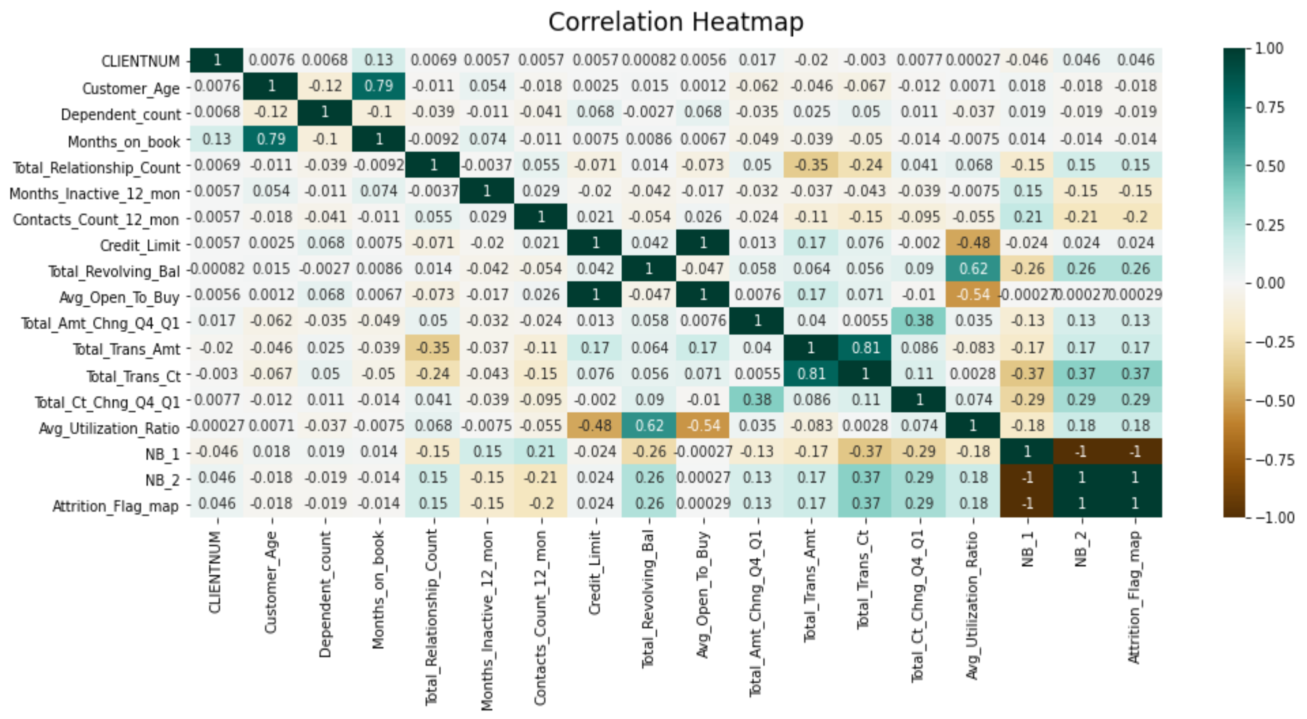
```
In [26]:
import plotly.express as ex
ex.pie(no_duplicates.toPandas(),names='Marital_Status',title='Propotion Of Different Marriage Statuses',hole=0.33)
```



Rom the figure we can infer that most of the customers are married.

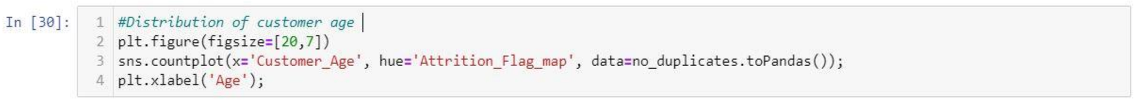
c. Multivariate Analysis

From the below correlation heat map, we can see that Naïve bayes

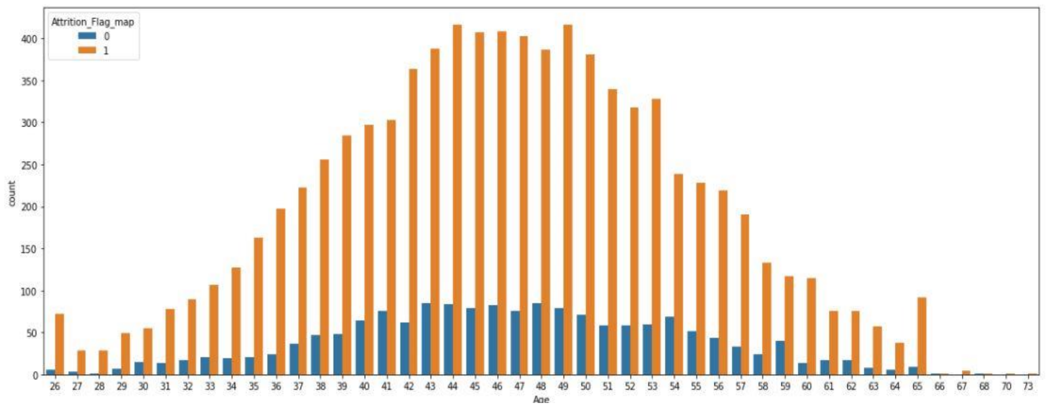


classier out puts show maximum +ve and -ve correlation this might affect the model outcome (over fitting).

d. Distribution of age attribute



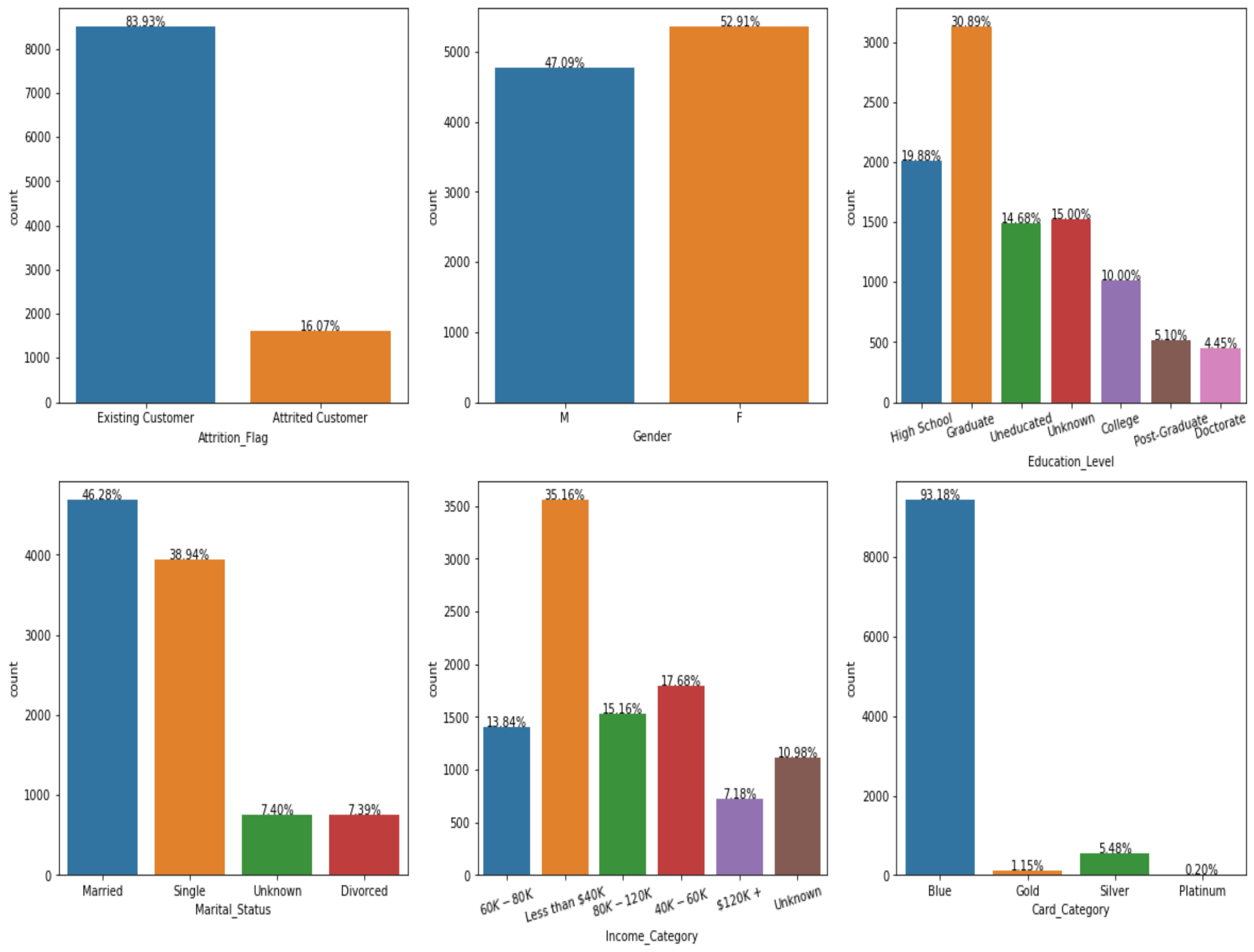
```
In [30]: #Distribution of customer age
         plt.figure(figsize=[20,7])
         sns.countplot(x='Customer_Age', hue='Attrition_Flag_map', data=no_duplicates.toPandas());
         plt.xlabel('Age');
```



From the graph we can infer that distribution of age shows normal distribution.

e. Exploring the categorical data

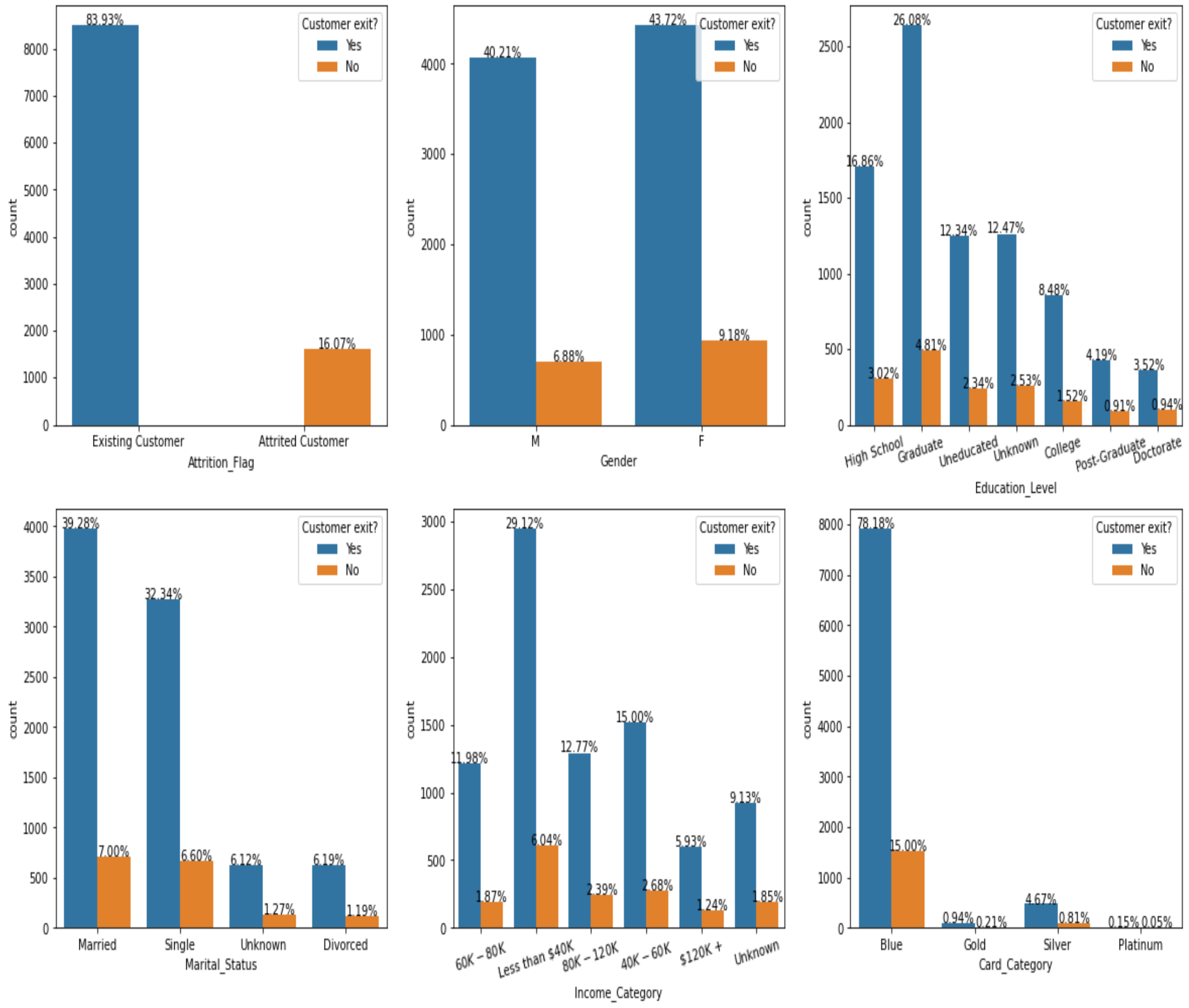
From the categorical data exploring we can see that those who has high



education don’t tend to you credit card. From the graph it is evident that most of the users us Blue card. In that case if we correlate the income category with card type, we can get more in sight.

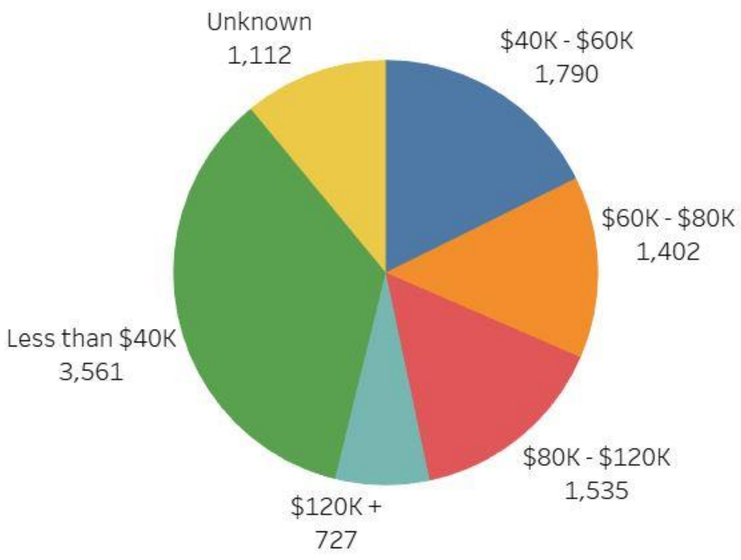
f. Exploring numerical features

From the figure we can females are the most customer for credit card.



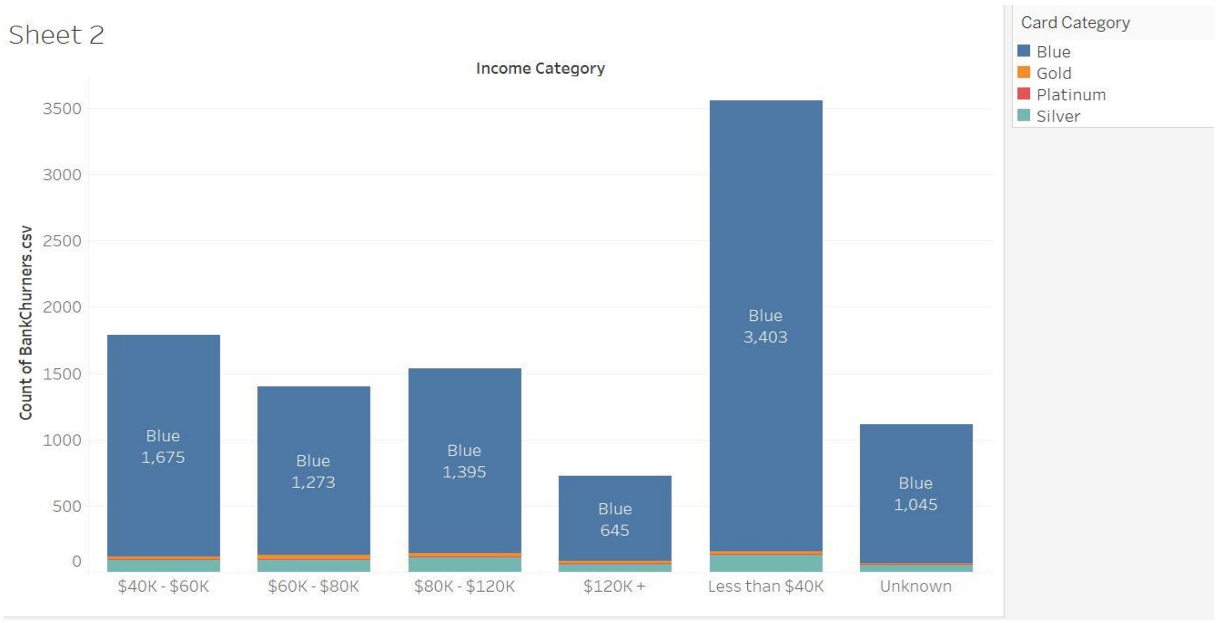
B. EDA using Tableau

a) Customer Income



Customers have those have less that $40k holds credit card. Those who have highest income holds least amount of credit card.

b) Correlation between income and card



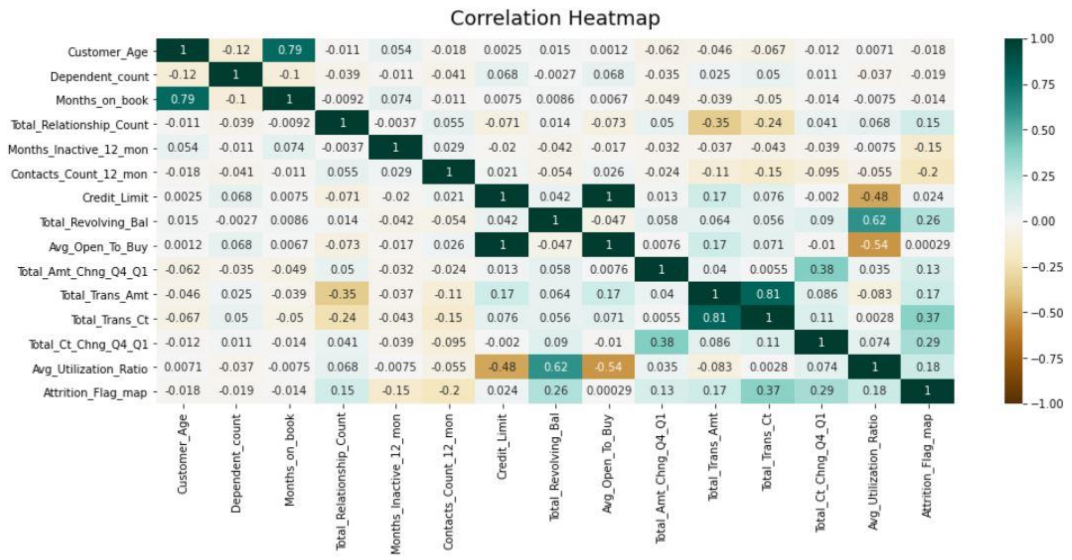
6. FEATURE SELECTION

From the EDA we found that Naïve Bayes results may affect the outcome of our model by over fitting. To avoid that scenario, we are removing that attribute along with CLIENTNUM. CLIENTNUM is just a unique number an no relevant for our model.

```
In [21]:
#Dropping Unwanted attributes
columns_to_drop = ['NB_1',
                   'NB_2',
                   'CLIENTNUM']
no_duplicates = no_duplicates.drop(*columns_to_drop)
```

Now we can check the correlation heat graph

```
In [22]:
plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(no_duplicates.toPandas().corr(), vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':18}, pad=12);
# save heatmap as .png file
# dpi - sets the resolution of the saved image in dots/inches
# bbox_inches - when set to 'tight' - does not allow the labels to be cropped
plt.savefig('heatmap.png', dpi=300, bbox_inches='tight')
```



	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	Attrition_Flag_map
Customer_Age	1	-0.12	0.79	-0.011	0.054	-0.018	0.0025	0.015	0.0012	-0.062	-0.046	-0.067	-0.012	0.0071	-0.018
Dependent_count	-0.12	1	-0.1	-0.039	-0.011	-0.041	0.068	-0.0027	0.068	-0.035	0.025	0.05	0.011	-0.037	-0.019
Months_on_book	0.79	-0.1	1	-0.0092	0.074	-0.011	0.0075	0.0086	0.0067	-0.049	-0.039	-0.05	-0.014	-0.0075	-0.014
Total_Relationship_Count	-0.011	-0.039	-0.0092	1	-0.0037	0.055	-0.071	0.014	-0.073	0.05	-0.35	-0.24	0.041	0.068	0.15
Months_Inactive_12_mon	0.054	-0.011	0.074	-0.0037	1	0.029	-0.02	-0.042	-0.017	-0.032	-0.037	-0.043	-0.039	-0.0075	-0.15
Contacts_Count_12_mon	-0.018	-0.041	-0.011	0.055	0.029	1	0.021	-0.054	0.026	-0.024	-0.11	-0.15	-0.095	-0.055	-0.2
Credit_Limit	0.0025	0.068	0.0075	-0.071	-0.02	0.021	1	0.042	1	0.013	0.17	0.076	-0.002	-0.48	0.024
Total_Revolving_Bal	0.015	-0.0027	0.0086	0.014	-0.042	-0.054	0.042	1	-0.047	0.058	0.064	0.056	0.09	0.62	0.26
Avg_Open_To_Buy	0.0012	0.068	0.0067	-0.073	-0.017	0.026	1	-0.047	1	0.0076	0.17	0.071	-0.01	-0.54	0.00029
Total_Amt_Chng_Q4_Q1	-0.062	-0.035	-0.049	0.05	-0.032	-0.024	0.013	0.058	0.0076	1	0.04	0.0055	0.38	0.035	0.13
Total_Trans_Amt	-0.046	0.025	-0.039	-0.35	-0.037	-0.11	0.17	0.064	0.17	0.04	1	0.81	0.086	-0.083	0.17
Total_Trans_Ct	-0.067	0.05	-0.05	-0.24	-0.043	-0.15	0.076	0.056	0.071	0.0055	0.81	1	0.11	0.0028	0.37
Total_Ct_Chng_Q4_Q1	-0.012	0.011	-0.014	0.041	-0.039	-0.095	-0.002	0.09	-0.01	0.38	0.086	0.11	1	0.074	0.29
Avg_Utilization_Ratio	0.0071	-0.037	-0.0075	0.068	-0.0075	-0.055	-0.48	0.62	-0.54	0.035	-0.083	0.0028	0.074	1	0.18
Attrition_Flag_map	-0.018	-0.019	-0.014	0.15	-0.15	-0.2	0.024	0.26	0.00029	0.13	0.17	0.37	0.29	0.18	1

From the correlation graph we can see that three attribute shows positive correlation.

- Total_Trans_Ct,
- Total_Ct_Chng_4_Q1
- Total_Revolving_Bal

These attributes influence the accuracy of our model more

7. MACHINE LEARNING CREDIT CUSTOMER CHURN

Applied Artificial Intelligence, machine learning allows the system to learn and enhance its experience automatically without intervention with other means or explicit programming. It focuses on computer development which accesses and utilises data for education

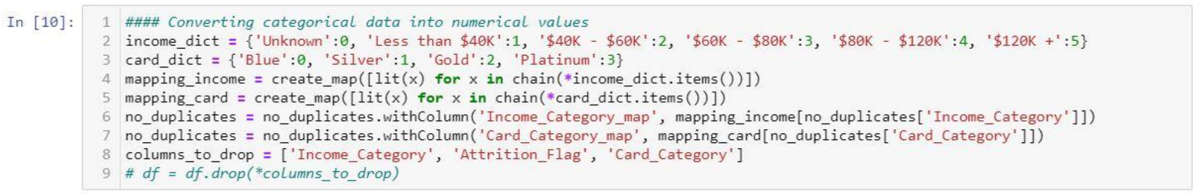
Brief: Credit card customer churn one of the major issue faced by the banks currently. Credit card interest and loans are one of the source of income for banks. So keeping the credit card customer is more important. Currently banks are trying to find a solution for anticipating the credit card customer churn. In order to take the required measures to prevent churning, it is therefore vital to determine the variables that contribute to customer sales. I've developed an ML model that can predict customer turnover by use of the 2 prominent machine learning classification method Logistic regression and Random Forest classification. In relation to other numerical and categorical columns, the models will predict 2 binary levels 1 and 0 for the Churn column in our data set.

1) Data preparation

We have performed necessary data pre-processing for our data. But for implementing our machine learning model we need to convert the categorical data into numerical data. Apart from our target attribute we need to convert 8 categorical data. Here I have used two methods to convert

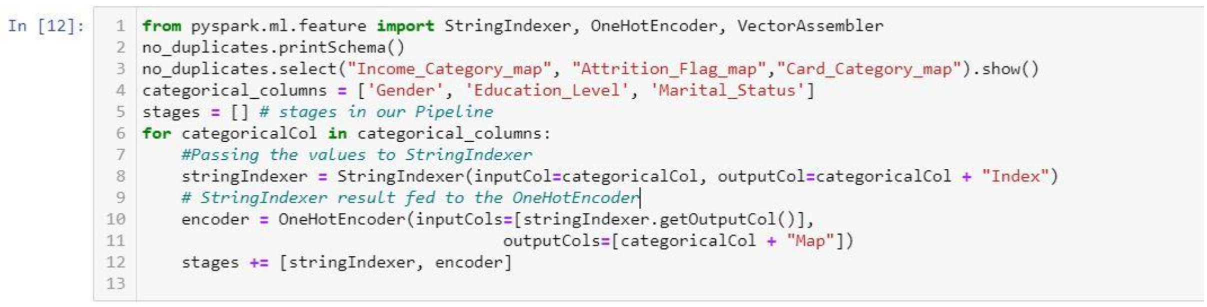
- Encoding using dictionary values
- One Hot Encoding

For 2 attributes the order was important, so we have encoded the values using dictionary. Income category is important while training and predicting the model. From the code snippet we can see that I have set the values according to its importance



```
In [10]:
#### Converting categorical data into numerical values
income_dict = {'Unknown':0, 'Less than $40K':1, '$40K - $60K':2, '$60K - $80K':3, '$80K - $120K':4, '$120K +':5}
card_dict = {'Blue':0, 'Silver':1, 'Gold':2, 'Platinum':3}
mapping_income = create_map([lit(x) for x in chain(*income_dict.items())])
mapping_card = create_map([lit(x) for x in chain(*card_dict.items())])
no_duplicates = no_duplicates.withColumn('Income_Category_map', mapping_income[no_duplicates['Income_Category']])
no_duplicates = no_duplicates.withColumn('Card_Category_map', mapping_card[no_duplicates['Card_Category']])
columns_to_drop = ['Income_Category', 'Attrition_Flag', 'Card_Category']
# df = df.drop(*columns_to_drop)
```

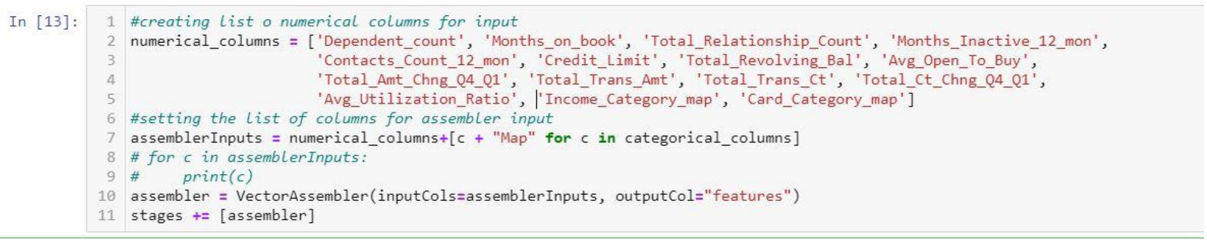
For rest of the categorical data I have done one hot encoding using OneHotEncoder from PySpark. For this first we fetch categorical data to StringIndexer and then to the OneHotEncoder. From the below code snippet we can understand the flow.



```
In [12]:
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
no_duplicates.printSchema()
no_duplicates.select("Income_Category_map", "Attrition_Flag_map","Card_Category_map").show()
categorical_columns = ['Gender', 'Education_Level', 'Marital_Status']
stages = [] # stages in our Pipeline
for categoricalCol in categorical_columns:
    #Passing the values to StringIndexer
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + "Index")
    # StringIndexer result fed to the OneHotEncoder
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()],
                                    outputCols=[categoricalCol + "Map"])
    stages += [stringIndexer, encoder]
```

From the snippet we can see that every conversion is being store as Indexer and encoder. These stages are kept for feed the pipeline, to execution. Since we haven't used any indexer or encoder for converting our other categorical data including the target variable, we are not storing those data in stage.

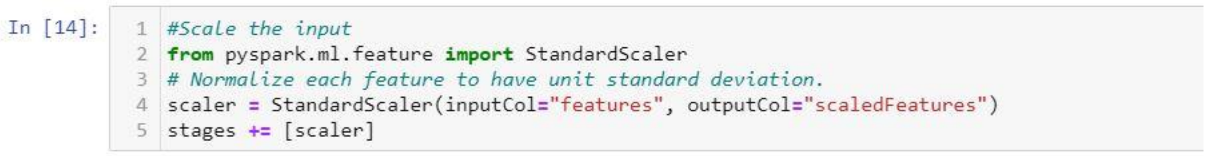
Apart from other ml model, for our PySpark ML model we need to give the input as vectorized input columns. Before creating the vectorized assembler output we need to add continuous variable. This includes adding our numerical columns as well as already converted categorical data. Below code snippet shows how I have done it.



```
In [13]:
#creating list o numerical columns for input
numerical_columns = ['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', 'Income_Category_map', 'Card_Category_map']
#setting the list of columns for assembler input
assemblerInputs = numerical_columns+[c + "Map" for c in categorical_columns]
# for c in assemblerInputs:
#     print(c)
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

Once vectorassembler is set we add it's to the output to the stages list.

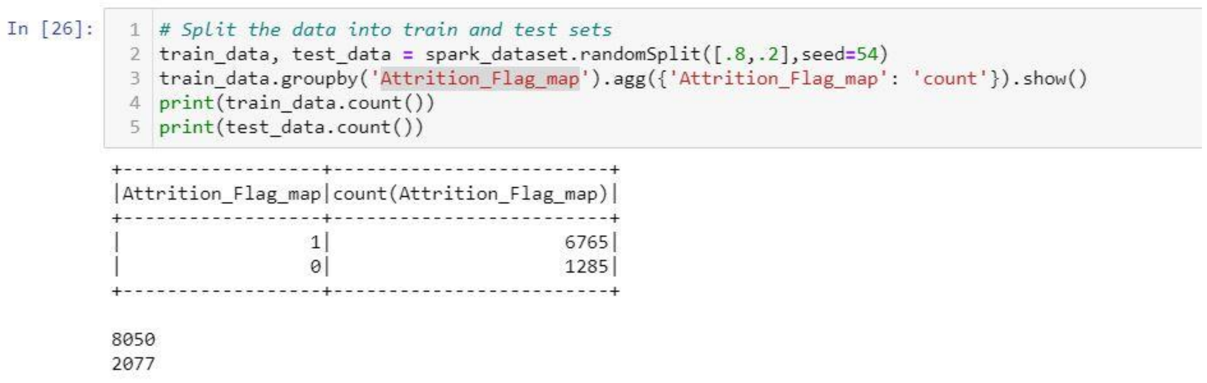
Standardization: there can be outliers in our data, so standardization can bring more meaning to our data. Here I have used StandardScaler from PySpark ml library. I have used standardscaler because it normalizes the data with zero mean and unit standard deviation. After performing the standardization, we will have one more attribute in our dataframe called scaledfeatures.



```
In [14]:
#Scale the input
from pyspark.ml.feature import StandardScaler
# Normalize each feature to have unit standard deviation.
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
stages += [scaler]
```

Once the stages are added we can build the pipeline. For that we can create the pipeline object using the stages. Then we create model by fitting the data and transform data using pipeline model. We get the new data which we have mentioned earlier while creating the pipeline.

Once the data has created after building pipeline, we split the data into two set one for training the machine leaning model and other or testing it. We split 80% of the data for training the model and 20% for testing.



```
In [26]:  1 # Split the data into train and test sets
          2 train_data, test_data = spark_dataset.randomSplit([.8,.2],seed=54)
          3 train_data.groupby('Attrition_Flag_map').agg({'Attrition_Flag_map': 'count'}).show()
          4 print(train_data.count())
          5 print(test_data.count())

+------------------+-------------------------+
|Attrition_Flag_map|count(Attrition_Flag_map)|
+------------------+-------------------------+
|                 1|                     6765|
|                 0|                     1285|
+------------------+-------------------------+

8050
2077
```

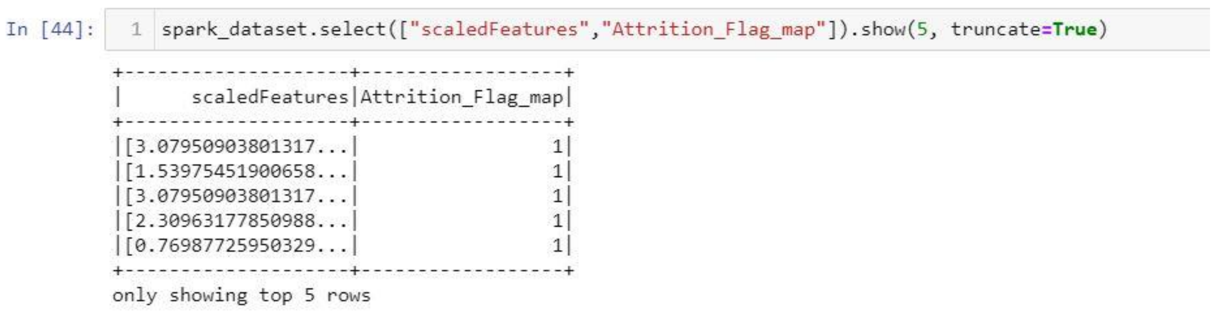
From the above code snippet, we seen data split into two part (in line 2).

2) Classification Model

As I have mentioned the starting of this section, we are using two different machine learning algorithms from PySpark library.

- Logistical Regression
- Random Forest classification

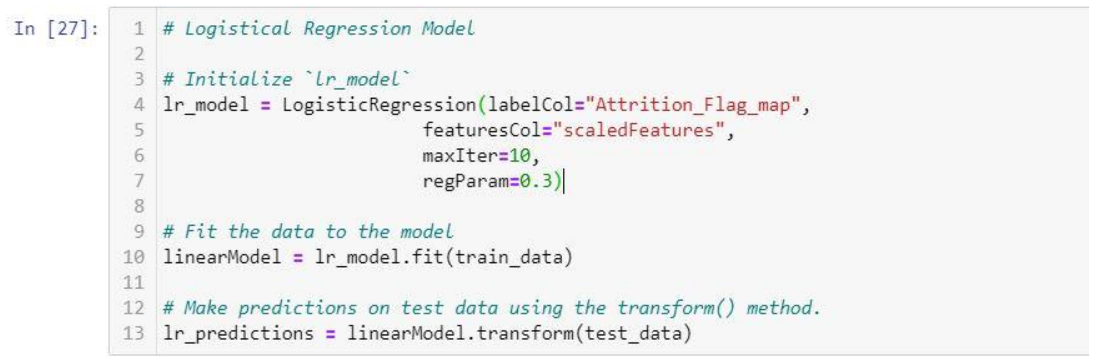
Since this churn is a classification problem, we are considering accuracy and area under the ROC curve as the measure of for selecting the best model. Both these models are prominent ML model. From the below code snippet we can see the vectorized input attribute and the target attribute.



```
In [44]:  spark_dataset.select(["scaledFeatures","Attrition_Flag_map"]).show(5, truncate=True)

+--------------------+------------------+
|      scaledFeatures|Attrition_Flag_map|
+--------------------+------------------+
|[3.07950903801317...|                 1|
|[1.53975451900658...|                 1|
|[3.07950903801317...|                 1|
|[2.30963177850988...|                 1|
|[0.76987725950329...|                 1|
+--------------------+------------------+
only showing top 5 rows
```

i. Logistical Regression

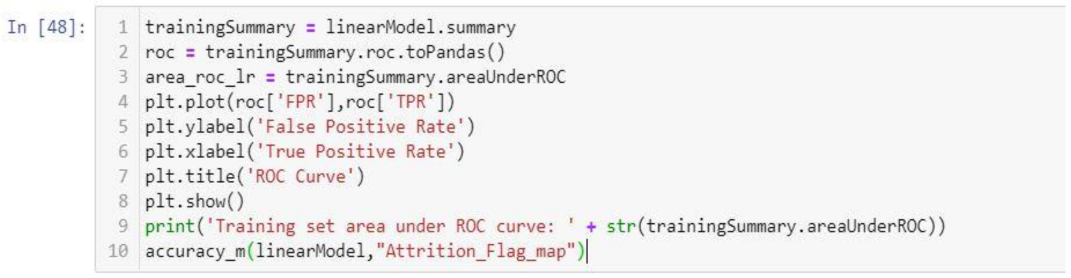
Here I have taken the class from PySpark ML library. First, we create model by specifying the target attribute and vectorized input attribute names.



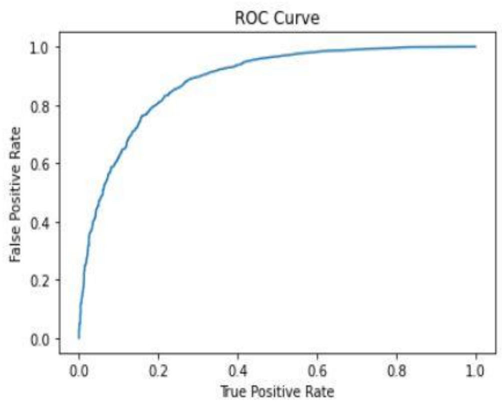
```
In [27]:
# Logistical Regression Model

# Initialize `lr_model`
lr_model = LogisticRegression(labelCol="Attrition_Flag_map",
                              featuresCol="scaledFeatures",
                              maxIter=10,
                              regParam=0.3)

# Fit the data to the model
linearModel = lr_model.fit(train_data)

# Make predictions on test data using the transform() method.
lr_predictions = linearModel.transform(test_data)
```

One the model is created we must train the model with our train dataset using the fit() method. Then using the resultant model we have to predict the outcome of our test dataset using transform() method. From the above code snippet we can see the these process.

Once we trained and predict the model we will calculate the accuracy and area under the ROC curve since we are focusing on business side. Accuracy is have higher importance. For our model we have 83.67% of accuracy and 0.88398 od area under the curve



```
In [48]:
trainingSummary = linearModel.summary
roc = trainingSummary.roc.toPandas()
area_roc_lr = trainingSummary.areaUnderROC
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
print('Training set area under ROC curve: ' + str(trainingSummary.areaUnderROC))
accuracy_m(linearModel,"Attrition_Flag_map")
```

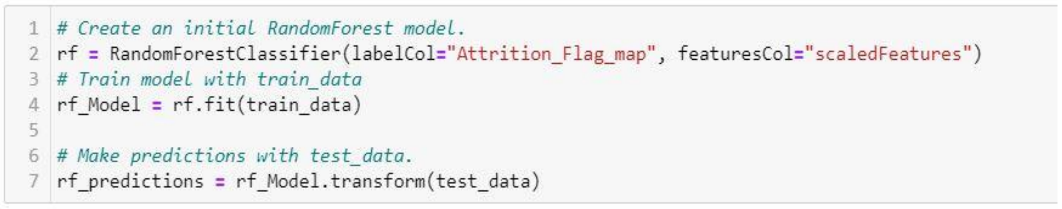


```
Training set area under ROC curve: 0.8839825032137834
Model accuracy: 83.678%
```

From the above code snippet, we can see the result of our LR model.

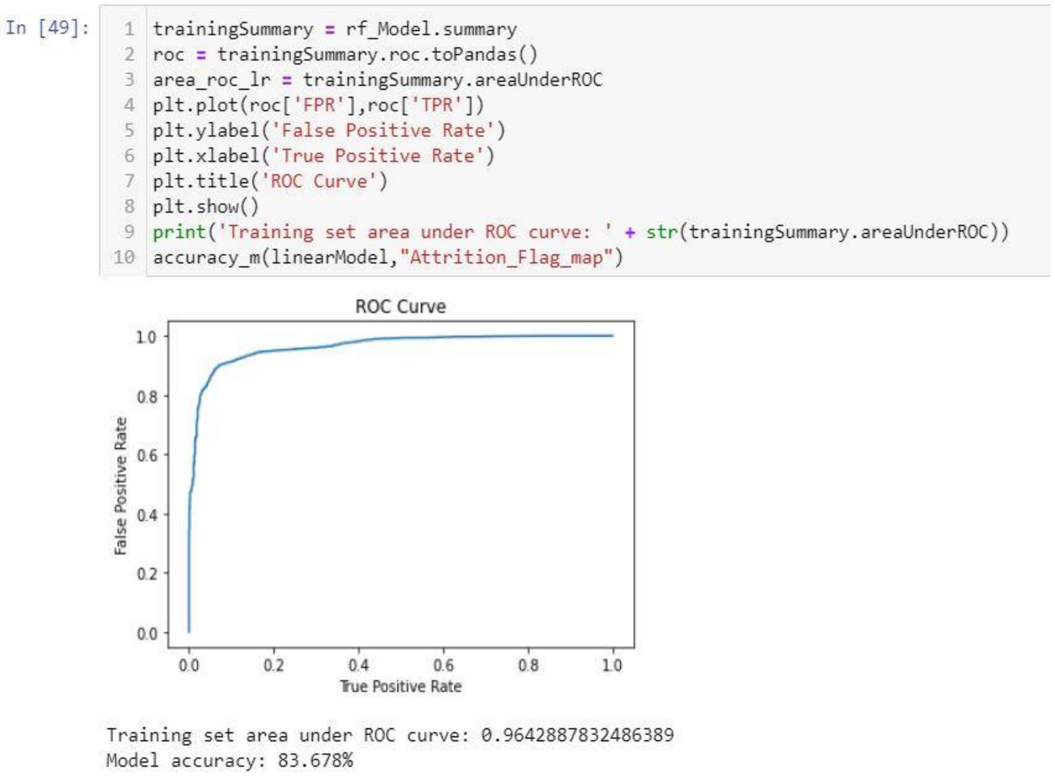
ii. Random Forest Classification

Similar to the implementation of Logistical regression model I have used pyspark ML library model for Random Forest. Similarly, we will train and test the model using the Random Forest model.



```
# Create an initial RandomForest model.
rf = RandomForestClassifier(labelCol="Attrition_Flag_map", featuresCol="scaledFeatures")
# Train model with train_data
rf_Model = rf.fit(train_data)

# Make predictions with test_data.
rf_predictions = rf_Model.transform(test_data)
```

Then calculated the accuracy and the area under the curve. We got 83.67% accuracy and area under the RoC curve .9642.



```
In [49]:
trainingSummary = rf_Model.summary
roc = trainingSummary.roc.toPandas()
area_roc_lr = trainingSummary.areaUnderROC
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
print('Training set area under ROC curve: ' + str(trainingSummary.areaUnderROC))
accuracy_m(linearModel,"Attrition_Flag_map")
```

```
Training set area under ROC curve: 0.9642887832486389
Model accuracy: 83.678%
```

8. DISCUSSION

In this experiment, we have tried to predict the credit card customer churn using two machine learning models. In this section, we are going to discuss the visualization and learning outcome of this experiment. Initially from our dataset we had only 16.07% of records were churned customers and 83.93% of them were existing customers. From the figure, we can see that as we discussed earlier using such a small set of scenarios developing a classification model was difficult. By proper data pre-processing we were able to get an accuracy of 83.678% for both models. This is not sufficient but it’s a really good percentage.

From EDA we were able to find more information about the data we were dealing with from the correlation heat map we were able to find out attributes that are necessary for our experiment and attributes that are not required. Thereby we have eliminated a few attributes from the dataset which has helped to improve our model. From EDA we were able to understand that most of the credit card customers are females and also churned customers. From the analysis, we can infer that those have done more transaction using credit card preferred to stay. By observing the correlation plot that we can see who has more education not interested to use a credit card. The largest set of customers uses Blue credit cards.

After all these scenarios we have found that the majority of the churned customers were using credit cards often. Thereby we can say that if we were able to predict credit card customer churn, we can prevent it. From the model's result, we can see both of them were performed efficiently. But we cannot say one is better than the other only with accuracy. That is why we have considered the area under the ROC curve. And also, with an oversampling technique, we can improve the accuracy of each model. In this scenario accuracy of both, the models are the same so we can consider the model that has more area or ROC can be a better model compared to other.

9. CONCLUSION

I have successfully installed the PySpark and initialized its essential components. First, we have done the necessary data pre-processing for our dataset. Then after EDA we took appropriate operation for our dataset and brought more meaning to it. Using the PysSpark machine learning libraries for Logistical Regression and Random Forest we have created the classifier models to predict the credit card customer churn.

We have evaluated the performance and accuracy of each model. Based on the evaluation result we can see that both have the same accuracy. Then by considering the area under the ROC curve, we can see that model with the Random Forest algorithm has a significant difference compared to the Logistical Regression model. From this experiment, we can say that the Random Forest model is the best model for our experiment.

Extensive reading and research are the aims of this course study. New technologies such as Pyspark were fascinating to learn. Learning the machine Pyspark modelling, and prediction is easy and comprehensible. This study helps to better analyse and comprehend data utilizing various pictures and tables for display. Working with Tableau for the display component was extremely fascinating.

The customers contemplate stopping using their credit cards. There are several causes. If the banks must keep its client, this data must be analysed and investigated

10. APPENDIX

Source Code: OneDrive

Dataset: Kaggle

11. REFERENCE

[1]. Data flair: https://data-flair.training/blogs/pyspark-sparkcontext/

[2]. Spark by Examples: https://sparkbyexamples.com/pyspark-rdd/

[3]. DataBricks: https://databricks.com/glossary/pyspark