Bank Customer Segmentation using Apache Spark's PySpark

(CISC-886)

Under the Supervision of:
Dr. Anwar Hussain

Group Number: 11

Team Member:

AbdelHafez, Yasmine	20398568
El-Ghobashy, Manar	20398569
El-Zahy, Yara	20398570
Sabry, Yasmine	20398566

1. Introduction

PySpark is Python API for Apache Spark, a distributed general-purpose computing platform that is free source written in Scala. PySpark has capabilities such as SQL querying, dealing with DataFrames, streaming and abstractions that aid in working with Big Data, Machine Learning, and a library called MLlib. Machine Learning in PySpark is simple to use and scalable. It is applicable to distributed systems. Spark Machine Learning may be used to analyze data. Machine Learning methods may be applied in a variety of ways, including regression, classification, and clustering, which will be explored in this project.

In this report, we will discuss bank customer segmentation to generate valuable client clusters that can be used to target different promotions and offers, as well as to find new markets and gain new customers.

2. Data Collection

The dataset we have worked on is of title "*Prediction of Churning Credit Card Customers.*" [1]

The dataset contains 10128 rows and 23 column .It offers an extensive collection of customer information gathered from a consumer credit card portfolio in order to assist analysts in forecasting client attrition. It comprises customer demographics, such as gender, marital status, education level and income category to predict which customer segment is more likely to churn. as well as information about each customer's connection with the credit card provider such as card type, number of months on book, and inactive periods. Furthermore, it contains critical information about customers' spending habits that are relevant to their churn decision, such as total revolving balance, credit limit, average open to buy rate, and analyzable metrics such as total amount of change from quarter 4 to quarter 1, average utilization ratio, and Naive Bayes classifier attrition flag. The card category is paired with the number of contacts in a 12-month period, as well as the number of dependents, education level, and months inactive.

- *CLIENTNUM*: unique identifier for each customer (Integer)
- *Attrition_Flag*: flag indicating whether or not the customer has churned out (Boolean)
- *Customer_Age*: age of customer (Integer)
- *Gender*: gender of customer (String)
- *Dependent_Count*: number of dependents that customer has (Integer)
- *Education_Level*: education level of customer (String)
- *Marital_Status*: marital status of customer (String)
- *Income_Category*: income category of customer (String)
- *Card_Category*: type of card held by customer (String)
- *Months_on_Book*: how long customer has been on the books (Integer)
- *Total_Relationship_Count*: total number of relationships customer has with the credit card provider (Integer)

- *Monthers_Inactive_12_mon*: number of months customer has been inactive in the last twelve months (Integer)
- *Contacts_Count_12_mon*: number of contacts customer has had in the last twelve months (Integer)
- *Credit_Limit*: credit limit of customer (Integer)
- *Total_Revolving_Bal*: total revolving balance of customer (Integer)
- *Avg_Open_To_Buy*: average open buy ration of customer (Integer)
- *Total_Amt_Chng_Q4_Q1*: total amount changed from quarter 4 to quarter 1 (Integer)
- *Total_Trans_Amt*: total transaction amount (Integer)
- *Total_Trans_Ct*: total transaction count (Integer)
- *Total_Ct_Chng_Q4_Q1*: total count changed from quarter 4 to quarter 1 (Integer)
- *Avg_Utilization_Ratio*: average utilization ration of customer (Integer)
- *Naïve_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1*: Naive Bayes classifier for predicting whether or not someone will churn based on characteristics.

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	.
0	768805383	Existing Customer	45	M	3	High School	Married	60K−80K	Blue	39	.
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	44	.
2	713982108	Existing Customer	51	M	3	Graduate	Married	80K−120K	Blue	36	.
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	34	.
4	709106358	Existing Customer	40	M	3	Uneducated	Married	60K−80K	Blue	21	.

Figure 1

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	Naiv
12691.0	777	11914.0	1.335	1144	42	1.625	0.061	
8256.0	864	7392.0	1.541	1291	33	3.714	0.105	
3418.0	0	3418.0	2.594	1887	20	2.333	0.000	
3313.0	2517	796.0	1.405	1171	20	2.333	0.760	
4716.0	0	4716.0	2.175	816	28	2.500	0.000	

Figure 2

3. Unsupervised Learning Model

In this section, we will go through the step-by-step process of building the model.

A. Data Preparation

First step is to install and load the Pyspark and needed libraries, which will be required for data loading and processing and create a PySpark session.

```
In [1]: import findspark
        findspark.init()
        import numpy as np
        import pandas as pd
        import os
        import seaborn as sns
        sns.set()
        from pyspark.ml import Pipeline
        import warnings
        warnings.filterwarnings('ignore')
        import pyspark.sql.functions as F
        from pyspark.sql.types import *
        from pyspark.sql import SparkSession
        import pyspark # only run after findspark.init()
        from pyspark.sql import SparkSession
        spark = SparkSession.builder.getOrCreate()
        from pyspark.sql.functions import when, count, isnull,isnan
        from pyspark.sql.functions import col,isnan, when, count
        from pyspark.ml.classification import LogisticRegression
        from pyspark.ml.clustering import KMeans
        import plotly
        from pyspark.ml.feature import StringIndexer
        from pyspark.ml.feature import OneHotEncoder, VectorAssembler
        import pyspark
        import matplotlib.pyplot as plt
        from pyspark.sql import SparkSession
        from pyspark.sql.types import *
        from pyspark.sql.functions import *
        from pyspark.sql.window import Window
        from pyspark.sql.functions import col, countDistinct
```

Figure 3

Next step is Loading dataset to Pyspark, as can be seen in figure 4.

```
data = spark.read.options(header='True', inferSchema='True', delimiter=',') \
  .csv("BankChurners.csv")
```

Figure 4

Pyspark, like pandas dtypes, offers an inbuilt function printSchema() that may be used to print the data types. The function along with the outcome is shown in figure 5.

```
data.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)
```

Figure 5

Spark DataFrames provide certain built-in statistical processing algorithms. In figure 6, the describe() method computes summary statistics on all numeric columns and provides the results as a Dataframe.

```python
#Summary Statistics
numeric_features = [t[0] for t in data.dtypes if t[1] == 'int' or t[1] == 'double']
data.describe(numeric_features).toPandas().transpose()
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
CLIENTNUM	10127	7.391776063336625E8	3.690378345023116E7	708082083	828343083
Customer_Age	10127	46.32596030413745	8.016814032549046	26	73
Dependent_count	10127	2.3462032191172115	1.29890834890379	0	5
Months_on_book	10127	35.928409203120374	7.98641633087208	13	56
Total_Relationship_Count	10127	3.8125802310654686	1.55440786533883	1	6
Months_Inactive_12_mon	10127	2.3411671768539546	1.0106223994182844	0	6
Contacts_Count_12_mon	10127	2.4553174681544387	1.1062251426359249	0	6
Credit_Limit	10127	8631.953698034848	9088.776650223148	1438.3	34516.0
Total_Revolving_Bal	10127	1162.8140614199665	814.9873352357533	0	2517
Avg_Open_To_Buy	10127	7469.139636614887	9090.685323679114	3.0	34516.0
Total_Amt_Chng_Q4_Q1	10127	0.7599406536980376	0.2192067692307027	0.0	3.397
Total_Trans_Amt	10127	4404.086303939963	3397.129253557085	510	18484
Total_Trans_Ct	10127	64.85869457884863	23.47257044923301	10	139
Total_Ct_Chng_Q4_Q1	10127	0.7122223758269962	0.23808609133294137	0.0	3.714
Avg_Utilization_Ratio	10127	0.2748935518909845	0.27569146925238736	0.0	0.999
_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1	10127	0.1599974639787803	0.36530101238046947	7.6642E-6	0.99958
_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2	10127	0.8400025708403275	0.36530103711017936	4.1998E-4	0.99999

Figure 6

In figure 7 and figure 8, we use Spark's built-in data processing functions, such as aggregation, to clean and prepare the data.

```python
data.groupBy("Attrition_Flag").sum().toPandas().transpose()
```

	0	1
Attrition_Flag	Existing Customer	Attrited Customer
sum(CLIENTNUM)	6289381352025	1196270267316
sum(Customer_Age)	393228	75915
sum(Dependent_count)	19851	3909
sum(Months_on_book)	304985	58862
sum(Total_Relationship_Count)	33274	5336
sum(Months_Inactive_12_mon)	19327	4382
sum(Contacts_Count_12_mon)	20029	4836
sum(Credit_Limit)	74178458.899999	13237336.2
sum(Total_Revolving_Bal)	10681135	1094683
sum(Avg_Open_To_Buy)	63497323.899999	12142653.2
sum(Total_Amt_Chng_Q4_Q1)	6566.331	1129.588
sum(Total_Trans_Amt)	39564575	5035607
sum(Total_Trans_Ct)	583717	73107
sum(Total_Ct_Chng_Q4_Q1)	6310.69	901.986
sum(Avg_Utilization_Ratio)	2519.5	264.347
es_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1)	1.578028	1618.71629
es_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2)	8498.42236	8.283675

Figure 7

```
data.groupBy("Attrition_Flag").agg({"Total_Revolving_Bal": "sum", "Credit_Limit": "max"}).show()
```

```
+-----------------+----------------+------------------------+
|   Attrition_Flag|max(Credit_Limit)|sum(Total_Revolving_Bal)|
+-----------------+----------------+------------------------+
|Existing Customer|         34516.0|                10681135|
|Attrited Customer|         34516.0|                 1094683|
+-----------------+----------------+------------------------+
```

Figure 8

For data cleaning we first remove duplicates and check for NaN and null values.

```
data.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in data.columns] ).toPandas().transpose()
```

	0
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
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1	0
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2	0

```
# Remove duplicate entries from people_df_sub
df = data.dropDuplicates()

# Count the number of rows
print("There were {} rows before removing duplicates,\
       and {} rows after removing duplicates".format(df.count(),
                                                      df.count()))
```

```
There were 10127 rows before removing duplicates,       and 10127 rows after removing duplicates
```

Figure 9

For encoding categorical variables, we use StringIndexer that maps a string column of labels to an ML column of label indices. If the input column is numeric, we cast it to string and index the string values. The indices are in [0, numLabels], as follows in figure 10 and figure 11.

```python
def indexing (df, inputCol, outputCol):
    indexer = StringIndexer(inputCol=inputCol, outputCol=outputCol)
    indexed = indexer.fit(df).transform(df)
    return indexed
```

```python
indexed = indexing (df, "Attrition_Flag", "Attrition_Flag_Index")
indexed = indexing (indexed, "Gender", "Gender_Index")
indexed = indexing (indexed, "Education_Level", "Education_Level_Index")
indexed = indexing (indexed, "Marital_Status", "Marital_Status_Index")
indexed = indexing (indexed, "Income_Category", "Income_Category_Index")
indexed = indexing (indexed, "Card_Category", "Card_Category_Index")
printdf(indexed)
```

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	.
0	789124683	Existing Customer	54	M	2	Unknown	Married	80K−120K	Blue	42	.
1	717296808	Existing Customer	67	F	1	Graduate	Married	Less than $40K	Blue	56	.
2	711551958	Attrited Customer	59	M	2	Post-Graduate	Married	60K−80K	Blue	46	.
3	713441958	Existing Customer	46	M	2	High School	Married	$120K +	Blue	36	.
4	789513858	Attrited Customer	43	M	3	Unknown	Married	40K−60K	Blue	35	.

Figure 10

_Level_Months_Inactive_12_mon_2	Attrition_Flag_Index	Gender_Index	Education_Level_Index	Marital_Status_Index	Income_Category_Index	Card_Category_Index
0.999790	0.0	1.0	2.0	0.0	2.0	0.0
0.999830	0.0	0.0	0.0	0.0	0.0	0.0
0.002801	1.0	1.0	5.0	0.0	3.0	0.0
0.999830	0.0	1.0	1.0	0.0	5.0	0.0
0.004558	1.0	1.0	2.0	0.0	1.0	0.0

Figure 11

B. The Model

As we want to do customer segmentation, we decided to build a K-means algorithm model to partition the dataset into four clusters based on our prior information about the data. The card categories consist of Blue, Gold, Silver, and Platinum. The clusters will be divided based on these categories.

Before building the model, a features column needed to be prepared. This features column will aggregate six columns that relate to the card category as is obvious from the correlation graph in figure 12. The six columns are:

- *Customer_Age*
- *Gender_Index*
- *Avg_Open_To_buy*
- *Credit_Limit*
- *Total_Trans_Amt*
- *Total_Trans_Ct*

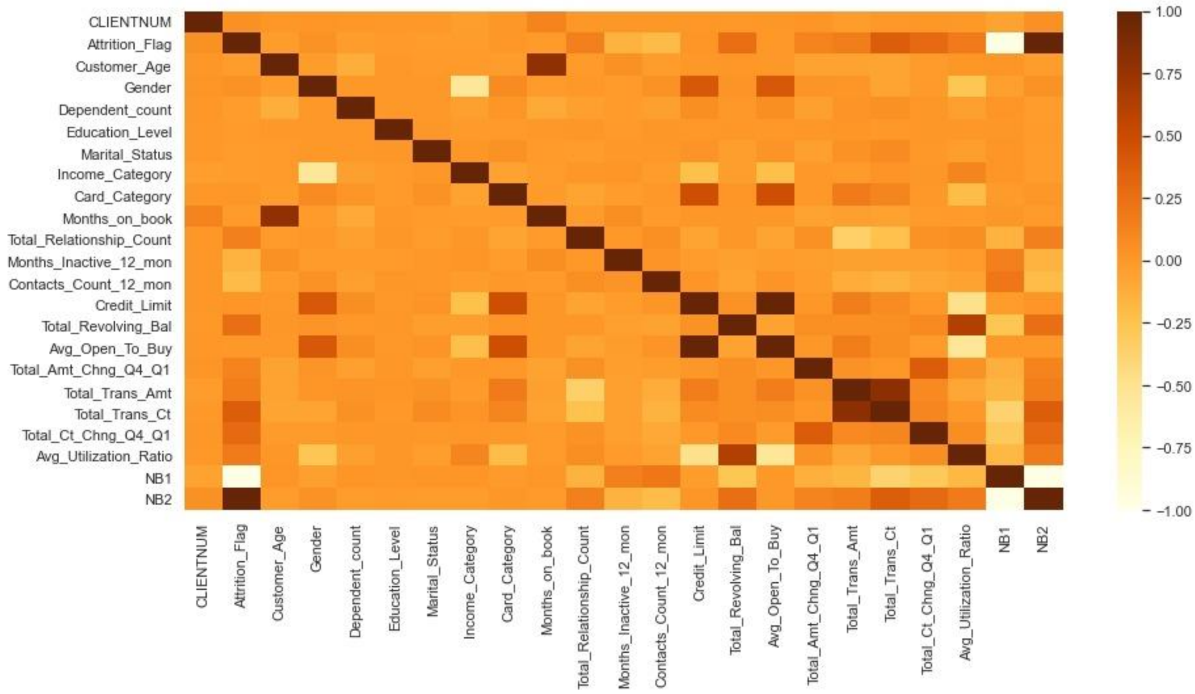

Figure 12

Furthermore, we use *VectorAssembler* on the chosen columns. VectorAssembler is a transformer that combines several columns into a single vector column. What we benefit from this transformer is instead of working on several separated raw features, we deal with only one column that contains the variety of features in a form of a vector. This eases the training of machine learning model and thus, smooths the process for us. Therefore, after encoding the mentioned columns, we use VectorAssembler to do the combination for us.

```
vecAssembler = VectorAssembler(outputCol="features")
```

```
vecAssembler.setInputCols (['Customer_Age', 'Gender_Index','Avg_Open_To_Buy',
                            'Credit_Limit', 'Total_Trans_Amt', 'Total_Trans_Ct'])
```

```
VectorAssembler_9589c4aa8193
```

```
transformed_X = vecAssembler.transform(X)
```

```
printdf(transformed_X)
```

Figure 13

Attrition_Flag_Index	Gender_Index	Education_Level_Index	Marital_Status_Index	Income_Category_Index	Card_Category_Index	features
0.0	1.0	2.0	0.0	2.0	0.0	[54.0, 1.0, 12217.0, 12217.0, 1110.0, 21.0]
0.0	0.0	0.0	0.0	0.0	0.0	[67.0, 0.0, 489.0, 3006.0, 1661.0, 32.0]
1.0	1.0	5.0	0.0	3.0	0.0	[59.0, 1.0, 1438.3, 1438.3, 844.0, 24.0]
0.0	1.0	1.0	0.0	5.0	0.0	[46.0, 1.0, 17942.0, 19727.0, 1245.0, 25.0]

Figure 14

Using PySpark and the machine learning library, an instance of the k-means clustering model had been made. The model will take the features column as input, the k will equal four as mentioned before, and the number of iterations will be 50. Then the model will be fitted on the transformed data as shown in figure 15 and the result will be a data frame containing two columns, the customer number and the cluster in which each customer is. The first column is called 'CLIENTNUM' and the second column will be 'prediction', which can be seen in figure 16.

```
k_means = KMeans(featuresCol='features', maxIter=50, k=4)
model = k_means.fit(transformed_X)
predictions = model.transform(transformed_X)
result = predictions.select('CLIENTNUM', 'prediction')
```

Figure 15

```
printdf(result)
```

	CLIENTNUM	prediction
0	789124683	2
1	717296808	0
2	711551958	0
3	713441958	2
4	789513858	0

Figure 16

4. Model Evaluation

In this section, we will discuss the outcomes of our K-Means model. To evaluate this unsupervised learning model, we have used two methods: *PCA* and *Silhouette Coefficient*.

A. Principal Component Analysis (PCA)

Principal Component Analysis is a well-known unsupervised learning technique for reducing data dimensionality. This is done by employing an orthogonal transformation to transform a set of potentially correlated observations into a set of values of linearly uncorrelated variables known as principal components.

PCA is used to improve interpretability while also minimizing information loss, and identify the most significant features in a dataset. In addition, it facilitates data plotting in 2D and 3D, also, it trains our model to project vectors to the top k main components' lower dimensional space.

In our case, we have started by training the model to project vectors to the top 3 main components in 3D space, as can be seen in figure 17.

```
from pyspark.ml.feature import PCA as PCAml
pca = PCAml(k=3, inputCol="features", outputCol="pca")
pca_model = pca.fit(transformed_X2)
pca_transformed = pca_model.transform(transformed_X2)
```

Figure 17

Then, we extract the principal components.

```
import numpy as np
x_pca = np.array(pca_transformed.rdd.map(lambda row: row.pca).collect())
```

Figure 18

Next, we retrieve cluster assignments from K-means assignments.

```
cluster_assignment = np.array(predictions.rdd.map(lambda row: row.prediction).collect()).reshape(-1,1)
```

Figure 19

Finally, we plot the PCA.

```
import seaborn as sns
import matplotlib.pyplot as plt

pca_data = np.hstack((x_pca,cluster_assignment))

pca_df = pd.DataFrame(data=pca_data, columns=("1st_principal", "2nd_principal","3nd_principal","cluster_assignment"))
sns.FacetGrid(pca_df,hue="cluster_assignment", height=40).map(plt.scatter, '1st_principal', '2nd_principal', "3nd_principal"


plt.show()
```

Figure 20

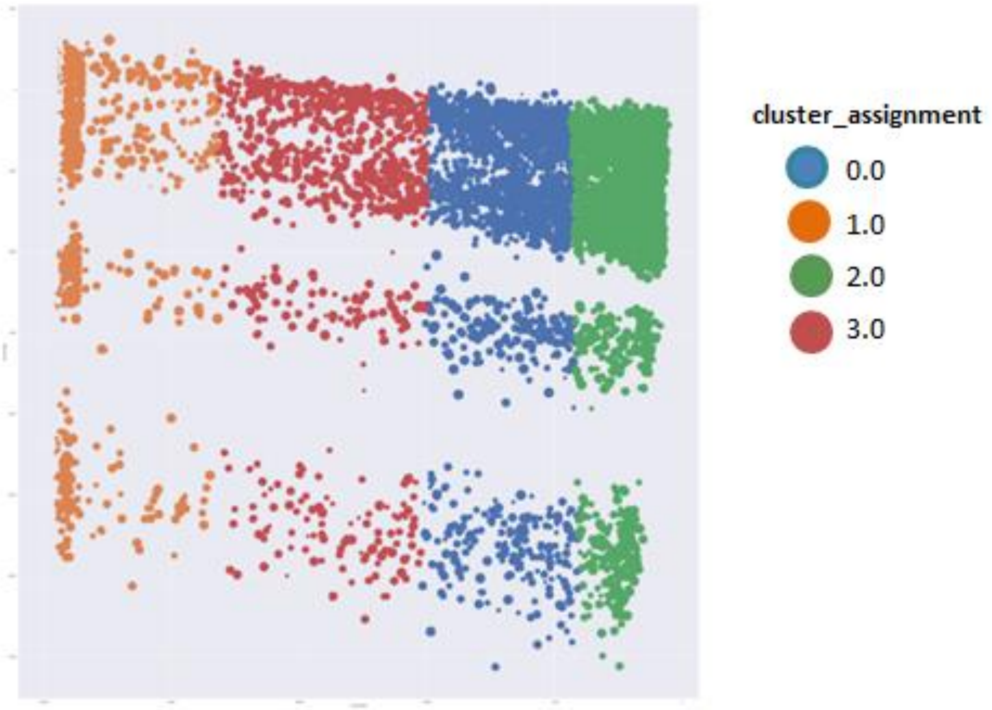

Figure 21

B. Silhouette Coefficient

The silhouette score is a measure of how similar a data point is within-cluster (cohesion) compared to other clusters (separation). The range it works with is [0, 1]. The closer the silhouette coefficient is to 1, the better.

The equation for calculating the silhouette coefficient for a particular data point is:

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Where *S(i)* is the silhouette coefficient of data point *i*

a(i) is the average distance between *i* and all the other points within a cluster (figure 22) , and

b(i) is the average distance from *i* to all clusters to which *i* does not belong (figure 23).

a(i)

Figure 22

b(i)

Figure 23

After calculating the silhouette score for our model, it has turned out to be approximately 0.7554, which is closer to 1 as shown in figure 24.

```
from pyspark.ml.evaluation import ClusteringEvaluator
evaluator = ClusteringEvaluator(featuresCol='features', metricName='silhouette', distanceMeasure='squaredEuclidean')
evaluation_score=evaluator.evaluate(predictions)
print(evaluation_score)

0.755350586635849
```

Figure 24

5. Conclusion

In this report, we have discussed the use of Apache Spark's PySpark in Customer Segmentation of bank customers. As PySpark is an interface for Spark in Python, machine learning models can be created and run easily on distributed systems by Python developers. Since our target in this problem is to segment or cluster the customer according to their card category, we have picked certain columns to work on based on their correlation. This have led to filtering the columns down to just with six, which are: *Customer_Age*, *Gender_Index*, *Avg_Open_To_Buy*, *Credit_Limit*, *Total_Trans_Amt*, and *Total_Trans_Ct*. In addition, we have used the unsupervised learning method K-Means in order to segment the customers in our dataset. This method is very beneficial, especially with our knowledge of the number of clusters expected from the model to present. After running the model for 50 iterations, the evaluation score for it is approximately 76%, which is a good model performance in case of unsupervised learning algorithm.

6. References

[1] zhyli, “Prediction of Churning Credit Card Customers,” Dec. 2020, doi: 10.5281/ZENODO.4322342.

7. Workload

Workload	Team Member
Data Collection	AbdelHafez, ElGhobashy, ElZahy, Sabry
Data Preparation	Sabry
Feature Engineering	ElZahy
Model Building	ElGhobashy
Model Evaluation	AbdelHafez
Report	AbdelHafez, ElGhobashy, ElZahy, Sabry