ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA HỆ THỐNG THÔNG TIN

BÁO CÁO CUỐI KÌ

MÔN: DỮ LIỆU LỚN

ĐỀ TÀI: CREDIT CARD FOR CLUSTERING

Giảng viên hướng dẫn:

ThS. Nguyễn Hồ Duy Tri

Nhóm sinh viên thực hiện:

Lê Quang Nhật - 20521705

Đinh Hoàng Trí – 20522046

Huỳnh Quốc Nguyên - 20521671

Thành phố Hồ Chí Minh, tháng 12 năm 2023

LỜI CẢM ƠN

Đầu tiên, nhóm em xin gởi lời cảm ơn chân thành đến thầy Nguyễn Hồ Duy Tri - giảng viên phụ trách môn học Dữ liệu lớn, đã hướng dẫn và cung cấp những kiến thức bổ ích, hỗ trợ nhóm trong quá trình nghiên cứu. Nhóm cũng muốn gửi lời cảm ơn đến các tác giả đã chia sẻ kiến thức của mình về Apache Spark, Hadoop trên các diễn đàn và tài liệu trực tuyến. Các thông tin này đã giúp nhóm hiểu sâu hơn về phân tán dữ liệu và phát triển các kỹ năng cần thiết để sử dụng công cụ này hiệu quả hơn.

Dựa trên những kiến thức được thầy cung cấp trên lớp, kết hợp với việc tự nghiên cứu các công cụ và kiến thức mới, nhóm cố gắng thực hiện dự án một cách tốt nhất có thể. Trong thời gian thực hiện dự án, nhóm đã sử dụng kiến thức nền tảng đã tích lũy được, kết hợp học tập và nghiên cứu kiến thức mới. Từ đó, nhóm của chúng em sử dụng đầy đủ các thông tin thu thập được để đưa ra báo cáo dự án tốt nhất có thể. Tuy nhiên, trong quá trình thực hiện, nhóm không tránh khỏi những thiếu sót. Vì vậy, nhóm rất mong nhận được sự góp ý của thầy để nhóm em hoàn thiện hơn kiến thức và chuẩn bị cho các đề tài khác trong tương lai.

Sau cùng, nhóm em xin kính chúc thầy thật dồi dào sức khỏe, niềm tin để tiếp tục thực hiện sứ mệnh cao đẹp là truyền đạt kiến thức cho các bạn sinh viên.

Nhóm sinh viên thực hiện

NHẬN XÉT CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MỤC LỤC

CHƯƠNG I: GIỚI THIỆU TỔNG QUAN

1. Lý do chọn đề tài

Trong xã hội ngày nay, với sự phát triển không ngừng, con người ngày càng đặt trọng tâm vào sự tiện lợi và nhanh chóng. Nhằm đáp ứng xu hướng này, các ngân hàng đã tích hợp dịch vụ thẻ tín dụng, biến chúng thành một phần quan trọng và không thể thiếu trong cuộc sống hiện đại, giúp đáp ứng mọi nhu cầu thanh toán một cách thuận tiện và nhanh chóng. Việc sở hữu thẻ tín dụng mang lại cảm giác an toàn, giúp người dùng không cần mang theo nhiều tiền mặt, mà vẫn có thể thanh toán linh hoạt ở mọi nơi và mọi thời điểm.

Tuy nhiên, khi số khách hàng sử dụng thẻ tín dụng tăng lên, nhiều vấn đề phức tạp cũng nảy sinh. Đối với những người thường xuyên sử dụng thẻ tín dụng, họ mang lại nhiều giá trị cho ngân hàng thông qua thuế, lãi suất, doanh thu và dữ liệu khách hàng. Ngược lại, người dùng thẻ không thường xuyên sẽ đóng góp ít giá trị cho ngân hàng. Điều này đặt ra thách thức cho các ngân hàng cần phải phát triển phương pháp quản lý thẻ tín dụng với hạn mức chi tiêu đa dạng, phù hợp với từng nhóm đối tượng khách hàng. Từ đó, có thể xây dựng chiến lược kinh doanh linh hoạt, cung cấp ưu đãi và chính sách khuyến mãi phù hợp để đáp ứng đa dạng đối tượng người dùng thẻ tín dụng.

Do đó, nhóm đã chọn đề tài: “Phân cụm khách hàng dựa trên thẻ tín dụng”. Đề tài này giúp chúng ta có thể dễ dàng xác định rõ ràng các nhóm khách hàng cũng như các mô hình, xu hướng hay các biểu đồ nhằm hỗ trợ các quyết định kinh doanh cũng như tạo ra cơ hội mới để cá nhân hay doanh nghiệp hiểu rõ hơn về nhu cầu khách hàng của chính mình.

Ngoài ra, việc nghiên cứu về việc phân cụm dữ liệu cũng mở ra những cơ hội mới trong việc áp dụng các kỹ thuật máy học, khai thác dữ liệu để tối ưu hóa các quy trình kinh doanh, cũng như tạo ra những sản phẩm và dịch vụ tài chính hiệu quả hơn.

2. Tổng quan về bộ dữ liệu

Tên bộ dữ liệu: Credit Card Dataset for Clustering

Kích thước: 902.88kB

Loại tệp sử dụng: .csv

Khả năng sử dụng: 5.88 (theo Kaggle)

Số lượt đánh giá cao: 495 (20/12/2023)

Link Dataset: Credit Card Dataset for Clustering

3. Chi tiết bộ dữ liệu

Bộ dữ liệu **Credit Card Dataset for Clustering** gồm 18 cột với gần 9000 dòng dữ liệu. Bao gồm:

STT	Tên cột	Mô tả
1	CUST_ID	Id của thẻ tín dụng.
2	BALANCE	Số dư trong tài khoản.
3	BALNCE_FREQUENCY	Tần suất cập nhật số dư (1 là thường xuyên cập nhật, 0 là không thường xuyên cập nhật).
4	PURCHASES	Tổng số tiền giao dịch.
5	ONEOFF_PURCHASES	Số tiền tối đa trong 1 lần giao dịch.
6	INSTALLMENTS_PURCHASES	Số tiền trả góp.
7	CASH_ADVANCE	Tiền mặt do người dùng trả trước.
8	PURCHASES_FREQUENCY	Tần suất giao dịch, giá trị 0 và 1 (1 = thường xuyên giao dịch, 0 = không thường xuyên giao dịch).
9	ONEOFFPURCHASESFREQUENCY	Tần suất giao dịch trong 1 lần (1 = thường xuyên giao dịch, 0 = không

		thường xuyên giao dịch).
10	PURCHASESINSTALLMENTSFREQUENCY	Tần suất mua hàng trả góp (1 = thường xuyên, 0 = không thường xuyên)
11	CASHADVANCEFREQUENCY	Tần suất thanh toán trả trước
12	CASHADVANCETRX	Số giao dịch được thanh toán bằng khoản trả trước
13	PURCHASES_TRX	Số giao dịch mua hàng được thực hiện
14	CREDIT_LIMIT	Hạn mức thẻ tín dụng
15	PAYMENTS	Số tiền đã thanh toán
16	MINIMUM_PAYMENTS	Số tiền tối thiểu để thanh toán
17	PRCFULLPAYMENT	Tỷ lệ thanh toán đầy đủ của khách hàng
18	TENURE	Thời hạn sử dụng dịch vụ thẻ tín dụng

4. Các chỉ số cơ bản dùng cho bộ dữ liệu

Chỉ số	**Mô tả**
Count	Đếm số phần tử
Standard Deviation	Độ lệch chuẩn
Min	Gía trị nhỏ nhất
Max	Gía trị lớn nhất
Quartile	Tứ phân vị
Median	Trung vị
Mean	Gía trị trung bình
Mode	Gía trị xuất hiện nhiều nhất
Variance	Phương sai
Trending	Xu hướng

CHƯƠNG II: MÔ TẢ BÀI TOÁN

1. Mô tả sơ lược về bài toán

Bài toán mà nhóm em đang nghiên cứu xoay quanh việc phân cụm khách hàng dựa trên thông tin liên quan đến việc sử dụng thẻ tín dụng. Trong bối cảnh một số lượng ngày càng tăng của người sử dụng thẻ tín dụng, nhóm em tin rằng ngân hàng cần quan tâm đến việc hiểu rõ hơn về đặc điểm và nhu cầu của từng nhóm đối tượng khách hàng.

Vấn đề cơ bản mà nhóm chúng em đặt ra là làm thế nào để phân loại khách hàng thành các nhóm dựa trên các yếu tố như tần suất sử dụng thẻ, hạn mức chi tiêu, loại giao dịch phổ biến, và các hành vi thanh toán khác nhau. Vì thế chúng em mong muốn xây dựng một cơ sở thông tin chi tiết về các đặc điểm khách hàng và giúp ngân hàng hiểu rõ hơn về cách tối ưu hóa dịch vụ và chính sách để đáp ứng nhu cầu đặc biệt của từng nhóm.

Chúng em sẽ sử dụng các kỹ thuật máy học và khai thác dữ liệu để phân loại và phân cụm khách hàng. Nhóm cũng mong muốn các mô hình máy học mà nhóm áp dụng để phân tích sẽ cho ra kết quả tốt và tạo ra các nhóm có ý nghĩa thực tế. Điều này giúp tăng cường khả năng đưa ra quyết định của ngân hàng về chiến lược phục vụ khách hàng một cách linh hoạt và hiệu quả.

Bài toán của chúng em không chỉ hướng đến việc hiểu rõ về khách hàng mà còn mở ra cơ hội để tối ưu hóa các quy trình kinh doanh bằng cách sử dụng thông tin được phân loại để đưa ra các chiến lược kinh doanh linh hoạt và hiệu quả. Đồng thời, nó cũng mở ra cánh cửa cho việc phát triển các sản phẩm và dịch vụ tài chính mới, được tối ưu hóa dựa trên hiểu biết sâu sắc về nhu cầu khách hàng đặc thù từng nhóm.

Đó cũng chính là lý do nhóm chúng em làm đề tài **Phân cụm khách hàng dựa trên thẻ tín dụng** của họ. Nhóm sẽ áp dụng kiến thức từ môn Dữ liệu lớn – BigData được học trên trường để xử lý đề bài này cụ thể nhóm sẽ sử dụng framework **Apache Spark** để xử lý dữ liệu đồng thời nhóm sẽ **không sử dụng** các thư viện xử lý dữ liệu tập trung để tiền xử lý như **Pandas**, … **không sử dụng** các thư viện có sẵn như **MLLib** hay các thư viện máy học bên ngoài như **Apache Mahout, Scikitlearn, …** để thực hiện việc khai thác dữ liệu cũng như **không sử dụng** thuật toán **Logistic Regression, Cây quyết định và Naïve Bayes.**

2. Các kỹ thuật áp dụng vào bài toán

- Framework Apache Spark (Pyspark), Hadoop.
- Dùng một số thư viện hỗ trợ tính toán như (math, random, re).
- Dùng module Pyspark SQL gồm Spark SQL và Dataframe API.
- Dùng máy ảo Ubuntu.
- Công cụ Jupyternotebook.

CHƯƠNG III: XỬ LÝ VÀ KHAI THÁC DỮ LIỆU

1. Tiền xử lý dữ liệu

1.1. Thêm thư viện và bộ dữ liệu

Thêm thư viện cần thiết:

```
import re
import math
import string
import random
import numpy as np
import random as rd
from pyspark import SparkContext
from pyspark.sql import *
from pyspark.sql.types import *
from pyspark.sql import functions as F
from pyspark.sql.functions import *
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DateType, FloatType, DoubleType
```

Dữ liệu sau khi được đọc từ file csv:

```
[7] _data.show()
```

CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CA
C10001	40.900749	0.818182	95.4	0.0	95.4	0.0	0.166667	0.0	0.083333	
C10002	3202.467416	0.909091	0.0	0.0	0.0	6442.945483	0.0	0.0	0.0	
C10003	2495.148862	1.0	773.17	773.17	0.0	0.0	1.0	1.0	0.0	
C10004	1666.670542	0.636364	1499.0	1499.0	0.0	205.788017	0.083333	0.083333	0.0	
C10005	817.714335	1.0	16.0	16.0	0.0	0.0	0.083333	0.083333	0.0	
C10006	1809.828751	1.0	1333.28	0.0	1333.28	0.0	0.666667	0.0	0.583333	
C10007	627.260806	1.0	7091.01	6402.63	688.38	0.0	1.0	1.0	1.0	
C10008	1823.652743	1.0	436.2	0.0	436.2	0.0	1.0	0.0	1.0	
C10009	1014.926473	1.0	861.49	661.49	200.0	0.0	0.333333	0.083333	0.25	
C10010	152.225975	0.545455	1281.6	1281.6	0.0	0.0	0.166667	0.166667	0.0	
C10011	1293.124939	1.0	920.12	0.0	920.12	0.0	1.0	0.0	1.0	
C10012	630.794744	0.818182	1492.18	1492.18	0.0	0.0	0.25	0.25	0.0	
C10013	1516.92862	1.0	3217.99	2500.23	717.76	0.0	1.0	0.25	0.916667	
C10014	921.693369	1.0	2137.93	419.96	1717.97	0.0	0.75	0.166667	0.75	
C10015	2772.772734	1.0	0.0	0.0	0.0	346.81139	0.0	0.0	0.0	
C10016	6886.213231	1.0	1611.7	0.0	1611.7	2301.491267	0.5	0.0	0.5	
C10017	2072.074354	0.875	0.0	0.0	0.0	2784.274703	0.0	0.0	0.0	
C10018	41.089489	0.454545	519.0	0.0	519.0	0.0	0.416667	0.0	0.333333	
C10019	1989.072228	1.0	504.35	166.0	338.35	0.0	0.666667	0.083333	0.583333	
C10020	3577.970933	1.0	398.64	0.0	398.64	0.0	1.0	0.0	1.0	

only showing top 20 rows

Sử dụng pyspark.sql.DataFrame.explain để giúp hiển thị kế hoạch thực hiện và tối ưu hóa của cho DataFrame.

Cung cấp thông tin về dữ liệu được phân phối, cách thức thực hiện các phép gộp, sắp xếp, lọc, và các thông tin hữu ích khác.

```
[8] _data.explain()

== Physical Plan ==
FileScan csv [CUST_ID#0,BALANCE#1,BALANCE_FREQUENCY#2,PURCHASES#3
```

Xem cấu trúc của DataFrame _data, giúp hiểu rõ các cột bao gồm tên cột, kiểu dữ liệu của cột và thuộc tính nullable của mỗi cột.

```
[9] _data.printSchema()
```

```
root
 |-- CUST_ID: string (nullable = true)
 |-- BALANCE: double (nullable = true)
 |-- BALANCE_FREQUENCY: double (nullable = true)
 |-- PURCHASES: double (nullable = true)
 |-- ONEOFF_PURCHASES: double (nullable = true)
 |-- INSTALLMENTS_PURCHASES: double (nullable = true)
 |-- CASH_ADVANCE: double (nullable = true)
 |-- PURCHASES_FREQUENCY: double (nullable = true)
 |-- ONEOFF_PURCHASES_FREQUENCY: double (nullable = true)
 |-- PURCHASES_INSTALLMENTS_FREQUENCY: double (nullable = true)
 |-- CASH_ADVANCE_FREQUENCY: double (nullable = true)
 |-- CASH_ADVANCE_TRX: integer (nullable = true)
 |-- PURCHASES_TRX: integer (nullable = true)
 |-- CREDIT_LIMIT: double (nullable = true)
 |-- PAYMENTS: double (nullable = true)
 |-- MINIMUM_PAYMENTS: double (nullable = true)
 |-- PRC_FULL_PAYMENT: double (nullable = true)
 |-- TENURE: integer (nullable = true)
```

Kiểm tra số dòng và số cột của DataFrame.

```
[10] print((_data.count(), len(_data.columns)))
```

```
(8950, 18)
```

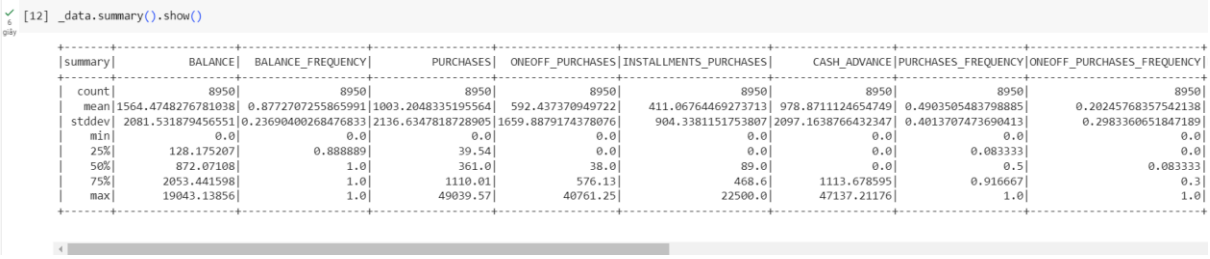
Tiến hành loại bỏ những cột không cần thiết cũng như không phục vụ vào quá trình tính toán. Với DataFrame _data, cột ‘CUST_ID’ không cần thiết trong quá trình phân cụm nên tiến hành loại bỏ.

```
[11] _data = _data.select([col for col in _data.columns if col != 'CUST_ID'])
```

1.2. Tính toán các bộ chỉ số

1.2.1. Các chỉ số cơ bản

Tiến hành tính toán các chỉ số đã đề cập trong Chương I mục 4.



```
[12] _data.summary().show()
```

summary	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY
count	8950	8950	8950	8950	8950	8950	8950	8950
mean	1564.4748276781038	0.8772707255865991	1003.2048335195564	592.4373709497 22	411.06764469273713	978.8711124654749	0.4903505483798885	0.20245768357542138
stddev	2081.5318794565 51	0.2369040026847 6833	2136.6347818728905	1659.8879174378076	904.3381151753807	2097.1638766432347	0.4013707473690413	0.2983360651847189
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
25%	128.175207	0.888889	39.54	0.0	0.0	0.0	0.083333	0.0
50%	872.07108	1.0	361.0	38.0	89.0	0.0	0.5	0.083333
75%	2053.441598	1.0	1110.01	576.13	468.6	1113.678595	0.916667	0.3
max	19043.13856	1.0	49039.57	40761.25	22500.0	47137.21176	1.0	1.0

Columns	count	stddev	min	25%	50%	75%	max
BALANCE	8950	2081.532	0	128.1752	872.0711	2053.442	19043.14
BALANCE_ FREQUENCY	8950	0.236904	0	0.888889	1	1	1
PURCHASES	8950	2136.635	0	39.54	361	1110.01	49039.57
ONEOFF_PURCHASES	8950	1659.888	0	0	38	576.13	40761.25
INSTALLMENTS_ PURCHASES	8950	904.3381	0	0	89	468.6	22500
CASH_ADVANCE	8950	2097.164	0	0	0	1113.679	47137.21
PURCHASES_ FREQUENCY	8950	0.401371	0	0.083333	0.5	0.916667	1
ONEOFF_PURCHASES_ FREQUENCY	8950	0.298336	0	0	0.083333	0.3	1
PURCHASES_ INSTALLMENTS_ FREQUENCY	8950	0.397448	0	0	0.166667	0.75	1
CASH_ADVANCE_ FREQUENCY	8950	0.200121	0	0	0	0.222222	1.5
CASH_ADVANCE_TRX	8950	6.824647	0	0	0	4	123
PURCHASES_TRX	8950	24.85765	0	1	7	17	358
CREDIT_LIMIT	8949	3638.816	50	1600	3000	6500	30000
PAYMENTS	8950	2895.064	0	383.2312	856.1963	1900.699	50721.48
MINIMUM_PAYMENTS	8637	2372.447	0.019163	169.0797	312.1673	825.2711	76406.21
PRC_FULL_PAYMENT	8950	0.292499	0	0	0	0.142857	1
TENURE	8950	1.338331	6	12	12	12	12

Trong đó:

- Count: đếm số phần tử.
- Stddev: độ lệch chuẩn của tập dữ liệu, nó thể hiện độ biến động hoặc phân tán trong một tập giá trị. Nó đo lường mức độ mà các điểm dữ liệu khác biệt so với giá trị trung bình.
- Min: giá trị nhỏ nhất của dữ liệu.
- Max: giá trị lớn nhất của dữ liệu.
- 25%: là phân vị thứ nhất (Q1), thể hiện 25% giá trị của tập dữ liệu.
- 50%: là phân vị trung bình (Q2), thể hiện 50% giá trị của tập dữ liệu.
- 75%: là phân vị thứ ba (Q3), thể hiện 75% giá trị của tập dữ liệu.

1.2.2. Các chỉ số khác

Median: là giá trị trung vị, giá trị ở giữa của tập dữ liệu khi dữ liệu được sắp xếp theo thứ tự tăng dần. Ví dụ: ngân hàng muốn biết được trung vị của số dư trong tài khoản của khách hàng, ta có thể sử dụng hàm để lấy được giá trị:

```
def median_formula(data_frame: DataFrame, column_name):
    # Calculate median value of the column name
    median_result = data_frame.agg(percentile_approx(column_name, 0.5, lit(1000000)).alias("Median"))

    # Return a dataframe
    return median_result
```

```
[15] median_formula(_data, "BALANCE").show()

+----------+
|    Median|
+----------+
|873.090183|
+----------+
```

Mode: là giá trị xuất hiện thường xuyên nhất trong tập dữ liệu. Một tập dữ liệu có thể không có Mode nếu mỗi giá trị chỉ xuất hiện 1 lần hoặc nếu có nhiều hơn 1 giá trị xuất hiện với tần số cao nhất. Ví dụ: ngân hàng muốn xem hạn mức xuất hiện nhiều nhất mà khách hàng sử dụng có thể sử dụng hàm:

```
def mode_formula(data_frame: DataFrame, column_name):
    # Group by the specified column and count occurrences
    mode_result = data_frame.groupBy(column_name).agg(count(column_name).alias('count'))
    # Calculate mode value of the column name
    mode_result = mode_result.withColumn('rank', row_number().over(Window.orderBy(desc('count'))))
    mode_result = mode_result.filter(mode_result['rank'] == 1).drop('rank', 'count')

    # Return a dataframe
    return mode_result
```

```
mode_formula(_data, "CREDIT_LIMIT").show()
```

```
+------------+
|CREDIT_LIMIT|
+------------+
|      3000.0|
+------------+
```

Variance: là phương sai đo lường độ lệch của mỗi giá trị từ giá trị trung bình của tập dữ liệu. Phương sai sẽ bằng 0 nếu các giá trị giống nhau.Ví dụ: ngân hàng muốn xem độ lệch của thời gian sử dụng thẻ của khách hàng có thể sử dụng hàm:

```
def variance_formula(data_frame: DataFrame, column_name):
    # Calculate median value of the column name
    variance_result = data_frame.select(variance(column_name))

    # Return a dataframe
    return variance_result
```

```
variance_formula(_data, "TENURE").show()
```

```
+------------------+
|  var_samp(TENURE)|
+------------------+
|1.7911292482353518|
+------------------+
```

Quartile: là điểm tứ phân vị, là các giá trị chia tập dữ liệu thành 4 phần bằng nhau. Quartile thường được sử dụng trong thống kê và phân tích dữ liệu để đo lường sự phân bố của một tập dữ liệu. Có 3 điểm tứ phân vị chính:

- Điểm tứ phân vị thứ nhất (Q1): Đây là điểm chia tách dữ liệu thành hai phần bằng nhau, trong đó 25% dữ liệu thấp hơn Q1 và 75% dữ liệu cao hơn Q1.

- Điểm tứ phân vị thứ hai (Q2): Điểm tứ phân vị thứ hai tương đương với giá trị trung vị của tập dữ liệu, tức là 50% dữ liệu thấp hơn Q2 và 50% dữ liệu cao hơn Q2. Q2 cũng thường được gọi là median (trung vị).
- Điểm tứ phân vị thứ ba (Q3): Đây là điểm chia tách dữ liệu thành hai phần bằng nhau, trong đó 75% dữ liệu thấp hơn Q3 và 25% dữ liệu cao hơn Q3.

Ví dụ: Ngân hàng muốn kiểm tra xem những tài khoản có mức giao dịch vượt ngưỡng:

```
def quartiles_formula(data_frame:DataFrame, column_name):
        quartiles_result = data_frame.approxQuantile(column_name, [0.25, 0.5, 0.75], 0)

        # Return a list
        return quartiles_result

def outliers_formula(data_frame:DataFrame, column_name, factor  = 1.5):
        q1, q3 = quartiles_formula(data_frame, column_name)[0], quartiles_formula(data_frame, column_name)[2]
        iqr = q3 - q1
        lower_bound = (q1 - factor * iqr)
        upper_bound = q3 + factor * iqr
        outlier_result = data_frame.filter((col(column_name) < lower_bound) | (col(column_name) > upper_bound))
        print(f"DataFrame with lower bound ({lower_bound}) <= {column_name} <= upper bound ({upper_bound}):")
        outlier_result = outlier_result.sort(asc(column_name))
        # Return a dataframe contain outliner
        return outlier_result
```

```
outliers_formula(_data, "PURCHASES").show()
```

```
DataFrame with lower bound (-1566.3050000000003) <= PURCHASES <= upper bound (2716.0550000000003):
+----------+-----------------+---------+---------------+--------------------+-----------+------------------+------------------------+----------------------------+
|   BALANCE|BALANCE_FREQUENCY|PURCHASES|ONEOFF_PURCHASES|INSTALLMENTS_PURCHASES|CASH_ADVANCE|PURCHASES_FREQUENCY|ONEOFF_PURCHASES_FREQUENCY|PURCHASES_INSTALLMENTS_FREQUENCY|
+----------+-----------------+---------+---------------+--------------------+-----------+------------------+------------------------+----------------------------+
|1646.715644|              1.0|  2720.68|        2580.19|              140.49|        0.0|               1.0|                0.833333|                    0.666667|
| 706.675991|              1.0|  2721.16|        2417.41|              303.75|        0.0|               1.0|                     0.5|                         1.0|
| 251.715214|         0.909091|  2722.34|        1744.63|              977.71|        0.0|              0.75|                    0.75|                    0.583333|
| 483.402486|         0.777778|  2723.47|        2723.47|                 0.0|        0.0|          0.666667|                0.666667|                         0.0|
|  376.15858|              1.0|  2724.76|         572.76|              2152.0|        0.0|               1.0|                     1.0|                         1.0|
| 297.004363|              1.0|   2729.0|         1409.0|              1320.0|        0.0|               1.0|                0.083333|                         1.0|
| 787.341687|              1.0|  2732.66|        2548.47|              184.19|        0.0|               1.0|                     1.0|                    0.416667|
|2283.366217|              1.0|  2734.77|        1645.87|              1088.9| 4303.863961|          0.833333|                0.583333|                         0.5|
| 328.436912|         0.727273|  2738.82|        2408.82|               330.0|        0.0|          0.583333|                0.333333|                    0.333333|
|5063.070919|              1.0|  2742.33|          517.3|              2225.03|  367.915403|          0.916667|                    0.75|                    0.916667|
|3676.225496|              1.0|  2743.35|        1299.63|             1443.72| 1095.478557|               1.0|                0.666667|                    0.916667|
|5630.793848|              1.0|  2743.66|        2743.66|                 0.0| 2638.893572|              0.75|                    0.75|                         0.0|
|  313.87913|              1.0|  2746.84|         2170.0|              576.84|        0.0|               1.0|                0.916667|                    0.916667|
|1508.513765|         0.818182|  2747.12|        1247.12|              1500.0|  246.046033|              0.25|                0.083333|                    0.166667|
| 421.796337|              1.0|  2748.74|        1623.74|              1125.0|        0.0|               1.0|                     1.0|                    0.666667|
| 180.238427|              1.0|  2749.92|            0.0|             2749.92|        0.0|               1.0|                     0.0|                         1.0|
|2895.765754|         0.909091|  2753.35|         1960.0|              793.35|        0.0|          0.727273|                0.090909|                    0.636364|
|1910.793773|              1.0|  2754.09|           40.0|             2714.09|        0.0|               1.0|                0.083333|                         1.0|
|2755.306896|         0.909091|  2754.49|         1974.0|               780.49| 3193.924548|              0.75|                0.416667|                    0.583333|
|5858.617133|              1.0|  2761.41|        2484.65|              276.76|        0.0|               1.0|                0.666667|                    0.666667|
+----------+-----------------+---------+---------------+--------------------+-----------+------------------+------------------------+----------------------------+
only showing top 20 rows
```

Trending: có thể sử dụng để phân tích xu hướng phổ biến. Ví dụ: ngân hàng muốn thống kê tổng số lượng tiền đã giao dịch đối với từng hạn mức thẻ tín dụng:

```
def trending_formula(data_frame: DataFrame, column_name_1, column_name_2):
        window_spec = Window.partitionBy(column_name_1)
        sum_total_amount = sum(column_name_2).over(window_spec).alias('Total')

        trending_result = data_frame.select(column_name_1, sum_total_amount).distinct()
        trending_result = trending_result.orderBy(col('Total').desc())
        return trending_result
```

```
trending_formula(_data, 'CREDIT_LIMIT', 'PURCHASES' ).show()
```

```
+------------+------------------+
|CREDIT_LIMIT|             Total|
+------------+------------------+
|      6000.0| 538218.8300000002|
|      3000.0| 470168.7900000003|
|      7500.0|467134.88000000006|
|      4000.0| 421076.3500000003|
|      5000.0| 410579.8099999998|
|      2500.0|377297.54999999976|
|      7000.0|361833.76999999996|
|      9000.0| 315217.7400000001|
|      4500.0| 296967.1199999999|
|      1000.0|283948.12000000005|
|     10000.0|         271653.21|
|      1500.0|         269842.26|
|      6500.0|252962.38999999996|
|      8500.0|241865.50999999995|
|     12000.0|218878.30000000005|
|      2000.0|215974.24999999997|
|      1200.0| 211335.9299999999|
|     10500.0|188954.34000000005|
|     18000.0|180403.42999999996|
|      9500.0|179578.72999999992|
+------------+------------------+
only showing top 20 rows
```

1.2.3. Sắp xếp các giá trị tăng dần, giảm dần

Asc: sắp xếp các giá trị trong tập dữ liệu theo thứ tự tăng dần. Ví dụ: ngân hàng muốn xem số dư của khách hàng theo thứ tự tăng dần có thể sử dụng hàm:

```
def asc_formula(data_frame: DataFrame, column_name):
    # Calculate median value of the column name
    asc_result = data_frame.sort(asc(column_name))

    # Return a dataframe
    return asc_result
```

```
asc_formula(_data, "BALANCE").show()
```

```
+-------+-----------------+---------+---------------+---------------------+-----------+------------------+--------------------------+
|BALANCE|BALANCE_FREQUENCY|PURCHASES|ONEOFF_PURCHASES|INSTALLMENTS_PURCHASES|CASH_ADVANCE|PURCHASES_FREQUENCY|ONEOFF_PURCHASES_FREQUENCY|
+-------+-----------------+---------+---------------+---------------------+-----------+------------------+--------------------------+
|    0.0|              0.0|   199.73|           22.0|               177.73|        0.0|          0.454545|                  0.090909|
|    0.0|              0.0|     63.3|            0.0|                 63.3|        0.0|              0.25|                       0.0|
|    0.0|              0.0|    300.0|            0.0|                300.0|        0.0|          0.857143|                       0.0|
|    0.0|              0.0|    70.65|          70.65|                  0.0|        0.0|          0.083333|                  0.083333|
|    0.0|              0.0|     23.0|            0.0|                 23.0|        0.0|          0.083333|                       0.0|
|    0.0|              0.0|    61.21|          61.21|                  0.0|        0.0|          0.083333|                  0.083333|
|    0.0|              0.0|      0.0|            0.0|                  0.0| 310.888779|               0.0|                       0.0|
|    0.0|              0.0|   103.68|         103.68|                  0.0|        0.0|          0.111111|                  0.111111|
|    0.0|              0.0|   313.89|            0.0|               313.89|        0.0|          0.916667|                       0.0|
|    0.0|              0.0|   125.37|            0.0|               125.37|        0.0|               1.0|                       0.0|
|    0.0|              0.0|    12.65|            0.0|                12.65|        0.0|          0.083333|                       0.0|
|    0.0|              0.0|   339.85|         339.85|                  0.0| 103.334378|               1.0|                       1.0|
|    0.0|              0.0|   2600.0|         2600.0|                  0.0|        0.0|          0.083333|                  0.083333|
|    0.0|              0.0|    990.0|            0.0|                990.0|        0.0|          0.818182|                       0.0|
|    0.0|              0.0|   174.12|            0.0|               174.12|        0.0|               1.0|                       0.0|
|    0.0|              0.0|   176.03|         176.03|                  0.0|        0.0|             0.125|                     0.125|
|    0.0|              0.0|    581.0|            0.0|                581.0|        0.0|          0.583333|                       0.0|
|    0.0|              0.0|    178.8|            0.0|                178.8|        0.0|               1.0|                       0.0|
|    0.0|              0.0|     94.5|            0.0|                 94.5|        0.0|          0.166667|                       0.0|
|    0.0|              0.0|   1200.0|         1200.0|                  0.0|        0.0|          0.083333|                  0.083333|
+-------+-----------------+---------+---------------+---------------------+-----------+------------------+--------------------------+
only showing top 20 rows
```

Desc: sắp xếp các giá trị trong tập dữ liệu theo thứ tự giảm dần. Ví dụ: ngân hàng muốn xem số dư trong tài khoản của khách hàng theo thứ tự giảm dần có thể sử dụng hàm:

```
def desc_formula(data_frame: DataFrame, column_name):
    # Calculate median value of the column name
    desc_result = data_frame.orderBy(desc(column_name))

    # Return a dataframe
    return desc_result
```

```
desc_formula(_data, "BALANCE").show()
```

```
+----------+-----------------+---------+---------------+---------------------+-----------+------------------+--------------------------+-----------------------------+
|   BALANCE|BALANCE_FREQUENCY|PURCHASES|ONEOFF_PURCHASES|INSTALLMENTS_PURCHASES|CASH_ADVANCE|PURCHASES_FREQUENCY|ONEOFF_PURCHASES_FREQUENCY|PURCHASES_INSTALLMENTS_FREQUENCY|
+----------+-----------------+---------+---------------+---------------------+-----------+------------------+--------------------------+-----------------------------+
|19043.13856|              1.0| 22009.92|        9449.07|             12560.85|        0.0|               1.0|                      0.75|                          1.0|
|18495.55855|              1.0|  5288.28|         3657.3|              1630.98|        0.0|               1.0|                  0.583333|                          1.0|
|16304.88925|              1.0|  1770.57|            0.0|              1770.57| 7424.094447|               0.5|                       0.0|                     0.416667|
|16259.44857|              1.0|  5024.68|         3582.45|              1442.23|  328.698275|               1.0|                  0.833333|                          1.0|
| 16115.5964|              1.0|   684.74|          105.3|               579.44| 4354.002428|               1.0|                  0.083333|                          1.0|
|15532.33972|              1.0|  1168.75|            0.0|              1168.75| 3183.037625|          0.916667|                       0.0|                     0.916667|
| 15258.2259|              1.0|    529.3|          529.3|                 0.0| 4100.891579|               0.5|                       0.5|                          0.0|
|15244.74865|              1.0|  7823.74|        7564.81|               258.93| 2621.049473|               1.0|                       1.0|                          1.0|
|15155.53286|              1.0|   717.24|         717.24|                 0.0| 4718.274895|               1.0|                       1.0|                          0.0|
|14581.45914|              1.0|      0.0|            0.0|                 0.0|  22665.7785|               0.0|                       0.0|                          0.0|
|14411.95798|              1.0|  5958.17|        3161.46|              2796.71|        0.0|               1.0|                  0.583333|                          1.0|
|14224.11541|              1.0|      0.0|            0.0|                 0.0| 4614.427403|               0.0|                       0.0|                          0.0|
| 14100.2511|              1.0|   4995.8|         3403.11|              1592.69| 20712.67008|               1.0|                       1.0|                     0.333333|
|13968.47957|              1.0|   281.71|            8.9|               272.81| 2710.679764|          0.416667|                  0.083333|                     0.333333|
|13777.37772|              1.0|      0.0|            0.0|                 0.0| 1675.249576|               0.0|                       0.0|                          0.0|
|13774.74154|              1.0|   404.24|            0.0|               404.24| 3369.474535|              0.25|                       0.0|                         0.25|
|13763.47358|              1.0|  9670.84|        6701.08|              2969.76| 1896.204934|               1.0|                      0.75|                          1.0|
|13673.07961|              1.0|  9792.23|        3959.81|              5832.42| 2444.445738|               1.0|                      0.75|                          1.0|
|13479.28821|              1.0| 41050.4|       40624.06|               426.34|        0.0|          0.833333|                  0.666667|                     0.416667|
|13318.65912|              1.0|  3504.74|        3266.29|               238.45| 1306.849608|          0.636364|                  0.545455|                     0.272727|
+----------+-----------------+---------+---------------+---------------------+-----------+------------------+--------------------------+-----------------------------+
only showing top 20 rows
```

1.2.4. Hàm Last_value, Last_row

Last_value: giá trị cuối cùng được lưu vào tập dữ liệu. Ví dụ: ngân hàng muốn xem được số dư của khách hàng cuối cùng khi được lưu vào tập dữ liệu:

```
def last_value_formula(data_frame: DataFrame, column_name):

        last_value_result = data_frame.select(last(column_name))

        # Return a dataframe
        return last_value_result
```

```
last_value_formula(_data, "BALANCE").show()
```

```
+-------------+
|last(BALANCE)|
+-------------+
|   372.708075|
+-------------+
```

Last_row: thông tin cuối cùng được lưu vào tập dữ liệu. Ví dụ: Ngân hàng muốn xem thông tin của khách hàng cuối cùng được lưu vào tập dữ liệu:

```
def last_row_formula(data_frame: DataFrame):

        last_row_result = data_frame.withColumn('id', monotonically_increasing_id())\
            .select(max(struct('id', *data_frame.columns))\
            .alias('x')).select(col('x.*')).drop('id')

        # Return a dataframe
        return last_row_result
```

```
last_row_formula(_data).show()
```

```
+----------+-----------------+---------+----------------+----------------------+------------+-------------------+--------------------------+--------------------------------+
|   BALANCE|BALANCE_FREQUENCY|PURCHASES|ONEOFF_PURCHASES|INSTALLMENTS_PURCHASES|CASH_ADVANCE|PURCHASES_FREQUENCY|ONEOFF_PURCHASES_FREQUENCY|PURCHASES_INSTALLMENTS_FREQUENCY|
+----------+-----------------+---------+----------------+----------------------+------------+-------------------+--------------------------+--------------------------------+
|372.708075|         0.666667|  1093.25|         1093.25|                   0.0|  127.040008|           0.666667|                  0.666667|                             0.0|
+----------+-----------------+---------+----------------+----------------------+------------+-------------------+--------------------------+--------------------------------+
```

1.3. Xử lý giá trị Null

Tiếp theo, chúng ta tiến hành kiểm tra các giá trị Null trong DataFrame _data

```
[31] _data.select([F.count( F.when( F.isnan(column) | F.col(column).isNull(), column)).alias(column) for column in _data.columns]).show()
```

```
+-------+-----------------+---------+----------------+----------------------+------------+-------------------+--------------------------+--------------------------------+------------
|BALANCE|BALANCE_FREQUENCY|PURCHASES|ONEOFF_PURCHASES|INSTALLMENTS_PURCHASES|CASH_ADVANCE|PURCHASES_FREQUENCY|ONEOFF_PURCHASES_FREQUENCY|PURCHASES_INSTALLMENTS_FREQUENCY|CASH_ADVANCE_F
+-------+-----------------+---------+----------------+----------------------+------------+-------------------+--------------------------+--------------------------------+------------
|      0|                0|        0|               0|                     0|           0|                  0|                         0|                               0|
+-------+-----------------+---------+----------------+----------------------+------------+-------------------+--------------------------+--------------------------------+------------
```

```
[31] _data.select([F.count( F.when( F.isnan(column) | F.col(column).isNull(), column)).alias(column) for column in _data.columns]).show()
```

```
-----+--------------------------+--------------------------------+----------------------+----------------+-------------+------------+--------+----------------+----------------+------+
QUENCY|ONEOFF_PURCHASES_FREQUENCY|PURCHASES_INSTALLMENTS_FREQUENCY|CASH_ADVANCE_FREQUENCY|CASH_ADVANCE_TRX|PURCHASES_TRX|CREDIT_LIMIT|PAYMENTS|MINIMUM_PAYMENTS|PRC_FULL_PAYMENT|TENURE|
-----+--------------------------+--------------------------------+----------------------+----------------+-------------+------------+--------+----------------+----------------+------+
    0|                         0|                               0|                     0|               0|            0|           1|       0|             313|               0|
-----+--------------------------+--------------------------------+----------------------+----------------+-------------+------------+--------+----------------+----------------+------+
```

Có thể thấy được rằng có 2 cột chứa giá trị Null là cột "CREDIT_LIMIT" với 1 giá trị Null và cột "MINIMUM_PAYMENTS" với 313 giá trị Null.

Tiếp theo ta sẽ tiến hành xử lý các giá trị Null này.

Đối với cột "CREDIT_LIMIT", cột này chỉ chứa 1 giá trị Null (quá ít) nên ta có thể hoàn toàn xóa hàng chứa giá trị đó mà không ảnh hưởng đến DataFrame.

Sau khi xóa xong ta tiến hành kiểm tra lại DataFrame.

```
[32] clear_data_1 = _data.where(_data['CREDIT_LIMIT'].isNotNull())

[33] clear_data_1.select([F.count( F.when( F.isnan(column) | F.col(column).isNull(), column)).alias(column) for column in clear_data_1.columns]).show(truncate=False)
```

```
------+---------------------------+--------------------------------+----------------------+----------------+-------------+------------+--------+----------------+----------------+------+
QUENCY|ONEOFF_PURCHASES_FREQUENCY|PURCHASES_INSTALLMENTS_FREQUENCY|CASH_ADVANCE_FREQUENCY|CASH_ADVANCE_TRX|PURCHASES_TRX|CREDIT_LIMIT|PAYMENTS|MINIMUM_PAYMENTS|PRC_FULL_PAYMENT|TENURE|
------+---------------------------+--------------------------------+----------------------+----------------+-------------+------------+--------+----------------+----------------+------+
      |0                          |0                               |0                     |0               |0            |0           |313     |0               |0               |      |
------+---------------------------+--------------------------------+----------------------+----------------+-------------+------------+--------+----------------+----------------+------+
```

Còn đối với cột "MINIMUM_PAYMENTS", ta sẽ tiến hành thay thế các giá trị Null bằng một giá trị khác sao cho phù hợp với DataFrame _data.

Chúng ta tiến hành xem xem những giá trị nào xuất hiện nhiều nhất trong cột "MINIMUM_PAYMENTS".

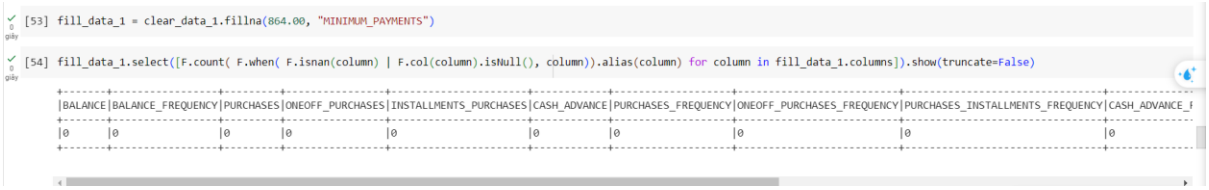
```
clear_data_1.groupBy("MINIMUM_PAYMENTS")\
       .agg(F.count("MINIMUM_PAYMENTS")\
       .alias("num_row"))\
       .sort(F.col("num_row")\
       .desc())\
       .show(clear_data_1.count(),truncate=False)
```

```
+----------------+-------+
|MINIMUM_PAYMENTS|num_row|
+----------------+-------+
|299.351881      |2      |
|128.834658      |1      |
|930.115009      |1      |
|885.376298      |1      |
|1287.324868     |1      |
|152.391012      |1      |
|285.440312      |1      |
|1860.347671     |1      |
|2716.326216     |1      |
|179.244755      |1      |
|1809.974326     |1      |
|886.458108      |1      |
|179.358734      |1      |
|526.749661      |1      |
|218.463565      |1      |
|233.202069      |1      |
|77.604225       |1      |
|292.103091      |1      |
```

Kết quả cho thấy, không có giá trị nào có số lần xuất hiện quá nhiều trong cột "MINIMUM_PAYMENTS". Chính vì thế ta tiến hành thay thế các giá trị Null trong cột thành giá trị trung bình của cột.

Giá trị trung bình của cột "MINIMUM_PAYMENTS" xấp xỉ bằng 864 nên ta dùng hàm fillna của Pyspark để thay thế.

Sau khi thay thế thì tiến hành kiểm tra lại DataFrame.



Vậy là chúng ta đã xong bước xử lý các giá trị Null của DataFrame _data. Giờ đây, _data đã không còn các giá trị Null, sẵn sàng cho việc tính toán tiếp theo.

1.4. Chuẩn hóa dữ liệu

Bởi vì dữ liệu trong DataFrame bao gồm cả số nguyên và số thực và có giá trị không đều nên ta tiến hành chuyển đổi dữ liệu sao cho phù hợp với mục đích tính toán.

Có 2 cách để chuyển đổi dữ liệu:

Cách 1: Dùng Normalization để chuẩn hóa dữ liệu. Tiến hành tìm giá trị nhỏ nhất (min) và giá trị lớn nhất (max) của toàn bộ DataFrame. Chuẩn hóa theo công thức:

Dữ liệu chuẩn hóa = (Dữ liệu gốc – min) / (max – min)

Cách 2: Dùng Standardization để chuẩn hóa dữ liệu. Tiến hành tìm giá trị trung bình (mean) và độ lệch chuẩn (standard_deviation) của toàn bộ DataFrame. Chuẩn hóa theo công thức:

Dữ liệu chuẩn hóa = (Dữ liệu gốc – mean) / standard_deviation

Nhóm đã sử dụng cách 2 để chuẩn hóa dữ liệu, dữ liệu sau khi chuẩn đoán:

```
[93] df_transform = standardize_dataframe(fill_data_1)
     df_transform.show()
```

```
+-------------+----------------------+---------------+---------------------+---------------------------+------------------+------------------------+-------------------------------
|BALANCE_trans|BALANCE_FREQUENCY_trans|PURCHASES_trans|ONEOFF_PURCHASES_trans|INSTALLMENTS_PURCHASES_trans|CASH_ADVANCE_trans|PURCHASES_FREQUENCY_trans|ONEOFF_PURCHASES_FREQUENCY_trans
+-------------+----------------------+---------------+---------------------+---------------------------+------------------+------------------------+-------------------------------
|       -0.732|               -0.2499|        -0.4249|              -0.3569|                    -0.3491|           -0.4668|                 -0.8066|                         -0.678
|       0.7868|                 0.134|        -0.4696|              -0.3569|                    -0.4546|            2.6053|                 -1.2219|                         -0.678
|        0.447|                 0.518|        -0.1077|               0.1088|                    -0.4546|           -0.4668|                  1.2697|                          2.673
|        0.049|               -1.0177|          0.232|               0.5461|                    -0.4546|           -0.3687|                 -1.0142|                           -0.3
|      -0.3588|                 0.518|        -0.4621|              -0.3473|                    -0.4546|           -0.4668|                 -1.0142|                           -0.3
|       0.1178|                 0.518|         0.1544|              -0.3569|                     1.0197|           -0.4668|                  0.4392|                         -0.678
|      -0.4503|                 0.518|         2.8491|               3.5001|                     0.3066|           -0.4668|                  1.2697|                          2.673
|       0.1244|                 0.518|        -0.2654|              -0.3569|                     0.0277|           -0.4668|                  1.2697|                         -0.678
|      -0.2641|                 0.518|        -0.0664|               0.0416|                    -0.2334|           -0.4668|                 -0.3914|                         -0.399
|      -0.6785|               -1.4016|         0.1302|               0.4151|                    -0.4546|           -0.4668|                 -0.8066|                          -0.12
|      -0.1304|                 0.518|        -0.0389|              -0.3569|                     0.5628|           -0.4668|                  1.2697|                         -0.678
|      -0.4486|               -0.2499|         0.2288|                0.542|                    -0.4546|           -0.4668|                  -0.599|                          0.159
|      -0.0229|                 0.518|         1.0365|               1.1493|                     0.3391|           -0.4668|                  1.2697|                          0.159
|      -0.3089|                 0.518|          0.531|              -0.1039|                      1.445|           -0.4668|                  0.6468|                          -0.12
|       0.5804|                 0.518|        -0.4696|              -0.3569|                    -0.4546|           -0.3014|                 -1.2219|                         -0.678
|       2.5565|                 0.518|         0.2847|              -0.3569|                     1.3275|            0.6306|                  0.0239|                         -0.678
|       0.2438|               -0.0099|        -0.4696|              -0.3569|                    -0.4546|            0.8608|                 -1.2219|                         -0.678
|      -0.7319|               -1.7855|        -0.2267|              -0.3569|                     0.1193|           -0.4668|                 -0.1837|                         -0.678
|       0.2039|                 0.518|        -0.2335|              -0.2569|                    -0.0805|           -0.4668|                  0.4392|                         -0.399
|       0.9672|                 0.518|         -0.283|              -0.3569|                    -0.0138|           -0.4668|                  1.2697|                         -0.678
+-------------+----------------------+---------------+---------------------+---------------------------+------------------+------------------------+-------------------------------
only showing top 20 rows
```

Sau đó, tiến hành gắn nhãn cho DataFrame, gắn nhãn cho những khách hàng có tần suất sử dụng thẻ tín dụng > 0.5 với giá trị tương ứng theo 2 cột label và targer.

```
[94] train = df_transform.withColumn("label", (col("BALANCE_FREQUENCY_trans") >= 0.5).cast("string"))
     train = train.withColumn("target", when(col("label") == "false", 1).otherwise(0))

     train.show(10)
```

```
-----+---------------------------+----------------------+-------------------+-----------------+--------------+----------------------+----------------------+------------+-----+------+
trans|CASH_ADVANCE_FREQUENCY_trans|CASH_ADVANCE_TRX_trans|PURCHASES_TRX_trans|CREDIT_LIMIT_trans|PAYMENTS_trans|MINIMUM_PAYMENTS_trans|PRC_FULL_PAYMENT_trans|TENURE_trans|label|target|
-----+---------------------------+----------------------+-------------------+-----------------+--------------+----------------------+----------------------+------------+-----+------+
.7074|                    -0.6753|               -0.4761|            -0.5114|          -0.9603|        -0.529|                -0.311|               -0.5256|      0.3605|false|     1|
0.917|                     0.5739|                  0.11|            -0.5918|           0.6886|        0.8185|                0.0893|                0.2341|      0.3605|false|     1|
0.917|                    -0.6753|               -0.4761|            -0.1091|            0.826|       -0.3838|               -0.1017|               -0.5256|      0.3605| true|     0|
0.917|                    -0.2589|               -0.3295|            -0.5516|            0.826|       -0.5987|               -1.0E-4|               -0.5256|      0.3605|false|     1|
0.917|                    -0.6753|               -0.4761|            -0.5516|          -0.9054|       -0.3644|               -0.2658|               -0.5256|      0.3605| true|     0|
.5506|                    -0.6753|               -0.4761|              -0.27|          -0.7405|       -0.1151|                 0.662|               -0.5256|      0.3605| true|     0|
1.599|                    -0.6753|               -0.4761|             1.9828|           2.4749|        1.5961|               -0.2858|                2.8931|      0.3605| true|     0|
1.599|                    -0.6753|               -0.4761|            -0.1091|          -0.6031|       -0.3641|               -0.1426|               -0.5256|      0.3605| true|     0|
0.288|                    -0.6753|               -0.4761|            -0.3907|           0.6886|        -0.361|                -0.237|               -0.5256|      0.3605| true|     0|
0.917|                    -0.6753|               -0.4761|            -0.4711|           1.7878|       -0.1964|               -0.3278|               -0.5256|      0.3605|false|     1|
-----+---------------------------+----------------------+-------------------+-----------------+--------------+----------------------+----------------------+------------+-----+------+
```

Cuối cùng của bước tiền xử lý dữ liệu là bước phân lớp dữ liệu để chuẩn bị đầu vào cho thuật toán.

Tiến hành tạo ra cột mới có tên là "features" chứa một mảng các giá trị từ các cột đã định nghĩa (loại bỏ cột target, CUST_ID, label).

```
[95] columns_to_exclude = ['target', 'CUST_ID', 'label']
     selected_columns = [col for col in train.columns if col not in columns_to_exclude]

     train_class = train.withColumn("features", array(*selected_columns))

     train_class.select("features", "target").show(5, truncate=False)
```

```
+------------------------------------------------------------------------------------------------------------------------------------------------------+------+
|features                                                                                                                                              |target|
+------------------------------------------------------------------------------------------------------------------------------------------------------+------+
|[-0.732, -0.2499, -0.4249, -0.3569, -0.3491, -0.4668, -0.8066, -0.6787, -0.7074, -0.6753, -0.4761, -0.5114, -0.9603, -0.529, -0.311, -0.5256, 0.3605]|1     |
|[0.7868, 0.134, -0.4696, -0.3569, -0.4546, 2.6053, -1.2219, -0.6787, -0.917, 0.5739, 0.11, -0.5918, 0.6886, 0.8185, 0.0893, 0.2341, 0.3605]           |1     |
|[0.447, 0.518, -0.1077, 0.1088, -0.4546, -0.4668, 1.2697, 2.6731, -0.917, -0.6753, -0.4761, -0.1091, 0.826, -0.3838, -0.1017, -0.5256, 0.3605]        |0     |
|[0.049, -1.0177, 0.232, 0.5461, -0.4546, -0.3687, -1.0142, -0.3994, -0.917, -0.2589, -0.3295, -0.5516, 0.826, -0.5987, -1.0E-4, -0.5256, 0.3605]      |1     |
|[-0.3588, 0.518, -0.4621, -0.3473, -0.4546, -0.4668, -1.0142, -0.3994, -0.917, -0.6753, -0.4761, -0.5516, -0.9054, -0.3644, -0.2658, -0.5256, 0.3605]|0     |
+------------------------------------------------------------------------------------------------------------------------------------------------------+------+
only showing top 5 rows
```

2. Thuật toán khai thác dữ liệu K-Means

2.1. Giới thiệu thuật toán

Thuật toán Kmeans là một thuật toán lặp cố gắng phân vùng tập dữ liệu thành K nhóm con (cụm) riêng biệt không chồng chéo, trong đó mỗi điểm dữ liệu chỉ thuộc về một nhóm. Nó cố gắng làm cho các điểm dữ liệu trong cụm giống nhau nhất có thể đồng thời giữ cho các cụm càng khác nhau càng tốt.

Nó gán các điểm dữ liệu cho một cụm sao cho tổng khoảng cách bình phương giữa các điểm dữ liệu và tâm là nhỏ nhất.

Cách thức hoạt động của thuật toán Kmeans như sau:

i. Chỉ định số cụm K.
ii. Khởi tạo danh sách trọng tâm bằng cách xáo trộn tập dữ liệu và sau đó chọn ngẫu nhiên K điểm dữ liệu.
iii. Tiếp tục lặp lại cho đến khi không có sự thay đổi nào ở tâm. tức là việc gán điểm dữ liệu cho các cụm không thay đổi.
iv. Tính tổng bình phương khoảng cách giữa các điểm dữ liệu và tất cả các tâm.
v. Gán từng điểm dữ liệu vào cụm gần nhất (trung tâm).
vi. Tính toán trọng tâm cho các cụm bằng cách lấy trung bình của tất cả các điểm dữ liệu thuộc mỗi cụm.

2.2. Khai báo hàm cần thiết

Hàm `**euclidean_distance**` để tính khoảng cách Euclidean của 2 đối tượng.

```
def euclidean_distance(arr1, arr2):
    # Convert input arrays to dense vectors
    vec1 = Vectors.dense(arr1)
    vec2 = Vectors.dense(arr2)

    # Calculate the squared Euclidean distance between vectors
    return float(vec1.squared_distance(vec2))

# Register the function as a UDF
euclidean_distance_udf = udf(euclidean_distance, DoubleType())
```

Hàm `**assign_cluster**` để phân chia các điểm dữ liệu vào từng cụm bằng cách tính khoảng cách từ điểm đó đến lần lượt các điểm trung tâm trong cụm. Điểm sẽ thuộc về cụm mả khoảng cách nó tới tâm cụm là nhỏ nhất.

```
def assign_cluster_df(data, centroids):
    # Calculate distances for each centroid
    for i, centroid in enumerate(centroids):
        centroid_features = centroid.select("features").first()["features"]
        distance_column = f"distance_{i}"
        data = data.withColumn(
            distance_column,
            euclidean_distance_udf("features", lit(centroid_features))
        )
    # Create a list of distance columns
    distance_columns = [col(f"distance_{i}") for i in range(len(centroids))]

    # Find the minimum distance among all distance columns
    min_distance_col = least(*distance_columns)

    # Add the 'cluster' column based on the minimum distance
    data = data.withColumn("cluster", min_distance_col)

    # Create a list to hold the conditions for each cluster
    conditions = [col(f"distance_{i}") == min_distance_col for i in range(K)]

    # Use a loop to dynamically create when conditions
    cluster_expr = None
    for i in range(K):
        cluster_expr = when(conditions[i], i) if cluster_expr is None else cluster_expr.when(conditions[i], i)

    # Add the new "cluster" column
    data = data.withColumn("cluster", cluster_expr)

    # Construct the list of columns to select
    selected_columns = ["index", "features", "cluster"] + [f"distance_{i}" for i in range(K)]

    # Select the desired columns
    selected_data = data.select(selected_columns)

    return selected_data
```

Sau khi phân chia các điểm vào cụm, thực hiện tính toán lại các trọng tâm bằng cách tính trung bình khoảng cách giữa các điểm trong cụm. Hàm `**update_centroids**` đảm nhận nhiệm vụ này.

```
def update_centroids(data, centroids):
    centroids_new = [
        data.filter(data.cluster == i)
            .selectExpr(*[f"avg(features[{j}]) as mean_feature_{j + 1}" for j in range(len(data.select("features").first()[0]))])
            .select(array(*[col(f"mean_feature_{j + 1}") for j in range(len(data.select("features").first()[0]))]).alias("features"))
            .first().features
        for i in range(K)
    ]

    for i in range(K):
        centroids[i] = centroids[i].withColumn("features", lit(centroids_new[i]))

    return centroids
```

Hàm `**is_converged**` trả về giả trị True nếu dữ liệu trong các cụm đã hội tụ, nghĩa là khoảng cách giữa các điểm tới tâm cụm luôn bé hơn một giá trị ngưỡng (threshold) cho trước.

```
def is_converged(prev_centroids, centroids, threshold=0.01):
    # false if lengths are not equal
    if len(prev_centroids) != len(centroids):
        return False

    # iterate over each entry of clusters and check if the distance is below the threshold
    for c in range(len(centroids)):
        centroid_features = centroids[c].select("features").first()["features"]
        prev_centroid_features = prev_centroids[c].select("features").first()["features"]
        distance = euclidean_distance(centroid_features, prev_centroid_features)
        if distance > threshold:
            return False

    return True
```

Hàm `**k_means**` bao gồm tất cả quá trình để chạy thuật toán.

```
def k_means(data, k=3, max_iterations=50, threshold=0.01):
    centroids = []
    # initialization, assign random data points as centroids
    count = 0
    hash_map = dict()
    while count < k:
        random_data_idx = np.random.randint(0, n_rows)
        if random_data_idx not in hash_map:
            count += 1
            hash_map[random_data_idx] = True
            centroids.append(train_data.filter(train_data.index == random_data_idx))
    iter_count = 0
    prev_centroids = []

    # run the iterations until not converged or until the max iteration is not reached
    while (is_converged(prev_centroids, centroids, threshold)) == False and (iter_count < max_iterations):
        print("running iteration " + str(iter_count))
        clusters = assign_cluster_df(train_data, centroids)

        # Create a copy of centroids to store as prev_centroids
        prev_centroids = [centroids[i].select("features") for i in range(k)]

        centroids = update_centroids(clusters, centroids)

        iter_count = iter_count + 1

    if (iter_count == max_iterations):
        print("max iterations reached, K means not converged")
    else:
        print("converged")

    return clusters
```

2.3. Triển khai thuật toán trên bộ dữ liệu

```
# Run k-means with the customer data
clusters= k_means(train_class, k=K, threshold=_THRESHOLD)
```

```
running iteration 0
running iteration 1
running iteration 2
running iteration 3
running iteration 4
running iteration 5
running iteration 6
running iteration 7
converged
```

Các cụm đã hội tụ sau 8 vòng lặp.

Bên dưới là số lượng từng khách hàng trong từng cụm.

```
# Print the results
for c in range(K):
    print(f"Cluster {c + 1}: {clusters.select('cluster').filter(clusters.cluster == c).count()} customers")
```

```
Cluster 1: 1335 customers
Cluster 2: 4555 customers
Cluster 3: 3059 customers
```

3. Đánh giá kết quả

3.1. Giới thiệu Silhouette

Silhouette score được sử dụng để đánh giá chất lượng của việc phân cụm trong các thuật toán như Kmeans. Nó cung cấp một số đo lường đối với mỗi điểm dữ liệu, đo lường mức độ "phù hợp" của nó trong cụm mà nó thuộc về. Điều này giúp đánh giá xem cách chia cụm có phù hợp và đồng đều không.

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

Trong đó:

- S(i) là hệ số bóng của điểm dữ liệu i.
- a(i) là khoảng cách trung bình giữa i và tất cả các điểm dữ liệu khác trong cụm mà i thuộc về.
- b(i) là khoảng cách trung bình từ i đến tất cả các cụm mà tôi không thuộc về

Giá trị Silhouette có các đặc điểm:

- Giá trị của hệ số bóng nằm trong khoảng [-1, 1].
- Điểm 1 biểu thị điểm tốt nhất, nghĩa là điểm dữ liệu i rất nhỏ gọn trong cụm mà nó thuộc về và cách xa các cụm khác.
- Giá trị tệ nhất là -1. Các giá trị gần 0 biểu thị các cụm chồng chéo.

3.2. Tiến hành đánh giá

Evaluation Metrics

```
[ ] # Create a VectorAssembler to assemble selected columns into a dense vector
    vec_assembler = VectorAssembler(inputCols=selected_columns, outputCol="features1")

    # Apply VectorAssembler to the DataFrame
    clusters1 = vec_assembler.transform(train)

    clusters1 = clusters1.select("index", "features1")
```

```
[ ] # Perform the join and add a new column to df1
    result = clusters.join(clusters1, clusters["index"] == clusters1["index"], "inner").drop(clusters1["index"])
```

```
[ ] evaluator = ClusteringEvaluator(featuresCol = "features1")

    evaluator.setPredictionCol("cluster")

    evaluator.evaluate(result)

    23/12/20 22:47:42 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
    0.379926851555233
```

Silhouette score cho giá trị xấp xỉ 0.38 cho thấy là kết quả tương đối tích cực. Nó có thể chỉ ra rằng phương pháp phân cụm đã tạo ra các cụm mà các dime dữ liệu bên trong cụm gần trung tâm của cụm hơn so với khoảng cách đến các cụm khác.

So sánh với giá trị Silhouette khi sử dụng thuật toán Kmeans trong thư viện pyspark.ml.clustering.

```
[ ] clusters2 = clusters1.select(col("features1").alias("features"))

    kmeans = KMeans(k=3)

    kmeans.setSeed(1)

    model = kmeans.fit(clusters2)
```

```
[ ] pred = model.transform(clusters2)
```

```
[ ] evaluator = ClusteringEvaluator()

    silhouette = evaluator.evaluate(pred)
    print("Silhouette with squared euclidean distance = " + str(silhouette))

    # Shows the result.
    centers = model.clusterCenters()

    Silhouette with squared euclidean distance = 0.26998417532412766
```

Silhouette score xấp xỉ 0.27.

Nhận xét:

Thuật toán do nhóm phát triển cho khả năng phân cụm tốt hơn so với khi sử dụng K-Means trong thư viện pyspark.ml.clustering

Tuy nhiên, thời gian của thuật toán do nhóm triển khai - 70 giây, dài hơn nhiều so với khi sử dụng thư viện – 15 giây.

CHƯƠNG IV: KẾT LUẬN

1. Ưu điểm

- ***Khả năng tùy chỉnh cao:*** chúng ta có thể dễ dàng tùy chỉnh những tham số, thông số dựa trên những yêu cầu bài toán khác nhau hoặc nhiều dữ liệu đa dạng.
- ***Hiểu sâu hơn về thuật toán:*** việc tự triển khai thuật toán từ đầu giúp hiểu rõ hơn về cách hoạt động của thuận toán K-means.
- ***Tương thích với Pyspark:*** Spark là một framework xử lý dữ liệu lớn. Việc dùng Pyspark đẻ xây dựng thuật toán K-means sẽ tận dụng khả năng xử lý phân tán của Spark.
- ***Đơn giản và hiệu quả:*** K–means là một thuật toán đơn giản và phổ biến trong việc phân cụm dữ liệu và có thể triển khai một cách linh hoạt trên nhiều dự án.
- ***Có cộng đồng hỗ trợ lớn:*** là một trong những thuật toán phân cụm phổ biến, có nhiều hỗ trợ và tài liệu tham khảo, cũng như sẵn có trong nhiều thư viện và framework machine learning.

2. Nhược điểm

- ***Phức tạp trong triển khai:*** để có thể sử dụng thì người dùng phải hiểu rõ được Pyspark và K-means để truyền đúng tham số phục vụ cho nhu cầu của bản thân.
- ***Thời gian và công suất tính toán:*** tự triển khai K- means có thể đòi hỏi nhiều thời gian và công suất tính toán, đặc biệt là khi áp dụng thuật toán trên dữ liệu lớn.
- ***Nhạy cảm với dữ liệu nhiễu:*** K- means có thể nhạy cảm với dữ liệu nhiễu và điểm dữ liệu khác biệt, có thể ảnh hưởng đến quá trình phân cụm.
- ***Không thích hợp cho dữ liệu không phân tán đều:*** nếu các cụm có kích thước không đồng đều hoặc có sự chồng lấn, K- means có thể không đưa ra kết quả tốt.
- ***Yêu cầu biết trước số lượng cụm:*** một số ứng dụng yêu cầu biết trước số lượng cụm, điều này có thể là một hạn chế trong thực tế khi số lượng cụm không rõ ràng.

3. Hướng phát triển

- ***Tối ưu hóa hiệu suất:*** giảm thời gian thực thi và tối ưu tài nguyên khi chạy thuật toán.

- ***Hỗ trợ cho các định dạng dữ liệu đa dạng:*** mở rộng thuật toán để hỗ trợ nhiều định dạng dữ liệu khác nhau, từ dữ liệu số đến dữ liệu không gian đa chiều, để có khả năng ứng dụng rộng rãi.
- ***Phát triển giao diện người dùng:*** xây dựng một giao diện người dùng đơn giản hoặc công cụ trực quan để giúp người dùng tương tác với thuật toán K-Means một cách dễ dàng và trực quan.
- ***Xử lý dữ liệu thời gian thực:*** ứng dụng khả năng xử lý dữ liệu lớn của Spark (bath-processing và streaming processing) đối với dữ liệu quá khứ, near real-time và real-time.
- ***Tích hợp giải pháp đối với dữ liệu nhiễu:*** xử lý hiệu quả với dữ liệu nhiễu bằng cách tích hợp các chiến lược phát hiện và xử lý nhiễu vào quy trình phân cụm.

CHƯƠNG V: TÀI LIỆU THAM KHẢO

[1] ARJUN BHASIN, Credit Card Dataset for Clustering: https://www.kaggle.com/datasets/arjunbhasin2013/ccdata/data?select=CC+GENERAL.csv&fbclid=IwAR3Owqwqu64MDmM7jdhw2pZ3AUcjXPYVfUCH5zT9XsLXUVHnnkZkrT5rXHo (Truy cập ngày 1/12/2023).

[2] Code Generator, Python K-Means Clustering without Libraries: https://codepal.ai/code-generator/query/dwwXcu1g/python-kmeans-clustering-without-libraries?fbclid=IwAR1d41etPmhuVBSnGYvAw_S_Rr1er6JjbipHu1B5WhaWK9S_ah612K4A-sA (Truy cập ngày 3/12/2023).

[3] EZZAHRAOUY HAFSA, Classification & Clustering with pyspark: https://www.kaggle.com/code/hafsaezzahraouy/classification-clustering-with-pyspark?fbclid=IwAR0CarG4LGiPYFKOLJ0uEUJcJJhG2F2hI2OqFTGVlcNbqabjJbtPyLjhjqo#Lest's-create-our-classification-model (Truy cập ngày 3/12/2023).

[4] Apache Spark, Clustering: https://spark.apache.org/docs/latest/ml-clustering.html?fbclid=IwAR2Eo_LGjnIDZ5uLPdkuHZ2P7vbc95FimkrAUCa4HVHPqAg-q5tAfFRxeu0 (Truy cập 4/12/2023).

[5] Sphinx-Gallery, Selecting the number of clusters with silhouette analysis on KMeans clustering: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html?fbclid=IwAR31aZUqcnxEVTYsMWeua_igpK15gYOYt6-TK3f-Lr7MytwnUY-G_6o9XeE (Truy cập 4/12/2023).