

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA HỆ THỐNG THÔNG TIN

BÁO CÁO ĐỒ ÁN

PHÂN CỤM KHÁCH HÀNG VỚI K-MEANS CLUSTERING

Môn học: Dữ liệu lớn

Lớp: IS405.O21

GVHD: ThS. Nguyễn Hồ Duy Trí

Nhóm thực hiện: Nhóm 07

Lê Minh Nguyệt – 21521211

Nguyễn Công Nguyên – 21521200

Hồ Đức Trưởng – 21522730

Nguyễn Phương Tùng – 21520524

TP. Hồ Chí Minh, Tháng 06, 2024

MỤC LỤC

PHẦN I. GIỚI THIỆU TỔNG QUAN	1
1. Lí do chọn đề tài.....	1
2. Ứng dụng của đề tài	1
3. Mục tiêu của đồ án	2
4. Bộ dữ liệu Customer Segmentation	2
4.1. Thông tin chung.....	2
4.2. Chi tiết bộ dữ liệu	2
4.3. Các chỉ số cơ bản.....	3
PHẦN II. HƯỚNG TRIỂN KHAI.....	4
1. Mô tả bài toán	4
2. Các bước thực hiện	4
3. Kỹ thuật, công cụ và phần mềm.....	4
4. Chuẩn bị môi trường	5
PHẦN III. KHÁM PHÁ VÀ XỬ LÝ DỮ LIỆU	7
1. Khám phá dữ liệu	7
1.1. Thêm các thư viện cần thiết	7
1.2. Đọc dữ liệu và các thông số chung.....	7
1.3. Tính toán các bộ chỉ số.....	9
1.4. Các loại biểu đồ.....	10
2. Xử lý dữ liệu	13
2.1. Mã hóa biến phân loại	13
2.2. Chuẩn hóa dữ liệu.....	14
2.3. Xử lý dữ liệu Null.....	16
PHẦN IV. TRIỂN KHAI K-MEANS	20
1. Giới thiệu thuật toán	20
2. Độ đo Silhouette Score	20
3. Triển khai thuật toán trên bộ dữ liệu.....	21
3.1. Khai báo các hàm cần thiết.....	21

3.2. Thực nghiệm.....	24
4. Trực quan hóa quá trình song song hóa K-means.....	27
PHẦN V. KẾT QUẢ VÀ ĐÁNH GIÁ.....	28
1. Kết quả thực nghiệm.....	28
2. So sánh với thư viện PySpark ML.....	29
PHẦN VI. KẾT LUẬN.....	30
1. Tổng kết.....	30
2. Ưu và nhược điểm.....	30
2.1. Ưu điểm.....	30
2.2. Nhược điểm.....	30
3. Hướng phát triển.....	31
TÀI LIỆU THAM KHẢO.....	32

DANH MỤC HÌNH ẢNH

Hình 1. Tập tin \$SPARK_HOME/conf/workers	5
Hình 2. Tập tin \$SPARK_HOME/conf/spark-default.conf.....	6
Hình 3. Tập tin \$SPARK_HOME/conf/spark-env.sh	6
Hình 4. Khởi tạo SparkSession và sparkContext	6
Hình 5. Các worker tại localhost:4040/	6
Hình 6. Import các thư viện cần thiết cho việc xử lý	7
Hình 7. Bộ dữ liệu ban đầu.....	7
Hình 8. Schema của bộ dữ liệu.....	8
Hình 9. Kích thước bộ dữ liệu	8
Hình 10. Thống kê giá trị Null.....	8
Hình 11. Thống kê biến phân loại và liên tục	8
Hình 12. Thống kê chỉ số cơ bản	9
Hình 13. Thống kê median cho từng biến	9
Hình 14. Thống kê variance cho từng biến.....	9
Hình 15. Thống kê mode cho từng biến	10
Hình 16. Thống kê các giá trị ít xuất hiện nhất của từng biến.....	10
Hình 17. Biểu đồ Histogram cho biến liên tục	11
Hình 18. Biểu đồ Barplot cho các biến phân loại.....	12
Hình 19. Biểu đồ Heatmap cho các biến liên tục	13
Hình 21. Các cột cần encoding	14
Hình 22. Kết quả sau khi encoding.....	14
Hình 23. Hàm standard_scaler.....	15
Hình 24. Hàm calculate_mean.....	15
Hình 25. Các cột cần chuẩn hóa	16
Hình 26. Kết quả sau khi chuẩn hóa.....	16
Hình 27. Thống kê các giá trị Null	17
Hình 28. Hàm calculate_mode	17
Hình 29. Điền các giá trị Null.....	18
Hình 30. Kết quả sau khi xử lý Null.....	19
Hình 31. Công thức tính Silhouette trên một điểm dữ liệu	20
Hình 32. Hàm đọc dữ liệu.....	21
Hình 33. Hàm kết hợp dữ liệu thành vector đặc trưng	21
Hình 34. Hàm tính khoảng cách Euclidean	22
Hình 35. Hàm kiểm tra sự hội tụ của K-means	22
Hình 36. Hàm phân cụm K-means	23
Hình 37. Hàm tính điểm Silhouette	24

Hình 38. Tìm K tốt nhất trên bộ dữ liệu	25
Hình 39. Điểm số Silhouette trên các K khác nhau.....	25
Hình 40. Tìm đặc trưng tối ưu để phân cụm.....	26
Hình 41. Kết quả tìm đặc trưng tốt nhất.....	26
Hình 42. Đọc và kết hợp đặc trưng trên 2 worker	27
Hình 43. K-means trên 2 worker	27
Hình 45. Số lượng điểm được phân chia vào mỗi cụm	28
Bảng 1. Silhouette Score trên các giá trị K.....	28
Bảng 2. Silhouette Score trên các đặc trưng.....	28
Hình 46. So sánh kết quả với sklearn	29



DANH MỤC TỪ VIẾT TẮT

Số thứ tự	Ký hiệu từ viết tắt	Từ đầy đủ
1	RDD	Resilient Distributed Dataset
2	HDFS	Hadoop Distributed File System
3	IDE	Integrated Development Environment
4	CSV	Comma-Separated Values
5	EDA	Exploratory Data Analysis



PHẦN I. GIỚI THIỆU TỔNG QUAN

1. Lí do chọn đề tài

Trong bối cảnh cạnh tranh ngày càng gay gắt và khách hàng ngày càng đòi hỏi sự cá nhân hóa, phân cụm khách hàng trở thành một chiến lược quan trọng để các doanh nghiệp duy trì và phát triển lợi thế cạnh tranh của mình. Sự phát triển của công nghệ và dữ liệu lớn đã cung cấp cho các công ty công cụ mạnh mẽ để phân tích và hiểu rõ hơn về khách hàng của họ. Đây là những lý do chính vì sao phân cụm khách hàng là một việc vô cùng cần thiết.

2. Ứng dụng của đề tài

Marketing nhắm đến mục tiêu

Phân cụm khách hàng cho phép các công ty tùy chỉnh các nỗ lực marketing của họ theo các nhóm khách hàng cụ thể. Bằng cách hiểu rõ đặc điểm và nhu cầu của từng phân cụm, các doanh nghiệp có thể tạo ra những chiến dịch marketing nhắm đến mục tiêu rõ ràng và hiệu quả hơn. Sự cá nhân hóa này không chỉ làm tăng cường sự tương tác của khách hàng mà còn thúc đẩy doanh số bán hàng. Ví dụ, một chiến dịch email marketing được cá nhân hóa dựa trên sở thích và hành vi mua sắm của khách hàng sẽ có tỷ lệ mở và chuyển đổi cao hơn so với một chiến dịch gửi chung cho tất cả khách hàng.

Hiểu rõ hơn về khách hàng

Bằng cách phân cụm khách hàng, các công ty có được những hiểu biết sâu sắc hơn về sở thích, hành vi và nhu cầu của họ. Điều này cho phép các doanh nghiệp thiết kế các sản phẩm và dịch vụ phù hợp hơn với mong đợi của khách hàng, từ đó cải thiện sự hài lòng và lòng trung thành của họ. Khi khách hàng cảm thấy được hiểu và được phục vụ theo cách cá nhân hóa, họ có xu hướng quay lại và gắn bó lâu dài với thương hiệu.

Tối ưu hóa nguồn lực

Phân cụm khách hàng giúp các doanh nghiệp phân bổ nguồn lực marketing hiệu quả hơn. Thay vì áp dụng một cách tiếp cận chung, các công ty có thể tập trung nguồn lực vào các phân cụm khách hàng có giá trị cao nhất, dẫn đến tỷ suất hoàn vốn (ROI) tốt hơn. Điều này đặc biệt quan trọng trong bối cảnh ngân sách marketing luôn có giới hạn và các doanh nghiệp cần đảm bảo rằng mỗi đồng chi tiêu đều mang lại kết quả tối ưu. Chẳng hạn, các chương trình khuyến mãi hoặc ưu đãi đặc biệt có thể được thiết kế dành

riêng cho các nhóm khách hàng trung thành hoặc có giá trị cao, từ đó tăng cường mối quan hệ và doanh thu từ các nhóm này.

Như vậy, phân cụm khách hàng không chỉ giúp các doanh nghiệp hiểu rõ hơn về khách hàng mà còn tối ưu hóa các chiến lược marketing và kinh doanh, đảm bảo sự phát triển bền vững và hiệu quả trong dài hạn.

3. Mục tiêu của đồ án

Trong đồ án này, nhóm sẽ thực hiện phân cụm khách hàng trên một bộ dữ liệu sẵn có, bằng thuật toán phân cụm K-means. Việc khai thác dữ liệu từ K-means sẽ được triển khai trong ngữ cảnh song song hóa trên Apache Spark để đáp ứng yêu cầu của đồ án.

4. Bộ dữ liệu Customer Segmentation

4.1. Thông tin chung

Tên bộ dữ liệu: Customer Segmentation

Kích thước: 715.23 kB

Loại tệp sử dụng: .csv

Khả năng sử dụng: 10.00 (theo Kaggle)

Số lượt đánh giá: 154 (31/5/2024)

Link dataset: [Customer Segmentation \(kaggle.com\)](https://www.kaggle.com/datasets/abhishekthakur/customer-segmentation)

4.2. Chi tiết bộ dữ liệu

Bộ dữ liệu Customer Segmentation gồm 9 cột với 10695 dòng dữ liệu, bao gồm:

STT	Tên cột	Mô tả
1	ID	ID của khách hàng
2	Gender	Giới tính của khách hàng
3	Ever_Married	Trạng thái hôn nhân của khách hàng
4	Age	Tuổi của khách hàng
5	Graduated	Khách hàng đã tốt nghiệp (1 là đã tốt nghiệp, 0 là chưa tốt nghiệp)
6	Profession	Nghề nghiệp của khách hàng
7	Work_Experience	Kinh nghiệm làm việc (tính bằng năm)

8	Spending_Score	Điểm chi tiêu của khách hàng
9	Family_Size	Số lượng thành viên trong gia đình

4.3. Các chỉ số cơ bản

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

PHẦN II. HƯỚNG TRIỂN KHAI

1. Mô tả bài toán

Bài toán phân cụm khách hàng (*customer segmentation*) là một kỹ thuật quan trọng trong lĩnh vực tiếp thị và quản lý quan hệ khách hàng. Mục tiêu chính của bài toán này là phân chia khách hàng thành các nhóm (phân khúc) khác nhau dựa trên những đặc điểm và hành vi chung. Việc phân khúc này giúp các doanh nghiệp hiểu rõ hơn về khách hàng của mình, từ đó có thể đưa ra các chiến lược tiếp thị, dịch vụ và sản phẩm phù hợp với từng nhóm khách hàng cụ thể.

Đầu vào của bài toán là các thông tin của khách hàng. Kết quả sau khi khai thác dữ liệu mỗi khách hàng sẽ được phân chia vào một phân khúc phù hợp dựa trên dữ liệu đầu vào của bài toán.

2. Các bước thực hiện

Thu thập dữ liệu: Các thông tin cá nhân khách hàng và hành vi sử dụng sản phẩm/dịch vụ cần được thu thập một cách hiệu quả.

Tiền xử lý dữ liệu: Làm sạch dữ liệu, xử lý các giá trị thiếu (*missing value*) và các giá trị ngoại lệ (*outlier*), chuẩn hóa dữ liệu, v.v.

Lựa chọn tiêu chí phân cụm: Có nhiều yếu tố khác nhau có thể được sử dụng để phân khúc khách hàng, chẳng hạn như thông tin cá nhân (độ tuổi, giới tính, thu nhập, nghề nghiệp, v.v.), hành vi sử dụng sản phẩm/dịch vụ, v.v.

Áp dụng phương pháp phân cụm: Có nhiều thuật toán giúp giải quyết bài toán phân cụm, chẳng hạn như: K-means, Hierarchical Clustering, DBSCAN,... Nhưng trong nghiên cứu này, nhóm sẽ sử dụng thuật toán K-means.

Đánh giá kết quả phân cụm: Silhouette Score là chỉ số được nhóm sử dụng để xác định chất lượng của kết quả phân cụm.

3. Kỹ thuật, công cụ và phần mềm

Đồ án được thực hiện trên môi trường máy ảo Ubuntu, hệ thống lưu trữ là HDFS và framework Apache Spark chạy trên IDE Jupyter Notebook. Cụ thể:

- Ubuntu 22.04
- Apache Hadoop 3.3.6

- Apache Spark 3.5.1
- Python 3.10

Cách cài đặt các công cụ, phần mềm trên sẽ không được đề cập trong báo cáo này.

Nhóm sẽ áp dụng kiến thức từ môn học Dữ liệu lớn để xử lý bài toán đã đề cập.

- Quá trình EDA được thực hiện trên các thư viện của Python như *matplotlib* và *seaborn*.
- Đối với tiền xử lý dữ liệu, nhóm sử dụng *PySpark DataFrame*.
- Phân cụm K-means sẽ được triển khai trên *PySpark RDD*.

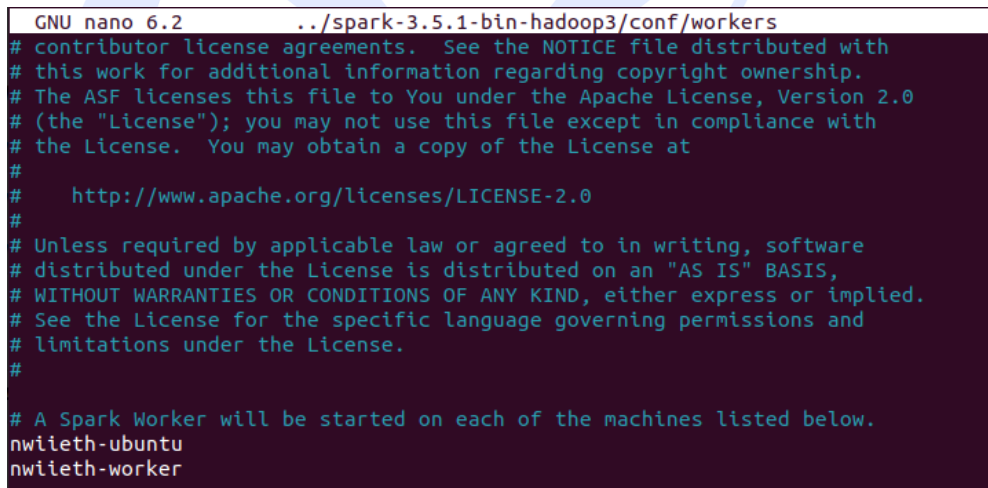
Ngoài ra, còn có sự hỗ trợ của các thư viện *numpy*, *math* cho việc tính toán khoa học và xử lý dữ liệu.

4. Chuẩn bị môi trường

Như đã đề cập trong phần trước, nhóm thực hiện giải quyết bài toán trên framework Apache Spark và Apache Hadoop.

Đối với bộ dữ liệu sau khi tiền xử lý, nhóm tiến hành lưu trữ trong HDFS tại thư mục `customer_segmentation/data/`.

Đối với Apache Spark, nhóm cấu hình 2 worker trong tập tin `$SPARK_HOME/conf/workers` như trong hình dưới đây.



```
GNU nano 6.2      ../spark-3.5.1-bin-hadoop3/conf/workers
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the license for the specific language governing permissions and
# limitations under the License.
#
# A Spark Worker will be started on each of the machines listed below.
nwlieth-ubuntu
nwlieth-worker
```

Hình 1. Tập tin `$SPARK_HOME/conf/workers`

Master được định nghĩa trong tập tin `$SPARK_HOME/conf/spark-default.conf`.

```

GNU nano 6.2 ../spark-3.5.1-bin-hadoop3/conf/spark-defaults.conf
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Default system properties included when running spark-submit.
# This is useful for setting default environmental settings.
#
# Example:
spark.master                spark://localhost:7077
# spark.eventLog.enabled    true
# spark.eventLog.dir        hdfs://namenode:8021/directory
# spark.serializer          org.apache.spark.serializer.KryoSerializer
# spark.driver.memory        5g
# spark.executor.extraJavaOptions -XX:+PrintGCDetails -Dkey=value -Dnumbers="0"

```

Hình 2. Tập tin `$SPARK_HOME/conf/spark-default.conf`

Tập tin `$SPARK_HOME/conf/spark-env.sh` như bên dưới.

```

GNU nano 6.2 ../spark-3.5.1-bin-hadoop3/conf/spark-env.sh
# Options read in any mode
# - SPARK_CONF_DIR, Alternate conf dir. (Default: ${SPARK_HOME}/conf)
# - SPARK_EXECUTOR_CORES, Number of cores for the executors (Default: 1).
# - SPARK_EXECUTOR_MEMORY, Memory per Executor (e.g. 1000M, 2G) (Default: 1G)
# - SPARK_DRIVER_MEMORY, Memory for Driver (e.g. 1000M, 2G) (Default: 1G)
#
# Options read in any cluster manager using HDFS
# - HADOOP_CONF_DIR, to point Spark towards Hadoop configuration files
#
# Options read in YARN client/cluster mode
# - YARN_CONF_DIR, to point Spark towards YARN configuration files when you use
#
# Options for the daemons used in the standalone deploy mode
SPARK_MASTER_HOST=localhost
SPARK_MASTER_PORT=7077
# - SPARK_MASTER_PORT / SPARK_MASTER_WEBUI_PORT, to use non-default ports for the
# - SPARK_MASTER_OPTS, to set config properties only for the master (e.g. "-Dx=>

```

Hình 3. Tập tin `$SPARK_HOME/conf/spark-env.sh`

Trong PySpark, khởi tạo `SparkSession` và `sparkContext` với master đã được chỉ định.

```

spark = SparkSession.builder.master("spark://localhost:7077").appName("customer_segmentation").getOrCreate()
sc = spark.sparkContext

```

Hình 4. Khởi tạo `SparkSession` và `sparkContext`

Tại `localhost:4040/` sẽ thấy có các worker đã được khởi tạo như sau.

Workers (3)

Worker Id	Address	State	Cores	Memory	Resources
worker-20240528200325-192.168.184.131-35823	192.168.184.131:35823	ALIVE	2 (0 Used)	6.7 GiB (0.0 B Used)	
worker-20240528200325-192.168.184.131-36131	192.168.184.131:36131	ALIVE	2 (0 Used)	6.7 GiB (0.0 B Used)	
worker-20240528201454-192.168.184.131-40173	192.168.184.131:40173	ALIVE	2 (0 Used)	6.7 GiB (0.0 B Used)	

Hình 5. Các worker tại `localhost:4040/`

PHẦN III. KHÁM PHÁ VÀ XỬ LÝ DỮ LIỆU

1. Khám phá dữ liệu

1.1. Thêm các thư viện cần thiết

Thêm các thư viện cần thiết

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col,expr,variance
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

Hình 6. Import các thư viện cần thiết cho việc xử lý

1.2. Đọc dữ liệu và các thông số chung

Dữ liệu sau khi được đọc từ tập tin CSV như hình dưới.

```
df.show()
```

ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size
458989	Female	Yes	36	Yes	Engineer	0	Low	1
458994	Male	Yes	37	Yes	Healthcare	8	Average	4
458996	Female	Yes	69	No	NULL	0	Low	1
459000	Male	Yes	59	No	Executive	11	High	2
459001	Female	No	19	No	Marketing	NULL	Low	4
459003	Male	Yes	47	Yes	Doctor	0	High	5
459005	Male	Yes	61	Yes	Doctor	5	Low	3
459008	Female	Yes	47	Yes	Artist	1	Average	3
459013	Male	Yes	50	Yes	Artist	2	Average	4
459014	Male	No	19	No	Healthcare	0	Low	4
459015	Male	No	22	No	Healthcare	0	Low	3
459016	Female	No	22	No	Healthcare	0	Low	6
459024	Male	Yes	50	Yes	Artist	1	Average	5
459026	Male	No	27	No	Healthcare	8	Low	3
459032	Male	No	18	No	Doctor	0	Low	3
459033	Female	Yes	61	Yes	Artist	0	Low	1
459036	Female	Yes	20	Yes	Lawyer	1	Average	3
459039	Male	Yes	45	Yes	Artist	1	Average	2
459041	Male	Yes	55	Yes	Artist	8	Low	1
459045	Female	Yes	88	Yes	Lawyer	1	Average	4

only showing top 20 rows

Hình 7. Bộ dữ liệu ban đầu

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

```
df.printSchema()

root
 |-- ID: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Ever_Married: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Graduated: string (nullable = true)
 |-- Profession: string (nullable = true)
 |-- Work_Experience: integer (nullable = true)
 |-- Spending_Score: string (nullable = true)
 |-- Family_Size: integer (nullable = true)
```

Hình 8. Schema của bộ dữ liệu

Kiểm tra số dòng, số cột của dữ liệu.

```
print("Số dòng của dữ liệu: ",df.count())
print("Số cột của dữ liệu: ",len(df.columns))
```

Số dòng của dữ liệu: 10695
Số cột của dữ liệu: 9

Hình 9. Kích thước bộ dữ liệu

Thống kê số lượng giá trị null của từng cột.

```
null_counts = [(col_name, df.where(col(col_name).isNull()).count()) for col_name in df.columns]

for col_name, count in null_counts:
    print(f"Column '{col_name}' has {count} null values.")

Column 'ID' has 0 null values.
Column 'Gender' has 0 null values.
Column 'Ever_Married' has 190 null values.
Column 'Age' has 0 null values.
Column 'Graduated' has 102 null values.
Column 'Profession' has 162 null values.
Column 'Work_Experience' has 1098 null values.
Column 'Spending_Score' has 0 null values.
Column 'Family_Size' has 448 null values.
```

Hình 10. Thống kê giá trị Null

Phân loại các cột có biến liên tục và biến phân loại.

```
category_columns = [col_name for col_name, data_type in df.dtypes if data_type == "string"]
numeric_columns = [col_name for col_name, data_type in df.dtypes if data_type != "string"]

# Summary cho các cột category
print("Category columns: ",category_columns)
print("Numeric columns: ",numeric_columns)

Category columns: ['Gender', 'Ever_Married', 'Graduated', 'Profession', 'Spending_Score']
Numeric columns: ['ID', 'Age', 'Work_Experience', 'Family_Size']
```

Hình 11. Thống kê biến phân loại và liên tục

1.3. Tính toán các bộ chỉ số

Tiến hành tính toán các chỉ số cơ bản (*count*, *std*, *min*, *max*, *quartile*, *mean*) cho các biến liên tục.

```
df.select(numeric_columns).summary().show()
```

summary	ID	Age	Work_Experience	Family_Size
count	10695	10695	9597	10247
mean	463468.0886395512	43.51182795698925	2.619777013650099	2.8440519176344297
stddev	2600.966410717563	16.77415816252163	3.390789548816187	1.5364271953729591
min	458982	18	0	1
25%	461220	30	0	2
50%	463450	41	1	2
75%	465733	53	4	4
max	467974	89	14	9

Hình 12. Thống kê chỉ số cơ bản

Tính toán chỉ số median cho các biến liên tục.

```
def median_formula(data_frame, column_name):
    median_result = data_frame.agg(expr(f"percentile_approx({column_name}, 0.5, 1000000)").alias("Median"))
    return median_result.collect()[0]["Median"]
for column in numeric_columns:
    median_result = median_formula(df, column)
    print(f"Median value of {column}:", median_result)
```

```
Median value of ID: 463451
Median value of Age: 41
Median value of Work_Experience: 1
Median value of Family_Size: 3
```

Hình 13. Thống kê median cho từng biến

Tính toán chỉ số variance cho các biến liên tục.

```
def variance_formula(data_frame, column_name):
    variance_result = data_frame.select(variance(column_name))
    return variance_result.collect()[0][0]
for column in numeric_columns:
    variance_result = variance_formula(df, column)
    print(f"Variance value of {column}:", variance_result)
```

```
Variance value of ID: 6765026.269681004
Variance value of Age: 281.372382061291
Variance value of Work_Experience: 11.497453764361081
Variance value of Family_Size: 2.360608526681617
```

Hình 14. Thống kê variance cho từng biến

Tính toán chỉ số mode cho các biến phân loại.

```
def mode_formula(data_frame, column_name):
    count_df = data_frame.groupBy(column_name).count()
    mode_result = count_df.orderBy(col("count").desc()).select(column_name).first()[0]
    return mode_result

for column in category_columns:
    mode_result = mode_formula(df, column)
    print(f"Mode value of {column}:", mode_result)
```

Mode value of Gender: Male
 Mode value of Ever_Married: Yes
 Mode value of Graduated: Yes
 Mode value of Profession: Artist
 Mode value of Spending_Score: Low

Hình 15. Thống kê mode cho từng biến

Tìm các giá trị có tần số xuất hiện ít nhất trong các biến phân loại.

```
def rarest_value(data_frame, column_name):
    filtered_df = data_frame.filter(col(column_name).isNotNull())
    count_df = filtered_df.groupBy(column_name).count()
    rarest_value = count_df.orderBy(col("count")).select(column_name).first()[0]
    return rarest_value

for column in category_columns:
    rarest_result = rarest_value(df, column)
    print(f"Rarest value of {column}:", rarest_result)
```

Rarest value of Gender: Female
 Rarest value of Ever_Married: No
 Rarest value of Graduated: No
 Rarest value of Profession: Homemaker
 Rarest value of Spending_Score: High

Hình 16. Thống kê các giá trị ít xuất hiện nhất của từng biến

1.4. Các loại biểu đồ

Vẽ biểu đồ histogram cho các biến liên tục.

```
def plot_histogram(data_frame, column_name):
    # Lọc ra các dòng không chứa giá trị null trong cột cụ thể
    filtered_data = data_frame.filter(data_frame[column_name].isNotNull())

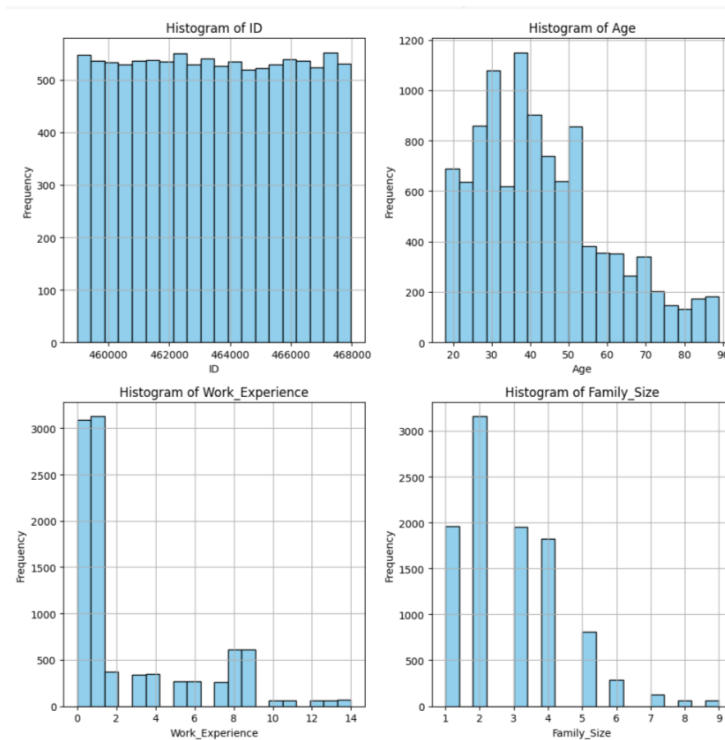
    # Lấy dữ liệu của cột cần vẽ histogram
    column_data = filtered_data.select(column_name).rdd.flatMap(lambda x: x).collect()

    # Vẽ histogram
    plt.hist(column_data, bins=20, color='skyblue', edgecolor='black')
    plt.xlabel(column_name)
    plt.ylabel('Frequency')
    plt.title(f'Histogram of {column_name}')
    plt.grid(True)

plt.figure(figsize=(10, 10)) # Khởi tạo một figure có kích thước 20x20 inches

for i, col in enumerate(numeric_columns):
    plt.subplot(2, 2, i+1) # Tạo subplot trong lưới 2x2, đặt subplot thứ i+1
    plot_histogram(df, col) # Vẽ histogram cho cột numeric thứ i

plt.tight_layout() # Tự động điều chỉnh layout
plt.show() # Hiển thị figure với các subplot
```

Hình 17. Biểu đồ Histogram cho biến liên tục

Vẽ biểu đồ barplot cho các biến phân loại.

```
def plot_barplot(data_frame, column_name):
    # Lọc ra các dòng không chứa giá trị null trong cột cụ thể
    filtered_data = data_frame.filter(data_frame[column_name].isNotNull())
    # Nhóm dữ liệu theo giá trị của cột phân loại và đếm số lần xuất hiện của mỗi giá trị
    category_counts = filtered_data.groupBy(column_name).count().orderBy(column_name).collect()

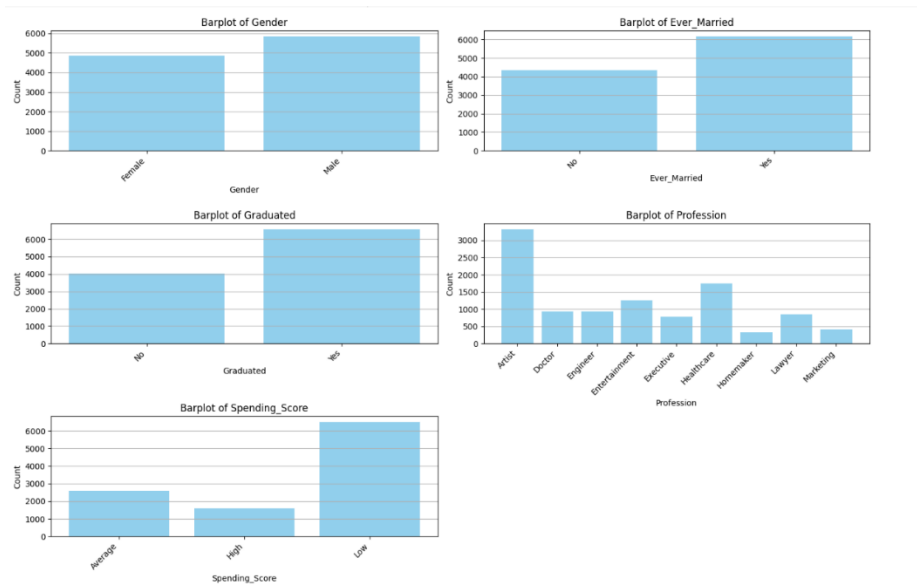
    # Lấy danh sách các giá trị của cột và số lần xuất hiện tương ứng
    categories = [row[column_name] for row in category_counts]
    counts = [row["count"] for row in category_counts]

    # Vẽ biểu đồ barplot
    plt.bar(categories, counts, color='skyblue')
    plt.xlabel(column_name)
    plt.ylabel('Count')
    plt.title(f'Barplot of {column_name}')
    plt.xticks(rotation=45, ha='right')
    plt.grid(axis='y')

plt.figure(figsize=(15, 10)) # Khởi tạo một figure có kích thước 15x10 inches

for i, col in enumerate(categorical_columns):
    plt.subplot(3, 2, i+1) # Tạo subplot trong lưới 3x2, đặt subplot thứ i+1
    plot_barplot(df, col) # Vẽ biểu đồ barplot cho cột phân loại thứ i

plt.tight_layout() # Tự động điều chỉnh layout
plt.show() # Hiển thị figure với các subplot
```



Hình 18. Biểu đồ Barplot cho các biến phân loại

Vẽ biểu đồ heatmap cho correlation matrix của các biến liên tục.

```
# Tính toán ma trận hệ số tương quan giữa các cột
correlation_matrix = []

for col1 in numeric_columns:
    row = []
    for col2 in numeric_columns:
        corr_value = df.stat.corr(col1, col2)
        row.append(corr_value)
    correlation_matrix.append(row)

# Chuyển đổi ma trận hệ số tương quan thành mảng Numpy
correlation_matrix_np = np.array(correlation_matrix)

plt.figure(figsize=(10,10))
# Vẽ biểu đồ nhiệt
sns.heatmap(correlation_matrix_np, annot=True, cmap="coolwarm", xticklabels=numeric_columns, yticklabels=numeric_columns)
plt.title("Heatmap of Correlation Matrix")

# Hiển thị biểu đồ
plt.tight_layout()
plt.show()
```



Hình 19. Biểu đồ Heatmap cho các biến liên tục

2. Xử lý dữ liệu

2.1. Mã hóa biến phân loại

Từ Hình 8, nhận thấy bộ dữ liệu bao gồm cả các biến số và biến phân loại. Để tối ưu hóa cho việc xử lý và mô hình hóa dữ liệu, nhóm đã quyết định mã hóa các biến phân loại thành các giá trị số.

Nhóm bắt đầu tiến hành mã hóa các cột chứa biến phân loại. Các cột cần mã hóa là: `Gender`, `Ever_Married`, `Graduated`, `Profession`, `Spending_Score`.

```
categorical_columns = ["Gender", "Ever_Married", "Graduated", "Profession", "Spending_Score"]
df_train = categorical_encoder(df_train, categorical_columns)
```

Hình 20. Các cột cần encoding

Và đây là kết quả nhóm đã mã hóa thành công các cột chứa biến phân loại thành số nguyên. Nhận thấy kiểu dữ liệu của các cột đều là integer hoặc double, vì vậy, việc mã hóa biến phân loại đã thành công.

df.show()

ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size
462809	0	0	22	0	2	1.0	0	4.0
460666	1	1	49	1	9	14.0	0	1.0
463622	0	0	27	1	2	8.0	0	1.0
467826	0	0	18	0	2	0.0	0	6.0
462040	0	0	19	0	2	5.0	0	4.0
467926	0	1	79	1	1	0.0	2	2.0
467761	0	0	39	1	1	1.0	0	2.0
459076	0	1	69	0	5	0.0	1	2.0
465694	0	1	37	1	2	1.0	1	2.0
459117	0	1	66	1	1	1.0	2	2.0
466312	1	0	21	0	2	0.0	0	3.0
464960	1	0	22	0	2	0.0	0	5.0
459662	0	1	51	1	3	0.0	2	5.0
464808	0	1	42	0	3	1.0	0	4.0
460229	1	1	68	1	6	2.0	2	2.0
463634	1	0	30	1	5	0.0	0	3.0
467877	1	1	46	1	1	0.0	1	2.0
460962	0	1	55	1	7	9.0	2	4.0
459784	1	1	52	1	4	0.0	2	5.0
465783	0	0	23	0	2	1.0	0	5.0

Hình 21. Kết quả sau khi encoding

2.2. Chuẩn hóa dữ liệu

Do nhu cầu tối ưu hóa và đồng nhất dữ liệu, nhóm đã quyết định chuẩn hóa một số cột nhất định trong bộ dữ liệu. Nhóm đã chọn các cột `Age`, `Work_Experience` và `Family_Size` để chuẩn hóa. Và sử dụng phương pháp StandardScaler: Trước tiên, chúng tôi tính toán giá trị trung bình (mean) và độ lệch chuẩn (standard deviation) của toàn bộ DataFrame. Sau đó, chúng tôi thực hiện chuẩn hóa bằng cách áp dụng công thức:

$$\text{Dữ liệu chuẩn hóa} = \frac{\text{Dữ liệu gốc} - \text{mean}}{\text{standard deviation}}$$

Để đơn giản hóa quá trình này, nhóm đã xây dựng thủ công một hàm chuẩn hóa, giúp chuyển đổi các giá trị của các cột phân loại thành giá trị số. Hàm này nhận hai đầu vào là DataFrame và danh sách các cột cần chuẩn hóa. Kết quả đầu ra là chính DataFrame đó với các cột đã được chuyển đổi thành các giá trị số, đảm bảo dữ liệu có cùng phân phối chuẩn, tạo điều kiện thuận lợi cho việc xử lý và mô hình hóa dữ liệu.

```

1 def standard_scaler(df, columns):
2     # Tính toán mean và standard deviation từ df
3     mean_values = {col_name: df.select(mean(col_name)).collect()[0][0] for col_name in columns}
4     std_dev_values = {col_name: df.select(stddev(col_name)).collect()[0][0] for col_name in columns}
5
6     # Chuẩn hóa df
7     for col_name in columns:
8         mean_val = mean_values[col_name]
9         std_dev_val = std_dev_values[col_name]
10        df = df.withColumn(col_name, (df[col_name] - lit(mean_val)) / lit(std_dev_val))
11
12    return df

```

Hình 22. Hàm *standard_scaler*

Hàm `standard_scaler` hoạt động bằng cách tính toán giá trị trung bình (*mean*) và độ lệch chuẩn (*standard deviation*) từ DataFrame huấn luyện (`df_train`) cho từng cột cần chuẩn hóa. Sau đó, hàm chuẩn hóa các cột này trong `df_train` bằng cách trừ đi giá trị trung bình và chia cho độ lệch chuẩn, biến đổi các giá trị thành phân phối chuẩn với mean bằng 0 và standard deviation bằng 1. Tiếp theo, hàm sử dụng các giá trị trung bình và độ lệch chuẩn đã tính từ `df_train` để chuẩn hóa các cột tương ứng trong DataFrame kiểm tra (`df_test`), đảm bảo tính nhất quán giữa dữ liệu huấn luyện và kiểm tra. Cuối cùng, hàm trả về cả hai DataFrame đã chuẩn hóa, sẵn sàng cho các bước phân tích và mô hình hóa tiếp theo.

Điểm đặc biệt trong hàm `standard_scaler` của nhóm là việc áp dụng hàm `calculate_mean`, một hàm đã được xây dựng thủ công trước đó để tính giá trị trung bình của các cột.

```

def calculate_mean(df, column_name):
    mean_value = df.select(mean(column_name)).collect()[0][0]
    return mean_value

```

Hình 23. Hàm *calculate_mean*

Hàm `calculate_mean` được thiết kế để tính toán giá trị trung bình của một cột trong DataFrame của PySpark. Đầu tiên, hàm sử dụng phương thức `select` để chọn cột cần tính giá trị trung bình, sau đó áp dụng hàm `mean` từ thư viện PySpark SQL Functions để thực hiện phép tính. Kết quả của phép tính trung bình được thu thập bằng phương thức `collect()`, giúp chuyển đổi kết quả từ DataFrame sang danh sách các hàng. Từ danh sách này, giá trị trung bình của cột được trích xuất và trả về.

Nhóm bắt đầu tiến hành chuẩn hóa các cột cần thiết như sau:

```

1 # Các cột cần chuẩn hóa
2 columns_to_scale = ['Age', 'Work_Experience', 'Family_Size']

1 # Áp dụng hàm standard_scaler
2 df = standard_scaler(df, columns_to_scale)

```

Hình 24. Các cột cần chuẩn hóa

Đây là kết quả mà nhóm đã đạt được sau khi thực hiện chuẩn hóa dữ liệu:

ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size
462809	0	0	-1.2837627107909002	0	2	-0.4790680244349246	0	0.7505623307375167
460666	1	1	0.32664871242027155	1	9	3.3512099731580194	0	-1.1996403317241193
463622	0	0	-0.9855383731592018	1	2	1.5833893588843526	0	-1.1996403317241193
467826	0	0	-1.522342180896259	0	2	-0.7737047934805356	0	2.050697439045274
462040	0	0	-1.4626973133699193	0	2	0.6994790517475196	0	0.7505623307375167
467926	0	1	2.1159947382104622	1	1	-0.7737047934805356	2	-0.5495727775702406
467761	0	0	-0.2697999628431254	1	1	-0.4790680244349246	0	-0.5495727775702406
459076	0	1	1.5195460629470654	0	5	-0.7737047934805356	1	-0.5495727775702406
465694	0	1	-0.38908969789580483	1	2	-0.4790680244349246	1	-0.5495727775702406
459117	0	1	1.3406114603680463	1	1	-0.4790680244349246	2	-0.5495727775702406
466312	1	0	-1.34340757831724	0	2	-0.7737047934805356	0	0.10049477658363802
464960	1	0	-1.2837627107909002	0	2	-0.7737047934805356	0	1.4006298848913952
459662	0	1	0.4459384474729509	1	3	-0.7737047934805356	2	1.4006298848913952
464808	0	1	-0.09086536026410633	0	3	-0.4790680244349246	0	0.7505623307375167
460229	1	1	1.4599011954207257	1	6	-0.18443125538931351	2	-0.5495727775702406
463634	1	0	-0.8066037705801827	1	5	-0.7737047934805356	0	0.10049477658363802
467877	1	1	0.14771410984125244	1	1	-0.7737047934805356	1	-0.5495727775702406
460962	0	1	0.6845179175783097	1	7	1.8780261279299637	2	0.7505623307375167
459784	1	1	0.5055833149992907	1	4	-0.7737047934805356	2	1.4006298848913952
465783	0	0	-1.2241178432645605	0	2	-0.4790680244349246	0	1.4006298848913952

Hình 25. Kết quả sau khi chuẩn hóa

Nhận thấy các cột được chọn để chuẩn hóa là 'Age', 'Work_Experience', và 'Family_Size' đã có các giá trị thập phân phù hợp với quy trình chuẩn hóa, thể hiện sự biến đổi chính xác theo phân phối chuẩn.

2.3. Xử lý dữ liệu Null

Bước tiền xử lý cuối cùng nhóm thực hiện là xử lý các giá trị Null. Nhóm tiến hành thống kê các giá trị Null:

```
missing_values = []

for col in df_train.columns:
    missing_count = df_train.filter(df_train[col].isNull()).count()
    missing_values.append((col, missing_count))
print("Thông kê giá trị Null trong tập TRAIN:\n" + "-"*20)
for col, missing_count in missing_values:
    if missing_count != 0:
        print(f" - '{col}': {missing_count} giá trị Null.")
```

Thông kê giá trị Null trong tập TRAIN:

```
-----
- 'Ever_Married': 140 giá trị Null.
- 'Graduated': 78 giá trị Null.
- 'Profession': 124 giá trị Null.
- 'Work_Experience': 829 giá trị Null.
- 'Family_Size': 335 giá trị Null.
```

Hình 26. Thông kê các giá trị Null

Nhận thấy có 5 cột chứa giá trị Null là các cột: `Ever_Married`, `Graduated`, `Profession`, `Work_Experience` và `Family_Size`.

Sau khi xác định các giá trị Null, nhóm đã quyết định cẩn thận và linh hoạt về cách xử lý chúng tùy thuộc vào đặc điểm của từng cột dữ liệu. Hai phương pháp chính đã được chọn để điền vào các giá trị Null bao gồm sử dụng giá trị trung bình của cột và sử dụng giá trị xuất hiện nhiều nhất trong cột. Điều này nhằm đảm bảo tính đáng tin cậy và chính xác của dữ liệu sau khi hoàn thiện quá trình xử lý.

Hàm `calculate_mean()`, đã được giới thiệu trước đó, chịu trách nhiệm tính toán giá trị trung bình của cột, đóng vai trò quan trọng trong quyết định điền giá trị cho các ô trống. Đối với hàm còn lại, nó được thiết kế để xác định giá trị xuất hiện nhiều nhất trong cột, một tiêu chí quan trọng khác để xử lý các giá trị Null một cách hiệu quả.

```
def calculate_mode(df, column_name):
    mode_df = (df.groupBy(column_name)
               .count()
               .orderBy("count", ascending=False)
               .first())
    return mode_df[column_name]
```

Hình 27. Hàm calculate_mode

Hàm `calculate_mode` được thiết kế để tìm ra giá trị xuất hiện nhiều nhất trong một cột của DataFrame trong PySpark. Đầu tiên, hàm nhóm các giá trị trong cột lại với nhau và đếm số lần xuất hiện của mỗi giá trị. Kết quả sau đó được sắp xếp theo số lần xuất hiện giảm dần và giá trị xuất hiện nhiều nhất được trích xuất. Cuối cùng, giá trị xuất hiện nhiều nhất này được trả về.

Đối với các cột 'Age', 'Work_Experience', 'Family_Size', chúng tôi sử dụng giá trị trung bình để thay thế giá trị Null vì tính ổn định và tránh ảnh hưởng của outliers. Trong khi đó, đối với các cột 'Gender', 'Ever_Married', 'Graduated', 'Profession', chúng tôi sử dụng giá trị phổ biến nhất để giữ tính nhất quán và đồng nhất của dữ liệu.

```
numeric_columns = ['Gender', 'Ever_Married', 'Graduated', 'Profession']

for column in numeric_columns:
    mode_value = calculate_mode(df_train, column)
    df_train = df_train.na.fill({column: mode_value})

numeric_columns = ['Gender', 'Ever_Married', 'Graduated', 'Profession']

for column in numeric_columns:
    mode_value = calculate_mode(df_train, column)
    df_test = df_test.na.fill({column: mode_value})

numeric_columns = ['Age', 'Work_Experience', 'Family_Size']
for column in numeric_columns:
    mean_value = calculate_mean(df_train, column)
    df_train = df_train.na.fill({column: mean_value})

for column in numeric_columns:
    mean_value = calculate_mean(df_train, column)
    df_test = df_test.na.fill({column: mean_value})
```

Hình 28. Điền các giá trị Null

Đoạn mã trên thực hiện xử lý các giá trị Null trong hai tập dữ liệu 'df_train' và 'df_test'.

Đầu tiên, các cột dạng số (numeric) cần xử lý được xác định trong danh sách `numeric_columns`, bao gồm 'Age', 'Work_Experience', và 'Family_Size'.

Đối với các cột số trong tập dữ liệu huấn luyện ('df_train'), chúng ta tính giá trị trung bình (*mean*) của từng cột sử dụng hàm 'calculate_mean'. Sau đó, các giá trị Null trong từng cột được thay thế bằng giá trị trung bình tương ứng trong 'df_train'.

Tiếp theo, đối với các cột không phải là số trong tập dữ liệu huấn luyện ('df_train'), chúng ta tính giá trị phổ biến nhất (*mode*) của từng cột bằng cách sử dụng hàm 'calculate_mode'. Các giá trị Null trong từng cột sau đó được thay thế bằng giá trị phổ biến nhất tương ứng trong 'df_train'.

Các giá trị Null trong tập dữ liệu kiểm tra ('df_test') được thay thế bằng các giá trị trung bình hoặc giá trị phổ biến nhất của các cột tương ứng trong tập dữ liệu huấn luyện ('df_train'). Điều này đảm bảo tính nhất quán giữa dữ liệu huấn luyện và dữ liệu kiểm tra, giúp mô hình học được từ dữ liệu phản ánh thực tế một cách chính xác và hiệu quả.


```
# Tạo một danh sách chứa các tên cột và số lượng giá trị bị thiếu tương ứng
missing_values = []

# Lặp qua các cột trong DataFrame
for col in df.columns:
    # Tính tổng số giá trị bị thiếu trong cột và thêm vào danh sách
    missing_count = df.filter(df[col].isnull()).count()
    missing_values.append((col, missing_count))

# Hiển thị kết quả
print("Thống kê giá trị Null trong tập DataFrame:\n" + "-"*20)
for col, missing_count in missing_values:
    if missing_count != 0:
        print(f" - '{col}': {missing_count} giá trị Null.")

Thống kê giá trị Null trong tập DataFrame:
-----
```

Hình 29. Kết quả sau khi xử lý Null

Đoạn code này để thống kê các giá trị Null ở mỗi cột nhưng chẳng có cột nào được in ra. Tức nhóm đã thành công xử lý các giá trị Null bằng cách điền vào giá trị phù hợp tương ứng.



PHẦN IV. TRIỂN KHAI K-MEANS

1. Giới thiệu thuật toán

Thuật toán K-means là một phương pháp phân cụm dữ liệu lặp lại, mục tiêu của nó là phân tách tập dữ liệu thành K nhóm con không chồng chéo, sao cho mỗi điểm dữ liệu chỉ thuộc về một nhóm. Thuật toán này cố gắng tối ưu hóa sự tương đồng giữa các điểm dữ liệu trong cùng một nhóm và tăng cường sự khác biệt giữa các nhóm.

Quá trình hoạt động của thuật toán Kmeans có các bước cụ thể như sau:

- Xác định số lượng cụm K.
- Khởi tạo các trọng tâm ban đầu bằng cách chọn ngẫu nhiên K điểm từ tập dữ liệu.
- Lặp lại các bước sau cho đến khi không có sự thay đổi nào ở vị trí của các trọng tâm:
 - Gán mỗi điểm dữ liệu vào cụm có trọng tâm gần nhất (theo khoảng cách Euclidean).
 - Tính toán lại vị trí của trọng tâm cho mỗi cụm bằng cách lấy trung bình của tất cả các điểm dữ liệu trong cụm đó.
- Tính tổng bình phương khoảng cách giữa mỗi điểm dữ liệu và trọng tâm của cụm mà nó thuộc vào, để đánh giá chất lượng của phân cụm.

2. Độ đo Silhouette Score

Silhouette Score được sử dụng để đánh giá chất lượng của việc phân cụm trong các thuật toán như K-means. 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.

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

Hình 30. Công thức tính Silhouette trên một điểm dữ liệu

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 mẫu i thuộc về (intra-cluster distance).
- $b(i)$ là khoảng cách trung bình từ i đến tất cả các điểm trong cụm gần nhất mà mẫu i không thuộc về (nearest-cluster distance)).

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

Điều kiện áp dụng (theo scikit-learn): Số lượng nhãn (n_labels) phải trong khoảng $[2, n_samples - 1]$.

3. Triển khai thuật toán trên bộ dữ liệu

Trong phần triển khai, nhóm lựa chọn sử dụng cấu trúc dữ liệu RDD của Apache Spark để thể hiện rõ ràng hơn sự song song hóa trong quá trình thực hiện thuật toán phân cụm K-means.

3.1. Khai báo các hàm cần thiết

Dữ liệu khi được đọc vào RDD sẽ được loại bỏ header và phân tách thành các phần tử bởi một dấu “,”. Sau đó, thực hiện map ID của điểm dữ liệu (ID của khách hàng) với một hàm kết hợp dữ liệu.

✓ Read data

```
[5] def read_data(path):
    rdd = sc.textFile(path)
    header = rdd.first()
    rdd_split_by_comma = rdd.filter(lambda x: x != header).map(lambda x: x.split(","))
    rdd_id_features = rdd_split_by_comma.map(lambda x: generate_features(x[1:]))
    return rdd_id_features
```

Hình 31. Hàm đọc dữ liệu

Hàm kết hợp dữ liệu nối các giá trị còn lại của dòng thành một mảng numpy.

Generate features values for RDD

```
def generate_features(values):
    res = []
    for v in values:
        res.append(float(v))
    return np.array(res)
```

Hình 32. Hàm kết hợp dữ liệu thành vector đặc trưng

Hàm `euclidean_distance` tính toán khoảng cách Euclidean của 2 mảng numpy (2 điểm dữ liệu). Hàm `get_distance_to_centroids` thực hiện tính khoảng cách Euclidean của một điểm dữ liệu đến trung tâm của tất cả các cụm, trả về một danh sách K phần tử các khoảng cách.

Compute euclidean distance from a point to all centroids

```
def euclidean_distance(point1, point2):
    return np.linalg.norm(point1 - point2)

def get_distance_to_centroids(point, centroids_list):
    res = []
    for c in centroids_list.value:
        distance = euclidean_distance(point, c)
        res.append(distance)
    return res
```

Hình 33. Hàm tính khoảng cách Euclidean

Hàm `has_converged` là một hàm Boolean để kiểm tra thuật toán đã hội tụ hay chưa. Sự hội tụ của K-means trong hàm này được định nghĩa bằng khoảng cách Euclidean giữa các điểm trung tâm cũ và mới của các cụm. Nếu tất cả các điểm trung tâm đều có sự chênh lệch nhỏ hơn hoặc bằng ngưỡng được xác định thì K-means hội tụ.

Check whether kmeans has converged

Converged conditions: The distance between new and current centroids is equal or lower than a threshold.

```
def has_converged(centroids, new_centroids, threshold):
    if len(centroids) != len(new_centroids):
        return False

    for c, nc in zip(centroids, new_centroids):
        distance = euclidean_distance(c, nc)
        if distance > threshold:
            return False
    return True
```

Hình 34. Hàm kiểm tra sự hội tụ của K-means

Hàm `kmeans_clustering` thực hiện thuật toán phân cụm K-means theo các bước đã được đề cập trong phần IV.1. Đầu vào của hàm là một RDD với cặp key – value là ID và vector đặc trưng của điểm dữ liệu. Các bước cụ thể trong đoạn mã như sau:

Bước 1: Các điểm trung tâm được khởi tạo ngẫu nhiên bằng hàm `takeSample` với số lượng cụm bằng `k`.

Bước 2: Thực hiện broadcast các điểm trung tâm đến các Worker Node. Với mỗi phần tử trong RDD đầu vào, tiến hành map vector đặc trưng của điểm dữ liệu với một danh sách khoảng cách với trung tâm của tất cả các cụm.

Bước 3: Gán điểm dữ liệu vào cụm bằng cách lấy index của phần tử nhỏ nhất trong value tại bước 2. Kết quả thu được một RDD với cặp key – value là vector đặc trưng của điểm dữ liệu và tên cụm điểm được gán (tên cụm được đánh số 0, 1, 2, ..., K theo thứ tự lưu trong danh sách khoảng cách).

Bước 4: Thực hiện map lại kết quả của bước 3, với key là tên cụm và value là một tuple chứa (vector đặc trưng của điểm, 1). Tiếp tục `reduceByKey` với hàm cộng trên vector đặc trưng và đếm số lượng điểm thuộc cụm.

Bước 5: Tính toán lại giá trị mới cho trung tâm các cụm bằng map và lấy vector tổng chia cho số lượng điểm (số lượng vector đặc trưng). Thực hiện `collect` kết quả để thu được danh sách các điểm trung tâm mới về một nơi.

Bước 6: Kiểm tra sự hội tụ của K-means với ngưỡng mặc định là 0.01. Nếu K-means chưa hội tụ cập nhật các điểm trung tâm mới và lặp lại từ bước 2. Kết thúc khi số vòng lặp đạt tối đa (100 vòng mặc định).

```
def kmeans_clustering(rdd, k, max_iterations=100, threshold=0.01):

    # Khởi tạo các điểm trung tâm ngẫu nhiên, các điểm được lấy ngẫu nhiên từ các worker khác nhau, kết quả được tập hợp và trả về driver
    centroids = rdd.takeSample(withReplacement=False, num=k)

    # Khởi tạo RDD lưu trữ việc gán nhãn các điểm dữ liệu vào các cụm
    assignments = None

    # Lặp qua các lần lặp
    for it in tqdm(range(max_iterations), desc=f"Clustering with k={k}..."):
        # Các điểm trung tâm được broadcast đến tất cả worker để có thể đọc được
        broadcast_centroids = sc.broadcast(centroids)

        # Tính khoảng cách từ mỗi điểm dữ liệu đến các điểm trung tâm
        # Mỗi dòng dữ liệu được thực hiện map tại worker chứa nó
        distances = rdd.map(lambda point: (point, get_distance_to_centroids(point, broadcast_centroids)))

        # Gán nhãn các điểm dữ liệu vào cụm dựa trên điểm trung tâm gần nhất, thực hiện tại worker chứa dòng dữ liệu
        assignments = distances.map(lambda x: (x[0], np.argmin(x[1])))

        # Tính toán vị trí mới của các điểm trung tâm
        # remapping trả về từng cặp (cluster_id, (mảng chứa tổng từng phần tử của các điểm dữ liệu thuộc cụm, số lượng phần tử thuộc cụm))
        # remapping thực hiện map trên từng worker và reduceByKey để gom dữ liệu từ tất cả các worker về 1 nơi
        remapping = assignments.map(lambda x: (x[1], (x[0], 1))).reduceByKey(
            lambda x, y: (x[0] + y[0], x[1] + y[1])
        )
        # Chia mỗi phần tử trong mảng cho số lượng phần tử thuộc từng cụm để lấy mảng chứa trung bình từng phần tử của các điểm trong cụm
        # Thực hiện với dữ liệu của tất cả các điểm tại 1 nơi duy nhất
        new_centroids = remapping.map(lambda x: x[1][0] / x[1][1]).collect()

        # Kiểm tra xem thuật toán đã hội tụ hay chưa
        if has_converged(centroids, new_centroids, threshold):
            print(f"Clustering has converged after {it + 1} iterations.")
            break

        # Cập nhật các điểm trung tâm
        centroids = new_centroids

    # Trả về vị trí của các điểm trung tâm và việc gán nhãn các điểm dữ liệu vào các cụm
    return centroids, assignments
```

Hình 35. Hàm phân cụm K-means

Hàm `evaluate_squared_euclidean_silhouette` tính điểm Silhouette trên 2 RDD, với đầu vào là `assignment_rdd` chứa (vector đặc trưng của điểm dữ liệu, tên cụm). Khoảng cách được sử dụng để tích Silhouette Score là Centroid Squared Inertia.

```

def evaluate_squared_euclidean_silhouette(assignments_rdd):
    # Tính toán CSI (Centroid Squared Inertia) cho mỗi điểm
    rdd_with_csi = assignments_rdd.map(lambda assignment: (assignment[0], assignment[1], sum([x**2 for x in assignment[0]])))

    # Tính toán các giá trị tổng hợp cho từng cụm
    clusters_aggregate_values = rdd_with_csi.groupBy(lambda x: x[1]).mapValues(
        lambda values: (
            sum(value[0] for value in values), # Tổng vector của các điểm trong cụm
            sum(value[2] for value in values), # Tổng csi của các điểm trong cụm
            len(values)                        # Số lượng điểm trong cụm
        )
    )
    # Chuyển đổi kết quả thành từ điển để truyền tới các worker nodes
    clusters_map = clusters_aggregate_values.collectAsMap()
    broadcasted_clusters_map = sc.broadcast(clusters_map)

    def compute_silhouette(vector, cluster_id, csi):
        # Lấy thông tin của các cụm
        cluster_stats = broadcasted_clusters_map.value

        def compute_csi_diff(csi, point, cluster_stats):
            # Tính toán sự khác biệt CSI giữa điểm và cụm
            y_multiply_point = sum(cluster_stats[0] * point)
            return csi + cluster_stats[1] / cluster_stats[2] - 2 * y_multiply_point / cluster_stats[2]

        # Tính toán b (khoảng cách trung bình đến cụm gần nhất khác)
        min_other = float('inf')
        for c in cluster_stats:
            if c != cluster_id:
                sil = compute_csi_diff(csi, vector, cluster_stats[c])
                if sil < min_other:
                    min_other = sil

        # Tính toán a (khoảng cách trung bình đến các điểm khác trong cùng cụm)
        cluster_current_point = cluster_stats[cluster_id]
        cluster_sil = compute_csi_diff(csi, vector, cluster_current_point) * cluster_current_point[2] / (cluster_current_point[2] - 1)
        # Tính toán hệ số silhouette
        silhouette_coeff = 0.0
        if cluster_sil < min_other:
            silhouette_coeff = 1 - (cluster_sil / min_other)
        elif cluster_sil > min_other:
            silhouette_coeff = (min_other / cluster_sil) - 1
        return silhouette_coeff

    # Tính hệ số silhouette cho mỗi điểm
    rdd_with_silhouette = rdd_with_csi.map(lambda row: (row[0], row[1], row[2], compute_silhouette(row[0], row[1], row[2])))

    # Tính giá trị trung bình của hệ số silhouette
    return rdd_with_silhouette.map(lambda row: row[3]).mean()

```

Hình 36. Hàm tính điểm Silhouette

3.2. Thực nghiệm

Để tìm được K-means với K tốt nhất, nhóm thực hiện thử nghiệm với K nằm trong khoảng [2, 10]. K tốt nhất sẽ được lựa chọn bằng điểm Silhouette. Tuy nhiên, vì bộ dữ liệu tương đối lớn với một hàm chạy thông thường, nhóm thực hiện dừng sớm bằng cách đặt một ngưỡng như sau: Nếu 2 lần liên tiếp điểm số Silhouette giảm xuống nhiều hơn 0.05 so với K trước đó, thực hiện dừng tìm K.

```

1 prev_score = None
2 count = 0
3 scores_on_k = []
4 for k in range (MIN_K, MAX_K):
5     centroids, assignments = kmeans_clustering(rdd_data, k)
6
7     t_start = time.time()
8     score = evaluate_squared_euclidean_silhouette(assignments)
9     t_end = time.time()
10
11     print(f"silhouette score = {score} - time = {t_end - t_start}")
12     print("=====")
13     scores_on_k.append(score)
14     if prev_score is None:
15         prev_score = score
16     elif score <= prev_score and prev_score - score > 0.05:
17         count += 1
18     if count >= 2:
19         break

```

Hình 37. Tìm K tốt nhất trên bộ dữ liệu

```

Clustering with k=2...: 5% ██████████ 5/100 [00:08<02:12, 1.39s/it]
Clustering has converged after 6 iterations.
silhouette score = 0.7066330036335957 - time = 2.1034257411956787
=====
Clustering with k=3...: 100% ██████████ 100/100 [01:49<00:00, 1.06s/it]
silhouette score = 0.5189459492276806 - time = 2.094111204147339
=====
Clustering with k=4...: 100% ██████████ 100/100 [01:49<00:00, 1.16s/it]
silhouette score = 0.4202850440436096 - time = 2.7958850860595703
=====

```

Hình 38. Điểm số Silhouette trên các K khác nhau

Kết quả cho thấy K = 2 cho kết quả tốt nhất, và số cụm từ 3 trở đi, Silhouette Score giảm xuống đáng kể. Có thể thấy khi đó, việc phân cụm không còn hoạt động tốt.

Với K = 2, nhóm tiến hành lựa chọn đặc trưng để kết quả phân cụm được tối ưu nhất. Xem xét từ phần III.1, nhóm thử nghiệm loại bỏ lần lượt các đặc trưng có nhiều giá trị thiếu nhất trong bộ dữ liệu.

Các trường hợp thử nghiệm:

- Trường hợp 0: Loại bỏ đặc trưng `Work_Experience`.
- Trường hợp 1: Loại bỏ thêm đặc trưng `Family_Size`.
- Trường hợp 2: Loại bỏ thêm đặc trưng `Ever_Married`.
- Trường hợp 3: Loại bỏ thêm đặc trưng `Profession`.

```

1 features_to_drop = [
2     [1], # drop work_experience    CASE 0
3     [1, 2], # drop work_experience, family_size    CASE 1
4     [1, 2, 4], # drop work_experience, family_size, ever_married    CASE 2
5     [1, 2, 4, 6] # drop work_experience, family_size, ever_married, profession    CASE 3
6 ]
7 best_score = max(scores_on_k)
8
9 best_case = 0
10 idx = 0
11
12 for case in tqdm(features_to_drop):
13     print(f"Computing on case {idx} ...")
14     rdd_data_drop = rdd_data.map(lambda x: np.array([x[i] for i in range(len(x)) if i not in case]))
15     centroids, assignments = kmeans_clustering(rdd_data_drop, k=2)
16
17     t_start = time.time()
18     score = evaluate_squared_euclidean_silhouette(assignments)
19     t_end = time.time()
20
21     if score > best_score:
22         best_case = idx
23         best_score = score
24
25     print(f"Case {idx}: silhouette score = {score} - time = {t_end - t_start}")
26     print("*****")
27     print()
28     idx += 1
29
30 print("=====")
31 print("=====")
32 print(f"Best Silhouette Score is {best_score} for case {best_case}")

```

Hình 39. Tìm đặc trưng tối ưu để phân cụm

```

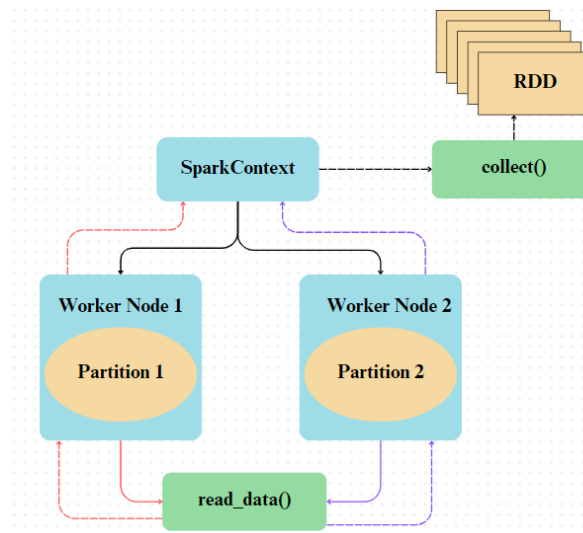
=====
=====
Best Silhouette Score is 0.7884810063536986 for case 2

```

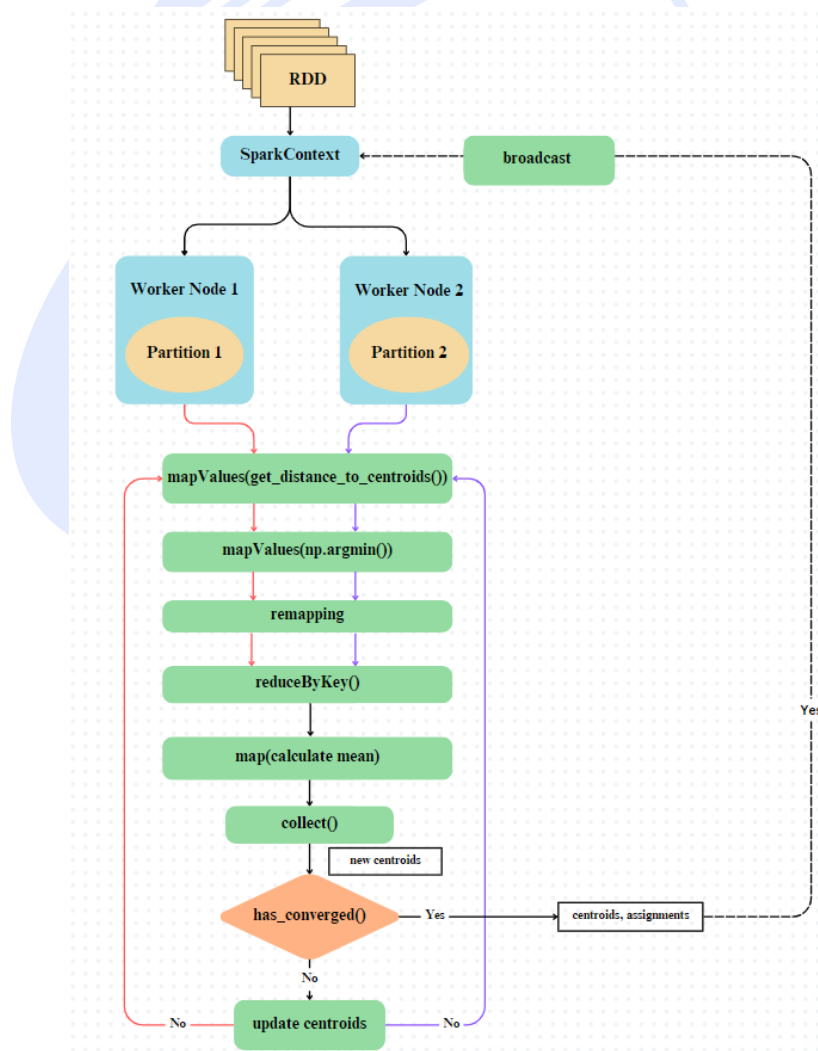
Hình 40. Kết quả tìm đặc trưng tốt nhất

Kết quả cho thấy trường hợp 2 cho kết quả tốt nhất. Vậy, các đặc trưng tối ưu cho K-means ($K = 2$) là { `Age`, `Gender`, `Graduated`, `Profession`, `Spending_Score` }.

4. Trực quan hóa quá trình song song hóa K-means



Hình 41. Đọc và kết hợp đặc trưng trên 2 worker



Hình 42. K-means trên 2 worker

PHẦN V. KẾT QUẢ VÀ ĐÁNH GIÁ

1. Kết quả thực nghiệm

```

rdd1 = sc.textFile("./segmentation_result_final/part-00000")
header = rdd1.first()
rdd2 = rdd1.filter(lambda x: x != header).map(lambda x: x[1:-1].split(",")).map(lambda x: (x[1], x[0])).countByKey()
rdd2
defaultdict(int, {'0': 4004, '1': 6689})

```

Hình 43. Số lượng điểm được phân chia vào mỗi cụm

K	Silhouette Score	Converged (Iterations)
2	0.7066	6
3	0.5189	10
4	0.4202	x

Bảng 1. Silhouette Score trên các giá trị K

Nhận xét: Điểm số Silhouette trong trường hợp K = 2 là tốt nhất với **0.7066** và hội tụ tương đối nhanh (chỉ sau 6 vòng lặp). Trong khi đó, K = 4 không thể hội tụ trong 100 vòng lặp, điểm số cũng rất thấp, chứng tỏ việc phân cụm đã diễn ra không hiệu quả.

Đặc trưng	Silhouette Score	Converged (Iterations)
Toàn bộ 8 đặc trưng	0.7066	6
Loại bỏ đặc trưng `Work_Experience`.	0.7387	4
Loại bỏ thêm đặc trưng `Family_Size`.	0.7767	4
Loại bỏ thêm đặc trưng `Ever_Married`.	0.7885	3
Loại bỏ thêm đặc trưng `Profession`.	0.5171	10

Bảng 2. Silhouette Score trên các đặc trưng

Nhận xét: Điểm số Silhouette trong trường hợp loại bỏ 3 đặc trưng {'Work_Experience', 'Family_Size', 'Ever_Married'} là tốt nhất với **0.7885**. Nhìn chung, việc loại bỏ bất cứ đặc trưng nào trong số các đặc trưng bị thiếu nhiều dữ liệu đều mang lại kết quả cải thiện so với trường hợp dùng toàn bộ đặc trưng. Tốc độ hội tụ giữa các trường hợp cũng tương đối ổn định.

Kết luận: Với Silhouette Score tốt nhất là 0.7885, có thể nói thuật toán đã phân cụm được các điểm dữ liệu tương đối tốt. Nó đã có thể phân tách được các điểm dữ liệu thành 2 cụm dựa vào độ phù hợp của bộ dữ liệu.

2. So sánh với thư viện PySpark ML

```

1 from pyspark.sql import SparkSession
2 from pyspark.ml.evaluation import ClusteringEvaluator
3 from pyspark.ml.linalg import Vectors, VectorUDT
4 from pyspark.ml.clustering import KMeans
5 from pyspark.sql import Row
6 from pyspark.sql.types import StructType, StructField

11 # Giả sử best_assignments đã được định nghĩa
12 # best_assignments = ...
13
14 # Chuyển đổi assignments_rdd thành DataFrame với kiểu dữ liệu đúng
15 rdd_with_index = best_assignments.map(lambda x: Row(features=Vectors.dense(x[0])))
16
17 # Định nghĩa schema rõ ràng cho DataFrame
18 schema = StructType([
19     StructField("features", VectorUDT(), False)
20 ])
21
22 df = spark.createDataFrame(rdd_with_index, schema)

kmeans = KMeans(k=2, maxIter= 100, tol= 0.01, seed=1, featuresCol='features', predictionCol='prediction')
model = kmeans.fit(df)

# Dự đoán kết quả phân cụm
predictions = model.transform(df)

# Tạo ClusteringEvaluator
evaluator = ClusteringEvaluator(featuresCol='features', predictionCol='prediction', metricName='silhouette', distanceMeasure='squaredEuclidean')

# Tính toán chỉ số Silhouette
silhouette_score = evaluator.evaluate(predictions)

print(f"Silhouette Score: {silhouette_score}")

Silhouette Score: 0.7884810063537133

```

Hình 44. So sánh kết quả với sklearn

Khi sử dụng `pyspark.ml.clustering.KMeans` với các tham số tương tự như hàm do nhóm thực hiện (số lượng cụm bằng 2, số vòng lặp tối đa là 100) và tính toán Silhouette Score bằng `pyspark.ml.evaluation.ClusteringEvaluator`, thu được kết quả là **0.7885**.

Có thể thấy hàm tính điểm Silhouette của nhóm đã hoạt động rất tốt khi cho ra kết quả giống hoàn toàn với kết quả của PySpark ML. Một điểm đáng chú ý là thời gian thực thi của hàm tính toán độ đo Silhouette Score của nhóm rất ngắn, gần như ngang ngửa với hàm đánh giá của thư viện PySpark ML, chứng tỏ thuật toán của nhóm đã hoạt động rất hiệu quả.

PHẦN VI. KẾT LUẬN

1. Tổng kết

Nghiên cứu Phân cụm Khách hàng sử dụng thuật toán K-means kết hợp với các kỹ thuật xử lý Dữ liệu lớn mà không sử dụng các thư viện hỗ trợ có sẵn đã thành công trong việc phân chia khách hàng thành các nhóm có đặc điểm đồng nhất, xây dựng cơ sở dữ liệu chi tiết về thông tin khách hàng và đề xuất các giải pháp tối ưu hóa dịch vụ và chính sách cho từng nhóm khách hàng.

Nghiên cứu giúp hiểu sâu hơn về thuật toán, việc tự triển khai thuật toán từ đầu giúp hiểu rõ hơn về cách hoạt động của thuật toán K-means.

Kết quả của đề tài này có thể giúp cho các ngân hàng nâng cao hiệu quả hoạt động kinh doanh, tăng cường khả năng cạnh tranh và mang lại lợi ích cho khách hàng.

2. Ưu và nhược điểm

2.1. Ưu điểm

Khả năng tùy chỉnh cao

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.

Tương thích với PySpark

Spark là một framework xử lý dữ liệu lớn. Việc dùng PySpark để 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.

2.2. Nhược điểm

Phức tạp trong triển khai

Yêu cầu phải hiểu rõ được PySpark và K-means để có thể tự triển khai mà không cần sử dụng thư viện có sẵn.

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.

Tài nguyên và thời gian không tối ưu

Việc sử dụng tác vụ MapReduce trên RDD trong PySpark tốn rất nhiều thời gian để triển khai, tài nguyên tính toán cũng đòi hỏi khá nhiều để có thể xử lý các công việc lớn.

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 (batch-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.

TÀI LIỆU THAM KHẢO

[1] Bowen Wang, Jun Yin, Qi Hua, Zhiang Wu, Jie Cao. *Parallelizing K-Means-Based clustering on Spark*. (2016, August 1). IEEE Conference Publication | IEEE Xplore. <https://ieeexplore.ieee.org/document/7815182/>

[2] Mohamed-Said-Ibrahim. (n.d.-b). *Parallel_K-Means_Using_Spark: An implementation of a parallel version of K-means clustering algorithm using the Spark map-reduce framework*. GitHub. https://github.com/mohamed-said-ibrahem/Parallel_K-Means_Using_Spark

[3] *Customer segmentation*. (2020, August 28). Kaggle. <https://www.kaggle.com/datasets/vetrirah/customer/>

[4] Trituenhantao.Io. (2024, February 11). *Bài 4: K-Means Clustering*. Trí Tuệ Nhân Tạo. <https://trituenhantao.io/machine-learning-co-ban/bai-4-k-means-clustering/>

[5] *silhouette_score*. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

[6] *KMEANS*. (n.d.). Scikit-learn. <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>