



BÁO CÁO ĐỒ ÁN

# PHÂN CỤM KHÁCH HÀNG VỚI K-MEANS CLUSTERING

NHÓM 07

**Môn học:** Dữ liệu lớn

**Lớp:** IS405.021

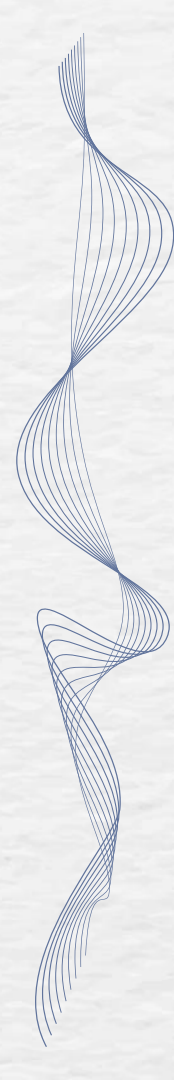

**GVHD:** ThS. Nguyễn Hồ Duy Trí

Lê Minh Nguyệt - 21521211

Nguyễn Công Nguyên - 21521200

Hồ Đức Trưởng - 21522730

Nguyễn Phương Tùng - 21520524





# NỘI DUNG

**01**

GIỚI THIỆU TỔNG QUAN

**02**

HƯỚNG TRIỂN KHAI

**03**

KHÁM PHÁ VÀ XỬ LÝ  
DỮ LIỆU

**04**

TRIỂN KHAI K-MEANS

**05**

KẾT QUẢ VÀ ĐÁNH GIÁ

**06**

KẾT LUẬN



01

# GIỚI THIỆU TỔNG QUAN



# 1.1 Lý do chọn đề tài

Trong bối cảnh cạnh tranh gay gắt và khách hàng đòi hỏi cá nhân hóa, phân đoạn khách hàng là chiến lược quan trọng để doanh nghiệp duy trì lợi thế. Công nghệ và dữ liệu lớn hỗ trợ phân tích khách hàng hiệu quả.

- **Marketing nhắm mục tiêu:** Tùy chỉnh chiến dịch marketing theo nhóm cụ thể, tăng tương tác và doanh số. Ví dụ, email marketing cá nhân hóa có tỷ lệ mở và chuyển đổi cao hơn.
- **Hiểu khách hàng hơn:** Cung cấp hiểu biết sâu sắc về sở thích và nhu cầu, giúp thiết kế sản phẩm và dịch vụ phù hợp, nâng cao sự hài lòng và trung thành.
- **Tối ưu nguồn lực:** Phân bổ nguồn lực hiệu quả, tập trung vào khách hàng giá trị cao, tăng ROI. Chương trình khuyến mãi cho khách hàng trung thành hoặc có giá trị cao tăng doanh thu.





## 1.2 Bộ dữ liệu Customer Segmentation

**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 \(kaggle\)](#)

Bộ dữ liệu Customer Segmentation chứa gần 10,695 dòng và 9 cột thông tin, cung cấp dữ liệu chi tiết về khách hàng.

STT	Tên cột	Mô tả
0	<b>ID</b>	Mã định danh khách hàng
1	<b>Gender</b>	Giới tính
2	<b>Ever_Married</b>	Tình trạng hôn nhân
3	<b>Age</b>	Tuổi

STT	Tên cột	Mô tả
4	<b>Graduated</b>	Trình độ học vấn
5	<b>Profession</b>	Nghề nghiệp
6	<b>Work_Experience</b>	Kinh nghiệm công việc
7	<b>Spending_Score</b>	Điểm chi tiêu
8	<b>Family_Size</b>	Quy mô gia đình



# NỘI DUNG

**01**

GIỚI THIỆU TỔNG QUAN

**02**

HƯỚNG TRIỂN KHAI

**03**

KHÁM PHÁ VÀ XỬ LÝ  
DỮ LIỆU

**04**

TRIỂN KHAI K-MEANS

**05**

KẾT QUẢ VÀ ĐÁNH GIÁ

**06**

KẾT LUẬN



02

# HƯỚNG TRIỂN KHAI



## 2.1 Mô tả bài toán

Mục tiêu chính là phân chia khách hàng thành các nhóm khác nhau dựa trên đặc điểm và hành vi chung. Đây là bài toán học không giám sát.

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.



Giới tính: Nữ

Tuổi: 20

...

**Cụm 1**



Giới tính: Nam

Tuổi: 30

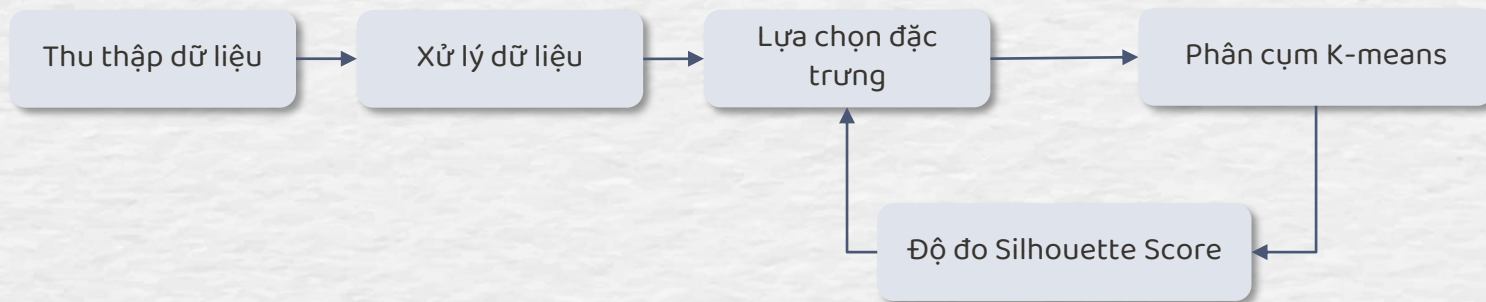
...

**Cụm 2**



## 2.2 Các bước thực hiện

- **Thu thập dữ liệu:** Dữ liệu về hành vi sử dụng sản phẩm/dịch vụ, thông tin cá nhân khách hàng, v.v.
- **Tiền xử lý dữ liệu:** Làm sạch, xử lý missing value và outlier, chuẩn hóa, v.v.
- **Lựa chọn tiêu chí phân đoạn:** Có thể sử dụng thông tin nhân khẩu học hoặc hành vi sử dụng sản phẩm/dịch vụ.
- **Phương pháp phân cụm:** Áp dụng các phương pháp phân cụm K-means.
- **Đánh giá:** đánh giá kết quả phân đoạn bằng Silhouette Score để xác định chất lượng của kết quả phân cụm.



## 2.3 Kỹ thuật, công cụ và phần mềm

- Ubuntu 22.04
- Apache Hadoop 3.3.6
- Apache Spark 3.5.1
- Python 3.10

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.

- 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*.

Các thư viện khác: *numpy*, *math*.





# NỘI DUNG

**01**

GIỚI THIỆU TỔNG QUAN

**02**

HƯỚNG TRIỂN KHAI

**03**

KHÁM PHÁ VÀ XỬ LÝ  
DỮ LIỆU

**04**

TRIỂN KHAI K-MEANS

**05**

KẾT QUẢ VÀ ĐÁNH GIÁ

**06**

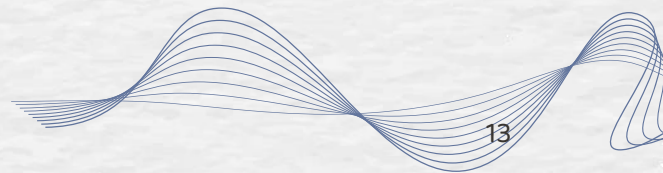
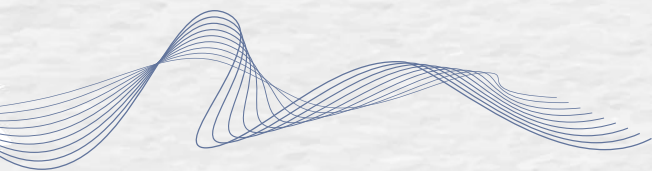
KẾT LUẬN

03

# KHÁM PHÁ VÀ XỬ LÝ DỮ LIỆU



## 3.1 Khám phá dữ liệu



## 3.1.1 Đọc dữ liệu và các thông số chung

Dữ liệu ban đầu sau khi được đọc từ file CSV.

Cấu trúc của dữ liệu.

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

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)
```

### 3.1.1 Đọc dữ liệu và các thông số chung

Thống kê số lượng giá trị Null trong mỗi cột.

STT	Tên cột	Null Count
0	ID	0
1	Gender	0
2	Ever_Married	190
3	Age	0
4	Graduated	102
5	Profession	162
6	Work_Experience	1098
7	Spending_Score	0
8	Family_Size	448

Thống kê biến phân loại và liên tục.

STT	Tên biến	Loại biến
0	ID	Liên tục
3	Age	
6	Work_Experience	
8	Family_Size	
1	Gender	Phân loại
2	Ever_Married	
4	Graduated	
5	Profession	
7	Spending_Score	

### 3.1.2 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.

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



### 3.1.2 Tính toán các bộ chỉ số

Tiến hành tính toán các chỉ số khác (median, variance) cho các biến liên tục.

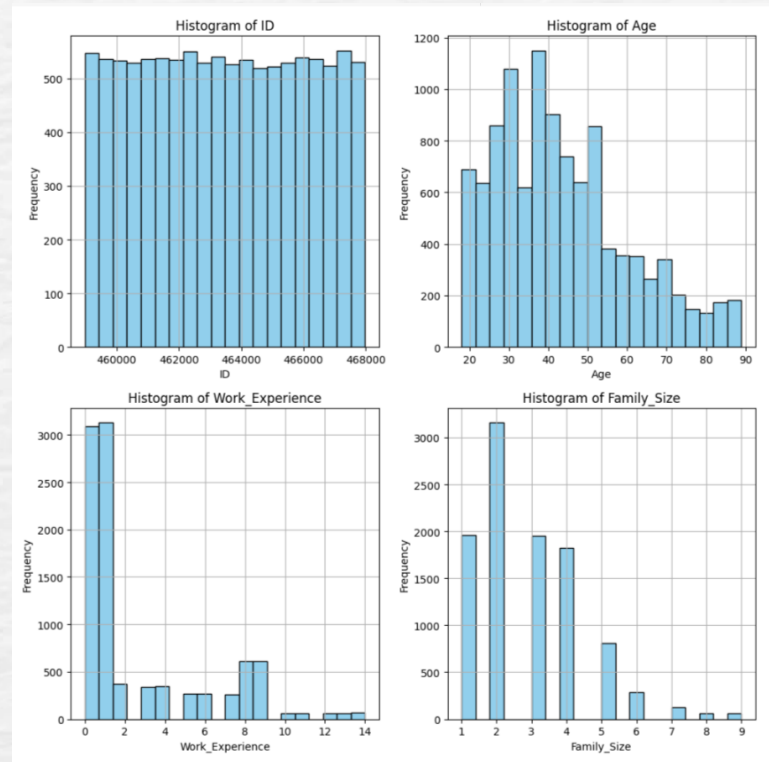
	ID	Age	Work_Experience	Family_Size
<b>median</b>	463451	41	1	3
<b>variance</b>	6765026.269681004	281.372382061291	11.497453764361081	2.360608526681617

Tính toán chỉ số mode và các giá trị có tần số xuất hiện ít nhất cho các biến phân loại.

	Gender	Ever_Married	Graduated	Profession	Spending_Score
<b>mode</b>	Male	Yes	Yes	Artist	Low
<b>rarest</b>	Female	No	No	Homemaker	High

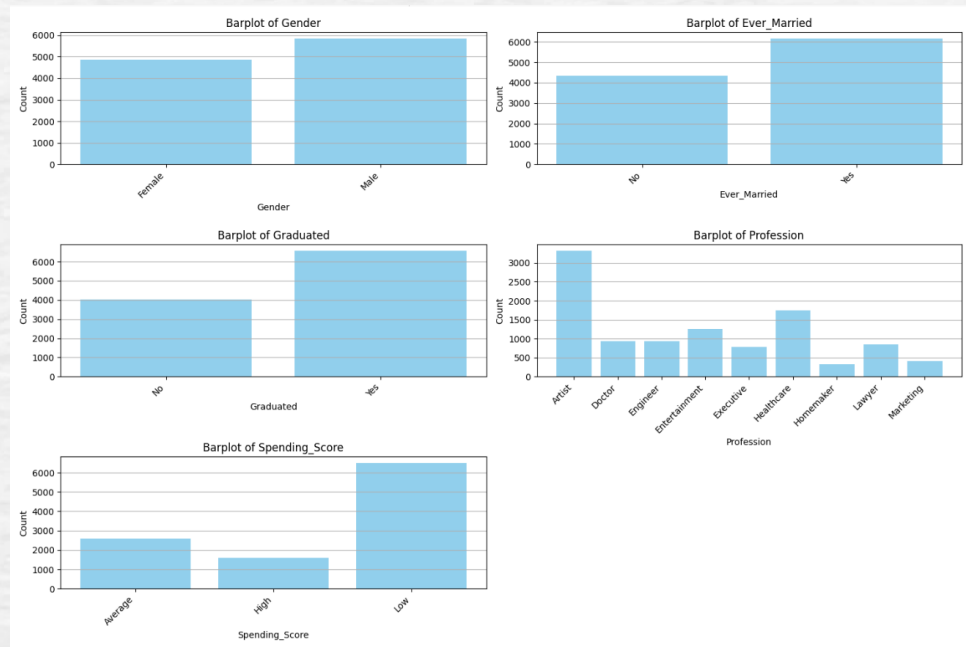
### 3.1.3 Các loại biểu đồ

Biểu đồ histogram cho các biến liên tục.



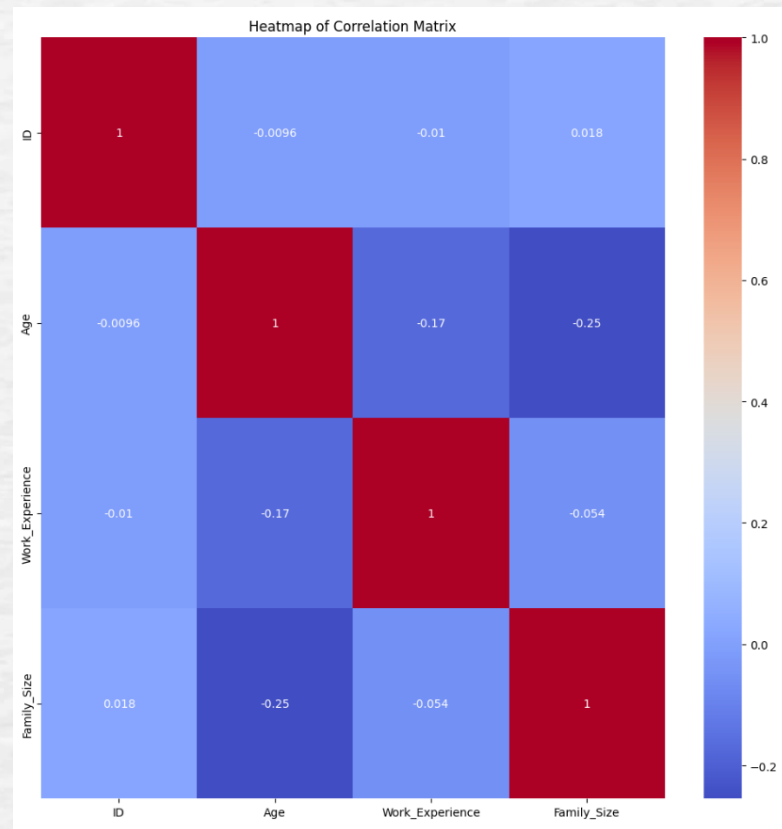
### 3.1.3 Các loại biểu đồ

Biểu đồ barplot cho các biến phân loại.



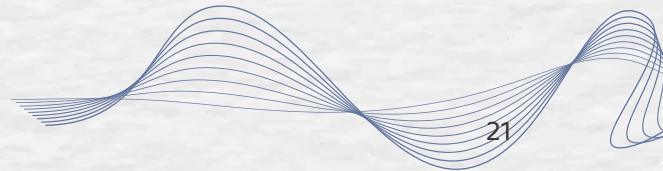
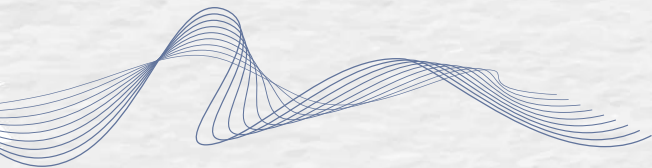
### 3.1.3 Các loại biểu đồ

Biểu đồ heatmap cho correlation matrix của các biến liên tục.





## 3.2 Xử lý dữ liệu



## 3.2.1 Mã hóa biến phân loại

- Bộ dữ liệu chứa cả biến số và biến phân loại. Biến phân loại cần được mã hóa thành giá trị số để xử lý và mô hình hóa hiệu quả.
- Có 5 cột chứa biến phân loại: "Gender", "Ever\_Married", "Graduated", "Profession", "Spending\_Score".
- Kết quả sau khi mã hóa biến phân loại như sau:

**Nhận xét:** Kết quả là các cột chứa biến phân loại đã được thành công mã hóa thành số nguyên. Các kiểu dữ liệu của các cột đều là integer hoặc double.

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

### 3.2.2 Chuẩn hóa dữ liệu

- Do dữ liệu trong một số cột của DataFrame chứa các giá trị không đồng đều, nhóm đã quyết định chuẩn hóa các cột "Age", "Work\_Experience", và "Family\_Size" để tối ưu hóa và đồng nhất dữ liệu.
- Sử dụng phương pháp StandardScaler, quá trình chuẩn hóa được thực hiện theo công thức sau đây:

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

- Việc chuẩn hóa này giúp đảm bảo dữ liệu có cùng phân phối chuẩn, tạo điều kiện thuận lợi cho việc xử lý và mô hình hóa sau này.

## 3.2.2 Chuẩn hóa dữ liệu

Kết quả sau khi thực hiện chuẩn hóa như sau:

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

**Nhận xét:** 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.



### 3.2.3 Xử lý dữ liệu Null

- Nhóm đã thực hiện thống kê và nhận thấy có 5 cột chứa giá trị Null là "Ever\_Married", "Graduated", "Profession", "Work\_Experience", và "Family\_Size".
- Nhóm lựa chọn xử lý các giá trị Null bằng cách thay thế Null thành giá trị trung bình (mean) hoặc giá trị xuất hiện nhiều nhất (mode) trong cột.
- Các cột "Age", "Work\_Experience", và "Family\_Size" : thay thế Null bằng mean tương ứng của cột.
- Các cột "Ever\_Married", "Graduated": thay thế Null bằng mode tương ứng của cột.

**Nhận xét:** Không có cột nào được in ra vì không còn giá trị Null nào cần được báo cáo.

```
# 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:  
-----



# NỘI DUNG

**01**

GIỚI THIỆU TỔNG QUAN

**02**

HƯỚNG TRIỂN KHAI

**03**

KHÁM PHÁ VÀ XỬ LÝ  
DỮ LIỆU

**04**

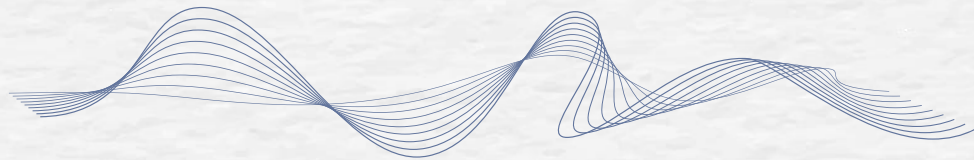
TRIỂN KHAI K-MEANS

**05**

KẾT QUẢ VÀ ĐÁNH GIÁ

**06**

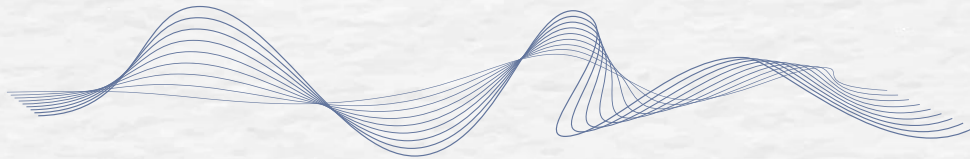
KẾT LUẬN



04

# TRIỂN KHAI K-MEANS

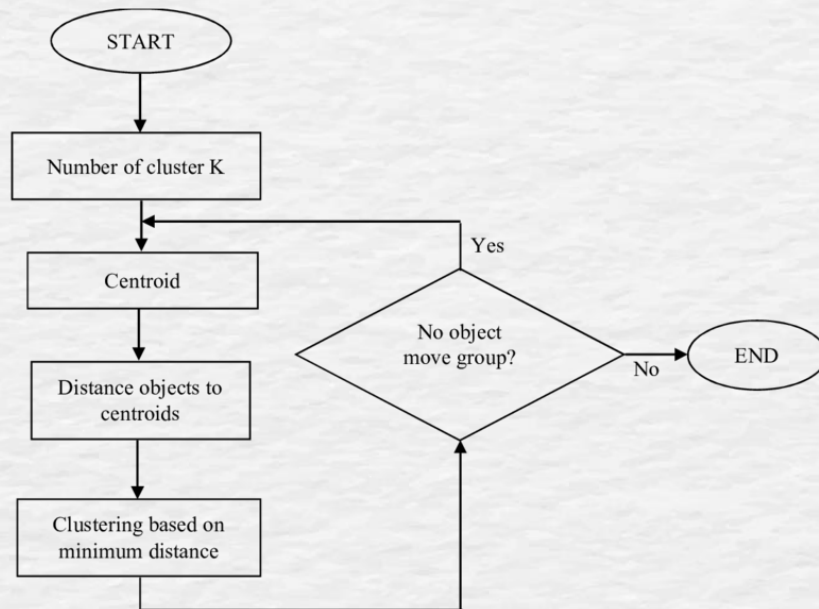
## 4.1 Giới thiệu thuật toán



**Thuật toán K-means** là một phương pháp phân cụm dữ liệu lặp lại, với mục tiêu phân tách tập dữ liệu thành K nhóm không chồng chéo. Nó tối ưu hóa sự tương đồng trong mỗi nhóm và tăng cường sự khác biệt giữa các nhóm.



## 4.1 Giới thiệu thuật toán

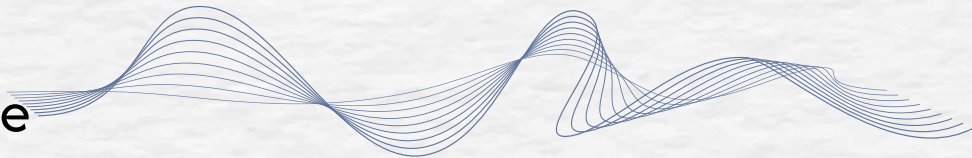


Quá trình hoạt động của thuật toán Kmeans:

- Xác định số lượng cụm K.
- Khởi tạo các trọng tâm ban đầu.
- 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 gần nhất.
- Tính toán lại vị trí của trọng tâm cho mỗi cụm.
- Đánh giá chất lượng của phân cụm.



## 4.2 Độ đo Silhouette Score



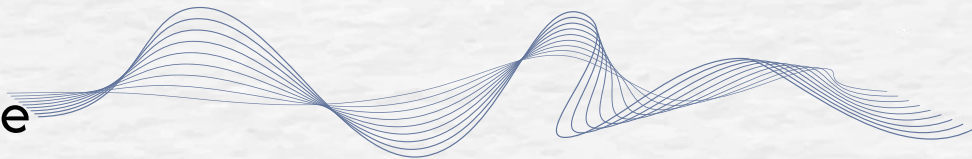
**Silhouette Score** được sử dụng để đánh giá chất lượng của việc phân cụm trong các thuật toán như Kmeans, đo lường mức độ "phù hợp" của nó trong cụm mà nó thuộc về.

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

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 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)).

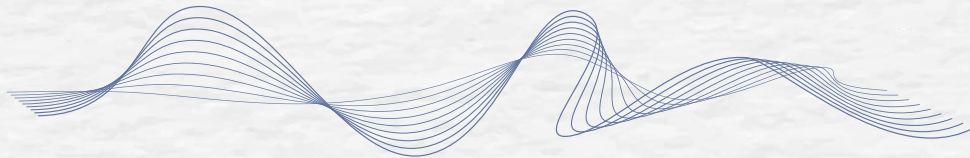
## 4.2 Độ đo Silhouette Score



### **Đặc điểm Silhouette Score:**

- 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.

## 4.3 Triển khai thuật toán



### Khai báo các hàm cần thiết

*a) Đọc dữ liệu và tạo vector đặc trưng*

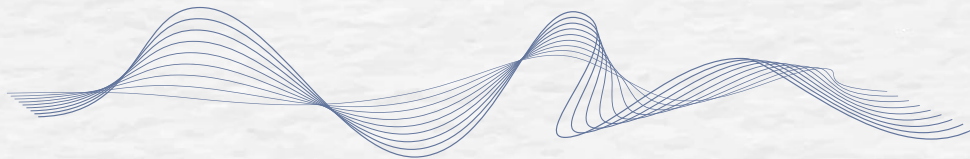
#### Read data

```
[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
```

#### Generate features values for RDD

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

## 4.3 Triển khai thuật toán



### Khai báo các hàm cần thiết

*b) Hàm tính khoảng cách Euclidean*

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
```

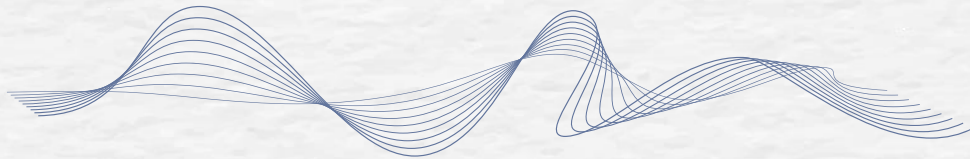
*c) Hàm kiểm tra sự hội tụ của K-means*

### 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
```

## 4.3 Triển khai thuật toán



### **Khai báo các hàm cần thiết**

*d) Hàm phân cụm K-means ( $max\_iterations = 100, threshold=0.01$ )*



#### d) Hàm phân cụm K-means ( $\text{max\_iterations} = 100, \text{threshold} = 0.01$ )

```
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])
        )

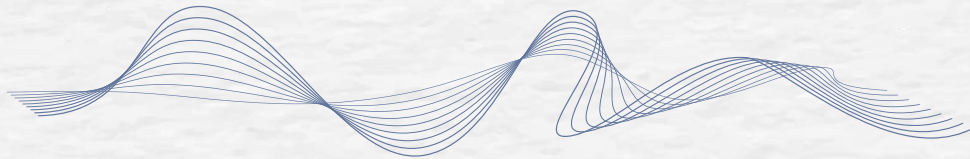
        # Chia mỗi phần tử trong mảng cho số lượng phần tử thuộc từng cụm để lấy mảng chứa trung bình từng phần tử của các điểm trong cụm
        # Thực hiện với dữ liệu của tất cả các điểm tại 1 nơi duy nhất
        new_centroids = remapping.map(lambda x: x[1][0] / x[1][1]).collect()

        # 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
```

## 4.3 Triển khai thuật toán



### **Khai báo các hàm cần thiết**

*e) Hàm tính Silhouette Score*

## e) Hàm tính Silhouette Score

```
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]

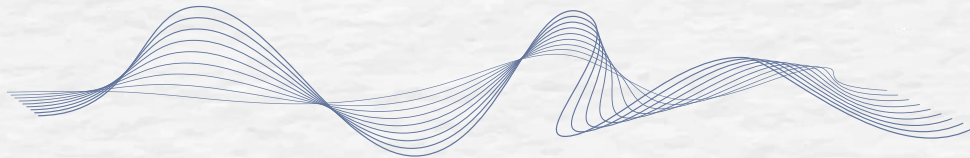
        # Tính toán b (khoảng cách trung bình đến cụm gần nhất khác)
        min_other = float('inf')
        for c in cluster_stats:
            if c != cluster_id:
                sil = compute_csi_diff(csi, vector, cluster_stats[c])
                if sil < min_other:
                    min_other = sil

        # 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()
```

## 4.3 Triển khai thuật toán



### Thực nghiệm

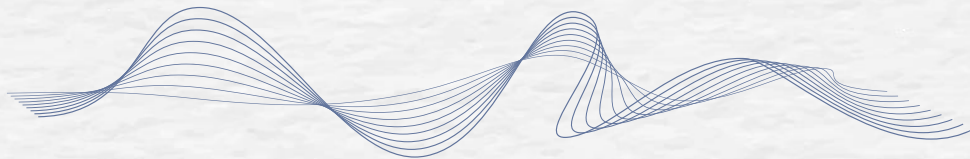
*a) Tìm K tốt nhất cho bộ dữ liệu*

- Khoảng tìm K: [2, 10].
- Độ đo đánh giá: Silhouette Score.
- Early Stopping: Nếu 2 lần liên tiếp điểm số Silhouette giảm xuống nhiều hơn 0.05 so với K trước đó, thực hiện dừng tìm K.

#### Find best k with all features

```
prev_score = None
count = 0
scores_on_k = []
for k in range(MIN_K, MAX_K):
    centroids, assignments = kmeans_clustering(rdd_data, k)
    score = silhouette_score_rdd(rdd_data, assignments)
    print(f"k = {k}: silhouette score = {score}")
    print("=====")
    scores_on_k.append(score)
    if prev_score is None:
        prev_score = score
    elif score <= prev_score and prev_score - score > 0.05:
        count += 1
    if count >= 2:
        break
```

## 4.3 Triển khai thuật toán





### Thực nghiệm


a) Tìm K tốt nhất cho bộ dữ liệu

- Khoảng tìm K: [2, 10].
- Độ đo đánh giá: Silhouette Score.
- Early Stopping: Nếu 2 lần liên tiếp điểm số Silhouette giảm xuống nhiều hơn 0.05 so với K trước đó, thực hiện dừng tìm K.

**Kết quả: best\_K=2**

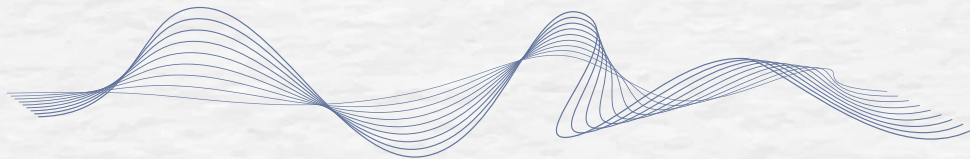
```
Clustering with k=2...: 5%  5/100 [00:08<02:12, 1.39s/it]
Clustering has converged after 6 iterations.
silhouette score = 0.7066330036335957 - time = 2.1034257411956787
=====

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
=====
```



## 4.3 Triển khai thuật toán



### Thực nghiệm

*b) Lựa chọn đặc trưng tốt nhất*

Với  $K=2$ :

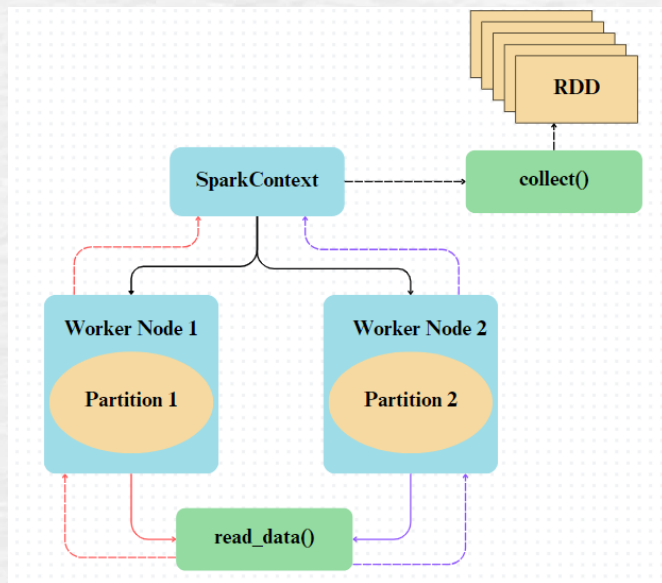
- Trường hợp 0: Loại bỏ đặc trưng "Work\_Experience".
- Trường hợp 1: Loại bỏ thêm đặc trưng "Family\_Size".
- Trường hợp 2: Loại bỏ thêm đặc trưng "Ever\_Married".
- Trường hợp 3: Loại bỏ thêm đặc trưng "Profession".

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

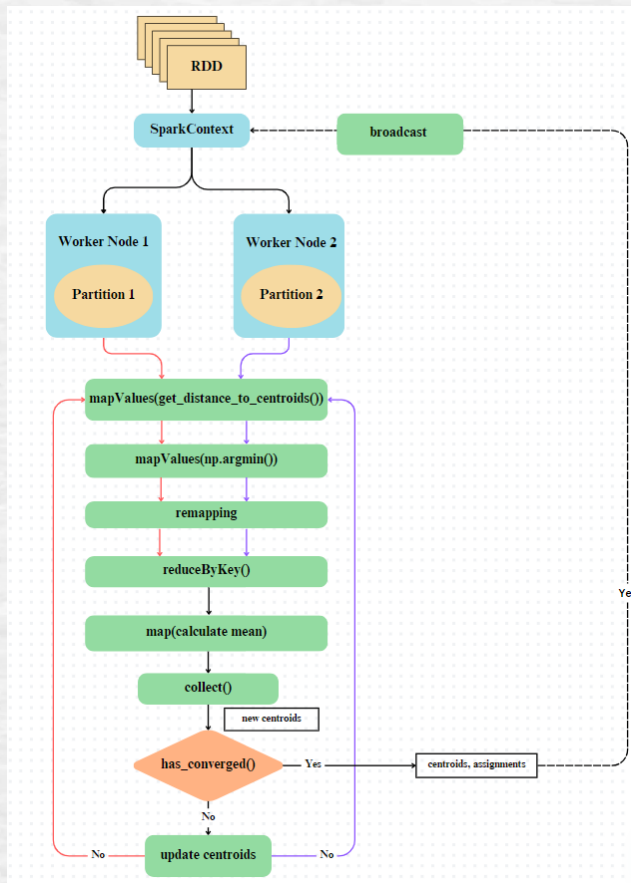
**Kết quả: best\_features=["Age", "Gender", "Graduated", "Profession", "Spending\_Score"]**

## 4.4 Trực quan hóa K-means

Song song hóa quá trình đọc dữ liệu và tạo vector đặc trưng.



## 4.4 Trực quan hóa K-means



Song song hóa giải thuật K-means.



# NỘI DUNG

**01**

GIỚI THIỆU TỔNG QUAN

**02**

HƯỚNG TRIỂN KHAI

**03**

KHÁM PHÁ VÀ XỬ LÝ  
DỮ LIỆU

**04**

TRIỂN KHAI K-MEANS

**05**

KẾT QUẢ VÀ ĐÁNH GIÁ

**06**

KẾT LUẬN

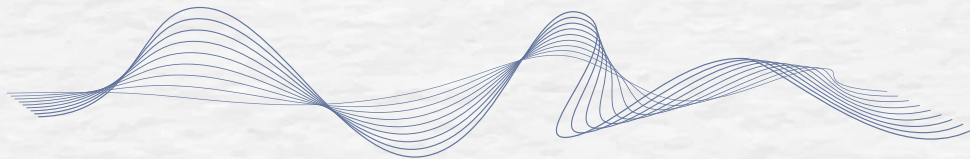


05

# KẾT QUẢ VÀ ĐÁNH GIÁ



## 5.1 Kết quả thực nghiệm



Silhouette Score trên các giá trị K.

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

### Nhận xét:

- Silhouette Score trong trường hợp K = 2 là tốt nhất với **0.7066** và hội tụ tương đối nhanh.
- 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.

## 5.1 Kết quả thực nghiệm

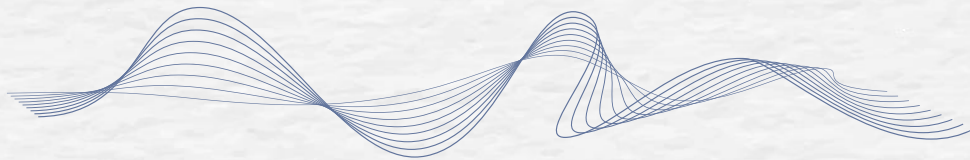
Silhouette Score trên các đặc trưng.

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".	<b>0.7885</b>	<b>3</b>
Loại bỏ thêm đặc trưng "Profession".	0.5171	10

### 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**.
- 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.

## 5.1 Kết quả thực nghiệm



### **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.

## 5.2 So sánh với thư viện PySpark ML

```
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

### Nhận xét:

- Với K=2 và các đặc trưng được lựa chọn, pyspark.ml.clustering.Kmeans và pyspark.ml.evaluation.ClusteringEvaluator cho kết quả **0.7885**.
- Thuật toán Kmeans và 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.



# NỘI DUNG

**01**

GIỚI THIỆU TỔNG QUAN

**02**

HƯỚNG TRIỂN KHAI

**03**

KHÁM PHÁ VÀ XỬ LÝ  
DỮ LIỆU

**04**

TRIỂN KHAI K-MEANS

**05**

KẾT QUẢ VÀ ĐÁNH GIÁ

**06**

KẾT LUẬN





06

# KẾT LUẬN

## 6.1 Tổng kết

- Áp dụng thành công các kỹ thuật xử lý Dữ liệu lớn kết hợp với Kmeans mà không sử dụng thư viện có sẵn để phân chia khách hàng thành các nhóm có đặc điểm đồng nhất.
- Giúp hiểu sâu hơn về thuật toán KMeans.
- Kết quả giúp nâng cao hiệu quả hoạt động kinh doanh, tăng cường cạnh tranh, mang lại lợi ích cho khách hàng.

## 6.2 Ưu và nhược điểm

Ưu điểm	Nhược điểm
Khả năng tùy chỉnh cao	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
Tích hợp tốt với Pyspark để xử lý dữ liệu lớn	Nhạy cảm với dữ liệu nhiễu
Đơn giản và hiệu quả, linh hoạt trên nhiều dự án	Không thích hợp cho dữ liệu không phân tán đều

## 6.3 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.
- Phát triển giao diện người dùng giúp tương tác với KMeans dễ dàng, trực quan.
- Xử lý dữ liệu thời gian thực: Ứng dụng khả năng xử lý dữ liệu lớn của Spark .
- Tích hợp giải pháp đối với dữ liệu nhiễu.

# CẢM ƠN THẦY VÀ CÁC BẠN ĐÃ CHÚ Ý LẮNG NGHE!

21521211@gm.uit.edu.vn

21521200@gm.uit.edu.vn

21522730@gm.uit.edu.vn

21520524@gm.uit.edu.vn





# TÀI LIỆU THAM KHẢO

---

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/>

---

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](https://github.com/mohamed-said-ibrahem/Parallel_K-Means_Using_Spark)

---

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

---

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/>

---

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

---

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

---