

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN



BÁO CÁO CUỐI KỲ
DỮ LIỆU LỚN

Chủ đề: Car Rates Clustering and Prediction

GVHD: Ths. Nguyễn Hồ Duy Trí

Tên Nhóm: Nhóm 5

Tên thành viên:

Nguyễn Lê Thái Hiền	20521308
Đinh Võ Xuân Hoàn	20521337
Dương Bảo Tâm	20521865
Trần Duy Khánh	20521457

TP Hồ Chí Minh, Tháng 5, Năm 2024

Mục lục

1. Thông tin chung.....	5
1.1. Lý do chọn đề tài	5
1.2. Mô tả dữ liệu.....	5
1.3. Mô tả bài toán	6
2. Kỹ thuật tiền xử lý	6
2.1. Thu thập dữ liệu.....	6
2.2. Tính toán thống kê.....	8
2.3. Tiền xử lý dữ liệu	11
2.3.1. Bỏ cột không cần thiết.....	11
2.3.2. Xử lý dữ liệu bị thiếu.....	11
2.3.3. Mã hóa các cột phân loại thành số nguyên.....	13
2.3.4. Chuẩn hóa các cột số và các cột đã được mã hóa.....	13
2.4. Trực quan hóa dữ liệu.....	15
3. Thuật toán khai thác dữ liệu	23
3.1. Linear Regression	24
3.1.1 Lý do chọn thuật toán :	24
3.1.2 Công thức Linear Regression (LN):	24
3.1.3 Lập trình Linear Regression (LN) trên Apache Spark:	25
3.2. GBM	31
3.3. KMeans.....	40
3.3.1. Lý do chọn lựa phương pháp KMeans	40
3.3.2. Phương Pháp Elbow và Within-Cluster Sum of Squares (WSS)	40
3.3.3. KMeans Song Song và Không Song Song	43
3.4. Ứng dụng Spark Cluster trong Data Mining	44
4. Kết quả đạt được.....	47
4.1. Phát biểu kết quả.....	47
4.1.1. Linear.....	47
4.1.2. GBM	48
4.1.3. KMeans.....	48

4.1.4. Spark Cluster	51
4.2. Độ đo	52
4.2.1. RMSE (Root Mean Squared Error):	52
4.2.2. MAE (Mean Absolute Error):	53
4.2.3. R ² (R-squared):	53
4.3. So sánh, đánh giá	53
4.3.1. So sánh Linear Regression và GBM	53
4.3.2. KMeans song song và không song song	54
4.3.3. Đánh giá hiệu quả Spark Cluster	54
5. Kết luận.....	55
5.1. Ưu điểm	55
5.2. Hạn chế	55
5.3. Hướng phát triển.....	55
6. Bảng phân công công việc.....	55
7. Tài liệu tham khảo	56

NHẬN XÉT CỦA GIẢNG VIÊN

1. Thông tin chung.

1.1. Lý do chọn đề tài

Tập dữ liệu Cars 2023 bao gồm nhiều thông tin về hàng ngàn mẫu xe hơi từ nhiều nhà sản xuất khác nhau. Các thông số kỹ thuật, giá cả, hiệu suất, độ tin cậy cho thấy được độ đa dạng của dữ liệu, là điều kiện thuận lợi để phân tích dữ liệu lớn. Tập dữ liệu được cập nhật mới nhất, đảm bảo thông tin chính xác, cho phép thực hiện nhiều loại phân tích khác nhau, đặc biệt nhờ vào các thuộc tính của xe để đánh giá tổng thể, qua đó giúp người tiêu dùng đánh giá chất lượng của xe.

1.2. Mô tả dữ liệu

Link nguồn kho dữ liệu: [New York Cars ~ Big Data \(2023\) \(kaggle.com\)](https://www.kaggle.com/datasets/mwaskom/us-cars-for-sale)

Tập dữ liệu chứa riêng các xếp hạng ô tô mang đến cơ hội làm việc với dữ liệu thực, cho phép thực hành các kỹ thuật phân tích dữ liệu khác nhau như trực quan hóa dữ liệu, phân tích hồi quy để dự đoán giá và các nhiệm vụ phân loại như phân loại thương hiệu.

Dataset: [Car_Rates.csv](#)

Tập dữ liệu này bao gồm các đánh giá về xe thu được từ trang web Cars.com. Nó sẽ có ích trong việc xử lý với tập dữ liệu New_York_cars. Bộ dữ liệu này bao gồm 7 loại xếp hạng ô tô riêng biệt:

1. General_rate
2. Comfort
3. Interior design
4. Performance
5. Value for the money
6. Exterior styling
7. Reliability

STT	Tên thuộc tính	Kiểu dữ liệu	Mô tả
1	Car_name	Varchar	Tên của xe
2	Num_of_reviews	Int	Số lượng đánh giá
3	General_rate	Float	Đánh giá chung
4	Comfort	Float	Đánh giá về sự thoải mái
5	Interior design	Float	Đánh giá về thiết kế nội thất

6	Performance	Float	Đánh giá về hiệu suất
7	Value for the money	Float	Đánh giá về giá trị so với tiền bỏ ra
8	Exterior styling	Float	Đánh giá về thiết kế ngoại thất
9	Reliability	Float	Đánh giá về độ tin cậy
10	Year	Int	Năm sản xuất
11	Brand	Varchar	Thương hiệu
12	Model	Varchar	Mẫu xe

1.3. Mô tả bài toán

Bài toán của nhóm em liên quan đến việc phân tích và dự đoán đánh giá chung của người dùng về các mẫu xe ô tô dựa trên một loạt các thuộc tính khác nhau. Dữ liệu bao gồm các thông tin như tên xe, số lượng đánh giá, đánh giá về sự thoải mái, thiết kế nội thất, hiệu suất, giá trị so với tiền bỏ ra, thiết kế ngoại thất, độ tin cậy, năm sản xuất, thương hiệu và mẫu xe.

Các thuật toán khai thác dữ liệu như KMeans, Linear Regression và GBM sẽ được sử dụng để phân tích và mô hình hóa dữ liệu này. KMeans có thể được sử dụng để phân nhóm các mẫu xe dựa trên các đặc điểm tương tự, trong khi Linear Regression và GBM có thể được sử dụng để dự đoán các yếu tố như đánh giá chung hoặc hiệu suất dựa trên các thuộc tính khác của xe.

Kết quả của bài toán này không chỉ giúp người tiêu dùng hiểu rõ hơn về các mẫu xe mà họ đang xem xét mua, mà còn giúp các nhà sản xuất và đại lý ô tô hiểu rõ hơn về những gì mà khách hàng đánh giá cao trong một chiếc xe, từ đó có thể tối ưu hóa sản phẩm và dịch vụ của họ.

2. Kỹ thuật tiền xử lý

2.1. Thu thập dữ liệu

Dữ liệu được tổng hợp từ file csv bao gồm các đánh giá về xe thu được từ trang web Cars.com ([Car Rates.csv](#)).

```
In [2]: # Tao Spark session
spark = SparkSession.builder.appName("KMeansClustering").getOrCreate()

# Đọc dữ liệu
data = spark.read.format("csv")\
    .option("header", "true")\
    .option("inferSchema", "true")\
    .load("Car_Rates.csv")

24/06/02 12:50:01 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.

In [3]: #Đếm số dòng
rows = data.count()
print(f"Number of rows: {rows}")

#Đếm số cột
cols = len(data.columns)
print(f"Number of columns: {cols}")

Number of rows: 4243
Number of columns: 12
```

Hình 1. Tạo Spark, đọc dữ liệu

Mô tả tập dữ liệu

```
In [4]: #Hiển thị schema dữ liệu
data.printSchema()

root
 |-- Car_name: string (nullable = true)
 |-- Num_of_reviews: double (nullable = true)
 |-- General_rate: double (nullable = true)
 |-- Comfort: double (nullable = true)
 |-- Interior design: double (nullable = true)
 |-- Performance: double (nullable = true)
 |-- Value for the money: double (nullable = true)
 |-- Exterior styling: double (nullable = true)
 |-- Reliability: double (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Brand: string (nullable = true)
 |-- Model: string (nullable = true)
```

Hình 2. Hiển thị schema dữ liệu

Truy vấn dữ liệu và hiển thị kết quả truy vấn

```
In [5]: # Tạo view tạm thời
data.createOrReplaceTempView("car_rates_view")

# Truy vấn dữ liệu
sqlDF = spark.sql("""
SELECT Car_name, Brand, Model, General_rate, Year
FROM car_rates_view
""")

# Hiển thị kết quả truy vấn
sqlDF.show()
```

	Car_name	Brand	Model	General_rate	Year
2023	Acura	...	Acura	Integra.	4.6
2023	Acura	...	Acura	MDX.	NULL
2023	Acura	...	Acura	TLX.	NULL
2023	Acura	...	Acura	RDX.	NULL
2022	Acura	...	Acura	TLX.	4.8
2022	Acura	...	Acura	MDX.	4.7
2022	Acura	...	Acura	RDX.	4.8
2022	Acura	...	Acura	ILX.	NULL
2022	Acura	...	Acura	NSX.	NULL
2021	Acura	...	Acura	ILX.	4.9
2021	Acura	...	Acura	TLX.	4.6
2021	Acura	...	Acura	RDX.	4.8
2021	Acura	...	Acura	NSX.	NULL
2020	Acura	...	Acura	ILX.	4.6
2020	Acura	...	Acura	RLX Sport Hybrid.	5.0
2020	Acura	...	Acura	MDX.	4.8
2020	Acura	...	Acura	TLX.	4.9
2020	Acura	...	Acura	MDX Sport Hybrid.	4.9
2020	Acura	...	Acura	RDX.	4.6
2020	Acura	...	Acura	RLX.	4.8

only showing top 20 rows

Hình 3. Truy vấn và hiển thị kết quả

2.2. Tính toán thống kê

Tính toán các giá trị max, min, avg, median cho các cột số

```
In [6]: # Tính giá trị cao nhất, thấp nhất, trung bình cho các cột số
numerical_columns = ["Num_of_reviews", "General_rate", "Comfort", "Interior design", "Performance", "Value for the mo
categorical_columns = ["Year", "Brand", "Model"]

stats = data.select(
    *[F.max(c).alias(f"max_{c}") for c in numerical_columns],
    *[F.min(c).alias(f"min_{c}") for c in numerical_columns],
    *[F.avg(c).alias(f"avg_{c}") for c in numerical_columns]
)
```

```
In [7]: # Tính giá trị trung vị cho các cột số
def median(col_name):
    return F.expr(f'percentile_approx(`{col_name}` , 0.5)')

medians = data.select(
    *[median(c).alias(f"median_{c}") for c in numerical_columns]
)
```

Hình 4. Tính toán các giá trị toán học

Thu thập kết quả đã tính toán và hiển thị kết quả

```
In [8]: # Thu thập kết quả từ Spark DataFrame
stats_dict = stats.collect()[0].asDict()
medians_dict = medians.collect()[0].asDict()

# Kết hợp kết quả
combined_stats = {**stats_dict, **medians_dict}

# Định dạng kết quả dưới dạng bảng
header = ["Metric", "Value"]
rows = [[k, v] for k, v in combined_stats.items()]

# Tính độ dài của mỗi cột để căn chỉnh đầu ra
max_lengths = [max(len(str(item))) for item in col] for col in zip(*([header] + rows))]

# Tao chuỗi định dạng cho các cột
format_str = ' | '.join(['{:<{length}}' for length in max_lengths])

# In tiêu đề
print(format_str.format(*header))
print('-' * (sum(max_lengths) + len(max_lengths) * 3 - 1))

# In các hàng dữ liệu
for row in rows:
    print(format_str.format(*row))

24/06/02 12:50:08 WARN SparkString_Utils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.
```

Hình 5. Thu thập kết quả

Metric	Value
max_Num_of_reviews	1024.0
max_General_rate	5.0
max_Comfort	5.0
max_Interior_design	5.0
max_Performance	5.0
max_Value_for_the_money	5.0
max_Exterior_styling	5.0
max_Reliability	5.0
min_Num_of_reviews	1.0
min_General_rate	1.0
min_Comfort	1.0
min_Interior_design	1.0
min_Performance	1.0
min_Value_for_the_money	1.0
min_Exterior_styling	1.0
min_Reliability	1.0
avg_Num_of_reviews	55.57977711334602
avg_General_rate	4.626963848871978
avg_Comfort	4.654879043218254
avg_Interior_design	4.62576787170426
avg_Performance	4.616716499048649
avg_Value_for_the_money	4.479179124762183
avg_Exterior_styling	4.718211470508266
avg_Reliability	4.629366812227053
median_Num_of_reviews	22.0
median_General_rate	4.7
median_Comfort	4.7
median_Interior_design	4.7
median_Performance	4.7
median_Value_for_the_money	4.6
median_Exterior_styling	4.8
median_Reliability	4.7

Hình 6. Hiển thị kết quả

Tìm giá trị phổ biến và hiếm nhất cho các cột chữ

```
In [9]: # Tìm giá trị phổ biến và hiếm nhất cho các cột chữ
categorical_columns = ["Brand", "Model", "Year"]

most_frequent = {}
least_frequent = {}

for col_name in categorical_columns:
    freq_df = data.groupBy(col_name).agg(count(col_name).alias("count"))

    # Tìm giá trị phổ biến nhất
    most_freq_row = freq_df.orderBy(col("count").desc()).first()
    most_frequent[col_name] = (most_freq_row[col_name], most_freq_row["count"])

    # Tìm giá trị hiếm nhất
    least_freq_row = freq_df.orderBy(col("count")).first()
    least_frequent[col_name] = (least_freq_row[col_name], least_freq_row["count"])
```

Hình 7. Tìm giá trị phổ biến nhất

Hiển thị các giá trị phổ biến và hiếm nhất cho các cột chữ

```
In [10]: # Định dạng kết quả dưới dạng bảng
header = ["Column", "Most Frequent Value", "Count", "Least Frequent Value", "Count"]
rows = [
    [col_name, most_frequent[col_name][0], most_frequent[col_name][1], least_frequent[col_name][0], least_frequent[col_name][1]]
]

# Tính độ dài của mỗi cột để căn chỉnh đầu ra
max_lengths = [max(len(str(item))) for item in col] for col in zip(*([header] + rows))]

# Tao chuỗi định dạng cho các cột
format_str = ' | '.join(['{:<{}s}' for length in max_lengths])

# In tiêu đề
print(format_str.format(*header))
print('-' * (sum(max_lengths) + len(max_lengths) * 3 - 1))

# In các hàng dữ liệu
for row in rows:
    print(format_str.format(*row))

Column | Most Frequent Value | Count | Least Frequent Value | Count
-----|-----|-----|-----|-----|
Brand | BMW | 436 | Tesla | 1
Model | Escalade ESV. | 12 | Rio5. | 1
Year | 2022 | 390 | 2012 | 278
```

Hình 8. Hiển thị giá trị

2.3. Tiền xử lý dữ liệu

2.3.1. Bỏ cột không cần thiết

Lý do: Cột Car_name không mang giá trị số lượng hoặc phân loại có ý nghĩa trong việc xây dựng mô hình.

Cách thực hiện: Dùng phương thức .drop() để loại bỏ cột này.

```
[11]: # Bỏ cột Car_name
data = data.drop("Car_name")
```

Hình 9. Dùng drop() bỏ cột name

Kết quả: Không còn cột Car_name trong dataset.

2.3.2. Xử lý dữ liệu bị thiếu

Numerical Columns (xử lý cột số):

Lý do: Điền giá trị trung bình (mean) cho các cột số để tránh mất dữ liệu và giữ được đặc điểm trung bình của các cột này.

Cách thực hiện:

- **data.select(_mean(col_name)).collect()[0][0]**: Tính giá trị trung bình của cột và lấy giá trị đó.
- **data.fill({col_name: mean_value})**: Điền giá trị trung bình vào các vị trí bị thiếu.

Categorical Columns (xử lý cột phân loại):

Lý do: Điền giá trị phổ biến (mode) cho các cột phân loại để giữ lại đặc điểm phân phối phổ biến của các cột này.

Cách thực hiện:

- `data.groupBy(col_name).count().orderBy('count', ascending=False).first()[0]`:
Tìm giá trị phổ biến nhất (mode) của cột.
- `data.na.fill({col_name: mode_value})`: Điền giá trị phổ biến vào các vị trí bị thiếu.

```
[12]: # Xử lý missing values: điền giá trị trung bình cho các cột số và giá trị phổ biến cho các cột phân loại
for col_name in numerical_columns:
    mean_value = data.select(_mean(col_name)).collect()[0][0]
    data = data.na.fill({col_name: mean_value})

for col_name in categorical_columns:
    mode_value = data.groupBy(col_name).count().orderBy('count', ascending=False).first()[0]
    data = data.na.fill({col_name: mode_value})
```

Hình 10. Xử lý giá trị bị thiếu

Kết quả:

Các giá trị thiếu trong các cột số đã được điền giá trị trung bình.

Các giá trị thiếu trong các cột phân loại đã được điền giá trị phổ biến.

```
+-----+-----+-----+-----+
|Brand|      Model|General_rate|Year|
+-----+-----+-----+-----+
|Acura|      Integra.|        4.6|2023|
|Acura|      MDX.|4.626963848871978|2023|
|Acura|      TLX.|4.626963848871978|2023|
|Acura|      RDX.|4.626963848871978|2023|
|Acura|      TLX.|        4.8|2022|
|Acura|      MDX.|        4.7|2022|
|Acura|      RDX.|        4.8|2022|
|Acura|      ILX.|4.626963848871978|2022|
|Acura|      NSX.|4.626963848871978|2022|
|Acura|      ILX.|        4.9|2021|
|Acura|      TLX.|        4.6|2021|
|Acura|      RDX.|        4.8|2021|
|Acura|      NSX.|4.626963848871978|2021|
|Acura|      ILX.|        4.6|2020|
|Acura|RLX Sport Hybrid.|        5.0|2020|
|Acura|      MDX.|        4.8|2020|
|Acura|      TLX.|        4.9|2020|
|Acura|MDX Sport Hybrid.|        4.9|2020|
|Acura|      RDX.|        4.6|2020|
|Acura|      RLX.|        4.8|2020|
+-----+-----+-----+-----+
only showing top 20 rows
```

Hình 11. Hiển thị

2.3.3. Mã hóa các cột phân loại thành số nguyên

Lý do: Các thuật toán học máy không thể xử lý trực tiếp dữ liệu phân loại dạng chuỗi. Cần chuyển đổi thành các giá trị số.

Cách thực hiện:

- `data.select(col_name).distinct().rdd.flatMap(lambda x: x).collect()`: Lấy danh sách các giá trị phân biệt trong cột.
- `when(col(col_name) == lit(categories[0]), 0)`: Khởi tạo biểu thức ánh xạ giá trị đầu tiên trong danh sách thành 0.
- `mapping_expr.when(col(col_name) == lit(category), idx)`: Tiếp tục thêm các giá trị khác trong danh sách với số thứ tự tương ứng.
- `data.withColumn(col_name+"_encoded", mapping_expr.otherwise(col(col_name)))`: Tạo cột mới với các giá trị đã được mã hóa.

```
In [14]: # Mã hóa các cột phân loại thành số nguyên
for col_name in categorical_columns:
    categories = data.select(col_name).distinct().rdd.flatMap(lambda x: x).collect()
    mapping_expr = when(col(col_name) == lit(categories[0]), 0)
    for idx, category in enumerate(categories[1:], 1):
        mapping_expr = mapping_expr.when(col(col_name) == lit(category), idx)
    data = data.withColumn(col_name + "_encoded", mapping_expr.otherwise(col(col_name)))
```

Hình 12. Mã hóa cột thành số nguyên

Kết quả: Các cột phân loại Year, Brand, Model đã được mã hóa thành các cột số Year_encoded, Brand_encoded, Model_encoded.

2.3.4. Chuẩn hóa các cột số và các cột đã được mã hóa

Lý do: Đảm bảo các cột có cùng phạm vi giá trị, tránh ảnh hưởng của các thuộc tính có giá trị lớn nhỏ khác nhau đến mô hình.

Cách thực hiện:

- `data.select(*feature_columns).summary().filter("summary = 'mean' or summary = 'stddev'").collect()`: Lấy giá trị trung bình và độ lệch chuẩn của các cột.
- `means = {row['summary']: {col: row[col] for col in feature_columns} for row in summary}`: Tạo dictionary chứa giá trị trung bình và độ lệch chuẩn cho các cột.

- **def standardize(col, mean, stddev):**: Hàm chuẩn hóa một cột.
- **data.withColumn(col_name, standardize(col(col_name), mean, stddev)):** Áp dụng chuẩn hóa cho từng cột.

```
In [15]: # Chọn tất cả các cột số và các cột đã được mã hóa
feature_columns = numerical_columns + [col + "_encoded" for col in categorical_columns]

# Chuẩn hóa các cột số và cột đã mã hóa
summary = data.select(*feature_columns).summary().filter("summary = 'mean' or summary = 'stddev'").collect()
means = {row['summary']: {col: row[col] for col in feature_columns} for row in summary}

def standardize(col, mean, stddev):
    return (col - mean) / stddev

for col_name in feature_columns:
    mean = float(means['mean'][col_name])
    stddev = float(means['stddev'][col_name])
    data = data.withColumn(col_name, standardize(col_name), mean, stddev))
```

Hình 13. Chuẩn hóa các cột đã được mã hóa

Kết quả: Tất cả các cột số và cột đã mã hóa đã được chuẩn hóa, đảm bảo các cột có cùng phạm vi giá trị, giúp cải thiện hiệu quả mô hình.

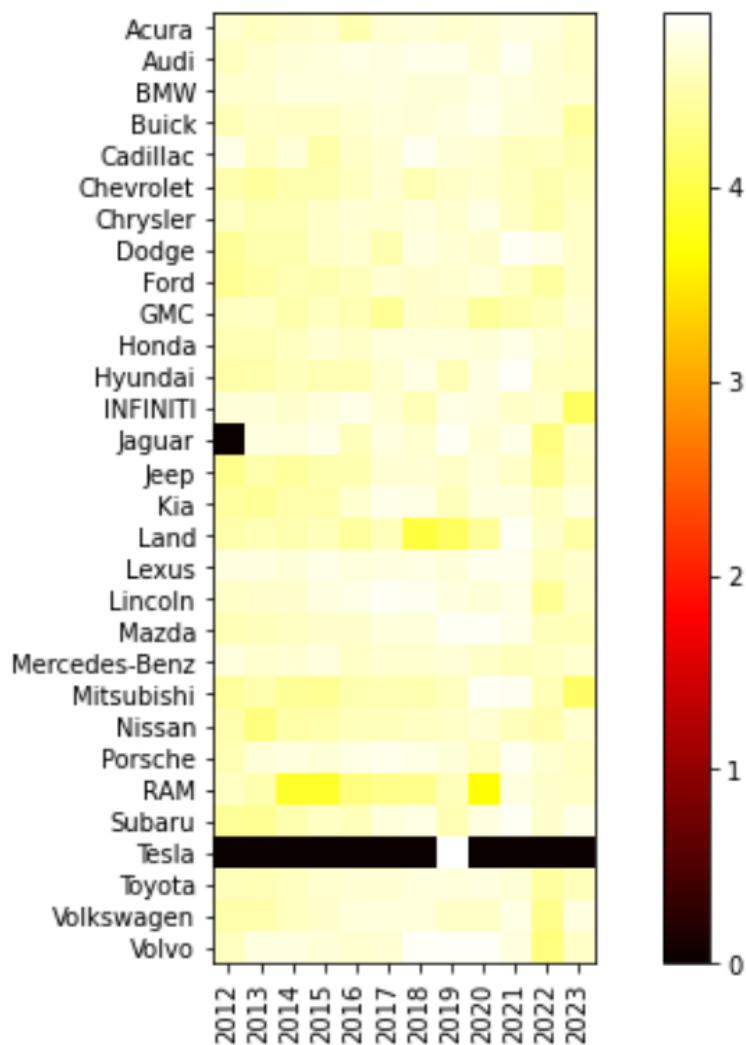
Brand	Model	General_rate	Year
Acura	Integra.	-0.08643390285902246	2023
Acura	MDX.	1.252723235112115...	2023
Acura	TLX.	1.252723235112115...	2023
Acura	RDX.	1.252723235112115...	2023
Acura	TLX.	0.5546756306465788	2022
Acura	MDX.	0.23412086389377956	2022
Acura	RDX.	0.5546756306465788	2022
Acura	ILX.	1.252723235112115...	2022
Acura	NSX.	1.252723235112115...	2022
Acura	ILX.	0.8752303973993808	2021
Acura	TLX.	-0.08643390285902246	2021
Acura	RDX.	0.5546756306465788	2021
Acura	NSX.	1.252723235112115...	2021
Acura	ILX.	-0.08643390285902246	2020
Acura	RLX Sport Hybrid.	1.19578516415218	2020
Acura	MDX.	0.5546756306465788	2020
Acura	TLX.	0.8752303973993808	2020
Acura	MDX Sport Hybrid.	0.8752303973993808	2020
Acura	RDX.	-0.08643390285902246	2020
Acura	RLX.	0.5546756306465788	2020

only showing top 20 rows

Hình 14. Hiển thị kết quả

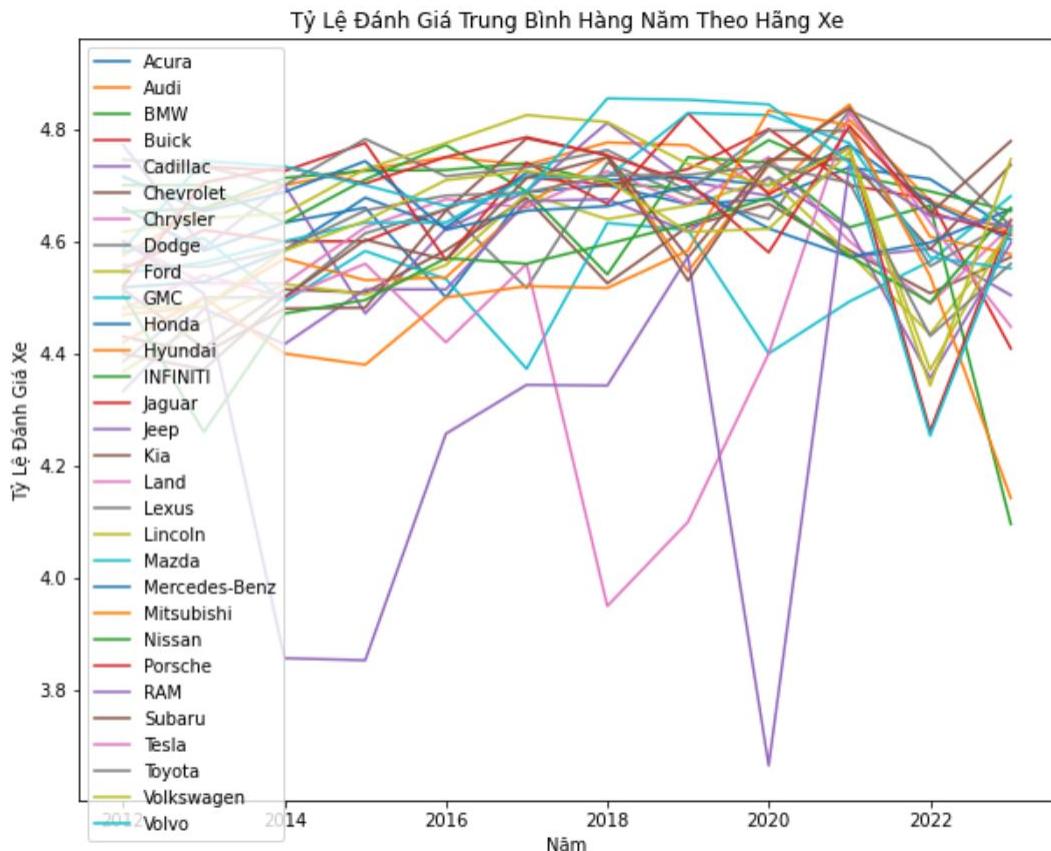
2.4. Trực quan hóa dữ liệu

Biểu đồ nhiệt (Heatmap) này có thể được sử dụng để phân tích và so sánh trung bình tỷ lệ đánh giá chung (General_rate) (được lấy giá trị từ 0 đến 5) của các hãng xe qua các năm (2012 – 2023).



Hình 15. Biểu đồ nhiệt

Biểu đồ dòng (Line Chart) thể hiện “Tỷ Lệ Đánh Giá Trung Bình” của các hãng xe từ năm 2012 đến 2023.



Hình 16. Biểu đồ dòng thể hiện tỷ lệ trung bình các hãng xe

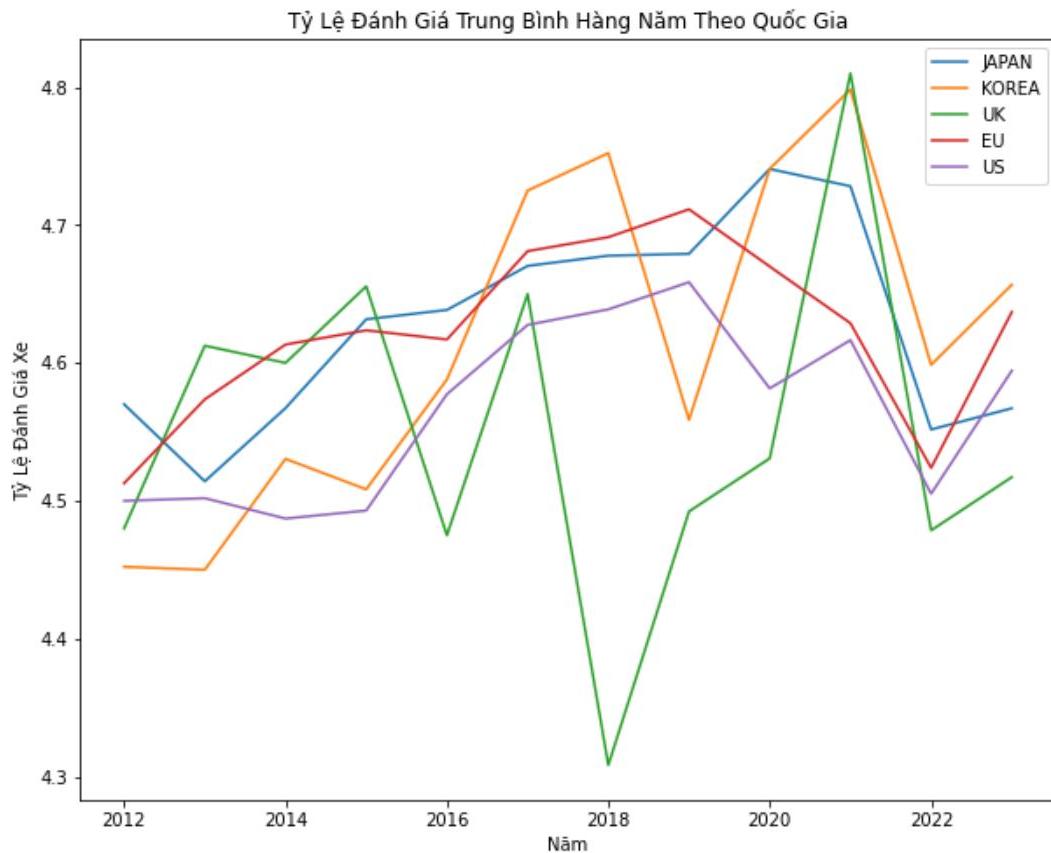
Phân loại các hãng xe thành các khu vực

```
In [14]: # Phân loại các hãng xe thành các khu vực như Châu Á hoặc Nhật Bản, Hàn Quốc, Mỹ, Châu Âu, Anh, Hàn Quốc.

JAPAN_cars = ("Acura", "Mazda", "Lexus", "Subaru", "Nissan", "Toyota", "Mitsubishi", "Honda", "INFINITI")
KOREA_cars = ("Hyundai", "Kia")
UK_cars = ("Jaguar", "Land")
EU_cars = ("Volvo", "Porsche", "Ford", "Mercedes-Benz")
US_cars = ("Ford", "Dodge", "Tesla", "Buick", "Cadillac", "Chevrolet", "Chrysler", "Jeep", "GMC", "Lincoln", "RAM")
Country_brands = (JAPAN_cars, KOREA_cars, UK_cars, EU_cars, US_cars)
Countries = ("JAPAN", "KOREA", "UK", "EU", "US")
```

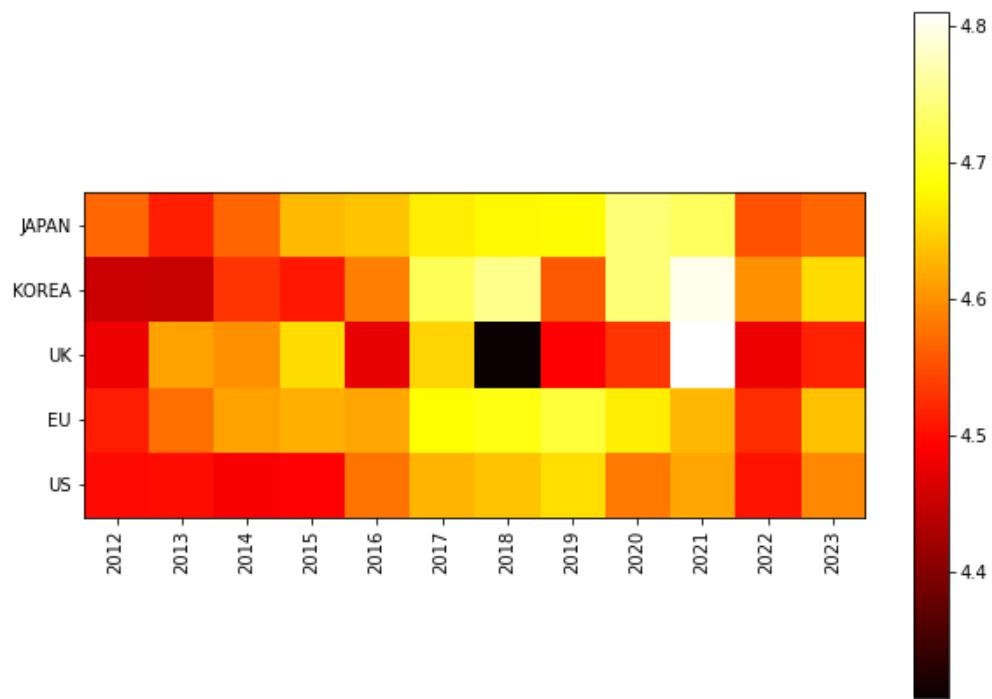
Hình 17. Phân loại hãng xe thành các khu vực

Sau đó tính toán và vẽ biểu đồ dòng (Line Chart) để thể hiện “Tỷ Lệ Đánh Giá Trung Bình” của các khu vực hãng xe từ năm 2012 đến 2023.



Hình 18. Biểu đồ dòng thể hiện tỷ lệ trung bình các khu vực hàng xe

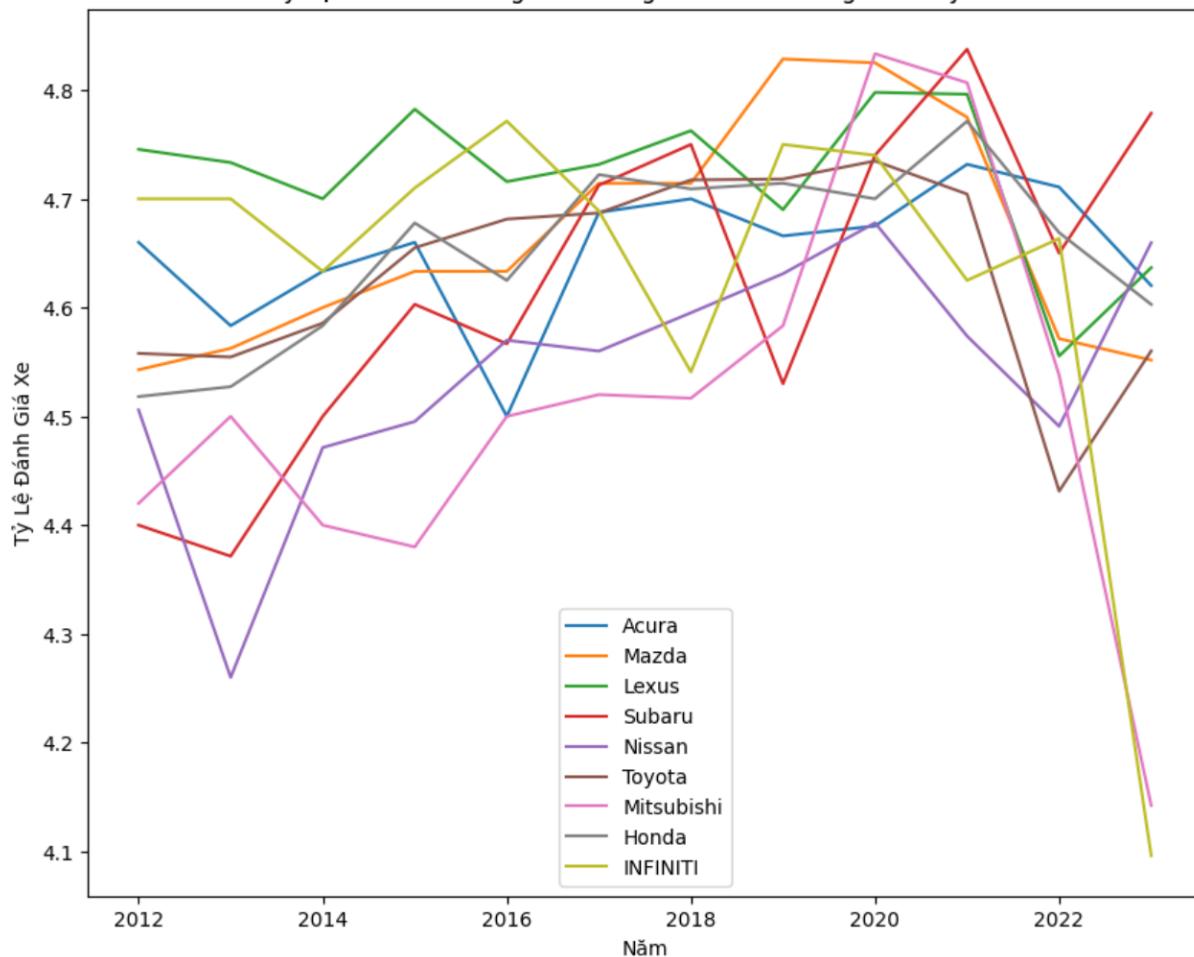
Và biểu đồ nhiệt (Heatmap) để phân tích và so sánh trung bình tỷ lệ đánh giá chung (General_rate) (được lấy giá trị từ 0 đến 5) của các khu vực hàng xe qua các năm (2012 – 2023).



Hình 19. Biểu đồ nhiệt phân tích General Rate

Và cuối cùng là các biểu đồ dòng (Line Chart) thể hiện “Tỷ Lệ Đánh Giá Trung Bình” cho các hãng xe của từng khu vực từ năm 2012 đến 2023.

Tỷ Lệ Đánh Giá Trung Bình Hàng Năm Theo Hãng Xe của JAPAN

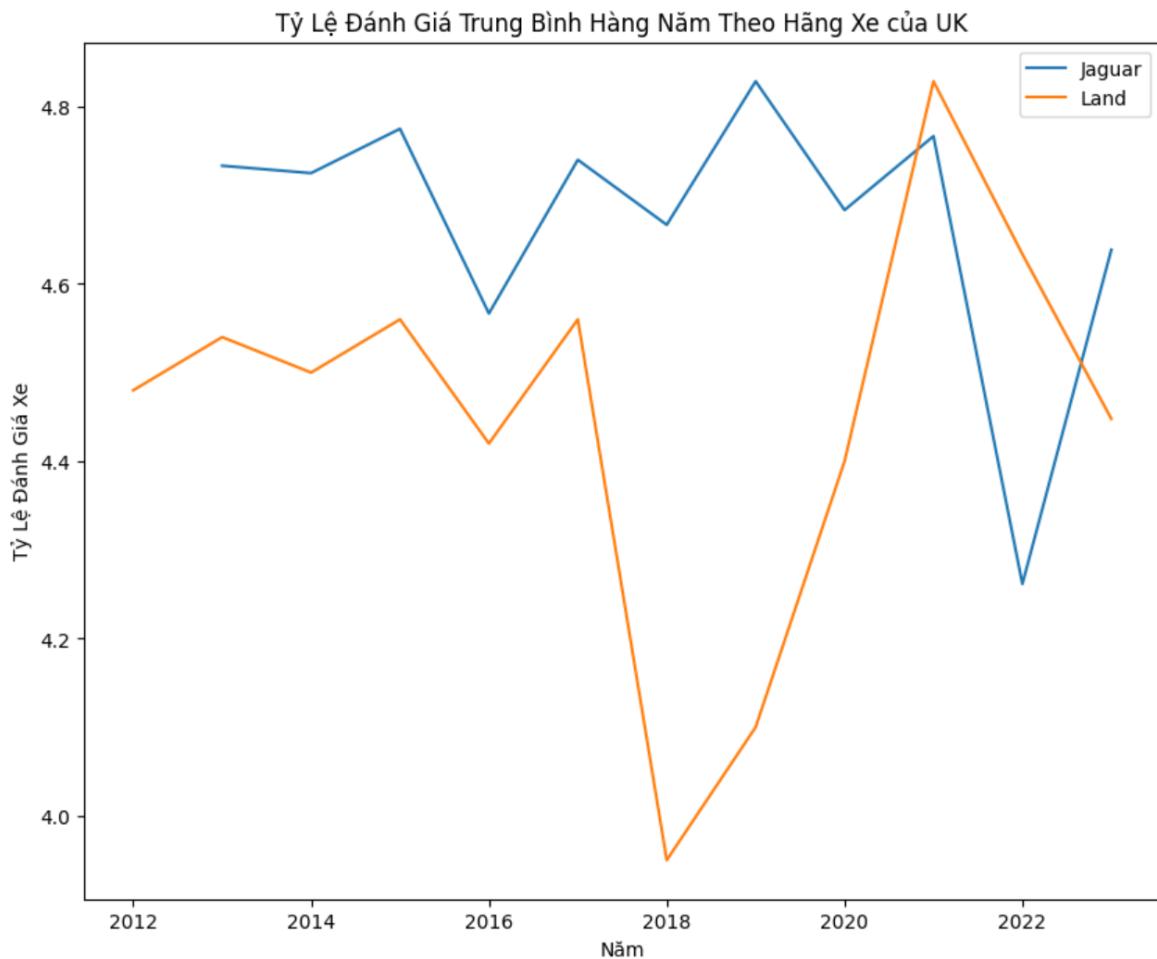


Hình 20. Biểu đồ dòng thể hiện tỷ lệ trung bình các hãng xe khu vực Japan

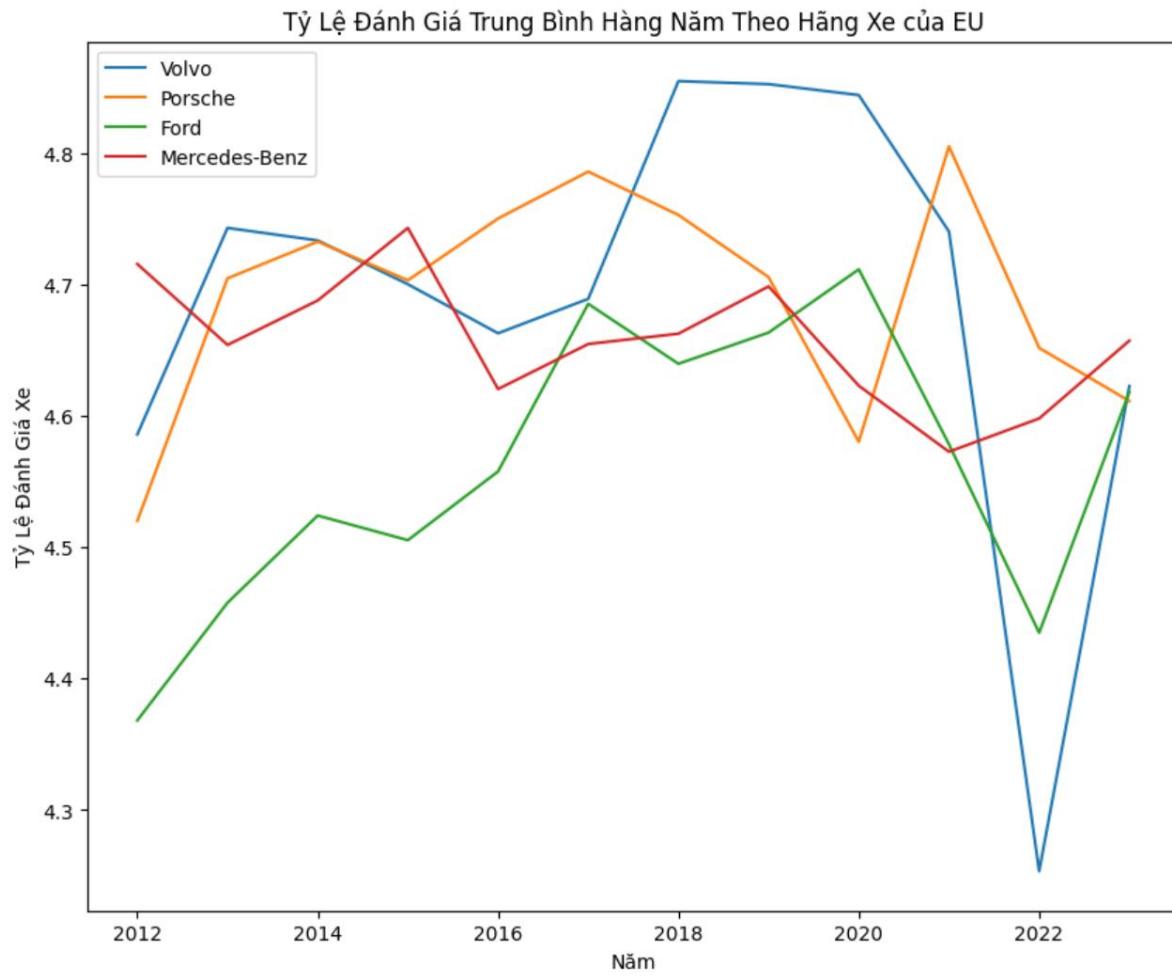
Tỷ Lệ Đánh Giá Trung Bình Hàng Năm Theo Hãng Xe của KOREA



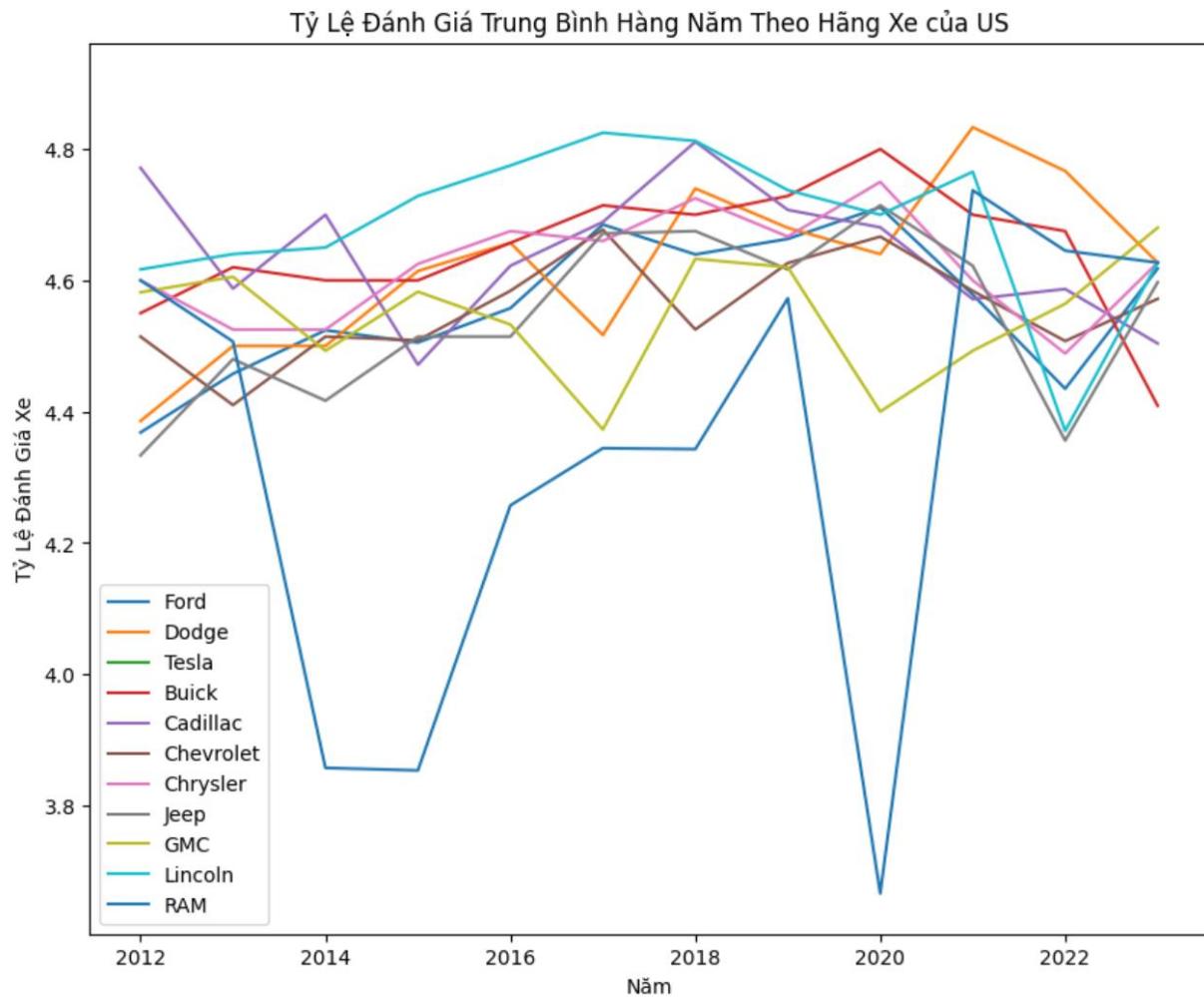
Hình 21. Biểu đồ dòng thể hiện tỷ lệ trung bình các hãng xe khu vực Korea



Hình 22. Biểu đồ dòng thể hiện tỷ lệ trung bình các hãng xe khu vực UK



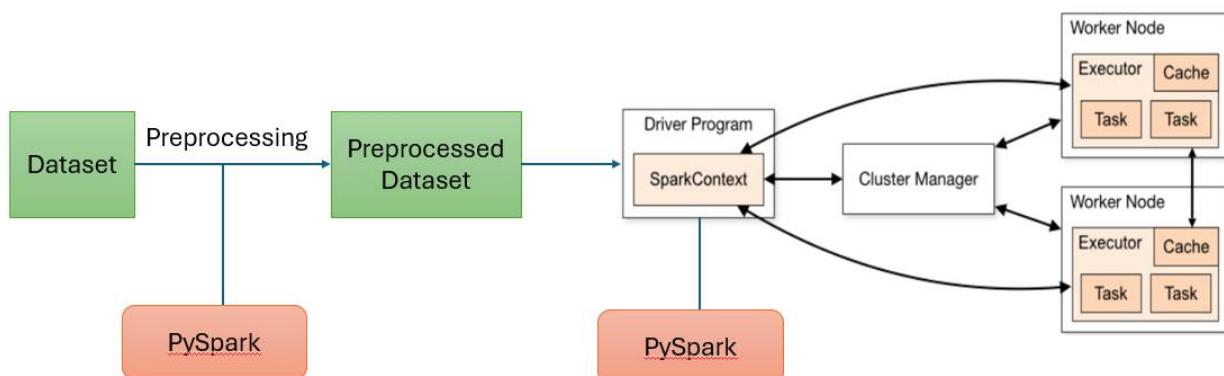
Hình 23. Biểu đồ dòng thể hiện tỷ lệ trung bình các hãng xe khu vực EU



Hình 24. Biểu đồ dòng thể hiện tỷ lệ trung bình các hãng xe khu vực US

3. Thuật toán khai thác dữ liệu

Minh họa việc song song hóa giải thuật:



3.1. Linear Regression

Linear Regression (Hồi quy tuyến tính) là một phương pháp thống kê để hồi quy dữ liệu với biến phụ thuộc có giá trị liên tục trong khi các biến độc lập có thể có một trong hai giá trị liên tục hoặc là giá trị phân loại. Nói cách khác "Hồi quy tuyến tính" là một phương pháp để dự đoán biến phụ thuộc (Y) dựa trên giá trị của biến độc lập (X)

3.1.1 Lý do chọn thuật toán :

- Đơn giản và dễ hiểu: Hồi quy tuyến tính là một phương pháp thống kê cơ bản và dễ hiểu.
- Hiệu quả trong xử lý dữ liệu lớn: Hồi quy tuyến tính, với tính đơn giản và hiệu quả của nó, hoạt động tốt trên các tập dữ liệu lớn, giúp việc triển khai và tính toán trở nên nhanh chóng và hiệu quả.
- Tính chất tuyến tính của vấn đề: Nếu mối quan hệ giữa biến phụ thuộc (tỷ lệ bán hàng) và các biến độc lập (như giá cả, số lượng xe bán, thời gian...) là tuyến tính hoặc gần tuyến tính, hồi quy tuyến tính sẽ cung cấp một mô hình chính xác và phù hợp
- Khả năng mở rộng và kết hợp với các kỹ thuật khác: Có thể dễ dàng mở rộng và kết hợp với các kỹ thuật tiền xử lý dữ liệu khác như chuẩn hóa, chọn lựa đặc trưng, và phân chia dữ liệu.
- Khả năng tối ưu hóa thông qua các thư viện PySpark: PySpark cung cấp các thư viện và API tối ưu hóa cho việc thực hiện hồi quy tuyến tính, giúp việc triển khai mô hình trở nên dễ dàng và hiệu quả.[1]

3.1.2 Công thức Linear Regression (LN):

Công thức hồi quy tuyến tính đa biến:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} + \epsilon$$

Trong đó, với i là số quan sát:

- y_i là biến phụ thuộc.
- x_i là các biến độc lập.
- β_0 là hệ số chặn (giá trị khi tất cả các biến độc lập bằng 0).
- β_p là các hệ số hồi quy cho từng biến độc lập.
- ϵ là sai số của mô hình (còn được gọi là phần dư).

Phương trình hồi quy tuyến tính bội:

$$\hat{y} = b_0 + b_1 X_1 + b_2 X_2 + \dots + b_p X_p$$

Trong đó:

- \hat{y} là giá trị dự đoán hoặc mong đợi của biến phụ thuộc.
- X_1 đến X_p là các biến độc lập hoặc biến dự đoán riêng biệt.
- b_0 là giá trị của y khi tất cả các biến độc lập X_1 đến X_p đều bằng không.
- b_1 đến b_p là các hệ số hồi quy ước lượng.

Với các ma trận X B Y:

$$X = \begin{pmatrix} 1 & x_{11} & \dots & x_{k1} \\ 1 & x_{12} & \dots & x_{k2} \\ \dots & \dots & \dots & \dots \\ 1 & x_{1n} & \dots & x_{kn} \end{pmatrix} \quad B = \begin{pmatrix} b_0 \\ b_1 \\ \dots \\ b_k \end{pmatrix} \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$

Ta có hệ phương trình:

$$\begin{cases} X^T \cdot X \cdot B = X^T \cdot Y \\ B = (X^T \cdot X)^{-1} (X^T \cdot Y) \end{cases}$$

Trong đó:

- B là ma trận cần tìm
- B là ma trận gồm các thuộc tính độc lập trong dữ liệu
- Y là ma trận 1 cột là cột cần dự đoán

3.1.3 Lập trình Linear Regression (LN) trên Apache Spark:

- Bước 1: Thêm các thư viện cần thiết

```
#Import các thư viện cần thiết
from pyspark.sql import SparkSession
from pyspark.sql.functions import mean, col, when, count, desc
import pyspark.sql.functions as F
from pyspark.sql.functions import round as spark_round
```

Hình 25. Thêm thư viện spark

- Bước 2: Đọc dữ liệu đầu vào bằng SparkSession:

```
#Sử dụng biến SparkSession để đọc dữ liệu đầu vào từ tập tin csv.
spark = SparkSession.builder.appName("BigData").getOrCreate()
data = spark.read.format("csv")\
    .option("header", "true")\
    .option("inferSchema", "true")\
    .load("Preprocessed1.csv")
data.cache()
```

Hình 26. Đọc dữ liệu

- Bước 3: Hiển thị Schema dữ liệu

```
#Hiển thị schema dữ liệu  
data.printSchema()
```

```
root  
| -- Num_of_reviews: double (nullable = true)  
| -- General_rate: double (nullable = true)  
| -- Comfort: double (nullable = true)  
| -- Interior design: double (nullable = true)  
| -- Performance: double (nullable = true)  
| -- Value for the money: double (nullable = true)  
| -- Exterior styling: double (nullable = true)  
| -- Reliability: double (nullable = true)  
| -- Year: integer (nullable = true)  
| -- Brand: string (nullable = true)  
| -- Model: string (nullable = true)  
| -- Brand_encoded: double (nullable = true)  
| -- Model_encoded: double (nullable = true)  
| -- Year_encoded: double (nullable = true)
```

Hình 27. Schema dữ liệu

- Bước 4: Dưa cột cần dự đoán là Genneral_rate xuống cuối để làm Y.

```
data_indexed = data.select([col(c).alias(c) for c in data.columns if c != 'General_rate'] + ['General_rate'])
```

Hình 28. Dưa Genneral_rate ra cuối

- Bước 5: Chia tập dữ liệu thành tập huấn luyện và kiểm tra.

```
# Chia tập dữ liệu thành tập huấn luyện và kiểm tra  
train_data, test_data = data_indexed.randomSplit([0.8, 0.2], seed=42)
```

Hình 29. Chia tập dữ liệu

- Bước 6: Viết hàm tính ma trận $X^T \cdot X$.

```

def XTX(row):
    #Xử lý đầu vào: Hàm nhận một hàng giá trị làm đầu vào.
    value_X = np.array(row[1:], dtype='float')
    #Tạo ma trận: Mảng value_X sau đó được chuyển đổi thành một ma trận X bằng cách chèn một cột của số 1 vào đầu
    matX = np.matrix(value_X)
    X = np.insert(matX, 0, 1, axis=1)
    #Tính toán ma trận chuyển vị: Hàm này thực hiện phép chuyển vị trên ma trận X
    transpose_X = X.T
    #Tính toán tích của ma trận chuyển vị với chính nó:
    #Sau đó, hàm tính tích của ma trận chuyển vị transpose_X với chính nó, tạo ra ma trận kết quả XTX
    XTX = np.dot(transpose_X, X)
    return XTX

```

Hình 30. Hàm tính ma trận $X^T \cdot X$

- Bước 7: Viết hàm tính ma trận $X^T \cdot Y$.

```

def XTY(row):
    #Xử lý đầu vào: Hàm nhận một hàng giá trị làm đầu vào
    value_X = np.array(row[1:-1], dtype='float')
    #Trừ cột cuối cùng, được coi là giá trị mục tiêu (hoặc kết quả) của mô hình.
    value_Y = row[-1]
    #Tạo ma trận: Mảng value_X sau đó được chuyển đổi thành một ma trận X bằng cách chèn một cột của số 1 vào đầu
    matX = np.matrix(value_X)
    X = np.insert(matX, 0, 1, axis=1)
    #Tính toán ma trận chuyển vị của X: Hàm này thực hiện phép chuyển vị trên ma trận X
    transpose_X = X.T
    #Tạo ma trận Y: Giá trị đầu vào cuối cùng của hàng (được lấy từ row[-1]) được sử dụng làm giá trị của vector cột Y
    Y = np.matrix([value_Y])
    #Tính toán tích của ma trận chuyển vị X với Y: Hàm tính tích của ma trận chuyển vị transpose_X với vector cột Y,
    XTY = np.dot(transpose_X, Y)
    #Trả về kết quả: Kết quả cuối cùng là ma trận XTY, chứa các giá trị tích của ma trận chuyển vị X với vector cột Y
    return XTY

```

Hình 31. Hàm tính ma trận $X^T \cdot Y$

- Bước 8: Thêm Cột intercept có index = 1 vào selected_data và selected_data2 sau đó drop General_rate ở selected_data2.

```

from pyspark.sql.functions import lit

train_data1 = train_data.withColumn("intercept", lit(1.0))
selected_data = train_data1.select("intercept", train_data.columns[0], train_data.columns[1], train_data.columns[2], train_data.
selected_data2 = selected_data.drop("General_rate")

```

Hình 32. Thêm cột intercept

- Bước 9: Tính ma trận B dựa trên Hàm XTX và XTY trong đó dòng inverseXTX = np.linalg.inv(XTX) để tính $(X^T \cdot X)^{-1}$.

```

# Tính X^T*X
XTX = np.array(selected_data2.rdd.map(XTX).reduce(lambda a, b: np.add(a, b)))
inverseXTX = np.linalg.inv(XTX)
# Tính X^T*Y
XTY = np.array(selected_data.rdd.map(XTY).reduce(lambda a, b: np.add(a, b)))
# Kết quả
beta = np.dot(inverseXTX, XTY)
print("beta: ")
for coeff in beta:
    print(coeff)

[Stage 18:>                                (0 + 1) / 1]

beta:
[-0.0001889]
[-0.00301309]
[0.17346495]
[0.17405569]
[0.23374433]
[0.2226173]
[0.14654116]
[0.21685411]
[0.00085288]
[0.00048487]
[2.28417833e-05]

```

Hình 33. Tính ma trận beta

- Bước 10: Tính ma trận \hat{y} dựa trên kết quả của ma trận beta[] và các biến phụ thuộc của tập test_data .

```

dataf = test_data.drop("General_rate")
rdd = dataf.rdd
# Khởi tạo mảng kết quả trống
results = []
for row in rdd.collect():
    Num_of_reviews_index = row["Num_of_reviews"]
    Comfort_index = row["Comfort"]
    Interior_design_index = row["Interior design"]
    Performance_index = row["Performance"]
    Value_for_the_money_index = row["Value for the money"]
    Exterior_styling_index = row["Exterior styling"]
    Reliability_index = row["Reliability"]
    Brand_encoded_index = row["Brand_encoded"]
    Model_encoded_index = row["Model_encoded"]
    Year_encoded_index = row["Year_encoded"]
    result = (
        beta[0] +
        beta[1] * Num_of_reviews_index +
        beta[2] * Comfort_index +
        beta[3] * Interior_design_index +
        beta[4] * Performance_index +
        beta[5] * Value_for_the_money_index +
        beta[6] * Exterior_styling_index +
        beta[7] * Reliability_index +
        beta[8] * Brand_encoded_index +
        beta[9] * Model_encoded_index +
        beta[10] * Year_encoded_index
    )
    # Thêm kết quả tính toán vào mảng kết quả
    results.append(result)
    print("Result:", result)

resultss = []

# Lặp qua từng mảng trong danh sách results và trích xuất giá trị đầu tiên từ mỗi mảng
for arr in results:
    resultss.append(arr[0])

```

Hình 34. Tính ma trận \hat{y}

Kết quả dự đoán sau khi được chuẩn hóa về dạng ban đầu:

```

Out[27]: array([1.15729582, 2.84296963, 2.75543514, 3.88858812, 3.8786639 ,
   4.20864843, 4.69557457, 4.66732259, 4.68728054, 4.68696083,
   4.78905849, 4.8491078 , 4.16736051, 4.34089188, 4.68521082,
   4.84716955, 4.65769214, 4.83918713, 4.83956222, 4.83995471,
   4.68993701, 4.85216653, 4.83912995, 4.94108412, 5.00079959,
   5.00103504, 5.00096707, 5.0011424 , 5.00110944, 5.00127172,
   5.00118145, 5.00137054, 5.00154668, 5.00153415, 5.00152057,
   5.00195604, 3.88066886, 4.05821053, 4.05893562, 4.11173407,
   3.85925064, 4.10233355, 4.1906669 , 4.51802505, 3.68760457,
   4.19310949, 4.35912519, 4.76775688, 4.84771128, 4.7632876 ,
   4.83287248, 4.68107893, 4.76940575, 4.92425052, 4.90995759,
   4.83937111, 4.84602474, 4.92018903, 4.92690846, 5.00100361,
   5.00109967, 5.00110701, 5.00123592, 5.0015577 , 5.0015258 ,
   5.00138528, 3.75223826, 4.04664992, 4.54683871, 4.18579248,
   4.19728845, 4.20100904, 4.73855647, 4.46768973, 4.58202986,
   4.52768725, 4.30409536, 4.81023918, 4.79627983, 4.86113748,
   4.86502074, 4.909702 , 4.85159426, 4.95495844, 4.65726938,
   4.87378481, 4.84348165, 4.88800573, 4.95219209, 4.95226105,
   4.95267777, 4.95266471, 4.95294297, 4.95309579, 5.0007939 ,
   ...])

```

Hình 35. Kết quả sau khi dự đoán

- Bước 11: Tính toán độ đo để đánh giá thuật toán .

```

from math import sqrt # Import sqrt for RMSE calculation
actuals = test_data.select('General_rate').rdd.flatMap(lambda x: x).collect()
def calculate_metrics(p_data, data):
    predictions = p_data
    actuals = data

    # Tính RMSE (Calculate RMSE)
    mse = sum([(p - a) ** 2 for p, a in zip(predictions, actuals)]) / len(actuals)
    rmse = sqrt(mse)

    # Tính MAE (Calculate MAE)
    mae = sum([abs(p - a) for p, a in zip(predictions, actuals)]) / len(actuals)

    # Tính R^2
    mean_actuals = sum(actuals) / len(actuals)
    total_sum_of_squares = sum((a - mean_actuals) ** 2 for a in actuals)
    residual_sum_of_squares = sum((p - a) ** 2 for p, a in zip(predictions, actuals))

    r_squared = 1 - (residual_sum_of_squares / total_sum_of_squares)

    return rmse, mae, r_squared

# Usage example:
mean, std = calculate_mean_std(actualsor)
original_values = convert_to_original(actuals, mean, std)
original_values_results = convert_to_original(resultss, mean, std)
rmse, mae, r_squared = calculate_metrics(original_values_results, original_values)
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"R²: {r_squared}")

```

Hình 36. Tính độ đo.

Kết quả dự đoán sau khi được chuẩn hóa về dạng ban đầu:

```

RMSE: 0.033171892833828036
MAE: 0.024764640904800798
R2: 0.987699165877885

```

Hình 37. Kết quả độ đo

- Bước 12: Trực quan hóa dữ liệu dự đoán so với dữ liệu gốc .

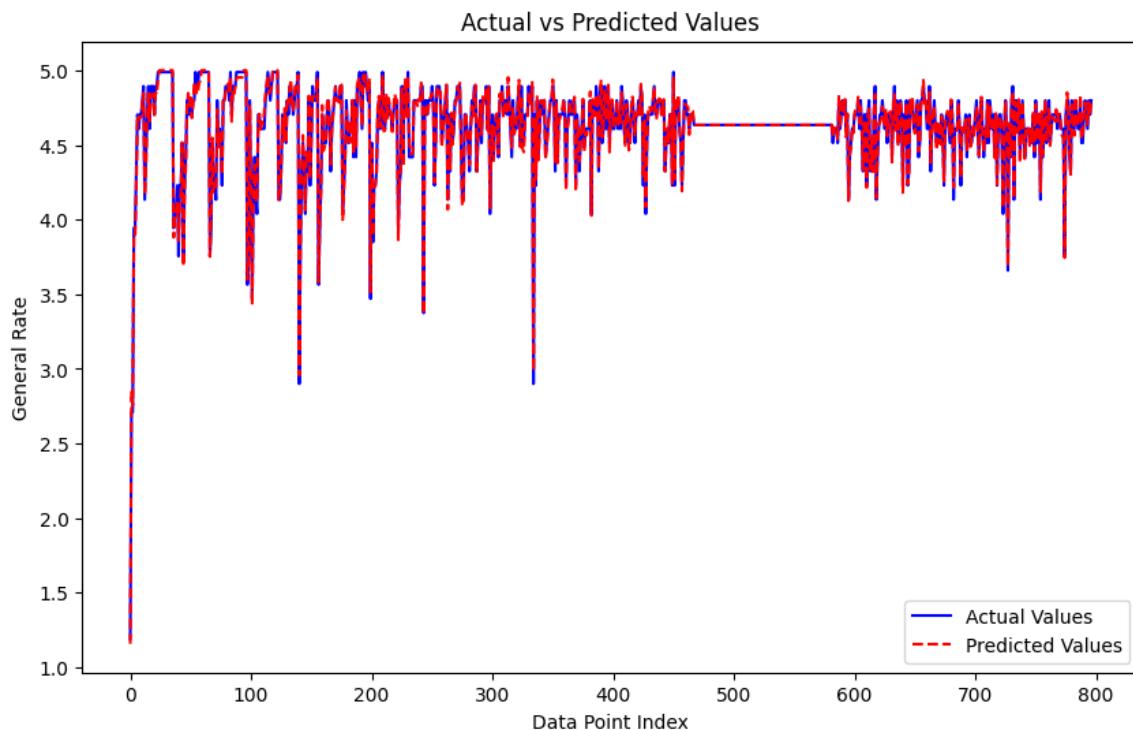
```

#visual
import matplotlib.pyplot as plt
# Trực quan hóa kết quả
plt.figure(figsize=(10, 6))
plt.plot(original_values, label='Actual Values', color='b')
plt.plot(original_values_results, label='Predicted Values', color='r', linestyle='--')
plt.xlabel('Data Point Index')
plt.ylabel('General Rate')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.show()

```

Hình 38. Tính độ đo.

Kết quả trực quan:



Hình 39. Kết quả trực quan

3.2. GBM

Thuật toán Gradient Boosting Machines mạnh mẽ cho việc dự đoán, đặc biệt với các vấn đề phân loại.

Lý do chọn thuật toán GBM:

- Hiệu suất cao: kết hợp nhiều mô hình đơn giản để tạo mô hình mạnh mẽ
- Khả năng điều chỉnh: điều chỉnh tham số như số lượng cây, độ sâu, tỷ lệ học hóa
- Khả năng giải thích: các mô hình trên cây có thể được giải thích tương đối dễ hiểu qua việc xem các đặc trưng của cây.[2]

Lập trình GBM trên Apache Spark:

- Bước 1: Thêm các thư viện cần thiết [2]

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, mean, count, desc, min, max
import matplotlib.pyplot as plt
```

Hình 40. Thêm thư viện spark

- Bước 2: Đọc dữ liệu đầu vào bằng SparkSession

```
#Sử dụng SparkSession để đọc dữ liệu đầu vào
spark = SparkSession.builder.appName("GBM").getOrCreate()

data = spark.read.format("csv")\
    .option("header", "true")\
    .option("inferSchema", "true")\
    .load("Preprocessed1.csv")
data.cache()
data.count()
```

Hình 41. Đọc dữ liệu

- Bước 3: Hiển thị Schema dữ liệu

```
#Hien thi schema du lieu
data.printSchema()

root
|-- Num_of_reviews: double (nullable = true)
|-- General_rate: double (nullable = true)
|-- Comfort: double (nullable = true)
|-- Interior design: double (nullable = true)
|-- Performance: double (nullable = true)
|-- Value for the money: double (nullable = true)
|-- Exterior styling: double (nullable = true)
|-- Reliability: double (nullable = true)
|-- Year: integer (nullable = true)
|-- Brand: string (nullable = true)
|-- Model: string (nullable = true)
|-- Brand_encoded: double (nullable = true)
|-- Model_encoded: double (nullable = true)
|-- Year_encoded: double (nullable = true)
```

Hình 42. Schema dữ liệu

- Bước 4: Import thư viện toán, thêm các hàm xử lý tính toán Gradient, dự đoán và cập nhật cây

```

from math import exp, sqrt

def sigmoid(x):
    return 1 / (1 + exp(-x))

def compute_gradient(tree, data):
    gradients = []
    for row in data:
        features = row[:-1]
        label = row[-1]
        prediction = predict(tree, features)
        gradient = label - prediction
        gradients.append((features, gradient))
    return gradients

```

Hình 43. Hàm xử lý tính toán gradient

```

def predict(tree, features):
    node = tree
    while isinstance(node, dict):
        if features[node['feature_index']] < node['threshold']:
            node = node['left']
        else:
            node = node['right']
    return node

def update_tree(tree, gradients, learning_rate):
    for features, gradient in gradients:
        node = tree
        while isinstance(node, dict):
            if features[node['feature_index']] < node['threshold']:
                node = node['left']
            else:
                node = node['right']
        node += learning_rate * gradient

```

Hình 44. Hàm dự đoán và cập nhật cây và các nodes

- Bước 5: Xây dựng cây quyết định đơn giản

```

# Xây dựng
def build_tree(data, depth=1, max_depth=3):
    if depth == max_depth or len(data) == 0:
        return sum([row[1] for row in data]) / len(data) if len(data) > 0 else 0

    feature_index = 0
    threshold = sum([row[0][feature_index] for row in data]) / len(data)

    left_data = [row for row in data if row[0][feature_index] < threshold]
    right_data = [row for row in data if row[0][feature_index] >= threshold]

    return {
        'feature_index': feature_index,
        'threshold': threshold,
        'left': build_tree(left_data, depth + 1, max_depth),
        'right': build_tree(right_data, depth + 1, max_depth)
    }

```

Hình 45. Xây dựng cây đơn giản

- Bước 6: Hàm train_gbm để huấn luyện GBM, thực hiện tính toán và cập nhật

```

#Huấn luyện
def train_gbm(data, num_trees=10, learning_rate=0.1, max_depth=3):
    trees = []
    predictions = [0.0] * len(data)

    for _ in range(num_trees):
        # Tính toán gradient
        gradients = []
        for i, row in enumerate(data):
            label = row[-1]
            gradient = label - sigmoid(predictions[i])
            gradients.append((row[:-1], gradient))

        # Huấn luyện cây trên gradient
        tree = build_tree(gradients, max_depth=max_depth)
        trees.append(tree)

        # Cập nhật dự đoán
        for i, row in enumerate(data):
            predictions[i] += learning_rate * predict(tree, row[:-1])

    return trees

```

Hình 46. Hàm huấn luyện GBM

- Bước 7: Chuyển đổi DataFrame, setup các thông số huấn luyện, loại bỏ các giá trị None trong dữ liệu và dự đoán

```

# Chuyển đổi DataFrame sang RDD
data_rdd = data.rdd.map(lambda row: (
    row['Num_of_reviews'], row['General_rate'], row['Comfort'], row['Interior design'],
    row['Performance'], row['Value for the money'], row['Exterior styling'],
    row['Reliability'], row['Year'], row['Brand'], row['Model'], row['General_rate']
))

# Huấn luyện mô hình GBM
num_trees = 10
learning_rate = 0.1
max_depth = 3

data_list = data_rdd.collect()
# Loại bỏ các giá trị None trong dữ liệu
clean_data = [row for row in data_list if None not in row]

trees = train_gbm(clean_data, num_trees=num_trees, learning_rate=learning_rate, max_depth=max_depth)

# Dự đoán
def predict_gbm(trees, features):
    prediction = 0
    for tree in trees:
        prediction += predict(tree, features)
    return sigmoid(prediction)

```

Hình 47. Chuyển đổi và setup thông số mô hình

- Bước 8: Tính toán các độ đo lỗi

```

def calculate_metrics(trees, data):
    predictions = []
    actuals = []

    for row in data:
        features = row[:-1]
        actual = row[-1]
        prediction = predict_gbm(trees, features)
        predictions.append(prediction)
        actuals.append(actual)

```

Hình 48. Hàm tính toán độ đo

```

# Tính RMSE
mse = sum([(p - a) ** 2 for p, a in zip(predictions, actuals)]) / len(actuals)
rmse = sqrt(mse)

# Tính MAE
mae = sum([abs(p - a) for p, a in zip(predictions, actuals)]) / len(actuals)

#Tính R^2
mean_actual = sum(actuals) / len(actuals)
ss_total = sum([(a - mean_actual) ** 2 for a in actuals])
ss_residual = sum([(a - p) ** 2 for a, p in zip(actuals, predictions)])
r2 = 1 - (ss_residual / ss_total)

return rmse, mae, r2

rmse, mae, r2 = calculate_metrics(trees, clean_data)
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
print(f"R^2: {r2}%")

```

Hình 49. Thông số độ đo RMSE, MAE, R²

RMSE: 0.9998410073086619
MAE: 0.6006477191004613
R²: 8.229719975061389e-05%

Hình 50. Kết quả thông số

- Bước 9: Visualize bằng biểu đồ histogram

```

# Chuyển đổi DataFrame sang RDD
data_rdd = data.rdd.map(lambda row: (
    row['Num_of_reviews'], row['General_rate'], row['Comfort'], row['Interior design'],
    row['Performance'], row['Value for the money'], row['Exterior styling'],
    row['Reliability'], row['Year'], row['Brand'], row['Model'], row['General_rate']
))

# Thu thập dữ liệu từ RDD
data_list = data_rdd.collect()

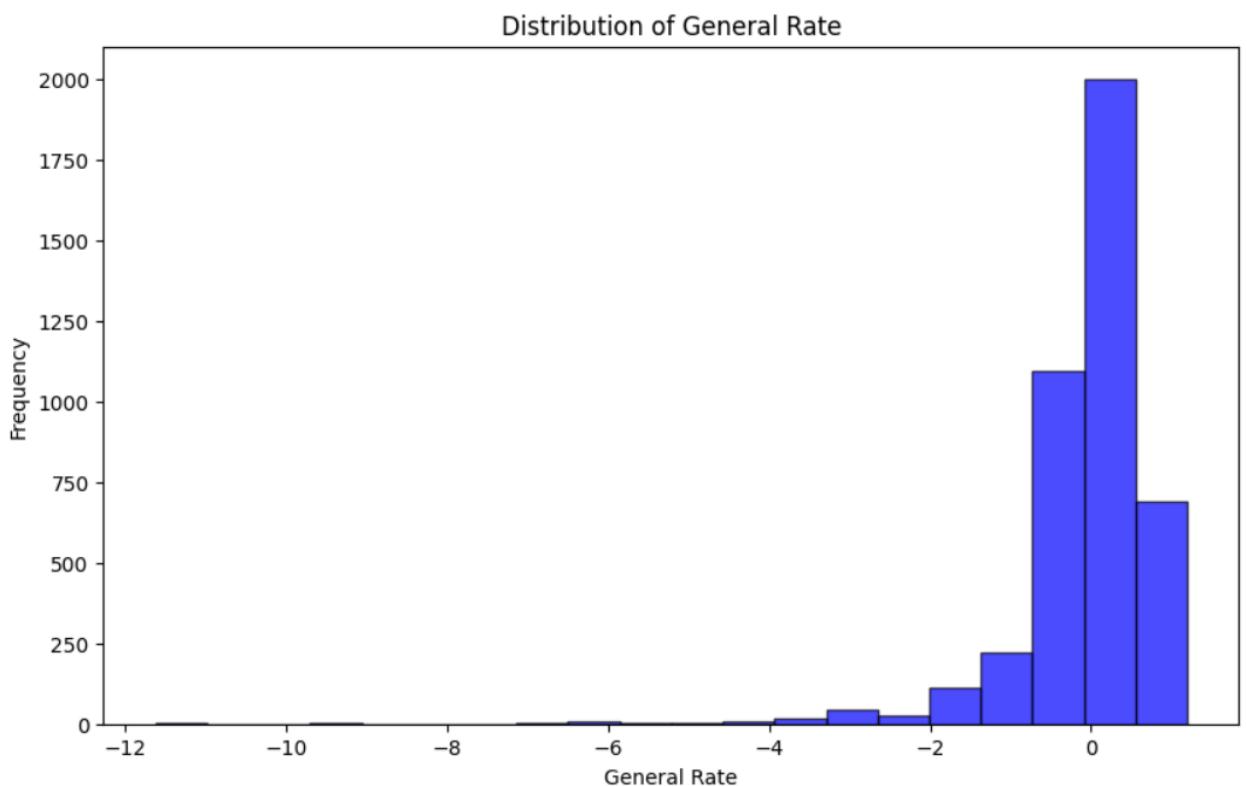
# Loại bỏ các giá trị None trong dữ liệu
clean_data = [row for row in data_list if None not in row]

# Lấy giá trị của General_rate
general_rates = [row[1] for row in clean_data]

# Biểu đồ histogram của General_rate
plt.figure(figsize=(10, 6))
plt.hist(general_rates, bins=20, alpha=0.7, color='b', edgecolor='black')
plt.xlabel('General Rate')
plt.ylabel('Frequency')
plt.title('Distribution of General Rate')
plt.show()

```

Hình 51. Phác họa biểu đồ



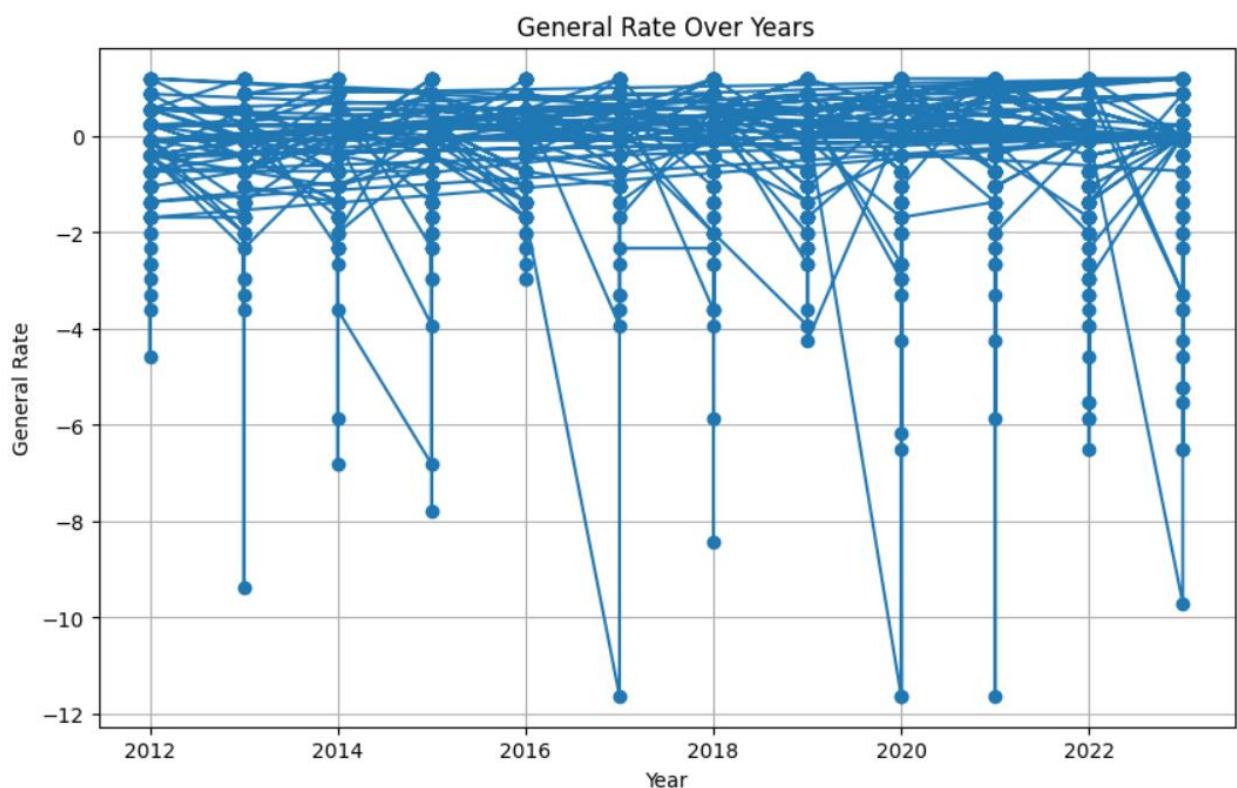
Hình 52. Kết quả

Biểu đồ cột thể hiện phân phối của General Rate qua tần suất. Biểu đồ cho thấy phân phối của tỷ lệ tổng quát với tần suất tương ứng, cho thấy phân phối lệch về phía bên phải gần giá trị 0, cho thấy tần suất cao liên quan đến giá trị gần 0

- Bước 10: Visualize bằng biểu đồ đường

```
# Biểu đồ line plot của General_rate theo năm
plt.figure(figsize=(10, 6))
plt.plot(years, general_rates, marker='o', linestyle='--')
plt.xlabel('Year')
plt.ylabel('General Rate')
plt.title('General Rate Over Years')
plt.grid(True)
plt.show()
```

Hình 53. Phác họa biểu đồ



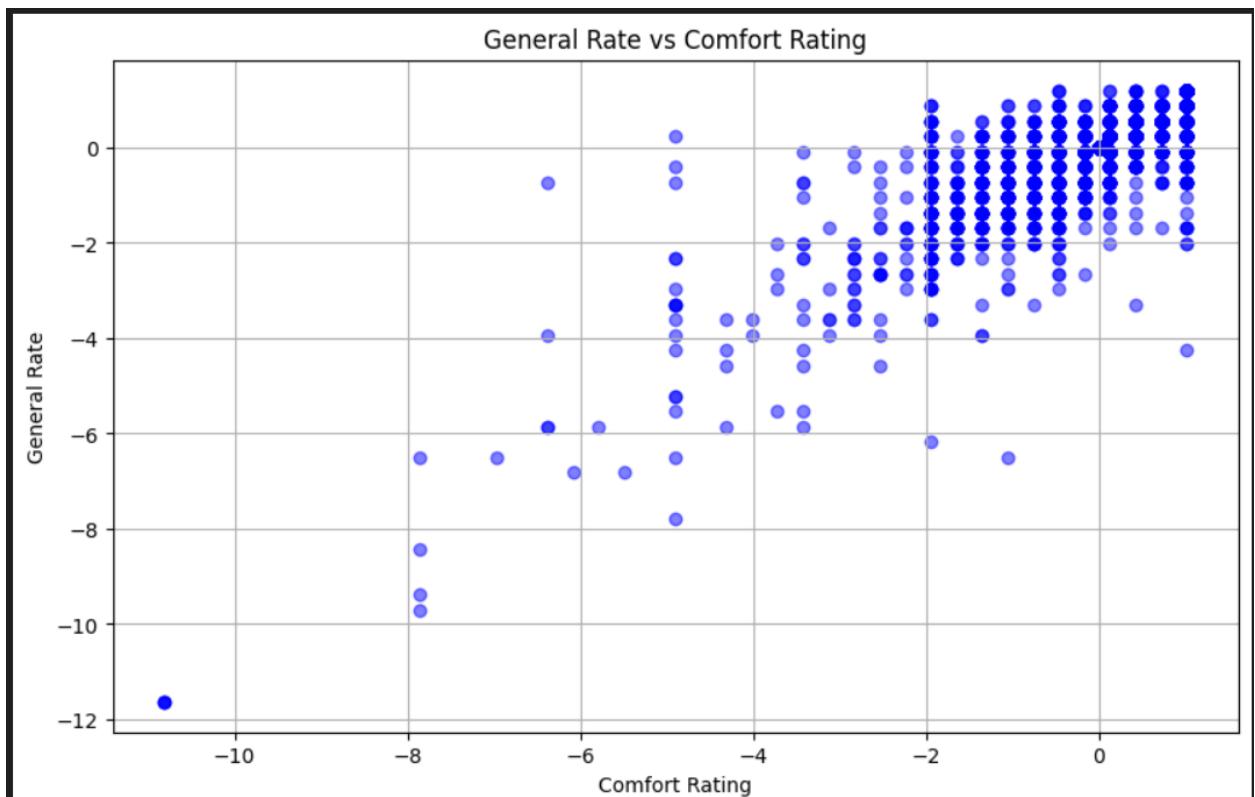
Hình 54. Kết quả

Biểu đồ đường (line plot) cho thấy xu hướng tổng quan giảm giá xe hơi theo thời gian và có sự biến động. Điều này có thể dự đoán giá trong tương lai, quyết định sản xuất chiến lược.

- Bước 11: Visualize bằng biểu đồ phân tán

```
# Biểu đồ scatter plot của General_rate và Comfort
plt.figure(figsize=(10, 6))
plt.scatter(comfort_ratings, general_rates, color='b', alpha=0.5)
plt.xlabel('Comfort Rating')
plt.ylabel('General Rate')
plt.title('General Rate vs Comfort Rating')
plt.grid(True)
plt.show()
```

Hình 55. Phác họa biểu đồ



Hình 56. Kết quả

Biểu đồ phân tán (Scatter plot) liên quan đến việc phân tích Comfort Rating ảnh hưởng đến General Rate như đánh giá sản phẩm,... Cho thấy Comfort Rating cao thường có General Rate cao, tập trung ở khu vực -2 đến 0

3.3. KMeans

3.3.1. Lý do chọn lựa phương pháp KMeans

KMeans là một thuật toán phân cụm phổ biến được sử dụng để chia các điểm dữ liệu thành các nhóm (cluster) dựa trên khoảng cách của chúng đến các centroid (tâm cụm). Đối với tập dữ liệu Car_Rates, phương pháp này hữu ích để phân nhóm các xe hơi dựa trên các thuộc tính như "General_rate", "Comfort", "Performance", "Reliability" và các yếu tố khác, giúp nhận diện các mẫu xe có đặc điểm tương tự nhau.

Cách thực hiện thuật toán khai thác dữ liệu

Chia dữ liệu thành training và test set:

```
# Chia dữ liệu thành training và test set
(train_data, test_data) = data.randomSplit([0.7, 0.3], seed=42)
```

Hình 57. Chia dữ liệu

Chuyển đổi DataFrame thành RDD để sử dụng takeSample:

```
# Chuyển đổi DataFrame thành RDD để sử dụng takeSample
train_rdd = train_data.select(feature_columns).rdd.map(lambda row: [row[col] for col in feature_columns])
test_rdd = test_data.select(feature_columns).rdd.map(lambda row: [row[col] for col in feature_columns])
```

Hình 58. Chuyển đổi DataFrame

Hàm tính khoảng cách Euclidean:

```
def euclidean_distance(row, centroid):
    return sqrt(sum((row[i] - centroid[i]) ** 2 for i in range(len(row))))
```

Hình 59. Hàm tính khoảng cách

Hàm gán cụm cho từng điểm dữ liệu:

```
def assign_cluster(row, centroids):
    distances = [euclidean_distance(row, centroid) for centroid in centroids]
    return distances.index(min(distances))
```

Hình 60. Hàm gán cụm

3.3.2. Phương Pháp Elbow và Within-Cluster Sum of Squares (WSS)

Phương pháp Elbow là một kỹ thuật phổ biến để xác định số lượng cụm tối ưu trong bài toán phân cụm KMeans. Nguyên lý cơ bản của phương pháp này là chạy thuật toán KMeans với các giá trị khác nhau của k (số lượng cụm) và tính toán tổng độ chênh trong cụm (Within-Cluster Sum of Squares, WSS) cho mỗi giá trị k. Sau đó, ta vẽ biểu đồ WSS theo

các giá trị k và tìm "khuỷu tay" (elbow) trên biểu đồ - điểm mà sau đó WSS giảm chậm lại. Giá trị k tại điểm elbow được chọn là số lượng cụm tối ưu.[3]

WSS là tổng các khoảng cách bình phương từ mỗi điểm dữ liệu đến centroid của cụm mà nó thuộc về. Nó là một thước đo để đánh giá mức độ chật chẽ của các cụm dữ liệu. Cụm càng chật chẽ thì giá trị WSS càng nhỏ. Công thức tính WSS như sau:[4]

$$WSS = \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2$$

trong đó:

- k là số lượng cụm.
- C_i là tập hợp các điểm dữ liệu trong cụm i .
- μ_i là centroid của cụm i .
- $\|x - \mu_i\|$ là khoảng cách Euclidean giữa điểm dữ liệu x và centroid μ_i .

Sử Dụng Phương Pháp Elbow và WSS Trong Bài Toán

Chạy thuật toán KMeans với các giá trị k khác nhau: Chạy thuật toán KMeans với các giá trị k từ 1 đến một giá trị lớn hơn, ví dụ như 10.

```
# Chạy K-means với các giá trị k khác nhau và tính WSS
k_values = range(1, 10)
wss_values = [kmeans(train_rdd, k)[1] for k in k_values]
```

Hình 61. Chạy K-means với các giá trị k khác nhau

Tính toán WSS cho mỗi giá trị k: Với mỗi giá trị k, tính toán WSS để đánh giá mức độ chật chẽ của các cụm.

```
def compute_wss(data_rdd, centroids):
    return data_rdd.map(lambda row: min([euclidean_distance(row, centroid) ** 2 for centroid in centroids])).sum()
```

Hình 62. Tính toán WSS

Vẽ biểu đồ Elbow: Vẽ biểu đồ WSS theo các giá trị k để tìm điểm "khuỷu tay".

```

# Vẽ biểu đồ Elbow
plt.plot(k_values, wss_values, 'bx-')
plt.xlabel('k')
plt.ylabel('WSS')
plt.title('Elbow Method For Optimal k')
plt.show()

```

Hình 63. Vẽ biểu đồ Elbow

Xác định số lượng cụm tối ưu: Tìm giá trị k tại điểm elbow trên biểu đồ - điểm mà sau đó WSS giảm chậm lại.

```

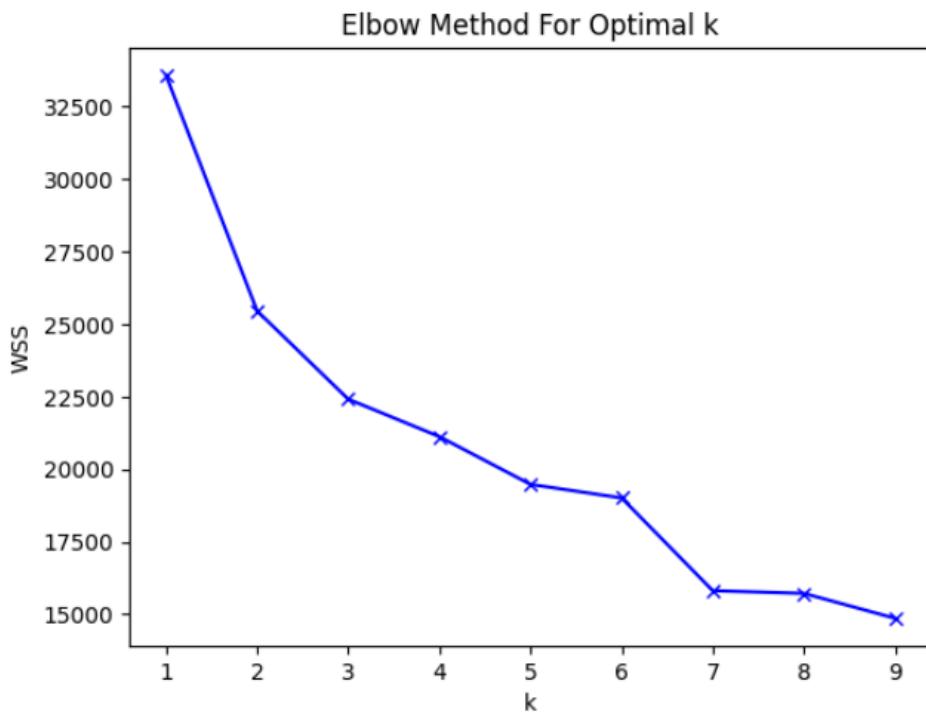
# Tìm k tối ưu dựa trên phương pháp Elbow
optimal_k = None
min_diff = float('inf')

for i in range(1, len(wss_values) - 1):
    diff = wss_values[i - 1] - wss_values[i]
    if diff < min_diff:
        min_diff = diff
        optimal_k = k_values[i]

print("Optimal k value based on Elbow Method:", optimal_k)

```

Hình 64. Tìm k tối ưu



Optimal k value based on Elbow Method: 8

Hình 65. Kết quả

Sau khi xác định được số lượng cụm tối ưu (optimal k), ta có thể chạy thuật toán KMeans với k tối ưu đó để phân cụm dữ liệu.

3.3.3. KMeans Song Song và Không Song Song

Hàm KMeans Song Song

Hàm KMeans song song được thiết kế để tận dụng lợi thế của Spark nhằm xử lý dữ liệu lớn bằng cách phân tán công việc qua nhiều nút trong một cụm (cluster) máy tính.

```
def kmeans(data_rdd, k, max_iterations=10):
    centroids = data_rdd.takeSample(False, k)

    for _ in range(max_iterations):
        clusters = data_rdd.map(lambda row: (assign_cluster(row, centroids), row)).groupByKey().mapValues(list)
        new_centroids = clusters.mapValues(lambda points: [sum([p[i] for p in points]) / len(points) for i in range(len(points[0]))])
        centroids = list(new_centroids.collectAsMap().values())

    wss = compute_wss(data_rdd, centroids)
    return centroids, wss
```

Hình 66. Hàm K-means song song

Cách thực hiện:

1. **Khởi tạo centroids:** Lấy mẫu ngẫu nhiên từ dữ liệu ban đầu để chọn các centroids khởi tạo.
2. **Gán cụm:** Sử dụng **map** để gán mỗi điểm dữ liệu vào cụm gần nhất. **groupByKey** nhóm các điểm dữ liệu theo cụm và **mapValues** để biến đổi các điểm dữ liệu trong mỗi cụm thành một danh sách.
3. **Tính toán centroids mới:** Với mỗi cụm, tính toán centroid mới bằng cách lấy trung bình của tất cả các điểm dữ liệu trong cụm đó.
4. **Lặp lại:** Quá trình gán cụm và tính toán lại centroids được lặp lại trong một số lần xác định (**max_iterations**).
5. **Tính toán WSS:** Tổng hợp bình phương khoảng cách từ mỗi điểm dữ liệu đến centroid của cụm của nó để tính WSS.

Hàm KMeans Không Song Song

Hàm KMeans không song song không sử dụng Spark, thay vào đó xử lý toàn bộ dữ liệu trên một máy tính duy nhất. Dữ liệu được lưu trong một danh sách (list) và tất cả các phép tính đều được thực hiện tuần tự.

```

# Hàm KMeans không song song
def kmeans_non_parallel(points, k, max_iterations):
    centers = random.sample(points, k)

    for _ in range(max_iterations):
        clusters = [[] for _ in range(k)]
        for point in points:
            center_idx = assign_cluster(point, centers)
            clusters[center_idx].append(point)

        new_centers = []
        for cluster in clusters:
            if cluster:
                new_center = [sum(dim) / len(cluster) for dim in zip(*cluster)]
                new_centers.append(new_center)
            else:
                new_centers.append(random.choice(points))

        centers = new_centers

    return centers, clusters

```

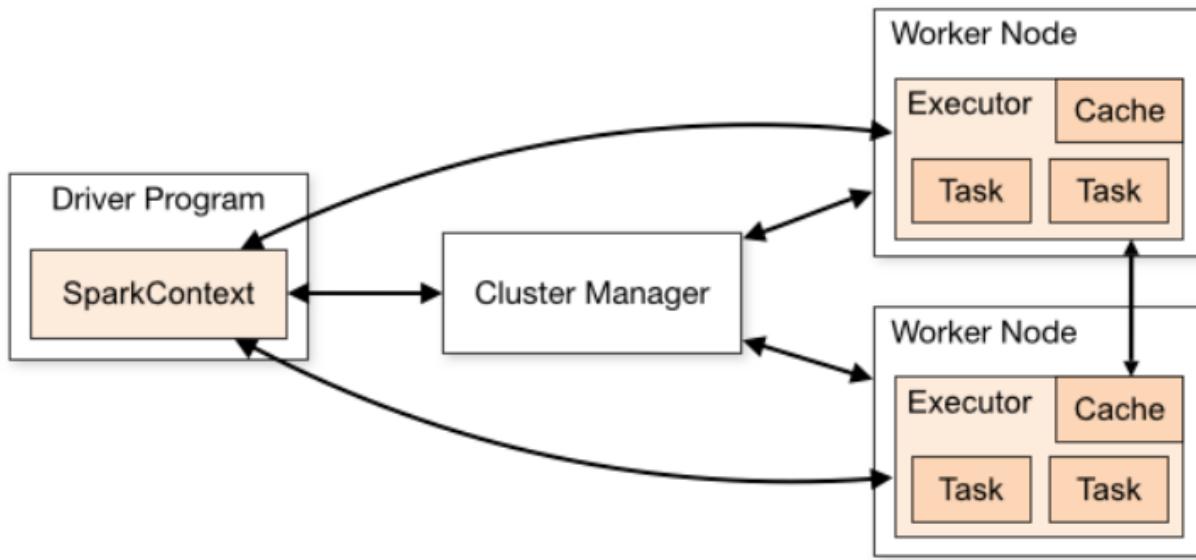
Hình 67. Hàm K-means không song song

Cách thực hiện:

- Khởi tạo centroids:** Lấy mẫu ngẫu nhiên từ dữ liệu ban đầu để chọn các centroids khởi tạo.
- Gán cụm:** Mỗi điểm dữ liệu được gán vào cụm gần nhất bằng cách tính toán khoảng cách Euclidean và chọn centroid gần nhất.
- Tính toán centroids mới:** Với mỗi cụm, tính toán centroid mới bằng cách lấy trung bình của tất cả các điểm dữ liệu trong cụm đó.
- Lặp lại:** Quá trình gán cụm và tính toán lại centroids được lặp lại trong một số lần xác định (**max_iterations**).

3.4. Ứng dụng Spark Cluster trong Data Mining

Apache Spark là một framework xử lý dữ liệu lớn phân tán, được thiết kế để xử lý nhanh và hiệu quả các tập dữ liệu lớn. Trong bối cảnh phân cụm KMeans, Spark Cluster giúp tăng tốc quá trình tính toán nhờ vào khả năng xử lý phân tán và song song.[5]



Hình 68. Spark Cluster Diagram

Cách Cấu Hình và Cài Đặt Spark Cluster

Để cài đặt và cấu hình một Spark Cluster, cần thiết lập các biến môi trường và chạy các dịch vụ Master và Worker. Dưới đây là các bước cụ thể:

Cài Đặt Apache Spark:

Tải Apache Spark từ trang chủ và giải nén.

Đảm bảo đã cài đặt Java Development Kit (JDK)

Cấu Hình Biến Môi Trường:

Tạo tệp **spark-env.sh** trong thư mục **conf** của Spark và thêm các dòng cấu hình sau:

```

spark-env.sh
~/spark/conf
spark-env.sh x
spark-defaults.conf x
1 PYSPARK_PYTHON=python3
2 PYSPARK_DRIVER_PYTHON=jupyter
3 PYSPARK_DRIVER_PYTHON_OPTS=notebook
4 export SPARK_MASTER_HOST='hien-VirtualBox'
5 export SPARK_WORKER_CORES=4
6 export SPARK_WORKER_MEMORY=8g
7 export SPARK_WORKER_PORT=8081
8 export SPARK_WORKER_INSTANCES=2

```

Hình 69. Cấu hình biến môi trường

Tạo tệp **spark-defaults.conf** trong thư mục **conf** của Spark và thêm các dòng cấu hình sau:

```
spark-defaults.conf
~/spark/conf
Save
spark-env.sh
spark-defaults.conf
x
x
1 # spark-defaults.conf
2 spark.master          spark://hien-VirtualBox:7077
3 spark.executor.memory 4g
4 spark.executor.cores   2
5 spark.driver.memory    2g
6 spark.driver.cores     1
7 spark.task.cpus        1
8 spark.sql.shuffle.partitions 100
9
```

Hình 70. Cấu hình config

Chạy Master và Worker:

Chạy Master bằng lệnh: sbin/start-master.sh

Master sẽ chạy và có thể truy cập thông qua giao diện web tại:

<http://hien-VirtualBox:8080>.

Chạy Worker bằng lệnh: sbin/start-worker.sh spark://hien-VirtualBox:7077

Đây là địa chỉ Master URL mà Worker sẽ kết nối tới.

Sau khi chạy xong có thể kiểm tra master và workers trên trang web:

The screenshot shows the Apache Spark Master UI at `localhost:8080`. The title bar says "Spark Master at spark://hien-VirtualBox:7077". The page displays the following information:

- URL:** `spark://hien-VirtualBox:7077`
- Alive Workers:** 2
- Cores in use:** 8 Total, 0 Used
- Memory in use:** 16.0 GiB Total, 0.0 B Used
- Resources in use:**
- Applications:** 0 Running, 21 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20240603235324-10.0.3.15-8081	10.0.3.15:8081	ALIVE	4 (0 Used)	8.0 GiB (0.0 B Used)	
worker-20240603235330-10.0.3.15-8083	10.0.3.15:8083	ALIVE	4 (0 Used)	8.0 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (21)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20240604113538-0020	PySparkShell	8	4.0 GiB		2024/06/04 11:35:38	hien	FINISHED	2.9 min
app-20240604102804-0019	PySparkShell	8	4.0 GiB		2024/06/04 10:28:04	hien	FINISHED	2.7 min

Hình 71. Kết quả hiển thị trên Spark Master

4. Kết quả đạt được

4.1. Phát biểu kết quả

4.1.1. Linear

Kết quả từ các độ đo:

Chạy Linear trên tập test:

- RMSE: 0.033171892833828036
- MAE: 0.024764640904800798
- R²: 0.987699165877885

Phát Biểu Kết Quả

1. Độ chính xác:

- **RMSE:** Giá trị RMSE thấp (0.033) cho thấy độ chính xác cao của mô hình. Sai số dự đoán trung bình trên toàn bộ dữ liệu là 0.033 đơn vị, tương đối nhỏ so với giá trị thực tế.
- **MAE:** MAE thấp (0.025) cũng cung cấp kết luận về độ chính xác cao. Sai số tuyệt đối trung bình cho mỗi dự đoán là 0.025 đơn vị, cho thấy mô hình ít sai lệch so với giá trị thực tế.

2. Mức độ giải thích:

- **R²:** Giá trị R² cao (0.9877) cho thấy mô hình có khả năng giải thích 98.77% biến động của biến mục tiêu. Điều này cho thấy mô hình có độ phù hợp tốt và các biến độc lập có mối quan hệ mạnh mẽ với biến phụ thuộc.

4.1.2. GBM

Kết quả các độ đo:

- RMSE: 0,9998410073086619
- MAE: 0.6006477191004613
- R²: 8.22971997506138-05%

Phát biểu kết quả:

1. Độ chính xác:

- RMSE: Giá trị khá cao cho thấy mô hình không phù hợp, sai số lên đến 0,99, tương đối lớn so với thực tế
- MAE: MAE nằm ở mức trung bình (0,6), sai số tuyệt đối trung bình trên mỗi dự đoán là 0,6, cho thấy mô hình sai lệch trung bình so với giá trị thực tế.

2. Mức độ giải thích:

- **R²:** Giá trị R² khá thấp (8.22-05%) cho thấy mô hình có khả năng giải thích biến động của biến mục tiêu khá thấp. Cho thấy mô hình này có độ phù hợp chưa tốt.

4.1.3. KMeans

Kết quả từ các độ đo:

Chạy KMeans trên tập training với k tối ưu và đánh giá trên tập test:

- **Train RMSE:** 2.318775686824766

- **Train MAE:** 2.079656445583553
- **Train R²:** -4.324164343791887
- **Test RMSE:** 2.3638399830709407
- **Test MAE:** 2.1001743783575733
- **Test R²:** -4.740477083207767

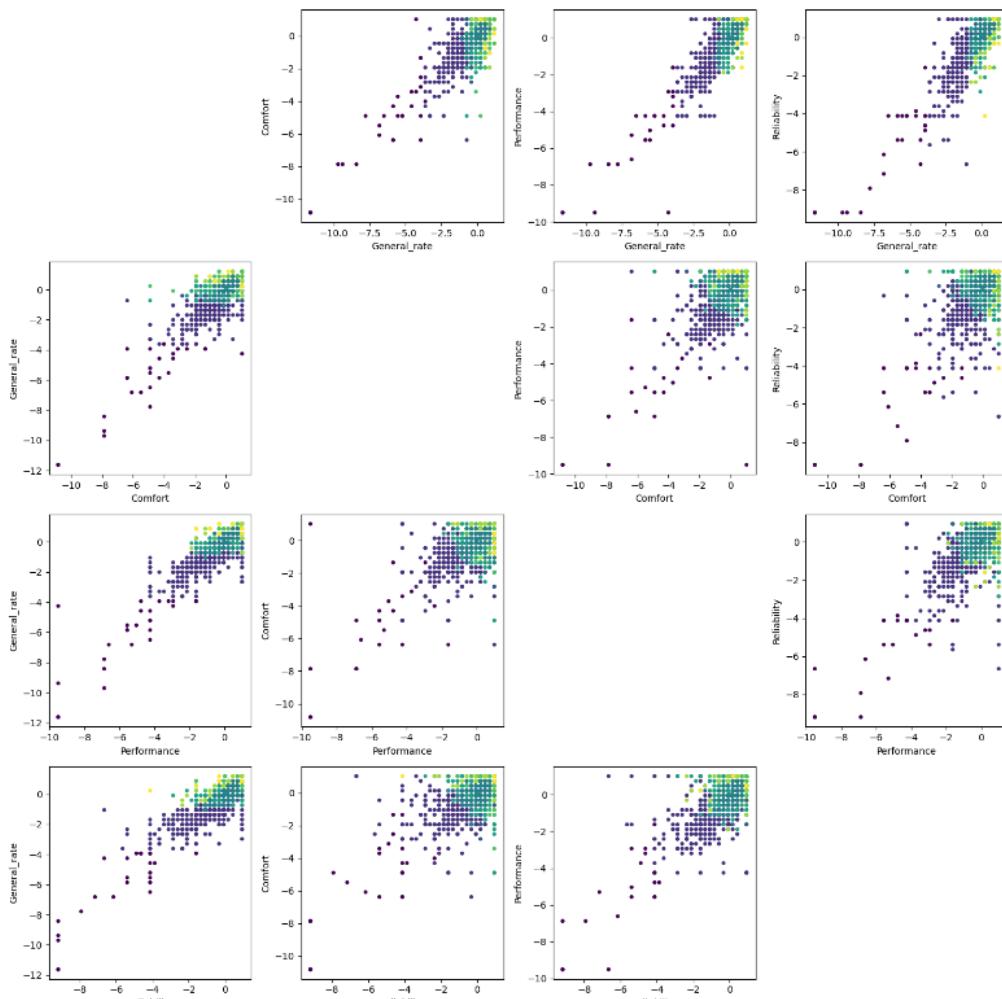
Chạy KMeans không song song:

- **Train RMSE (Non-Parallel):** 2.2342420918117973
- **Train MAE (Non-Parallel):** 2.0049339511492614
- **Train R² (Non-Parallel):** -3.9430435348156063
- **Test RMSE (Non-Parallel):** 2.2689364346645013
- **Test MAE (Non-Parallel):** 2.008581921309755
- **Test R² (Non-Parallel):** -4.288792121892889

So sánh thời gian thực thi giữa phiên bản song song và không song song của thuật toán KMeans.

- **Execution time:** 9.161375522613525 (song song)
- **Execution time (Non-Parallel):** 1.0981440544128418 (không song song)

Kết quả từ trực quan hoá phân cụm KMeans:



Hình 72. Kết quả trực quan K-means

Biểu đồ hiển thị là một ma trận các biểu đồ phân tán (scatter plot matrix) thể hiện mối quan hệ giữa các thuộc tính: General_rate, Comfort, Performance, và Reliability. Mỗi điểm dữ liệu trong các biểu đồ này tương ứng với một mẫu trong tập dữ liệu, và các màu sắc khác nhau biểu thị các cụm khác nhau được xác định bởi thuật toán KMeans.

Phát Biểu Kết Quả

General_rate vs. Comfort:

Có một sự phân cụm rõ ràng giữa các điểm, với đa số các điểm tập trung tại khu vực có giá trị General_rate và Comfort cao.

Các cụm khác nhau được phân biệt tốt, mặc dù có một số điểm nằm xa nhóm chính.

General_rate vs. Performance:

Tương tự, có một sự phân cụm rõ ràng với đa số các điểm tập trung ở khu vực có giá trị General_rate và Performance cao.

Sự phân bố các cụm tương đối rõ ràng, nhưng có một số điểm ngoài rìa có thể cần được xem xét.

General_rate vs. Reliability:

Mỗi quan hệ tương tự giữa General_rate và Reliability. Các cụm chính nằm ở góc phần tư phía trên bên phải của biểu đồ, cho thấy một mối quan hệ tích cực giữa General_rate và Reliability.

Comfort vs. Performance:

Các cụm chính nằm ở khu vực có giá trị Comfort và Performance cao, với một số điểm ngoài rìa.

Comfort vs. Reliability:

Mỗi quan hệ tương tự giữa Comfort và Reliability với các cụm chính nằm ở góc phần tư phía trên bên phải.

Performance vs. Reliability:

Mỗi quan hệ giữa Performance và Reliability cũng cho thấy một sự phân cụm rõ ràng, với các cụm chính nằm ở khu vực có giá trị cao cho cả hai thuộc tính.

4.1.4. Spark Cluster

Khi chạy các ứng dụng trên Spark Cluster với nhiều workers, ta có thể thấy sự cải thiện đáng kể về thời gian xử lý. Dưới đây là các kết quả ứng dụng với số worker khác nhau:

app-20240603234716-0015:

Cores: 12

Memory: 4.0 GiB

Thời gian: 4.1 phút

Kết quả: Hoàn thành

app-20240603234107-0014:

Cores: 8

Memory: 4.0 GiB

Thời gian: 3.5 phút

Kết quả: Hoàn thành

app-20240603224334-0013:

Cores: 4

Memory: 4.0 GiB

Thời gian: 8.0 phút

Kết quả: Hoàn thành

app-20240603235447-0016	PySparkShell	8	4.0 GiB		2024/06/03 23:54:47	hien	FINISHED	3.3 min
app-20240603234716-0015	PySparkShell	12	4.0 GiB		2024/06/03 23:47:16	hien	FINISHED	4.1 min
app-20240603234107-0014	PySparkShell	8	4.0 GiB		2024/06/03 23:41:07	hien	FINISHED	3.5 min
app-20240603224334-0013	PySparkShell	4	4.0 GiB		2024/06/03 22:43:34	hien	FINISHED	8.0 min
app-20240603223355-0012	PySparkShell	4	4.0 GiB		2024/06/03 22:33:55	hien	FINISHED	9.5 min

Hình 73. Kết quả hoàn thành

4.2. Độ đo

4.2.1. RMSE (Root Mean Squared Error):

$$\text{Công thức: } \text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (X_i - Y_i)^2}$$

Trong công thức trên:

- n là số lượng quan sát.
- Phần tử X_i là giá trị thứ i được dự đoán.
- Phần tử Y_i là giá trị thứ i thực tế.

Ý nghĩa: RMSE đo lường độ lớn trung bình của lỗi dự đoán bằng cách tính căn bậc hai của trung bình cộng bình phương sai số. Nó nhạy cảm với các lỗi lớn hơn, tức là các sai số lớn sẽ ảnh hưởng nhiều đến giá trị RMSE.

Lý do lựa chọn: RMSE cung cấp cái nhìn rõ ràng về độ lệch trung bình của dự đoán so với giá trị thực tế và rất hữu ích khi lỗi lớn cần được ưu tiên xem xét.

4.2.2. MAE (Mean Absolute Error):

Công thức: $MAE = \frac{1}{n} \sum_{i=1}^n |X_i - Y_i|$

Trong công thức trên:

- n là số lượng quan sát.
- Phần tử X_i là giá trị thứ i được dự đoán.
- Phần tử Y_i là giá trị thứ i thực tế.

Ý nghĩa: MAE đo lường độ lớn trung bình của lỗi dự đoán bằng cách tính trung bình cộng của các sai số tuyệt đối. Nó không nhạy cảm với các lỗi lớn như RMSE.

Lý do lựa chọn: MAE dễ hiểu và giải thích, phản ánh trung bình các sai số trong dự đoán mà không bị ảnh hưởng bởi các lỗi lớn bất thường.

4.2.3. R² (R-squared):

Công thức: $R^2 = 1 - \frac{\sum_{i=1}^n (X_i - Y_i)^2}{\sum_{i=1}^n (X_i - \bar{X})^2}$

Trong công thức trên:

- n là số lượng quan sát.
- Phần tử X_i là giá trị thứ i được dự đoán.
- Phần tử Y_i là giá trị thứ i thực tế.
- Phần tử \bar{X} là giá trị trung bình.

Ý nghĩa: R^2 đo lường tỷ lệ biến thiên của giá trị phụ thuộc (giá trị thực tế) được mô hình dự đoán giải thích. Nó nằm trong khoảng từ 0 đến 1, với giá trị 1 biểu thị mô hình hoàn hảo.

Lý do lựa chọn: R^2 cho phép đánh giá hiệu suất tổng thể của mô hình dự đoán, giúp hiểu rõ mức độ mô hình giải thích được sự biến thiên trong dữ liệu.

4.3. So sánh, đánh giá

4.3.1. So sánh Linear Regression và GBM

Có thể nói rằng thuật toán Linear Regression cho thấy kết quả của các độ đo chính xác hơn, phù hợp với mô hình hơn so với GBM. Ngoài ra kết quả của mức độ giải thích cao hơn rất nhiều so với GBM, cho thấy Linear có độ phù hợp tốt và đánh giá mạnh mẽ các biến độc lập → Kết luận Linear Regression là mô hình phù hợp cho dự đoán Car Rates này.

4.3.2. KMeans song song và không song song

Có một số lý do có thể giải thích tại sao phiên bản song song của KMeans có thể chậm hơn so với phiên bản không song song:

- **Overhead của Spark:** Spark thêm vào một số overhead khi quản lý các nhiệm vụ phân tán. Việc khởi tạo, điều phối, và thu thập kết quả từ nhiều nút có thể tốn thời gian hơn so với việc xử lý toàn bộ dữ liệu trên một máy tính duy nhất, đặc biệt là khi dữ liệu không quá lớn.
- **Kích thước dữ liệu nhỏ:** Nếu tập dữ liệu không quá lớn, lợi ích của việc song song hóa có thể bị giảm do overhead của Spark. Trong những trường hợp này, việc xử lý tuần tự có thể nhanh hơn.
- **Chi phí truyền thông:** Khi sử dụng Spark, dữ liệu phải được truyền giữa các nút trong cụm, điều này có thể tạo ra độ trễ và làm chậm quá trình tính toán.
- **Cấu hình và tài nguyên của cụm:** Nếu cụm Spark không được cấu hình tối ưu hoặc thiếu tài nguyên (như RAM hoặc CPU), thời gian xử lý có thể lâu hơn so với việc xử lý trên một máy tính mạnh mẽ.

4.3.3. Đánh giá hiệu quả Spark Cluster

Số Worker Cores và Thời Gian Xử Lý:

Các ứng dụng với số cores cao hơn (8, 12) cho thấy thời gian xử lý nhanh hơn so với ứng dụng chỉ có 4 cores.

Điều này là do khả năng xử lý song song của Spark. Khi số lượng cores tăng lên, nhiều tác vụ có thể được thực hiện đồng thời, do đó giảm thời gian xử lý tổng thể.

Hiệu Suất và Khả Năng Mở Rộng:

Khi tăng số lượng cores, thời gian xử lý giảm đi, nhưng không phải luôn luôn tỉ lệ thuận. Ví dụ, ứng dụng với 12 cores có thời gian xử lý dài hơn so với ứng dụng với 8 cores. Điều này có thể do hiệu suất bị giới hạn bởi các yếu tố khác như băng thông mạng, I/O hoặc độ phức tạp của tác vụ.

Hiệu suất cũng phụ thuộc vào cách dữ liệu được phân phối giữa các worker và overhead trong việc quản lý tài nguyên.

5. Kết luận

5.1. Ưu điểm

- Sử dụng mô hình hồi quy tuyến tính giúp đơn giản và dễ hiểu, dễ dàng triển khai và có hiệu quả tốt với dữ liệu tuyến tính.
- Phương pháp Elbow giúp xác định số lượng cụm tối ưu một cách trực quan.
- KMeans song song có khả năng xử lý tập dữ liệu lớn.

5.2. Hạn chế

- Giới hạn trong việc xử lý dữ liệu phi số.
- Overhead của Spark có thể làm chậm tốc độ xử lý so với phiên bản không song song, đặc biệt là với tập dữ liệu nhỏ.
- R^2 ám cho thấy các cụm không giải thích tốt biến động của dữ liệu.

5.3. Hướng phát triển

- Tối ưu hóa cấu hình cụm Spark để giảm overhead.
- Kiểm tra và xử lý các điểm ngoài rìa có thể giúp cải thiện chất lượng cụm KMeans.
- Thử nghiệm với các thuật toán phân cụm khác như DBSCAN hoặc Gaussian Mixture Models để xem liệu có thể cải thiện kết quả phân cụm KMeans.

6. Bảng phân công công việc

Thành viên	Công việc	Hoàn thành
Nguyễn Lê Thái Hiền	<ul style="list-style-type: none">• Tìm kiếm dữ liệu và bài toán• Viết báo cáo chương 1, 3• Thuật toán KMeans• Ứng dụng Spark Cluster• Quay Demo	100%

Đinh Võ Xuân Hoàn	<ul style="list-style-type: none"> Viết báo cáo chương 3, 4, 5 Thuật toán GBM Ứng dụng Spark Cluster 	100%
Dương Bảo Tâm	<ul style="list-style-type: none"> Viết báo cáo chương 1, 2, 4 Trực quan hóa dữ liệu Viết bản tóm tắt Làm slide 	100%
Trần Duy Khánh	<ul style="list-style-type: none"> Tìm kiếm dữ liệu và bài toán Thuật toán Linear Regression Viết báo cáo chương 3, 4 Làm slide 	100%

7. Tài liệu tham khảo

- [1] “What Is Linear Regression? | IBM.” Accessed: Jun. 04, 2024. [Online]. Available: <https://www.ibm.com/topics/linear-regression>
- [2] “Gradient Boosting: A Step-by-Step Guide.” Accessed: Jun. 04, 2024. [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/>
- [3] “Elbow Method for optimal value of k in KMeans,” GeeksforGeeks. Accessed: Jun. 04, 2024. [Online]. Available: <https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/>
- [4] H. J. Liang, “The number of clusters in the K-means and the within-cluster SS,” Cross Validated. Accessed: Jun. 04, 2024. [Online]. Available: <https://stats.stackexchange.com/q/542149>
- [5] “Cluster Mode Overview - Spark 3.5.1 Documentation.” Accessed: Jun. 04, 2024. [Online]. Available: <https://spark.apache.org/docs/latest/cluster-overview.html>