

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

BÁO CÁO TIỂU LUẬN

DỮ LIỆU LỚN

Đề tài:

**Nghiên cứu và triển khai các thuật toán phân cụm để dự báo
thời tiết sử dụng PySpark**

Giảng viên hướng dẫn:

Ths. Nguyễn Hồ Duy Trí

Sinh viên thực hiện:

Hồ Văn Vinh – 21520530

Hồ Quang Đỉnh - 21520190

Nguyễn Lý Đăng Khoa - 21522229

Trương Công Quốc Triệu – 215222714

Thành phố Hồ Chí Minh, tháng 6 năm 2024

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



BÁO CÁO TIỂU LUẬN

DỮ LIỆU LỚN

Đề tài:

**Nghiên cứu và triển khai các thuật toán phân cụm để dự báo
thời tiết sử dụng PySpark**

Giảng viên hướng dẫn:

Ths. Nguyễn Hồ Duy Trí

Sinh viên thực hiện:

Hồ Văn Vinh – 21520530

Hồ Quang Đình - 21520190

Nguyễn Lý Đăng Khoa - 21522229

Trương Công Quốc Triệu - 215222714

Thành phố Hồ Chí Minh, tháng 6 năm 2024

LỜI CẢM ƠN

Trước hết, nhóm chúng em xin gửi lời cảm ơn sâu sắc đến tập thể quý thầy cô trường Đại học Công nghệ Thông tin - Đại học Quốc gia TP.HCM và quý thầy cô khoa Hệ thống thông tin đã tạo điều kiện, giúp chúng em học tập và có được những kiến thức cơ bản làm tiền đề giúp chúng em hoàn thành được dự án này.

Đặc biệt, nhóm chúng em xin gửi lời cảm ơn chân thành và sâu sắc tới thầy Nguyễn Hồ Duy Trí (Giảng viên giảng dạy lý thuyết và thực hành môn DỮ LIỆU LỚN – IS405). Nhờ sự hướng dẫn tận tình và chu đáo của thầy, nhóm chúng em đã học hỏi được nhiều kinh nghiệm và hoàn thành thuận lợi, đúng tiến độ cho dự án của mình.

Ngoài ra, chúng em cũng gửi lời cảm ơn đến tập thể lớp IS405.O21 khoảng thời gian qua đã đồng hành cùng nhau. Cảm ơn sự đóng góp của tất cả các bạn cho những buổi học luôn sôi nổi, thú vị và dễ tiếp thu.

Trong quá trình thực hiện tiểu luận, nhóm chúng em luôn giữ một tinh thần cầu tiến, học hỏi và cải thiện từ những sai lầm, tham khảo từ nhiều nguồn tài liệu khác nhau và luôn mong đạt được kết quả nhất có thể. Tuy nhiên, do vốn kiến thức còn hạn chế trong quá trình trau dồi từng ngày, nhóm chúng em không thể tránh được những sai sót, vì vậy chúng em mong rằng quý thầy cô sẽ đưa ra nhận xét một cách chân thành để chúng em học hỏi thêm kinh nghiệm nhằm mục đích phục vụ tốt các dự án khác trong tương lai. Xin chân thành cảm ơn quý thầy cô!

Nhóm thực hiện

NHẬN XÉT CỦA GIẢNG VIÊN

This image shows a full page of white paper with horizontal dotted lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting practice. There are no margins, text, or other markings on the page.

....., ngày.....tháng.....năm 2024

Người nhận xét

(Ký tên và ghi rõ họ tên)

MỤC LỤC

LỜI CẢM ƠN	3
NHẬN XÉT CỦA GIẢNG VIÊN.....	4
MỤC LỤC	5
DANH MỤC HÌNH ẢNH.....	Error! Bookmark not defined.
CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI	8
1.1. Lý do chọn đề tài	8
1.2. Giới thiệu nguồn dữ liệu	9
1.3. Mô tả dữ liệu	10
1.4. Mô tả bài toán	12
CHƯƠNG 2: TIỀN XỬ LÝ DỮ LIỆU	13
2.1. Cơ sở lý thuyết	13
2.1.1. Kỹ thuật <i>StringIndexer</i>	13
2.1.2. Kỹ thuật <i>OneHotEncoder</i>	13
2.1.3. Kỹ thuật <i>VectorAssembler</i>	14
2.1.4. Kỹ thuật chuẩn hoá <i>Min-max</i>	14
2.1.5. Kỹ thuật chuẩn hoá <i>Z-score</i>	15
2.2. Thực nghiệm	15











2.2.1. Khám phá dữ liệu.....	15
2.2.2. Chuyển đổi dữ liệu	21
2.3. Kết quả đạt được	25
CHƯƠNG 3: CÁC THUẬT TOÁN KHAI THÁC DỮ LIỆU	29
3.1. K-Means	29
3.1.1. Cơ sở lý thuyết.....	29
3.1.2. Thực nghiệm	35
3.2. Simple Moving Average (SMA)	45
3.2.1. Cơ sở lý thuyết.....	45
3.2.2. Thực nghiệm	49
3.3. KNN	64
3.3.1. Cơ sở lý thuyết.....	64
3.3.2. Thực nghiệm	66
CHƯƠNG 4: SO SÁNH KẾT QUẢ ĐẠT ĐƯỢC.....	71
4.1. Phát biểu kết quả.....	71
4.1.1. K-Means.....	71
4.1.2. Simple Moving Average (SMA).....	73
4.1.3. KNN	79
4.2. So sánh, đánh giá	80
4.2.1. K-Means.....	80

4.2.2. Simple Moving Average (SMA).....	87
4.2.3. KNN	91
CHƯƠNG 5: KẾT LUẬN	94
5.1. Ưu điểm	94
5.2. Hạn chế.....	94
5.3. Hướng phát triển	94
PHÂN CÔNG CÔNG VIỆC	96
TÀI LIỆU THAM KHẢO	97

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

1.1. Lý do chọn đề tài

Dự báo thời tiết là một yếu tố không thể thiếu trong cuộc sống hàng ngày của con người. Từ việc lên kế hoạch cho các hoạt động ngoài trời đến việc chuẩn bị cho các tình huống khẩn cấp, thông tin thời tiết đóng vai trò then chốt trong việc đảm bảo an toàn và hiệu quả. Bằng cách dự báo chính xác các điều kiện thời tiết, chúng ta có thể giảm thiểu rủi ro, bảo vệ tài sản và con người, cũng như tối ưu hóa các hoạt động kinh tế và xã hội.

90% accurate					80% accurate		50% accurate		
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed
									
76°	74°	70°	70°	71°	76°	75°			

Với sự phát triển nhanh chóng của công nghệ và khoa học dữ liệu, việc ứng dụng các phương pháp tiên tiến như học máy và xử lý dữ liệu lớn trong dự báo thời tiết đã trở nên phổ biến. Điều này không chỉ giúp nâng cao độ chính xác của dự báo mà còn mở ra nhiều hướng nghiên cứu mới về các hiện tượng khí hậu. Chọn đề tài này cho phép khai thác tiềm năng của công nghệ hiện đại trong việc giải quyết một vấn đề có tính thực tiễn cao.

Dự báo thời tiết là một lĩnh vực đầy thách thức do tính phức tạp và biến động không ngừng của các yếu tố khí hậu. Việc xử lý và phân tích một lượng lớn dữ liệu từ nhiều nguồn khác nhau đòi hỏi kỹ năng và kiến thức chuyên sâu về khoa học dữ liệu và lập trình. Tuy nhiên, chính những thách thức này cũng tạo ra cơ hội để học hỏi và phát triển các kỹ năng mới, đồng thời đóng góp vào sự tiến bộ của lĩnh vực dự báo thời tiết.

Một hệ thống dự báo thời tiết chính xác không chỉ có giá trị khoa học mà còn mang

lại lợi ích kinh tế và xã hội đáng kể. Ví dụ, ngành nông nghiệp có thể sử dụng dự báo thời tiết để lên kế hoạch gieo trồng và thu hoạch, giúp tăng năng suất và giảm thiểu thiệt hại do thời tiết xấu. Ngành giao thông vận tải cũng có thể cải thiện hiệu quả và an toàn thông qua các dự báo thời tiết chính xác. Chọn đề tài này là một cách để đóng góp vào sự phát triển bền vững của xã hội và kinh tế.

Dự báo thời tiết không chỉ giới hạn ở việc dự đoán mưa hay nắng, mà còn có thể mở rộng sang các hiện tượng thời tiết cực đoan như bão, lũ lụt hay hạn hán. Các mô hình dự báo thời tiết có thể được áp dụng trong nhiều lĩnh vực khác nhau, từ quản lý tài nguyên nước đến phòng chống thiên tai. Điều này cho thấy tầm quan trọng và khả năng ứng dụng rộng rãi của việc nghiên cứu và phát triển các phương pháp dự báo thời tiết.

1.2. Giới thiệu nguồn dữ liệu

Tên nguồn dữ liệu: Weather Prediction – ananthr1

Nguồn dữ liệu: <https://www.kaggle.com/datasets/ananthr1/weather-prediction>

Dataset bao gồm 6 thuộc tính x 1461 dòng dữ liệu được lấy từ 01/01/2012 – 31/12/2015 tại thành Phố Seattle – Hoa Kỳ.

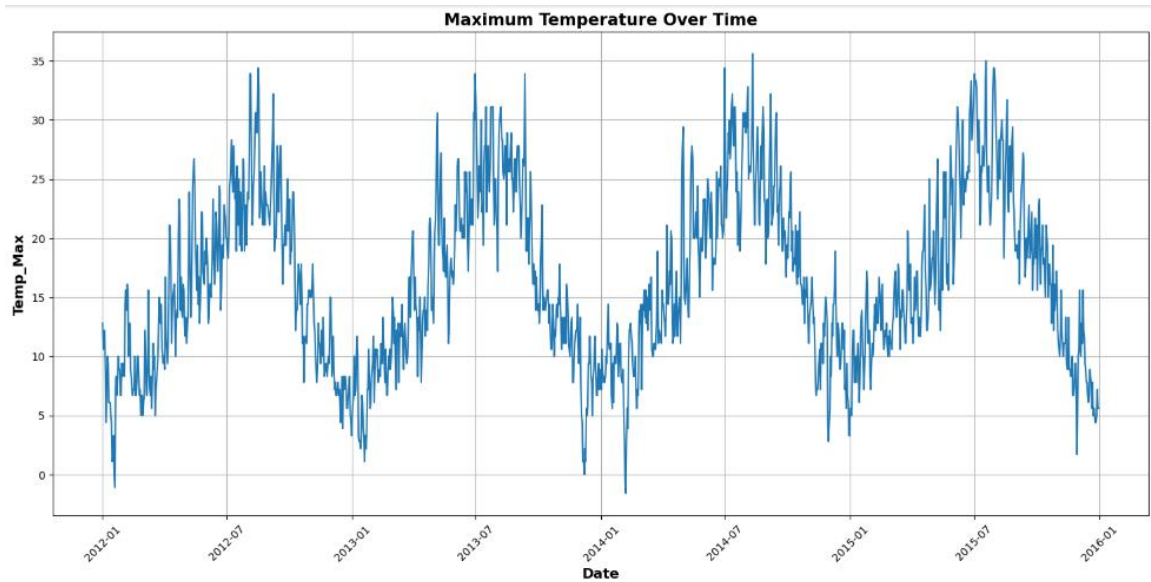
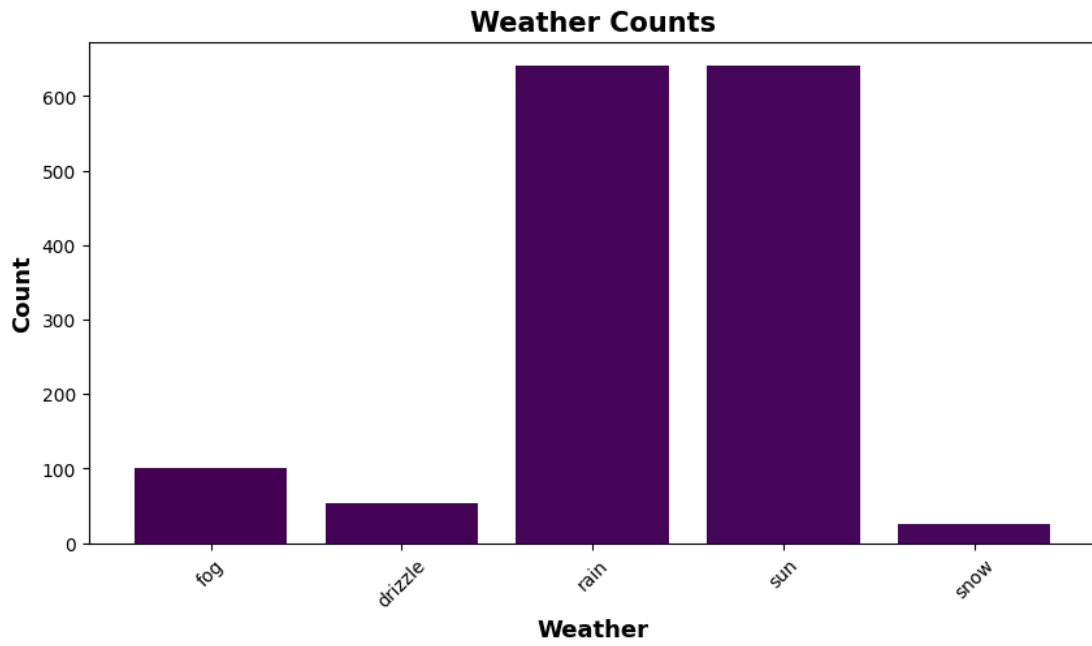
Dữ liệu liên quan đến dự báo thời tiết ở thành phố Seattle. Mỗi dòng bao gồm thông tin về thời gian ghi nhận các thông số thời tiết và nhãn thời tiết đó.

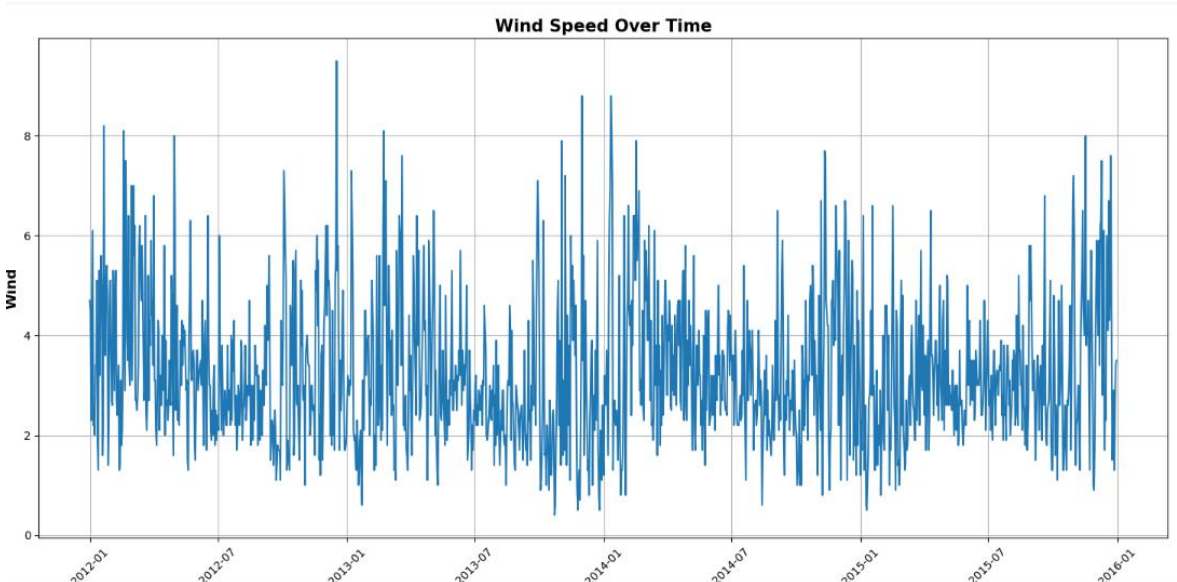
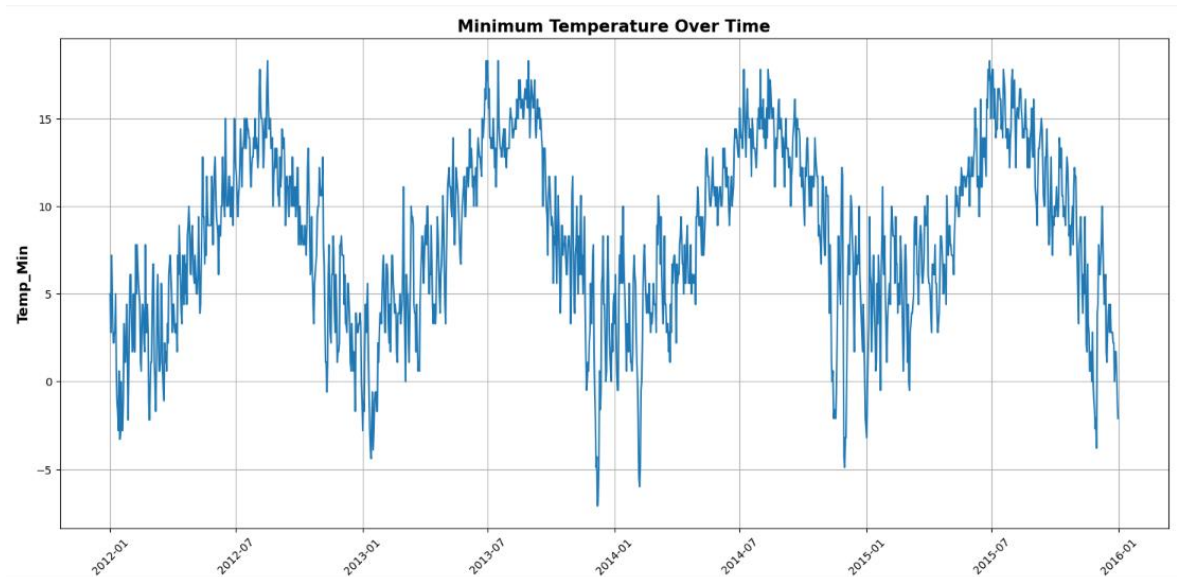
1.3. Mô tả dữ liệu

STT	Tên thuộc tính	Kiểu dữ liệu	Ý nghĩa
1	date	String	Ngày ghi nhận
2	precipitation	String	Chỉ số giáng thủy* (Inch)
3	temp_max	String	Nhiệt độ cao nhất trong ngày (Celsius)
4	temp_min	int	Nhiệt độ thấp nhất trong ngày (Celsius)
5	wind	int	Tốc độ gió (Miles/hour)
6	weather	int	Nhãn thời tiết (rain, sunny, snow, dizzle, fog)

*Giáng thủy (trong khí tượng học) là tên gọi chung các hiện tượng nước thoát ra khỏi những đám mây dưới các dạng lỏng (mưa) và dạng rắn (mưa tuyết, mưa đá, tuyết), nhằm phân biệt với các hiện tượng nước tách ra từ không khí (sương), lượng mưa xảy ra khi một phần của khí quyển trở nên bão hòa với hơi nước (đạt độ ẩm tương đối 100%), để nước cô đọng và "ngưng tụ" sương móc, sương băng). Do đó, bụi và sương mù không phải là ngưng tụ mà là huyền phù, vì hơi nước không cô đọng đủ để ngưng tụ

summary	date	precipitation	temp_max	temp_min	wind	weather
count	1461	1461	1461	1461	1461	1461
mean	null	3.0294318959616757	16.43908281998628	8.234770704996588	3.241136208076654	null
stddev	null	6.68019432231474	7.349758097360173	5.023004179961267	1.4378250588746202	null
min	2012-01-01	0.0	-1.6	-7.1	0.4	drizzle
max	2015-12-31	55.9	35.6	18.3	9.5	sun





1.4. Mô tả bài toán

Trong báo cáo này, nhóm thực hiện phương pháp phân cụm bằng **thuật toán KMeans và SMA, thuật toán phân lớp bằng KNN** dựa trên sự tương đồng của dữ liệu và tìm ra tập luật để dự đoán. Từ đó giúp cho các tổ chức cơ quan có nguồn dự đoán thời tiết với độ tin cậy cao hơn.

CHƯƠNG 2: TIỀN XỬ LÝ DỮ LIỆU

2.1. Cơ sở lý thuyết

2.1.1. Kỹ thuật *StringIndexer*

Kỹ thuật *StringIndexer* là một kỹ thuật mã hóa dữ liệu chuỗi thành các số nguyên. Kỹ thuật này được sử dụng trong các bài toán học máy để chuyển đổi các thuộc tính chuỗi thành các thuộc tính số, giúp cho việc huấn luyện mô hình dễ dàng hơn và đạt hiệu quả tốt hơn

Khi sử dụng kỹ thuật *StringIndexer*, các giá trị chuỗi sẽ được gán một số nguyên duy nhất tương ứng. Các giá trị phổ biến sẽ được gán số nguyên nhỏ hơn, trong khi các giá trị ít xuất hiện sẽ được gán số nguyên lớn hơn.

Ví dụ ta có một thuộc tính “pet” với các giá trị “cat”, “dog”, “pig”, thì kỹ thuật *StringIndexer* sẽ gán số nguyên 0 cho “cat”, 1 cho “dog”, và 2 cho “pig”.

2.1.2. Kỹ thuật *OneHotEncoder*

Kỹ thuật *OneHotEncoder* là một phương pháp mã hóa dữ liệu danh mục thành các biến nhị phân (binary variables). Kỹ thuật này thường được sử dụng sau khi áp dụng *StringIndexer* để chuyển đổi các giá trị danh mục (categorical values) thành các giá trị số nguyên. *OneHotEncoder* giúp mô hình học máy không bị ảnh hưởng bởi thứ tự của các giá trị danh mục.

Khi sử dụng kỹ thuật *OneHotEncoder*, mỗi giá trị danh mục sẽ được biểu diễn thành một vector nhị phân, trong đó chỉ có một phần tử có giá trị là 1 và các phần tử còn lại có giá trị là 0. Ví dụ, nếu ta có một thuộc tính “color” với các giá trị “red”, “green”, “blue”, thì kỹ thuật *OneHotEncoder* sẽ chuyển đổi thành các vector nhị phân tương ứng như sau:

“red” được chuyển thành [1, 0, 0]

“green” được chuyển thành [0, 1, 0]

“blue” được chuyển thành [0, 0, 1]

Kỹ thuật này giúp mô hình học máy xử lý tốt hơn với các thuộc tính danh mục, tránh

được sự thiên vị do thứ tự của các giá trị danh mục.

2.1.3. Kỹ thuật VectorAssembler

Kỹ thuật VectorAssembler là một phương pháp dùng để kết hợp nhiều cột (hoặc các thuộc tính) thành một cột duy nhất dưới dạng vector. Kỹ thuật này thường được sử dụng trong quá trình tiền xử lý dữ liệu để chuẩn bị dữ liệu cho các mô hình học máy, đặc biệt là các mô hình yêu cầu dữ liệu đầu vào dưới dạng vector như mô hình hồi quy hoặc mô hình phân loại.

Khi sử dụng kỹ thuật VectorAssembler, các cột đầu vào (input columns) có thể là các thuộc tính số (numerical attributes) hoặc các vector, và kết quả là một cột đầu ra (output column) dưới dạng vector. Ví dụ, nếu ta có các thuộc tính “precipitation”, “temp_max”, “temp_min”, và “wind”, thì VectorAssembler sẽ kết hợp các thuộc tính này thành một cột vector duy nhất, giúp cho việc huấn luyện mô hình trở nên đơn giản và hiệu quả hơn.

Kỹ thuật này rất hữu ích trong việc chuẩn bị dữ liệu đầu vào cho các mô hình học máy, đảm bảo rằng tất cả các thuộc tính được kết hợp một cách có hệ thống và nhất quán, đồng thời giúp cải thiện hiệu suất và độ chính xác của mô hình.

2.1.4. Kỹ thuật chuẩn hoá Min-max

Min-max normalization là một kỹ thuật tiền xử lý dữ liệu được sử dụng trong máy học và thống kê để scale các đặc trưng số của tập dữ liệu về một phạm vi chung, khoảng [0,1].

Công thức chuẩn hoá như sau:

$$X_{nor} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Trong đó:

- X_{nor} là giá trị đã được chuẩn hóa của X
- X là giá trị gốc của đặc trưng

- X_{min} là giá trị nhỏ nhất của đặc trưng trong tập dữ liệu
- X_{max} là giá trị lớn nhất của đặc trưng trong tập dữ liệu

Min-max normalization đảm bảo rằng tất cả các đặc trưng có cùng một tỷ lệ, ngăn chặn các đặc trưng có phạm vi số lớn hơn không chế các đặc trưng có phạm vi nhỏ hơn trong quá trình huấn luyện các mô hình máy học. Đây là một kỹ thuật đặc biệt hữu ích cho các thuật toán dựa trên tính toán khoảng cách, như **K-Nearest Neighbors (KNN)** và **K-Means** nơi tỷ lệ các đặc trưng có thể ảnh hưởng đáng kể đến kết quả.

2.1.5. Kỹ thuật chuẩn hoá Z-score

Z-score standardization biến đổi các giá trị của các đặc trưng số để chúng có giá trị trung bình gần bằng 0 và độ lệch chuẩn gần bằng 1.

Công thức chuẩn hoá:

$$Z = \frac{X - \mu}{\sigma}$$

Trong đó:

- Z là Z-score của giá trị X
- X là giá trị ban đầu
- μ là giá trị trung bình (mean) của đặc trưng
- σ là độ lệch chuẩn (standard deviation) của đặc trưng

Quá trình standardization giúp làm cho việc phân phối của giá trị dữ liệu trở nên chuẩn hóa, với giá trị trung bình bằng 0 và độ lệch chuẩn bằng 1. Điều này giúp thuận tiện trong việc so sánh các biến số với nhau và giúp giảm thiểu ảnh hưởng của các giá trị ngoại lệ.

2.2. Thực nghiệm

2.2.1. Khám phá dữ liệu

Bộ dữ liệu thời tiết Seattle (2012-2015) bao gồm ngày, lượng mưa, nhiệt độ, tốc độ

gió và những loại thời tiết. Khám phá dữ liệu cho thấy xu hướng theo mùa và sự bất thường. Phân tích mô hình lượng mưa xác định các thời kỳ ẩm ướt và các đợt mưa cực đoan. Phân tích tốc độ gió cho thấy sự khác biệt giữa các loại thời tiết khác nhau. Việc khám phá này đưa ra cái nhìn tổng quan, nâng cao hiểu biết về khí hậu của một tiểu bang ở Hoa Kỳ - Seattle, hỗ trợ việc dự báo và nghiên cứu khí hậu.

Một số truy vấn từ cơ bản đến nâng cao cho việc khám phá bộ dữ liệu này:

- Lược đồ của dataframe và 3 dòng dữ liệu trong dataframe sau khi đọc file dữ liệu csv từ lệnh `spark.read.csv`

```
df = spark.read.csv("./seattle-weather.csv", header=True, inferSchema=True)
df.printSchema()
df.show(3)
```

```
root
 |-- date: date (nullable = true)
 |-- precipitation: double (nullable = true)
 |-- temp_max: double (nullable = true)
 |-- temp_min: double (nullable = true)
 |-- wind: double (nullable = true)
 |-- weather: string (nullable = true)
```

date	precipitation	temp_max	temp_min	wind	weather
2012-01-01	0.0	12.8	5.0	4.7	drizzle
2012-01-02	10.9	10.6	2.8	4.5	rain
2012-01-03	0.8	11.7	7.2	2.3	rain

only showing top 3 rows

- Các chỉ số thống kê cơ bản như số lượng (count), giá trị trung bình (mean), độ lệch chuẩn (stddev), giá trị thấp nhất (min), giá trị lớn nhất (max) và số lượng null ở từng thuộc tính của dữ liệu

```
df.describe().drop('weather').show()
dict_null = {col:df.filter(df[col].isNull()).count() for col in df.columns}
df_null_counts = pd.DataFrame.from_dict(dict_null, orient="index", columns=['Số giá trị null'])
print(df_null_counts)
```

summary	precipitation	temp_max	temp_min	wind
count	1461	1461	1461	1461
mean	3.0294318959616757	16.43908281998628	8.234770704996588	3.241136208076654
stddev	6.68019432231474	7.349758097360173	5.023004179961267	1.4378250588746202
min	0.0	-1.6	-7.1	0.4
max	55.9	35.6	18.3	9.5

```
Số giá trị null
date          0
precipitation 0
temp_max      0
temp_min      0
wind          0
weather       0
```


- Thống kê số lượng dữ liệu ở mỗi loại thời tiết (weather)

```
df.groupBy('weather').count().orderBy('count', ascending=False).show()
print("Số giá trị của cột weather:", df.select('weather').distinct().count())
```

weather	count
rain	641
sun	640
fog	101
drizzle	53
snow	26

Số giá trị của cột weather: 5

- Đếm số ngày có mưa

```
# đếm số ngày có mưa
sqlDF = spark.sql("""
    SELECT
        COUNT(*) AS rainy_days_count
    FROM weather
    WHERE weather = 'rain'
""")
sqlDF.show()
```

rainy_days_count
641

- Tìm ngày có lượng mưa lớn nhất qua các năm

```
# Tìm ngày có lượng mưa lớn nhất của các năm
sqlDF = spark.sql("""
    SELECT
        year,
        date,
        precipitation
    FROM (
        SELECT
            YEAR(TO_DATE(date, 'yyyy-MM-dd')) AS year,
            date,
            precipitation,
            ROW_NUMBER() OVER (PARTITION BY YEAR(TO_DATE(date, 'yyyy-MM-dd')) ORDER BY precipitation DESC) AS rank
        FROM weather
    ) tmp
    WHERE rank = 1
""")
sqlDF.show()
```

```
+---+-----+-----+
|year|      date|precipitation|
+---+-----+-----+
|2015|2015-10-07|          9.9|
|2013|2013-03-20|          9.9|
|2014|2014-12-09|          9.9|
|2012|2012-11-01|          9.7|
+---+-----+-----+
```

- Tính tốc độ gió trung bình cho từng loại thời tiết

```
# Tính tốc độ gió trung bình cho từng loại thời tiết
sqlDF = spark.sql("""
    SELECT
        weather,
        AVG(wind) AS avg_wind_speed
    FROM weather
    GROUP BY weather
    ORDER BY avg_wind_speed DESC
""")
sqlDF.show()
```

```
⇒ +-----+-----+
|weather| avg_wind_speed|
+-----+-----+
|  snow| 4.411538461538461|
|  rain|3.6698907956318254|
|   sun| 2.956406250000002|
|   fog|2.4811881188118816|
|drizzle|2.3679245283018866|
+-----+-----+
```

- Liệt kê những tháng có nhiệt độ trung bình trên 25°C qua các năm

```
# Liệt kê những tháng có nhiệt độ trung bình trên 25 độ qua các năm
sqlDF = spark.sql("""
    SELECT
        YEAR(date) AS year,
        MONTH(date) AS month,
        AVG(temp_max) AS avg_temp_max
    FROM weather
    GROUP BY YEAR(date), MONTH(date)
    HAVING AVG(temp_max) > 25
    ORDER BY year, month
""")
sqlDF.show()
```

```
⇒ +----+-----+-----+
   |year|month| avg_temp_max|
   +----+-----+-----+
   |2012|    8|25.858064516129033|
   |2013|    7|26.093548387096785|
   |2013|    8| 26.11935483870968|
   |2014|    7|26.899999999999995|
   |2014|    8| 26.38387096774193|
   |2015|    6|26.063333333333333|
   |2015|    7|28.093548387096778|
   |2015|    8|26.087096774193547|
   +----+-----+-----+
```

- Nhiệt độ trung bình các tháng cao nhất của năm

```
# nhiệt độ trung bình cao nhất các tháng của năm
sqlDF = spark.sql("""
    WITH monthly_avg_temp AS (
        SELECT
            YEAR(date) AS year,
            MONTH(date) AS month,
            AVG(temp_max) AS avg_max_temp
        FROM weather
        GROUP BY YEAR(date), MONTH(date)
    )
    SELECT
        month,
        year,
        avg_max_temp
    FROM (
        SELECT
            month,
            year,
            avg_max_temp,
            ROW_NUMBER() OVER (PARTITION BY month ORDER BY avg_max_temp DESC) AS rank
        FROM monthly_avg_temp
    ) ranked
    WHERE rank = 1
    ORDER BY month
""")
sqlDF.show()
```

```
+-----+-----+-----+
|month|year|    avg_max_temp|
+-----+-----+-----+
| 1|2015|10.154838709677419|
| 2|2015| 12.51785714285714|
| 3|2015|14.377419354838713|
| 4|2015|15.503333333333336|
| 5|2015|20.025806451612905|
| 6|2015|26.063333333333333|
| 7|2015|28.093548387096778|
| 8|2014| 26.38387096774193|
| 9|2014|23.163333333333334|
|10|2014| 17.96129032258065|
|11|2013|12.053333333333331|
|12|2014|10.138709677419357|
+-----+-----+-----+
```

2.2.2. Chuyển đổi dữ liệu

2.2.2.a. Tiền xử lý dữ liệu cho KNN và SMA

Import các lớp OneHotEncoder, StringIndexer, và VectorAssembler từ thư viện

pyspark.ml.feature. Các lớp này được sử dụng để mã hóa dữ liệu và chuẩn bị dữ liệu cho mô hình học máy.

Import lớp Pipeline từ thư viện pyspark.ml. Pipeline giúp dễ dàng quản lý và tổ chức các bước tiền xử lý dữ liệu và xây dựng mô hình.

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.ml import Pipeline
```

```
categorical_cols = ['weather']
```

Định nghĩa một danh sách chứa tên của các cột danh mục (categorical columns) cần chuyển đổi. Ở đây chỉ có một cột danh mục là weather.

```
indexers = [StringIndexer(inputCol=col, outputCol=f"{col}_index").fit(df_cleaned) for col
in categorical_cols]:
```

Sử dụng StringIndexer để chuyển đổi các giá trị chuỗi trong cột danh mục thành các số nguyên. Mỗi giá trị chuỗi sẽ được gán một số nguyên duy nhất. Kết quả là danh sách indexers chứa các đối tượng StringIndexer đã được fit vào dữ liệu df_cleaned.

```
# Convert categorical variables into numerical values
categorical_cols = ['weather']
indexers = [StringIndexer(inputCol=col, outputCol=f"{col}_index").fit(df_cleaned) for col in categorical_cols]
```

```
encoder = OneHotEncoder(inputCols=[f"{col}_index" for col in categorical_cols],
outputCols=[f"{col}_encoded" for col in categorical_cols])
```

Sử dụng OneHotEncoder để mã hóa các giá trị số nguyên từ StringIndexer thành các vector nhị phân (one-hot vectors). Các cột đầu vào (inputCols) là các cột đã được StringIndexer chuyển đổi, và các cột đầu ra (outputCols) là các cột chứa vector nhị phân.

```
encoder = OneHotEncoder(inputCols=[f"{col}_index" for col in categorical_cols],
outputCols=[f"{col}_encoded" for col in categorical_cols])
```

```
feature_cols = ['precipitation', 'temp_max', 'temp_min', 'wind', 'weather_encoded']
```

Định nghĩa một danh sách chứa tên của các cột đặc trưng (features) cần sử dụng trong mô hình. Bao gồm các cột số học và cột danh mục đã được mã hóa.

```
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
```

Sử dụng VectorAssembler để kết hợp các cột đặc trưng thành một cột vector duy nhất có tên là features. Cột này sẽ được sử dụng làm đầu vào cho mô hình học máy.

```
# Assemble features
feature_cols = ['precipitation', 'temp_max', 'temp_min', 'wind', 'weather_encoded']
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
```

Tạo một đối tượng Pipeline chứa các bước tiền xử lý dữ liệu, bao gồm các bước sử dụng StringIndexer, OneHotEncoder, và VectorAssembler. Các bước này sẽ được thực hiện tuần tự để chuẩn bị dữ liệu. Fit Pipeline vào dữ liệu df_cleaned. Quá trình này sẽ thực hiện các bước tiền xử lý dữ liệu (indexing, encoding, và assembling) trên dữ liệu.

Áp dụng Pipeline đã fit vào dữ liệu df_cleaned để chuyển đổi dữ liệu. Kết quả là DataFrame transformed_data chứa các cột đã được mã hóa và cột vector đặc trưng.

```
# Create pipeline
pipeline = Pipeline(stages=indexers + [encoder, assembler])

# Fit and transform data
pipeline_model = pipeline.fit(df_cleaned)
transformed_data = pipeline_model.transform(df_cleaned)
```

Chọn cột features (chứa vector đặc trưng) và cột weather_index (chứa nhãn đã được mã hóa) từ DataFrame transformed_data để sử dụng trong quá trình huấn luyện và kiểm tra mô hình.

```
# Select features and label
data = transformed_data.select("features", "weather_index")
```

2.2.2.b. Tiền xử lý dữ liệu cho K-Means

Các cột dữ liệu cần chuẩn hoá là *precipitation*, *temp_max*, *temp_min*, *wind*

❖ Chuẩn hoá Min-max

Code chuẩn hoá Min-max

```
normalized_cols = ['precipitation', 'temp_max', 'temp_min', 'wind'] # Các cột dữ liệu cần chuẩn hoá

# chuẩn hoá min-max
min_values = df.agg(*[F.min(col).alias(col) for col in normalized_cols]).first().asDict()
max_values = df.agg(*[F.max(col).alias(col) for col in normalized_cols]).first().asDict()
print("min_values:\n", min_values)
print("max_values:\n", max_values)
def MinMaxNormalize(r):
    nor = r.asDict()
    for col in normalized_cols:
        nor[col + '_nor'] = (r[col] - min_values[col]) / (max_values[col] - min_values[col])
    return Row(**nor)

df_min_max = df.rdd.map(MinMaxNormalize).toDF()
print('\nChuẩn hoá Min-Max:\n')
df_min_max.show(3)
df_min_max.select(['precipitation_nor', 'temp_max_nor', 'temp_min_nor', 'wind_nor']).describe().show()
```

- **min_values:** dictionary chứa giá trị min của các thuộc tính cần chuẩn hoá
- **max_values:** dictionary chứa giá trị max của các thuộc tính cần chuẩn hoá
- **MinMaxNormalize:** hàm chuẩn hoá Min-max
 - Tham số:
 - *r*: một dòng dữ liệu của RDD
 - Trả về: dòng dữ liệu sau khi chuẩn hoá. Các cột sau khi chuẩn hoá được thêm vào dòng dữ liệu là *precipitation_nor*, *temp_max_nor*, *temp_min_nor*, *wind_nor*

❖ Chuẩn hoá Z-score

Code chuẩn hoá Z-score


```
# chuẩn hoá z-score
mean_values = df.agg(*[F.mean(col).alias(col) for col in normalized_cols]).first().asDict()
stddev_values = df.agg(*[F.stddev(col).alias(col) for col in normalized_cols]).first().asDict()
print("mean_values:\n", mean_values)
print("stddev_values:\n", stddev_values)
def Z_Score(r):
    nor = r.asDict();
    for col in normalized_cols:
        nor[col + '_nor'] = (r[col] - mean_values[col]) / stddev_values[col]
    return Row(**nor)

df_zscore = df.rdd.map(Z_Score).toDF()
print('Chuẩn hoá Z-Score:\n')
df_zscore.show(3)
df_zscore.select(['precipitation_nor', 'temp_max_nor', 'temp_min_nor', 'wind_nor']).describe().show(truncate=False)
```

- **mean_values:** dictionary chứa giá trị mean của các thuộc tính cần chuẩn hoá
- **stddev_values:** dictionary chứa giá trị độ lệch chuẩn của các thuộc tính cần chuẩn hoá
- **Z_score:** hàm chuẩn hoá Z-score
 - Tham số:
 - r: một dòng dữ liệu của RDD
 - Trả về: dòng dữ liệu sau khi chuẩn hoá. Các cột sau khi chuẩn hoá được thêm vào dòng dữ liệu là *precipitation_nor*, *temp_max_nor*, *temp_min_nor*, *wind_nor*

2.3. Kết quả đạt được

```
Weather Index Mapping:
0.0: rain
1.0: sun
2.0: snow
3.0: drizzle
4.0: fog
```

```
+-----+
|weather_index|
+-----+
|1.0          |
|0.0          |
|0.0          |
|0.0          |
|1.0          |
|1.0          |
|1.0          |
|1.0          |
|0.0          |
|1.0          |
|1.0          |
|0.0          |
|0.0          |
|2.0          |
|1.0          |
|1.0          |
|0.0          |
|0.0          |
|2.0          |
|0.0          |
|1.0          |
+-----+
only showing top 20 rows
```

```

+-----+
| features |
+-----+
| (8,[1,2,3,5],[27.2,13.9,2.4,1.0]) |
| [5.3,7.2,3.9,1.8,1.0,0.0,0.0,0.0] |
| [0.5,17.8,12.8,5.0,1.0,0.0,0.0,0.0] |
| [0.5,23.9,13.3,3.2,1.0,0.0,0.0,0.0] |
| (8,[1,2,3,5],[7.2,-2.7,1.0,1.0]) |
| (8,[1,2,3,5],[25.0,15.0,2.9,1.0]) |
| (8,[1,2,3,5],[24.4,13.9,3.0,1.0]) |
| (8,[1,2,3,5],[3.3,-2.1,3.6,1.0]) |
| [1.0,13.9,6.1,3.0,1.0,0.0,0.0,0.0] |
| (8,[1,2,3,5],[11.7,5.6,6.5,1.0]) |
| (8,[1,2,3,5],[25.6,11.1,3.0,1.0]) |
| [0.3,17.2,12.2,2.6,1.0,0.0,0.0,0.0] |
| (8,[1,2,3,6],[6.1,0.6,2.8,1.0]) |
| (8,[1,2,3,5],[17.2,6.1,1.7,1.0]) |
| (8,[1,2,3,5],[15.6,11.1,3.0,1.0]) |
| [29.5,13.3,6.7,8.0,1.0,0.0,0.0,0.0] |
| [1.0,16.1,11.7,4.7,1.0,0.0,0.0,0.0] |
| (8,[1,2,3,6],[12.8,5.6,1.0,1.0]) |
| [9.4,11.7,7.8,1.4,1.0,0.0,0.0,0.0] |
| (8,[1,2,3,5],[16.7,2.8,2.4,1.0]) |
+-----+
only showing top 20 rows

```

Kết quả sau khi chuẩn hoá theo Min-max

```

min_values:
{'precipitation': 0.0, 'temp_max': -1.6, 'temp_min': -7.1, 'wind': 0.4}
max_values:
{'precipitation': 55.9, 'temp_max': 35.6, 'temp_min': 18.3, 'wind': 9.5}

```

```

Chuẩn hoá Min-Max:
+-----+-----+-----+-----+-----+-----+-----+-----+
| date|precipitation|temp_max|temp_min|wind|weather| precipitation_nor| temp_max_nor| temp_min_nor| wind_nor|
+-----+-----+-----+-----+-----+-----+-----+-----+
| 2012-01-01| 0.0| 12.8| 5.0| 4.7| drizzle| 0.0| 0.3870967741935484| 0.4763779527559055| 0.4725274725274725|
| 2012-01-02| 10.9| 10.6| 2.8| 4.5| rain| 0.19499105545617174| 0.32795698924731176| 0.38976377952755903| 0.4505494505494505|
| 2012-01-03| 0.8| 11.7| 7.2| 2.3| rain| 0.014311270125223615| 0.35752688172043007| 0.5629921259842521| 0.2087912087912088|
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 3 rows

+-----+-----+-----+-----+
| summary| precipitation_nor| temp_max_nor| temp_min_nor| wind_nor|
+-----+-----+-----+-----+
| count| 1461| 1461| 1461| 1461|
| mean| 0.0541937727363446| 0.4849215811824274| 0.6037311301179765| 0.312212770118315|
| stddev| 0.11950258179453911| 0.19757414240215532| 0.19775607007721535| 0.15800275372248568|
| min| 0.0| 0.0| 0.0| 0.0|
| max| 1.0| 1.0| 1.0| 1.0|
+-----+-----+-----+-----+

```

Giá trị các cột chuẩn hoá *precipitation_nor*, *temp_max_nor*, *temp_min_nor*, *wind_nor* thuộc khoảng [0, 1]

Kết quả sau khi chuẩn hoá theo Z-score

```
mean_values:
{'precipitation': 3.0294318959616757, 'temp_max': 16.43908281998628, 'temp_min': 8.234770704996588, 'wind': 3.241136208076654}
stddev_values:
{'precipitation': 6.68019432231474, 'temp_max': 7.349758097360173, 'temp_min': 5.023004179961267, 'wind': 1.4378250588746202}
```

Chuẩn hoá Z-Score:

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      date|precipitation|temp_max|temp_min|wind|weather| precipitation_nor| temp_max_nor| temp_min_nor| wind_nor|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|2012-01-01|      0.0|    12.8|      5.0| 4.7|drizzle|-0.45349457662362636|-0.4951296045094783| -0.6439912429102402| 1.0146323316031178|
|2012-01-02|     10.9|     10.6|      2.8| 4.5|  rain|  1.17819448421541|-0.7944591839129391| -1.081976146203107|  0.8755333509826664|
|2012-01-03|      0.8|     11.7|      7.2| 2.3|  rain| -0.3337375813326879|-0.6447943942112089| -0.20600633961737366|-0.6545554358422977|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 3 rows
```

```
+-----+-----+-----+-----+-----+
|summary|precipitation_nor| temp_max_nor| temp_min_nor| wind_nor|
+-----+-----+-----+-----+-----+
|count| 1461| 1461| 1461| 1461|
|mean| -8.072484045580451E-16| 3.872178264394215E-15| -2.356621248437738E-15| 3.971251041146785E-15|
|stddev| 1.0000000000000009| 1.0000000000000004| 0.9999999999999999| 1.0|
|min| -0.45349457662362636| -2.454377760604858| -3.0529082110210064| -1.9759957517365847|
|max| 7.9145254693306955| 2.607013309308205| 2.0038265815420897| 4.353007866493948|
+-----+-----+-----+-----+-----+
```

Giá trị mean của các cột chuẩn hoá *precipitation_nor*, *temp_max_nor*, *temp_min_nor*, *wind_nor* đều gần bằng 0 và độ lệch chuẩn (stddev) của chúng đều gần bằng 1

CHƯƠNG 3: CÁC THUẬT TOÁN KHAI THÁC DỮ LIỆU

3.1. K-Means

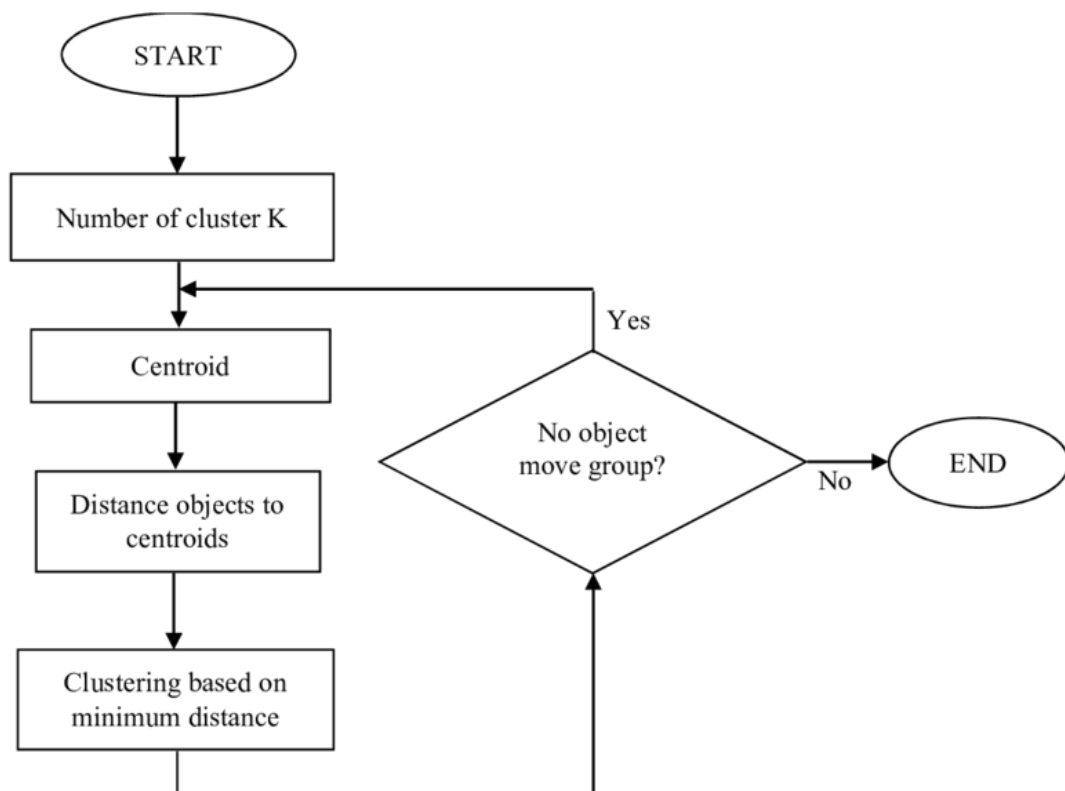
3.1.1. Cơ sở lý thuyết

3.1.1.a. Thuật toán phân cụm K-Means

K-Means là một thuật cơ bản thuộc loại Unsupervised learning, học không giám sát. Trong thuật toán K-Means clustering, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thế nào để phân dữ liệu thành K cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau.

Đầu vào thuật toán: tập dữ liệu X và số lượng cụm K

Đầu ra thuật toán: K centroid M (các điểm trung tâm của mỗi cụm) và tập **prediction** chứa kết quả phân cụm cho mỗi điểm dữ liệu trong tập X



Các bước của thuật toán:

Bước 1: Chọn K điểm bất kì thuộc tập X làm các centroid ban đầu.

Bước 2: Phân mỗi điểm dữ liệu trong **X** vào cluster có centroid gần nó nhất, tính theo khoảng cách Euclid.

Bước 3: Kiểm tra điều kiện dừng (hội tụ - converged)

- Nếu việc gán dữ liệu vào từng cluster ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán.

Bước 4: Cập nhật centroid cho từng cluster bằng cách lấy trung bình cộng của tất cả các điểm dữ liệu đã được gán vào cluster đó sau bước 2.

Bước 5: Quay lại bước 2.

Một số ưu điểm:

- Dễ cài đặt
- Có thể xử lý dữ liệu có nhiều chiều

Một số hạn chế:

- Cần chọn số lượng cụm cần clustering.

Có một số phương pháp giúp xác định số lượng clusters như *Elbow method*, *Average Silhouette*, ...

Nhóm sử dụng hai phương pháp kể trên để xác định số cụm **K** cho thuật toán K-Means. Nội dung của hai phương pháp được trình bày sau phần này.

- Kết quả clustering phụ vào các centroid được khởi tạo ban đầu.

Tùy vào các centroid ban đầu mà thuật toán có thể có tốc độ hội tụ rất chậm hoặc thậm chí cho chúng ta kết quả không chính xác.

Có một vài cách khắc phục đó là:

- Chạy K-Means clustering nhiều lần với các centroids ban đầu khác nhau rồi chọn cách độ đo **WCSS** đạt giá trị nhỏ nhất. **WCSS** là một độ đo dùng để đánh giá chất lượng của việc phân cụm, chi tiết được trình bày sau phần này. Nhóm chọn phương án này cho việc cài đặt K-Means, cụ thể là thuật toán sẽ được chạy 5 lần và lấy kết quả của lần chạy có độ đo **WCSS** kể trên là nhỏ nhất.

- Dùng thuật toán ***k-means++*** để chọn các giá trị centroid khởi tạo ban đầu. Thuật toán này được đề xuất vào năm 2007 bởi nhóm tác giả David Arthur và Sergei Vassilvitskii.
- Nhạy cảm với dữ liệu ngoại lai.
K-Means có xu hướng bị ảnh hưởng nhiều bởi dữ liệu ngoại lai (outliers). Một vài điểm dữ liệu ngoại lai có thể làm thay đổi vị trí của trung tâm cụm, dẫn đến kết quả phân cụm không chính xác.
- Các cluster cần có dạng hình tròn và có số lượng điểm dữ liệu xấp xỉ nhau.

Một số ứng dụng:

- **Phân khúc khách hàng:** K-Means có thể được sử dụng để phân loại khách hàng vào các nhóm dựa trên các đặc điểm như hành vi mua sắm, lịch sử giao dịch, đánh giá sản phẩm, ... Điều này giúp doanh nghiệp hiểu rõ hơn về khách hàng và tạo ra các chiến lược tiếp thị hiệu quả hơn.
- **Phân loại hình ảnh:** K-Means có thể được sử dụng để phân loại hình ảnh, ví dụ như phân loại các loại hoa dựa trên các đặc điểm của hình ảnh.
- **Giảm kích thước dữ liệu:** K-Means có thể được sử dụng để giảm kích thước dữ liệu bằng cách thay thế các giá trị dữ liệu bằng trung tâm cụm tương ứng.

3.1.1.b. Các phương pháp chọn số cụm K tối ưu & đánh giá việc phân cụm của K-Means**❖ Elbow method**

Elbow method là một phương pháp được sử dụng để chọn số lượng cụm tối ưu trong thuật toán K-Means. Phương pháp này dựa trên việc tính toán và quan sát sự biến đổi của độ đo **WCSS** với số lượng cụm khác nhau.

WCSS là viết tắt của "*Within-Cluster Sum of Squares*" (tổng bình phương khoảng cách trong cụm). Đây là một phép đo được sử dụng trong thuật toán K-Means để đánh giá sự tập trung của các điểm dữ liệu trong từng cụm. Cụ thể, **WCSS** của một cụm được tính bằng tổng của bình phương khoảng cách

từ mỗi điểm dữ liệu trong cụm đó tới centroid của cụm đó.

$$WCSS = \sum_{c_i}^{c_k} \left(\sum_{d_j \in C_i}^{d_m} distance(d_j, c_i)^2 \right)$$

Trong đó:

c_i là centroid của cluster C_i

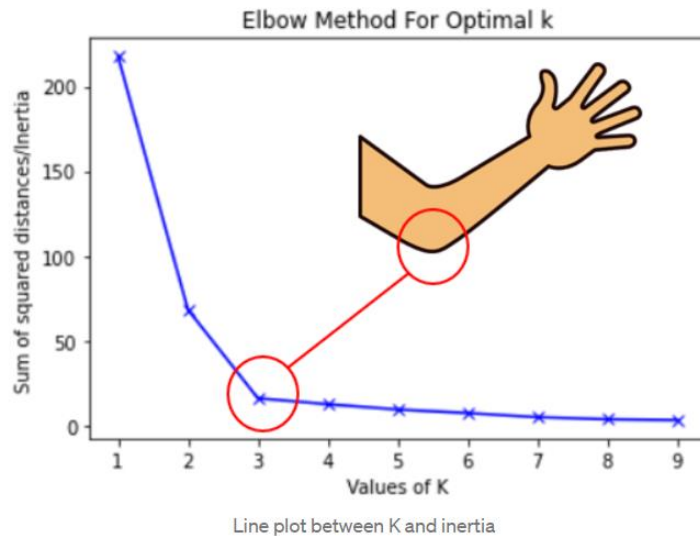
d_j là điểm dữ liệu thuộc cluster C_i

$distance(d_j, c_i)$ là khoảng cách euclid của d_j và c_i

Trong quá trình huấn luyện thuật toán K-means, mục tiêu là tối ưu hóa **WCSS** bằng cách điều chỉnh vị trí của các trung tâm cụm sao cho đạt được giá trị nhỏ nhất có thể.

Quay lại Elbow method, phương pháp này được thực hiện như sau:

- Bước 1:** Thực hiện thuật toán K-Means với một loạt các giá trị **K** khác nhau (thường là từ 2 đến một giới hạn nào đó).
- Bước 2:** Đối với mỗi giá trị **K**, dựa vào kết quả phân cụm của K-Means tính toán độ đo **WCSS**.
- Bước 3:** Vẽ biểu đồ đường thể hiện sự biến thiên của **WCSS** theo số lượng cụm.
- Bước 4:** Quan sát biểu đồ và tìm điểm mà tại đó sự biến thiên của **WCSS** giảm nhanh đột ngột, làm cho biểu đồ có hình dạng giống như khuỷu tay (elbow). Giá trị **K** ứng với điểm này thường được chọn là số lượng cụm tối ưu cho thuật toán K-Means.



Ý tưởng chính của Elbow method là chọn số lượng cụm sao cho khi tăng số lượng cụm, sự giảm của **WCSS** không còn đáng kể, tức là không cải thiện rõ rệt hiệu suất của việc phân cụm nữa.

❖ Average Silhouette

Hệ số Silhouette trung bình của tập dữ liệu sau khi được phân cụm bởi thuật toán K-Means là một tiêu chí hữu ích khác để đánh giá chất lượng việc phân cụm. Hệ số Silhouette của một điểm dữ liệu là một độ đo thể hiện mức độ liên hệ của điểm dữ liệu với các điểm dữ liệu khác trong cùng cluster và với các điểm dữ liệu trong cluster láng giềng (neighboring cluster) của nó. Giá trị Silhouette trong khoảng $[-1, 1]$, giá trị càng gần 1 thể hiện điểm dữ liệu phù hợp với cụm của nó và kém phù hợp với các cụm láng giềng còn giá trị càng gần -1 thì điểm dữ liệu được phân cụm không thích hợp. Nếu đa số các điểm dữ liệu có hệ số Silhouette cao thì việc phân cụm là đúng đắn. Việc phân cụm đạt Silhouette trung bình cao hơn 0.7 được coi là “mạnh”, cao hơn 0.5 là “hợp lý” và trên 0.25 là “yếu”. Silhouette có thể được tính toán với bất kỳ phương pháp đo khoảng cách nào, như khoảng cách Euclidean hoặc khoảng cách Manhattan.

- Phương pháp tính

Giả sử một tập dữ liệu đã được phân thành K cụm bởi thuật toán K-Means.

Cho điểm dữ liệu $i \in C_I$ (điểm dữ liệu i thuộc cụm C_I), có:

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} d(i, j)$$

là khoảng cách trung bình giữa i và tất cả các điểm dữ liệu khác trong cùng cụm với i . $a(i)$ còn được xem là độ đo thể hiện mức độ phù hợp của điểm i với cụm của nó (giá trị càng thấp thì càng phù hợp).

Tiếp đó cần xác định giá trị không tương đồng trung bình của điểm i với một số cụm C_J là khoảng cách trung bình từ i đến tất cả điểm dữ liệu trong C_J ($C_J \neq C_I$). Với điểm dữ liệu $i \in C_I$, tính:

$$b(i) = \min_{J \neq I} \left(\frac{1}{|C_J|} \sum_{j \in C_J} d(i, j) \right)$$

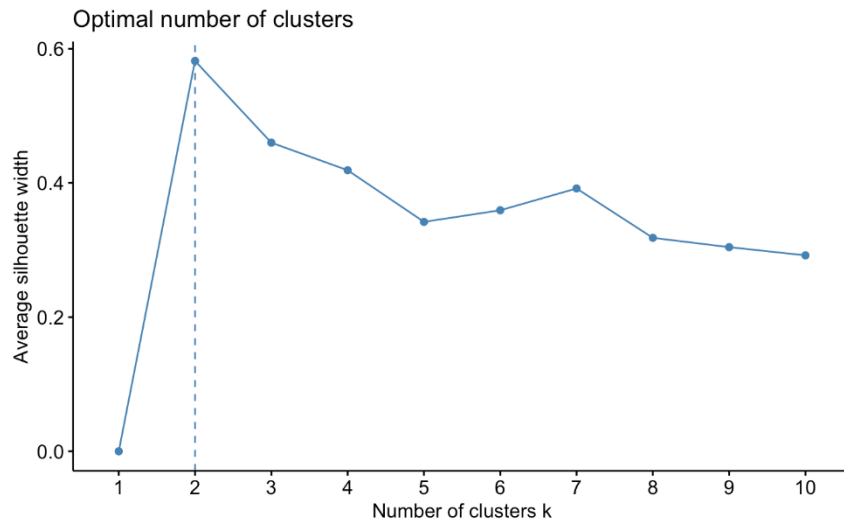
là trung bình khoảng cách nhỏ nhất từ i đến mọi điểm trong các cluster khác cluster của i . Cluster có giá trị không tương đồng với điểm i thấp nhất được gọi là cluster láng giềng (neighboring cluster) của điểm i và cluster đó phù hợp thứ hai để phân cụm cho điểm i .

Cuối cùng hệ số Silhouette của điểm i được tính theo công thức:

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

$$-1 \leq s(i) \leq 1$$

Để tính điểm Silhouette cho việc phân cụm, cần tính trung bình hệ số Silhouette của tất cả các điểm dữ liệu trong tập dữ liệu. Điểm Silhouette trung bình càng cao, thuật toán phân cụm càng tốt.



3.1.2. Thực nghiệm

3.1.2.a. Cài đặt thuật toán K-Means

Nhóm cài đặt thuật toán K-Means thông qua class K_Mean. Class K_Mean có các phương thức sau:

1. `__init__(self, k=2, predictCol='prediction', seed=1)`

Phương thức khởi tạo đối tượng class K_Mean

```
class K_Mean:
    def __init__(self, k=2, predictCol='prediction', seed=1):
        self.k = k
        self.centroids = None
        self.predictCol = predictCol
        self.seed=seed
        self.wcss = None
```

- k: số cụm cần clustering, mặc định là 2
- predictCol: tên cột chứa kết quả sau khi phân cụm, mặc định là 'prediction'
- seed: dùng cho việc chọn ngẫu nhiên k điểm dữ liệu làm centroid ban đầu
- centroids: list chứa các centroid của các cụm
- wcss: độ đo dùng để đánh giá việc phân cụm sau khi huấn luyện mô hình bằng phương thức `fit()` phía dưới

2. `squared_euclidean_distance(x, y)`

Phương thức tính bình phương khoảng cách euclid giữa 2 điểm dữ liệu x, y

```
@staticmethod
def squared_euclidean_distance(x, y): # Dùng squared euclid, không lấy căn bậc 2
    return sum((x[i] - y[i])**2 for i in range(len(x)))
```

3. `clusters_distances(self, features)`

Phương thức tính khoảng cách của một điểm dữ liệu đến các centroid hiện tại

```
def clusters_distances(self, features):
    return [K_Mean.squared_euclidean_distance(self.centroids[i], features) for i in range(self.k)]
```

- features: list chứa giá trị các đặc trưng của điểm dữ liệu
- Lần lượt duyệt qua list chứa các centroids, `self.centroids`, để tính khoảng cách với điểm dữ liệu, features, bằng phương thức `squared_euclidean_distance` đã đề cập trên
- Kết quả trả về là một danh sách chứa bình phương khoảng cách từ một điểm dữ liệu đến các centroid

4. `assign_to_centroid(self, features)`

Phương thức chọn cluster có centroid gần nhất với điểm dữ liệu

```
def assign_to_centroid(self, features):
    distances = self.clusters_distances(features)
    nearest_cluster = min(range(self.k), key=lambda i: distances[i])
    return nearest_cluster
```

- features: list chứa giá trị các đặc trưng của điểm dữ liệu
- distances: list chứa bình phương khoảng cách từ điểm dữ liệu đang xét đến các centroid hiện tại
- nearest_cluster: là cluster có centroid gần nhất với điểm đang xét. Có được bằng cách duyệt qua list distances để lấy vị trí index của phần tử có giá trị nhỏ nhất trong list
- Trả về cluster vừa tìm được để thực hiện phân cụm điểm dữ liệu vào cluster đó

5. `fit(self, df, feature_cols)`

Phương thức dùng để train mô hình K-Means trên tập dữ liệu df, kết quả có được là các centroid ở các cluster và độ đo **WCSS** giúp đánh giá việc phân cụm của thuật toán

```
def fit(self, df, feature_cols):
    # tạo cột features
    df = df.withColumn('features', F.array(feature_cols))
    self.wcss = float('inf')
    self.centroids = None
    for i in range(5):
        centroids = self.centroids
        # khởi tạo k centers ban đầu
        self.centroids = df.select('features').rdd.map(lambda r: r.features).takeSample(False, num=self.k, seed=self.seed)
        self.seed += 1

    # bắt đầu chạy thuật toán
    while True:
        # hàm udf để chọn cluster cho một điểm dữ liệu
        assign_to_centroid_udf = F.udf(self.assign_to_centroid, IntegerType())
        df = df.withColumn(self.predictCol, assign_to_centroid_udf(F.col('features')))

        avg_clusters = df.groupBy(self.predictCol).agg(*[F.mean(c).alias(c) for c in feature_cols]).withColumn('avg_features', F.array(feature_cols)).collect()
        new_centroids = self.centroids
        for cluster in avg_clusters:
            new_centroids[cluster[self.predictCol]] = cluster['avg_features']

        if (max(K_Mean.squared_euclidean_distance(self.centroids[i], new_centroids[i]) for i in range(self.k))**0.5) < 0.0001:
            break
        self.centroids = new_centroids
    # Kết thúc while loop

    # Tính độ đo WCSS dùng đánh giá việc phân cụm
    dist_udf = F.udf(lambda features, prediction: K_Mean.squared_euclidean_distance(features, self.centroids[prediction]), DoubleType())
    wcss = df.withColumn('dist', dist_udf(F.col('features'), F.col(self.predictCol))).agg(F.sum('dist')).first()[0]
    if self.wcss > wcss:
        self.wcss = wcss
    else:
        self.centroids = centroids
    # Kết thúc for loop

# Kết thúc method fit()
```

- df: pyspark dataframe chứa dữ liệu để huấn luyện mô hình K-Means
- feature_cols: danh sách tên các cột dùng làm đặc trưng để thuật toán dựa vào đó phân cụm dữ liệu
- df = df.withColumn('features', F.array(feature_cols)): tạo cột 'features', là danh sách chứa các giá trị ở các cột trong feature_cols của dữ liệu. Cột 'features' này được xem là một vector đại diện cho điểm dữ liệu
- self.wcss = float('inf'): khởi tạo giá trị wcss là dương vô cùng
- for i in range(5): chạy thuật toán 5 lần (vì kết quả thuật toán phụ thuộc vào các giá trị centroid được chọn ngẫu nhiên ban đầu, random 5 lần) và lấy kết quả của lần chạy có độ đo wcss thấp nhất
- centroids = self.centroids: lưu giá trị self.centroids (kết quả của lần chạy tốt nhất) vào biến tạm centroids. Dùng sau khi kết thúc lần chạy của thuật toán, nếu kết quả wcss của lần chạy này lớn hơn (kém hơn) lần

chạy tốt nhất thì gán lại `self.centroids = centroids`

- `self.centroids = df.select('features').rdd.map(lambda r: r.features) \`
`.takeSample(False, num=self.k, seed=self.seed)`

Chọn ngẫu nhiên k điểm dữ liệu trong tập dữ liệu để khởi tạo giá trị ban đầu cho các centroid bằng cách map từng dòng dữ liệu trong rdd để lấy giá trị cột features được thêm vào ban đầu và dùng phương thức `takeSample()` để chọn ngẫu nhiên k điểm dữ liệu

- While True: bắt đầu vòng lặp của thuật toán đến khi đạt được các centroid hội tụ (converged)
- `assign_to_centroid_udf = \`
`F.udf(self.assign_to_centroid, IntegerType())`

Tạo hàm udf (user define function) dựa trên phương thức `assign_to_centroid` để các worker trong cụm phân tán spark hiểu và thực thi được trên phân dữ liệu phân tán chúng đang giữ. `IntegerType()` để cho biết loại dữ liệu trả về từ hàm udf này là một số nguyên

- `df = \`
`df.withColumn(self.predictCol, \`
`assign_to_centroid_udf(F.col('features')))`

Tạo / cập nhật cột `self.predictCol` (cột chứa kết quả phân cụm cho mỗi điểm dữ liệu) bằng kết quả trả về của hàm `assign_to_centroid_udf`. Dòng này cập nhật giá trị cụm cho mỗi điểm dữ liệu bằng việc tìm cụm có centroid gần nhất với điểm dữ liệu đó.

- `avg_clusters = df.groupBy(self.predictCol) \`
`.agg(*[F.mean(c).alias(c) for c in feature_cols]) \`
`.withColumn('avg_features', F.array(feature_cols)).collect()`

Tính toán giá trị trung bình cho các cluster sau khi thực hiện phân cụm các điểm dữ liệu bằng cách nhóm dữ liệu theo kết quả phân cụm để tính giá trị trung bình của các cột trong danh sách `feature_cols`. Vì số cụm k

nhỏ nên có thể dùng `collect()` để lấy giá trị trung bình mỗi cụm gán cho biến `avg_clusters`

- `new_centroids`: lưu các giá trị trung bình của mỗi cụm có được từ biến `avg_cluster`. Các giá trị trung bình này sẽ là các centroid mới cho lần lặp tiếp theo của thuật toán. Đối với trường hợp cụm không có điểm dữ liệu thì centroid của cụm được giữ nguyên, không cập nhật
- `max(K_Mean.squared_euclidean_distance(self.centroids[i], new_centroids[i]) for i in range(self.k))**0.5) < 0.0001`

Điều kiện dừng thuật toán: Nếu các khoảng cách các các centroid mới và centroid hiện tại đều nhỏ hơn 0.0001 thì dừng, do các centroid mới thay đổi không đáng kể, hay có thể nói các centroid đã hội tụ (converged). Nếu không thỏa điều kiện dừng, thuật toán tiếp tục thực hiện với các giá trị centroid mới

- Khi đã tìm được các centroid hội tụ thì tính toán độ đo wcss để đánh giá chất lượng phân cụm của lần chạy này

- `dist_udf = \`

- `F.udf(lambda features, prediction: \`

- `K_Mean.squared_euclidean_distance(features, \`

- `self.centroids[prediction]), DoubleType())`

Tạo hàm udf từ hàm `squared_euclidean_distance` để tính bình phương khoảng cách từ một điểm dữ liệu đến centroid của cụm nó thuộc về

- `wcss = \`

```
df.withColumn('dist', dist_udf(F.col('features'), F.col(self.predictCol))) \
    .agg(F.sum('dist')).first()[0]
```

Đối với mỗi điểm dữ liệu, tính bình phương khoảng cách từ nó đến centroid của cụm nó thuộc bằng hàm `dist_udf` và lưu kết quả vào cột 'dist'. Sau đó lấy tổng giá trị của cột 'dist' này chính là độ đo wcss. Nếu kết quả wcss này thấp hơn (tốt hơn) của `self.wcss` thì lấy kết quả

centroid của lần chạy này

6. transform(self, df, feature_cols)

Thực hiện phân cụm và trả về kết quả phân cụm cho dữ liệu của dataframe df dựa trên các thuộc tính có trong danh sách feature_cols sau khi mô hình đã được huấn luyện (gọi hàm fit()) để tìm được các centroid của các cluster

```
def transform(self, df, feature_cols):
    df = df.withColumn('features', F.array(feature_cols))
    assign_to_centroid_udf = F.udf(self.assign_to_centroid, IntegerType())
    df = df.withColumn(self.predictCol, assign_to_centroid_udf(F.col('features')))
    return df
```

- df = df.withColumn('features', F.array(feature_cols)): tạo cột features là một mảng chứa giá trị các đặc trưng được liệt kê trong feature_col của mỗi điểm dữ liệu dùng cho việc phân cụm dữ liệu
- assign_to_centroid_udf = \

F.udf(self.assign_to_centroid, IntegerType())

Tạo hàm udf từ phương thức assign_to_centroid để thực hiện chọn cụm cho các điểm dữ liệu của df dựa vào giá trị của các centroid tìm được qua fit(). Nhãn cụm trả về là một số nguyên, IntegerType()
- df = \

df.withColumn(self.predictCol, \

assign_to_centroid_udf(F.col('features')))

Thực hiện phân cụm cho từng điểm dữ liệu và kết quả được lưu trong cột self.predictCol
- Trả về dataframe sau khi clustering. Dataframe trả về có 2 cột thuộc tính mới là 'features' và self.predictCol

3.1.2.b. Cài đặt lớp SilhouetteEvaluator để tính hệ số Silhouette cho clustering

Nhóm cài đặt class SilhouetteEvaluator để tính Silhouette score cho kết quả clustering của thuật toán. Đầu vào là một dataframe đã được phân cụm bởi lớp K_Mean, cụ thể là df trả về từ phương thức transform() của lớp K_Mean. Các phương thức trong lớp:

1. `__init__(self, predictCol='prediction')`

Hàm khởi tạo của lớp

```
class SilhouetteEvaluator:
    def __init__(self, predictCol='prediction'):
        self.predictCol = predictCol
```

2. `evaluate(self, df)`

Hàm đánh giá, trả về Silhouette score của kết quả phân cụm dữ liệu của dataframe df

```
def evaluate(self, df):
    df = df.select('date', 'features', self.predictCol)
    cluster_df = df.withColumnRenamed('features', 'features_') \
        .withColumnRenamed(self.predictCol, 'cluster') \
        .withColumnRenamed('date', 'date_')
    df = df.crossJoin(cluster_df).filter(F.col('date') != F.col('date_'))
    distance_udf = F.udf(lambda x, y: sum((x[i]-y[i])**2 for i in range(len(x))), DoubleType()) # Dùng squared euclid, không lấy căn bậc 2
    df = df.withColumn('dist', distance_udf(F.col('features'), F.col('features_')))

    # Tính giá trị b(i) cho từng điểm dữ liệu
    b = df.filter(F.col(self.predictCol) != F.col('cluster')) \
        .groupBy('date', 'cluster').agg(F.mean('dist').alias('avg_dist')) \
        .groupBy('date').agg(F.min('avg_dist').alias('b_i'))

    # Tính giá trị a(i) cho từng điểm dữ liệu
    a = df.filter(F.col(self.predictCol) == F.col('cluster')) \
        .groupBy('date').agg(F.mean('dist').alias('a_i'))

    # Tính hệ số trung bình Silhouette cho việc phân cụm
    silhouette_score = a.join(b, 'date') \
        .withColumn('s_i', (F.col('b_i') - F.col('a_i')) / F.greatest(F.col('a_i'), F.col('b_i'))) \
        .agg(F.mean('s_i')).first()[0]

    return silhouette_score
```

- Các cột thuộc tính cần dùng trong df là:
 - o 'date': đóng vai trò như khoá chính của dòng dữ liệu
 - o 'features': mảng các giá trị đặc trưng dùng phân cụm
 - o self.predictCol: cột chứa kết quả phân cụm, cụm mà điểm dữ liệu được phân vào
- cluster_df chứa các dữ liệu của df với các cột được thay đổi tên:
 - o 'date' thành 'date_'
 - o 'features' thành 'features_'
 - o self.predictCol thành 'cluster'
- `df = df.crossJoin(cluster_df).filter('date != date_')`
 Thực hiện kết từng dòng dữ liệu trong cluster_df với mỗi dòng dữ liệu

trong df với điều kiện giá trị cột 'date' khác cột 'date_' để tránh kết một điểm dữ liệu với chính nó

- `distance_udf = F.udf(lambda x, y: \`
`sum((x[i]-y[i])**2 for i in range(len(x))), DoubleType())`

Tạo hàm udf trả về bình phương khoảng cách euclid của 2 điểm dữ liệu x, y. Giá trị trả về kiểu số thực, DoubleType()

- `df = df.withColumn('dist', \`
`distance_udf(F.col('features'), F.col('features_')))`

Tính bình phương khoảng cách của 2 điểm dữ liệu ở mỗi dòng dữ liệu của df bằng hàm distance_udf

- `b = df.filter(F.col(self.predictCol) != F.col('cluster')) \`
`.groupBy('date', 'cluster').agg(F.mean('dist').alias('avg_dist')) \`
`.groupBy('date').agg(F.min('avg_dist').alias('b_i'))`

Tính giá trị b(i) của mỗi điểm dữ liệu có khoá chính là 'date'. b(i) của mỗi điểm dữ liệu là trung bình bình phương khoảng cách ngắn nhất của nó đến các điểm thuộc các cluster khác cluster của nó nên điều kiện filter là `F.col(self.predictCol) != F.col('cluster')`.

Trước tiên cần tính trung bình bình phương khoảng cách của điểm dữ liệu đến với các điểm dữ liệu khác ở các cluster khác nó. Thực hiện việc này bằng cách nhóm các dòng dữ liệu theo khoá chính 'date' và 'cluster' (các cluster khác cluster của nó) sau đó tính giá trị trung bình cho mỗi nhóm đã group và kết quả tính toán được chứa trong cột 'avg_dist'.

Đến đây đã có được khoảng cách mỗi điểm tới các cluster không chứa nó, giờ thì cần tìm min của chúng bằng việc nhóm theo mỗi điểm dữ liệu, là nhóm theo thuộc tính khoá 'date' và dùng hàm F.min() với cột 'avg_dist', đặt tên cột chứa kết quả là 'b_i'. Lúc này dataframe b có 2 cột thuộc tính là 'date' và 'b_i'

```
- a = df.filter(F.col(self.predictCol) == F.col('cluster')) \
    .groupBy('date').agg(F.mean('dist').alias('a_i'))
```

Tính giá trị $a(i)$ của mỗi điểm dữ liệu có khoá chính là 'date'. $a(i)$ của mỗi điểm dữ liệu là trung bình bình phương khoảng cách của nó đến các điểm thuộc cùng cluster với nó nên điều kiện filter là 'F.col(self.predictCol) == F.col('cluster')'.

Nhóm theo khoá chính 'date' của mỗi điểm dữ liệu và dùng F.mean() với cột 'dist' và chứa kết quả trong cột 'a_i'. Lúc này dataframe a có 2 cột thuộc tính là 'date' và 'a_i'

```
- silhouette_score =
    a.join(b, 'date') \
    .withColumn('s_i', \
        (F.col('b_i') - F.col('a_i')) / F.greatest(F.col('a_i'), F.col('b_i'))) \
    .agg(F.mean('s_i')).first()[0]
```

Hệ số Silhouette được tính bằng cách kết dataframe a và b theo cột 'date' và tính giá trị cho cột 's_i' ở mỗi dòng bằng giá trị cột 'b_i' trừ 'a_i', sau đó chia cho $\max\{a_i, b_i\}$. Hệ Silhouette của của tập dữ liệu là giá trị trung bình của cột 's_i'

3.1.2.c. Dùng Elbow method và Silhouette score để chọn số cụm K

Chạy thuật toán K-Means số cụm k từ 2 đến 10. Nhóm chọn chuẩn hoá theo z-score nên df_zscore được dùng để chạy thuật toán K-Means

```
# Chọn số cụm k cho thuật toán K-Mean dựa vào Elbow Method và hệ số Silhouette

import matplotlib.pyplot as plt
import numpy as np

feature_cols = ['precipitation_nor', 'temp_max_nor', 'temp_min_nor', 'wind_nor']
fig, axs = plt.subplots(1, 2, figsize=(15, 6))

wcss_list = []
silhouette_list = []

for k in range(2,11):
    kmean = K_Mean(k)
    kmean.fit(df_zscore, feature_cols)
    df_fit = kmean.transform(df_zscore, feature_cols)
    evaluator = SilhouetteEvaluator()
    silhouette_score = evaluator.evaluate(df_fit)
    wcss_list.append(kmean.wcss)
    silhouette_list.append(silhouette_score)
```

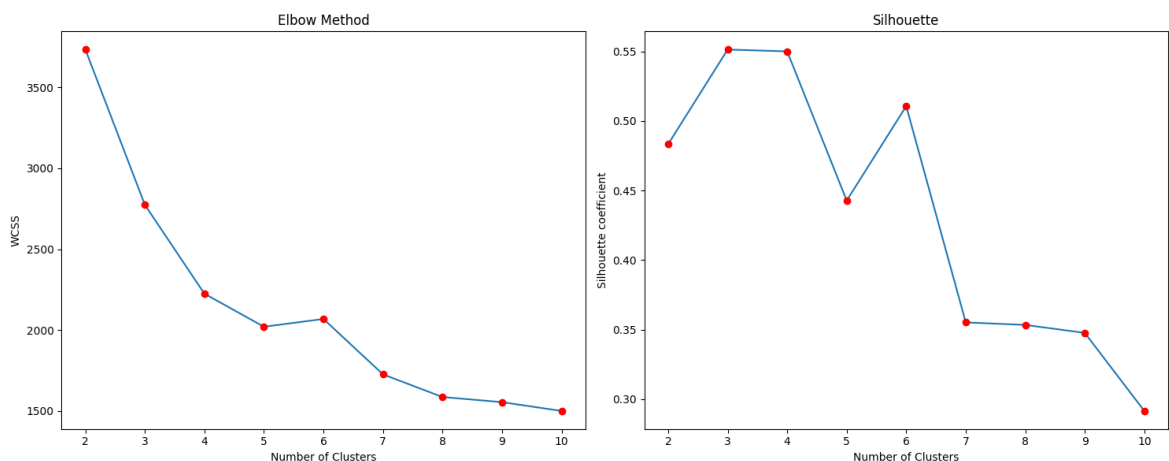
Hiển thị kết quả đánh giá **WCSS** (Elbow method) và **Silhouette score** có được qua các lần chạy thuật toán với số lượng cụm k khác nhau bằng biểu đồ đường

```
# Elbow Method
axs[0].plot(np.arange(2, 11), wcss_list, '-')
axs[0].plot(np.arange(2,11), wcss_list, 'o', color='red')
axs[0].set_xlabel('Number of Clusters')
axs[0].set_ylabel('WCSS')
axs[0].set_title('Elbow Method')

# Silhouette
axs[1].plot(np.arange(2, 11), silhouette_list, '-')
axs[1].plot(np.arange(2,11), silhouette_list, 'o', color='red')
axs[1].set_xlabel('Number of Clusters')
axs[1].set_ylabel('Silhouette coefficient')
axs[1].set_title('Silhouette')

plt.tight_layout()
plt.show()
```

Kết quả



Dựa vào 2 cách chọn k trên, số cụm tối ưu cho clustering là 4, $k = 4$

3.1.2.d. Thực hiện training mô hình và phân cụm dữ liệu

```
[ ] # Thực hiện training mô hình trên dữ liệu df_zscore với k = 4

kmean = K_Mean(k=4)
kmean.fit(df_zscore, feature_cols)

# Giá trị các centroids và wcss sau khi train

print('centroids:')
for centroid in kmean.centroids:
    print(centroid)
print('\nWCSS:', kmean.wcss)

centroids:
[-0.05115713282887737, -0.5144709004093946, -0.4031301744140794, 1.2023089562248654]
[-0.18888445099916307, -0.6387667848666028, -0.6878702451702778, -0.6269401970426495]
[2.803215257816656, -0.35461786806223405, 0.06492894973362687, 1.08734316238199]
[-0.3920632133753182, 1.1487554752849358, 1.0333538082707596, -0.3512503971864999]

WCSS: 2224.8662391929133
```

```
# Phân cụm df_zscore

df_result = kmean.transform(df_zscore, feature_cols)
df_result.show(5)
```

	date	precipitation	temp_max	temp_min	wind	weather	precipitation_nor	temp_max_nor	temp_min_nor	wind_nor	features	prediction
2012-01-01		0.0	12.8	5.0	4.7	drizzle	-0.45349457662362636	-0.4951296045094783	-0.6439912429102402	1.0146323316031178	[-0.4534945766236...	0]
2012-01-02		10.9	10.6	2.8	4.5	rain	1.17819448421541	-0.7944591839129391	-1.081976146203107	0.8755333509826664	[1.17819448421541...	0]
2012-01-03		0.8	11.7	7.2	2.3	rain	-0.3337375813326879	-0.6447943942112089	-0.20600633961737366	-0.6545554358422977	[-0.3337375813326...	1]
2012-01-04		20.3	12.2	5.6	4.7	rain	2.5853391788839364	-0.5767649443467859	-0.5245408147394586	1.0146323316031178	[2.58533917888393...	2]
2012-01-05		1.3	8.9	2.8	6.1	rain	-0.2588894592758514	-1.025759313451977	-1.081976146203107	1.9883251959462762	[-0.2588894592758...	0]

3.2. Simple Moving Average (SMA)

3.2.1. Cơ sở lý thuyết

Thuật toán Simple Moving Average (SMA) là một phương pháp phổ biến trong phân tích kỹ thuật, được sử dụng để làm mịn dữ liệu thời gian nhằm xác định xu hướng của một dãy số liệu theo thời gian. SMA được tính bằng cách lấy trung bình cộng của một tập hợp con các điểm dữ liệu trong một khoảng thời gian cố định và di chuyển qua từng thời điểm trong dãy số liệu.

Sau đây là các bước của thuật toán:

Bước 1: Khởi tạo các biến và thông số

- **Input:**

- **data:** Mảng các giá trị đầu vào (giá cổ phiếu, nhiệt độ, v.v.)
- **n:** Số lượng phần tử trong khoảng thời gian trung bình động (window size)

- **Output:**

- **SMA:** Mảng các giá trị trung bình động đơn giản

Bước 2: Kiểm tra điều kiện ban đầu

- Kiểm tra nếu **n** lớn hơn độ dài của **data**. Nếu đúng, không thể tính toán SMA, kết thúc thuật toán.

Bước 3: Tính tổng ban đầu của khoảng thời gian đầu tiên

- Khởi tạo **sum** bằng tổng của **n** phần tử đầu tiên trong **data**.
 - **sum = data[0] + data[1] + ... + data[n-1]**

Bước 4: Tính giá trị SMA đầu tiên

- Tính giá trị SMA đầu tiên và lưu vào mảng **SMA**.
 - **SMA[0] = sum / n**

Bước 5: Lặp qua các phần tử tiếp theo trong data

- Sử dụng vòng lặp để duyệt qua từ phần tử thứ **n** đến phần tử cuối cùng của **data**.
 - **for i in range(n, len(data)):**
 - Cập nhật **sum** bằng cách trừ giá trị của phần tử đã ra khỏi cửa sổ và cộng giá trị của phần tử mới vào cửa sổ.
 - **sum = sum - data[i-n] + data[i]**
 - Tính giá trị SMA mới và lưu vào mảng **SMA**.
 - **SMA[i-n+1] = sum / n**

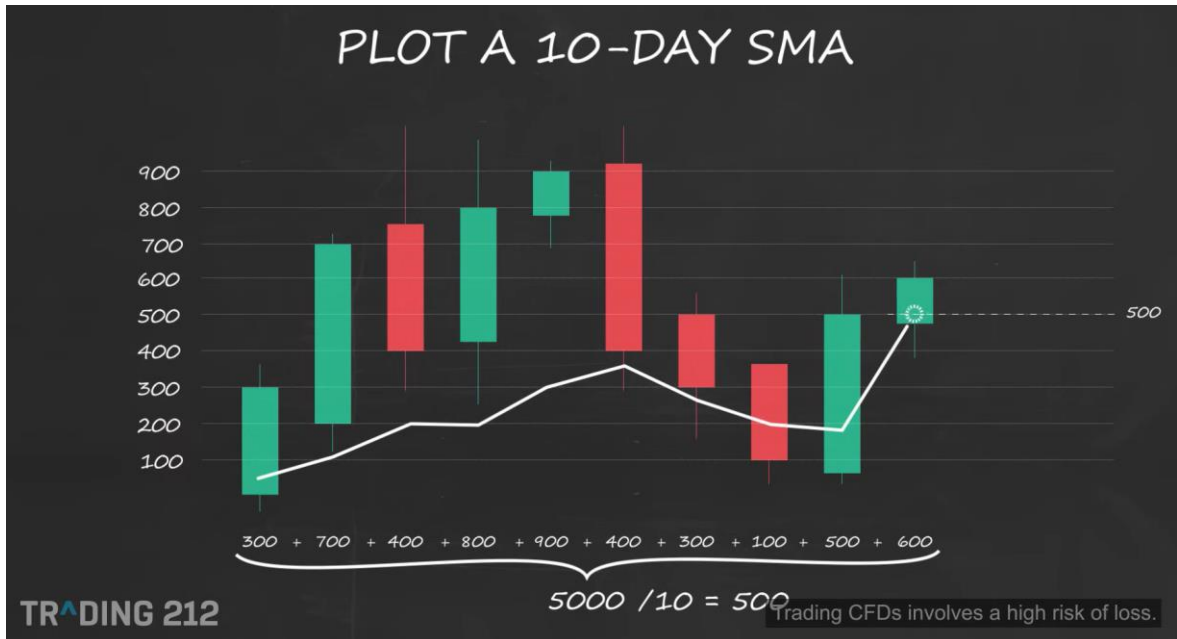
Bước 6: Kết thúc thuật toán

- Kết quả cuối cùng là mảng **SMA** chứa các giá trị trung bình động đơn giản tương ứng với dữ liệu đầu vào **data**.

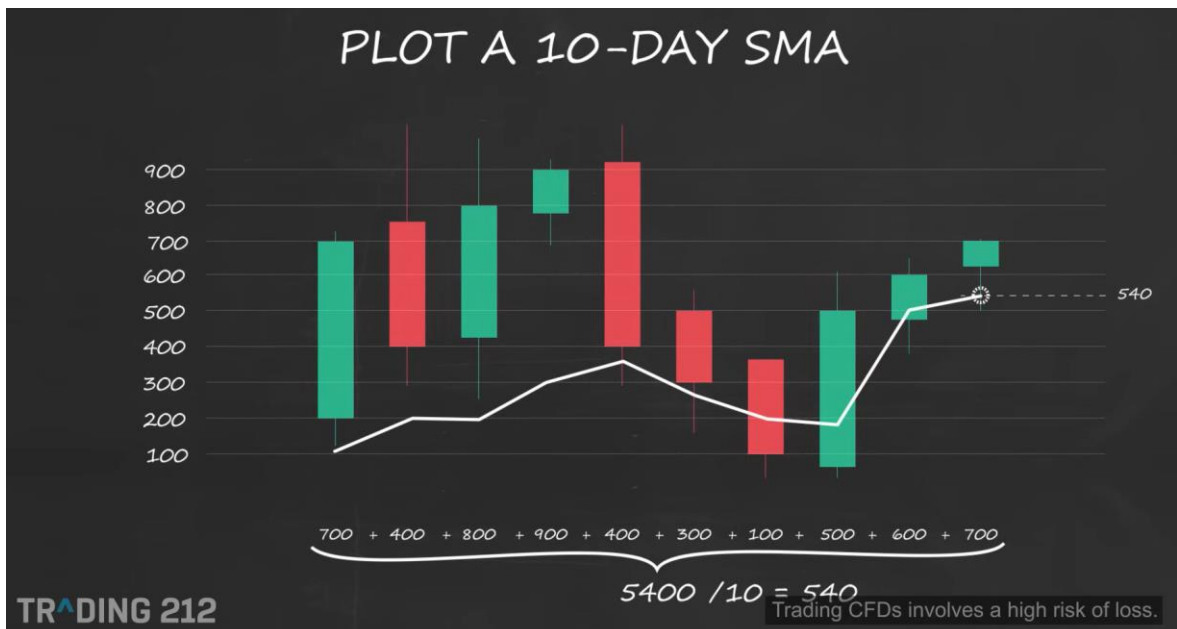
Ví dụ minh họa: giả sử cho bộ dữ liệu về giá cổ phiếu như hình, ta cần tính giá

trị SMA với `window_size = 10`, tại điểm đánh dấu màu trắng như hình, thực hiện tính SMA tại điểm đó bằng cách lấy trung bình giá trị cổ phiếu 10 ngày gần nhất

$$(300 + 700 + 400 + 800 + 900 + 400 + 300 + 100 + 500 + 600) / 10 = 500$$



Tiếp theo, tính giá trị SMA cho điểm tiếp theo (điểm được đánh dấu), cũng tương tự lấy 10 giá trị gần nhất, lúc này điểm giá cổ phiếu đầu tiên bị lược bỏ thay bằng điểm mới nhất. Ta tính được giá trị là 540.



Quá trình tính toán sẽ diễn ra liên tục cho đến điểm cuối cùng của bộ dữ liệu và kết thúc.

Về ưu và nhược điểm của SMA:

- **Ưu điểm**

1. **Dễ hiểu và dễ tính toán:** SMA là một trong những công cụ phân tích kỹ thuật cơ bản và dễ hiểu nhất, chỉ cần tính trung bình cộng của một số giá trị dữ liệu.
2. **Làm mịn dữ liệu:** SMA giúp làm mịn các biến động ngẫu nhiên và nhiễu trong dữ liệu chuỗi thời gian, giúp nhận ra các xu hướng chính.
3. **Phổ biến và dễ sử dụng:** SMA được sử dụng rộng rãi trong phân tích kỹ thuật và tích hợp trong hầu hết các phần mềm phân tích và giao dịch chứng khoán.
4. **Cung cấp cái nhìn tổng quát:** SMA cung cấp cái nhìn tổng quát về xu hướng dữ liệu trong một khoảng thời gian nhất định, hỗ trợ quyết định dựa trên xu hướng dài hạn.

- **Nhược điểm**

1. **Phản ứng chậm với thay đổi mới:** Do SMA dựa trên các giá trị quá khứ, nó có thể phản ứng chậm với những thay đổi mới nhất trong dữ liệu, bỏ lỡ cơ hội giao dịch ngắn hạn.
2. **Độ trễ (Lag):** SMA luôn có độ trễ so với dữ liệu thực tế, đặc biệt là khi sử dụng với khoảng thời gian lớn, làm giảm tính nhạy bén đối với biến động ngắn hạn.
3. **Trọng số không phản ánh giá trị mới:** SMA coi các giá trị trong khoảng thời gian tính toán có trọng số bằng nhau, không phản ánh đúng tầm quan trọng của các giá trị mới nhất.
4. **Bị ảnh hưởng bởi giá trị cực đoan:** SMA có thể bị ảnh hưởng bởi các giá trị cực đoan hoặc ngoại lệ trong dữ liệu, dẫn đến việc làm sai lệch trung bình và xu hướng thực sự.
5. **Không hiệu quả trong thị trường không có xu hướng:** Trong các thị trường không có xu hướng rõ ràng hoặc dao động ngang, SMA có thể đưa ra các tín

hiệu sai lệch, gây khó khăn trong việc đưa ra quyết định giao dịch chính xác.

3.2.2. Thực nghiệm

Sử dụng kết quả là biến `transformed_data` từ bước tiền xử lý dữ liệu để thực hiện.

```
# Create pipeline
pipeline = Pipeline(stages=indexers + [encoder, assembler])

# Fit and transform data
pipeline_model = pipeline.fit(df_cleaned)
transformed_data = pipeline_model.transform(df_cleaned)
```

1. Đánh giá xu hướng tổng quan của `max_temp` và `precipitation`

```
# Define a function to calculate the SMA for a given window size
def calculate_sma(data, window_size):
    sma_values = []
    for i in range(len(data)):
        if i < window_size - 1:
            sma_values.append(None) # Not enough data points to fill the window
        else:
            window = [val for val in data[i-window_size+1:i+1] if val is not None]
            if not window:
                sma_values.append(None) # If all values in the window are None
            else:
                window_avg = sum(window) / len(window)
                sma_values.append(window_avg)
    return sma_values
```

- **def calculate_sma(data, window_size):** Định nghĩa hàm để tính SMA (Simple Moving Average) cho một khoảng thời gian xác định.
 - Hàm nhận vào hai tham số: **data** là danh sách các giá trị dữ liệu và **window_size** là kích thước cửa sổ để tính SMA.
- **sma_values = []:** Khởi tạo danh sách rỗng để lưu trữ các giá trị SMA.
- **for i in range(len(data)):** Duyệt qua từng phần tử trong dữ liệu.
- **if i < window_size - 1:** Kiểm tra nếu không đủ số điểm dữ liệu để tính toán SMA.
 - Nếu chỉ số hiện tại nhỏ hơn **window_size - 1**, thêm **None** vào danh sách

sma_values vì không đủ dữ liệu để tính SMA.

- **else::** Nếu đủ số điểm dữ liệu để tính toán SMA.
 - **window = [val for val in data[i-window_size+1:i+1] if val is not None]**: Tạo danh sách cửa sổ chứa các giá trị dữ liệu không phải None cho khoảng thời gian hiện tại.
 - Lấy các giá trị từ **data[i-window_size+1]** đến **data[i]** và loại bỏ các giá trị **None**.
 - **if not window::** Kiểm tra nếu tất cả các giá trị trong cửa sổ là None.
 - Nếu danh sách **window** rỗng, thêm **None** vào danh sách **sma_values** vì không có giá trị hợp lệ để tính SMA.
 - **else::** Nếu cửa sổ chứa các giá trị hợp lệ.
 - **window_avg = sum(window) / len(window)**: Tính trung bình của các giá trị trong cửa sổ.
 - **sma_values.append(window_avg)**: Thêm giá trị SMA đã tính vào danh sách **sma_values**.
- **return sma_values**: Trả về danh sách các giá trị SMA

```
# Helper function to calculate SMA for each partition
def calculate_sma_partition(iterator, window_size):
    data = list(iterator)
    temp_max_values = [float(row[1]) for row in data] # Convert to float
    precipitation_values = [float(row[2]) for row in data] # Convert to float
    dates = [row[0] for row in data]
    sma_temp_max = calculate_sma(temp_max_values, window_size)
    sma_precipitation = calculate_sma(precipitation_values, window_size)

    result = zip(dates, sma_temp_max, sma_precipitation)
    return result
```

- **def calculate_sma_partition(iterator, window_size)::** Định nghĩa hàm để tính SMA cho từng phân vùng của dữ liệu.
 - Hàm nhận vào hai tham số: **iterator** là một iterator chứa dữ liệu của phân vùng và **window_size** là kích thước cửa sổ để tính SMA.

- **data = list(iterator)**: Chuyển iterator thành danh sách để dễ dàng xử lý.
- **temp_max_values = [float(row[1]) for row in data]**: Chuyển đổi giá trị nhiệt độ tối đa từ chuỗi sang kiểu float.
- **precipitation_values = [float(row[2]) for row in data]**: Chuyển đổi giá trị lượng mưa từ chuỗi sang kiểu float.
- **dates = [row[0] for row in data]**: Lấy danh sách ngày từ dữ liệu.
- **sma_temp_max = calculate_sma(temp_max_values, window_size)**: Tính SMA cho giá trị nhiệt độ tối đa.
- **sma_precipitation = calculate_sma(precipitation_values, window_size)**: Tính SMA cho giá trị lượng mưa.
- **result = zip(dates, sma_temp_max, sma_precipitation)**: Ghép các giá trị ngày, SMA nhiệt độ tối đa và SMA lượng mưa lại với nhau.
- **return result**: Trả về kết quả.

```
# Extract date, temp_max, precipitation and weather_index from the transformed data
rdd = transformed_data.select("date", "features", "weather_index").rdd
data_rdd = rdd.map(lambda row: (row['date'], row['features'][1], row['features'][0], row['weather_index']))
# Sort the data
data_rdd = data_rdd.sortBy(lambda row: row[0])
# Extract data to variables using map
dates_rdd = data_rdd.map(lambda row: row[0])
temp_max_rdd = data_rdd.map(lambda row: (row[0], float(row[1]))) # Convert to float
precipitation_rdd = data_rdd.map(lambda row: (row[0], float(row[2]))) # Convert to float
weather_indices_rdd = data_rdd.map(lambda row: (row[0], float(row[3]))) # convert to float
```

- **rdd = transformed_data.select("date", "features", "weather_index").rdd**: Trích xuất các cột "date", "features", "weather_index" từ dữ liệu đã biến đổi và chuyển đổi thành RDD.
 - **.rdd**: Chuyển đổi DataFrame thành RDD.
- **data_rdd = rdd.map(lambda row: (row['date'], row['features'][1], row['features'][0], row['weather_index']))**: Ánh xạ các hàng của RDD thành bộ giá trị chứa ngày, nhiệt độ tối đa, lượng mưa, và chỉ số thời tiết.
 - **row['date']**: Lấy giá trị ngày.
 - **row['features'][1]**: Lấy giá trị nhiệt độ tối đa.
 - **row['features'][0]**: Lấy giá trị lượng mưa.

- **row['weather_index']**: Lấy chỉ số thời tiết.
- **data_rdd = data_rdd.sortBy(lambda row: row[0])**: Sắp xếp RDD theo ngày.
- **dates_rdd = data_rdd.map(lambda row: row[0])**: Trích xuất cột ngày từ **data_rdd**.
- **temp_max_rdd = data_rdd.map(lambda row: (row[0], float(row[1])))**: Trích xuất và chuyển đổi giá trị nhiệt độ tối đa từ chuỗi sang số thực.
- **precipitation_rdd = data_rdd.map(lambda row: (row[0], float(row[2])))**: Trích xuất và chuyển đổi giá trị lượng mưa từ chuỗi sang số thực.
- **weather_indices_rdd = data_rdd.map(lambda row: (row[0], float(row[3])))**: Trích xuất và chuyển đổi giá trị chỉ số thời tiết từ chuỗi sang số thực.

```
# Calculate the SMA using Spark's mapPartitions function
window_size = 200
sma_rdd = data_rdd.mapPartitions(lambda iterator: calculate_sma_partition(iterator, window_size))

sma_temp_max_rdd = sma_rdd.map(lambda row: (row[0], row[1]))
sma_precipitation_rdd = sma_rdd.map(lambda row: (row[0], row[2]))
```

- **window_size = 200**: Đặt kích thước cửa sổ để tính SMA là 200.
- **sma_rdd = data_rdd.mapPartitions(lambda iterator: calculate_sma_partition(iterator, window_size))**: Tính SMA bằng hàm **mapPartitions** của Spark.
 - **mapPartitions(lambda iterator: calculate_sma_partition(iterator, window_size))**: Ánh xạ từng phân vùng của RDD qua hàm **calculate_sma_partition**.
 - **lambda iterator: calculate_sma_partition(iterator, window_size)**: Sử dụng hàm lambda để truyền từng iterator (phân vùng dữ liệu) và **window_size** vào hàm **calculate_sma_partition**.
- **sma_temp_max_rdd = sma_rdd.map(lambda row: (row[0], row[1]))**:

Trích xuất giá trị SMA nhiệt độ tối đa từ **sma_rdd**.

- **sma_precipitation_rdd = sma_rdd.map(lambda row: (row[0], row[2])):**

Trích xuất giá trị SMA lượng mưa từ **sma_rdd**.

```
# Define the custom schema
from pyspark.sql.types import StructType, StructField, DateType, FloatType

# Schema for sma_temp_max_rdd and sma_precipitation_rdd
schema = StructType([
    StructField("date", DateType(), True),
    StructField("value", FloatType(), True)
])
```

- **from pyspark.sql.types import StructType, StructField, DateType, FloatType:** Nhập các kiểu dữ liệu cần thiết từ PySpark để định nghĩa schema.
 - **StructType:** Để định nghĩa một cấu trúc schema.
 - **StructField:** Để định nghĩa từng trường trong schema.
 - **DateType:** Kiểu dữ liệu ngày tháng.
 - **FloatType:** Kiểu dữ liệu số thực (float).
- **schema = StructType([:** Khởi tạo một schema mới với cấu trúc xác định.
 - **StructField("date", DateType(), True):** Định nghĩa một trường có tên "date" với kiểu dữ liệu **DateType**.
 - **"date":** Tên của trường.
 - **DateType():** Kiểu dữ liệu của trường là ngày tháng.
 - **True:** Trường này có thể có giá trị null.
 - **StructField("value", FloatType(), True):** Định nghĩa một trường có tên "value" với kiểu dữ liệu **FloatType**.
 - **"value":** Tên của trường.
 - **FloatType():** Kiểu dữ liệu của trường là số thực.
 - **True:** Trường này có thể có giá trị null

```
def converter(values_rdd, schema, withDate):  
    values_df = spark.createDataFrame(values_rdd, schema)  
    values = values_df.select("value").rdd.flatMap(lambda x: x).collect()  
    if (withDate):  
        dates = values_df.select("date").rdd.flatMap(lambda x: x).collect()  
        return dates, values  
    return values
```

- **def converter(values_rdd, schema, withDate):**: Định nghĩa hàm **converter** nhận ba tham số: **values_rdd** là RDD chứa dữ liệu, **schema** là cấu trúc dữ liệu để tạo DataFrame, và **withDate** chỉ định liệu hàm sẽ trả về cả danh sách ngày hay không.
- **values_df = spark.createDataFrame(values_rdd, schema)**: Tạo DataFrame từ RDD **values_rdd** và cấu trúc **schema**.
- **values = values_df.select("value").rdd.flatMap(lambda x: x).collect()**: Lấy danh sách các giá trị từ cột "value" của DataFrame.
 - **.select("value")**: Chọn cột "value".
 - **.rdd**: Chuyển đổi DataFrame thành RDD để sử dụng **flatMap**.
 - **.flatMap(lambda x: x)**: Đảm bảo mỗi phần tử trong RDD là một giá trị duy nhất, loại bỏ các cấu trúc phụ bên trong.
 - **.collect()**: Thu thập tất cả các phần tử trong RDD vào một danh sách Python.
- **if (withDate):**: Kiểm tra xem có cần trả về cả danh sách ngày hay không.
 - **dates = values_df.select("date").rdd.flatMap(lambda x: x).collect()**: Nếu cần, lấy danh sách các giá trị từ cột "date" của DataFrame.
- **return dates, values**: Trả về cả danh sách ngày và giá trị nếu **withDate** là **True**.
- **return values**: Trả về chỉ danh sách giá trị nếu không cần ngày

```
# Calculate and plotting
def longterm_plot(values_rdd, sma_values_rdd, title, schema):
    dates, values = converter(values_rdd, schema, True)
    sma_values = converter(sma_values_rdd, schema, False)
    # Plot the original temp_max values and SMA values using Matplotlib
    plt.figure(figsize=(12, 6))
    plt.plot(dates, values, label=title)
    plt.plot(dates, sma_values, label=f'SMA (window size = {window_size})')
    plt.xlabel('Date')
    plt.ylabel(title)
    plt.title(f'Simple Moving Average (SMA) of ' + title)
    plt.legend()
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
    # Provide insights based on the SMA
    # Filter out None values from SMA for comparison
    valid_sma_values = [sma for sma in sma_values if sma is not None]
    # Insight 2: Volatility observation
    volatility = sum([abs(values[i] - sma_values[i]) for i in range(window_size - 1, len(sma_values)) if sma_values[i] is not None])
    average_volatility = volatility / len(valid_sma_values)
    print(f"The average volatility in the {title} over the observed period is {average_volatility:.2f}.")
```

- **dates, values = converter(values_rdd, schema, True):** Gọi hàm **converter** để lấy danh sách ngày và giá trị từ RDD **values_rdd**.
- **sma_values = converter(sma_values_rdd, schema, False):** Gọi hàm **converter** để lấy danh sách giá trị SMA từ RDD **sma_values_rdd**.
- **plt.plot(dates, values, label=title):** Vẽ đường dữ liệu gốc trên biểu đồ với nhãn là **title**.
- **plt.plot(dates, sma_values, label=f'SMA (window size = {window_size})'):** Vẽ đường SMA trên biểu đồ với kích thước cửa sổ là **window_size**.
- **volatility = sum([abs(values[i] - sma_values[i]) for i in range(window_size - 1, len(sma_values)) if sma_values[i] is not None]):** Tính sự biến động bằng cách tính tổng các hiệu giữa giá trị gốc và giá trị SMA, sau đó chia cho số lượng giá trị SMA không phải là None để tránh sai sót do các giá trị SMA bị thiếu

```
▶ longterm_plot(temp_max_rdd, sma_temp_max_rdd, 'Temperature', schema)
```

```
▶ longterm_plot(precipitation_rdd, sma_precipitation_rdd, 'Precipitation', schema)
```

- Sau cùng ta chạy 2 câu lệnh để plot ra biểu đồ kết quả cho 2 giá trị phân tích là Temp Max và Precipitation.

2. Đưa ra nhận xét dự đoán về xu hướng nằm ngoài phạm vi dataset

```
from pyspark.sql.types import StructType, StructField, DateType, FloatType

# Calculate the SMA using Spark's mapPartitions function
window_size_short = 20
window_size_long = 200
sma_short_rdd = data_rdd.mapPartitions(lambda iterator: calculate_sma_partition(iterator, window_size_short))
sma_long_rdd = data_rdd.mapPartitions(lambda iterator: calculate_sma_partition(iterator, window_size_long))

# Extract data to variables using map
sma_temp_max_short_rdd = sma_short_rdd.map(lambda row: (row[0], row[1]))
sma_temp_max_long_rdd = sma_long_rdd.map(lambda row: (row[0], row[1]))
```

- **from pyspark.sql.types import StructType, StructField, DateType, FloatType**: Nhập các kiểu dữ liệu cần thiết từ PySpark để định nghĩa schema cho dữ liệu.
- **window_size_short = 20** và **window_size_long = 200**: Đặt kích thước cửa sổ để tính SMA là 20 và 200, tương ứng với SMA ngắn hạn và dài hạn.
- **sma_short_rdd = data_rdd.mapPartitions(lambda iterator: calculate_sma_partition(iterator, window_size_short))** và **sma_long_rdd = data_rdd.mapPartitions(lambda iterator: calculate_sma_partition(iterator, window_size_long))**: Tính SMA sử dụng hàm **mapPartitions** của Spark cho cả SMA ngắn hạn và dài hạn.
 - Sử dụng hàm **calculate_sma_partition** để tính toán SMA cho mỗi phân vùng dữ liệu.
 - **window_size_short** và **window_size_long** được truyền vào để xác định kích thước cửa sổ cho SMA.
- **sma_temp_max_short_rdd = sma_short_rdd.map(lambda row: (row[0], row[1]))** và **sma_temp_max_long_rdd = sma_long_rdd.map(lambda row: (row[0], row[1]))**: Trích xuất dữ liệu nhiệt độ tối đa từ kết quả SMA ngắn hạn và dài hạn thành biến riêng cho mỗi loại SMA


```
# Calculate last 10% rdd
def get_last_10_percent(rdd):
    # Step 1: Determine the total number of elements in the RDD
    total_count = rdd.count()
    # Step 2: Calculate the number of elements corresponding to the last 10%
    percentage = 0.1
    num_elements_to_take = int(total_count * percentage)
    start_index = total_count - num_elements_to_take
    # Step 3: Use zipWithIndex to pair elements with their indices
    rdd_with_index = rdd.zipWithIndex()
    # Step 4: Filter elements based on the calculated starting index for the last 10%
    last_10_percent_rdd = rdd_with_index.filter(lambda x: x[1] >= start_index).map(lambda x: x[0])
    # Step 5: Use take() to retrieve the elements (this fetches the data to the driver)
    last_10_percent = last_10_percent_rdd.take(num_elements_to_take)
    return last_10_percent
```

- **total_count = rdd.count()**: Đếm tổng số phần tử trong RDD.
- **percentage = 0.1**: Xác định phần trăm cuối cùng mà bạn muốn lấy, trong trường hợp này là 10%.
- **num_elements_to_take = int(total_count * percentage)**: Tính số lượng phần tử tương ứng với 10% cuối cùng của RDD.
- **start_index = total_count - num_elements_to_take**: Tính chỉ số bắt đầu cho phần cuối cùng mà bạn muốn lấy.
- **rdd_with_index = rdd.zipWithIndex()**: Ghép mỗi phần tử của RDD với chỉ số của nó.
- **last_10_percent_rdd = rdd_with_index.filter(lambda x: x[1] >= start_index).map(lambda x: x[0])**: Lọc các phần tử dựa trên chỉ số bắt đầu được tính toán cho 10% cuối cùng và loại bỏ chỉ số, chỉ giữ lại các phần tử.
- **last_10_percent = last_10_percent_rdd.take(num_elements_to_take)**: Sử dụng **take()** để lấy các phần tử cuối cùng (trích xuất dữ liệu về máy driver) dựa trên số lượng phần tử đã tính toán trước đó

```
last_values_rdd = get_last_10_percent(temp_max_rdd)
last_sma_temp_max_short_rdd = get_last_10_percent(sma_temp_max_short_rdd)
last_sma_temp_max_long_rdd = get_last_10_percent(sma_temp_max_long_rdd)

dates, values = converter(last_values_rdd, schema, True)
sma_values1 = converter(last_sma_temp_max_short_rdd, schema, False)
sma_values2 = converter(last_sma_temp_max_long_rdd, schema, False)
```

- **last_values_rdd = get_last_10_percent(temp_max_rdd)**: Lấy ra 10% cuối cùng của dữ liệu nhiệt độ tối đa từ RDD **temp_max_rdd**.
- **last_sma_temp_max_short_rdd** = **get_last_10_percent(sma_temp_max_short_rdd)** và **last_sma_temp_max_long_rdd** = **get_last_10_percent(sma_temp_max_long_rdd)**: Tương tự, lấy ra 10% cuối cùng của giá trị SMA ngắn hạn và dài hạn từ các RDD tương ứng.
- **dates, values = converter(last_values_rdd, schema, True)**: Chuyển đổi 10% cuối cùng của dữ liệu nhiệt độ tối đa thành danh sách ngày và giá trị.
- **sma_values1 = converter(last_sma_temp_max_short_rdd, schema, False)** và **sma_values2 = converter(last_sma_temp_max_long_rdd, schema, False)**: Tương tự, chuyển đổi 10% cuối cùng của các giá trị SMA ngắn hạn và dài hạn thành danh sách giá trị

```
# Plot the original values and SMA values using Matplotlib
plt.figure(figsize=(12, 6))
plt.plot(dates, values, label = "Temp_Max")
plt.plot(dates, sma_values1, label=f'SMA (window size = {window_size_short})')
plt.plot(dates, sma_values2, label=f'SMA (window size = {window_size_long})')
plt.xlabel('Date')
plt.ylabel("Temp_Max")
plt.title(f'Simple Moving Average (SMA) - Nearest 10% dates ')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

- Plot các giá trị lên trên hình

3. Dự đoán xu hướng và kiểm tra các điểm bất thường

```
# Continue from the previous steps
window_size = 3
sma_final_rdd = data_rdd.mapPartitions(lambda iterator: calculate_sma_partition(iterator, window_size))
sma_temp_max_final_rdd = sma_final_rdd.map(lambda row: (row[0], row[1]))

sma_values = converter(sma_temp_max_final_rdd, schema, False)
dates, temp_max_values = converter(temp_max_rdd, schema, True)
weather_indices = converter(weather_indices_rdd, schema, False)
```

- **window_size = 3**: Đặt kích thước cửa sổ là 3 cho SMA.
- **sma_final_rdd = data_rdd.mapPartitions(lambda iterator: calculate_sma_partition(iterator, window_size))**: Tính toán SMA bằng cách sử dụng kích thước cửa sổ là 3.
- **sma_temp_max_final_rdd = sma_final_rdd.map(lambda row: (row[0], row[1]))**: Trích xuất giá trị SMA của nhiệt độ tối đa từ RDD
- **sma_values = converter(sma_temp_max_final_rdd, schema, False)**: Chuyển đổi danh sách giá trị SMA thành dạng Python.
- **dates, temp_max_values = converter(temp_max_rdd, schema, True)**: Chuyển đổi danh sách ngày và giá trị nhiệt độ tối đa thành dạng Python.
- **weather_indices = converter(weather_indices_rdd, schema, False)**: Chuyển đổi danh sách chỉ số thời tiết thành dạng Python

```
# Detect periods of increase or decrease in the SMA
def detect_trends(sma_values):
    trends = []
    for i in range(1, len(sma_values)):
        if sma_values[i] is not None and sma_values[i-1] is not None:
            if sma_values[i] > sma_values[i-1]:
                trends.append('increasing')
            elif sma_values[i] < sma_values[i-1]:
                trends.append('decreasing')
            else:
                trends.append('stable')
        else:
            trends.append(None)
    return trends
```

- **def detect_trends(sma_values)::** Định nghĩa hàm **detect_trends** nhận vào

một danh sách các giá trị SMA (**sma_values**).

- **trends = []**: Khởi tạo một danh sách rỗng để lưu trữ các xu hướng.
- **for i in range(1, len(sma_values))**:: Bắt đầu một vòng lặp từ phần tử thứ hai đến phần tử cuối cùng của **sma_values**
 - **if sma_values[i] is not None and sma_values[i-1] is not None**:: Kiểm tra xem cả giá trị hiện tại (**sma_values[i]**) và giá trị trước đó (**sma_values[i-1]**) có khác **None** hay không
 - **if sma_values[i] > sma_values[i-1]**:: Nếu giá trị hiện tại lớn hơn giá trị trước đó, thêm 'increasing' vào danh sách **trends**
 - **elif sma_values[i] < sma_values[i-1]**:: Nếu giá trị hiện tại nhỏ hơn giá trị trước đó, thêm 'decreasing' vào danh sách **trends**
 - **else**:: Nếu giá trị hiện tại bằng giá trị trước đó, thêm 'stable' vào danh sách **trends**
 - **else**:: Nếu một trong hai giá trị là **None**, thêm **None** vào danh sách **trends**

```
# Identify anomalies where the temp_max deviates significantly from the SMA
def detect_anomalies(temp_values, sma_values, threshold=5.0):
    anomalies = []
    for i in range(len(temp_values)):
        if sma_values[i] is not None and abs(temp_values[i] - sma_values[i]) > threshold:
            anomalies.append((dates[i], temp_values[i]))
    return anomalies
```

- **def detect_anomalies(temp_values, sma_values, threshold=5.0)**:: Định nghĩa hàm **detect_anomalies** nhận vào ba tham số: danh sách giá trị nhiệt độ (**temp_values**), danh sách giá trị SMA (**sma_values**), và ngưỡng (**threshold**) mặc định là 5.0.
- **anomalies = []**: Khởi tạo một danh sách rỗng để lưu trữ các điểm dữ liệu bất thường (**anomalies**)
- **for i in range(len(temp_values))**:: Bắt đầu một vòng lặp qua tất cả các chỉ số của **temp_values**

- **if sma_values[i] is not None and abs(temp_values[i] - sma_values[i]) > threshold::** Kiểm tra nếu giá trị SMA tại chỉ số **i** khác **None** và độ lệch tuyệt đối giữa giá trị nhiệt độ và giá trị SMA lớn hơn ngưỡng **threshold**
- **anomalies.append((dates[i], temp_values[i])):** Nếu điều kiện thỏa mãn, thêm một tuple chứa ngày và giá trị nhiệt độ vào danh sách **anomalies**

```
# Detect trends in the SMA
trends = detect_trends(sma_values)
# Detect anomalies
anomalies = detect_anomalies(temp_max_values, sma_values)

# Print detected trends
print("Detected Trends in SMA:")
for date, trend in zip(dates[1:], trends[1:]):
    if trend is not None:
        print(f>Date: {date}, Trend: {trend}")

# Print detected anomalies
print("\nDetected Anomalies:")
for anomaly in anomalies:
    print(f>Date: {anomaly[0]}, Temp Max: {anomaly[1]}")
```

- Chạy các hàm tính toán và tính các giá trị **trend** và **anomalies**
- Sau đó chạy vòng lặp và in ra từng giá trị

```

# Plot trends and anomalies on the graph
plt.figure(figsize=(12, 6))
plt.plot(dates, temp_max_values, label='Temp Max')
plt.plot(dates, sma_values, label=f'SMA (window size = {window_size})')

# Highlight trends
for i in range(1, len(trends)):
    if trends[i] == 'increasing':
        plt.axvspan(dates[i-1], dates[i], color='green', alpha=0.3, lw=0)
    elif trends[i] == 'decreasing':
        plt.axvspan(dates[i-1], dates[i], color='red', alpha=0.3, lw=0)
# Highlight anomalies
for anomaly in anomalies:
    plt.plot(anomaly[0], anomaly[1], 'ro') # red dot for anomalies

plt.xlabel('Date')
plt.ylabel('Temperature Max')
plt.title('Simple Moving Average (SMA) of Temp Max with Trends and Anomalies')
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```

- Vẽ đồ thị đường cho giá trị thực tế của temp_max và giá trị sma_values
- In nổi bật các xu hướng trên hình vẽ
 - **if trends[i] == 'increasing':** Nếu xu hướng tăng thì
plt.axvspan(dates[i-1], dates[i], color='green', alpha=0.3, lw=0):
 Tô màu xanh lá nhạt
 - **elif trends[i] == 'decreasing':** Nếu xu hướng giảm thì
plt.axvspan(dates[i-1], dates[i], color='red', alpha=0.3, lw=0): Tô
 màu đỏ nhạt
- **plt.plot(anomaly[0], anomaly[1], 'ro')**: Vẽ điểm đỏ cho các điểm bất thường

```
# Analyze the relationship between weather index and detected trends/anomalies
trend_counts_by_weather = {weather: {'increasing': 0, 'decreasing': 0, 'stable': 0} for weather in set(weather_indices)}

for i in range(1, len(trends)):
    if trends[i] is not None:
        weather = weather_indices[i]
        trend_counts_by_weather[weather][trends[i]] += 1

# Print the trend counts by weather index
print("Trend Counts by Weather Index:")
for weather, counts in trend_counts_by_weather.items():
    print(f"Weather Index {weather}: {counts}")

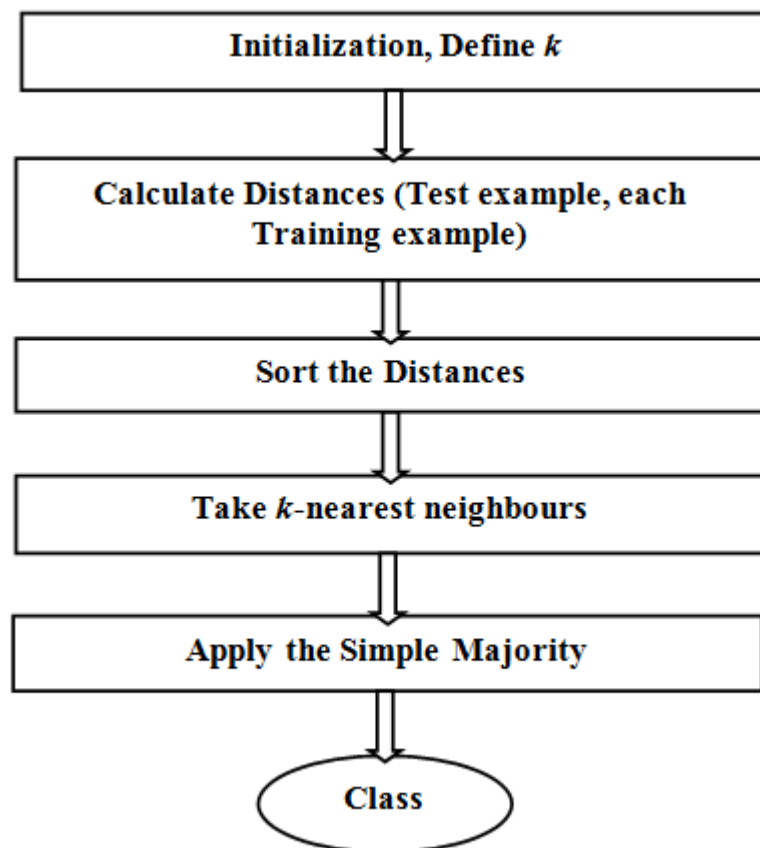
# Plot trend counts by weather index
plt.figure(figsize=(10, 6))
for trend_type in ['increasing', 'decreasing', 'stable']:
    plt.bar(trend_counts_by_weather.keys(),
            [trend_counts_by_weather[weather][trend_type] for weather in trend_counts_by_weather.keys()],
            label=trend_type)
plt.xlabel('Weather Index')
plt.ylabel('Count of Trends')
plt.title('Trend Counts by Weather Index')
plt.legend()
plt.grid(True)
plt.show()
```

- **trend_counts_by_weather = {weather: {'increasing': 0, 'decreasing': 0, 'stable': 0} for weather in set(weather_indices)}**: Tạo một dictionary với các chỉ số thời tiết khác nhau (từ **weather_indices**). Mỗi chỉ số thời tiết có một từ điển lồng nhau để đếm số lượng xu hướng ('increasing', 'decreasing', 'stable')
- Chạy vòng lặp để tính **trend_counts_by_weather[weather][trends[i]] += 1**: để đếm các mục increasing, decreasing, stable
- **for weather, counts in trend_counts_by_weather.items()**: Lặp qua các mục trong từ điển **trend_counts_by_weather** và in ra chỉ số increasing, decreasing và stable đối với từng mục **weather_index** (0.0 đến 4.0)
- Vẽ biểu đồ plot để minh họa
 - **for trend_type in ['increasing', 'decreasing', 'stable']**: Lặp qua các loại xu hướng.
 - **plt.bar(trend_counts_by_weather.keys(), [trend_counts_by_weather[weather][trend_type] for weather in trend_counts_by_weather.keys()], label=trend_type)**: Vẽ biểu đồ thanh cho từng loại với nhãn tương ứng

3.3. KNN

3.3.1. Cơ sở lý thuyết

Thuật toán K-Nearest Neighbors (KNN) là một thuật toán học máy không giám sát được sử dụng chủ yếu cho các bài toán phân loại và hồi quy. KNN hoạt động dựa trên nguyên tắc rằng các điểm dữ liệu gần nhau trong không gian đặc trưng có xu hướng có nhãn giống nhau. Thuật toán này không thực hiện bất kỳ giả định nào về cấu trúc của dữ liệu, do đó nó được coi là một trong những thuật toán đơn giản và dễ hiểu nhất.



Dưới đây là quy trình cơ bản của thuật toán KNN:

- Bước 1: Chọn Số Lượng Láng Giềng (K): Xác định số lượng láng giềng gần nhất (K) mà bạn muốn xem xét để đưa ra dự đoán.
- Bước 2: Tính Khoảng Cách: Đối với mỗi điểm dữ liệu cần phân loại, tính

khoảng cách giữa điểm đó và tất cả các điểm dữ liệu trong tập huấn luyện. Khoảng cách thường được tính bằng công thức khoảng cách Euclidean

- Công thức tính khoảng cách Euclidean giữa hai điểm:

$$distance(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

- Bước 3: Xác Định K Láng Giềng Gần Nhất: Lấy K điểm dữ liệu có khoảng cách nhỏ nhất từ điểm cần phân loại.
- Bước 4: Dự Đoán Nhãn: Đối với bài toán phân loại, dự đoán nhãn của điểm dữ liệu mới bằng cách lấy nhãn phổ biến nhất trong số K láng giềng gần nhất. Đối với bài toán hồi quy, dự đoán giá trị bằng cách lấy trung bình các giá trị của K láng giềng gần nhất.
- Bước 5: Lặp Lại: Thực hiện các bước trên cho tất cả các điểm dữ liệu cần phân loại hoặc hồi quy.

Ưu Điểm Và Nhược Điểm Của KNN

- Đơn Giản Và Dễ Hiểu: KNN là một thuật toán đơn giản, dễ hiểu và dễ triển khai.
- Hiệu Quả Với Dữ Liệu Ít: KNN có thể hoạt động hiệu quả với các tập dữ liệu nhỏ và các trường hợp mà mối quan hệ giữa các điểm dữ liệu có thể được biểu diễn qua khoảng cách.
- Tính Toán Chậm: KNN yêu cầu tính khoảng cách giữa điểm cần phân loại và tất cả các điểm trong tập huấn luyện, điều này có thể tốn nhiều thời gian khi xử lý các tập dữ liệu lớn.
- Nhạy Cảm Với Dữ Liệu Nhiễu: KNN có thể bị ảnh hưởng bởi các điểm dữ liệu nhiễu hoặc không liên quan.

3.3.2. Thực nghiệm

- Import lớp Row từ pyspark.sql để làm việc với các hàng dữ liệu trong DataFrame của Spark.
- Import hàm sqrt từ thư viện math để tính căn bậc hai, dùng trong công thức tính khoảng cách Euclidean.
- Import hàm itemgetter từ thư viện operator để lấy các phần tử trong danh sách dựa trên chỉ số.

```
from pyspark.sql import Row
from math import sqrt
from operator import itemgetter
|
```

- Chia data thành tập train và tập test theo tỉ lệ 7: 3

```
# Calculate the split index based on the desired ratio
split_index = int(sorted_data.count() * 0.7) # 70% training, 30% testing

# Split the data into training and testing sets
train_data = sorted_data.limit(split_index)
test_data = sorted_data.subtract(train_data)

# Define X_train, X_test, y_train, y_test
X_train = train_data.select("features")
y_train = train_data.select("weather_index")
X_test = test_data.select("features")
y_test = test_data.select("weather_index")
```

- Định nghĩa hàm euclidean_distance để tính khoảng cách Euclidean giữa hai điểm dữ liệu p1 và p2.
- Tính tổng bình phương của hiệu các tọa độ tương ứng của hai điểm và lấy căn bậc hai của tổng đó để ra khoảng cách Euclidean.

```
# Define a function to calculate Euclidean distance between two points
def euclidean_distance(p1, p2):
    return sqrt(sum([(p1[i] - p2[i]) ** 2 for i in range(len(p1))]))
```

- Chuyển đổi các hàng trong DataFrame X_train thành các tuple. Sử dụng phương thức toArray() để chuyển đổi vector đặc trưng thành mảng, sau đó chuyển thành tuple. Kết quả là danh sách các tuple đặc trưng cho tập huấn luyện.
- Chuyển đổi các nhãn trong DataFrame y_train thành danh sách các giá trị weather_index. Kết quả là danh sách các nhãn cho tập huấn luyện.

```
# Convert DataFrame rows to tuples
train_tuples = X_train.rdd.map(lambda row: tuple(row.features.toArray())).collect()
train_labels = y_train.rdd.map(lambda row: row.weather_index).collect()
```

- def knn_predict(test_point, k): Định nghĩa hàm knn_predict để dự đoán nhãn cho một điểm dữ liệu kiểm tra test_point dựa trên số lượng láng giềng gần nhất k.
- distances = []: Khởi tạo danh sách distances để lưu trữ khoảng cách từ điểm kiểm tra đến các điểm trong tập huấn luyện.
- for i, train_point in enumerate(train_tuples): Lặp qua từng điểm trong tập huấn luyện, sử dụng enumerate để lấy chỉ số và giá trị của từng điểm.
- distance = euclidean_distance(test_point, train_point): Tính khoảng cách Euclidean giữa điểm kiểm tra và điểm huấn luyện hiện tại.
- distances.append((i, distance)): Thêm chỉ số và khoảng cách của điểm huấn luyện vào danh sách distances.
- distances.sort(key=itemgetter(1)): Sắp xếp danh sách distances theo khoảng cách (giá trị thứ hai trong mỗi tuple).
- neighbors = [train_labels[i] for i, _ in distances[:k]]: Lấy nhãn của k điểm huấn luyện gần nhất.

- `return max(set(neighbors), key=neighbors.count)`: Trả về nhãn xuất hiện nhiều nhất trong số các cluster gần nhất.

```
# Define a function to predict labels for test data
def knn_predict(test_point, k):
    distances = []
    for i, train_point in enumerate(train_tuples):
        distance = euclidean_distance(test_point, train_point)
        distances.append((i, distance))
    distances.sort(key=itemgetter(1))
    neighbors = [train_labels[i] for i, _ in distances[:k]]
    return max(set(neighbors), key=neighbors.count)
```

Predict labels for test data: Đoạn mã này dự đoán nhãn cho dữ liệu kiểm tra.

- `k = 5`: Đặt số lượng láng giềng gần nhất (K) là 5. KNN sẽ xem xét 5 láng giềng gần nhất để đưa ra dự đoán.
- `X_test.rdd`: Chuyển đổi DataFrame `X_test` thành một RDD.
- `map(lambda row: knn_predict(tuple(row.features.toArray()), k))`: Áp dụng hàm `knn_predict` cho từng hàng của `X_test` để dự đoán nhãn. Chuyển đổi các đặc trưng của mỗi hàng thành một tuple và truyền vào hàm `knn_predict`.
- `test_predictions_rdd`: RDD chứa các nhãn dự đoán cho mỗi hàng trong `X_test`.

```
# Predict labels for test data
k = 5 # Number of neighbors
test_predictions_rdd = X_test.rdd.map(lambda row: knn_predict(tuple(row.features.toArray()), k))
```

So sánh nhãn dự đoán với nhãn thực tế.

- `test_labels_rdd = y_test.rdd.map(lambda row: row.weather_index)`:
- `y_test.rdd`: Chuyển đổi DataFrame `y_test` thành một RDD.
- `map(lambda row: row.weather_index)`: Trích xuất cột `weather_index` (nhãn thực tế) từ mỗi hàng trong `y_test`.
- `test_labels_rdd`: RDD chứa các nhãn thực tế.

```
# Compare predictions with actual labels
test_labels_rdd = y_test.rdd.map(lambda row: row.weather_index)
```

Kết hợp nhãn dự đoán và nhãn thực tế thành một RDD.

- combined_rdd = test_predictions_rdd.zip(test_labels_rdd):
- zip(test_labels_rdd): Ghép cặp từng phần tử của test_predictions_rdd và test_labels_rdd lại với nhau.
- combined_rdd: RDD chứa các cặp (dự đoán, nhãn thực tế).

```
# Combine test predictions and actual labels
combined_rdd = test_predictions_rdd.zip(test_labels_rdd)
```

Tính số lượng dự đoán chính xác.

- map(lambda x: 1 if x[0] == x[1] else 0): Áp dụng hàm lambda để kiểm tra nếu dự đoán (x[0]) bằng với nhãn thực tế (x[1]). Nếu đúng, trả về 1, nếu sai, trả về 0.
- RDD mới chứa 1 cho các dự đoán chính xác và 0 cho các dự đoán sai.
- reduce(lambda a, b: a + b): Tổng hợp tất cả các giá trị 1 và 0 để tính tổng số dự đoán chính xác.
- correct_predictions: Tổng số dự đoán chính xác.

```
# Calculate number of correct predictions
correct_predictions = combined_rdd.map(lambda x: 1 if x[0] == x[1] else 0).reduce(lambda a, b: a + b)
```

Tính độ chính xác của mô hình.

- total_predictions = combined_rdd.count():
- count(): Đếm tổng số cặp (dự đoán, nhãn thực tế) trong combined_rdd.
- total_predictions: Tổng số dự đoán.
- accuracy = correct_predictions / total_predictions:
- accuracy: Độ chính xác của mô hình.
- print("Test Accuracy:", accuracy):

```
# Calculate accuracy
total_predictions = combined_rdd.count()
accuracy = correct_predictions / total_predictions

print("Test Accuracy:", accuracy)
```

Test Accuracy: 0.8378995433789954

CHƯƠNG 4: SO SÁNH KẾT QUẢ ĐẠT ĐƯỢC

4.1. Phát biểu kết quả

4.1.1. K-Means

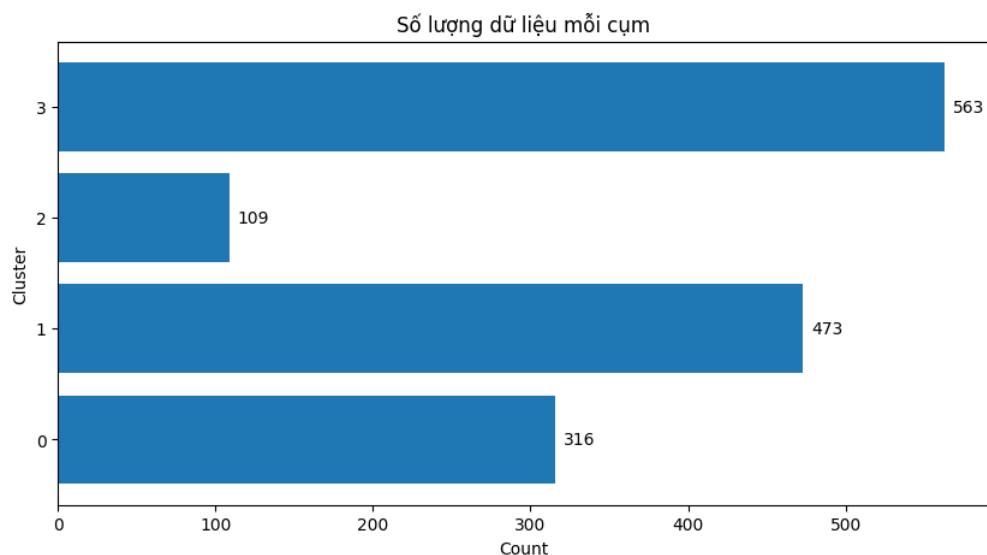
- ❖ Thống kê số lượng dữ liệu ở mỗi cụm

Nhóm collect() dữ liệu mục đích để trực quan kết quả sau khi phân cụm

```
# Thống kê số dữ liệu của mỗi cụm
df_tmp = df_result.groupBy('prediction').count().orderBy(['prediction'])
df_tmp.show()
clusters = df_tmp.select('prediction').rdd.flatMap(lambda r: r).collect()
cluster_counts = df_tmp.select('count').rdd.flatMap(lambda r: r).collect()
plt.figure(figsize=(10, 5))
bars = plt.barh(clusters, cluster_counts)
plt.yticks(clusters)
plt.xlabel('Count')
plt.ylabel('Cluster')
plt.title('Số lượng dữ liệu mỗi cụm')
for bar in bars:
    plt.text(bar.get_width() + 5, bar.get_y() + bar.get_height()/2, f'{bar.get_width():.0f}', va='center')
plt.show()
```

Số lượng dữ liệu lần lượt ở các cụm 0, 1, 2 và 3 là 316, 473, 109, 563

prediction	count
0	316
1	473
2	109
3	563



❖ Thống kê số lượng dữ liệu các loại thời tiết ở mỗi cụm

```
# Thống kê số dữ liệu mỗi loại thời tiết ở mỗi cụm
df_tmp = df_result.groupby('prediction', 'weather').count().orderBy('prediction', 'weather')
df_tmp.show()
pandas_df = df_tmp.toPandas()

# Create a figure and axes
plt.figure(figsize=(10, 7))

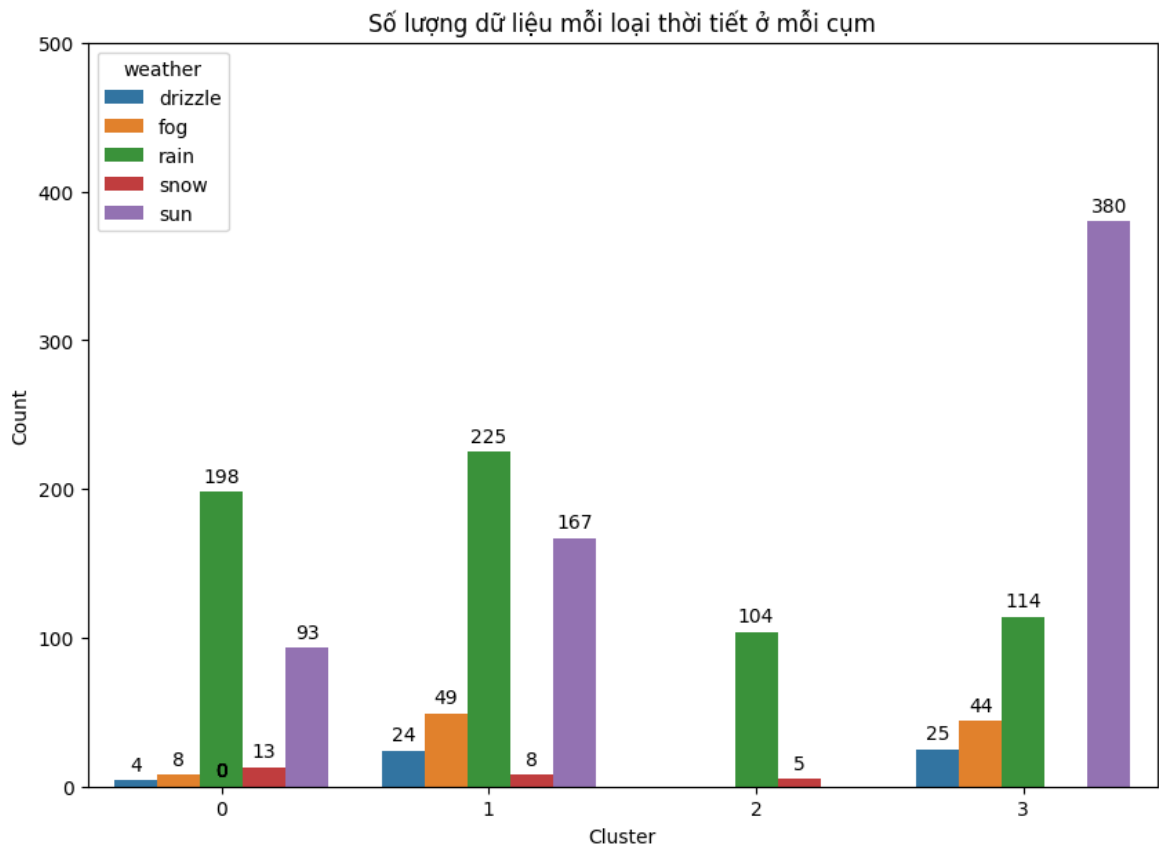
# Create a bar plot
barplot = sns.barplot(x='prediction', y='count', hue='weather', data=pandas_df)

# Set labels and title
plt.xlabel('Cluster')
plt.ylabel('Count')
plt.title('Số lượng dữ liệu mỗi loại thời tiết ở mỗi cụm')
plt.yticks(range(0, pandas_df['count'].max() + 200, 100))

for p in barplot.patches:
    barplot.annotate(format(p.get_height(), '.0f'),
                     (p.get_x() + p.get_width() / 2, p.get_height()),
                     ha = 'center', va = 'center',
                     xytext = (0, 8),
                     textcoords = 'offset points')

plt.show()
```

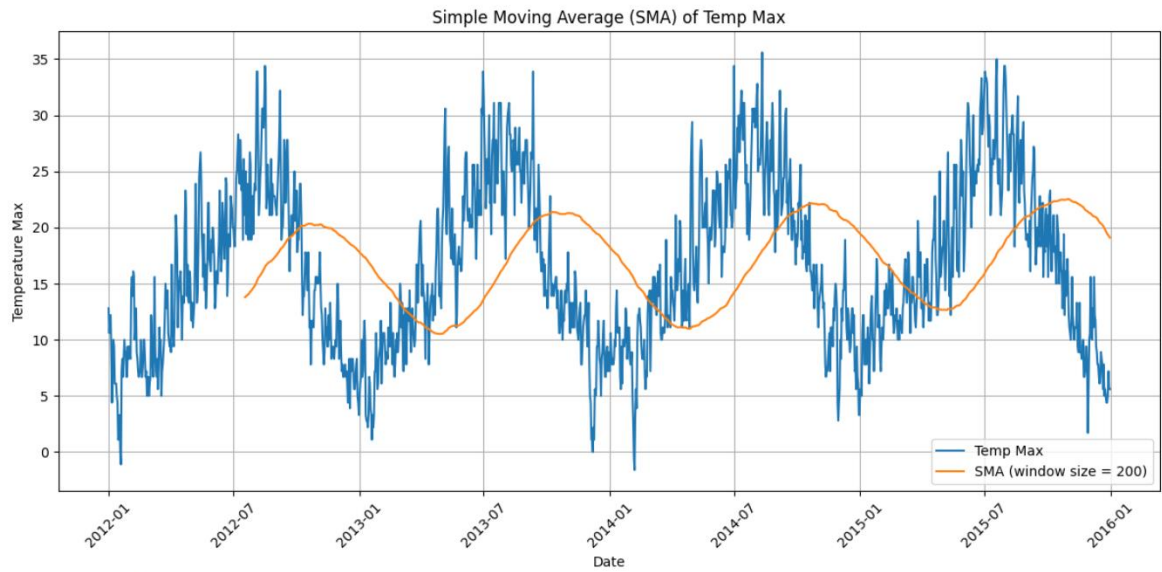
prediction	weather	count
0	drizzle	4
0	fog	8
0	rain	198
0	snow	13
0	sun	93
1	drizzle	24
1	fog	49
1	rain	225
1	snow	8
1	sun	167
2	rain	104
2	snow	5
3	drizzle	25
3	fog	44
3	rain	114
3	sun	380



4.1.2. Simple Moving Average (SMA)

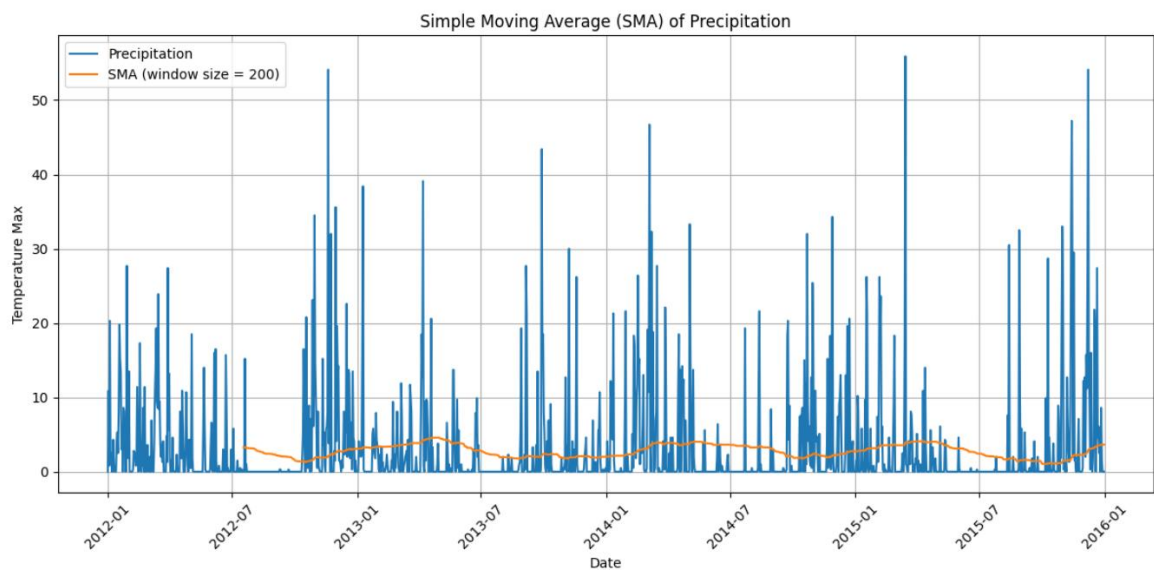
1. Đánh giá xu hướng tổng quan của max_temp và precipitation

Với window size = 200, SMA cho ra một đường biểu diễn biểu thị xu hướng chung của nhiệt độ tối đa qua các năm, với window size càng cao cho ra mức độ nhiễu thấp và đường cong mượt hơn, ổn định, từ đó dễ dự đoán các xu hướng dài hạn. Dựa vào hình có thể thấy các điểm cực đại và cực tiểu của đường SMA tăng dần theo các năm, điều đó chứng tỏ xu hướng nóng lên toàn cầu khi nhiệt độ tối đa qua các năm tăng ở khu vực được thống kê. Thống kê cũng cho thấy mức độ chênh lệch trung bình của giá trị thực tế và sma là 7.41.



The average volatility in the Temp Max over the observed period is 7.41.

Với trường thuộc tính là precipitation, đường sma cho ra kết quả khá cân bằng và bền vững, chưa thể hoàn toàn kết luận xu hướng tăng của lượng mưa trung bình qua các năm. Mức độ chênh lệch trung bình của giá trị thực tế và sma là 4.07



The average volatility in the Precipitation over the observed period is 4.07.

2. Đưa ra nhận xét dự đoán về xu hướng nằm ngoài phạm vi dataset

Large vs small moving average using 2 moving average

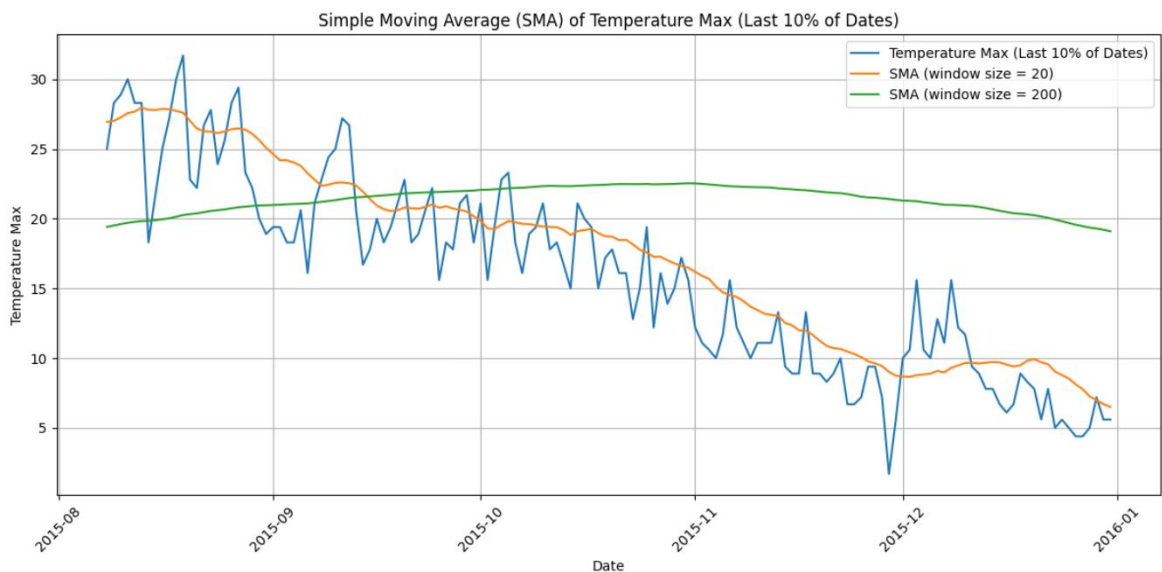
Reversal by moving average crossover

Ở đây ta sử dụng 2 đường SMA với window size = 20 và 200 lần lượt để đưa ra

dự đoán xu hướng. Nếu chỉ xét riêng đường màu cam (window size = 20), nhận thấy phần lớn giá trị temp_max nằm dưới đường màu cam, ta nói giá trị có xu hướng giảm

Tuy nhiên để mang tính chính xác và rõ ràng hơn, sử dụng kết hợp 2 đường với 2 window-size khác nhau cho cái nhìn tổng quát hơn khi so sánh giữa dài hạn và ngắn hạn. Ở đây ta gọi đường SMA với winsize = 20 và winsize = 200 lần lượt là MA nhỏ và MA lớn.

- Nhận thấy MA lớn vẫn đang tăng cho đến giữa tháng 10/2025
- MA nhỏ giảm trong khoảng thời gian
- Đánh giá xu hướng bằng cách xét giao điểm, tại giao điểm giữa tháng 9/2025 và 10/2025 giữa 2 đường cho biết rằng thời gian trước đó MA nhỏ > MA lớn suy ra temp_max có xu hướng tăng, ngược lại sau đó MA nhỏ < MA lớn lúc này temp_max đã biểu thị xu hướng giảm.
- Điểm giao giữa MA nhỏ và lớn gọi là điểm đảo ngược



3. Dự đoán xu hướng và kiểm tra các điểm bất thường

Khi tính toán và in ra dự đoán về trend thì kết quả có dạng, vì kết quả chạy ra khá nhiều, ở đây minh họa một phần kết quả:



Detected Trends in SMA:

Date: 2012-01-03, Trend: decreasing
Date: 2012-01-04, Trend: decreasing
Date: 2012-01-05, Trend: decreasing
Date: 2012-01-06, Trend: decreasing
Date: 2012-01-07, Trend: increasing
Date: 2012-01-08, Trend: increasing
Date: 2012-01-09, Trend: decreasing
Date: 2012-01-10, Trend: decreasing
Date: 2012-01-11, Trend: decreasing
Date: 2012-01-12, Trend: decreasing
Date: 2012-01-13, Trend: decreasing
Date: 2012-01-14, Trend: decreasing
Date: 2012-01-15, Trend: decreasing
Date: 2012-01-16, Trend: decreasing
Date: 2012-01-17, Trend: decreasing
Date: 2012-01-18, Trend: decreasing
Date: 2012-01-19, Trend: increasing
Date: 2012-01-20, Trend: increasing

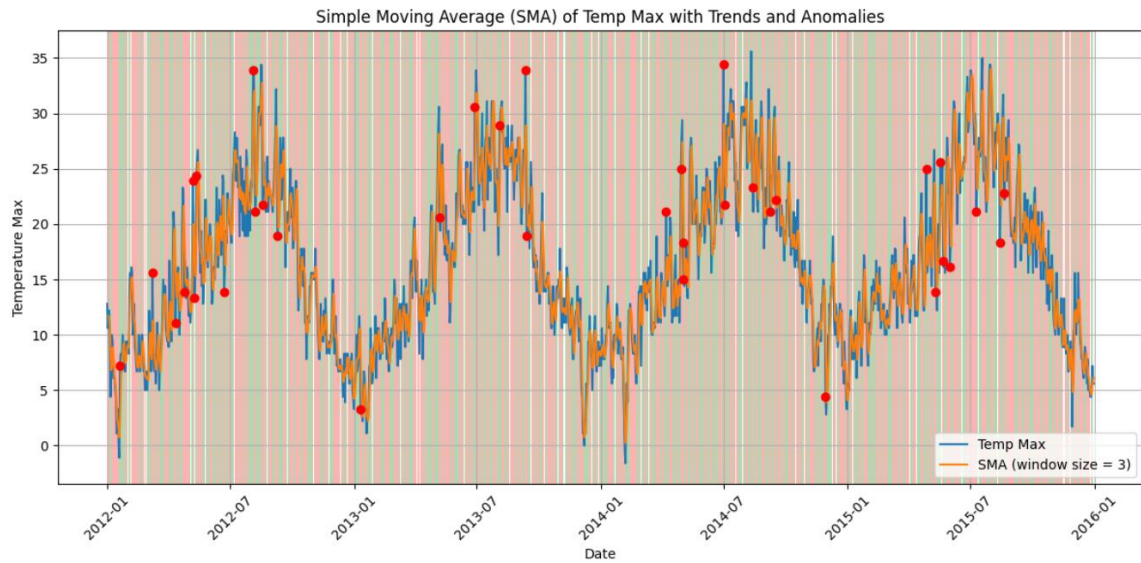
Kết quả các điểm dị biệt:

Detected Anomalies:

Date: 2012-01-20, Temp Max: 7.2
Date: 2012-03-08, Temp Max: 15.6
Date: 2012-04-11, Temp Max: 11.1
Date: 2012-04-24, Temp Max: 13.9
Date: 2012-05-07, Temp Max: 23.9
Date: 2012-05-09, Temp Max: 13.3
Date: 2012-05-12, Temp Max: 24.4
Date: 2012-06-22, Temp Max: 13.9
Date: 2012-08-04, Temp Max: 33.9
Date: 2012-08-07, Temp Max: 21.1
Date: 2012-08-18, Temp Max: 21.7
Date: 2012-09-09, Temp Max: 18.9
Date: 2013-01-10, Temp Max: 3.3
Date: 2013-05-07, Temp Max: 20.6

Date: 2013-06-28, Temp Max: 30.6
Date: 2013-08-04, Temp Max: 28.9
Date: 2013-09-11, Temp Max: 33.9
Date: 2013-09-13, Temp Max: 18.9
Date: 2014-04-07, Temp Max: 21.1
Date: 2014-04-29, Temp Max: 25.0
Date: 2014-05-02, Temp Max: 18.3
Date: 2014-05-03, Temp Max: 15.0
Date: 2014-07-01, Temp Max: 34.4
Date: 2014-07-03, Temp Max: 21.7
Date: 2014-08-13, Temp Max: 23.3
Date: 2014-09-08, Temp Max: 21.1
Date: 2014-09-16, Temp Max: 22.2
Date: 2014-11-29, Temp Max: 4.4
Date: 2015-04-27, Temp Max: 25.0
Date: 2015-05-11, Temp Max: 13.9
Date: 2015-05-18, Temp Max: 25.6
Date: 2015-05-22, Temp Max: 16.7
Date: 2015-06-01, Temp Max: 16.1
Date: 2015-07-10, Temp Max: 21.1
Date: 2015-08-14, Temp Max: 18.3
Date: 2015-08-20, Temp Max: 22.8

Biểu thị SMA, giá trị temp_max, trends và anomalies lên trên hình biểu diễn. Hình vẽ mô tả được rõ ràng các điểm dị biệt khi sử dụng SMA và threshold là 5.0, cũng như đánh giá được xu hướng chung là tăng hay giảm dựa trên màu xanh và đỏ.



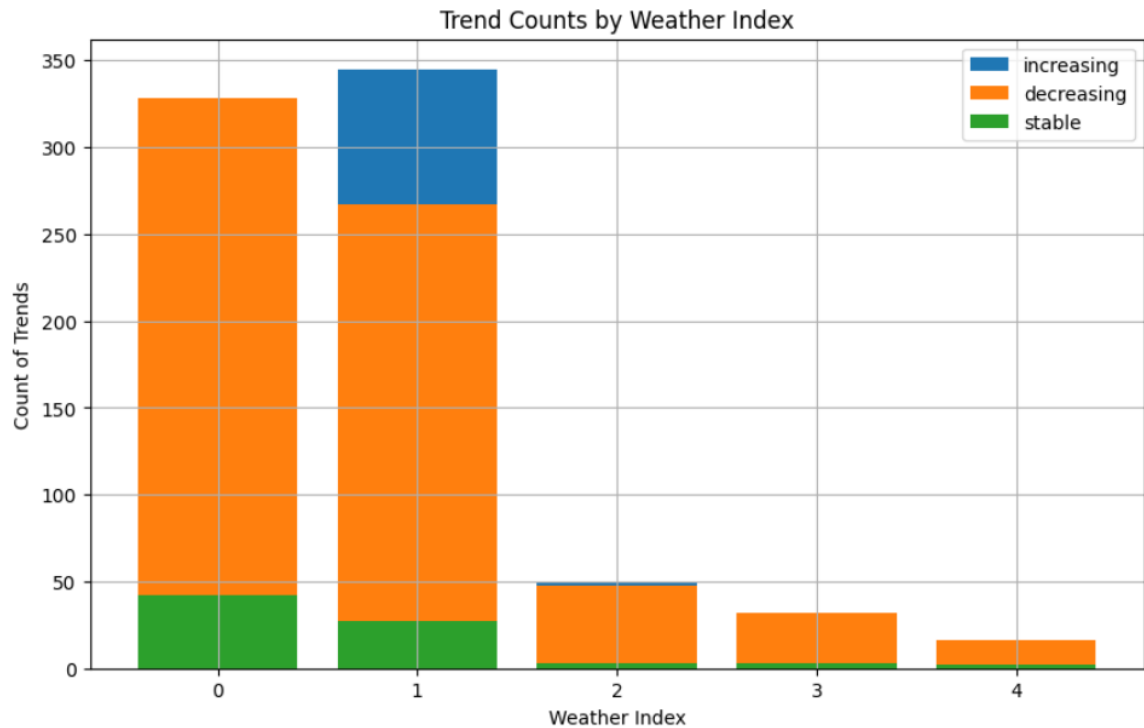
Giá trị trend count theo từng weather-index, nhận thấy:

- Với index từ 0.0 đến 4.0 lần lượt là rain, sun, snow, drizzle, fog
- Hai thời tiết phổ biến nhất là rain và sun
- Trong các thời tiết rain, drizzle và fog, nhiệt độ có xu hướng giảm nhiều hơn khi decreasing > increasing
- Tương tự với các thời tiết còn lại thì ngược lại

Trend Counts by Weather Index:

```
Weather Index 0.0: {'increasing': 270, 'decreasing': 328, 'stable': 42}
Weather Index 1.0: {'increasing': 345, 'decreasing': 267, 'stable': 27}
Weather Index 2.0: {'increasing': 50, 'decreasing': 48, 'stable': 3}
Weather Index 3.0: {'increasing': 17, 'decreasing': 32, 'stable': 3}
Weather Index 4.0: {'increasing': 8, 'decreasing': 16, 'stable': 2}
```

Vẽ biểu đồ biểu diễn giá trị trên:



4.1.3. KNN

- Với độ chính xác trên tập test là 83,79% cho thấy mô hình KNN hoạt động khá tốt

```
print("Test Accuracy:", accuracy)
```

Test Accuracy: 0.8378995433789954

- Cho mô hình dự đoán thời tiết giả sử trong 10 ngày tiếp theo:

```
# Create 10 input rows for prediction
input_data = [
    [0.0, 10.0, 2.8, 2.0], # Example 1
    [10.9, 10.6, 2.8, 4.5], # Example 2
    [0.8, 11.7, 7.2, 2.3], # Example 3
    [20.3, 12.2, 5.6, 4.7], # Example 4
    [1.3, 8.9, 2.8, 6.1], # Example 5
    [2.5, 4.4, 2.2, 2.2], # Example 6
    [0.0, 7.2, 2.8, 2.3], # Example 7
    [4.3, 9.4, 5.0, 3.4], # Example 8
    [1.0, 6.1, 0.6, 3.4], # Example 9
    [0.0, 6.7, -2.2, 1.4] # Example 10
]

# Predict labels for input data
input_predictions = [knn_predict(row, k) for row in input_data]

# Display the predicted weather index for each input
for i, pred in enumerate(input_predictions, start=1):
    print(f"Input {i}: Predicted Weather Index = {pred}")
```

- Kết quả:

```
Input 1: Predicted Weather Index = 1.0
Input 2: Predicted Weather Index = 0.0
Input 3: Predicted Weather Index = 0.0
Input 4: Predicted Weather Index = 0.0
Input 5: Predicted Weather Index = 0.0
Input 6: Predicted Weather Index = 0.0
Input 7: Predicted Weather Index = 1.0
Input 8: Predicted Weather Index = 0.0
Input 9: Predicted Weather Index = 0.0
Input 10: Predicted Weather Index = 1.0
```

4.2. So sánh, đánh giá

4.2.1. K-Means

❖ Đánh giá bằng hệ số Silhouette

Nhóm lần lượt dùng lớp SilhouetteEvaluator tự cài đặt và lớp ClusteringEvaluator

được cung cấp bởi thư viện pyspark.ml để đánh giá kết quả phân cụm của lớp K_Mean nhóm đã cài đặt

```
[ ] # Dùng hệ số Silhouette đã cài đặt để đánh giá việc phân cụm df_zscore của mô hình

evaluator = SilhouetteEvaluator()
silhouette_score = evaluator.evaluate(df_result)
print('Silhouette score: ', silhouette_score)

Silhouette score: 0.5500840143533829

# Dùng thư viện để tính hệ số Silhouette

from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.ml.feature import VectorAssembler

feature_cols = ['precipitation_nor', 'temp_max_nor', 'temp_min_nor', 'wind_nor']
vec_assembler = VectorAssembler(inputCols=feature_cols, outputCol='features_vec')
df_result2 = vec_assembler.transform(df_result)
evaluator = ClusteringEvaluator(featuresCol='features_vec')
silhouette = evaluator.evaluate(df_result2)
print("Silhouette = " + str(silhouette))

Silhouette = 0.5500840143533835
```

Hình trên cho thấy kết quả Silhouette score từ lớp SilhouetteEvaluator() (nhóm tự cài đặt) và lớp ClusteringEvaluator (của thư viện pyspark.ml) xấp xỉ nhau và >0.5 , nghĩa là việc phân cụm dữ liệu của lớp K_Mean là “hợp lý”.

❖ So sánh việc phân cụm với các lớp có sẵn trong thư viện pyspark.ml

Import các lớp thuật toán K-Means và đánh giá được cung cấp bởi thư viện pyspark.ml

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import ClusteringEvaluator
```

- Chọn số cụm **K** tối ưu để phân cụm

Tương tự nhóm cũng chọn 2 phương pháp là *Elbow method* và *Silhouette score*, nhưng lần này là dùng thư viện có sẵn, cụ thể là dùng lớp KMeans để thực hiện thuật toán K-Means và dùng lớp Clustering Evaluator để đánh giá việc phân cụm của lớp KMeans. Cả hai lớp này đều thuộc thư viện pyspark.ml

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import ClusteringEvaluator
import matplotlib.pyplot as plt
import numpy as np

assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
df_train = assembler.transform(df_zscore)

fig, axs = plt.subplots(1, 2, figsize=(15, 6))

wcss_list = []
silhouette_list = []

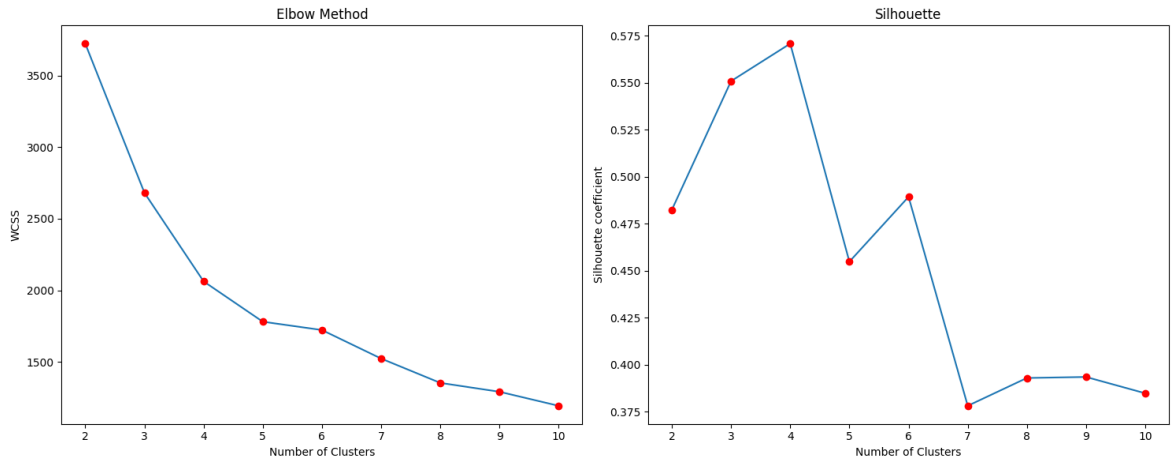
for k in range(2,11):
    kmeans = KMeans(k=k, seed=1) # thư viện pyspark.ml
    model = kmeans.fit(df_train)
    df_result = model.transform(df_train)
    evaluator = ClusteringEvaluator()
    silhouette_score = evaluator.evaluate(df_result)
    wcss_list.append(model.summary.trainingCost)
    silhouette_list.append(silhouette_score)

# Elbow Method
axs[0].plot(np.arange(2, 11), wcss_list, '-')
axs[0].plot(np.arange(2,11), wcss_list, 'o', color='red')
axs[0].set_xlabel('Number of Clusters')
axs[0].set_ylabel('WCSS')
axs[0].set_title('Elbow Method')

# Silhouette
axs[1].plot(np.arange(2, 11), silhouette_list, '-')
axs[1].plot(np.arange(2,11), silhouette_list, 'o', color='red')
axs[1].set_xlabel('Number of Clusters')
axs[1].set_ylabel('Silhouette coefficient')
axs[1].set_title('Silhouette')

plt.tight_layout()
plt.show()
```

Kết quả:



Theo như biểu đồ thì số cụm k tốt nhất vẫn là 4, $k = 4$

- Thực hiện training mô hình

```
# Thực hiện training mô hình trên dữ liệu df_zscore với k = 4

kmeans = KMeans(k=4, seed=1)
kmeans = kmeans.setFeaturesCol("features")
model = kmeans.fit(df_train)
```

Sau khi training mô hình, có có kết quả các centroids và độ đo wcss

```
# Giá trị các centroids và wcss sau khi train

from pyspark.sql.types import *

# Chuyển đổi các nhãn cluster của các centers tạo từ thư viện pyspark.ml tương đồng với các centroids
# của lớp K-Mean nhóm cài đặt bằng cách phân cụm các centers này bằng K-Mean nhóm cài đặt
cols = ['col1', 'col2', 'col3', 'col4']
centers = model.clusterCenters()
centers = [tuple(float(x) for x in centers[i]) + (i,) for i in range(len(centers))]
df_mllib = spark.createDataFrame(centers, [*cols, 'index'])

df_mllib = kmean.transform(df_mllib, cols)
cluster_map = df_mllib.orderBy('index').rdd.map(lambda r: r['prediction']).collect()

centers = df_mllib.orderBy('prediction') \
    .rdd.map(lambda r: [r[col] for col in cols]) \
    .collect()

print('centroids:')
for centroid in centers:
    print(centroid)
print('\nWCSS:', model.summary.trainingCost)
```

```
centroids:
[0.06663644105947496, -0.6170291543343396, -0.4669751564176076, 1.3274871575070826]
[-0.21022139784876032, -0.8142273827973082, -0.919494995776482, -0.6503450406889882]
[3.0680331905157807, -0.4334865927955932, -0.017892229947060856, 0.8897271245153653]
[-0.36425443624203485, 0.9447760439713934, 0.8836107360037216, -0.30345350726656023]

WCSS: 2061.961114837525
```

Trong đoạn code trên nhóm đã thực hiện chuyển đổi nhãn cụm sao cho các centroid của lớp K_Mean (nhóm tự cài đặt) và của lớp KMeans (thư viện pyspark.ml) gần nhau nhất thì có cùng nhãn cụm, điều này giúp dễ dàng quan sát và so sánh kết quả phân cụm của 2 lớp. Ý tưởng thực hiện là phân cụm các centroid của lớp KMeans bằng lớp K_Mean.

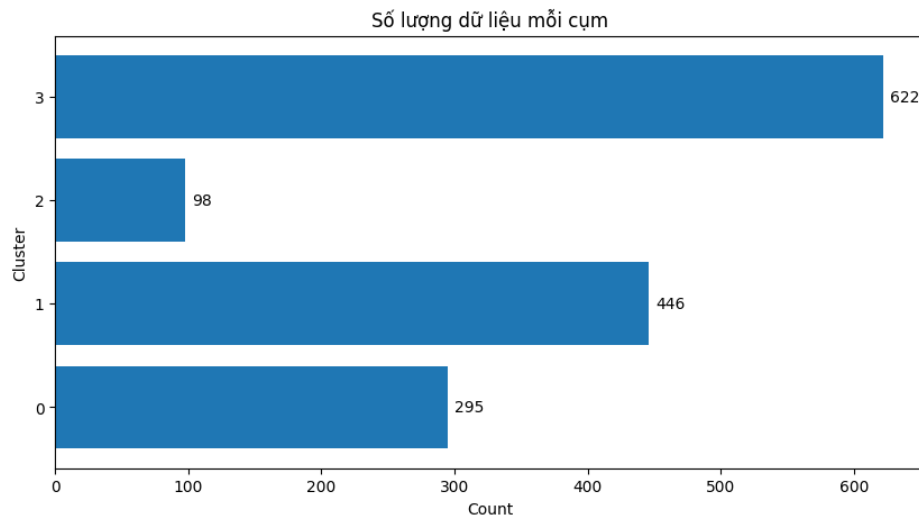
Kết quả phân cụm của lớp K_Mean (nhóm tự cài đặt) trước đó:

```
centroids:
[-0.05115713282887737, -0.5144709004093946, -0.4031301744140794, 1.2023089562248654]
[-0.18888445099916307, -0.6387667848666028, -0.6878702451702778, -0.6269401970426495]
[2.803215257816656, -0.35461786806223405, 0.06492894973362687, 1.08734316238199]
[-0.3920632133753182, 1.1487554752849358, 1.0333538082707596, -0.3512503971864999]

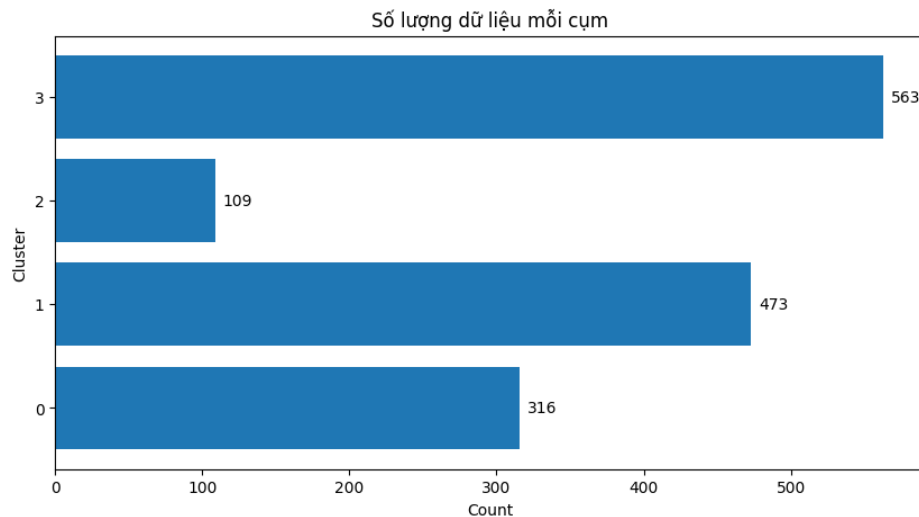
WCSS: 2224.8662391929133
```

- **Nhận xét:** Độ đo WCSS của nhóm lớn hơn (thấp hơn là tốt hơn) của thư viện ($2224.8662 > 2061.9611$).
- Thống kê số lượng dữ liệu ở mỗi cụm

Lớp KMeans của thư viện:

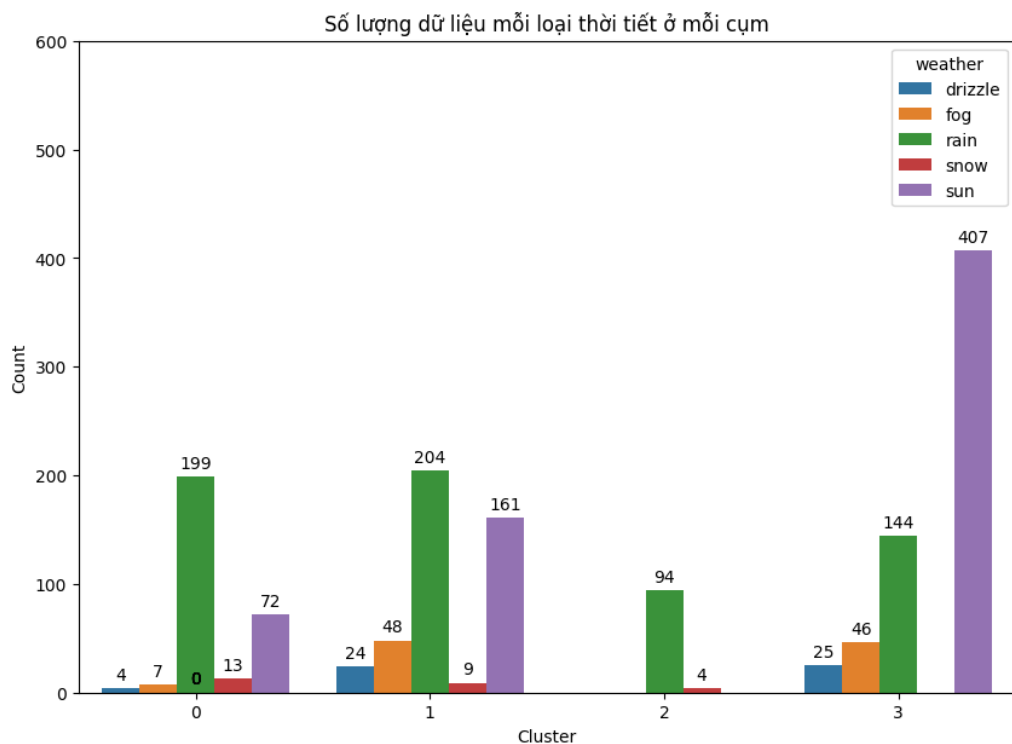


Lớp K_Mean nhóm tự cài đặt:

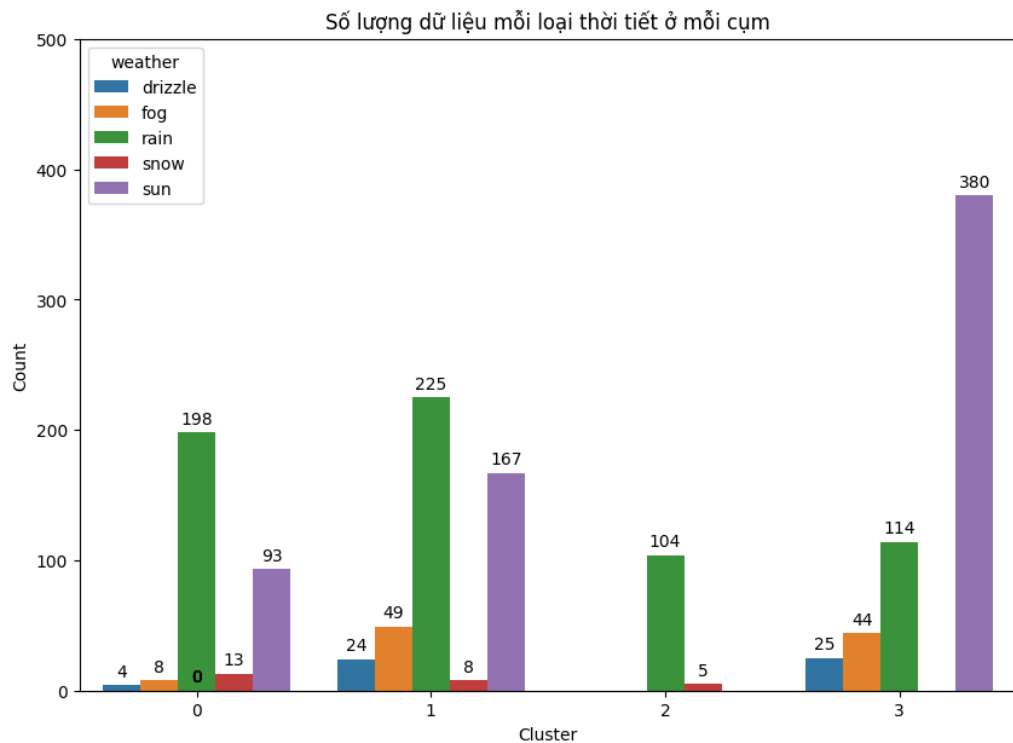


- **Nhận xét:** Thứ tự về số lượng dữ liệu ở mỗi cụm không thay đổi. Số lượng dữ liệu phân vào cụm 3 là nhiều nhất rồi đến cụm 1, cụm 0 và cụm 2 là ít nhất.
- Thống kê số lượng dữ liệu các loại thời tiết ở mỗi cụm

Lớp KMeans của thư viện:



Lớp K_Mean nhóm tự cài đặt:



- **Nhận xét:** Dữ liệu phân cụm khá tương đồng nhau ở cả hai cách cài đặt thuật toán K-Means
- Hệ số Silhouette của kết quả phân cụm

Lớp KMeans của thư viện:

```
[20] # Hệ số Silhouette
      from pyspark.ml.evaluation import ClusteringEvaluator

      evaluator = ClusteringEvaluator()
      silhouette_score = evaluator.evaluate(df_result)
      print('Silhouette score: ', silhouette_score)

      Silhouette score: 0.5707688798107416
```

Lớp K_Mean nhóm tự cài đặt:

```
[ ] # Dùng hệ số Silhouette đã cài đặt để đánh giá việc phân cụm df_zscore của mô hình

      evaluator = SilhouetteEvaluator()
      silhouette_score = evaluator.evaluate(df_result)
      print('Silhouette score: ', silhouette_score)

      Silhouette score: 0.5500840143533829
```

- **Nhận xét:** Silhouette score của kết quả phân cụm ở cả hai cách cài đặt xấp xỉ nhau, đều >0.5 nhưng của thư viện có phần tốt hơn.
- Tốc độ thực thi
- **Nhận xét:** Do chưa được tối ưu tốt nên các lớp nhóm tự cài đặt là K_Mean, SilhouetteEvaluator mất nhiều thời gian hơn để chạy so với các lớp được cung cấp bởi thư viện pyspark.ml.

4.2.2. Simple Moving Average (SMA)

Đánh giá thuật toán bằng MSE, MAE, RMSE

```
import numpy as np
import matplotlib.pyplot as plt

def calculate_metrics(actual_values, predicted_values):
    # Removing None values for comparison
    valid_indices = [i for i in range(len(predicted_values)) if predicted_values[i] is not None]
    actual_values = np.array([actual_values[i] for i in valid_indices])
    predicted_values = np.array([predicted_values[i] for i in valid_indices])

    mse = np.mean((actual_values - predicted_values) ** 2)
    mae = np.mean(np.abs(actual_values - predicted_values))
    rmse = np.sqrt(mse)

    return mse, mae, rmse
```

- Import các thư viện cần thiết
- **def calculate_metrics(actual_values, predicted_values):**: Định nghĩa hàm **calculate_metrics** nhận vào hai danh sách giá trị: **actual_values** (giá trị thực tế) và **predicted_values** (giá trị dự đoán) để tính các thông số đánh giá và hiển thị ra màn hình
 - **valid_indices = [i for i in range(len(predicted_values)) if predicted_values[i] is not None]**: Tạo một danh sách các chỉ số hợp lệ bằng cách loại bỏ các giá trị **None** trong **predicted_values**
 - **actual_values = np.array([actual_values[i] for i in valid_indices])**: Tạo một mảng NumPy từ các giá trị thực tế tại các chỉ số hợp lệ
 - **predicted_values = np.array([predicted_values[i] for i in valid_indices])**: Tạo một mảng NumPy từ các giá trị dự đoán tại các

chỉ số hợp lệ

- **mse = np.mean((actual_values - predicted_values) ** 2)**: Tính mean squared error (MSE) bằng cách lấy trung bình của bình phương của sự sai lệch giữa **actual_values** và **predicted_values**
- **mae = np.mean(np.abs(actual_values - predicted_values))**: Tính mean absolute error (MAE) bằng cách lấy trung bình của giá trị tuyệt đối của sự sai lệch giữa **actual_values** và **predicted_values**
- **rmse = np.sqrt(mse)**: Tính square root of mean squared error (RMSE) bằng cách lấy căn bậc hai của MSE

```
# Sample data (assuming already collected as in the provided code)
# rdd = transformed_data.select("date", "features", "weather_index").rdd
data = rdd.map(lambda row: (row['date'], row['features'][1], row['weather_index'], row['features'][0])).collect()
dates = [row[0] for row in data]
temp_max_values = [row[1] for row in data]
weather_indices = [row[2] for row in data]
```

- Tính lại các giá trị để sử dụng từ rdd

```
# Calculate SMA for different window sizes
window_sizes = [3, 7, 20, 50, 200]
sma_results = {}
metrics_results = {}

for window_size in window_sizes:
    sma_values = calculate_sma(temp_max_values, window_size)
    sma_results[window_size] = sma_values
    mse, mae, rmse = calculate_metrics(temp_max_values, sma_values)
    metrics_results[window_size] = {'MSE': mse, 'MAE': mae, 'RMSE': rmse}

# Print the results
for window_size, metrics in metrics_results.items():
    print(f"Window Size: {window_size}")
    print(f"MSE: {metrics['MSE']}")
    print(f"MAE: {metrics['MAE']}")
    print(f"RMSE: {metrics['RMSE']}\n")
```

- **window_sizes = [3, 7, 20, 50, 200]**: Tạo một danh sách các kích thước cửa sổ SMA khác nhau để tính toán.
- **sma_results = {}**: Khởi tạo một từ điển để lưu trữ kết quả của SMA với mỗi

kích thước cửa sổ.

- **metrics_results = {}:** Khởi tạo một từ điển để lưu trữ các độ đo của mô hình (MSE, MAE, RMSE) với mỗi kích thước cửa sổ
- Vòng lặp **for window_size in window_sizes::**
 - Tính toán SMA và các độ đo chất lượng tương ứng cho mỗi kích thước cửa sổ:
 - **sma_values = calculate_sma(temp_max_values, window_size):** Tính toán SMA cho danh sách **temp_max_values** với kích thước cửa sổ là **window_size**.
 - **sma_results[window_size] = sma_values:** Lưu trữ kết quả SMA trong từ điển **sma_results**.
 - **mse, mae, rmse = calculate_metrics(temp_max_values, sma_values):** Tính toán các độ đo chất lượng (MSE, MAE, RMSE) cho SMA tính được.
 - **metrics_results[window_size] = {'MSE': mse, 'MAE': mae, 'RMSE': rmse}:** Lưu trữ các độ đo trong **metrics_results**
- Cuối cùng chạy vòng lặp và in ra các giá trị

Kết quả sau khi chạy lệnh đánh giá:

↔ Window Size: 3
MSE: 4.337439646637727
MAE: 1.6236234864062145
RMSE: 2.0826520704711404

Window Size: 7
MSE: 8.911285503892277
MAE: 2.3425233186057928
RMSE: 2.9851776335575537

Window Size: 20
MSE: 12.028621203190015
MAE: 2.730142163661581
RMSE: 3.4682302696317637

Window Size: 50
MSE: 19.24149633427762
MAE: 3.465101983002833
RMSE: 4.386513004001882

Window Size: 200
MSE: 74.86849875059433
MAE: 7.407109746434234
RMSE: 8.652658478791032

Nhận xét: Dựa trên kết quả tính toán cho thuật toán SMA với các kích thước cửa sổ khác nhau, ta có thể kết luận như sau:

MSE, MAE và RMSE tăng đáng kể khi kích thước cửa sổ tăng. Điều này cho thấy rằng hiệu suất dự đoán của thuật toán giảm đi khi ta sử dụng các cửa sổ lớn hơn. Ví dụ kích thước cửa sổ 20 cho thấy một sự cân bằng tốt giữa độ chính xác và tính ổn định của thuật toán. Chẳng hạn, với kích thước cửa sổ này, ta nhận được các giá trị độ đo:

- MSE: 12.03
- MAE: 2.73
- RMSE: 3.47

Điều này có nghĩa là trung bình bình phương sai số là khoảng 12.03, sai số trung bình tuyệt đối là khoảng 2.73 và sai số trung bình của các dự đoán so với giá trị thực tế là khoảng 3.47.

So với các kích thước cửa sổ khác, chẳng hạn như kích thước cửa sổ 200, ta thấy một tăng đáng kể trong các độ đo:

- MSE: 74.87
- MAE: 7.41
- RMSE: 8.65

Nhận thấy, RMSE tăng nhanh hơn so với MSE và MAE khi kích thước của số tăng. Điều này cho thấy sự ảnh hưởng của các giá trị dữ liệu cách xa giữa dự đoán và thực tế. Trong một số trường hợp, RMSE có thể tăng gấp đôi hoặc gấp ba khi kích thước của số tăng từ 20 lên 200.

Các giá trị này biểu thị rằng độ chính xác giảm đi đáng kể khi sử dụng các kích thước của số lớn hơn, đồng thời cũng tăng đáng kể sự biến động giữa dự đoán và giá trị thực tế. Do đó, việc lựa chọn kích thước của số phù hợp là rất quan trọng và nên dựa trên một sự cân nhắc kỹ lưỡng giữa độ chính xác và tính ổn định của thuật toán trong bối cảnh cụ thể của bài toán. Trong trường hợp này, kích thước của số 20 có thể là một lựa chọn phù hợp, vì nó cung cấp một sự cân bằng tốt giữa độ chính xác và tính ổn định, với RMSE tương đối thấp so với các kích thước của số khác.

4.2.3. KNN

- Sử dụng các công thức độ đo chính xác để tính các thông số: precision, f1-score, recall, và vẽ ma trận nhầm lẫn:

```
import seaborn as sns
from collections import defaultdict
# Calculate confusion matrix
confusion_matrix = defaultdict(lambda: defaultdict(int))

for true_label, prediction in zip(test_labels, test_predictions):
    confusion_matrix[true_label][prediction] += 1
```

```
# Calculate precision, recall, and F1 score for each class
metrics = {}
for label in confusion_matrix.keys():
    tp = confusion_matrix[label][label]
    fp = sum(confusion_matrix[row][label] for row in confusion_matrix.keys() if row != label)
    fn = sum(confusion_matrix[label][col] for col in confusion_matrix[label].keys() if col != label)

    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    f1 = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0

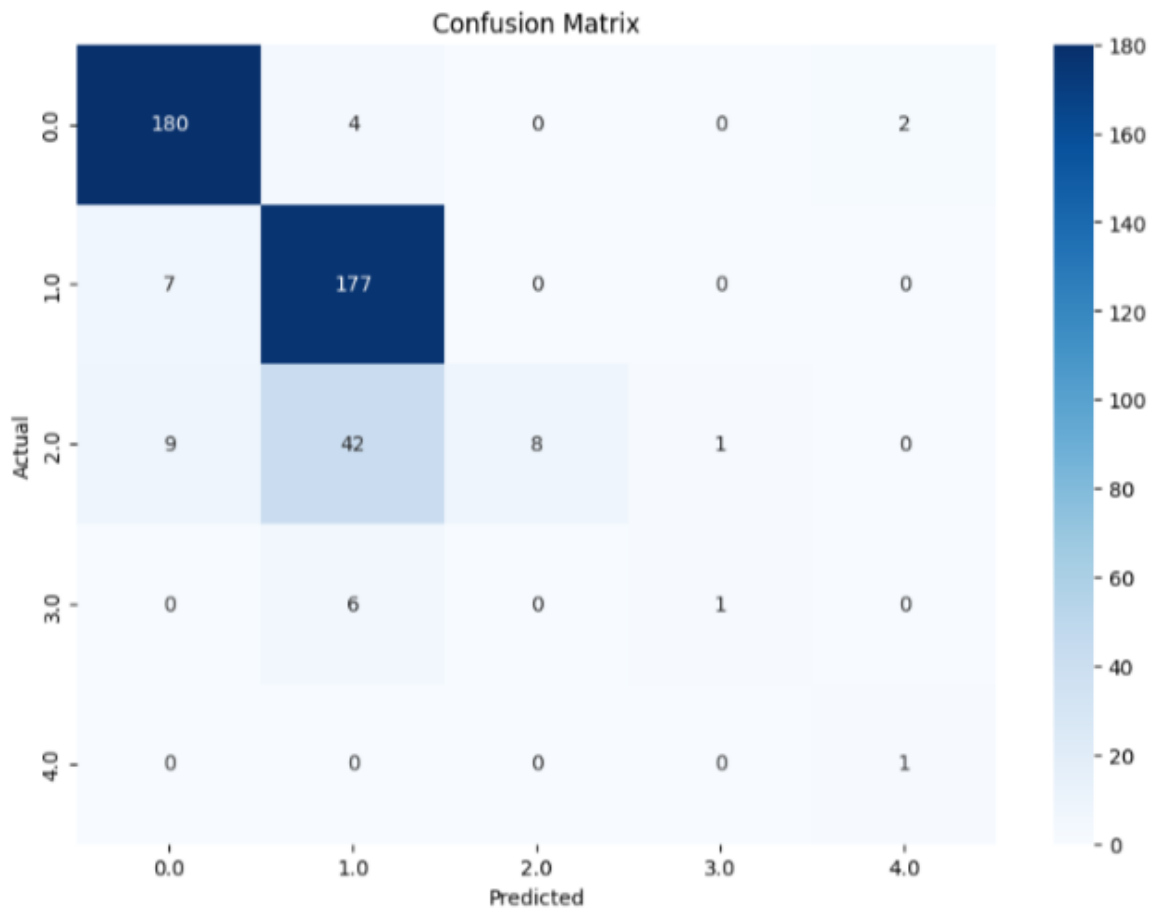
    metrics[label] = {'precision': precision, 'recall': recall, 'f1': f1}

# Print metrics for each class
for label, metric in metrics.items():
    print(f"Class {label}: Precision = {metric['precision']}, Recall = {metric['recall']}, F1 Score = {metric['f1']}")
```

```
# Plot confusion matrix
confusion_matrix_list = [[confusion_matrix[row][col] for col in sorted(confusion_matrix.keys())] for row in sorted(confusion_matrix.keys())]
plt.figure(figsize=(10, 7))
sns.heatmap(confusion_matrix_list, annot=True, fmt="d", cmap="Blues", xticklabels=sorted(confusion_matrix.keys()), yticklabels=sorted(confusion_matrix.keys()))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

- Kết quả:

```
Class 1.0: Precision = 0.7729257641921398, Recall = 0.9619565217391305, F1 Score = 0.8571428571428571
Class 0.0: Precision = 0.9183673469387755, Recall = 0.967741935483871, F1 Score = 0.9424083769633509
Class 2.0: Precision = 1.0, Recall = 0.1333333333333333, F1 Score = 0.23529411764705882
Class 3.0: Precision = 0.5, Recall = 0.14285714285714285, F1 Score = 0.22222222222222224
Class 4.0: Precision = 0.3333333333333333, Recall = 1.0, F1 Score = 0.5
```



CHƯƠNG 5: KẾT LUẬN

5.1. Ưu điểm

- Nhận diện được bài toán khai thác dữ liệu lớn và áp dụng được các giải thuật khai thác dữ liệu song song, phân tán.
- Ứng dụng công nghệ học máy hiện đại: Đề tài sử dụng các thuật toán phân cụm tiên tiến như K-means và SMA, KNN, đại diện cho ứng dụng thực tiễn của học máy trong lĩnh vực dự báo thời tiết.
- Phân tích sâu về dự báo thời tiết: Đề tài cung cấp cái nhìn sâu sắc về phân tích, đánh giá và dự đoán thời tiết theo ngày, có tính ứng dụng cao

5.2. Hạn chế

- Bộ dữ liệu được thu thập từ năm 2012 đến 2015, nên cũng không theo kịp xu thế thời tiết hiện nay, nhất là trong thời kỳ biến đổi khí hậu
- Nhóm sử dụng 3 thuật toán phân cụm K-means và KNN, SMA để phân tích dữ liệu, nó phụ thuộc nhiều vào việc lựa chọn tham số và cách xử lý dữ liệu đầu vào, nên còn một số hạn chế về việc tối ưu hóa thuật toán như K-means yêu cầu số lượng cụm cần được xác định trước gây nên khó khăn trong việc thực hiện thuật toán; Cả 3 thuật toán đều có thể không chính xác nếu dữ liệu có nhiều nhiễu hoặc nếu các cụm không rõ ràng.

5.3. Hướng phát triển

- Thực hiện thu thập và tích hợp dữ liệu thời tiết mới từ nhiều nguồn khác nhau với thông tin đa dạng, phong phú và đầy đủ hơn và tiến hành cập nhật bộ dữ liệu bộ dữ liệu mới.
- Nghiên cứu tối ưu hóa thuật toán K-means SMA và KNN để đưa ra kết quả chính xác hơn.
- Nghiên cứu và phát triển các phương pháp để xử lý dữ liệu real-time, giúp phân tích và đưa ra quyết định dựa trên thông tin cập nhật liên tục, từ đó tăng cường khả năng ứng dụng vào thực tiễn.

- Nghiên cứu và thử nghiệm với nhiều phương pháp phân cụm khác để cải thiện độ chính xác.

PHÂN CÔNG CÔNG VIỆC

	21520530 (Nhóm trưởng)	21520190	21522229	21522714
Lựa chọn, thống nhất đề tài	x	x	x	x
Lý do chọn đề tài, giới thiệu dataset	x			
Mô tả dữ liệu, bài toán	x			
Thực quan hóa dữ liệu	x	x		
Kỹ thuật tiền xử lý dữ liệu	x	x		x
Tiền xử lý dữ liệu		x		x
Truy vấn dữ liệu		x		
Triển khai thuật toán K-Means				x
Triển khai thuật toán SMA			x	
Triển khai thuật toán KNN	x			
Phát biểu kết quả	x	x	x	x
So sánh, đánh giá	x	x	x	x
Kết luận	x	x	x	x
Bản tóm tắt nội dung đề tài				
Làm báo cáo	x	x	x	x
Làm slide		x		
Chỉnh sửa format các file báo cáo	x	x	x	x

TÀI LIỆU THAM KHẢO

STT	Link tài liệu
1	https://machinelearningcoban.com/2017/01/01/kmeans/
2	https://www.youtube.com/watch?v=Q5n5yy8dA7A
3	https://www.youtube.com/watch?v=5w5iUbTlpMQ&t=172s
4	https://www.linkedin.com/advice/0/how-can-you-calculate-silhouette-score-clustering-algorithm-w9bcc#:~:text=The%20silhouette%20score%20for%20the,points%20in%20the%20data%20set.
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	