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

KHOA HỆ THỐNG THÔNG TIN

BÁO CÁO ĐỒ ÁN

LỚP: IS405.P11.HTCL

ỨNG DỤNG DỮ LIỆU LỚN TRONG DỰ ĐOÁN MẮC BỆNH TIM

Giáo viên hướng dẫn: Ths. Nguyễn Hồ Duy Tri

Nguyễn Thế Vinh – 21522794

Nguyễn Bình Nguyên - 21522391

Bùi Văn Thái - 21522577

Chế Duy Khang – 21522187

THÀNH PHỐ HỒ CHÍ MINH, 2024

MỤC LỤC

1. LÝ DO CHỌN ĐỀ TÀI

Bệnh tim mạch là một trong những nguyên nhân hàng đầu gây tử vong trên toàn cầu, đặc biệt ở các nước đang phát triển. Việc phát hiện sớm nguy cơ bệnh tim mạch có thể giúp giảm thiểu hậu quả và đồng thời nâng cao hiệu quả điều trị.

Trong kỷ nguyên dữ liệu lớn hiện nay, Big Data đem lại khả năng phân tích sâu rộng về dữ liệu y tế, đặc biệt là đề xuất nguy cơ dài hạn dựa trên nhiều thuộc tính khác nhau của bệnh nhân.

Do đó, đề tài này nhấn mạnh vào việc sử dụng các kỹ thuật phân tích dữ liệu lớn để xác định nguy cơ mắc bệnh tim mạch, nhằm hỗ trợ quyết định y khoa chính xác và nhanh chóng.

2. DỮ LIỆU

Dữ liệu được sử dụng trong bài phân tích được lấy từ: https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset/data.

Nguồn dữ liệu có 14 thuộc tính với 1025 dòng.

Tập dữ liệu này, xuất phát từ năm 1988, được xây dựng từ bốn cơ sở dữ liệu: Cleveland, Hungary, Thụy Sĩ và Long Beach V. Mặc dù bao gồm 76 thuộc tính, chỉ có 14 thuộc tính được sử dụng phổ biến trong các nghiên cứu và thử nghiệm công bố. Trường 'mục tiêu' thể hiện tình trạng mắc bệnh tim của bệnh nhân, với giá trị 0 biểu thị không mắc bệnh và 1 biểu thị có mắc bệnh."

Bao gồm các thuộc tính chính:

- Tuổi tác (age): Độ tuổi bệnh nhân.
- Giới tính (sex): Nam hoặc nữ.
- Loại đau ngực (cp): Phân loại theo các dạng đau ngực khác nhau. (4 giá trị)
- Huyết áp (trestbps): Huyết áp khi nghỉ ngơi.
- Mỡ trong máu (chol): Nồng độ mỡ trong máu (đơn vị mg/dl)
- Đường huyết (fbs): Chỉ số đường trong máu lúc đói > 120 mg/dl
- Kết quả điện tâm đồ (restecg): Điện tim đồ khi nghỉ (các giá trị: 0, 1, 2)
- Nhịp tim tối đa (thalach)

- Đau thắt ngực do gắng sức (exang)
- ST depression khi gắng sức (oldpeak): Độ giảm ST do vận động so với lúc nghỉ
- Độ dốc đoạn ST (slope): Độ dốc đoạn ST ở đỉnh gắng sức
- Số lượng mạch máu chính (ca): Số mạch máu chính (0-3) được đánh dấu bằng X-quang huỳnh quang.
- Thalassemia (thal): Trạng thái thalassemia (0 = bình thường; 1 = tổn thương cố định; 2 = tổn thương có thể hồi phục).
- Nguy cơ mắc bệnh (target): Nhãn dữ liệu (đã mắc bệnh hay chưa).

Giá trị thấp nhất, cao nhất, trung bình, phương sai, độ lệch chuẩn, trung vị của cột trestbps (Huyết áp khi nghỉ ngơi)

min_bp	max_bp	mean_bp	variance_bp	stddev_bp	median_bp
94	200	131.60264900662253	308.4728168797168	17.563394230037563	130

Giá trị thấp nhất, cao nhất, trung bình, phương sai, độ lệch chuẩn, trung vị của cột chol (Nồng độ mỡ trong máu)

[Stage 97:> (0 + 1) / 1]

min_cholestrol	max_cholestrol	mean_cholestrol	variance_cholestrol	stddev_cholestrol	median_cholestrol
126	564	246.5	2678.423588039868	51.75348865574057	240

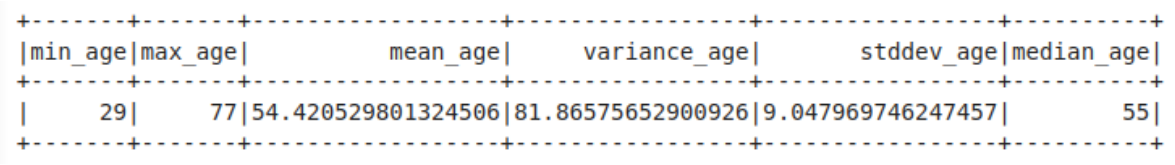
Giá trị thấp nhất, cao nhất, trung bình, phương sai, độ lệch chuẩn, trung vị của cột thalach (Nhịp tim tối đa)

min_max_heart_rate	max_max_heart_rate	mean_max_heart_rate	variance_max_heart_rate	stddev_max_heart_rate	median_max_heart_rate
71	202	149.56953642384107	524.5715605817253	22.903527251969845	152

Giá trị thấp nhất, cao nhất, trung bình, phương sai, độ lệch chuẩn, trung vị của cột oldpeak (Độ giảm ST do vận động so với lúc nghỉ)

min_old_peak	max_old_peak	mean_old_peak	variance_old_peak	stddev_old_peak	median_old_peak
0.0	6.2	1.0430463576158941	1.3489714197707428	1.1614522890634564	0.8

Giá trị thấp nhất, cao nhất, trung bình, phương sai, độ lệch chuẩn, trung vị của cột age (Tuổi)



min_age	max_age	mean_age	variance_age	stddev_age	median_age
29	77	54.420529801324506	81.86575652900926	9.047969746247457	55

3. MÔ TẢ BÀI TOÁN

Cho một tập dữ liệu y tế bao gồm các thông tin nhân khẩu học (tuổi, giới tính) và dấu hiệu sinh học (huyết áp, cholesterol, nhịp tim, v.v.) của các bệnh nhân, bài toán yêu cầu:

- Xác định đầu vào:
    - Một tập dữ liệu chứa các thông tin của bệnh nhân dưới dạng bảng dữ liệu.
    - Biến mục tiêu là nhãn phân loại target, trong đó 1 biểu thị bệnh nhân có nguy cơ mắc bệnh tim và 0 biểu thị không có nguy cơ.
- Xác định đầu ra: Một mô hình dự đoán dựa trên các thuật toán học máy (Machine Learning) có khả năng phân loại chính xác bệnh nhân có nguy cơ mắc bệnh tim dựa trên dữ liệu đầu vào.
- Các ràng buộc và yêu cầu:
    - Dữ liệu phải được xử lý và làm sạch trước khi xây dựng mô hình (ví dụ: xử lý giá trị thiếu, mã hóa biến, chuẩn hóa đặc trưng).
    - Mô hình phải được đánh giá thông qua các chỉ số như accuracy, precision, recall, và F1-score để đảm bảo hiệu quả.

Mục tiêu bài toán:

- Dự đoán chính xác nguy cơ mắc bệnh tim dựa trên thông tin y tế.
- Phân tích và phát hiện các yếu tố quan trọng ảnh hưởng đến bệnh tim (như huyết áp, nhịp tim).
- Tối ưu hóa các phương pháp giải thuật để đạt độ chính xác cao và áp dụng hiệu quả vào khai thác dữ liệu y tế.

4. KỸ THUẬT TIỀN XỬ LÝ DỮ LIỆU

Các kỹ thuật tiền xử lý này đều nhằm đảm bảo chất lượng dữ liệu trước khi đưa vào mô hình. Dữ liệu sạch, nhất quán, và không dư thừa sẽ giúp mô hình học tốt hơn, đưa ra dự đoán chính xác và hiệu quả hơn.

4.1. Kiểm tra các giá trị NULL

Xác định các cột hoặc hàng có giá trị NULL hoặc NaN trong dữ liệu.

Kiểm tra các giá trị NULL

```
[5]: # Đếm giá trị null trong từng cột
null_counts = data.select([F.sum(F.when(F.col(c).isNull(), 1).otherwise(0)).alias(c) for c in data.columns])
null_counts.show()
```

```
+---+---+---+--------+----+---+-------+-------+-----+-------+-----+---+----+------+
|age|sex| cp|trestbps|chol|fbs|restecg|thalach|exang|oldpeak|slope| ca|thal|target|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----+---+----+------+
|  0|  0|  0|       0|   0|  0|      0|      0|    0|      0|    0|  0|   0|     0|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----+---+----+------+
```

4.1.1. Lý do lựa chọn:

- Ảnh hưởng đến quá trình tính toán và mô hình hóa:
    - Các giá trị NULL/NaN gây lỗi trong quá trình thực hiện các phép tính hoặc đào tạo mô hình học máy, đặc biệt với các thuật toán không hỗ trợ dữ liệu bị thiếu.
- Bảo toàn thông tin quan trọng:
    - Phát hiện giá trị bị thiếu giúp đưa ra quyết định phù hợp:
        - Xóa hàng/cột: Nếu giá trị bị thiếu quá nhiều, xóa chúng để tránh nhiễu.
        - Điền giá trị thay thế: Dùng giá trị trung bình, trung vị, hoặc mode để thay thế, nhằm bảo toàn thông tin trong dữ liệu.
- Đảm bảo dữ liệu sạch và đáng tin cậy:
    - Dữ liệu không sạch có thể dẫn đến kết quả không chính xác khi xây dựng mô hình hoặc phân tích.

4.1.2. Cách thức áp dụng:

- Sử dụng các hàm trong thư viện PySpark như isNull() hoặc isNotNull() để xác định các giá trị NULL trong từng cột.
- Dùng hàm filter() để lọc các bản ghi chứa giá trị NULL, hoặc dùng fillna() để điền giá trị thay thế.

Kết quả

Sau khi kiểm tra và xử lý giá trị NULL:

- Dữ liệu sạch hơn: Loại bỏ được các hàng bị lỗi hoặc điền giá trị thay thế hợp lý.
- Tỷ lệ NULL trong dữ liệu giảm xuống 0%.

4.2. Loại bỏ các dòng trùng lặp

Xóa các hàng trùng lặp

```
6]: data = data.dropDuplicates()
```

Phát hiện các hàng trong dữ liệu có giá trị giống nhau ở tất cả các cột.

4.2.1. Lý do lựa chọn

- Loại bỏ dữ liệu dư thừa:
    - Dữ liệu trùng lặp không mang thêm thông tin mới và có thể làm tăng kích thước dữ liệu một cách không cần thiết.
    - Ví dụ: Nếu dữ liệu có 10.000 bản ghi nhưng 1.000 bản ghi là trùng lặp, mô hình sẽ bị chậm và tốn tài nguyên hơn mà không cải thiện độ chính xác.
- Ngăn thiên lệch trong mô hình:
    - Dữ liệu trùng lặp có thể khiến mô hình học thiên về các mẫu lặp lại, dẫn đến giảm tính tổng quát khi dự đoán trên dữ liệu mới.
- Đảm bảo tính độc nhất của dữ liệu:

- Dữ liệu sạch, không trùng lặp giúp phản ánh đúng bản chất của vấn đề cần phân tích.

4.2.2. Cách thức áp dụng

- Sử dụng hàm dropDuplicates() của PySpark để loại bỏ các bản ghi trùng lặp.
- Xác định các dòng trùng lặp dựa trên toàn bộ cột hoặc một tập hợp cột cụ thể.

Kết quả:

- Số dòng trùng lặp được loại bỏ: Ví dụ: 100 dòng.
- Kích thước dữ liệu giảm: Từ 10.000 dòng xuống còn 9.900 dòng (giảm 1%).
- Dữ liệu sau khi xử lý trở nên gọn gàng và không dư thừa.

4.3. Đổi tên cột trong DataFrame PySpark

```
]: # Đổi tên các cột trong DataFrame PySpark
data_visual = data.withColumnRenamed('age', 'Age') \
            .withColumnRenamed('sex', 'Sex') \
            .withColumnRenamed('cp', 'Chest Pain') \
            .withColumnRenamed('trestbps', 'Resting_BP') \
            .withColumnRenamed('chol', 'Cholestrol') \
            .withColumnRenamed('fbs', 'Fasting_Blood_Sugar') \
            .withColumnRenamed('restecg', 'Resting_Electrocardiographic') \
            .withColumnRenamed('thalach', 'Max_Heart_Rate') \
            .withColumnRenamed('exang', 'Exercise_Induced_Angina') \
            .withColumnRenamed('oldpeak', 'Old_Peak') \
            .withColumnRenamed('slope', 'Slope') \
            .withColumnRenamed('ca', 'No_Major_Vessels') \
            .withColumnRenamed('thal', 'Thal') \
            .withColumnRenamed('target', 'Target')

# Kiểm tra tên các cột sau khi đổi
data_visual.columns
```

4.3.1. Lý do lựa chọn

- Cải thiện khả năng đọc hiểu dữ liệu: Các tên cột được đổi từ viết tắt hoặc tên kỹ thuật (như cp, trestbps, chol) thành tên đầy đủ và dễ hiểu hơn (như Chest Pain, Resting_BP, Cholestrol).
- Tăng tính trực quan: Dễ dàng nắm bắt ý nghĩa của từng cột.
- Hỗ trợ phân tích dữ liệu: Sử dụng các cột trong các bước xử lý tiếp theo (như trực quan hóa hoặc phân tích) trở nên thuận tiện và chính xác.

4.3.2. Cách thức áp dụng

Sử dụng hàm withColumnRenamed của PySpark:

- Hàm này cho phép đổi tên từng cột của DataFrame mà không làm thay đổi dữ liệu bên trong.
- Chuỗi lệnh liên tiếp (\) giúp thực hiện đổi tên nhiều cột một cách tuần tự và rõ ràng.

Kiểm tra sau khi đổi tên: Lệnh data_visual.columns sẽ hiển thị danh sách tên cột mới, xác nhận rằng việc đổi tên đã được thực hiện thành công.

Kết quả:

```
['Age',
 'Sex',
 'Chest Pain',
 'Resting_BP',
 'Cholestrol',
 'Fasting_Blood_Sugar',
 'Resting_Electrocardiographic',
 'Max_Heart_Rate',
 'Exercise_Induced_Angina',
 'Old_Peak',
 'Slope',
 'No_Major_Vessels',
 'Thal',
 'Target']
```

5. PHÂN TÍCH DỮ LIỆU

5.1. Tính tỷ lệ mắc bệnh tim theo huyết áp theo độ tuổi

```
# Tính tỷ lệ mắc bệnh tim theo huyết áp theo độ tuổi
data_visual.createOrReplaceTempView("heart_disease")
query = """
    SELECT
        CONCAT(CAST(Age_Group * 10 AS STRING), '-', CAST(Age_Group * 10 + 10 AS STRING)) AS Age_Range,
        Target,
        COUNT(*) AS count,
        ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(PARTITION BY Age_Group), 2) AS percentage
    FROM (
        SELECT
            FLOOR(Age / 10) AS Age_Group,
            Target
        FROM heart_disease
    ) sub
    GROUP BY Age_Group, Target
    ORDER BY Age_Group;
"""

result_age_target = spark.sql(query)
result_age_target.show()
```

```
+---------+------+-----+----------+
|Age_Range|Target|count|percentage|
+---------+------+-----+----------+
|    20-30|     1|    1|    100.00|
|    30-40|     1|   10|     71.43|
|    30-40|     0|    4|     28.57|
|    40-50|     1|   50|     69.44|
|    40-50|     0|   22|     30.56|
|    50-60|     0|   60|     48.00|
|    50-60|     1|   65|     52.00|
|    60-70|     0|   48|     60.00|
|    60-70|     1|   32|     40.00|
|    70-80|     0|    4|     40.00|
|    70-80|     1|    6|     60.00|
+---------+------+-----+----------+
```

Nhận Xét:

Xu hướng chung:

- Tỷ lệ mắc bệnh tim có xu hướng cao trong các nhóm tuổi trung niên (40-50 và 50-60 tuổi).
- Tỷ lệ mắc bệnh giảm nhẹ ở các nhóm tuổi lớn hơn như 60-70 tuổi, nhưng dữ liệu ở nhóm tuổi 70-80 lại tăng nhẹ.

Dữ liệu nhóm tuổi trẻ (20-30):

- Chỉ có 1 mẫu trong nhóm này nên tỷ lệ 100% mắc bệnh không thể phản ánh xu hướng chính xác.

Tính cân bằng giữa nhãn 0 và 1:

- Ở nhóm tuổi 50-60, tỷ lệ giữa người mắc và không mắc bệnh tim gần như cân bằng (48% và 52%).
- Trong các nhóm khác như 40-50 tuổi, tỷ lệ mắc bệnh cao hơn đáng kể (gần 70%).

Ảnh hưởng của tuổi tác:

- Tỷ lệ mắc bệnh tim cao nhất trong nhóm trung niên (40-50 tuổi). Điều này có thể do ảnh hưởng của lối sống, bệnh lý mãn tính hoặc các yếu tố khác tích lũy theo thời gian.

Kết Luận:

- Tuổi tác là một yếu tố quan trọng liên quan đến tỷ lệ mắc bệnh tim, với đỉnh điểm ở nhóm 40-50 tuổi.
- Ở nhóm 50-60 tuổi, tỷ lệ mắc bệnh giảm nhẹ và có sự cân bằng giữa mắc và không mắc bệnh.
- Dữ liệu trong nhóm 20-30 và 70-80 còn hạn chế, cần thu thập thêm để đưa ra kết luận chính xác.

5.2. Tính tỷ lệ mắc bệnh tim theo mức đường huyết

```
# Tính tỷ lệ mắc bệnh tim theo mức đường huyết
query = """
    SELECT
        Fasting_Blood_Sugar,
        Target,
        COUNT(*) AS count,
        ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(PARTITION BY Fasting_Blood_Sugar), 2) AS percentage
    FROM heart_disease
    GROUP BY Fasting_Blood_Sugar, Target
    ORDER BY Fasting_Blood_Sugar, Target
"""

result_fasting_blood_sugar_target = spark.sql(query)
result_fasting_blood_sugar_target.show()
```

Fasting_Blood_Sugar	Target	count	percentage
0	0	116	45.14
0	1	141	54.86
1	0	22	48.89
1	1	23	51.11

Nhận Xét:

Nhóm có FBS = 0 (đường huyết bình thường):

- Tỷ lệ mắc bệnh tim chiếm 54.86% cao hơn tỷ lệ không mắc bệnh (45.14%).
- Điều này cho thấy ngay cả khi mức đường huyết bình thường, nguy cơ mắc bệnh tim vẫn đáng kể.

Nhóm có FBS = 1 (đường huyết cao):

- Tỷ lệ mắc bệnh tim và không mắc bệnh trong nhóm này gần bằng nhau (51.11% và 48.89%).
- Sự cân bằng này có thể do kích thước mẫu nhỏ (tổng cộng 45 người), cần thu thập thêm dữ liệu để khẳng định mối liên hệ.

Tổng Quan:

- Tỷ lệ mắc bệnh tim trong nhóm có đường huyết cao (FBS = 1) không cao hơn đáng kể so với nhóm đường huyết bình thường.

Kết Luận:

- Đường huyết cao có thể là một yếu tố rủi ro, nhưng kết quả hiện tại không cho thấy sự chênh lệch rõ ràng giữa hai nhóm.
- Cần kết hợp với các yếu tố khác như tuổi tác, huyết áp để phân tích sâu hơn mối quan hệ giữa đường huyết và bệnh tim.

5.3. Tính tỷ lệ mắc bệnh tim theo loại cơn đau ngực

```
# Tính tỷ lệ mắc bệnh tim theo loại cơn đau ngực
query = """
    SELECT
        `Chest Pain`,
        Target,
        COUNT(*) AS count,
        ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(PARTITION BY `Chest Pain`), 2) AS percentage
    FROM heart_disease
    GROUP BY `Chest Pain`, Target
    ORDER BY `Chest Pain`, Target
"""

result_chest_pain_target = spark.sql(query)
result_chest_pain_target.show()
```

```
+----------+------+-----+----------+
|Chest Pain|Target|count|percentage|
+----------+------+-----+----------+
|         0|     0|  104|     72.73|
|         0|     1|   39|     27.27|
|         1|     0|    9|     18.00|
|         1|     1|   41|     82.00|
|         2|     0|   18|     20.93|
|         2|     1|   68|     79.07|
|         3|     0|    7|     30.43|
|         3|     1|   16|     69.57|
+----------+------+-----+----------+
```

Nhận Xét:

- Nhóm không đau ngực (Chest Pain = 0): Tỷ lệ mắc bệnh tim thấp (27.27%), phản ánh rõ ràng rằng không đau ngực ít liên quan đến bệnh tim.
- Nhóm đau ngực nhẹ (Chest Pain = 1): Tỷ lệ mắc bệnh tim rất cao (82%), cho thấy cơn đau ngực nhẹ là dấu hiệu đáng lo ngại cần được theo dõi.
- Nhóm đau ngực trung bình (Chest Pain = 2): 79.07% mắc bệnh tim, tương tự như nhóm đau ngực nhẹ.
- Nhóm đau ngực nặng (Chest Pain = 3): Tỷ lệ mắc bệnh tim vẫn cao (69.57%), nhưng thấp hơn so với nhóm đau ngực nhẹ và trung bình.

Kết Luận:

- Cơn đau ngực là yếu tố quan trọng và có mối liên hệ chặt chẽ với bệnh tim.

- Nhóm không có đau ngực có tỷ lệ mắc bệnh rất thấp, trong khi các nhóm có cơn đau ngực nhẹ, trung bình, và nặng đều có tỷ lệ mắc bệnh tim cao (trên 69%).
- Cần đặc biệt chú ý đến nhóm bệnh nhân có triệu chứng đau ngực (Chest Pain = 1 hoặc 2).

5.4. Tính tỷ lệ mắc bệnh tim theo huyết áp khi nghỉ ngơi

```
# Tính tỷ lệ mắc bệnh tim theo huyết áp khi nghỉ ngơi
query = """
    SELECT
        CONCAT(CAST(BP_Group * 50 AS STRING), '-', CAST(BP_Group * 50 + 50 AS STRING)) AS Resting_BP_Range,
        Target,
        COUNT(*) AS count,
        ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(PARTITION BY BP_Group), 2) AS percentage
    FROM (
        SELECT
            FLOOR(Resting_BP / 50) AS BP_Group,
            Target
        FROM heart_disease
    ) sub
    GROUP BY BP_Group, Target
    ORDER BY BP_Group;
"""

result_resting_bp_target = spark.sql(query)
result_resting_bp_target.show()
```

```
+----------------+------+-----+----------+
|Resting_BP_Range|Target|count|percentage|
+----------------+------+-----+----------+
|          50-100|     1|    2|    100.00|
|         100-150|     0|  109|     43.78|
|         100-150|     1|  140|     56.22|
|         150-200|     1|   22|     44.00|
|         150-200|     0|   28|     56.00|
|         200-250|     0|    1|    100.00|
+----------------+------+-----+----------+
```

Nhận xét:

Nhóm huyết áp nghỉ ngơi 100-150:

- Tỷ lệ mắc bệnh tim cao hơn không mắc bệnh (56.22% so với 43.78%).
- Đây là nhóm phổ biến nhất với số mẫu lớn nhất (tổng cộng 249 người).

Nhóm huyết áp nghỉ ngơi 150-200:

- Tỷ lệ không mắc bệnh cao hơn (56.00%) so với tỷ lệ mắc bệnh (44.00%).

- Kích thước mẫu nhỏ hơn (50 người), kết quả cần cẩn trọng khi khái quát hóa.

Nhóm huyết áp cực cao hoặc cực thấp (50-100 và 200-250):

- Rất ít mẫu, tỷ lệ không đủ để rút ra kết luận chính xác.

Kết luận:

- Huyết áp trong khoảng 100-150 có tỷ lệ mắc bệnh cao nhất.
- Kết quả ở các nhóm huyết áp cực đoan có thể không đáng tin cậy do số lượng mẫu nhỏ.

5.5. Tính tỷ lệ mắc bệnh tim khi tập thể dục

5]:
```
# Tính tỷ lệ mắc bệnh tim khi tập thể dục
query = """
    SELECT
        Exercise_Induced_Angina,
        Target,
        COUNT(*) AS count,
        ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(PARTITION BY Exercise_Induced_Angina), 2) AS percentage
    FROM heart_disease
    GROUP BY Exercise_Induced_Angina, Target
    ORDER BY Exercise_Induced_Angina, Target
"""

result_exercise_angina_target = spark.sql(query)
result_exercise_angina_target.show()
```

```
+----------------------+------+-----+----------+
|Exercise_Induced_Angina|Target|count|percentage|
+----------------------+------+-----+----------+
|                     0|     0|   62|     30.54|
|                     0|     1|  141|     69.46|
|                     1|     0|   76|     76.77|
|                     1|     1|   23|     23.23|
+----------------------+------+-----+----------+
```

Nhận xét:

Nhóm không bị đau thắt ngực khi tập thể dục (Exercise_Induced_Angina = 0):

- Tỷ lệ mắc bệnh tim rất cao (69.46%) so với không mắc bệnh (30.54%).
- Điều này cho thấy những người không bị đau thắt ngực khi tập thể dục có khả năng mắc bệnh tim cao hơn.

Nhóm bị đau thắt ngực khi tập thể dục (Exercise_Induced_Angina = 1):

- Tỷ lệ không mắc bệnh tim vượt trội (76.77%) so với mắc bệnh (23.23%).

Tổng quan: Kết quả gây bất ngờ, vì thường kỳ vọng đau thắt ngực khi tập thể dục là một dấu hiệu rõ rệt của bệnh tim. Điều này cho thấy có thể có các yếu tố khác đang ảnh hưởng, chẳng hạn như tuổi tác, mức hoạt động thể chất hoặc các biến số khác.

Kết luận:

- Những người không bị đau thắt ngực khi tập thể dục có nguy cơ mắc bệnh tim cao hơn.
- Điều này có thể phản ánh sự tương tác phức tạp giữa tập thể dục và các yếu tố khác.

5.6. Tính tỷ lệ mắc bệnh tim theo Cholestrol

```
#Tính tỷ lệ mắc bệnh tim theo Cholestrol
query = """
    SELECT
        CONCAT(CAST(Chol_Group * 50 AS STRING), '-', CAST(Chol_Group * 50 + 50 AS STRING)) AS Cholestrol_Range,
        Target,
        COUNT(*) AS count,
        ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER(PARTITION BY Chol_Group), 2) AS percentage
    FROM (
        SELECT
            FLOOR(Cholestrol / 50) AS Chol_Group,
            Target
        FROM heart_disease
    ) sub
    GROUP BY Chol_Group, Target
    ORDER BY Chol_Group, Target;
"""

result_cholesterol_range = spark.sql(query)
result_cholesterol_range.show()
```

```
+----------------+------+-----+----------+
|Cholestrol_Range|Target|count|percentage|
+----------------+------+-----+----------+
|         100-150|     0|    2|     40.00|
|         100-150|     1|    3|     60.00|
|         150-200|     0|   18|     40.91|
|         150-200|     1|   26|     59.09|
|         200-250|     0|   50|     40.32|
|         200-250|     1|   74|     59.68|
|         250-300|     0|   47|     55.29|
|         250-300|     1|   38|     44.71|
|         300-350|     0|   18|     50.00|
|         300-350|     1|   18|     50.00|
|         350-400|     0|    1|     25.00|
|         350-400|     1|    3|     75.00|
|         400-450|     0|    2|     66.67|
|         400-450|     1|    1|     33.33|
|         550-600|     1|    1|    100.00|
+----------------+------+-----+----------+
```

Nhận xét và kết luận:

Cholesterol ở mức 100-250:

- Đây là nhóm phổ biến nhất trong dữ liệu và cũng có nguy cơ mắc bệnh tim cao nhất (~59-60%).
- Điều này cho thấy ngay cả ở mức cholesterol thấp đến trung bình, nguy cơ mắc bệnh tim vẫn đáng kể và cần được theo dõi kỹ lưỡng.

Cholesterol ở mức 250-350:

- Ở mức cholesterol cao hơn, tỷ lệ mắc bệnh tim và không mắc bệnh tương đối cân bằng (~50%).
- Điều này có thể phản ánh rằng các yếu tố khác (tuổi tác, tiền sử gia đình, thói quen sống) cũng có vai trò quan trọng ngoài mức cholesterol.

Cholesterol cực cao (>350):

- Tỷ lệ mắc bệnh tim có xu hướng rất cao trong các nhóm này (75%-100%), tuy nhiên kích thước mẫu nhỏ không đủ để rút ra kết luận chắc chắn.
- Nhóm này nên được quan tâm đặc biệt và cần thêm dữ liệu để khẳng định kết quả.

6. THUẬT TOÁN KHAI THÁC DỮ LIỆU

6.1. Khái niệm thuật toán Random Forest

Random Forest là một thuật toán học máy thuộc nhóm ensemble learning, nghĩa là nó sử dụng nhiều mô hình cơ bản (cụ thể là Decision Trees) để đưa ra kết quả tổng hợp. Mô hình này có thể được áp dụng cho cả bài toán phân loại và hồi quy.

Ý tưởng chính là xây dựng nhiều cây quyết định độc lập, mỗi cây được huấn luyện trên một tập con dữ liệu ngẫu nhiên và một tập con đặc trưng ngẫu nhiên, sau đó kết hợp kết quả từ các cây để tạo ra một dự đoán mạnh mẽ hơn.

Các thành phần chính của Random Forest

- Decision Trees:
    - Mỗi cây quyết định trong rừng là một mô hình phân loại hoặc hồi quy độc lập.
    - Mỗi cây được xây dựng bằng cách chia dữ liệu thành các nhánh dựa trên tiêu chí tối ưu (chẳng hạn Gini Index hoặc Entropy đối với bài toán phân loại).

- Bootstrap Aggregation (Bagging):
    - Random Forest sử dụng phương pháp Bagging để tạo ra các tập con ngẫu nhiên từ dữ liệu gốc.
    - Mỗi tập con được lấy mẫu ngẫu nhiên có hoàn lại (random sampling with replacement). Điều này giúp các cây có sự đa dạng về dữ liệu huấn luyện, tăng khả năng tổng quát hóa của mô hình.
- Random Subset of Features:
    - Để xây dựng mỗi cây, Random Forest chọn một tập con ngẫu nhiên các đặc trưng thay vì sử dụng toàn bộ đặc trưng.
    - Điều này làm giảm mối tương quan giữa các cây, giúp mô hình trở nên mạnh mẽ hơn.
- Tích hợp kết quả (Aggregation):
    - Bài toán phân loại: Random Forest sử dụng bỏ phiếu đa số (majority voting) từ các cây để xác định nhãn cuối cùng.
    - Bài toán hồi quy: Random Forest tính trung bình giá trị đầu ra của các cây.

6.2. Đặc điểm của Random Forest

- Khả năng chống overfitting:
    - Việc kết hợp nhiều cây giúp giảm rủi ro của việc một cây riêng lẻ học quá mức vào dữ liệu huấn luyện.
- Xử lý dữ liệu phức tạp:
    - Random Forest hoạt động tốt trên các tập dữ liệu có nhiều đặc trưng khác nhau (định lượng, định danh) và không đòi hỏi phải chuẩn hóa dữ liệu.
- Tính linh hoạt:
    - Random Forest có thể áp dụng cho nhiều loại bài toán khác nhau (phân loại, hồi quy, phát hiện bất thường).
- Ước lượng tầm quan trọng của đặc trưng:
    - Random Forest cung cấp khả năng đánh giá mức độ quan trọng của các đặc trưng, giúp hiểu được đặc trưng nào đóng vai trò quan trọng trong việc dự đoán.

6.3. Quy trình hoạt động của Random Forest

- Chuẩn bị dữ liệu:
    - Chia tập dữ liệu thành các tập huấn luyện và kiểm tra.

    - Nếu cần, thực hiện các bước tiền xử lý dữ liệu như mã hóa, loại bỏ giá trị NULL.
- Huấn luyện mô hình:
    - Tạo các tập con dữ liệu bằng phương pháp bootstrap sampling.
    - Xây dựng một cây quyết định cho mỗi tập con.
    - Ở mỗi nút trong cây, chọn một tập con đặc trưng ngẫu nhiên để tìm điểm chia nhánh tốt nhất.
- Dự đoán:
    - Mỗi cây trong rừng đưa ra một dự đoán riêng lẻ.
    - Kết hợp kết quả của các cây để đưa ra dự đoán cuối cùng.
- Đánh giá mô hình:
    - Sử dụng các độ đo như Accuracy, Precision, Recall, F1-score (đối với phân loại) hoặc RMSE, R^2 (đối với hồi quy).

6.4. Lý do chọn Random Forest cho bài toán này

- Khả năng xử lý dữ liệu đa dạng:
    - Random Forest hoạt động tốt với dữ liệu gồm cả thuộc tính định lượng (như age, chol) và định tính (như cp, thal).
    - Với dữ liệu này, các đặc trưng hỗn hợp và mối quan hệ phi tuyến tính giữa chúng có thể được Random Forest mô hình hóa tốt.
- Xử lý dữ liệu không cân bằng: Dựa trên giá trị trung bình của cột target (~0.51), dữ liệu khá cân bằng. Nhưng ngay cả khi mất cân đối, Random Forest vẫn hoạt động ổn định nhờ cơ chế lấy mẫu ngẫu nhiên.
- Độ chính xác cao nhờ tính tổng hợp: Random Forest giảm thiểu vấn đề overfitting (thường xảy ra với một cây quyết định đơn lẻ), nhờ đó đạt độ chính xác cao hơn, ngay cả với dữ liệu lớn.
- Khả năng xử lý dữ liệu phức tạp và nhiều chiều: Random Forest xây dựng nhiều cây quyết định (decision trees) và kết hợp các dự đoán từ chúng. Cách tiếp cận này giúp xử lý tốt dữ liệu có nhiều biến đầu vào (features) mà không cần giảm chiều dữ liệu.

6.5. Lập trình thuật toán

6.5.1. Xử lý dữ liệu song song trên RDD

```
data_rdd = data.rdd.map(lambda row: ",".join(map(str, row)))
```

```
data_rdd.take(3)
```

```
['49,1,2,120,188,0,1,139,0,2.0,1,3,3,0',
 '57,0,1,130,236,0,0,174,0,0.0,1,1,2,0',
 '59,1,0,170,326,0,0,140,1,3.4,0,0,3,0']
```

```
# Xử lý dữ liệu thành dạng RDD [(label, [features])]
header = data_rdd.first()
data_rdd = data_rdd.map(lambda line: line.split(",")) \
    .map(lambda fields: (int(fields[-1]), list(map(float, fields[:-1]))))  # label, features
```

```
data_rdd.take(5)
```

```
[(0, [49.0, 1.0, 2.0, 120.0, 188.0, 0.0, 1.0, 139.0, 0.0, 2.0, 1.0, 3.0, 3.0]),
 (0, [57.0, 0.0, 1.0, 130.0, 236.0, 0.0, 0.0, 174.0, 0.0, 0.0, 1.0, 1.0, 2.0]),
 (0, [59.0, 1.0, 0.0, 170.0, 326.0, 0.0, 0.0, 140.0, 1.0, 3.4, 0.0, 0.0, 3.0]),
 (1, [40.0, 1.0, 3.0, 140.0, 199.0, 0.0, 1.0, 178.0, 1.0, 1.4, 2.0, 0.0, 3.0]),
 (1, [51.0, 0.0, 2.0, 130.0, 256.0, 0.0, 0.0, 149.0, 0.0, 0.5, 2.0, 0.0, 2.0])]
```

Mỗi hàng (row) trong dữ liệu được chuyển thành chuỗi các giá trị, cách nhau bằng dấu phẩy (join).

Các phép map được thực thi song song trên các phân vùng dữ liệu (partitions) của data_rdd.

Dữ liệu được xử lý đồng thời bởi nhiều tác nhân tính toán (executors) trong hệ thống phân tán Spark.

6.5.2. Hàm bootstrap_sample_partition - Lấy mẫu bootstrap trong mỗi phân vùng

```
def bootstrap_sample_partition(partition_data, sample_fraction):
    """
    Lấy mẫu bootstrap từ dữ liệu trong một phân vùng.
    """
    partition_data_list = list(partition_data)
    n = len(partition_data_list)
    sample_size = int(n * sample_fraction)
    if n > 0:
        return [partition_data_list[random.randint(0, n - 1)] for _ in range(sample_size)]
    return []
```

Bootstrap sampling: Kỹ thuật lấy mẫu ngẫu nhiên với thay thế. Dữ liệu mẫu có thể chứa các phần tử lặp lại.

Kết quả:

Trả về một mẫu ngẫu nhiên từ dữ liệu của phân vùng, có kích thước bằng sample_size.

6.5.3. Hàm gini_impurity - Tính chỉ số Gini

```python
def gini_impurity(groups, classes):
    """
    Tính chỉ số Gini cho các nhóm sau khi split
    """
    n_instances = sum([len(group) for group in groups])
    gini = 0.0
    for group in groups:
        size = len(group)
        if size == 0:
            continue
        score = 0.0
        for class_val in classes:
            proportion = [row[0] for row in group].count(class_val) / size
            score += proportion ** 2
        gini += (1.0 - score) * (size / n_instances)
    return gini
```

Chỉ số Gini đo mức độ không thuần khiết của một nhóm (group).

Chỉ số Gini cho một nhóm được tính theo công thức sau:

$$\text{Gini(t)} = 1 - \sum_{k=1}^{K} p_k^2$$

Trong đó:

- K là số lượng lớp (classes)
- p_k là tỉ lệ của lớp k trong nhóm t.

Công thức tổng thể cho tất cả các nhóm:

$$Gini_{total} = \sum_{i=1}^{M} \frac{n_i}{N} . Gini(t_i)$$

Trong đó:

- M là số lượng nhóm (groups)
- n_i là số lượng mẫu trong nhóm i,
- N là tổng số mẫu trong tất cả các nhóm n(instances),
- Gini(t_i) là chỉ số Gini của nhóm i.

Kết quả:

Trả về chỉ số Gini của toàn bộ các nhóm (càng nhỏ càng tốt).

6.5.4. Hàm split_node - Chia dữ liệu theo một đặc trưng

```
5]: def split_node(data, feature_index, threshold):
        """
        Chia dữ liệu dựa trên một ngưỡng của đặc trưng
        """
        left = [row for row in data if row[1][feature_index] < threshold]
        right = [row for row in data if row[1][feature_index] >= threshold]
        return left, right
```

Ý nghĩa:

Chia dữ liệu thành hai nhóm (left và right) dựa trên một giá trị ngưỡng (threshold) của một đặc trưng (feature_index).

Kết quả:

Trả về hai nhóm dữ liệu:

- left: Những hàng có giá trị đặc trưng nhỏ hơn ngưỡng.
- right: Những hàng có giá trị đặc trưng lớn hơn hoặc bằng ngưỡng.

6.5.5. Hàm find_best_split - Tìm điểm chia tốt nhất

```
: def find_best_split(data, feature_indices):
    """
    Tìm điểm split tốt nhất với chỉ xét các đặc trưng trong feature_indices.
    """
    classes = list(set([row[0] for row in data]))
    b_index, b_value, b_groups = None, None, None
    b_score = float('inf')
    for feature_index in feature_indices:
        for row in data:
            groups = split_node(data, feature_index, row[1][feature_index])
            gini = gini_impurity(groups, classes)
            if gini < b_score:
                b_index, b_value, b_score, b_groups = feature_index, row[1][feature_index], gini, groups
    return {'index': b_index, 'value': b_value, 'groups': b_groups}
```

Ý nghĩa:

Tìm giá trị ngưỡng (threshold) và đặc trưng (feature_index) tốt nhất để chia dữ liệu, dựa trên chỉ số Gini.

6.5.6. Xây dựng cây quyết định đệ quy

```
def build_tree(data, max_depth, min_size, depth=0, n_features=None):
    """
    Xây dựng cây quyết định đệ quy
    """
    classes = [row[0] for row in data]
    if len(set(classes)) == 1 or depth >= max_depth or len(data) <= min_size:
        return max(set(classes), key=classes.count)

    # Chọn ngẫu nhiên n_features đặc trưng để chia
    feature_indices = random.sample(range(len(data[0][1])), n_features)

    node = find_best_split(data, feature_indices)
    left, right = node['groups']
    node['left'] = build_tree(left, max_depth, min_size, depth + 1, n_features)
    node['right'] = build_tree(right, max_depth, min_size, depth + 1, n_features)
    return node
```

Dừng đệ quy khi:

- Tất cả các nhãn (classes) giống nhau (len(set(classes)) == 1).
- Cây đạt đến độ sâu tối đa (depth >= max_depth).
- Dữ liệu không đủ để chia tiếp (len(data) <= min_size). Trong trường hợp này, trả về nhãn có tần suất lớn nhất trong dữ liệu hiện tại (majority class).

Chọn ngẫu nhiên các đặc trưng:

- Sử dụng random.sample để chọn một số đặc trưng ngẫu nhiên trong data[0][1].

Tìm điểm chia tốt nhất:

- Gọi find_best_split để xác định điểm chia tối ưu (feature và threshold).

Chia dữ liệu:

- Phân chia dữ liệu thành left và right.

Tạo cây đệ quy:

- Gọi đệ quy build_tree cho nhánh trái (left) và phải (right).

6.5.7. Hàm map_reduce_process

```
# Xử lý MapReduce để xây dựng cây quyết định từ mỗi phân vùng dữ liệu
def map_reduce_process(partition_data, max_depth, min_size, n_features, sample_fraction):
    """
    Huấn luyện cây quyết định trên mẫu bootstrap từ mỗi phân vùng dữ liệu.
    """
    bootstrap_data = bootstrap_sample_partition(partition_data, sample_fraction)
    if bootstrap_data:
        tree = build_tree(bootstrap_data, max_depth, min_size, n_features=n_features)
        return [tree]
    return []
```

Mục đích:

Huấn luyện cây quyết định trên từng phân vùng dữ liệu bằng phương pháp Bootstrap Sampling.

Chi tiết:

- Lấy mẫu bootstrap: Sử dụng hàm bootstrap_sample_partition để lấy mẫu dữ liệu ngẫu nhiên (có lặp).
- Xây dựng cây: Nếu dữ liệu bootstrap không rỗng, gọi build_tree để xây dựng cây quyết định.
- Trả về cây dưới dạng danh sách: Kết quả là một danh sách [tree] hoặc [] nếu dữ liệu rỗng.

6.5.8. Hàm predict tree

```
# Dự đoán với một cây quyết định
def predict_tree(tree, row):
    """
    Dự đoán một hàng dữ liệu dựa vào cây quyết định
    """
    if row[tree['index']] < tree['value']:
        if isinstance(tree['left'], dict):
            return predict_tree(tree['left'], row)
        else:
            return tree['left']
    else:
        if isinstance(tree['right'], dict):
            return predict_tree(tree['right'], row)
        else:
            return tree['right']
```

Mục đích:

Dự đoán nhãn của một hàng dữ liệu dựa vào cây quyết định.

Logic:

- Dựa trên giá trị của đặc trưng tree['index'], so sánh với ngưỡng tree['value'].
- Nếu nhỏ hơn ngưỡng, duyệt xuống nhánh trái (tree['left']).
- Nếu lớn hơn hoặc bằng ngưỡng, duyệt xuống nhánh phải (tree['right']).
- Khi gặp node lá, trả về nhãn của node lá đó.

6.5.9. Hàm predict forest

```
]: # Dự đoán với Random Forest
def predict_forest(trees, row):
    """
    Dự đoán bằng cách lấy mode từ các cây trong Random Forest
    """
    predictions = [predict_tree(tree, row[1]) for tree in trees]
    return max(set(predictions), key=predictions.count)
```

Mục đích:

Dự đoán nhãn bằng cách tổng hợp dự đoán từ nhiều cây trong Random Forest.

Logic:

- Gọi predict_tree cho từng cây trong trees để thu được danh sách các dự đoán.
- Trả về nhãn xuất hiện nhiều nhất (mode) trong danh sách dự đoán.

6.5.10. Chia dữ liệu train-test, huấn luyện Random Forest, dự đoán trên test set

```
1]: # Chia dữ liệu train-test
    train_rdd, test_rdd = data_rdd.randomSplit([0.8, 0.2])

2]: # Huấn luyện Random Forest bằng MapReduce
    n_trees = 100
    max_depth = 5
    min_size = 10
    n_features = int(sqrt(len(train_rdd.first()[1])))  # Chọn số đặc trưng ngẫu nhiên tại mỗi nút

3]: # Sử dụng Bootstrap Sampling và huấn luyện cây cho mỗi mẫu bootstrap
    trees_rdd = train_rdd.mapPartitions(
        lambda partition_data: map_reduce_process(
            partition_data, max_depth, min_size, n_features, 1
        )
    )

4]: # Lấy ra n cây quyết định từ các phân vùng
    forest = trees_rdd.take(n_trees)

5]: # Dự đoán trên test set
    predictions = test_rdd.map(lambda row: (row[0], predict_forest(forest, row)))
```

Mục đích:

Tạo n_trees cây quyết định bằng cách áp dụng MapReduce trên từng phân vùng dữ liệu.

Chi tiết:

- Mỗi phân vùng dữ liệu được xử lý bằng map_reduce_process để sinh ra một cây.
- Kết quả của tất cả phân vùng là forest gồm n_trees cây.
- Dự đoán nhãn của từng hàng dữ liệu trong test_rdd bằng Random Forest.

7. KẾT QUẢ ĐẠT ĐƯỢC

7.1. Khái niệm các độ đo

7.1.1. Accuracy (Độ chính xác)

Công thức:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- TP: True Positives (dự đoán đúng cho lớp dương).
- TN: True Negatives (dự đoán đúng cho lớp âm).
- FP: False Positives (dự đoán sai dương).
- FN: False Negatives (dự đoán sai âm).

Ý nghĩa:

Accuracy đo lường tỷ lệ dự đoán chính xác trên toàn bộ tập dữ liệu. Nó cung cấp cái nhìn tổng quan về hiệu suất mô hình.

Lý do lựa chọn:

Accuracy dễ hiểu và tính toán đơn giản. Nó phù hợp trong bài toán nếu dữ liệu giữa các lớp (dương/âm) được phân phối đồng đều.

7.1.2. F1-Score

$$\text{F1} - \text{Score} = 2 \times \frac{Precision \times \text{Recall}}{Precision + \text{Recall}}$$

Ý nghĩa:

- Precision (Độ chính xác của lớp dương) cho biết trong tất cả các dự đoán là dương, bao nhiêu là đúng.
  - $\text{Precision} = \frac{TP}{TP + FP}$
- Recall đo lường khả năng của mô hình trong việc phát hiện tất cả các trường hợp dương.
  - $\text{Recall} = \frac{TP}{TP + FN}$
- F1-Score là trung bình điều hòa giữa Precision và Recall, giúp cân bằng giữa việc giảm dương tính giả (FP) và âm tính giả (FN).

Ý nghĩa trong bài toán dự đoán bệnh tim

Precision: Tỷ lệ mẫu dự đoán bệnh tim chính xác so với tổng số mẫu được mô hình dự đoán là bệnh tim.

Ví dụ: Nếu Precision cao, mô hình ít dự đoán nhầm những người không bị bệnh là bị bệnh.

Recall: Tỷ lệ mẫu bệnh tim thực sự được mô hình nhận diện chính xác.

Ví dụ: Nếu Recall cao, mô hình ít bỏ sót những người bị bệnh tim thực sự.

F1-Score kết hợp cả hai, mang ý nghĩa:

- Đảm bảo mô hình vừa không bỏ sót bệnh nhân bị bệnh tim (FN thấp), vừa không gây lo ngại không cần thiết với người khỏe mạnh (FP thấp).

Lý do lựa chọn F1-Score

Dữ liệu bệnh tim thường mất cân bằng:

- Trong thực tế, số lượng bệnh nhân không bị bệnh (lớp âm) thường lớn hơn số lượng bệnh nhân bị bệnh (lớp dương). Khi đó, các độ đo khác như Accuracy có thể không phản ánh đúng chất lượng mô hình.
- Ví dụ: Nếu 95% bệnh nhân không bị bệnh, mô hình dự đoán tất cả không bị bệnh vẫn đạt Accuracy 95% nhưng hoàn toàn vô dụng.

F1-Score cân bằng Precision và Recall:

- Precision cao giúp giảm FP, tránh báo động sai và gây lo lắng không cần thiết.
- Recall cao giúp giảm FN, quan trọng để không bỏ sót bệnh nhân bị bệnh thật sự.
- F1-Score hỗ trợ đánh giá tổng quan hiệu quả mô hình trong các bài toán như phát hiện bệnh, nơi mà cả FN và FP đều cần được kiểm soát chặt chẽ.

7.2 Kết quả đạt được sau khi đánh giá mô hình

```
accuracy = correct_predictions / total_predictions
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.71
```

```
# In kết quả
print(f"F1-Score: {f1_score:.2f}")
```

```
F1-Score: 0.73
```

7.2.1. Giải thích nghĩa các chỉ số

Accuracy (0.71): Tỷ lệ dự đoán chính xác của mô hình trên tổng số mẫu dữ liệu.

- Với 71% mẫu được dự đoán đúng, mô hình có độ chính xác tương đối ổn định.

F1-Score (0.73):

- Trung bình điều hòa giữa Precision và Recall, F1-Score phản ánh sự cân bằng giữa việc giảm dương tính giả (FP) và âm tính giả (FN).
- Với giá trị 0.73, mô hình đã thể hiện hiệu suất khá tốt trong việc cân bằng giữa Precision và Recall.

7.2.2. Đánh giá từng khía cạnh

- F1-Score = 0.73 cho thấy Precision và Recall tương đối đồng đều, không có sự chênh lệch quá lớn.

Accuracy so với F1-Score:

- Accuracy cao hơn F1-Score, nhưng không phải là thước đo tối ưu nếu dữ liệu mất cân bằng (ví dụ, số lượng người không bị bệnh lớn hơn số lượng người bị bệnh).
- F1-Score 0.73 phản ánh tốt hơn về khả năng tổng quát của mô hình trong việc xử lý dữ liệu mất cân bằng, đặc biệt trong bài toán bệnh tim.

7.2.3. Phân tích kết quả

Điểm mạnh:

- Mô hình đã đạt độ chính xác tương đối tốt với Accuracy = 0.71.
- F1-Score = 0.73 cho thấy mô hình có khả năng cân bằng tốt giữa việc giảm FP và FN.
- Phù hợp cho bài toán phát hiện bệnh tim, nơi cần giảm thiểu FN (không bỏ sót bệnh nhân mắc bệnh) và kiểm soát FP (tránh báo động sai).

Hạn chế:

- Accuracy = 0.71 vẫn để lại khoảng 29% mẫu không được dự đoán chính xác, có thể ảnh hưởng đến việc ra quyết định y khoa.
- F1-Score = 0.73 cho thấy mô hình có thể cần cải thiện thêm trong việc tối ưu hóa Precision hoặc Recall.

8. KẾT LUẬN

Phương pháp Random Forest đã cho kết quả tương đối tốt trong bài toán dự đoán bệnh tim, đặc biệt với dữ liệu lớn và khả năng xử lý phức tạp thông qua PySpark. Tuy nhiên, vẫn còn nhiều dư địa để cải thiện cả về hiệu quả mô hình lẫn hiệu suất tính toán.

8.1. Ưu điểm

- **Khả năng song song hóa:** Việc huấn luyện từng cây có thể được thực hiện độc lập, cho phép song song hóa và tăng tốc độ huấn luyện trên các hệ thống đa lõi hoặc cụm máy tính.
- **Độ chính xác ổn định:** Kết hợp dự đoán từ nhiều cây giúp giảm phương sai và tạo ra dự đoán ổn định, chính xác hơn so với cây quyết định đơn lẻ.
- **Chống overfitting**: Random Forest, nhờ việc kết hợp nhiều cây quyết định (ensemble learning), giúp giảm nguy cơ overfitting so với một cây quyết định đơn lẻ.
- **Đánh giá tầm quan trọng của đặc trưng:** Random Forest cung cấp thước đo về tầm quan trọng của các đặc trưng bằng cách đánh giá sự giảm độ lỗi (impurity) khi sử dụng từng đặc trưng trong các cây. Giúp nhận biết đặc trưng nào quan trọng nhất, từ đó cải thiện hiệu suất hoặc giảm kích thước dữ liệu.

8.2 Hạn chế

- **Kết quả còn chưa cao:** Với Accuracy đạt 71% và F1-Score là 73%, mô hình vẫn có thể được cải thiện.
- **Không tối ưu cho dữ liệu tuần tự hoặc phụ thuộc thời gian:** Random Forest không khai thác được thông tin từ dữ liệu tuần tự hoặc dữ liệu thời gian vì nó hoạt động dựa trên các điểm chia độc lập. Hiệu suất kém hơn các thuật toán khác trong các bài toán liên quan đến chuỗi thời gian.

8.3 Hướng phát triển

- Tối ưu hóa các tham số trong mô hình để có thể tăng độ chính xác lên cao hơn
- Phát triển ứng dụng thực tiễn: Tích hợp mô hình vào các ứng dụng y tế thực tế, ví dụ như hỗ trợ bác sĩ trong việc ra quyết định chẩn đoán bệnh tim.
- Tính toán để loại bỏ các thuộc tính ảnh hưởng kém đến kết quả cuối cùng.

9. NGUỒN THAM KHẢO

1. Random Forest Classifier using Scikit-learn. https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/
2. Phân lớp bằng Random Forests trong Python. https://viblo.asia/p/phan-lop-bang-random-forests-trong-python-djeZ1D2QKWz
3. Machine Learning From Scratch by AssemblyAI. https://github.com/AssemblyAI-Community/Machine-Learning-From-Scratch.
4. Apache Spark - A Unified engine for large-scale data analytics. https://spark.apache.org/docs/3.5.3/