

TRƯỜNG ĐẠI HỌC THỦY LỢI
KHOA CÔNG NGHỆ THÔNG TIN



BÀI TẬP LỚN
MÔN HỌC: KHAI PHÁ DỮ LIỆU

Tên đề tài:

KHAI PHÁ DỮ LIỆU BỆNH SUY TIM VÀ ĐƯA RA DỰ ĐOÁN

Giáo viên hướng dẫn: TRẦN MẠNH TUẤN

Các thành viên trong Nhóm 3_64TTNT1:

Họ tên sinh viên	Mã sinh viên
Nguyễn Thị Quỳnh Anh	2251262573
Nguyễn Lâm Tùng	2251262657
Đồng Anh Quân	2251262626

Hà Nội - 2024

MỤC LỤC

	Trang
LỜI CẢM ƠN	1
CHƯƠNG I: MÔ TẢ BÀI TOÁN	2
1. Lý do chọn đề tài	2
2. Tổng quan bài toán	2
3. Quy trình thực hiện	2
4. Phân tích dữ liệu thô	3
CHƯƠNG II: QUY TRÌNH KHAI PHÁ DỮ LIỆU	5
1. Tiền xử lý và trực quan hóa dữ liệu	
1.1 Khám phá dữ liệu ban đầu	5
1.2 Làm sạch dữ liệu	10
1.2.1 Xử lý dữ liệu bị thiếu	10
1.2.2 Xử lý dữ liệu trùng lặp	10
1.2.3 Xử lý điểm ngoại lai	11
1.3 Khám phá dữ liệu sau khi tiền xử lý	15
2. Biến đổi dữ liệu	19
3. Phân tách dữ liệu và chuẩn hóa dữ liệu	22
3.1 Phân tách dữ liệu	22
3.2 Chuẩn hóa dữ liệu cho từng tập	23
CHƯƠNG III. LỰA CHỌN PHƯƠNG PHÁP PHÂN LỚP VÀ ĐÁNH GIÁ .	25
1. Trainng mô hình từ bộ dữ liệu đã phân tách	25
2. Train mô hình với Cross-validate(đánh giá chéo)	33
3. Tối ưu hóa mô hình và đánh giá	35
CHƯƠNG IV. ỨNG DỤNG MÔ HÌNH	40
KẾT LUẬN	42
TÀI LIỆU THAM KHẢO	42

LỜI CẢM ƠN

Ngày nay, việc ứng dụng công nghệ thông tin đã trở nên phổ biến trong hầu hết mọi cơ quan, doanh nghiệp, trường học đặc biệt là việc áp dụng các giải pháp tin học trong y tế. Trong ít năm trở lại đây, với tốc độ phát triển như vũ bão, CNTT đang dần làm cho cuộc sống của con người trở nên thú vị và đơn giản hơn. Để bắt kịp với nhịp độ phát triển của xã hội, những kiến thức học được trên giảng đường là vô cùng quan trọng đối với mỗi sinh viên chúng em. Vì vậy chúng em chọn đề tài ***“Khai phá dữ liệu Bệnh suy tim và đưa ra dự đoán bằng phương pháp phân lớp”*** để làm báo cáo kết thúc môn học của mình.

Nhóm thực hiện chúng em xin gửi lời cảm ơn đặc biệt đến thầy giáo Trần Mạnh Tuấn đã tận tình hướng dẫn chúng em trong bộ môn Khai phá dữ liệu và giúp đỡ chúng em hoàn thành bài tập lớn cuối kỳ này.

Bên cạnh những kết quả mà chúng em đạt được thì sẽ không khó tránh khỏi những thiếu sót trong quá trình làm đề tài vì thời gian không cho phép và chưa có kinh nghiệm thực tế. Chính vì vậy chúng em rất mong được sự cảm thông, chỉ bảo góp ý của thầy cô. Những lời nhận xét, góp ý của thầy cô chính là một bài học, kiến thức cho chúng em trên con đường sau này.

Chúng em xin chân thành cảm ơn!

CHƯƠNG I: MÔ TẢ BÀI TOÁN

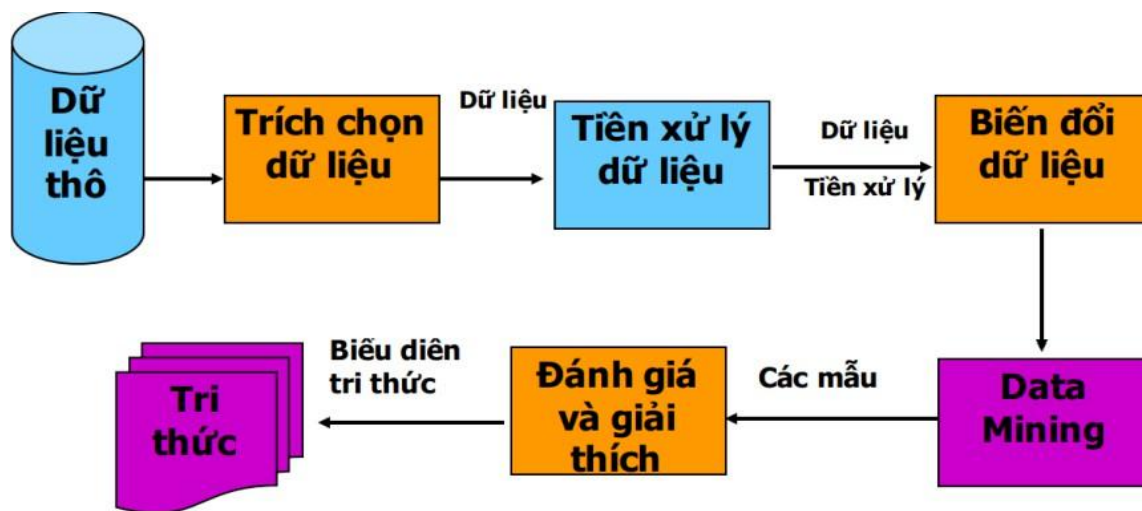
1. Lý do chọn đề tài

Hiện nay các bệnh tim mạch (CVD) là nguyên nhân gây ra tử vong số 1 trên toàn cầu. Trong số các bệnh do vấn đề về tim mạch gây ra, Suy tim là một bệnh vô cùng phổ biến và có thể sử dụng bộ dữ liệu để đưa ra dự đoán bệnh tim có thể xảy ra. Những người mắc bệnh tim mạch hoặc những người có nguy cơ mắc bệnh tim mạch cao (do có một hoặc nhiều yếu tố nguy cơ như tăng huyết áp, tiểu đường, tăng mỡ máu hoặc bệnh đã hình thành) cần được phát hiện và quản lý sớm. Vì vậy, nhóm em chọn đề tài ***“Khai phá dữ liệu Bệnh suy tim và dự đoán bằng phương pháp phân lớp”*** vì trong đó mô hình học máy có thể giúp ích rất nhiều.

2. Tổng quan bài toán

Dataset bao gồm các mô tả về các thuộc tính tương ứng với mức độ đánh giá các chỉ số sức khỏe của người khám bệnh. Áp dụng các thuật toán để xác định xem 1 người: bị bệnh tim(1), bình thường(0)

3. Quy trình thực hiện



- Quy trình thực hiện khai phá bao gồm 6 bước:

Bước 1: Tạo tập tin dữ liệu đầu vào

Bước 2: Tiền xử lý, làm sạch tập dữ liệu

Bước 3: Chọn tác vụ khai phá dữ liệu (phân lớp)

Bước 4: Khai phá dữ liệu: tìm kiếm tri thức

Bước 5: Đánh giá mẫu tìm được

Bước 6: Biểu diễn tri thức

- Ở bài này, nhóm em tóm tắt các bước thành những mục sau:

1. Thu thập dữ liệu
2. Tiền xử lý và trực quan hóa dữ liệu
3. Biến đổi dữ liệu
4. Phân tích dữ liệu và chuẩn hóa dữ liệu

4. Phân tích dữ liệu thô

```
df = pd.read_csv("heart1.csv")
```

```
df.head(10)
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
5	39	M	NAP	120	339	0	Normal	170	N	0.0	Up	0
6	45	F	ATA	130	237	0	Normal	170	N	0.0	Up	0
7	54	M	ATA	110	208	0	Normal	142	N	0.0	Up	0
8	37	M	ASY	140	207	0	Normal	130	Y	1.5	Flat	1
9	48	F	ATA	120	284	0	Normal	120	N	0.0	Up	0

```
df.shape
```

```
(918, 12)
```

Hình ảnh đọc dữ liệu thô

=> Sau khi hiển thị dữ liệu thô thì ta thấy: bộ dữ liệu hiển thị có 918 dòng và 12 cột

Bảng các thuộc tính

STT	Thuộc tính	Ý nghĩa thuộc tính
1	Age	Tuổi của bệnh nhân
2	Sex	Giới tính của bệnh nhân: + M: Nam + F: Nữ
3	ChesPainType	Loại đau ngực: + TA: Đau thắt ngực điển hình + ATA: Đau thắt ngực không điển hình + NAP: Đau không đau thắt ngực + ASY: Không có triệu chứng
4	RestingBP	Huyết áp lúc nghỉ [mm Hg]
5	Cholesterol	Mức cholesterol huyết thanh [mm/dl]
6	FastingBS	Chỉ số đường huyết lúc đói: + 1: nếu chỉ số đường huyết lúc đói >120mg/dl + 0: ngược lại
7	RestingECG	Kết quả điện tâm đồ khi nghỉ: + Normal: bình thường + ST: có bất thường sóng ST-T + LVH: cho thấy phì đại thất trái có thể xảy ra hoặc xác định theo tiêu chuẩn Estes
8	MaxHR	Nhịp tim tối đa đạt được [Giá trị số từ 60 đến 202]
9	ExeciseAngina	Đau thắt ngực do gắng sức: + Y: có + N: không
10	Oldpeak	Mức độ suy giảm ST (ST depression) trong bài kiểm tra gắng sức (exercise stress test)
11	ST_Slope	Độ dốc của đoạn ST trong điện tâm đồ: + Up: dốc lên + Flat: bằng phẳng +Down: dốc xuống
12	HeartDisease	Kết quả đầu ra + 1: bệnh tim + 0: Bình thường

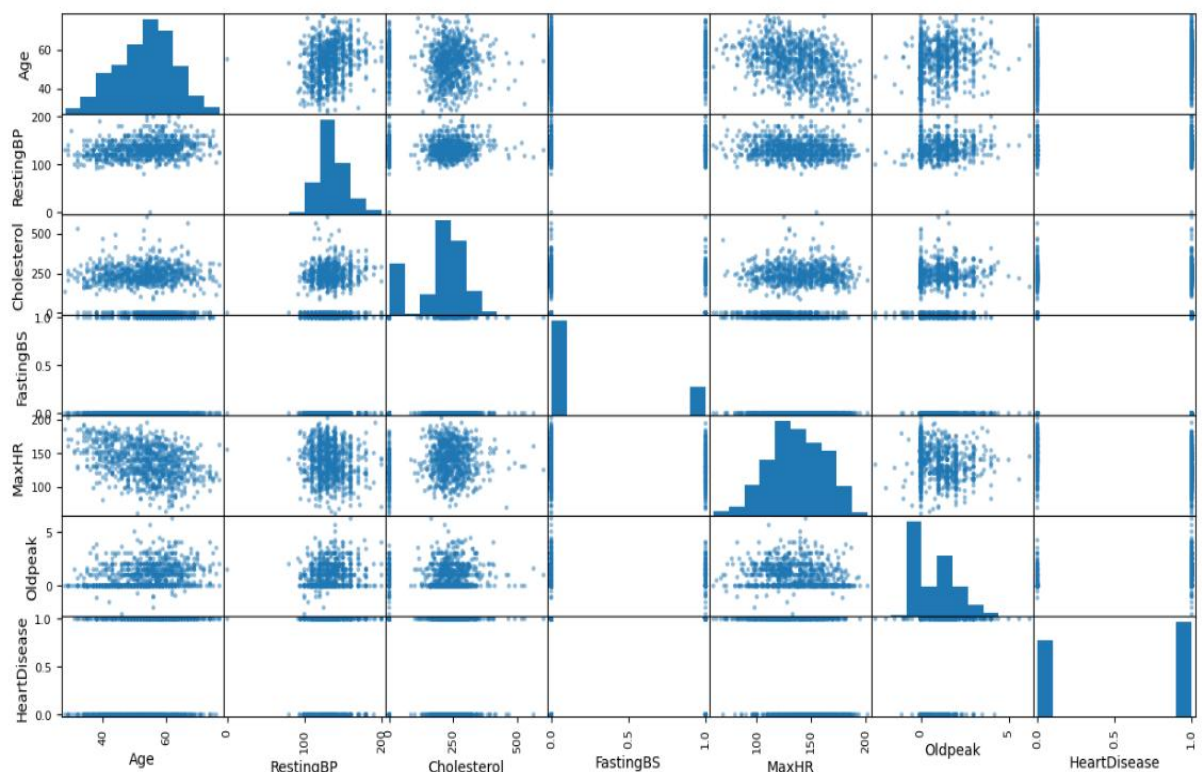
CHƯƠNG II: QUY TRÌNH KHAI PHÁ DỮ LIỆU

1. Tiền xử lý và trực quan hóa dữ liệu

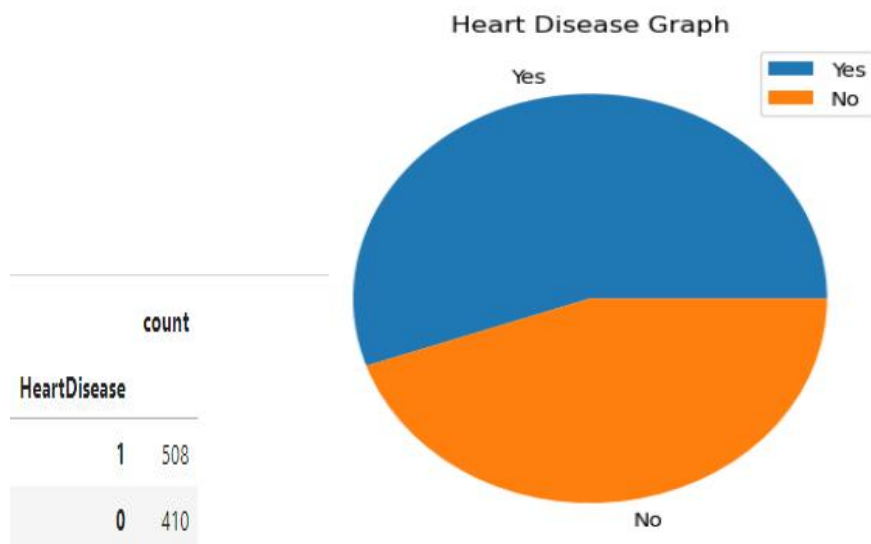
- Tiền xử lý: là quá trình xử lý dữ liệu thô/gốc nhằm cải thiện chất lượng dữ liệu và chất lượng của kết quả KPDL

- Trực quan hóa dữ liệu: là quá trình biểu diễn dữ liệu dưới dạng các đồ thị, biểu đồ hoặc hình ảnh trực quan. Mục đích chính là làm cho dữ liệu trở nên dễ hiểu hơn, từ đó hỗ trợ việc ra quyết định, phát hiện xu hướng và mẫu mà có thể không dễ dàng nhận ra khi chỉ nhìn vào dữ liệu thô.

1.1 Khám phá dữ liệu ban đầu



Hình ảnh sự tương quan giữa các thuộc tính khi chưa tiền xử lý



Hình ảnh sau khi trực quan hóa biến mục tiêu

- Sau khi trực quan hóa biến mục tiêu ta thấy:

+> Số lượng người bị bệnh tim(1) là: 508 người

+> Số lượng người không bị bệnh tim(0) là: 410 người

=> Từ biểu đồ tròn ta thấy tổng quát số người bị bệnh tim(1) và số người không bị bệnh tim(0) là khá cân bằng

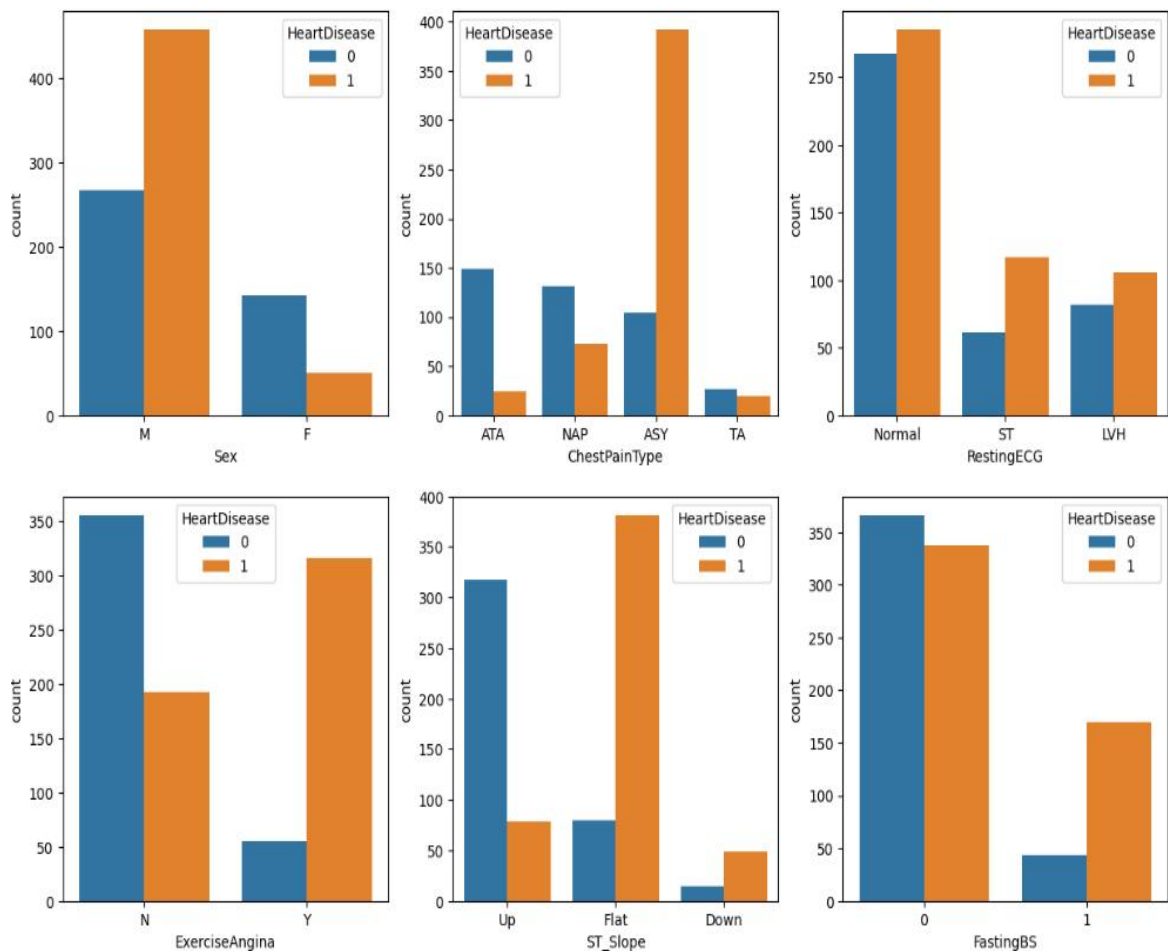
- Từ dữ liệu ta kiểm tra các dữ liệu nghi vấn là dữ liệu rời rạc:

```
columns = ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'ExerciseAngina', 'ST_Slope', 'HeartDisease']
for column in columns:
    print(column, ":", df[column].unique())
```

Sex : ['M' 'F']
 ChestPainType : ['ATA' 'NAP' 'ASY' 'TA']
 FastingBS : [0 1]
 RestingECG : ['Normal' 'ST' 'LVH']
 ExerciseAngina : ['N' 'Y']
 ST_Slope : ['Up' 'Flat' 'Down']
 HeartDisease : [0 1]

Hình ảnh thực hiện

- Sau đó ta trực quan hóa các dữ liệu rời rạc với biến mục tiêu để xem sự tương quan giữa chúng:



Hình ảnh sau khi trực quan

- Ở thuộc tính Sex (Giới tính):

- + Tỷ lệ mắc bệnh của Nam bị mắc bệnh cao hơn nam không bị mắc bệnh
- + Tỷ lệ mắc bệnh của Nữ không bị mắc bệnh cao hơn nữ bị mắc bệnh.

- Ở thuộc tính ChestPainType (Loại đau ngực):

- + Tỷ lệ đau thắt ngực điển hình(ATA) của người không bị bệnh tim cao hơn người bị bệnh
- + Tỷ lệ đau không đau thắt ngực (NAP) của người không bị bệnh cao hơn người bị bệnh
- + Tỷ lệ không có triệu chứng (ASY) của người bị bệnh cao hơn người không bị bệnh
- + Tỷ lệ đau thắt ngực điển hình(TA) của người không bị bệnh cao hơn người bị bệnh.

- Ở thuộc tính RestingECG(Kết quả điện tâm đồ khi nghỉ):

- + Tỷ lệ bình thường(normal) của người bị bệnh cao hơn người không bị bệnh
- + Tỷ lệ có bất thường sóng ST-T (ST) của người bị bệnh cao hơn người không bị bệnh
- + Tỷ lệ cho thấy phì đại thất trái có thể xảy ra hoặc xác định theo tiêu chuẩn Estes (LVH) của người bị bệnh cao hơn người không bị bệnh

-Ở thuộc tính ExerciseAngina (Đau thắt ngực do gắng sức):

- +N(No) của người không bị bệnh lớn hơn người bị bệnh
- +Y(yes) của người bị bệnh lớn hơn người không bị bệnh

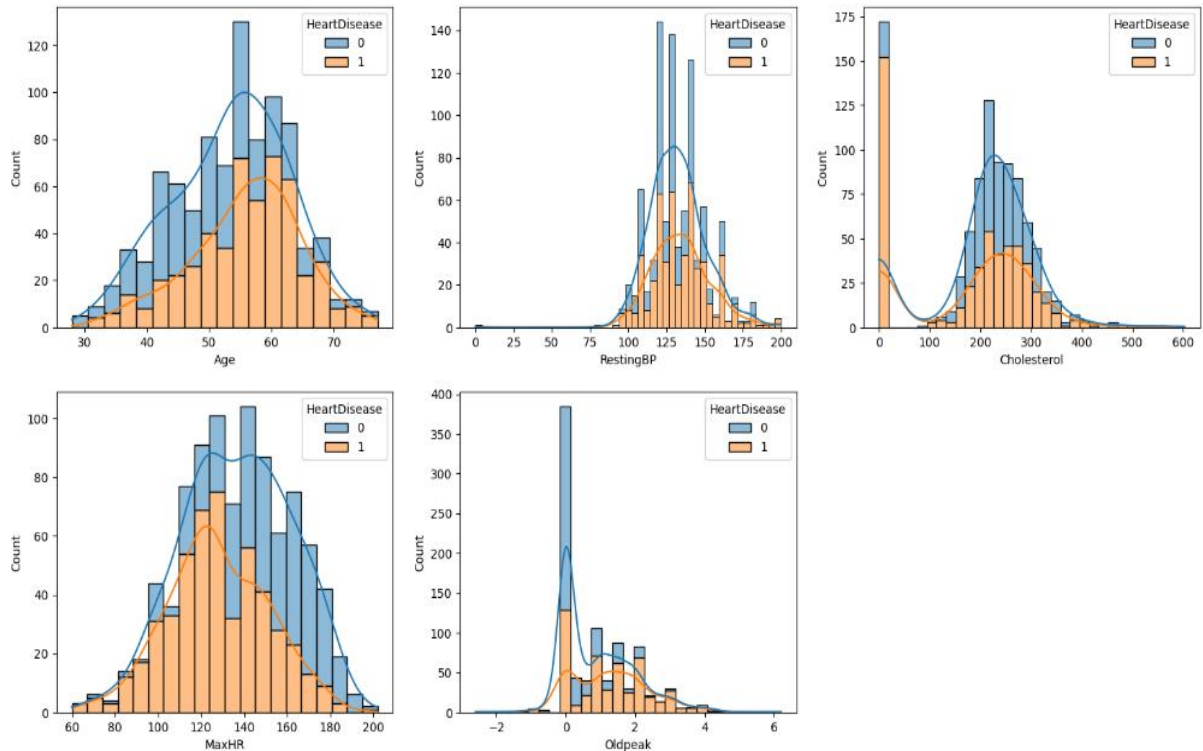
- Ở thuộc tính ST_Slope(Độ dốc của đoạn ST trong điện tâm đồ):

- + Tỷ lệ dốc lên (up) của người không bị bệnh cao hơn người bị bệnh
- + Tỷ lệ bằng phẳng (Flat) của người bị bệnh cao hơn người không bị bệnh
- + Tỷ lệ dốc xuống (Down) của người bị bệnh cao hơn người không bị bệnh

- Ở thuộc tính FastingBS(Chỉ số đường huyết lúc đói):

- + Tỷ lệ nếu chỉ số đường huyết lúc đói $>120\text{mg/dl}$ (1) của người không bị bệnh cao hơn người bị bệnh
- + Tỷ lệ ngược lại (0) của người bị bệnh cao hơn người không bị bệnh

- Tương tự ta trực quan hóa các dữ liệu liên tục với biến mục tiêu:



Hình ảnh sau khi trực quan

- Ở thuộc tính Age những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 50 đến 60 tuổi.
- Ở thuộc tính RestingBP những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 110 đến 150
- Ở thuộc tính Cholesterol những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 200 đến 300
- Ở thuộc tính MaxHR những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 100 đến 150
- Ở thuộc tính Oldpeak những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 0 đến 3

1.2 Làm sạch dữ liệu

- Làm sạch dữ liệu: là quá trình phát hiện và sửa chữa hoặc loại bỏ các lỗi, giá trị không hợp lệ, thiếu dữ liệu, hoặc các điểm bất thường trong bộ dữ liệu nhằm cải thiện chất lượng và độ tin cậy của dữ liệu

1.2.1 Xử lý dữ liệu bị thiếu

```
df.isnull().sum()
```

```
Age          0
Sex          0
ChestPainType 0
RestingBP    0
Cholesterol  0
FastingBS    0
RestingECG   0
MaxHR        0
ExerciseAngina 0
Oldpeak      0
ST_Slope     0
HeartDisease 0
dtype: int64
```

Hình ảnh kiểm tra dữ liệu thiếu

=> Từ hình trên ta thấy bộ dữ liệu không có dữ liệu bị thiếu

1.2.2 Xử lý dữ liệu trùng lặp

```
df.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
...
913     False
914     False
915     False
916     False
917     False
Length: 918, dtype: bool
```

```
df.drop_duplicates()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
...
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up	0

918 rows × 12 columns

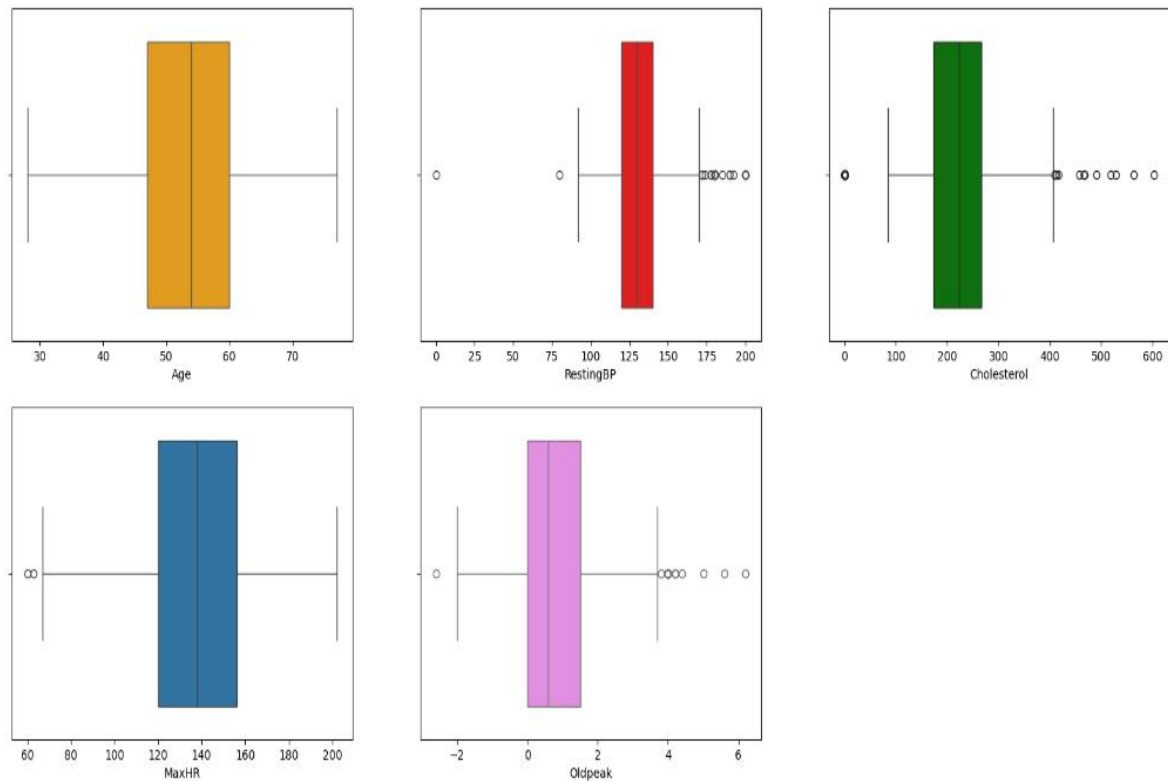
```
df.shape
```

```
(918, 12)
```

Hình ảnh sau khi thực hiện

=> Từ hình trên ta thấy bộ dữ liệu không bị trùng lặp

1.2.3 Xử lý điểm ngoại lai



Hình ảnh sau khi trực quan hóa dữ liệu liên tục

Nhận thấy với từng cột:

- Cột age : Không có điểm ngoại lai
- Cột RestingBP: Có điểm ngoại lai
- Cột Cholesterol: Có điểm ngoại lai
- Cột maxHR: Có điểm ngoại lai
- Cột Oldpeak: Có điểm ngoại lai

=> Nhận thấy cột cholesterol có 174 bản ghi chứa dữ liệu ngoại lai do đó chúng ta chọn thuộc tính Cholesterol để xử lý dữ liệu ngoại lai đầu tiên

A. Xử lý điểm ngoại lai cho dữ liệu cholesterol

```
new_df = df.copy()

new_df.shape

(918, 12)

upper_limit = new_df['Cholesterol'].quantile(0.99)
lower_limit = new_df['Cholesterol'].quantile(0.1899)

upper_limit, lower_limit

(411.4900000000001, 101.38300000000015)

new_df = new_df.loc[(new_df['Cholesterol'] <= upper_limit) & (new_df['Cholesterol'] >= lower_limit)]
```

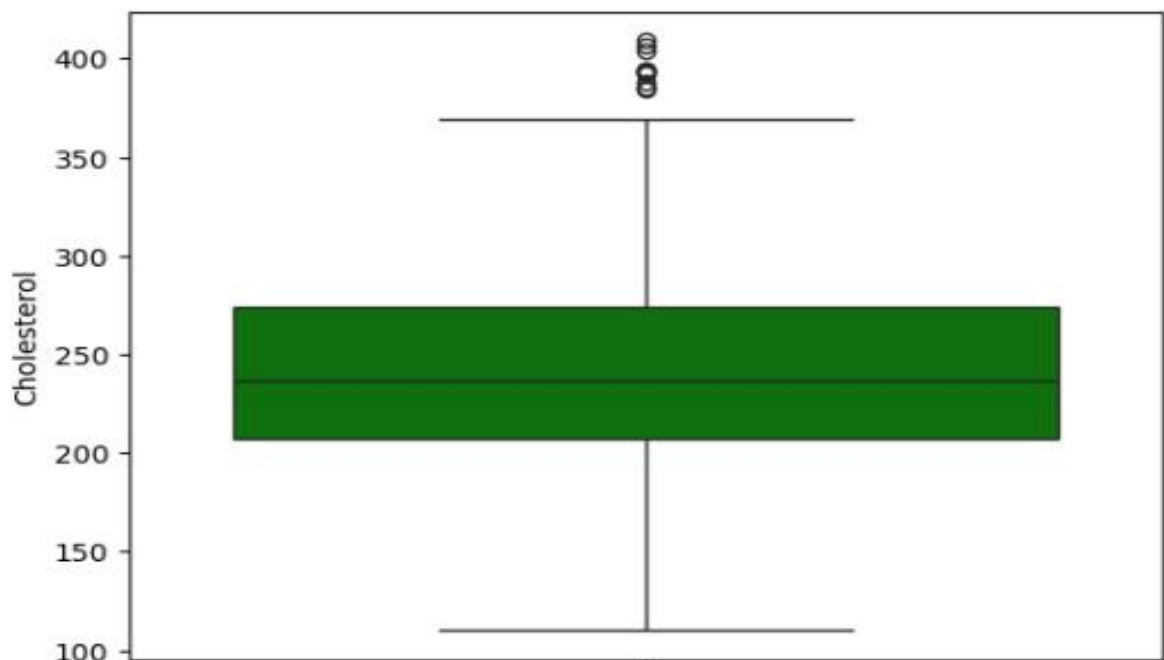
- Ta tạo 1 dataframe mới để copy dữ liệu ban đầu, điều này để tránh ảnh hưởng đến dữ liệu gốc

- Ta dùng phương pháp chọn khoảng phần trăm (percentile method) để tìm ra giá trị upper_limit (Giá trị giới hạn trên) và lower_limit (Giá trị giới hạn dưới), in ra kết quả ta được:

+> upper_limit có giá trị là: 411,490

+> lower_limit có giá trị là: 101,383

- Sau đó ta lọc dữ liệu trong khoảng [lower_limit, upper_limit] để loại bỏ giá trị ngoại lai



Hình ảnh sau khi lọc dữ liệu Cholesterol

=> Ta thấy mặc dù vẫn còn điểm ngoại lai được biểu hiện bằng chấm tròn nhưng những điểm ngoại lai đó phản ánh đúng trong hiện thực. Nếu bỏ chúng đi có thể khiến chúng mất đi tính thực tế và khiến cho mô hình có thể dự đoán sai với dữ liệu mà chưa từng gặp phải

B. Xử lý điểm ngoại lai cho dữ liệu *RestingBP*

```
upper_limit = new_df['RestingBP'].quantile(0.99)
lower_limit = new_df['RestingBP'].quantile(0.005)

upper_limit , lower_limit

(180.0, 97.32)

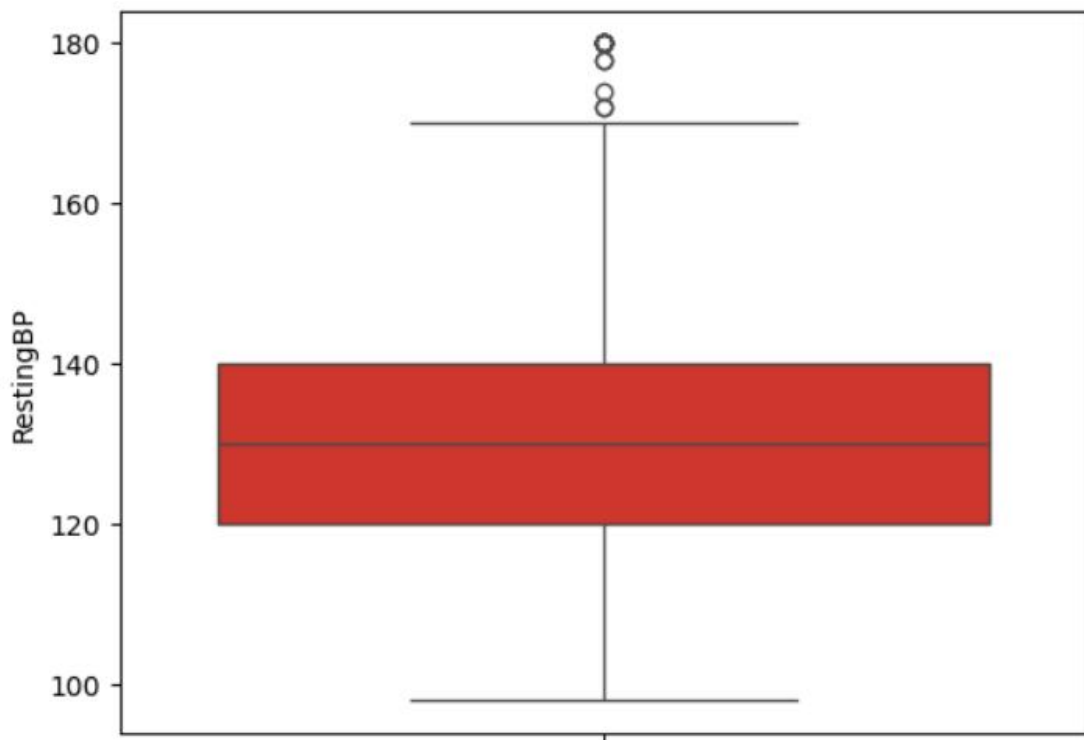
new_df = new_df.loc[(new_df['RestingBP'] <= upper_limit) & (new_df['RestingBP'] >= lower_limit)]
```

- Tương tự ta dùng phương pháp chọn khoảng phần trăm(percentile method) ta được:

+> upper_limit có giá trị là: 180,0

+> lower_limit có giá trị là: 97,32

- Sau đó ta lọc dữ liệu trong khoảng [lower_limit,upper_limit] để loại bỏ giá trị ngoại lai



Hình ảnh sau khi lọc dữ liệu RestingBP

C. Xử lý điểm ngoại lai cho dữ liệu Oldpeak

```
upper_limit = new_df['Oldpeak'].quantile(0.99)
lower_limit = new_df['Oldpeak'].quantile(0.1)
```

```
upper_limit , lower_limit
```

```
(4.0, 0.0)
```

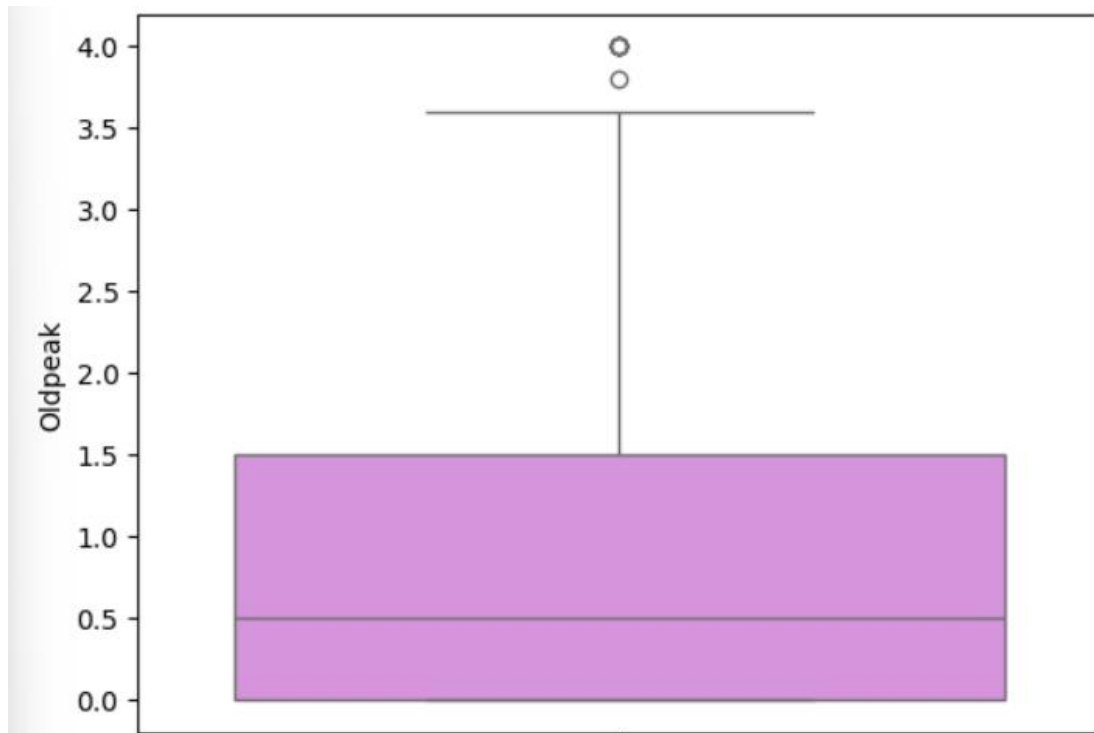
```
new_df = new_df.loc[(new_df['Oldpeak']<=upper_limit) & (new_df['Oldpeak']>=lower_limit)]
```

- Tương tự ta dùng phương pháp chọn khoảng phần trăm(percentile method) ta được:

+> upper_limit có giá trị là: 4,0

+> lower_limit có giá trị là: 0,0

- Sau đó ta lọc dữ liệu trong khoảng [lower_limit,upper_limit] để loại bỏ giá trị ngoại lai



Hình ảnh sau khi lọc dữ liệu Oldpeak


```
print(f'Số outlier đã loại bỏ là: {len(df)-len(new_df)}')
```

Số outlier đã loại bỏ là: 201

Bảng dữ liệu mới sau khi đã loại bỏ outlier

```
new_df.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

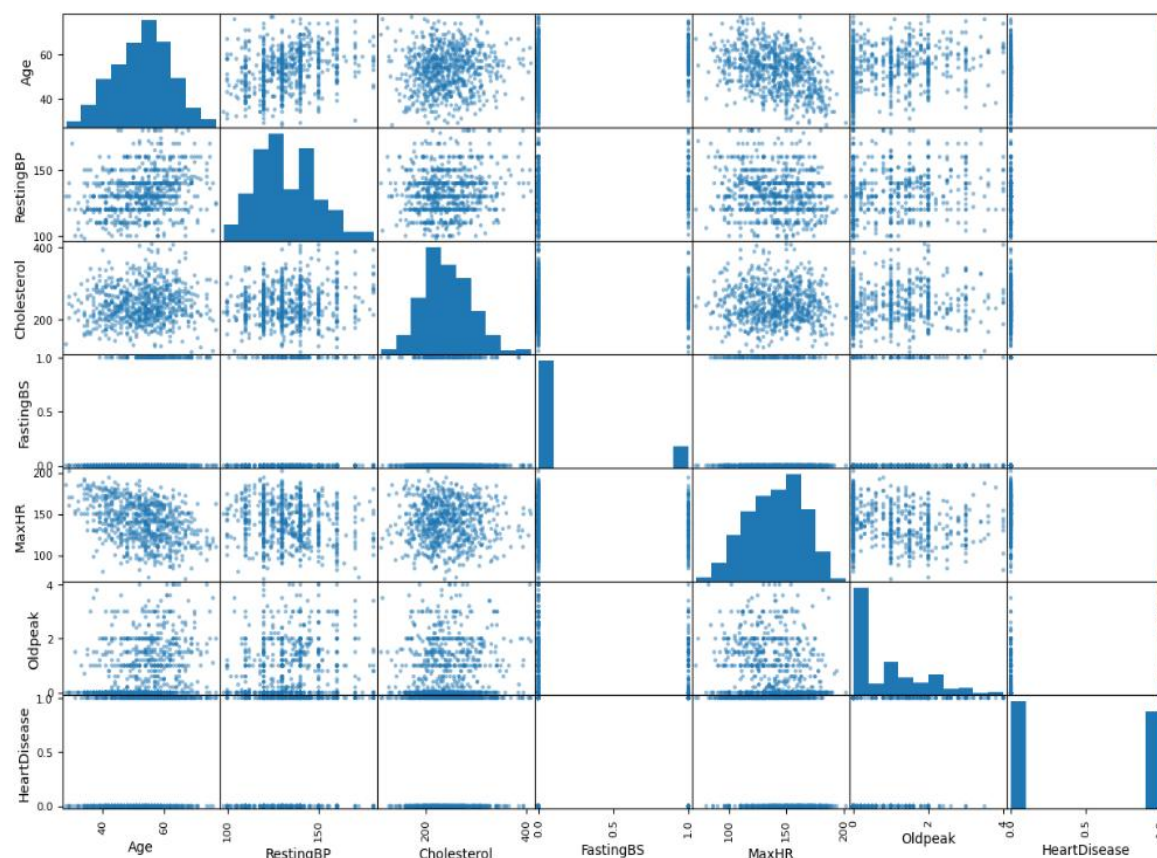
```
new_df.shape
```

(717, 12)

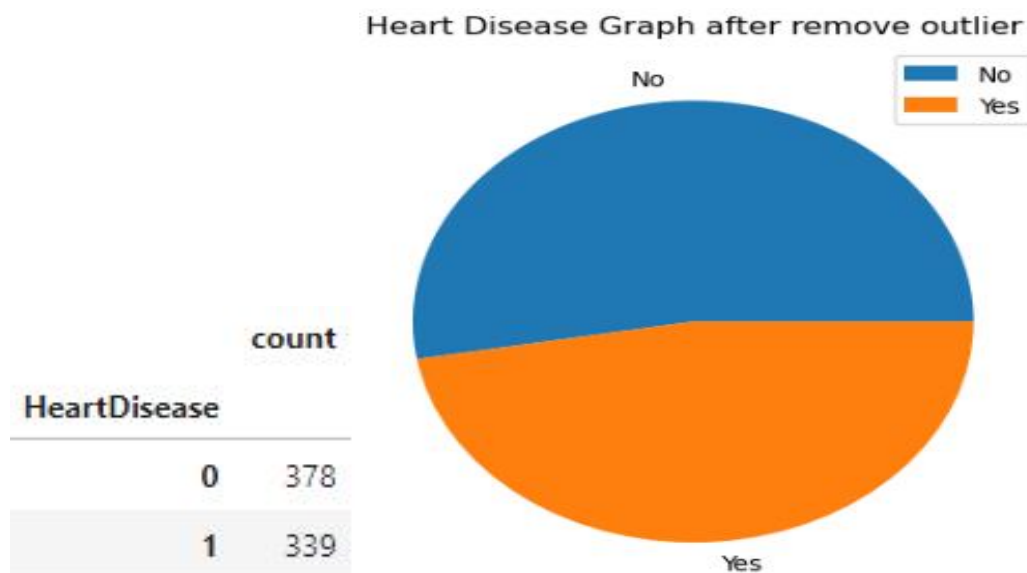
Hình ảnh sau khi loại bỏ điểm ngoại lai xong

=> Số điểm ngoại lai ta đã loại bỏ là 201 điểm, dữ liệu được thu gọn về còn 717 dòng và 11 cột

1.3 Khám phá dữ liệu sau khi tiền xử lý



Hình ảnh sự tương quan giữa các thuộc tính khi đã tiền xử lý



Hình ảnh trực quan hóa biến mục tiêu sau khi loại bỏ các điểm ngoại lai

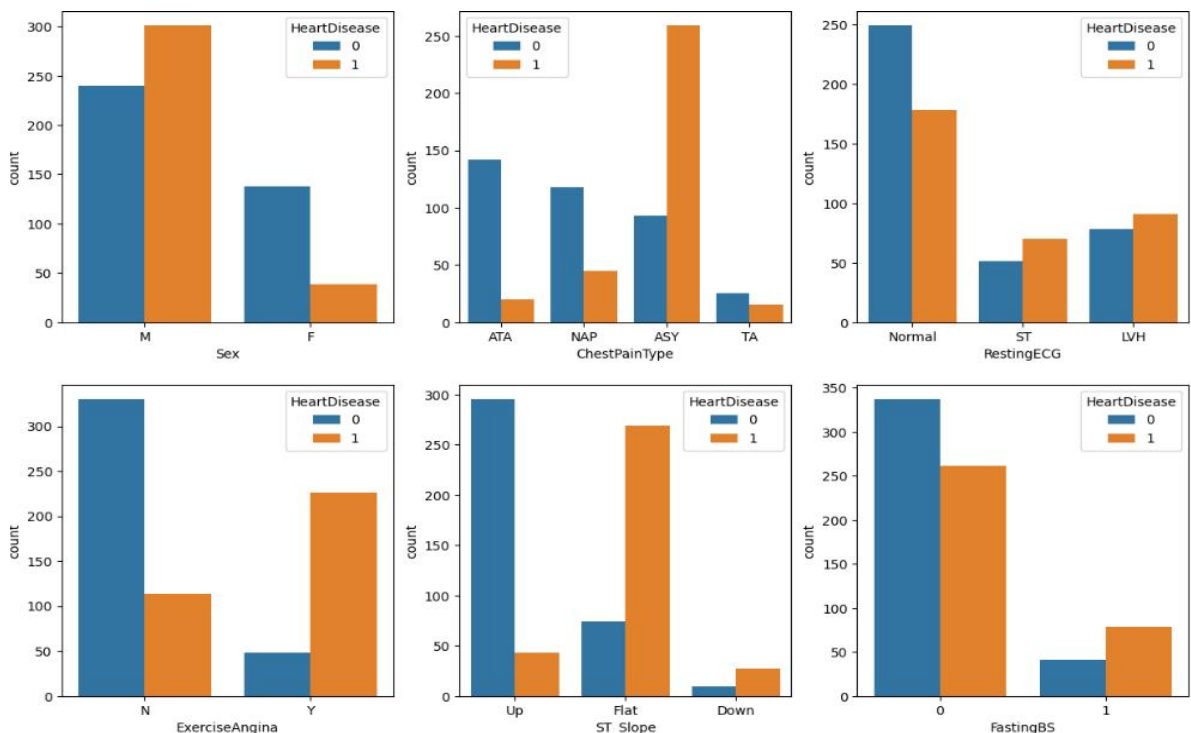
- Ta thấy:

+> Số lượng người bị bệnh tim(1) là: 339 người

+> Số lượng người không bị bệnh tim(0) là: 378 người

=> Từ biểu đồ tròn ta thấy tổng quát số người bị bệnh tim(1) và số người không bị bệnh tim(0) sau khi loại bỏ điểm ngoại lai vẫn khá cân bằng

- Ta trực quan hóa các dữ liệu rời rạc với biến mục tiêu sau khi tiền xử lý để xem sự tương quan giữa chúng:



Hình ảnh sau khi trực quan

- Ở thuộc tính Sex (Giới tính):

- + Tỷ lệ mắc bệnh của Nam bị mắc bệnh cao hơn nam không bị mắc bệnh
- + Tỷ lệ mắc bệnh của Nữ không bị mắc bệnh cao hơn nữ bị mắc bệnh.

- Ở thuộc tính ChestPainType (Loại đau ngực):

- + Tỷ lệ đau thắt ngực điển hình(ATA) của người không bị bệnh tim cao hơn người bị bệnh
- + Tỷ lệ đau không đau thắt ngực (NAP) của người không bị bệnh cao hơn người bị bệnh
- + Tỷ lệ không có triệu chứng (ASY) của người bị bệnh cao hơn người không bị bệnh
- + Tỷ lệ đau thắt ngực điển hình(TA) của người không bị bệnh cao hơn người bị bệnh.

- Ở thuộc tính RestingECG(Kết quả điện tâm đồ khi nghỉ):

- + Tỷ lệ bình thường(normal) của người bị bệnh cao hơn người không bị bệnh
- + Tỷ lệ có bất thường sóng ST-T (ST) của người bị bệnh cao hơn người không bị bệnh
- + Tỷ lệ cho thấy phì đại thất trái có thể xảy ra hoặc xác định theo tiêu chuẩn Estes (LVH) của người bị bệnh cao hơn người không bị bệnh

-Ở thuộc tính ExerciseAngina (Đau thắt ngực do gắng sức):

- +N(No) của người không bị bệnh lớn hơn người bị bệnh
- +Y(yes) của người bị bệnh lớn hơn người không bị bệnh

- Ở thuộc tính ST_Slope(Độ dốc của đoạn ST trong điện tâm đồ):

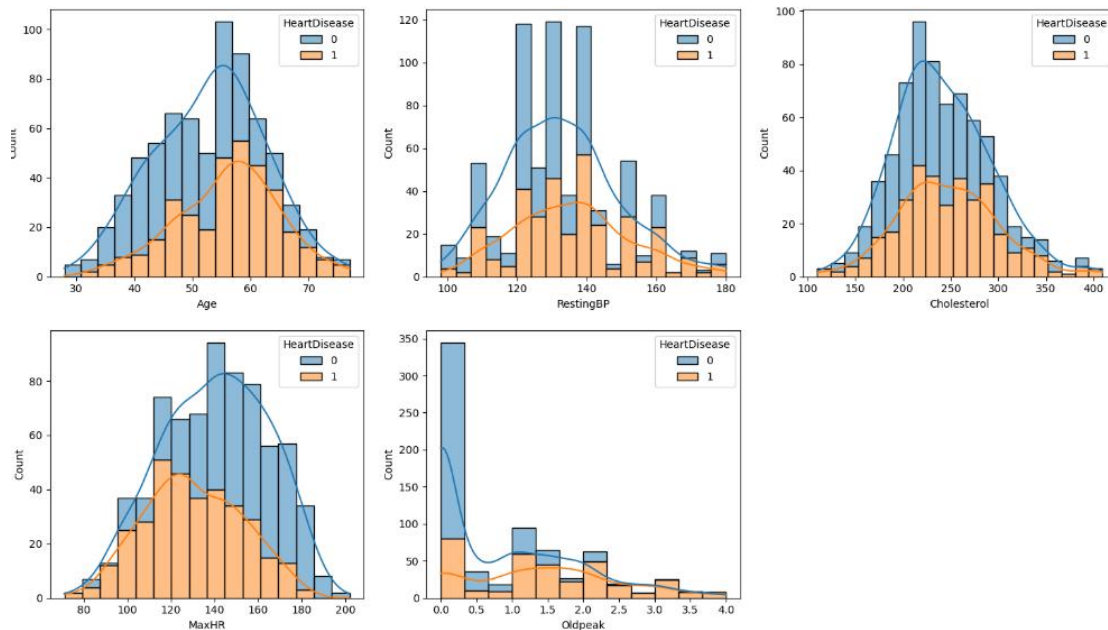
- + Tỷ lệ dốc lên (up) của người không bị bệnh cao hơn người bị bệnh
- + Tỷ lệ bằng phẳng (Flat) của người bị bệnh cao hơn người không bị bệnh
- + Tỷ lệ dốc xuống (Down) của người bị bệnh cao hơn người không bị bệnh

- Ở thuộc tính FastingBS(Chỉ số đường huyết lúc đói):

+ Tỷ lệ nếu chỉ số đường huyết lúc đói $>120\text{mg/dl}$ (1) của người không bị bệnh cao hơn người bị bệnh

+ Tỷ lệ ngược lại (0) của người bị bệnh cao hơn người không bị bệnh

- Tương tự ta trực quan hóa các dữ liệu liên tục với biểu đồ mục tiêu sau khi tiền xử lý:



Hình ảnh sau khi trực quan

- Ở thuộc tính Age những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 50 đến 60 tuổi.

- Ở thuộc tính RestingBP những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 110 đến 140

- Ở thuộc tính Cholesterol những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 200 đến 300

- Ở thuộc tính MaxHR những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 90 đến 150

- Ở thuộc tính Oldpeak những người bị bệnh và không bị bệnh phân bố nhiều nhất trong khoảng từ 0 đến 3

2. Biến đổi dữ liệu

```
columns = ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'ExerciseAngina', 'ST_Slope', 'HeartDisease']
for column in columns:
    print(column, ":", new_df[column].unique())
```

```
Sex : ['M' 'F']
ChestPainType : ['ATA' 'NAP' 'ASY' 'TA']
FastingBS : [0 1]
RestingECG : ['Normal' 'ST' 'LVH']
ExerciseAngina : ['N' 'Y']
ST_Slope : ['Up' 'Flat' 'Down']
HeartDisease : [0 1]
```

```
for column in columns:
    new_df[column] = new_df[column].astype('category')
```

- Ta thực hiện biến đổi dữ liệu các cột trong hình thành biến phân loại (category variable)

```
Index: 717 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  -
 0   Age             717 non-null   int64
 1   Sex             717 non-null   category
 2   ChestPainType   717 non-null   category
 3   RestingBP       717 non-null   int64
 4   Cholesterol     717 non-null   int64
 5   FastingBS       717 non-null   category
 6   RestingECG      717 non-null   category
 7   MaxHR           717 non-null   int64
 8   ExerciseAngina  717 non-null   category
 9   Oldpeak         717 non-null   float64
10  ST_Slope        717 non-null   category
11  HeartDisease    717 non-null   category
dtypes: category(7), float64(1), int64(4)
memory usage: 39.5 KB
```

```
new_df.describe(include=['category'])
```

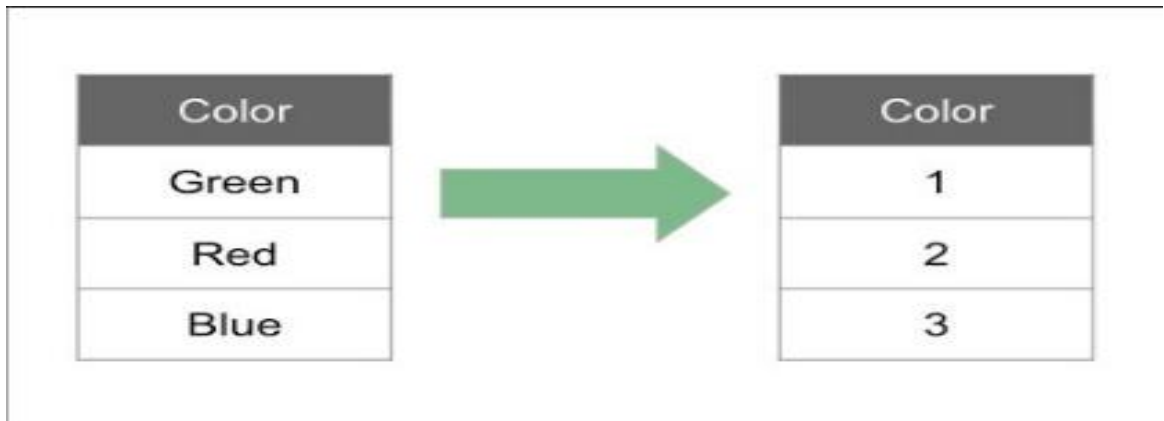
	Sex	ChestPainType	FastingBS	RestingECG	ExerciseAngina	ST_Slope	HeartDisease
count	717	717	717	717	717	717	717
unique	2	4	2	3	2	3	2
top	M	ASY	0	Normal	N	Flat	0
freq	541	352	598	427	443	343	378

Hình ảnh sau khi thực hiện

- Sau đó ta thực hiện mã hóa dữ liệu để máy có thể hiểu được

```
from sklearn.preprocessing import LabelEncoder
columns = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope', 'FastingBS', 'HeartDisease']
def encoder(columns):
    for column in columns:
        lbe = LabelEncoder()
        new_df[column] = lbe.fit_transform(new_df[column])
```

- Ta dùng phương pháp LabelEncoder để mã hóa dữ liệu



Hình ảnh ý tưởng của Label Encoder

```
encoder(columns)
for column in columns:
    print(column, ":", new_df[column].unique())
```

```
Sex : [1 0]
ChestPainType : [1 2 0 3]
RestingECG : [1 2 0]
ExerciseAngina : [0 1]
ST_Slope : [2 1 0]
FastingBS : [0 1]
HeartDisease : [0 1]
```

Hình ảnh sau khi thực hiện mã hóa dữ liệu

- Sau khi mã hóa dữ liệu ta thực hiện 1 bước nhỏ để chuyển các dữ liệu liên tục lên đầu và dữ liệu rời rạc về sau để chuẩn bị cho bước tiếp theo

```
columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak', 'Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope', 'FastingBS', 'HeartDisease']
result_df = pd.DataFrame(new_df, columns=columns)
```

```
result_df.head()
```

	Age	RestingBP	Cholesterol	MaxHR	Oldpeak	Sex	ChestPainType	RestingECG	ExerciseAngina	ST_Slope	FastingBS	HeartDisease
0	40	140	289	172	0.0	1	1	1	0	2	0	0
1	49	160	180	156	1.0	0	2	1	0	1	0	1
2	37	130	283	98	0.0	1	1	2	0	2	0	0
3	48	138	214	108	1.5	0	0	1	1	1	0	1
4	54	150	195	122	0.0	1	2	1	0	2	0	0

```
result_df.shape
```

```
(717, 12)
```

Hình ảnh sau khi thực hiện

```
result_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 717 entries, 0 to 917
Data columns (total 12 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Age              717 non-null    int64
1   RestingBP        717 non-null    int64
2   Cholesterol       717 non-null    int64
3   MaxHR            717 non-null    int64
4   Oldpeak          717 non-null    float64
5   Sex              717 non-null    int32
6   ChestPainType    717 non-null    int32
7   RestingECG       717 non-null    int32
8   ExerciseAngina   717 non-null    int32
9   ST_Slope         717 non-null    int32
10  FastingBS        717 non-null    int64
11  HeartDisease     717 non-null    int64
dtypes: float64(1), int32(5), int64(6)
```

Hình ảnh thông tin của dữ liệu sau khi mã hóa và chuyển thứ tự cột

=> Ta nhận thấy trong dữ liệu liên tục có kiểu dữ liệu của thuộc tính Oldpeak là float64, vậy nên ta thống nhất chuyển đổi các kiểu dữ liệu liên tục về thành 1 kiểu là float64

```
columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR']
for column in columns:
    result_df[column] = result_df[column].astype('float64')
```

```
result_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 717 entries, 0 to 917
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Age                   717 non-null   float64
1   RestingBP             717 non-null   float64
2   Cholesterol           717 non-null   float64
3   MaxHR                 717 non-null   float64
4   Oldpeak               717 non-null   float64
5   Sex                   717 non-null   int32
6   ChestPainType         717 non-null   int32
7   RestingECG            717 non-null   int32
8   ExerciseAngina        717 non-null   int32
9   ST_Slope              717 non-null   int32
10  FastingBS             717 non-null   int64
11  HeartDisease          717 non-null   int64
dtypes: float64(5), int32(5), int64(2)
memory usage: 58.8 KB
```

Hình ảnh sau khi thực hiện

3 .Phân tách dữ liệu và chuẩn hóa dữ liệu

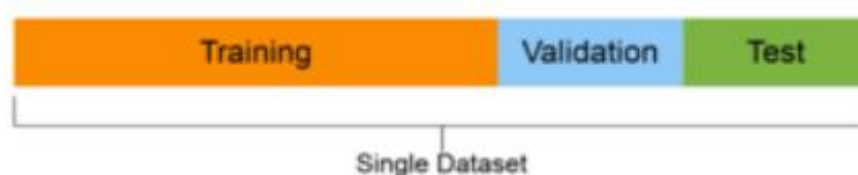
3.1 Phân tách dữ liệu

- Ta thực hiện tách dữ liệu thành 2 phần X và y
 - +> X gồm những cột dữ liệu chứa các thuộc tính
 - +> y là cột dữ liệu chứa biến mục tiêu

```
X = result_df.iloc[:, :-1]
y = result_df.iloc[:, -1]
```

Hình ảnh sau khi thực hiện

- Từ X và y ta thực hiện chia bộ dữ liệu thành 3 phần:
 - +> Training_set (60%)
 - +> Validation_set(20%)
 - +> Test_set (20%)




```

from sklearn.model_selection import train_test_split
X_train,X_val,y_train,y_val = train_test_split(X,y,test_size=0.4,random_state=42)
X_val,X_test,y_val,y_test = train_test_split(X_val,y_val,test_size=0.5,random_state=42)

X_train.shape,X_val.shape,X_test.shape

((430, 11), (143, 11), (144, 11))

```

Hình ảnh sau khi thực hiện

3.2 Chuẩn hóa dữ liệu cho từng tập

- Trong bài này ta sử dụng phương pháp chuẩn hóa là StandardScaler để chuẩn hóa dữ liệu cho các dữ liệu liên tục

$$z = \frac{x - \mu}{\sigma}$$

μ == Mean
 σ == Standard Deviation

Công thức của StandardScaler

- Chuẩn hóa dữ liệu cho Training_set

```

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train.iloc[:,0:5] = scaler.fit_transform(X_train.iloc[:,0:5])

X_train.head()

```

	Age	RestingBP	Cholesterol	MaxHR	Oldpeak	Sex	ChestPainType	RestingECG	ExerciseAngina	ST_Slope	FastingBS
133	0.338335	1.004959	-0.236517	-0.702177	0.675273	1	0	2	1	1	0
784	1.292678	0.276032	0.787795	1.318274	0.572633	1	3	0	0	1	1
432	1.080602	2.219838	-1.280527	-2.318538	1.701668	1	0	1	1	0	0
251	-0.509969	-0.817359	0.354432	-1.065858	1.188470	1	0	1	0	1	0
0	-1.358274	0.397520	0.925683	1.237456	-0.864320	1	1	1	0	2	0

Hình ảnh sau khi chuẩn hóa

- Chuẩn hóa dữ liệu cho Validation_set

```
X_val.iloc[:,0:5] = scaler.transform(X_val.iloc[:,0:5])
```

```
X_val.head()
```

	Age	RestingBP	Cholesterol	MaxHR	Oldpeak	Sex	ChestPainType	RestingECG	ExerciseAngina	ST_Slope	FastingBS
897	0.232297	-0.331408	-0.728975	-0.459723	1.18847	0	0	2	1	1	0
556	2.353059	1.612398	1.339347	-1.187085	1.18847	1	0	1	1	0	1
91	-1.464312	-0.209920	1.280252	-0.055633	-0.86432	1	0	1	0	2	0
116	-1.570350	-0.817359	0.787795	1.156638	-0.86432	1	0	1	0	1	0
910	-1.252236	-0.817359	-1.674493	1.641546	-0.86432	1	1	1	0	2	0

Hình ảnh sau khi chuẩn hóa

- Chuẩn hóa dữ liệu cho Test_set

```
X_test.iloc[:,0:5] = scaler.transform(X_test.iloc[:,0:5])
```

```
X_test.head()
```

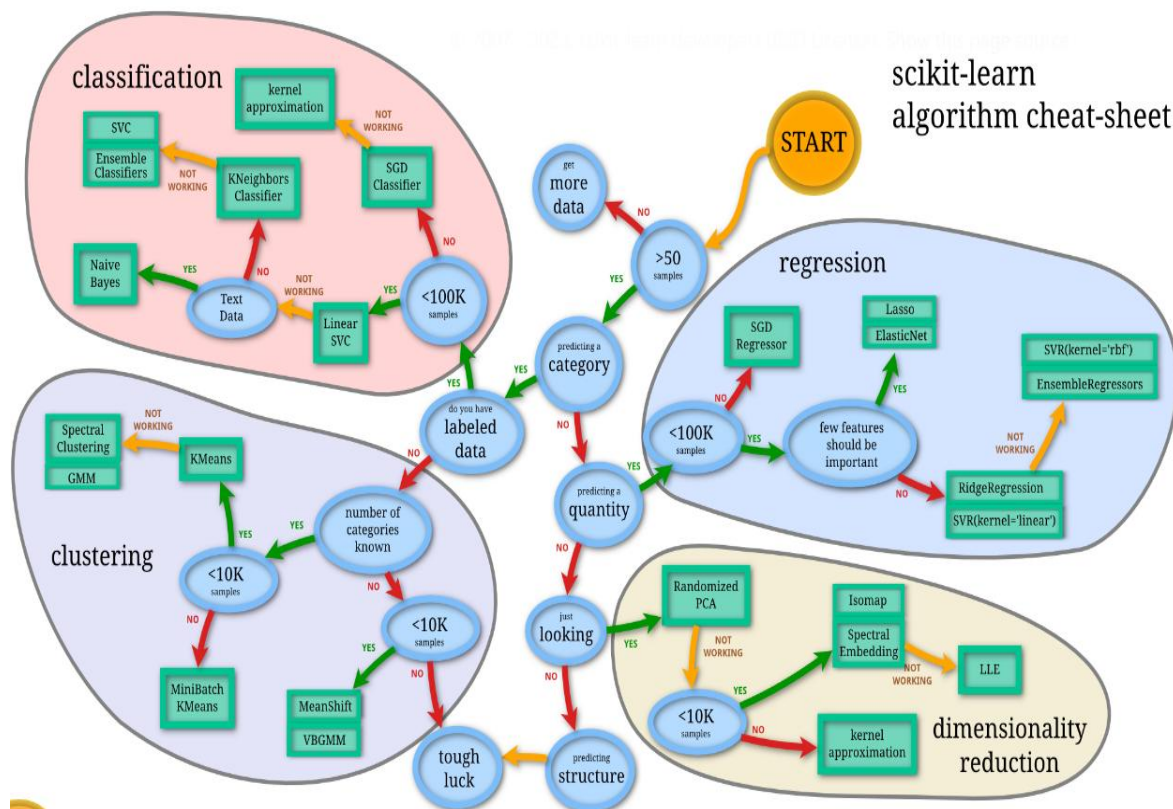
	Age	RestingBP	Cholesterol	MaxHR	Oldpeak	Sex	ChestPainType	RestingECG	ExerciseAngina	ST_Slope	FastingBS
854	-0.085817	-0.817359	1.634822	1.237456	-0.659041	1	1	1	0	2	0
35	-2.206579	-0.513639	0.236242	0.550502	-0.864320	1	1	1	0	2	0
58	0.126259	1.004959	2.422754	-0.298087	0.162075	1	0	2	0	2	0
234	0.126259	0.397520	1.319649	-0.055633	-0.864320	0	1	2	0	2	0
490	2.034945	-0.817359	-0.551690	-1.591176	0.162075	1	2	1	1	1	0

Hình ảnh sau khi chuẩn hóa

CHƯƠNG III. LỰA CHỌN PHƯƠNG PHÁP PHÂN LỚP VÀ ĐÁNH GIÁ

1. Trainng mô hình từ bộ dữ liệu đã phân tách

Để đánh giá 1 cách khách quan ta nên dựa vào nhiều mô hình để đánh giá và cách lựa chọn mô hình ta dựa vào trang chủ của SciKit-learn



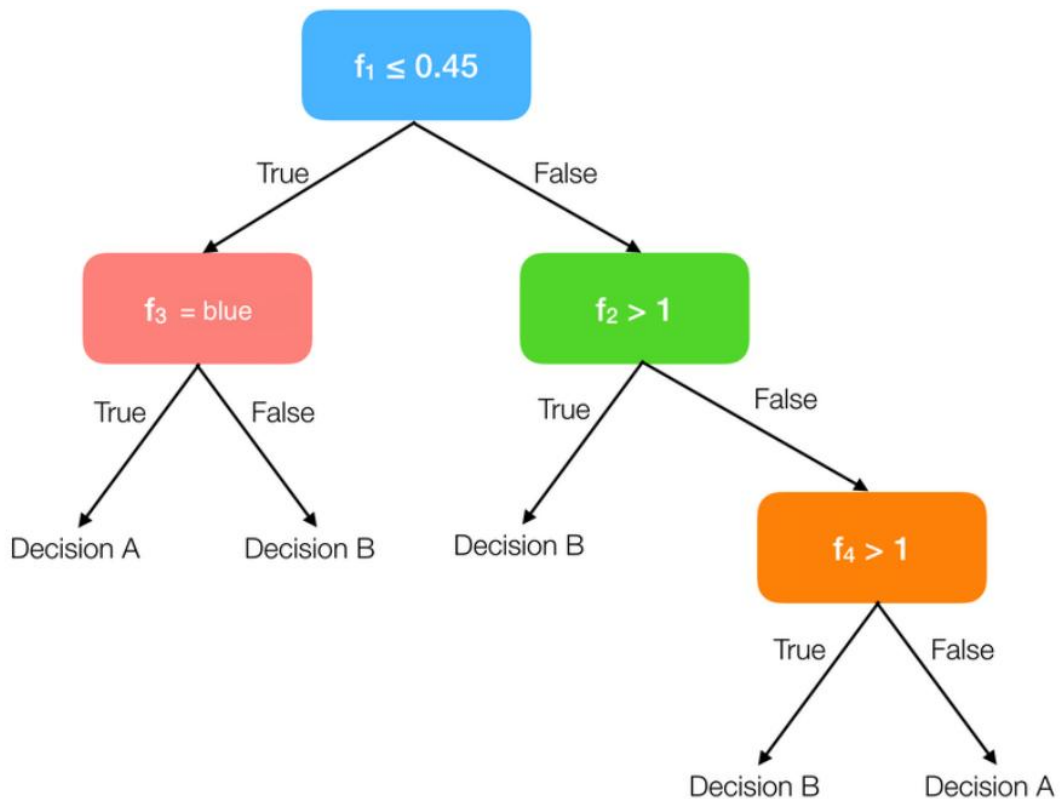
Hình ảnh minh họa cách lựa chọn

- Từ hình trên ta lựa chọn được những mô hình là:

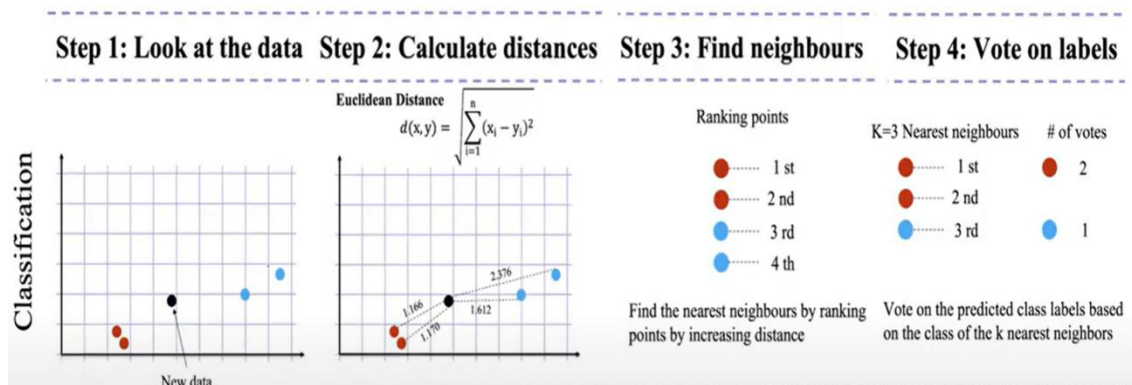
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import accuracy_score, classification_report
```

Hình ảnh sau khi chọn lựa mô hình

- ý tưởng của DecisionTreeClassifier

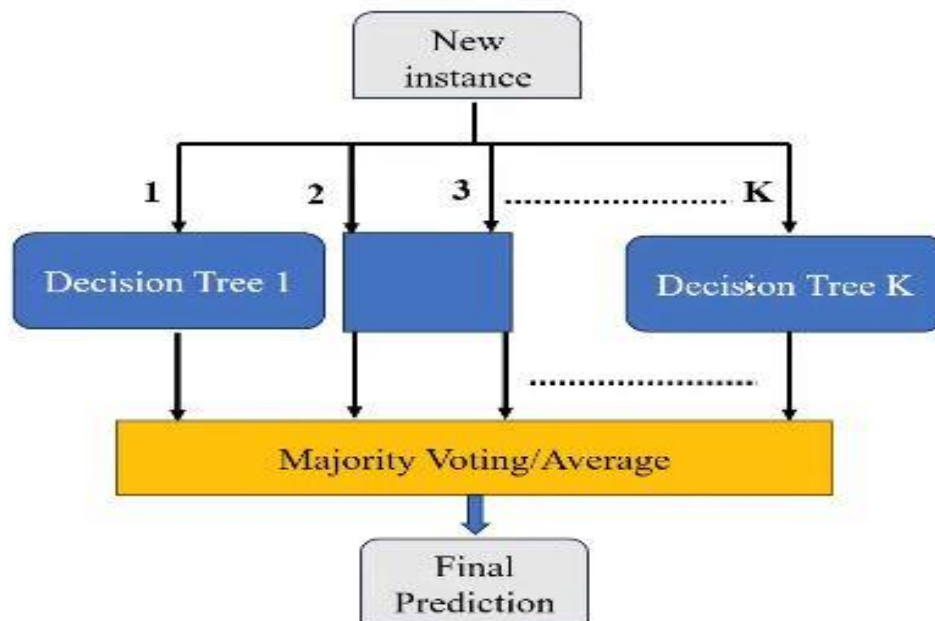


- Ý tưởng của KNN



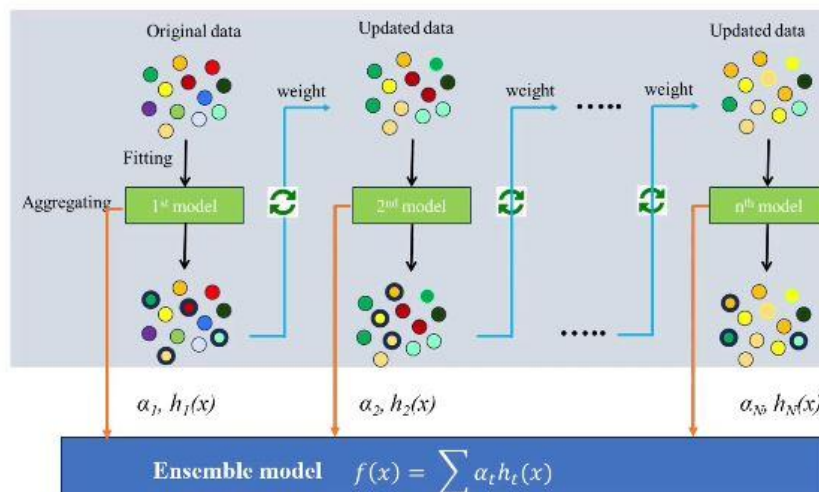
- Ý tưởng của RandomForestClassifier

Prediction of RF



- Ý tưởng của AdaboostClassifier, GradientBoostingClassifier

AdaBoost



Ensemble model

$$f(x) = \sum_t \alpha_t h_t(x)$$

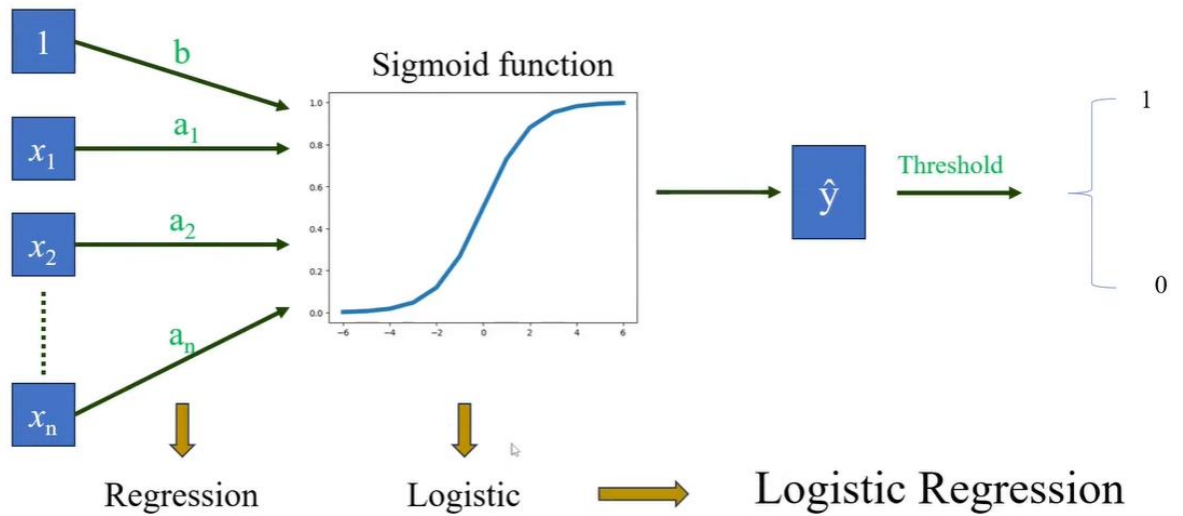
↓
Stronger learner

Weak learners → Weights

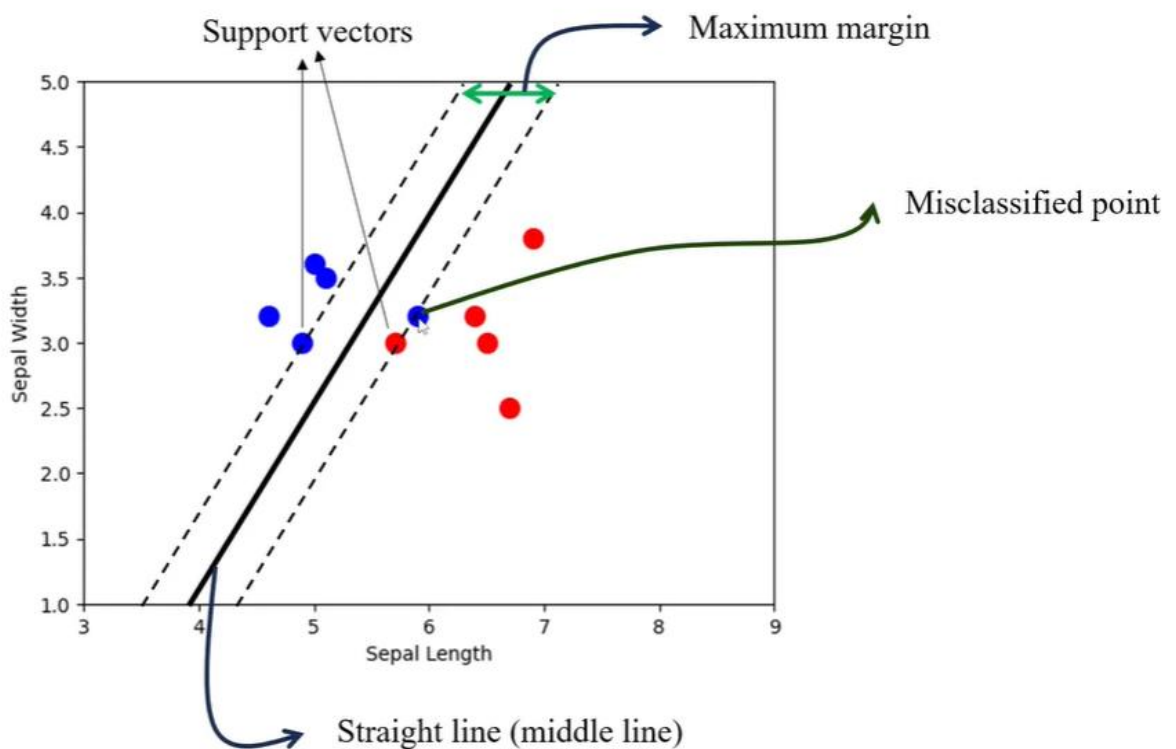
- Ý tưởng của LogisticRegression

Features

Output



- Ý tưởng của SVC, LinearSVC



- Ta thực hiện huấn luyện với các mô hình trên

```
seed = 2024
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=seed),
    "KNN": KNeighborsClassifier(metric='minkowski', p=2),
    "LogisticRegression": LogisticRegression(solver='liblinear', max_iter=1200, random_state=seed),
    "SVM": SVC(random_state=seed),
    "LinearSVC": LinearSVC(max_iter=12000, random_state=seed),
    "RandomForest": RandomForestClassifier(random_state=seed),
    "AdaBoostClassifier": AdaBoostClassifier(random_state=seed),
    "GradientBoostingClassifier": GradientBoostingClassifier(random_state=seed)
}
for name, model in models.items():
    print(f"Training model {name}...")
    model = model.fit(X_train, y_train)
    # y_pred = model.predict(X_test)
    y_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)
    print(f"\t{name} accuracy_score in train set: {model.score(X_train, y_train)}\n")
    print(f"\t{name} accuracy_score in val set: {accuracy}\n")
    print(f"\t Classification report {name}")
    print(classification_report(y_val, y_pred))
    print("*"*60)
```

Hình ảnh thực hiện

- Sau khi thực hiện Train mô hình và đánh giá trên Validation_set ta được kết quả với từng mô hình tương ứng:
- Ta đánh giá mô hình theo ma trận nhầm lẫn:

Confusion Matrix (Ma trận nhầm lẫn)

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP Type 1 Error	FP Type 1 Error
	Negative (0)	FN Type 2 Error	TN

TP: True positive (dương tính thật)
FP: False positive (dương tính giả)
TN: True negative (âm tính thật)
FN: False negative (âm tính giả)

Email is spam (**positive**, 1) or not spam (**negative**, 0)

Email is spam (positive, 1)

- Predicted result is spam : TP
- Predicted result is NOT spam: FP

Email is NOT spam (negative, 0)

- Predicted result is spam : FN
- Predicted result is NOT spam: TN

Hình ảnh của ma trận nhầm lẫn

-Với các thang đo là:

+ **Precision**: Tỷ lệ số lượng các mẫu được phân loại chính xác vào nhãn Positive so với tổng số các mẫu được phân loại vào nhãn Positive.

Precision

$$Precision = \frac{TP}{TP + FP}$$



Actual Positive
Total predicted positive

Detection of email spam

-  Predicted to be not spam
-  Predicted to be spam
-  Not spam (true)
-  Spam (true)

Predicted Values
Positive (1)
Negative (0)

Actual Values	
Positive (1)	Negative (0)
Predicted Values Positive (1)	TP Type 1 Error
Predicted Values Negative (0)	FN Type 2 Error
	TN



Precision is used when **False positive** is more important than **False negative** (e-commerce website, high quality products,...) **We want to minimize FP - Higher Precision, Better Model**

Hình ảnh minh họa

+ **Recall**: Tỷ lệ số lượng các mẫu được phân loại chính xác vào nhãn Positive so với tổng số mẫu Positive trong tập dữ liệu.

Recall (sensitivity)

$$Recall = \frac{TP}{TP + FN}$$



True Positive
Total actual positive

Actual Values	
Positive (1)	Negative (0)
Predicted Values Positive (1)	TP
Predicted Values Negative (0)	FN Type 2 Error
	TN

TP: True positive (dương tính thật)
FP: False positive (dương tính giả)
TN: True negative (âm tính thật)
FN: False negative (âm tính giả)

FN is more important than FP

We want to minimize FN

Hình ảnh minh họa

+ **F-Measure:** Kết hợp giữa Precision và Recall để đánh giá hiệu quả phân loại.

F-Measure càng lớn thì phân loại càng chính xác.

F1 Score

$$F1\ score = 2 \frac{Precision * Recall}{Precision + Recall}$$

- F1 score is considered as harmonic mean of precision and recall.
- Thus F1 score can be used well for:
 - ✓ Both **balanced** and **imbalanced** data
 - ✓ Both recall and precision are equally important
 - ✓ When we want to use single metric
 - ✓ When model selection and tuning
- F1 score depends on data and problem to predict

General rule of thumb of F1 score

- 0.9: very good
- 0.8 – 0.9: Good
- 0.5 – 0.8: Accept
- < 0.5: Not good

Training model Decision Tree...

Decision Tree accuracy_score in train set: 1.0

Decision Tree accuracy_score in val set: 0.7902097902097902

Classification report Decision Tree

	precision	recall	f1-score	support
0	0.82	0.78	0.80	78
1	0.75	0.80	0.78	65
accuracy			0.79	143
macro avg	0.79	0.79	0.79	143
weighted avg	0.79	0.79	0.79	143

Training model KNN...

KNN accuracy_score in train set: 0.8651162790697674

KNN accuracy_score in val set: 0.8041958041958042

Classification report KNN

	precision	recall	f1-score	support
0	0.80	0.86	0.83	78
1	0.81	0.74	0.77	65
accuracy			0.80	143
macro avg	0.81	0.80	0.80	143
weighted avg	0.80	0.80	0.80	143

```

Training model LogisticRegression...
LogisticRegression accuracy_score in train set: 0.8534883720930233

LogisticRegression accuracy_score in val set: 0.8321678321678322

Classification report LogisticRegression
      precision    recall  f1-score   support

     0       0.88       0.81       0.84         78
     1       0.79       0.86       0.82         65

 accuracy          0.83         143
 macro avg       0.83       0.83       0.83         143
weighted avg       0.84       0.83       0.83         143

```

```

Training model SVM...
SVM accuracy_score in train set: 0.8813953488372093

SVM accuracy_score in val set: 0.8391608391608392

```

```

Classification report SVM
      precision    recall  f1-score   support

     0       0.88       0.82       0.85         78
     1       0.80       0.86       0.83         65

 accuracy          0.84         143
 macro avg       0.84       0.84       0.84         143
weighted avg       0.84       0.84       0.84         143

```

```

Training model LinearSVC...
LinearSVC accuracy_score in train set: 0.8511627906976744

LinearSVC accuracy_score in val set: 0.8321678321678322

```

```

Classification report LinearSVC
      precision    recall  f1-score   support

     0       0.88       0.81       0.84         78
     1       0.79       0.86       0.82         65

 accuracy          0.83         143
 macro avg       0.83       0.83       0.83         143
weighted avg       0.84       0.83       0.83         143

```

```

Training model RandomForest...
RandomForest accuracy_score in train set: 1.0

RandomForest accuracy_score in val set: 0.8671328671328671

```

```

Classification report RandomForest
      precision    recall  f1-score   support

     0       0.92       0.83       0.87         78
     1       0.82       0.91       0.86         65

 accuracy          0.87         143
 macro avg       0.87       0.87       0.87         143
weighted avg       0.87       0.87       0.87         143

```

```

Training model AdaBoostClassifier...
AdaBoostClassifier accuracy_score in train set: 0.8953488372093024

AdaBoostClassifier accuracy_score in val set: 0.8601398601398601

Classification report AdaBoostClassifier
precision    recall  f1-score   support

     0       0.88       0.86       0.87        78
     1       0.84       0.86       0.85        65

 accuracy          0.86          0.86          0.86          143
  macro avg       0.86          0.86          0.86          143
 weighted avg     0.86          0.86          0.86          143

*****
Training model GradientBoostingClassifier...
GradientBoostingClassifier accuracy_score in train set: 0.9674418604651163

GradientBoostingClassifier accuracy_score in val set: 0.8741258741258742

Classification report GradientBoostingClassifier
precision    recall  f1-score   support

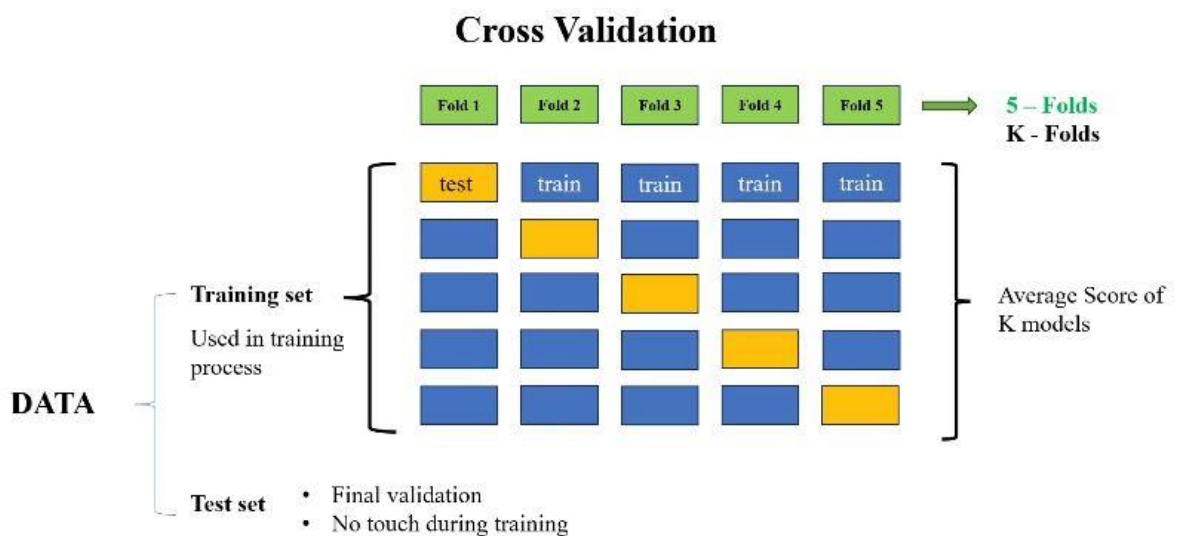
     0       0.89       0.87       0.88        78
     1       0.85       0.88       0.86        65

 accuracy          0.87          0.87          0.87          143
  macro avg       0.87          0.87          0.87          143
 weighted avg     0.87          0.87          0.87          143

*****

```

2. Train mô hình với Cross-validate(đánh giá chéo)



Hình ảnh của phương pháp đánh giá chéo

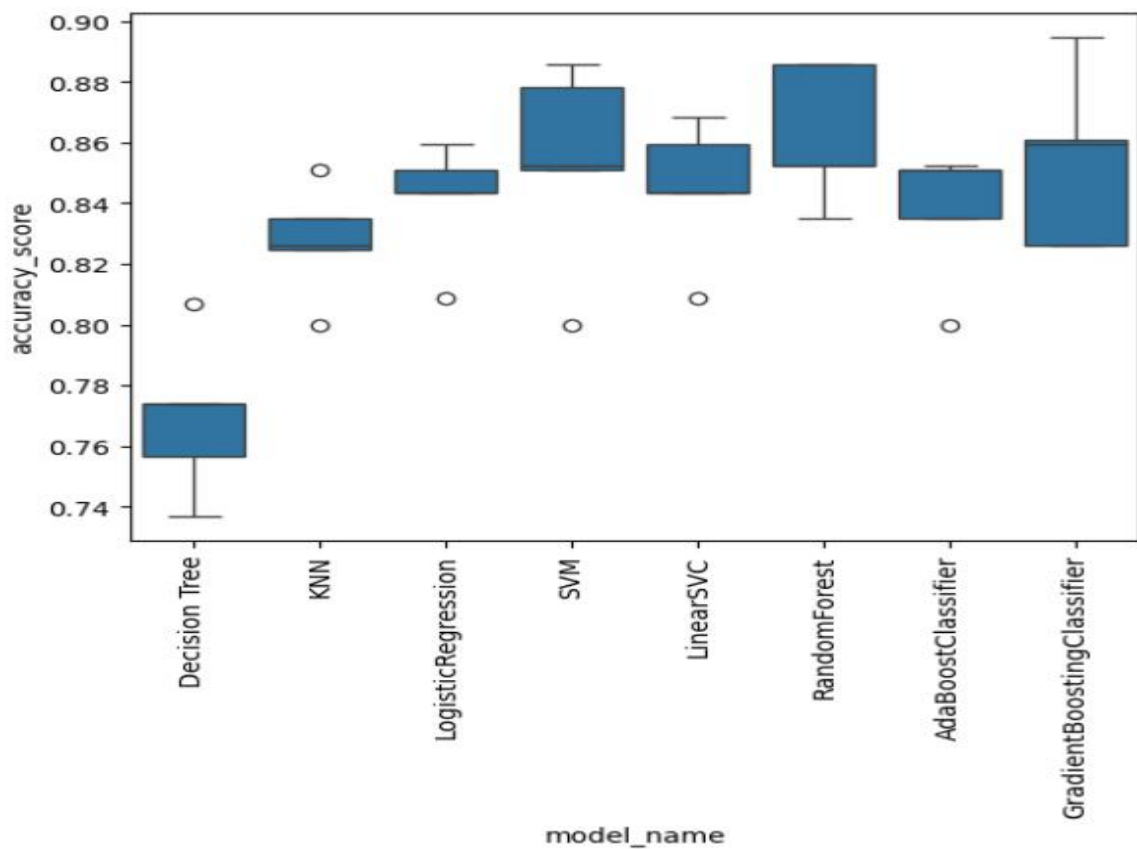
```

from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
def models_results(models,X_train,y_train,X_val,y_val,metrics,cv=5,plot_results=False):
    k_fold = StratifiedKFold(cv,shuffle=True,random_state=seed)
    entries = []
    X = pd.concat([X_train,X_val],axis=0,ignore_index=True)
    y = pd.concat([y_train,y_val],axis=0,ignore_index=True)
    for name,model in models.items():
        scores = cross_val_score(model,X,y,scoring=metrics,cv=k_fold)
        for fold_idx,score in enumerate(scores):
            entries.append((name,fold_idx,score))
    cv_df = pd.DataFrame(entries,columns=['model_name','fold_id','accuracy_score'])

    if plot_results:
        sns.boxplot(x='model_name',y='accuracy_score',data=cv_df)
        plt.xticks(rotation=90)
        plt.show()
    mean = cv_df.groupby('model_name')['accuracy_score'].mean()
    std = cv_df.groupby('model_name')['accuracy_score'].std()
    models_accuracy_result = pd.concat([mean,std],axis=1)
    models_accuracy_result.columns = ['Mean','Std']
    models_accuracy_result.sort_values(by=['Mean'],ascending=False,inplace=True)
    return models_accuracy_result
models_results(models,X_train,y_train,X_val,y_val,metrics='accuracy',cv=5,plot_results=True)

```

Hình ảnh thực hiện



Hình ảnh độ chính xác của từng phương pháp khi thực hiện đánh giá chéo

	Mean	Std
model_name		
RandomForest	0.862212	0.022816
GradientBoostingClassifier	0.853486	0.028703
SVM	0.853455	0.033682
LinearSVC	0.844744	0.022833
LogisticRegression	0.841236	0.019371
AdaBoostClassifier	0.837742	0.022285
KNN	0.827262	0.018481
Decision Tree	0.769641	0.025891

Hình ảnh giá trị chính xác trung bình và độ lệch chuẩn của từng phương pháp

3. Tối ưu hóa mô hình và đánh giá

- Sau khi thực hiện Training mô hình bằng 2 cách ta thấy mô hình RandomForestClassifier có độ chính xác trung bình cao nhất là: 86.22%

- Cách cài đặt mô hình RandomForestClassifier ở trong file:

[BuildRandomForestClassifier.ipynb](#)

=> Ta lựa chọn mô hình RandomForest làm mô hình cuối để dự đoán Test_set

- Ta thực hiện đánh giá lại mô hình RandomForestClassifier:

```
from sklearn.model_selection import GridSearchCV

model = RandomForestClassifier(random_state=seed)

model.fit(X_train,y_train)
y_pred_train = model.predict(X_train)
accuracy_train_set = accuracy_score(y_train,y_pred_train)
print(f"\t{name} accuracy_score in Train set: {accuracy_train_set}\n")
y_pred_val = model.predict(X_val)
accuracy_val_set = accuracy_score(y_val,y_pred_val)
print(f"\t{name} accuracy_score in Val set: {accuracy_val_set}\n")
print(classification_report(y_val,y_pred_val))
```

GradientBoostingClassifier accuracy_score in Train set: 1.0

GradientBoostingClassifier accuracy_score in Val set: 0.8671328671328671

	precision	recall	f1-score	support
0	0.92	0.83	0.87	78
1	0.82	0.91	0.86	65
accuracy			0.87	143
macro avg	0.87	0.87	0.87	143
weighted avg	0.87	0.87	0.87	143

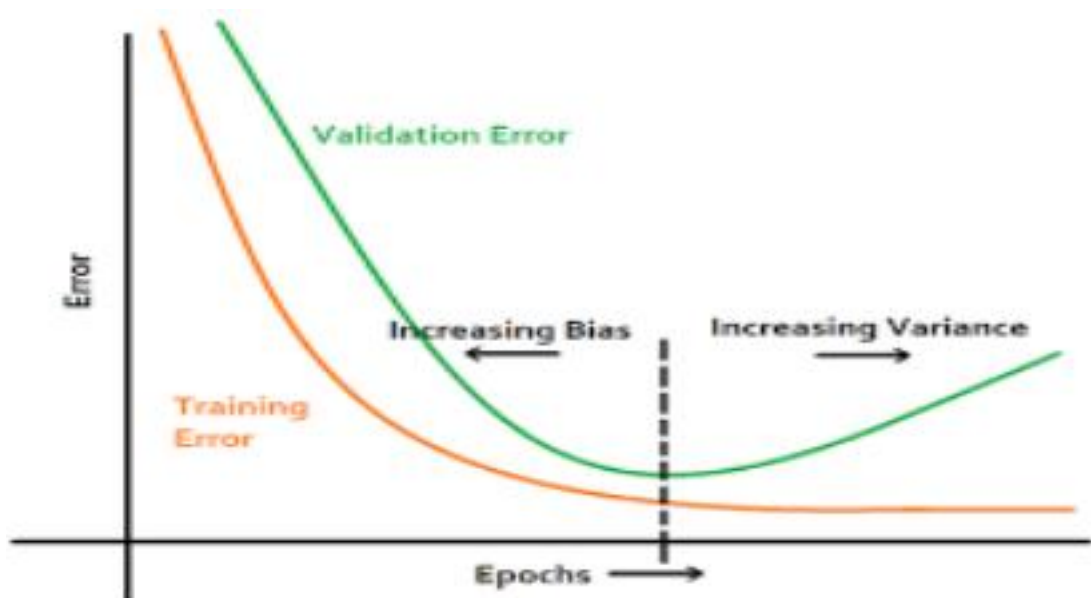
Hình ảnh sau khi thực hiện

- Sau khi thực hiện đánh giá lại mô hình ta thấy độ chính xác trong tập Training là 100% nhưng khi đánh giá lại với tập Validation_set lại là 86,7%

=> Mô hình đang bị overfitting (Lỗi quá khớp)

Overfitting and Underfitting

Overfitting	Underfitting
<ul style="list-style-type: none">• Mô hình quá phức tạp• Sai số thấp trên dữ liệu huấn luyện• Sai số cao trên dữ liệu kiểm tra (test set)	<ul style="list-style-type: none">• Mô hình quá đơn giản• Sai số cao trên dữ liệu huấn luyện (training set)• Khả năng dự đoán kém



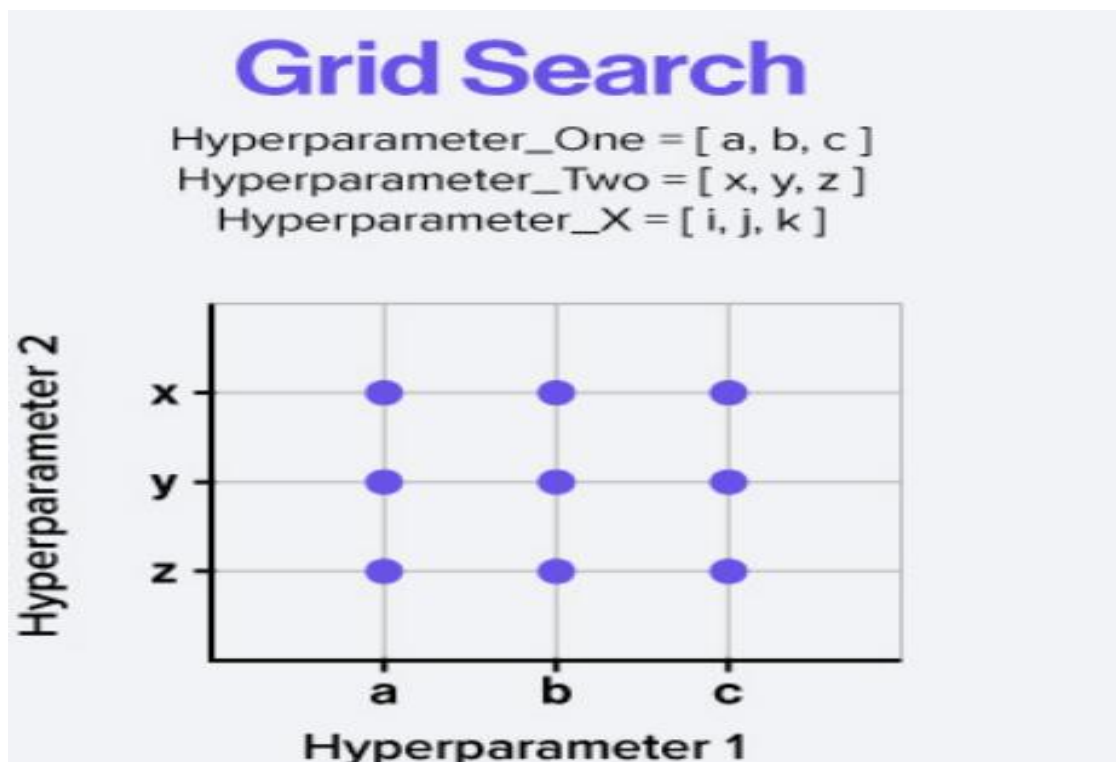
=> Vậy nên ta cần điều chỉnh lại tham số của mô hình từ đó tìm ra siêu tham số (hyperparameter) để giúp mô hình tránh bị overfitting và underfitting giúp mô hình đánh giá đúng khi đưa vào ứng dụng

- Ta tạo ra những tham số đầu vào cho mô hình

```
model2 = RandomForestClassifier(random_state=seed)
param_grid = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [None, 10, 20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

Hình ảnh sau khi thực hiện

- Với các tham số là:
 - +> n_estimators: Số lượng cây
 - +> max_depth: Chiều sâu của cây
 - +> min_sample_split: số lượng mẫu tối thiểu để chia node
 - +> min_sample_leaf: số lượng mẫu tối thiểu ở mỗi lá
- Sau đó ta sử dụng GridSearchCV để tìm ra siêu tham số:



Hình ảnh minh họa

- Các tham số đầu vào của phương pháp GridSearchCV gồm:

- +> estimator: mô hình cần đánh giá
- +> param_grid: lưới tham số (đã được khai báo ở trên)
- +> scoring: đánh giá theo 'accuracy' (độ chính xác)
- +> cv: Số lần đánh giá
- +> n_job = -1: Sử dụng các CPU Core
- +> verbose = 1: hiển thị tiến trình

- Sau khi tạo ra các tham số đầu vào cho GridSearchCV ta sẽ dùng nó để training mô hình với training_set để tìm ra siêu tham số và dùng nó để dự đoán ở tập validation_set

```
grid_search = GridSearchCV(  
    estimator=model2,  
    param_grid=param_grid,  
    scoring='accuracy',  
    cv=5,  
    n_jobs=-1,  
    verbose=1  
)  
grid_search.fit(X_train, y_train)  
  
print("Best parameters found: ", grid_search.best_params_)  
  
print("Best cross-validation accuracy: {:.2f}".format(grid_search.best_score_))  
  
best_model = grid_search.best_estimator_  
val_accuracy = best_model.score(X_val, y_val)  
print("Val accuracy: {:.2f}".format(val_accuracy))  
  
Fitting 5 folds for each of 180 candidates, totalling 900 fits  
Best parameters found: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500}  
Best cross-validation accuracy: 0.87  
Val accuracy: 0.87  
  
y_val_pred = best_model.predict(X_val)  
  
print(classification_report(y_val_pred, y_val))
```

	precision	recall	f1-score	support
0	0.85	0.92	0.88	72
1	0.91	0.83	0.87	71
accuracy			0.87	143
macro avg	0.88	0.87	0.87	143
weighted avg	0.88	0.87	0.87	143

Hình ảnh thực hiện

- Sau khi thực hiện ta được 1 bộ siêu tham số gồm:

`{'max_depth':None,'min_samples_leaf':1,'min_samples_split':2,'n_estimators': 500}`

=> Ta thấy độ chính xác của 2 tập training_set và validation_set đều bằng nhau => Mô hình đã chạy ổn định. Vậy nên ta chọn bộ siêu tham số trên để làm siêu tham số cho mô hình

- Đánh giá mô hình trên tập Test_set

```
test_accuracy = best_model.score(X_test, y_test)
print("Test accuracy: {:.2f}".format(test_accuracy))
```

Test accuracy: 0.87

```
y_test_pred = best_model.predict(X_test)
```

```
print(classification_report(y_test_pred,y_test))
```

	precision	recall	f1-score	support
0	0.82	0.91	0.86	65
1	0.92	0.84	0.87	79
accuracy			0.87	144
macro avg	0.87	0.87	0.87	144
weighted avg	0.87	0.87	0.87	144

```
compare_df = pd.DataFrame({"y_test":y_test,"y_test_pred":y_test_pred})
```

```
compare_df.head(10)
```

	y_test	y_test_pred
854	0	0
35	0	0
58	0	0
234	0	0
490	1	1
494	1	1
716	1	1
595	1	1
725	1	1
660	0	0

Hình ảnh thực hiện

=> Ta nhận thấy khi thực hiện dự đoán với 1 bộ dữ liệu mới độ chính xác vẫn là 87% => mô hình đã chạy ổn định

=> Đánh giá chung:

- Với bài toán dự đoán bệnh suy tim việc sử dụng mô hình RandomForestClassifier là phù hợp vì:

- + Giảm thiểu overfitting
- + Độ chính xác khi dự đoán cao
- + Xử lý tốt giá trị ngoại lai
- + Xử lý tốt với dữ liệu lớn và nhiều chiều

- Bên cạnh đó mô hình còn 1 số nhược điểm như:

- + Đòi hỏi nhiều tài nguyên chạy máy
- + Độ phức tạp trong việc tính toán

CHƯƠNG IV. ỨNG DỤNG MÔ HÌNH

- Phần này ta sẽ đưa mô hình vào chạy thành 1 ứng dụng hoàn chỉnh

- Đầu tiên ta sẽ lưu mô hình ra 1 tệp có tên là: 'model_randomforest.pkl'

```
import joblib

joblib.dump(best_model, 'model_randomforest.pkl')

['model_randomforest.pkl']
```

Hình ảnh thực hiện

- Sau đó ta tạo ra các file app.py(Tạo ứng dụng), index.html và style.css (Giao diện người dùng) => Xem kỹ các file trong thư mục đã tải về

- Để chạy ứng dụng, ta sẽ truy cập đến thư mục có chứa file bài tập lớn

```
C:\>cd C:\Users\DUC-PC\BTLKDPL
```

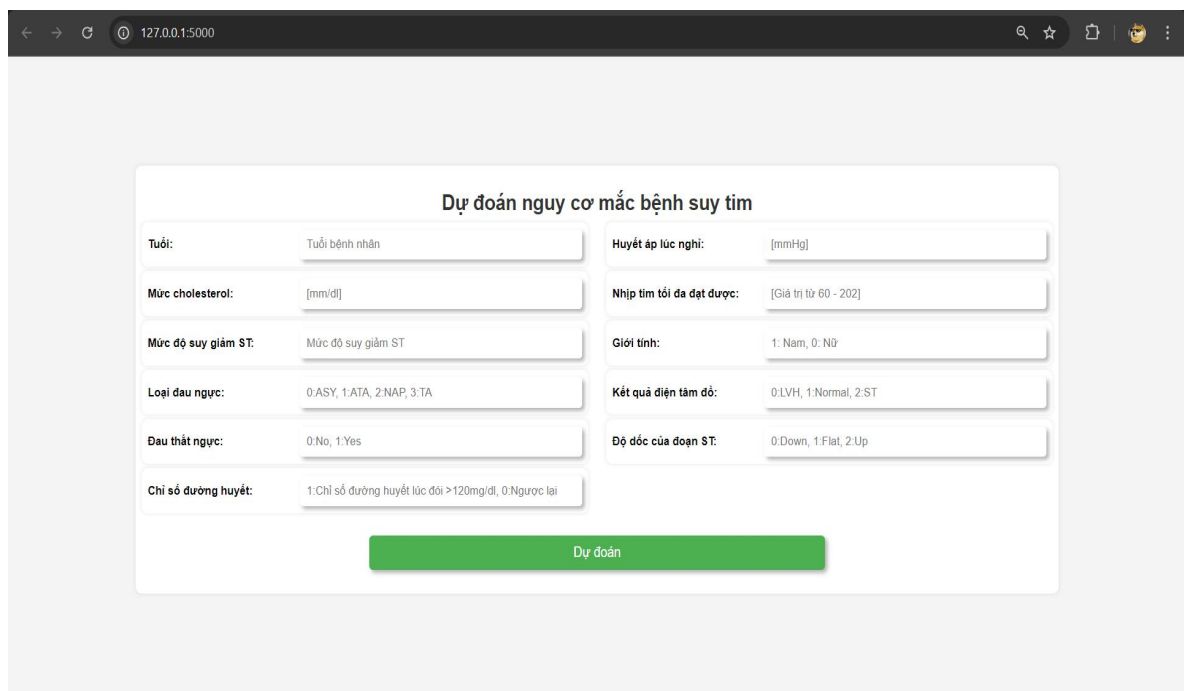
- Sau đó ta sẽ sử dụng câu lệnh dưới để chạy ứng dụng:

```
C:\Users\DUC-PC\BTLKDPL>python app.py
```

- Khi chạy câu lệnh ta được hình dưới đây:

```
C:\Users\DUC-PC\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator DecisionTreeClassifier from version 1.4.2 when using version 1.5.2. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to: https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
warnings.warn(
C:\Users\DUC-PC\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator RandomForestClassifier from version 1.4.2 when using version 1.5.2. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to: https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
warnings.warn(
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

- Sau đó ta truy cập vào đường dẫn <http://127.0.0.1:5000> để đi đến ứng dụng



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000'. The main content area features a form titled 'Dự đoán nguy cơ mắc bệnh suy tim' (Predict the risk of heart failure). The form contains several input fields for user data, organized in two columns. Below the input fields is a green button labeled 'Dự đoán' (Predict).

Dự đoán nguy cơ mắc bệnh suy tim	
Tuổi:	Tuổi bệnh nhân
Mức cholesterol:	[mm/dl]
Mức độ suy giảm ST:	Mức độ suy giảm ST
Loại đau ngực:	0:ASY, 1:ATA, 2:NAP, 3:TA
Đau thắt ngực:	0:No, 1:Yes
Chỉ số đường huyết:	1:Chỉ số đường huyết lúc đói >120mg/dl, 0:Ngược lại
Huyết áp lúc nghỉ:	[mmHg]
Nhịp tim tối đa đạt được:	[Giá trị từ 60 - 202]
Giới tính:	1: Nam, 0: Nữ
Kết quả điện tâm đồ:	0:LVH, 1:Normal, 2:ST
Độ dốc của đoạn ST:	0:Down, 1:Flat, 2:Up

Dự đoán

Hình ảnh sau khi truy cập đến đường dẫn

- Sau đó ta nhập dữ liệu để đưa ra kết quả

KẾT LUẬN

Trong quá trình thực hiện bài tập lớn về khai phá dữ liệu để dự đoán bệnh suy tim, nhóm chúng em đã áp dụng các bước cơ bản trong quy trình khai phá dữ liệu, bao gồm thu thập, tiền xử lý, và phân tích dữ liệu.

Việc lựa chọn mô hình RandomForestClassifier cho thấy kết quả tốt nhất với độ chính xác cao, mặc dù vẫn còn một số điểm cần tối ưu hóa để tránh tình trạng quá khớp. Bên cạnh đó, việc xử lý các dữ liệu ngoại lai và chuẩn hóa dữ liệu đã giúp cải thiện chất lượng đầu vào, tạo điều kiện cho các mô hình học máy hoạt động hiệu quả hơn.

Nhóm chúng em nhận thấy rằng việc áp dụng các phương pháp khai phá dữ liệu trong lĩnh vực y tế, cụ thể là dự đoán bệnh suy tim, có thể mang lại lợi ích lớn cho việc phát hiện và điều trị sớm các bệnh lý liên quan đến tim mạch. Tuy nhiên, để đạt được kết quả tốt nhất trong thực tế, cần có thêm dữ liệu phong phú và chính xác hơn cũng như các công cụ và phương pháp cải tiến để phát triển mô hình dự đoán tối ưu hơn.

TÀI LIỆU THAM KHẢO

1. Han, J., Pei, J., & Kamber, M. (2011). *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann.
2. Quinlan, J. R. (1986). *Induction of decision trees*. Machine learning, 1(1), 81-106.
3. Breiman, L. (2001). *Random forests*. Machine learning, 45(1), 5-32.
4. <https://openscience.vn/chi-tiet-du-lieu/bo-du-lieu-du-doan-suy-tim->
5. Pedregosa, F., et al. (2011). *Scikit-learn: Machine learning in Python*. Journal of machine learning research, 12, 2825-2830.
6. <https://scikit-learn.org/>
7. <https://d2l.aivivn.com/>