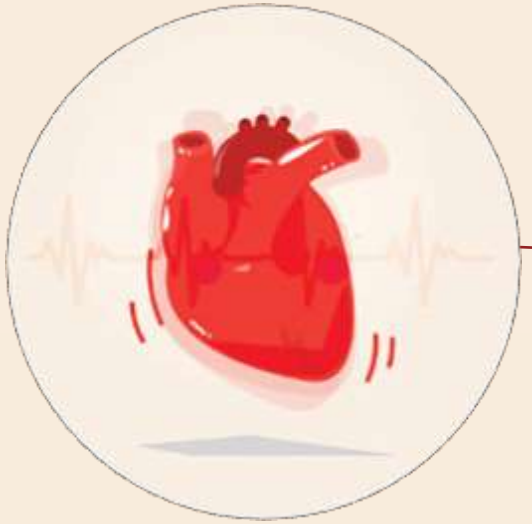
心臟病預測

黃柏禎

什麼是心臟病



心臟是人體中一個很特殊的器官，它能夠自動地收縮、跳動，目的只有一個，那就是無時無刻輸送血液到身體各器官組織，就像扮演著汽車中的靈魂人物「馬達」。當心臟中各個結構發生問題時，心臟無法發揮正常功能，人體就會出現相關症狀，就是我們所謂的「心臟病」。

專案目標

本專案的目標是開發一個模型，以預測患者是否患有心臟病。這個模型基於病人的生理數據，包括年齡、性別、胸痛類型、靜息心率、血壓、膽固醇中的各種生化指標等。通過使用機器學習技術，我們的目標是建立一個高度準確的模型，以預測患有心臟病的風險。這將有助於醫療保健專業人員更好的了解患者的狀況，制定更好的治療方案，減少患者病情的惡化，提高醫療保健效率。

資料讀取

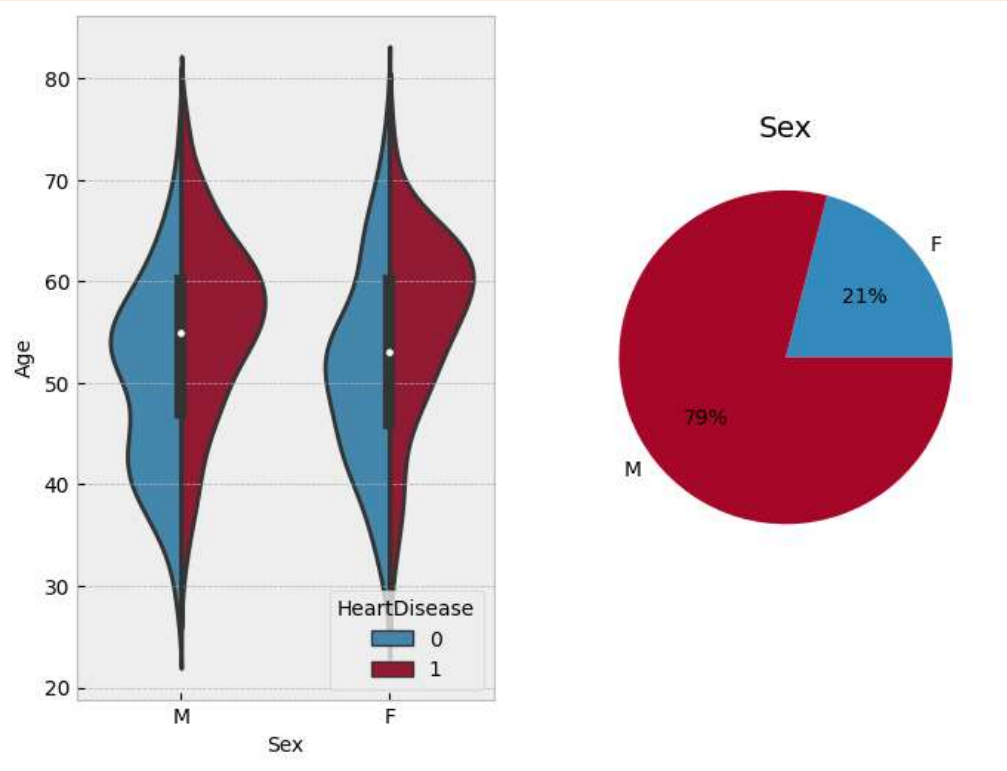
# Age	患者年齡[歲]
# Sex	患者性別[M=男, F=女]
# ChestPainType	胸痛類型【TA=典型心絞痛, ATA=非典型心絞痛, NAP=非心絞痛, ASY=無症狀】
# RestingBP	靜息血壓 [mm Hg]
# Cholesterol	血清膽固醇 [mm/dl]
# FastingBS	空腹血糖 [if FastingBS > 120 mg/dl, else 0]
# RestingECG	靜息心電圖結果[正常：正常，ST：有 ST-T 波異常（T 波倒置和/或 ST 抬高或壓低 > 0.05 mV），LVH：根據 Estes 標準顯示可能或明確的左心室肥厚]
# MaxHR	達到的最大心率 [60 到 202 之間的數值]
# ExerciseAngina	運動誘發的心絞痛[Y：是，N：否]
# Oldpeak	ST [抑鬱症測量的數值]
# ST_Slope	峰值運動ST段的斜率[Up：向上傾斜，Flat：平坦，Down：向下傾斜]
# HeartDisease	輸出類[1：心髒病，0：正常]

```
✓ print(df.shape) ···
(918, 12)
```

```
✓ print(df.columns) ···
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

年齡、性別小提琴分佈圖、圓餅圖

資料前處理




```python
# 年齡 ( Age ) : 性別與年齡 小提琴圖 圓餅圖 ( hue = HeartDisease )
plt.subplot(1, 2, 1)
sns.violinplot(data=df, x="Sex", y="Age", hue="HeartDisease", split=True)
plt.subplot(1, 2, 2)
df.groupby('Sex').size().plot(kind='pie', autopct='%.0f%%')
plt.title("Sex")
plt.show()
```

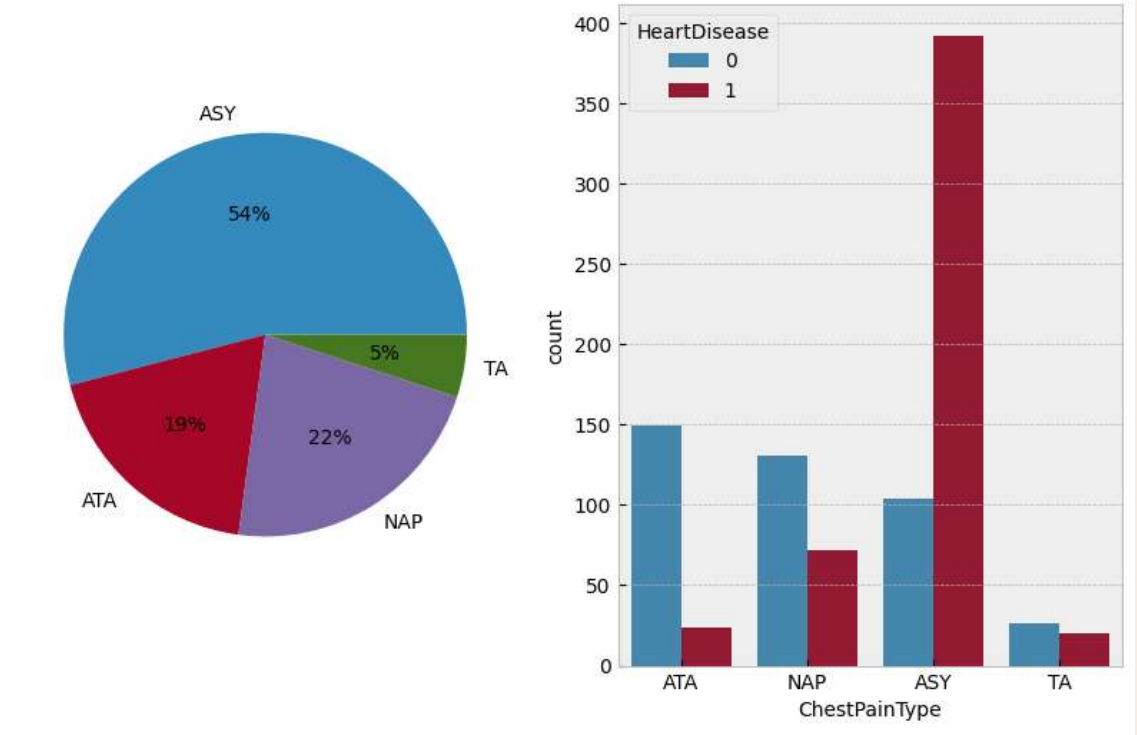
男生比例占**79%**、女生比例占**21%**

男生年齡**50~70**歲有心臟病人數較多

女生年齡**50~70**歲有心臟病人數較多

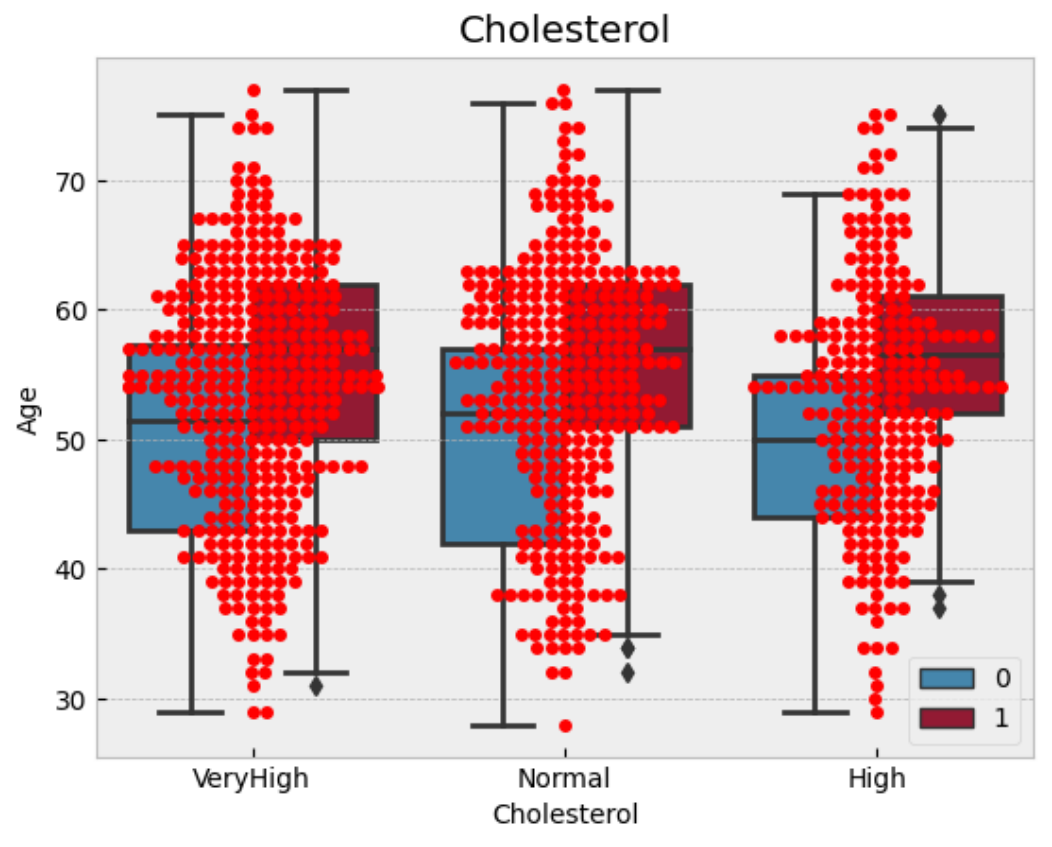
ChestPainType 胸痛類型長條圖、圓餅圖

```
# 胸痛類型（ChestPainType）：類型圓餅圖比例，長條圖（ hue = HeartDisease ）
plt.figure(figsize=(10,6))
plt.title("ChestPainType")
plt.subplot(1, 2, 1)
df.groupby("ChestPainType").size().plot(kind='pie', autopct='%.0f%%')
plt.subplot(1, 2, 2)
sns.countplot(data=df, x="ChestPainType", hue="HeartDisease")
plt.show()
```



Cholesterol 血清膽固醇箱型圖

```
# 血清膽固醇（Cholesterol） : 膽固醇與年紀 盒狀圖 ( hue = HeartDisease )
chol = df["Cholesterol"].apply(
    lambda x: "VeryHigh" if x >= 240 else ("High" if x >= 200 else "Normal"))
sns.boxplot(x=chol, y=df["Age"], hue=df["HeartDisease"])
sns.swarmplot(y="Age",x=chol, data=df,color="r")
plt.title("Cholesterol")
plt.show()
```



VeryHigh:在超標膽固醇具有心臟病的病患落在**50~60**歲左右,人數最多

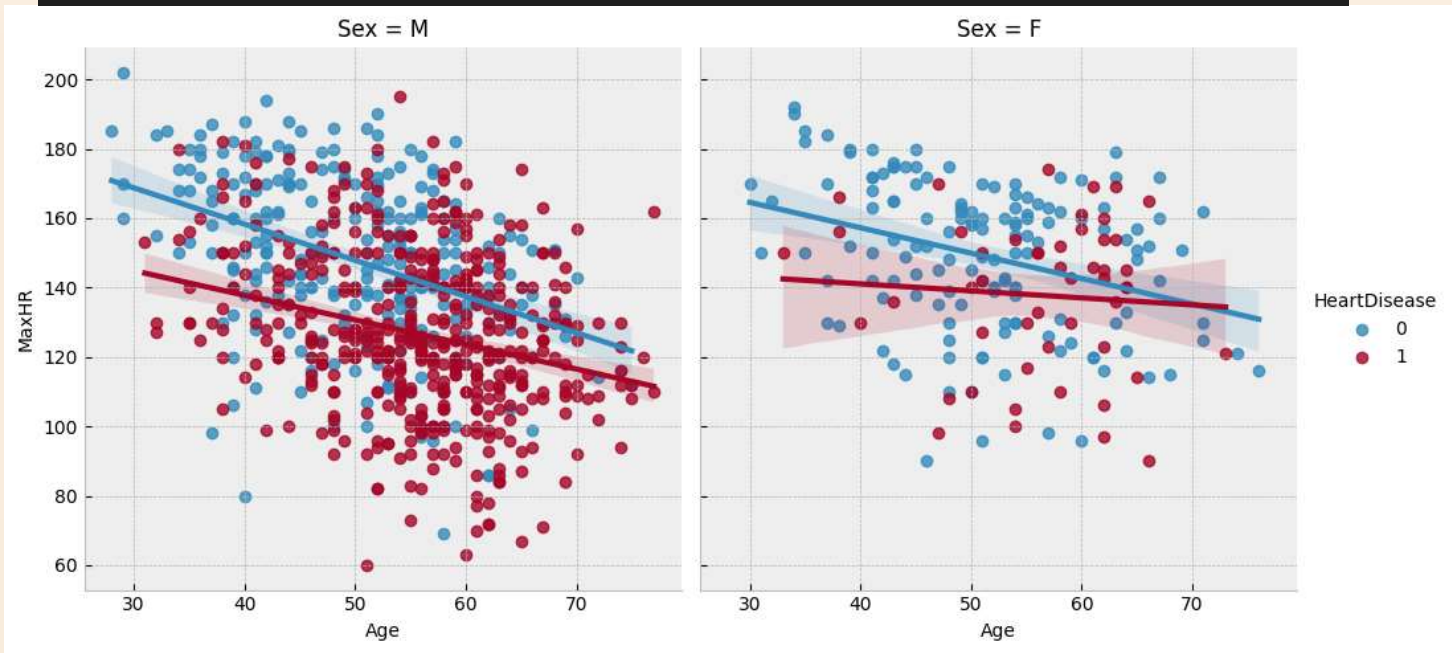
Normal:一般膽固醇具有心臟病的病患落在**50~60**歲左右,人數次之

High:在高膽固醇具有心臟病的病患落在**50~60**歲左右,人數較少

將膽固醇分類為三類:

MaxHR 最大心率與年齡散佈圖

```
# 達到的最大心率（MaxHR）： 年齡與最大心率 散佈圖 ( hue = HeartDisease )
sns.lmplot(data=df, x="Age", y="MaxHR", hue="HeartDisease",col="Sex")
plt.show()
```



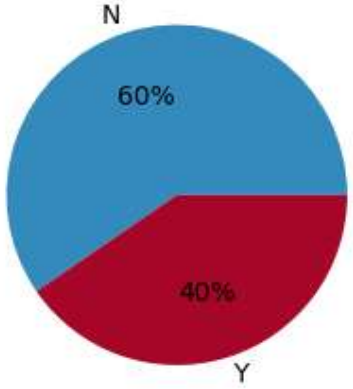
回歸線呈現負成長(年紀越大,最大心率越低,較容易有心臟病)

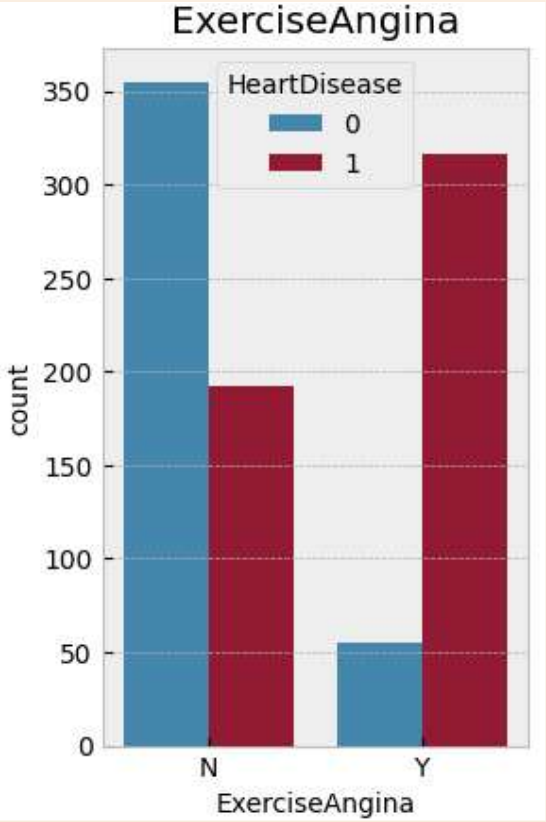
男性有心臟病落在年齡**50~70**歲,最大心率落在**100~140**左右

女性有心臟病落在年齡**50~70**歲,最大心率落在**100~140**左右

ExerciseAngina 運動誘發心絞痛長條圖、圓餅圖

```
# 運動誘發的心絞痛（ExerciseAngina）：可以使用計數圖或餅圖顯示運動誘發的心絞痛的分佈情況
# ，並將其與患心臟病的人進行對比。
plt.subplot(1,2,1)
df.groupby("ExerciseAngina").size().plot(kind="pie", autopct='%.0f%%')
plt.subplot(1,2,2)
plt.title("ExerciseAngina")
sns.countplot(data=df, x="ExerciseAngina", hue="HeartDisease")
plt.show()
```

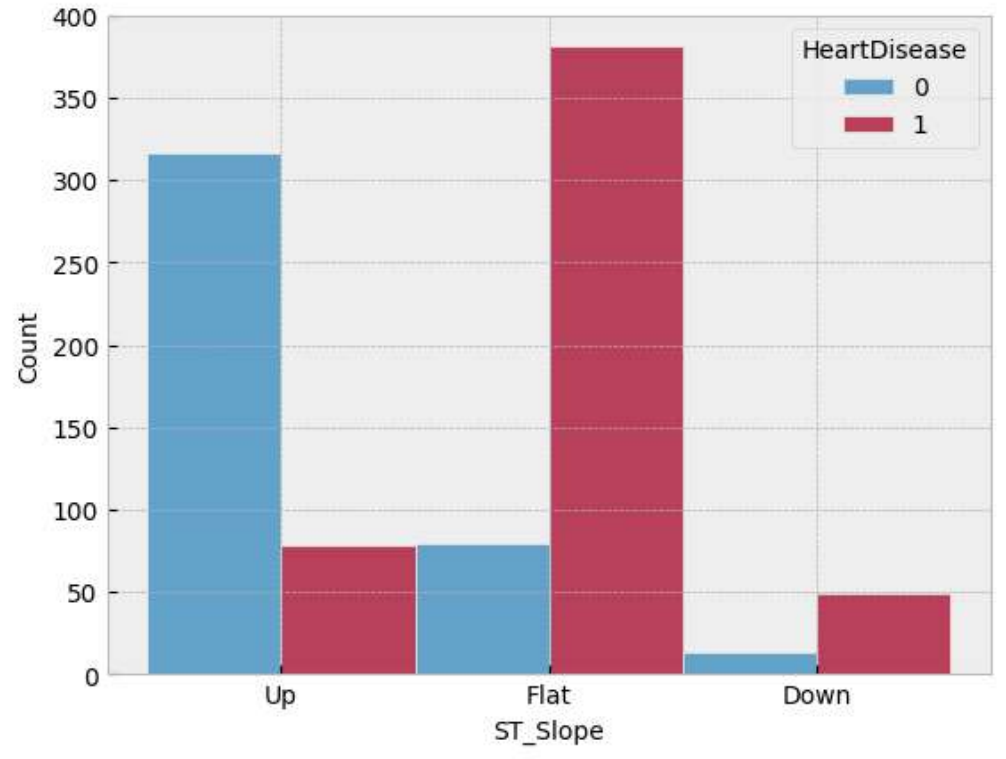




由長條圖可知，運動誘發心絞痛診斷出心臟病的機率非常高

ST_Slope 峰值運動ST段的斜率

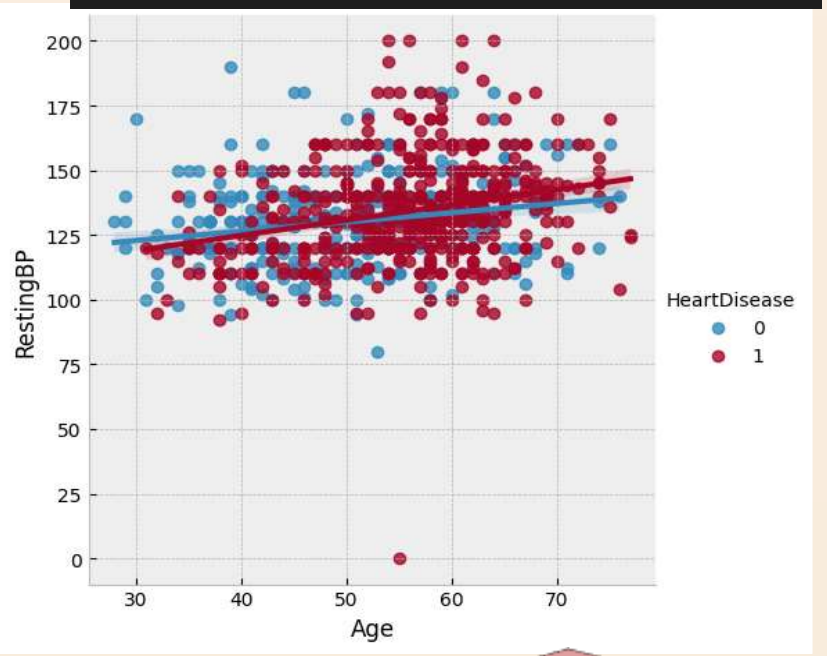
```
# ST_Slope 峰值運動ST段的斜率
sns.histplot(data=df,x="ST_Slope",hue="HeartDisease",multiple="dodge")
plt.show()
```



由長條圖可知，在**Flat**平坦的心臟病人數比其它類型的區段多非常多。

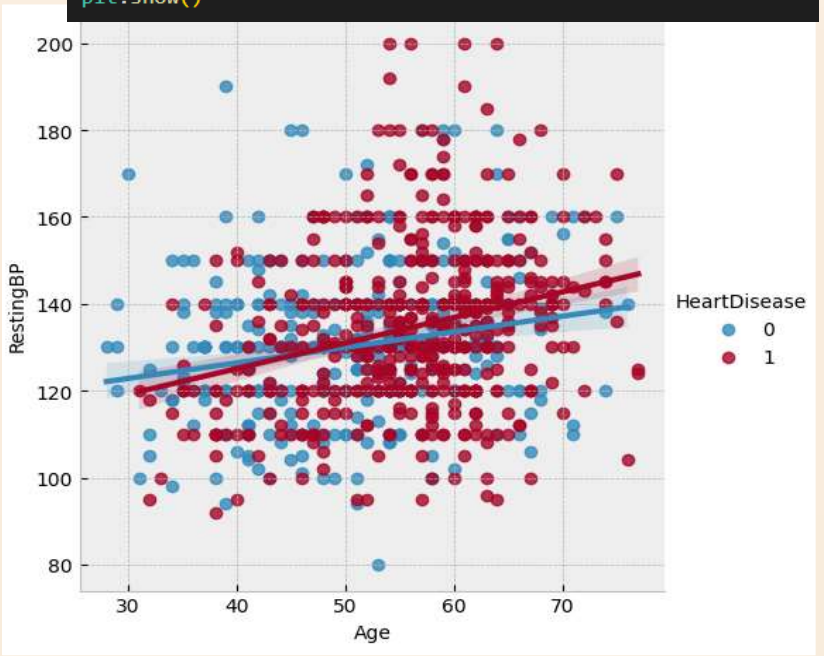
Resting 靜息血壓、年齡散佈圖

```
# RestingBP 靜息血壓
sns.lmplot(data=df,x="Age",y="RestingBP",hue="HeartDisease")
plt.show()
print(df[df["RestingBP"] == 0])
```



在靜息血壓0的位置，若正常人是不可能有血壓為零的數值，所以代表數據上有異常。

```
RestingBP_mean=df["RestingBP"].replace(0,np.nan).mean()
df["RestingBP"]=df["RestingBP"].replace(0,RestingBP_mean)
print(df[df["RestingBP"] == 0])
sns.lmplot(data=df,x="Age",y="RestingBP",hue="HeartDisease")
plt.show()
```



將錯誤的數據修改為平均值後正常的視圖。

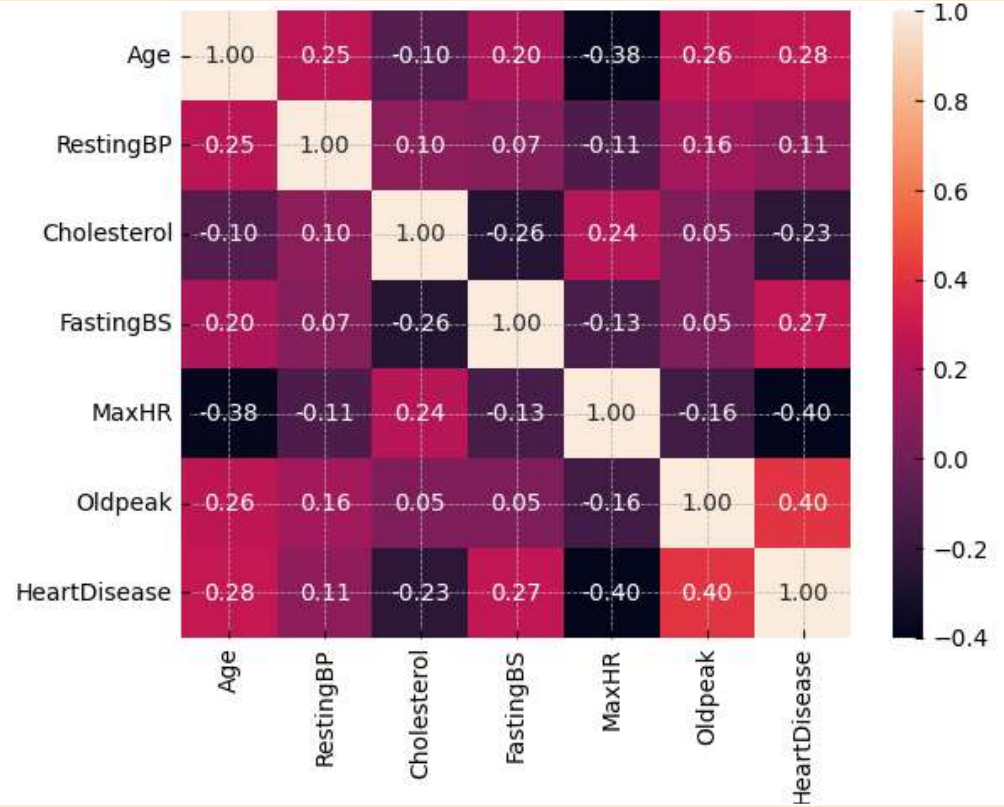
特徵工程

```python
# Sex 患者性別[M=男, F=女]
df["Sex"] = df["Sex"].apply(lambda x: 1 if x == "M" else 0)
print(df["Sex"].value_counts())

# ST_Slope 峰值運動ST段的斜率[Up：向上傾斜，Flat：平坦，Down：向下傾斜]
def st_slope_num(x):
    if x == "Up":
        return 1
    elif x == "Flat":
        return 2
    else:
        return 3
df["ST_Slope"] = df["ST_Slope"].apply(st_slope_num)
print(df["ST_Slope"].value_counts())

# ChestPainType 胸痛類型【TA=典型心絞痛, ATA=非典型心絞痛, NAP=非心絞痛, ASY=無症狀】
def cptype(x):
    if x == "TA":
        return 1
    elif x == "ATA":
        return 2
    elif x == "NAP":
        return 3
    else:
        return 4
df["ChestPainType"] = df["ChestPainType"].apply(cptype)
print(df["ChestPainType"].value_counts())
# RestingECG 靜息心電圖結果
# normal：正常
# ST：有 ST-T 波異常（T 波倒置和/或 ST 抬高或壓低 > 0.05 mV）
# LVH：根據 Estes 標準顯示可能或明確的左心室肥厚
df["RestingECG"] = df["RestingECG"].apply(
    lambda x: 0 if x == "Normal" else (1 if x == "LVH" else 2))
print(df["RestingECG"].value_counts())
```

資料類別轉換成數據，便於機器學習訓練與預測。

心臟病相關係數熱力圖

```python
# 各特徵相關熱力圖
df_corr = df.corr()
print(df_corr["HeartDisease"].sort_values(ascending=False))
sns.heatmap(df_corr, annot=True, fmt=".2f")
plt.show()
```



心臟病相關係數

```
✓ df_corr = df.corr() ···

HeartDisease    1.000000
Oldpeak         0.403951
Age             0.282039
FastingBS       0.267291
RestingBP       0.107589
Cholesterol    -0.232741
MaxHR          -0.400421
Name: HeartDisease, dtype: float64
```

其它Feature配對圖

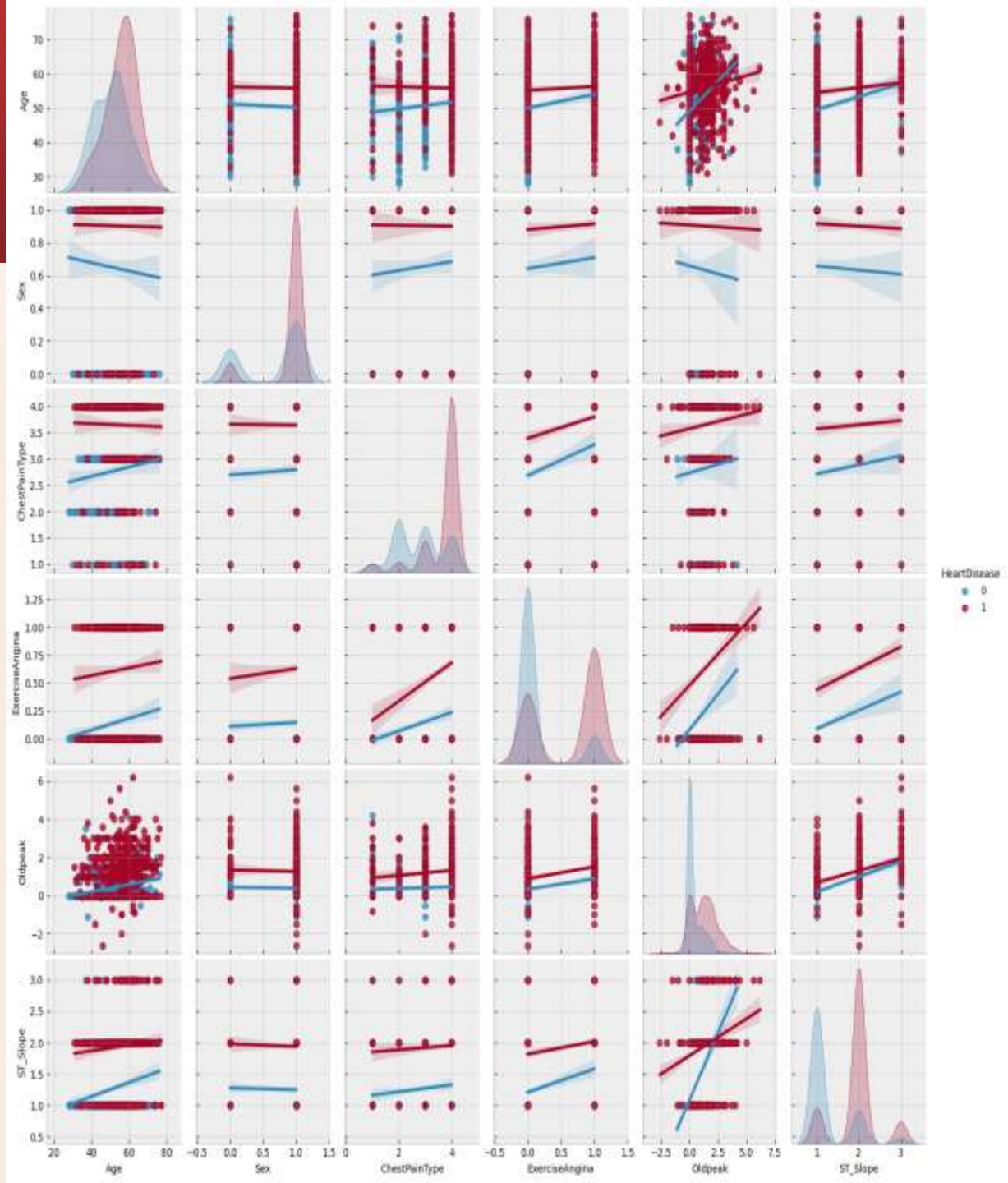
刪除與**HeartDisease**相關性最低的前五個特徵

```
✓ df_corr = df.corr() ···
```

MaxHR	-0.400421
Cholesterol	-0.232741
RestingECG	0.107628
RestingBP	0.117938
FastingBS	0.267291
Age	0.282039
Sex	0.305445
Oldpeak	0.403951
ChestPainType	0.471354
ExerciseAngina	0.494282
ST_Slope	0.558771
HeartDisease	1.000000

Name: HeartDisease, dtype: float64

```python
# 刪除最不相關的前五個特徵
print(df_corr["HeartDisease"].sort_values(ascending=True))
top_feature = df.drop(
    ["FastingBS", "RestingBP", "RestingECG", "Cholesterol", "MaxHR"], axis=1)
print(top_feature.columns)
sns.pairplot(top_feature, kind="reg", diag_kind="auto", hue="HeartDisease")
plt.show()
```



建模、拆分資料

```
#%%   Modeling


X = df.drop(["HeartDisease"], axis=1)
y = df["HeartDisease"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=20)

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

RandomForset 隨機森林

```python
rfc = RandomForestClassifier(random_state=20)

# # GridSearchCV 參數最佳化
param_grid = {
    "n_estimators": list(range(20, 100, 10)),
    "max_depth": list(range(2, 9, 1))
}
rfc_grid_cv = GridSearchCV(rfc, param_grid, cv=5)

# # 訓練及預測
rfc_grid_cv.fit(X_train, y_train)
print(
    f"best_estimator: {rfc_grid_cv.best_estimator_}\n\
    best_score: {rfc_grid_cv.best_score_:.2f}")
y_rfc_pred = rfc_grid_cv.predict(X_test)
print(f"預測結果: \n{y_rfc_pred}")

# # 準確率
accuracy = metrics.accuracy_score(y_test, y_rfc_pred)
print(f"rfc_Accuracy: : {accuracy:.3f}")

# # 混淆矩陣
cm = confusion_matrix(y_test, y_rfc_pred)
print("Matrix: \n", cm)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=cm, display_labels=[True, False])
cm_display.plot()
plt.show()
```

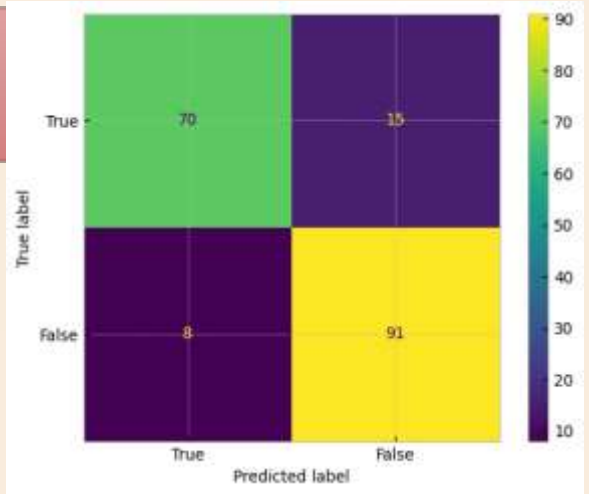
GridSeachCV 得最佳化參數

```
best_estimator: RandomForestClassifier(max_depth=6, n_estimators=50, random_state=20)
    best_score: 0.87
```

訓練後得預測結果、精確度

```
預測結果:
[1 1 1 1 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1 0 0 1 1 1 0 0 1 1 1
 0 1 1 0 0 0 1 0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 1 0 1 0
 1 1 1 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1
 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1
 1 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 0 1 0 0]
rfc_Accuracy: : 0.875
```

混淆矩陣




```
Matrix:
 [[70 15]
 [ 8 91]]
```

SVM 支持向量機(標準化)

建立**Pipe**結合**SVM**、**StanderScaler**標準化提高精確度。

```python
# # 建立 Pipeline
pipe = Pipeline([('scaler', StandardScaler()),
                 ('svc', SVC())])

# # GridSearchCV 參數最佳化
param_grid = {
    "svc__C": np.linspace(1, 1.5, 10),
    "svc__kernel": ['linear', 'poly', 'rbf', 'sigmoid'],
    "svc__gamma": ['scale', 'auto']
}
svc_grid_cv = GridSearchCV(estimator=pipe, param_grid=param_grid, cv=5)

# # 訓練及預測
svc_grid_cv.fit(X_train, y_train)
print(
    f"best_params: {svc_grid_cv.best_params_}\n\
    best_score: {svc_grid_cv.best_score_:.2f}")
y_svc_pred = svc_grid_cv.predict(X_test)
print(f"預測結果: \n{y_svc_pred}")

# # 準確率
accuracy = metrics.accuracy_score(y_test, y_svc_pred)
print(f"SVC_Accuracy: {accuracy:.3f}")

# # 混淆矩陣
cm = confusion_matrix(y_test, y_svc_pred)
print("Matrix: \n", cm)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=cm, display_labels=[True, False])
cm_display.plot()
plt.show()
```

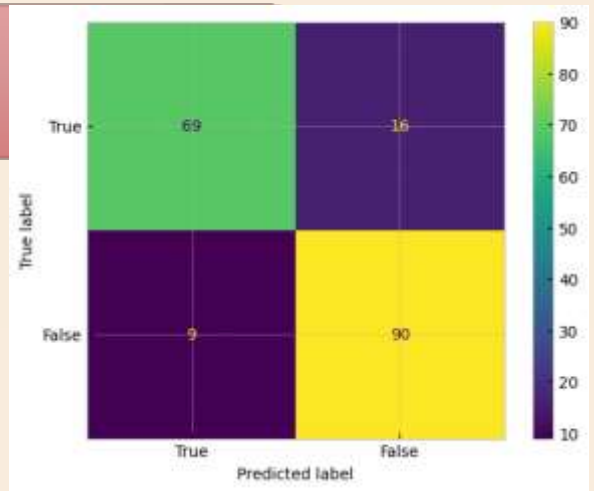
GridSeachCV 得最佳化參數

```
best_params: {'svc__C': 1.5, 'svc__gamma': 'scale', 'svc__kernel': 'rbf'}
    best_score: 0.87
```

訓練後得預測結果、精確度

```
預測結果:
[1 1 1 1 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1
 0 1 1 0 0 1 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 1 0 1 0 0 1 0 0 1 0 0 1 0
 1 1 1 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1
 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1
 1 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1]
SVC_Accuracy: 0.864
```

混淆矩陣




```
Matrix:
 [[69 16]
 [ 9 90]]
```

SVM 支持向量機(無標準化)

無標準化

```
svc = SVC(random_state=20, C=1.5, kernel="rbf", gamma="scale")

# # 訓練及預測
svc.fit(X_train, y_train)
y_svc_pred = svc.predict(X_test)
print(f"預測結果: \n{y_svc_pred}")

# # 準確率
accuracy = metrics.accuracy_score(y_test, y_svc_pred)
print(f"SVC_Accuracy: {accuracy:.3f}")

# # 混淆矩陣
cm = confusion_matrix(y_test, y_svc_pred)
print("Matrix: \n", cm)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=cm, display_labels=[True, False])
cm_display.plot()
plt.show()
```

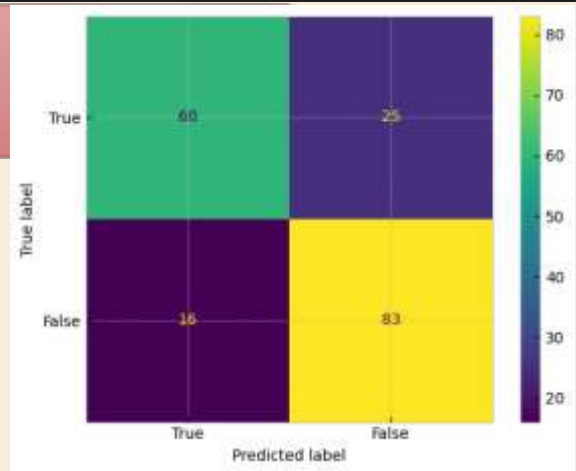
利用剛才**GridSeachCV** 得到的最佳化套進**SVM**模型裡面。

```
best_params: {'svc__C': 1.5, 'svc__gamma': 'scale', 'svc__kernel': 'rbf'}
    best_score: 0.87
```

訓練後得預測結果、精確度。無標準化精確度只有0.777

```
預測結果:
[1 1 0 1 1 0 1 0 0 1 1 0 1 1 1 1 0 1 0 1 0 0 1 1 1 1 1 0 0 1 1 1 1 0 1 0 0
 1 1 1 0 1 1 1 1 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 0 1 0 0 1 0 0 1 1
 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1
 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1
 1 1 1 0 1 1 1 1 0 1 0 0 1 1 0 1 1 0 0 1 1 0 1 1 0 0 0 0 1 0 1 0 0 1 0 1]
SVC_Accuracy: 0.777
```

混淆矩陣




```
Matrix:
 [[60 25]
 [16 83]]
```

LogisticRegression 邏輯回歸(標準化)

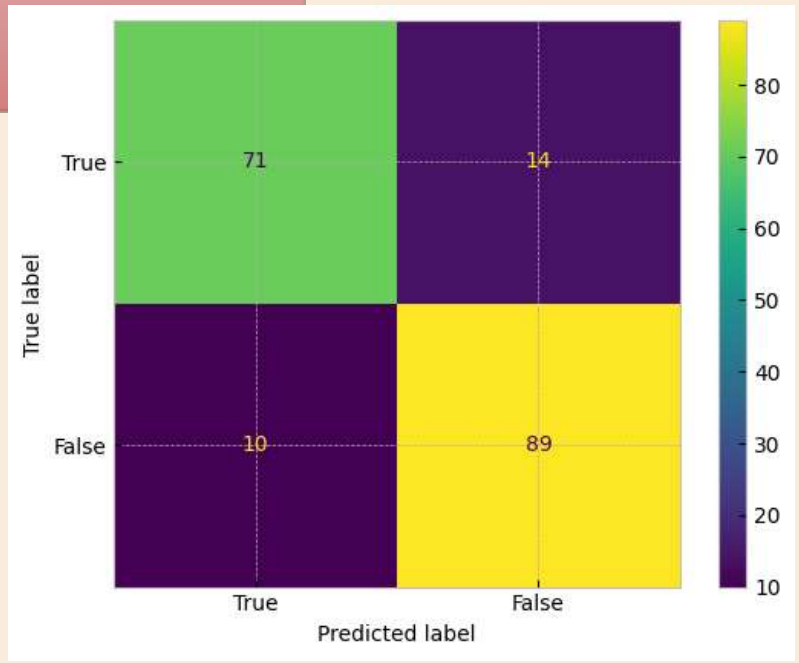
建立**Pipe**結合**LogisticRegression** 、**StanderScaler**標準化提高精確度。

```
# # 邏輯回歸 標準化
lr = LogisticRegression(random_state=20)

# # 建立Pipeline 標準化與模型
pipe = Pipeline([("scaler", StandardScaler()),
                 ("lr", lr)])

# # 訓練及預測
pipe.fit(X_train, y_train)
y_lr_pred = pipe.predict(X_test)
print(f"預測結果: \n{y_lr_pred}")

# # 準確率
accuracy = metrics.accuracy_score(y_test, y_lr_pred)
print(f"lr_Accuracy: {accuracy:.3f}")

# # 混淆矩陣
cm = confusion_matrix(y_test, y_lr_pred)
print("Matrix: \n", cm)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=cm, display_labels=[True, False])
cm_display.plot()
plt.show()
```

訓練後得預測結果、精確度

```
預測結果:
[1 1 1 1 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 0 1 1 1 0 1 1 1 1
 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 1 1 1 0 0 1 0 0 1 0
 1 1 1 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1
 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1
 1 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1]
lr_Accuracy: 0.8696
```

混淆矩陣




```
Matrix:
 [[71 14]
 [10 89]]
```

LogisticRegression 邏輯回歸(無標準化)

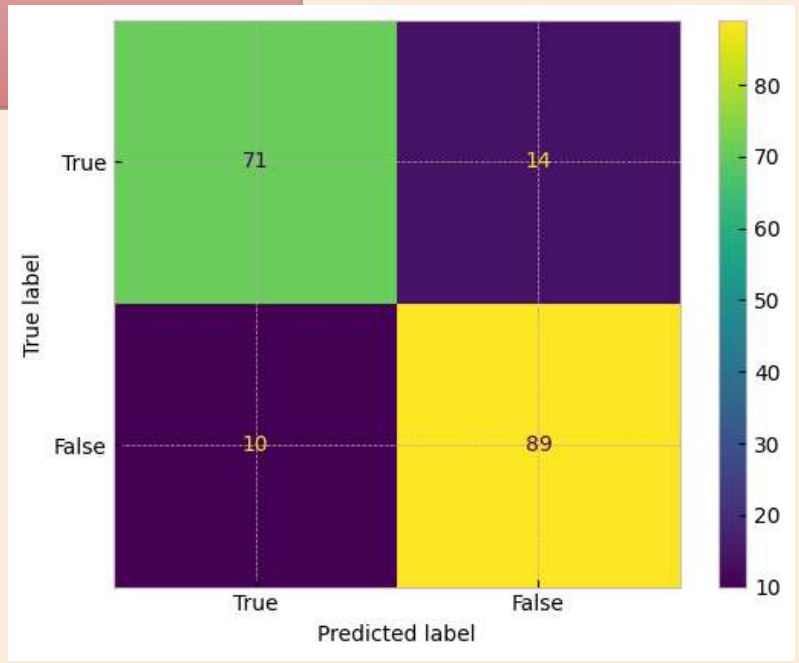
無標準化

```python
lr = LogisticRegression(max_iter=1000, random_state=20)

# # 訓練及預測
lr.fit(X_train, y_train)
y_lr_pred = lr.predict(X_test)
print(f"預測結果: \n{y_lr_pred}")

# # 準確率
accuracy = metrics.accuracy_score(y_test, y_lr_pred)
print(f"lr_Accuracy: {accuracy:.4f}")

# 混淆矩陣
cm = confusion_matrix(y_test, y_lr_pred)
print("Matrix: \n", cm)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=cm, display_labels=[True, False])
cm_display.plot()
plt.show()
```

訓練後得預測結果、精確度

```
預測結果:
[1 1 1 1 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 0 1 1 1 0 1 1 1 1
 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 0 0 1 0 0 1 0
 1 1 1 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1
 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1
 1 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1]
lr_Accuracy: 0.8696
```

混淆矩陣




```
Matrix:
 [[71 14]
 [10 89]]
```

預測結論

支持向量機

- 精確度達**0.864**
- 標準化與無標準化結果差異很大

邏輯回歸

- 精確度達**0.870**
- 標準化與無標準化結果差異不大

隨機森林

- 精確度達**0.875**

總結

在預測心髒病的問題中，我使用了一個包含**12**個特徵的數據集以及**918**個樣本，包括年齡、性別、胸痛類型、靜息血壓等等。首先對數據進行了探索性數據分析，然後進行了數據預處理，包括缺失值填充、特徵工程和標準化等步驟。

我還使用了各種機器學習模型進行了訓練和評估，包括邏輯回歸、支持向量機和隨機森林等模型。在模型評估方面，我們使用了交叉驗證和網格搜索來優化模型參數，以獲得更好的預測性能。最終，我選擇了隨機森林模型，並使用測試集進行了最終的評估，得到了高達 **87.5%** 的準確率。

總結來說，在預測心髒病這個問題上，我們使用了數據預處理、機器學習模型選擇、參數調優等多個步驟，最終得到了一個準確率比較高的模型，為預防心臟疾病提供參考價值。

使用模組

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

參考資料

https://www.epochtimes.com/b5/19/3/1/n11082192.htm

kaggle