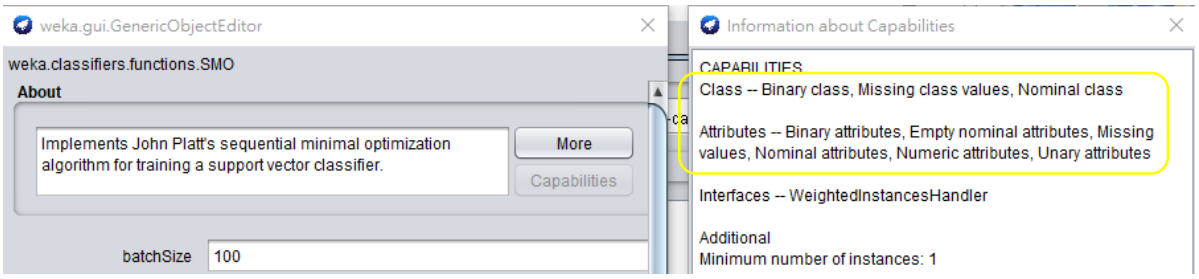
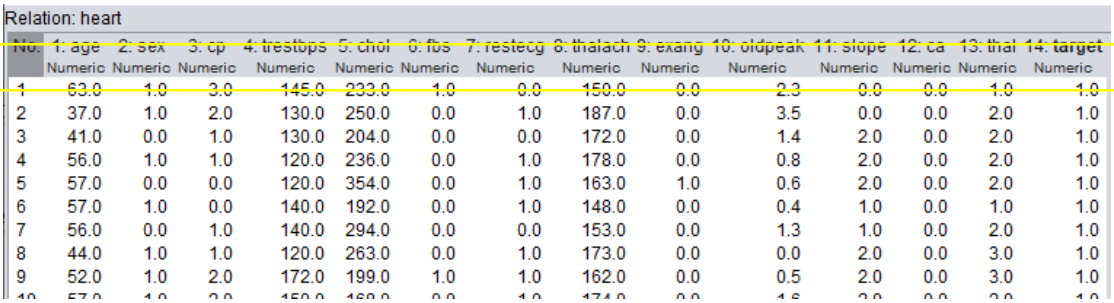
# ECT HW4

## Weka 部分:

## (a)




➔ 首先，去看看 Weka SMO 的使用條件，需要何種資料型態



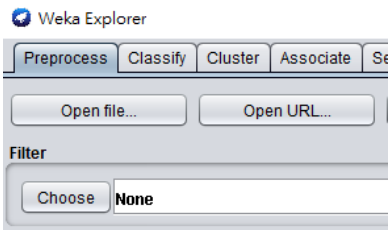
Relation: heart

| No. | 1: age | 2: sex | 3: cp | 4: trestbps | 5: chol | 6: fbs | 7: restecg | 8: thalach | 9: exang | 10: oldpeak | 11: slope | 12: ca | 13: thal | 14: target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric | Numeric |
| 1 | 63.0 | 1.0 | 3.0 | 145.0 | 233.0 | 1.0 | 0.0 | 150.0 | 0.0 | 2.3 | 0.0 | 0.0 | 1.0 | 1.0 |
| 2 | 37.0 | 1.0 | 2.0 | 130.0 | 250.0 | 0.0 | 1.0 | 187.0 | 0.0 | 3.5 | 0.0 | 0.0 | 2.0 | 1.0 |
| 3 | 41.0 | 0.0 | 1.0 | 130.0 | 204.0 | 0.0 | 0.0 | 172.0 | 0.0 | 1.4 | 2.0 | 0.0 | 2.0 | 1.0 |
| 4 | 56.0 | 1.0 | 1.0 | 120.0 | 236.0 | 0.0 | 1.0 | 178.0 | 0.0 | 0.8 | 2.0 | 0.0 | 2.0 | 1.0 |
| 5 | 57.0 | 0.0 | 0.0 | 120.0 | 354.0 | 0.0 | 1.0 | 163.0 | 1.0 | 0.6 | 2.0 | 0.0 | 2.0 | 1.0 |
| 6 | 57.0 | 1.0 | 0.0 | 140.0 | 192.0 | 0.0 | 1.0 | 148.0 | 0.0 | 0.4 | 1.0 | 0.0 | 1.0 | 1.0 |
| 7 | 56.0 | 0.0 | 1.0 | 140.0 | 294.0 | 0.0 | 0.0 | 153.0 | 0.0 | 1.3 | 1.0 | 0.0 | 2.0 | 1.0 |
| 8 | 44.0 | 1.0 | 1.0 | 120.0 | 263.0 | 0.0 | 1.0 | 173.0 | 0.0 | 0.0 | 2.0 | 0.0 | 3.0 | 1.0 |
| 9 | 52.0 | 1.0 | 2.0 | 172.0 | 199.0 | 1.0 | 1.0 | 162.0 | 0.0 | 0.5 | 2.0 | 0.0 | 3.0 | 1.0 |

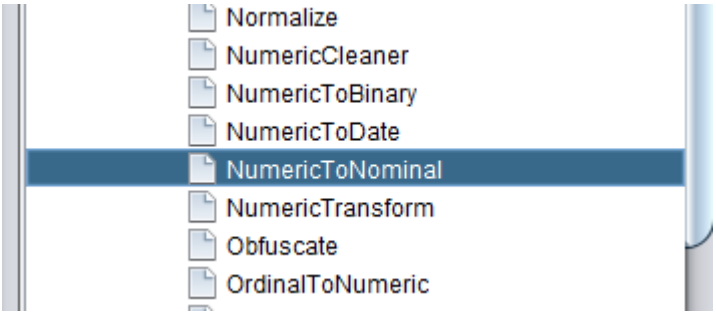
➔ 再來，去觀察看看目前的資料型態。

無法使用的原因 ：

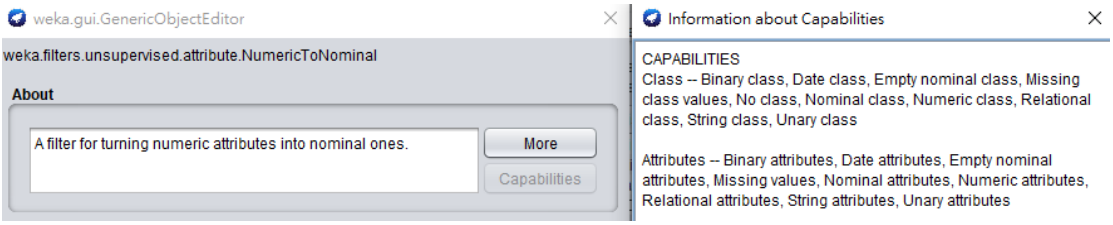
➔ 對比兩張圖後可以看出，Output 不能接受 Numeric 的參數，但是我們的資料中所有型態皆是 Numeric，因此我們要把 Output 的 Target 屬性改成 Nominal。



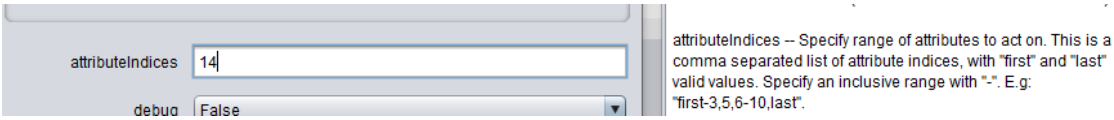

➔ 在 Weka 的 Preprocess 中找到 Filter




➔ 在 unsupervised → attribute 中有一個 Numeric 轉 Nominal 的方法

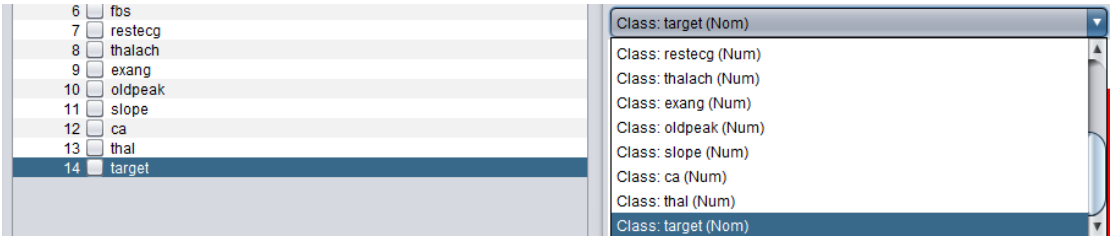



➔ 觀看使用條件，他幾乎甚麼都可以轉，連空值都行(雖然 target 沒有空值)，因此可以使用。

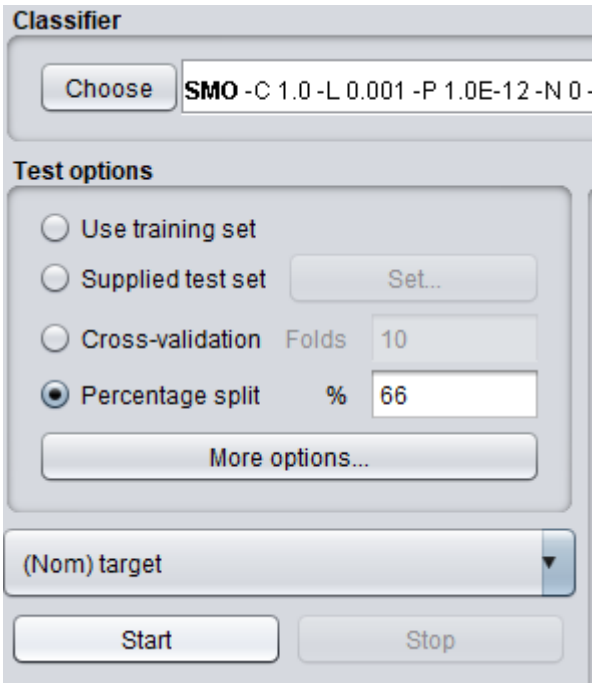



➔ 查看參數意義，可知道這一參數為指定 attribute，因 target 為第 14 個屬性，因此填入 14

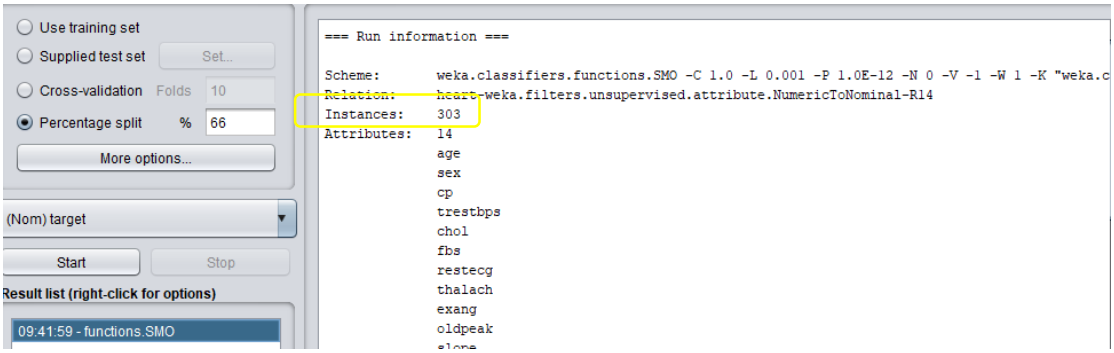

➔ 記得點選 Apply，不然不會生效。這些操作都是暫存的，原始檔不會修改，想要修改記得另存新檔。




➔ 可以看到 target 屬性已經變為 Nominal

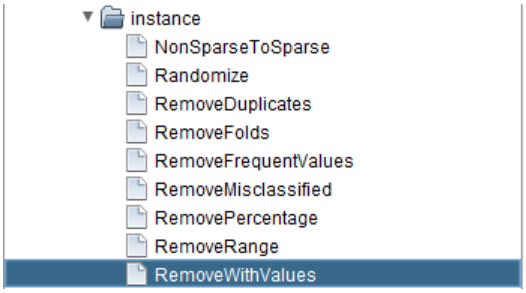



➔ 回去查看 SMO 的「Start」Button 已經可以使用了

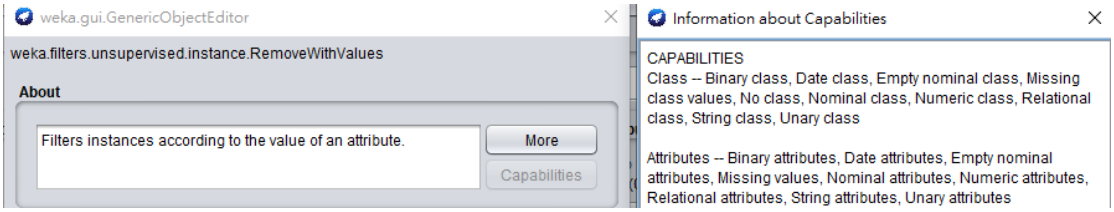



➔ 這裡雖然沒有處理 missing data，但 Weka 的 SMO 已經可以分析了。可以注意 Instance 還是 303，代表沒有刪除任何擁有 missing data 的實體。

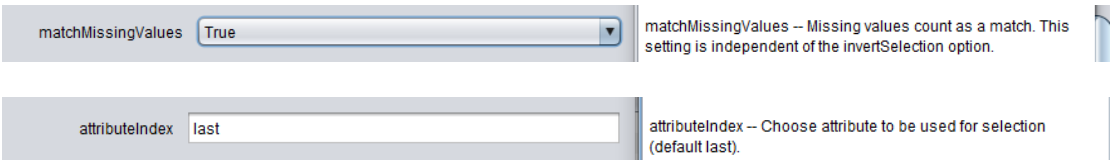
## (b)




➔ 因為要移除空值的資料，因此這次要去「Instance」中找方法使用




➔ 一樣先去看使用條件

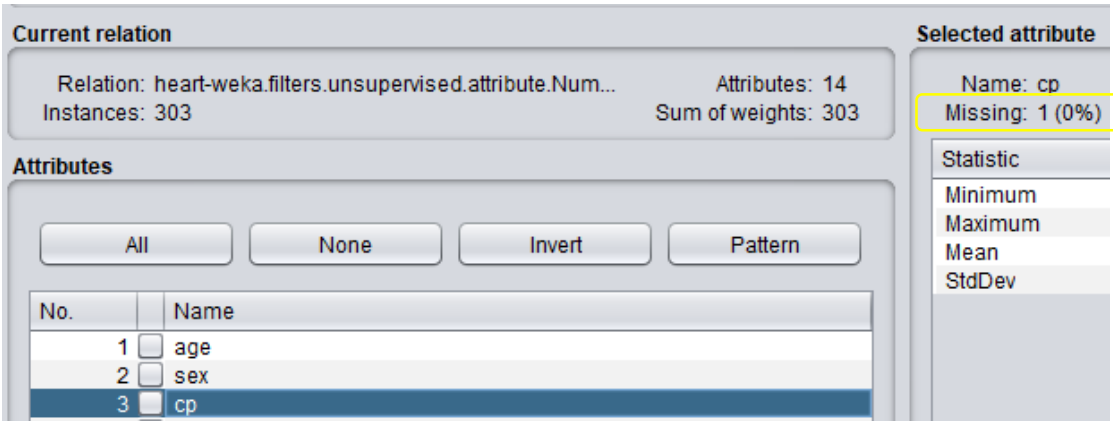



➔ 然後再去看參數意義，這裡有 2 個參數特別重要：

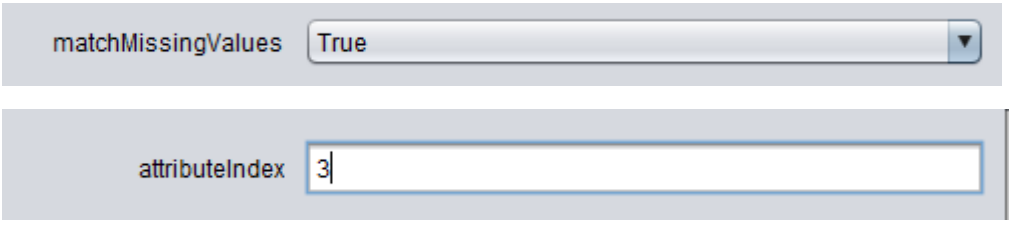
「matchMissingValues」代表要 match 到 missing data 才刪除。

「attributeIndex」代表處理指定的屬性，這裡真的是「index」而不是

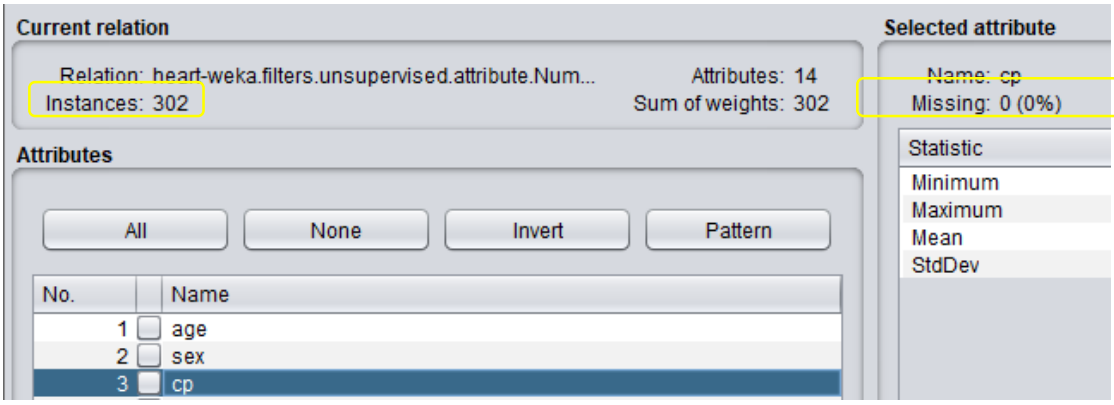
「indices」，對!他是單數……所以你要一個一個去選擇屬性來處理。




➔ 舉例來說，cp 屬性有 1 個 missing data，他是第 3 個屬性

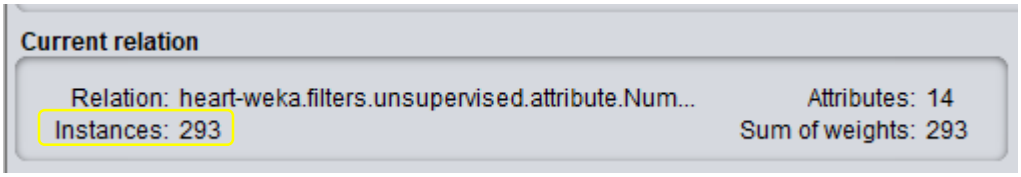



➔ Match 設為 true，Index 設為 3




➔ 處理完後可看到 missing 變成 0 了，總 Instance 也少一個了(303-1) = 302

重複這個動作直到處理完所有資料，




➔ 處理完後會剩下 293 筆資料。

這是 Weka 的作法。我有另外寫了一個 Python 的小程序來處理。

```
import pandas as pd
def removeAllMissData(raw_df):
    print("Before handle:\n", raw_df.isnull().sum())
    clean_df = raw_df.dropna()
    print("After handle:\n", clean_df.isnull().sum())
    clean_df.to_csv('clean_data.csv', index = False)
```

```
df = pd.read_csv('heart.csv')
removeAllMissData(df)
```

➔ 用 dropa()來處理空值，用 to_csv()來輸出檔案，index = False 記得要設，

不然 Weka 無法讀取，因為他會多一個 column。

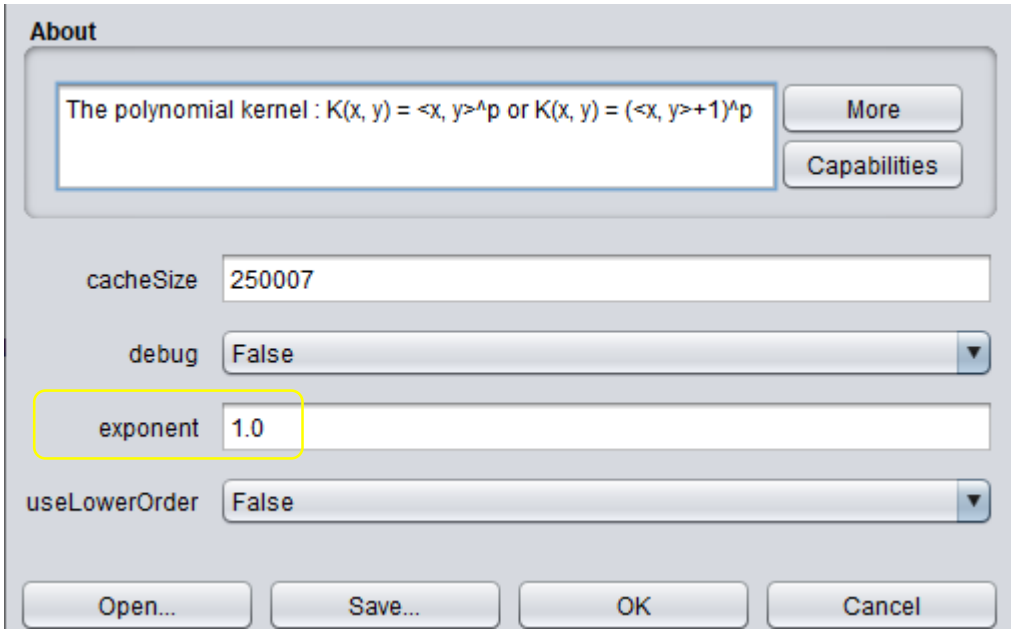
```
Before handle:
 age         0
sex          0
cp           1
trestbps     0
chol         2
fbs          0
restecg      3
thalach      0
exang        1
oldpeak      2
slope        1
ca           0
thal         0
target       0
dtype: int64
```

```
After handle:
 age         0
sex          0
cp           0
trestbps     0
chol         0
fbs          0
restecg      0
thalach      0
exang        0
oldpeak      0
slope        0
ca           0
thal         0
target       0
dtype: int64
```

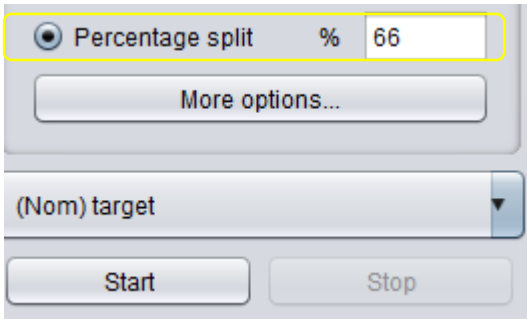
➔ 並且可看到這種輸出結果。

## (c)

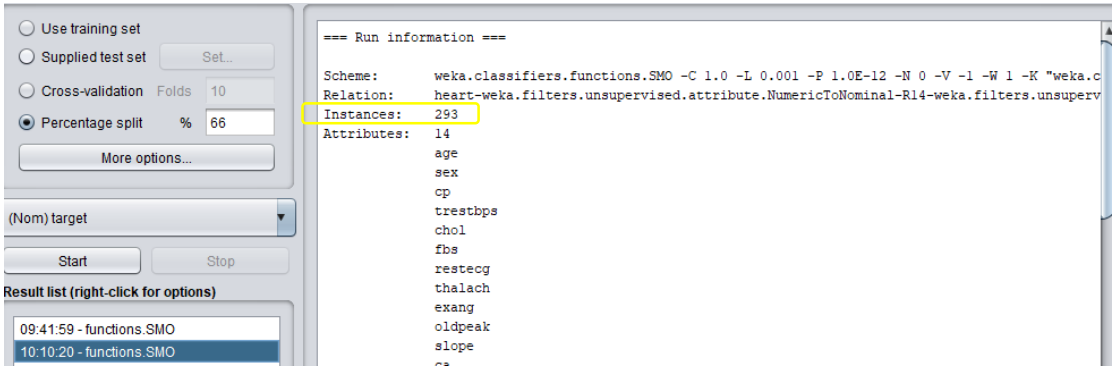

➔ 先到 SMO 的 kernel 部分，選擇好 PolyKernal，這個方法可以決定要用幾次方的 Kernel 來訓練。



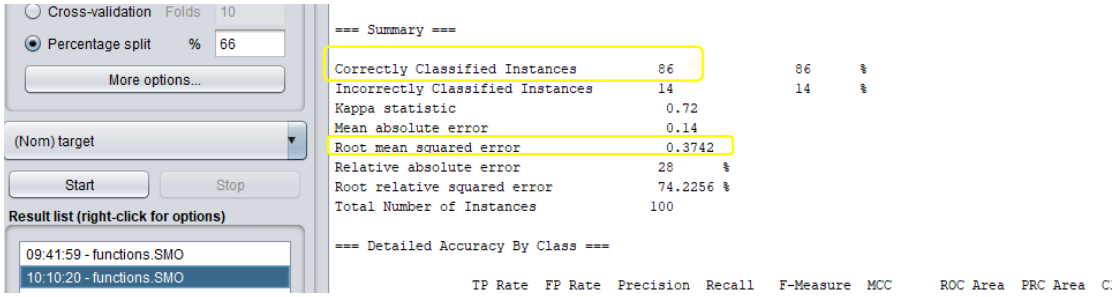

➔ 因題目要求「Linear」，所以 exponent 設為 1




➔ 依照題目要求，Percentage spilt 設為 66%。




➔ 訓練後可看到這次並非用 303 筆資料，而是 203 筆資料在訓練。




➔ 準確率為 86%，比未處理 missing data 時的 83%還準，因此可判斷此步驟是有一定程度的意義。

## Python 部分 ：

## (d)

```
import pandas as pd
df = pd.read_csv('heart.csv')
df
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3.0 | 145 | 233.0 | 1 | 0.0 | 150 | 0.0 | 2.3 | 0.0 | 0 | 1 | 1 |
| **1** | 37 | 1 | 2.0 | 130 | 250.0 | 0 | 1.0 | 187 | 0.0 | 3.5 | 0.0 | 0 | 2 | 1 |
| **2** | 41 | 0 | 1.0 | 130 | 204.0 | 0 | 0.0 | 172 | 0.0 | 1.4 | 2.0 | 0 | 2 | 1 |
| **3** | 56 | 1 | 1.0 | 120 | 236.0 | 0 | 1.0 | 178 | 0.0 | 0.8 | 2.0 | 0 | 2 | 1 |
| **4** | 57 | 0 | 0.0 | 120 | 354.0 | 0 | 1.0 | 163 | 1.0 | 0.6 | 2.0 | 0 | 2 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **298** | 57 | 0 | 0.0 | 140 | 241.0 | 0 | 1.0 | 123 | 1.0 | 0.2 | 1.0 | 0 | 3 | 0 |
| **299** | 45 | 1 | 3.0 | 110 | 264.0 | 0 | 1.0 | 132 | 0.0 | 1.2 | NaN | 0 | 3 | 0 |
| **300** | 68 | 1 | 0.0 | 144 | 193.0 | 1 | 1.0 | 141 | 0.0 | 3.4 | 1.0 | 2 | 3 | 0 |
| **301** | 57 | 1 | 0.0 | 130 | 131.0 | 0 | 1.0 | 115 | 1.0 | 1.2 | 1.0 | 1 | 3 | 0 |
| **302** | 57 | 0 | 1.0 | 130 | 236.0 | 0 | 0.0 | 174 | 0.0 | 0.0 | 1.0 | 1 | 2 | 0 |

303 rows × 14 columns

➔ 先導入資料

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        302 non-null    float64
 3   trestbps  303 non-null    int64
 4   chol      301 non-null    float64
 5   fbs       303 non-null    int64
 6   restecg   300 non-null    float64
 7   thalach   303 non-null    int64
 8   exang     302 non-null    float64
 9   oldpeak   301 non-null    float64
 10  slope     302 non-null    float64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(6), int64(8)
memory usage: 33.3 KB
```

➔ Info()函數可以看到各種資料，可看出總共有 303 筆資料，但有 6 種屬性的

資料 non-null 的數量並非 303 筆，因此可推斷有 missing data。

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

```
age         0
sex         0
cp          1
trestbps    0
chol        2
fbs         0
restecg     3
thalach     0
exang       1
oldpeak     2
slope       1
ca          0
thal        0
target      0
dtype: int64
```

➔ 使用 isnull().sum()直接統整出有多少筆 missing data，在此為 10 筆。

```
# df.dropna()
df = df.dropna()
```

➔ 使用 dropnd()函數來去除 missing data，記住要重新，不然 DataFrame 並不會改變。

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

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

➔ 在觀察一次可看出 missing data 已經沒有了

## (e)

```
#x:input
x = df.loc[:,"age":"thal"]
print(x)
#y:output
y = df.loc[:,["target"]]
print(y)
```

➔ 用 loc 函數切分為 input、output。

```
     age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1  3.0       145  233.0    1      0.0      150    0.0      2.3
1     37    1  2.0       130  250.0    0      1.0      187    0.0      3.5
2     41    0  1.0       130  204.0    0      0.0      172    0.0      1.4
3     56    1  1.0       120  236.0    0      1.0      178    0.0      0.8
4     57    0  0.0       120  354.0    0      1.0      163    1.0      0.6
..   ...  ...  ...       ...    ...  ...      ...      ...    ...      ...
297   59    1  0.0       164  176.0    1      0.0       90    0.0      1.0
298   57    0  0.0       140  241.0    0      1.0      123    1.0      0.2
300   68    1  0.0       144  193.0    1      1.0      141    0.0      3.4
301   57    1  0.0       130  131.0    0      1.0      115    1.0      1.2
302   57    0  1.0       130  236.0    0      0.0      174    0.0      0.0

     slope  ca  thal
0      0.0   0     1
1      0.0   0     2
2      2.0   0     2
3      2.0   0     2
4      2.0   0     2
..     ...  ..   ...
297    1.0   2     1
298    1.0   0     3
300    1.0   2     3
301    1.0   1     3
302    1.0   1     2

[293 rows x 13 columns]

[293 rows x 13 columns]
     target
0         1
1         1
2         1
3         1
4         1
..      ...
297       0
298       0
300       0
301       0
302       0

[293 rows x 1 columns]
```

## (f)

```
from sklearn.preprocessing import StandardScaler
print("Before:\n", x)
scaler = StandardScaler()
x = scaler.fit_transform(x)
print("After:\n", x)
```

➔ 我在這裡先有印出來看原始狀態，在用 fit_transform()轉換後 assign 回給 x，並且再次印出來檢查是否有轉換成功。結果如下:

```
Before:
      age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0      63    1  3.0       145  233.0    1      0.0      150    0.0      2.3
1      37    1  2.0       130  250.0    0      1.0      187    0.0      3.5
2      41    0  1.0       130  204.0    0      0.0      172    0.0      1.4
3      56    1  1.0       120  236.0    0      1.0      178    0.0      0.8
4      57    0  0.0       120  354.0    0      1.0      163    1.0      0.6
..    ...  ...  ...       ...    ...  ...      ...      ...    ...      ...
297    59    1  0.0       164  176.0    1      0.0       90    0.0      1.0
298    57    0  0.0       140  241.0    0      1.0      123    1.0      0.2
300    68    1  0.0       144  193.0    1      1.0      141    0.0      3.4
301    57    1  0.0       130  131.0    0      1.0      115    1.0      1.2
302    57    0  1.0       130  236.0    0      0.0      174    0.0      0.0

     slope  ca  thal
0      0.0   0     1
1      0.0   0     2
2      2.0   0     2
3      2.0   0     2
4      2.0   0     2
..     ...  ..   ...
297    1.0   2     1
298    1.0   0     3
300    1.0   2     3
301    1.0   1     3
302    1.0   1     2

[293 rows x 13 columns]
After:
 [[ 0.96008384  0.68190908  1.97865831 ... -2.26047188 -0.71658705
  -2.12994828]
 [-1.90587175  0.68190908  1.00592864 ... -2.26047188 -0.71658705
  -0.50051004]
 [-1.46495551 -1.46647115  0.03319897 ...  0.9703489  -0.71658705
  -0.50051004]
 ...
 [ 1.51122914  0.68190908 -0.93953071 ... -0.64506149  1.23652928
   1.1289282 ]
 [ 0.29870947  0.68190908 -0.93953071 ... -0.64506149  0.25997112
   1.1289282 ]
 [ 0.29870947 -1.46647115  0.03319897 ... -0.64506149  0.25997112
  -0.50051004]]
```

➔ 看的出來已經轉為標準化的形式了。都以 0 為中心點左右偏移。

## (g)

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=1)
print(x_train)
print(y_train)
print(x_test)
print(y_test)
```

➔ 依照題目要求，用 train_test_split()切分為 training、testing data。

「最前面的兩個參數 x, y」：代表我們要切割的資料

「test_size=0.33」：代表 testing data 佔 33%

「random_state=1」：為了讓他每次切的結果都一樣，可以想像成它是一個 seed，每次都基於這個來產生相同的結果。1 會有 1 的結果、2 也會有 2 個結果。若要每次切的結果不同可以不要設定這個參數。(P.S. 這是為了評分時大家答案一樣用的吧)

```
[196 rows x 1 columns]
[97 rows x 1 columns]
```

➔ 印出來可以看到他們被切割為 training : 196、testing : 97。驗算一下 293*0.33 = 96.69 ≒ 97。結果是正確的。

## (h)

```
from sklearn.svm import SVC
model = SVC(kernel='linear')
model.fit(x_train, y_train.values.ravel())
print(model.score(x_train, y_train))
print(model.score(x_test, y_test))
```

```
0.8367346938775511
0.865979381443299
```

➔ 先 import SVC，並在用 SVC()建立 model 時設定 kernal = linear。之後就把 data fit 進去訓練。用 score 函數分析準確度，因題目只說分析準確度，所以我把 training、testing data 都拿進去分析了。但真正重要的是 test。