

# ML\_Project

September 22, 2023

CS-C3240 - Machine Learning D Project

Authors: Aaron Gutierrez-Hernandez & Alexandre Cojot

Date created: 10-sep-2023

Last modified: 22-sep-2023

## 1 Import Libraries and Data

```
[1]: # Import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score, KFold

%matplotlib inline

# Read the .csv file containing the dataset
data = pd.read_csv('heart_data.csv')
data.head()
```

```
[1]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
0	52	1	0	125	212	0	1	168	0	1.0	2	
1	53	1	0	140	203	1	0	155	1	3.1	0	
2	70	1	0	145	174	0	1	125	1	2.6	0	
3	61	1	0	148	203	0	1	161	0	0.0	2	
4	62	0	0	138	294	1	1	106	0	1.9	1	

	ca	thal	target
0	2	3	0
1	0	3	0
2	0	3	0
3	1	3	0
4	3	2	0

## 2 Clean Data

```
[2]: data.info() # check basic dataframe's information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   age         1025 non-null   int64
 1   sex         1025 non-null   int64
 2   cp          1025 non-null   int64
 3   trestbps    1025 non-null   int64
 4   chol        1025 non-null   int64
 5   fbs         1025 non-null   int64
 6   restecg     1025 non-null   int64
 7   thalach     1025 non-null   int64
 8   exang       1025 non-null   int64
 9   oldpeak     1025 non-null   float64
10   slope       1025 non-null   int64
11   ca          1025 non-null   int64
12   thal        1025 non-null   int64
13   target      1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

```
[3]: data.duplicated().sum() # look if there are duplicates
```

```
[3]: 723
```

```
[4]: data.drop_duplicates(inplace=True) # drop druplicates
```

## 3 EDA

```
[5]: data.describe() # basic descriptive statistics of the dataframe
```

```
[5]:
```

	age	sex	cp	trestbps	chol	fbs \
count	302.00000	302.000000	302.000000	302.000000	302.000000	302.000000
mean	54.42053	0.682119	0.963576	131.602649	246.500000	0.149007
std	9.04797	0.466426	1.032044	17.563394	51.753489	0.356686
min	29.00000	0.000000	0.000000	94.000000	126.000000	0.000000
25%	48.00000	0.000000	0.000000	120.000000	211.000000	0.000000
50%	55.50000	1.000000	1.000000	130.000000	240.500000	0.000000
75%	61.00000	1.000000	2.000000	140.000000	274.750000	0.000000
max	77.00000	1.000000	3.000000	200.000000	564.000000	1.000000

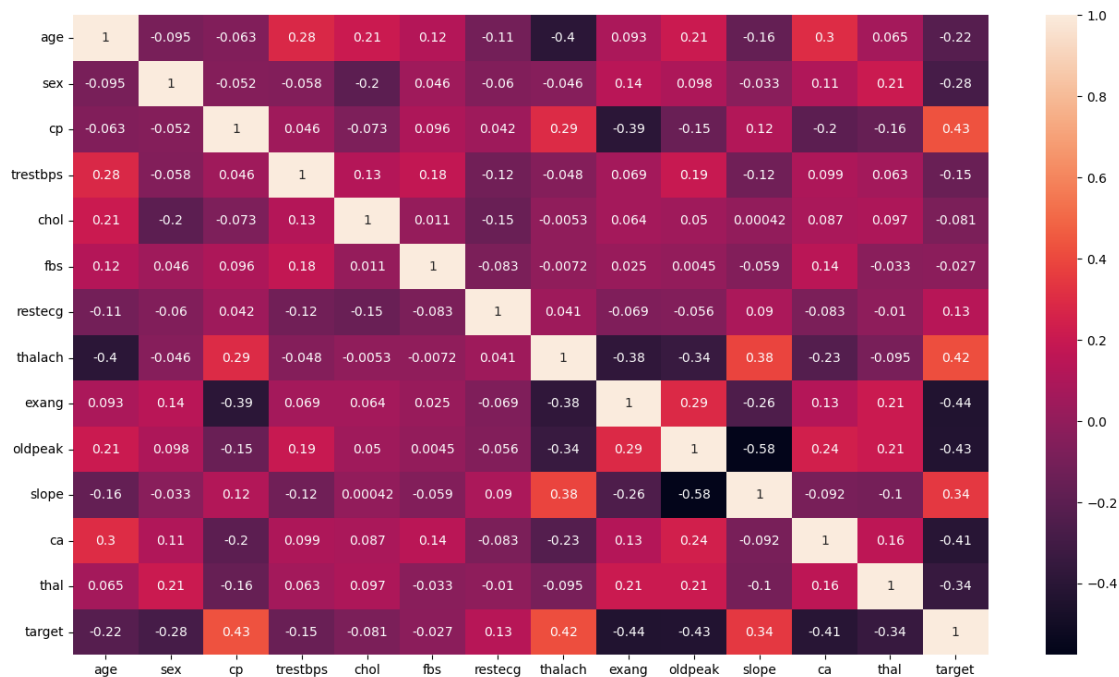
	restecg	thalach	exang	oldpeak	slope	ca \
--	---------	---------	-------	---------	-------	------

count	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000
mean	0.526490	149.569536	0.327815	1.043046	1.397351	0.718543
std	0.526027	22.903527	0.470196	1.161452	0.616274	1.006748
min	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	133.250000	0.000000	0.000000	1.000000	0.000000
50%	1.000000	152.500000	0.000000	0.800000	1.000000	0.000000
75%	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

	thal	target
count	302.000000	302.000000
mean	2.314570	0.543046
std	0.613026	0.498970
min	0.000000	0.000000
25%	2.000000	0.000000
50%	2.000000	1.000000
75%	3.000000	1.000000
max	3.000000	1.000000

```
[6]: plt.figure(figsize=(16,9))
sns.heatmap(data.corr(),annot=True) # see correlation between features and
↳ features with response variable
```

```
[6]: <AxesSubplot:>
```



```
[7]: plt.figure(figsize=(16,9))
sns.pairplot(data) # see distribution between each pair of features and
↳ features with response variable
```

```
[7]: <seaborn.axisgrid.PairGrid at 0x24582030ac0>
```

<Figure size 1600x900 with 0 Axes>

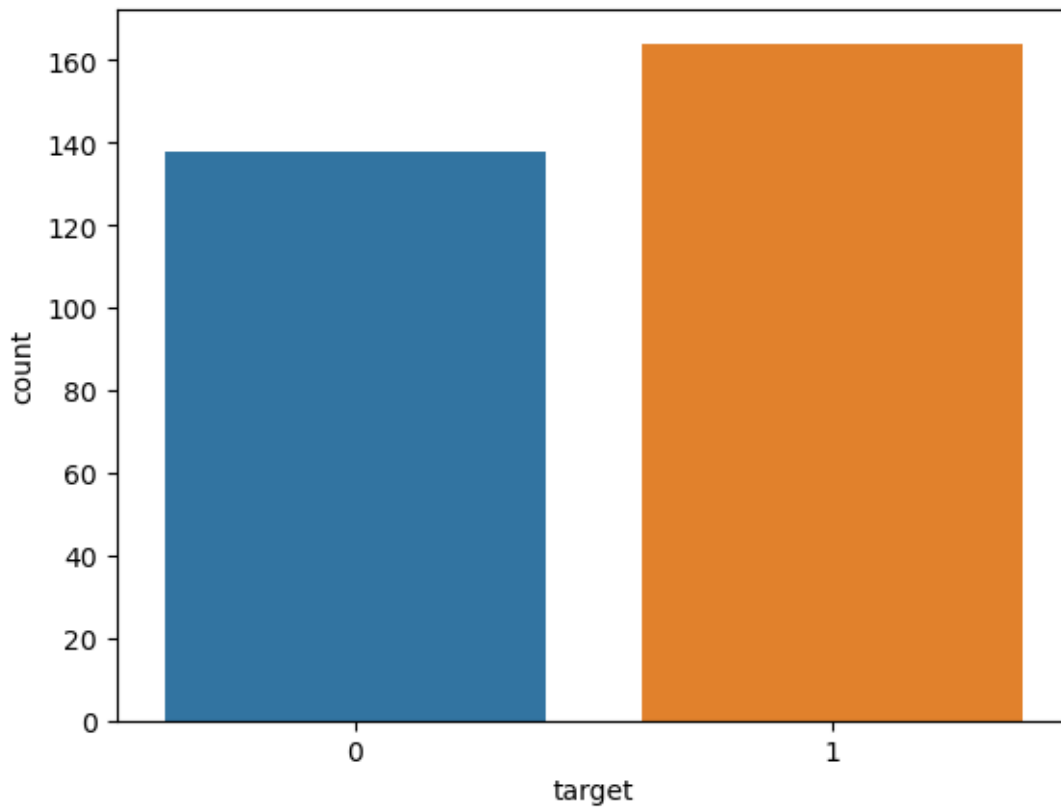


```
[8]: # Look at how the response variable is splitted to avoid fitting the model to
↳ just one value
target_counts = data['target'].value_counts() # count response variable values
↳ to see if they are not unbalanced
```

```
target_ratios = target_counts/len(data)    # get the same information in a
    ↪ratios format
print(target_counts)
print(target_ratios)
sns.countplot(x=data['target'])
```

```
1    164
0    138
Name: target, dtype: int64
1     0.543046
0     0.456954
Name: target, dtype: float64
```

```
[8]: <AxesSubplot:xlabel='target', ylabel='count'>
```



## 4 Prepare Data

```
[9]: # Make categorical data to the corresponding features
categorical_features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'thal', 'target'] #
    ↪columns containing categorical features
```

```
data[categorical_features] = data[categorical_features].astype('category') #  
    ↪converting dtype to categorical  
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 302 entries, 0 to 878  
Data columns (total 14 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   age         302 non-null    int64  
1   sex         302 non-null    category  
2   cp          302 non-null    category  
3   trestbps    302 non-null    int64  
4   chol        302 non-null    int64  
5   fbs         302 non-null    category  
6   restecg     302 non-null    category  
7   thalach     302 non-null    int64  
8   exang       302 non-null    category  
9   oldpeak     302 non-null    float64  
10  slope       302 non-null    int64  
11  ca          302 non-null    int64  
12  thal        302 non-null    category  
13  target      302 non-null    category  
dtypes: category(7), float64(1), int64(6)  
memory usage: 30.1 KB
```

```
[10]: # Standardize numerical features  
numeric_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'slope',  
    ↪'ca'] # columns containing numerical features  
scaler = StandardScaler() # initialize scaler  
scaler.fit(data[numeric_features]) # fit the scaler to the selected numeric  
    ↪columns  
data[numeric_features] = scaler.transform(data[numeric_features]) # standardize  
    ↪the numeric columns  
data.describe() # basic descriptive statistics of the dataframe
```

```
[10]:
```

	age	trestbps	chol	thalach	oldpeak	\
count	3.020000e+02	3.020000e+02	3.020000e+02	3.020000e+02	3.020000e+02	
mean	-2.721103e-16	-6.690748e-16	-4.411482e-18	-5.418771e-16	8.822964e-17	
std	1.001660e+00	1.001660e+00	1.001660e+00	1.001660e+00	1.001660e+00	
min	-2.814192e+00	-2.144521e+00	-2.332210e+00	-3.436149e+00	-8.995441e-01	
25%	-7.107878e-01	-6.617119e-01	-6.870826e-01	-7.137164e-01	-8.995441e-01	
50%	1.195033e-01	-9.140084e-02	-1.161266e-01	1.281605e-01	-2.096081e-01	
75%	7.283833e-01	4.789102e-01	5.467629e-01	7.185677e-01	4.803280e-01	
max	2.499671e+00	3.900776e+00	6.145034e+00	2.292987e+00	4.447460e+00	

```

slope
ca
```

```
count    3.020000e+02  3.020000e+02
mean     -1.158014e-16 -1.139633e-17
std       1.001660e+00  1.001660e+00
min       -2.271182e+00 -7.149112e-01
25%       -6.458337e-01 -7.149112e-01
50%       -6.458337e-01 -7.149112e-01
75%        9.795144e-01  2.800344e-01
max        9.795144e-01  3.264871e+00
```

```
[11]: X = data.drop('target',axis=1) # split the features from the labels
      y = data['target']           # split the labels from the features
      print(X.shape,y.shape)
```

```
(302, 13) (302,)
```

## 4.1 K-Fold Cross Validation

```
[12]: clf = LogisticRegression(solver='liblinear', penalty='l2') # initialize
      ↪proposed classifier
      kf = KFold(n_splits=5, shuffle=True, random_state=0)       # initialize
      ↪k-folds
      scrs = cross_val_score(clf, X, y, cv=kf, scoring='accuracy') # apply cv
      print(scrs)
      scrs.mean()
```

```
[0.83606557 0.73770492 0.81666667 0.86666667 0.85      ]
```

```
[12]: 0.8214207650273224
```

## 4.2 Split data

```
[13]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
      ↪random_state=0) # split data in training and testing sets
      print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)
```

```
(241, 13) (241,) (61, 13) (61,)
```

## 5 ML Model