Heart Disease Classification

AZZA ALI ELNAGGAR

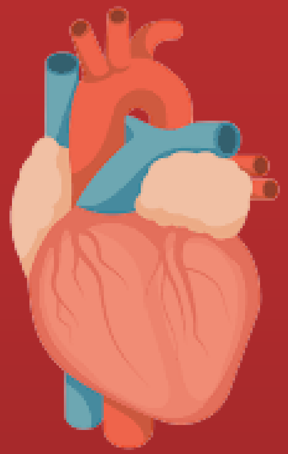
The problem

- This project will introduce some foundation Machine Learning and Data Science concepts by exploring the problem of heart disease **classification**.
- Cardiovascular disease (CVD) or heart disease is one of the leading causes of death in the United States. The Center for Disease Control Prevention estimates 647,000 deaths per year.

Steps

- Step 1: Basic understanding of data
- Step 2: Data Analysis and Insights
- Step 4: Data preparation
- Step 5 :Modelling and Evaluation

Heart Disease Dataset



Basic understanding of data

Data

- The data used to conduct this analysis is from a dataset compiled by four hospitals in Cleveland, Hungary, Switzerland, and VA Long Beach. The data is referred to as the UCI Heart Disease dataset. This dataset consists of 303 individuals with 14 attributes where 138 individuals are presented with no CVD and 165 individuals presented with CVD.

- Data sourse:Heart Disease Dataset | Kaggle

Attributes Information

- **AGE:** Age in years
- **SEX:** 1 = Male; 0 = Female
- **CP:** Chest Pain type
- **TRESTBPS:** Resting Blood Pressure (in mm Hg on Admission to the Hospital)
- **CHOL:** Serum Cholesterol in mg/dl
- **FPS:** Fasting Blood Sugar > 120 mg/dl (1 = True; 0 = False)
- **RESTECG:** Resting Electrocardiographic Results
- **THALACH:** Maximum Heart Rate Achieved
- **EXANG:** Exercise induced Angina (1 = yes; 0 = no)
- **OLDPEAK:** ST Depression induced by Exercise Relative to Rest
- **SLOPE:** The Slope of the Peak Exercise ST Segment
- **CA:** Number of Major Vessels (0-3) Colored by Flourosopy
- **THAL:** A blood disorder called Thalassemia (3 = Normal; 6 = Fixed Defect; 7 = Reversable Defect)
- **TARGET:** 1 or 0

Columns Data-types and information

```
1  df.info()
```

✓ 0.9s

```
<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
```

Data description (statistical methods)

```
1  df.describe().T
```

[6] ✓ 0.2s

...

	count	mean	std	min	25%	50%	75%	max
age	1025.0	54.434146	9.072290	29.0	48.0	56.0	61.0	77.0
sex	1025.0	0.695610	0.460373	0.0	0.0	1.0	1.0	1.0
cp	1025.0	0.942439	1.029641	0.0	0.0	1.0	2.0	3.0
trestbps	1025.0	131.611707	17.516718	94.0	120.0	130.0	140.0	200.0
chol	1025.0	246.000000	51.592510	126.0	211.0	240.0	275.0	564.0
fbs	1025.0	0.149268	0.356527	0.0	0.0	0.0	0.0	1.0
restecg	1025.0	0.529756	0.527878	0.0	0.0	1.0	1.0	2.0
thalach	1025.0	149.114146	23.005724	71.0	132.0	152.0	166.0	202.0
exang	1025.0	0.336585	0.472772	0.0	0.0	0.0	1.0	1.0
oldpeak	1025.0	1.071512	1.175053	0.0	0.0	0.8	1.8	6.2
slope	1025.0	1.385366	0.617755	0.0	1.0	1.0	2.0	2.0
ca	1025.0	0.754146	1.030798	0.0	0.0	0.0	1.0	4.0
thal	1025.0	2.323902	0.620660	0.0	2.0	2.0	3.0	3.0
target	1025.0	0.513171	0.500070	0.0	0.0	1.0	1.0	1.0

Categorical columns

```
cat_values = []
conti_values = []

for col in df.columns:
    if len(df[col].unique()) >= 10:
        conti_values.append(col)
    else:
        cat_values.append(col)

print("catageroy values: ", cat_values)
print("continous values: ", conti_values)
```

✓ 0.9s

```
catageroy values:  ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', 'target']
continous values:  ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
```

Check Nulls!

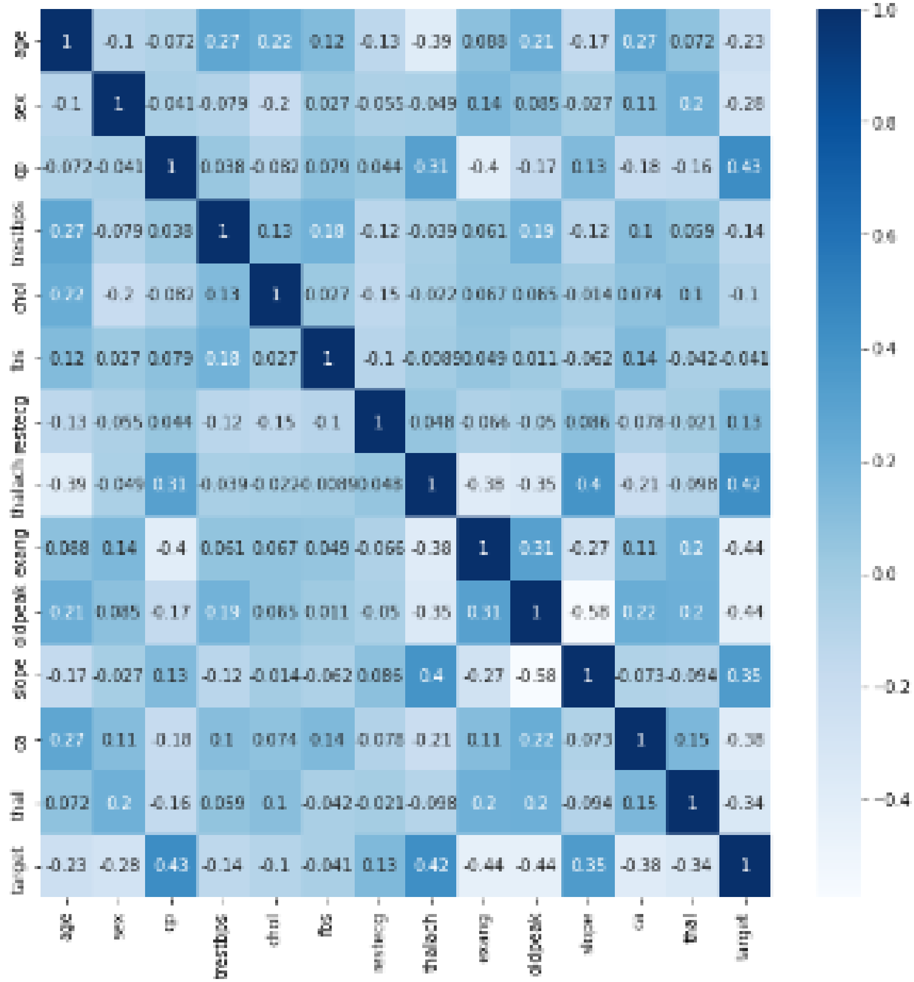
```
#Showing if there are any null values in the dataset
df.isnull().sum()
```

```
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
```

there is no nulls

Feature Selection

- No feature engineering was done because there were only 14 features, and each feature was treated as an independent variable from each other to see what features contributed to the prediction. Moreover, to ensure this point, I checked if there was any collinearity amongst the 14 attributes. From looking at Pearson's correlation, a strong *correlation* between variables did not exist



correlation

- From the heatmap in last slide There are Four features(cp, restecg, thalach, slope) are positively correlated with the target .Rest of the features are negatively correlated with target but none of them found to be strongly correlated with target.
- the greater amount of chest pain results in a greater chance of having heart disease so the positive correlation between chest pain (cp) & target This makes sense .
- the negative correlation between exercise induced angina (exang) & our predictor. This makes sense because when you excercise, your heart requires more blood, but narrowed arteries slow down blood flow.

Data Analysis and Insights

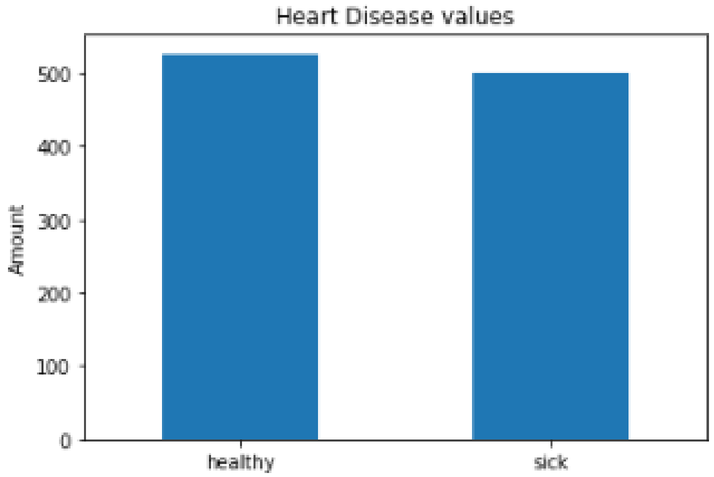
We have a lot of questions that we need to answer it by analyze the data .

1 - how many people have heart disease (label = 1)

```
df.target.value_counts()
```

```
1    526
0    499
Name: target, dtype: int64
```

```
#plotting bar chart.
fig = df.target.value_counts().plot(kind = 'bar')
fig.set_xticklabels(labels=['healthy','sick'], rotation=0)
plt.title("Heart Disease values")
plt.ylabel("Amount")
```



2- WHICH SEX HAS MOST HEART DISEASE?

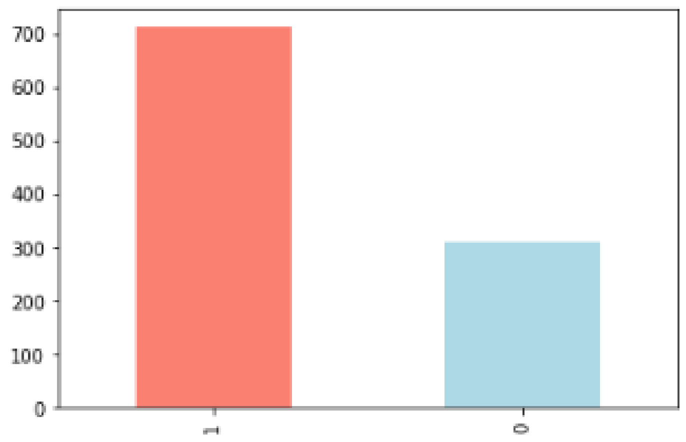
```
1  df['sex'].value_counts()
```

```
1     713
0     312
Name: sex, dtype: int64
```

==> 713 records are of males and 312 records are of females

```
1  df['sex'].value_counts().plot(kind='bar', color=['salmon', 'lightblue'])
```

```
<AxesSubplot:>
```



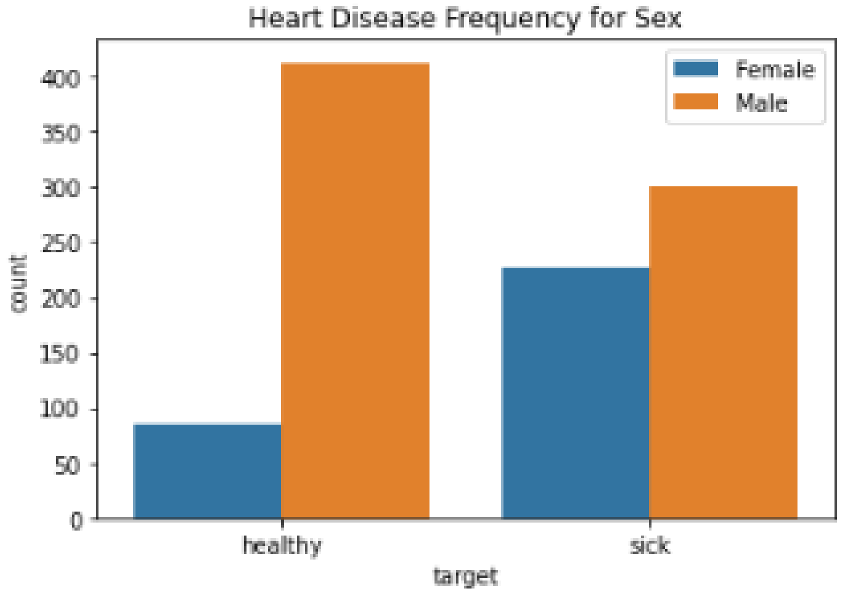
3- relation between disease and gender in our data

```
pd.crosstab(df.target, df.sex)
```

sex	0	1
target		
0	86	413
1	226	300

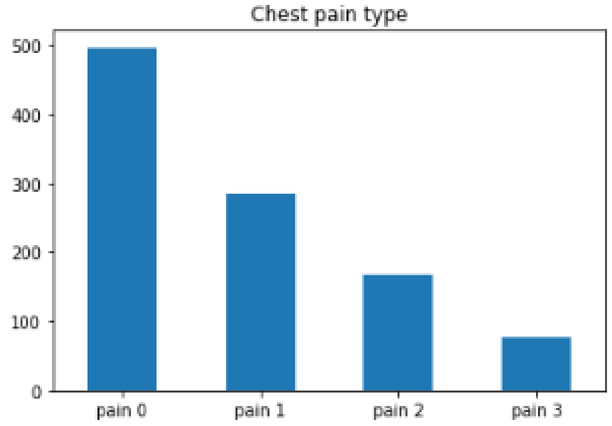
1 > MALE , 0 > FMALE 1 > HAVE DISEASE , 0 > DOEN'T HAVE DISEASE

```
fig = sns.countplot(x = 'target', data = df, hue = 'sex')
fig.set_xticklabels(labels=['healthy','sick'], rotation=0)
plt.legend(['Female', 'Male'])
plt.title("Heart Disease Frequency for Sex");
```

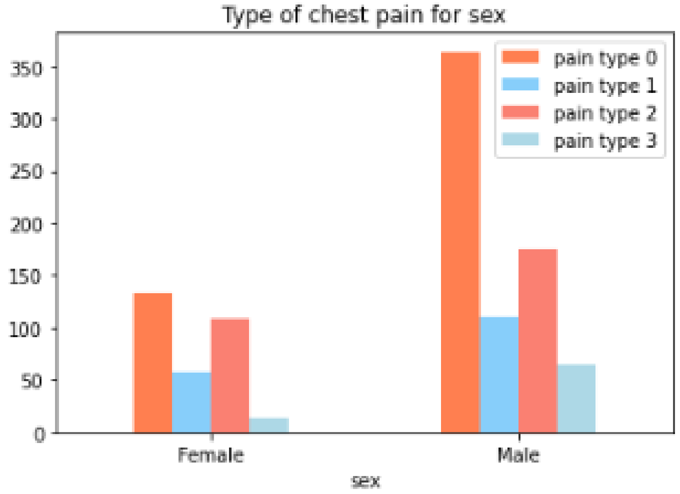


4 - which sex has which type of chest pain most?

```
#plotting a bar chart
fig = df.cp.value_counts().plot(kind = 'bar')
fig.set_xticklabels(labels=['pain 0', 'pain 1', 'pain 2', 'pain 3'], rotation=0)
plt.title('Chest pain type');
```




```
fig = pd.crosstab(df.sex, df.cp).plot(kind = 'bar',
                    color = ['coral', 'lightskyblue','salmon', 'lightblue'])
plt.title('Type of chest pain for sex')
fig.set_xticklabels(labels=['Female', 'Male'], rotation=0)
plt.legend(['pain type 0', 'pain type 1', 'pain type 2', 'pain type 3']);
```

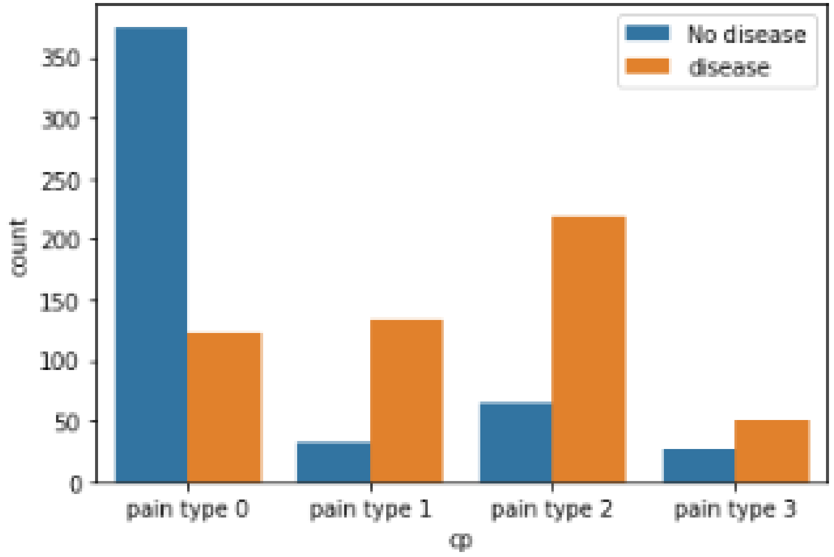


5 - which chest pain are most Pron to have heart disease?

```
fig = sns.countplot(x = 'cp', data = df, hue = 'target')
fig.set_xticklabels(labels=['pain type 0', 'pain type 1', 'pain type 2', 'pain type 3'], rotation=0)
plt.legend(['No disease', 'disease']);
```

[16]

...



- cp: chest pain
  0: Typical angina: chest pain related decrease blood supply to the heart
  1: Atypical angina: chest pain not related to heart
  2: Non-anginal pain: typically esophageal spasms (non-heart related)
  3: Asymptomatic: chest pain not showing signs of disease

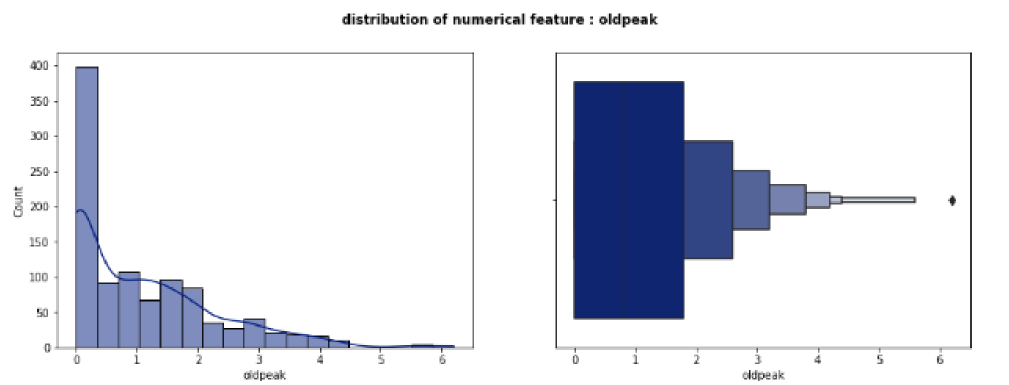
Skewness

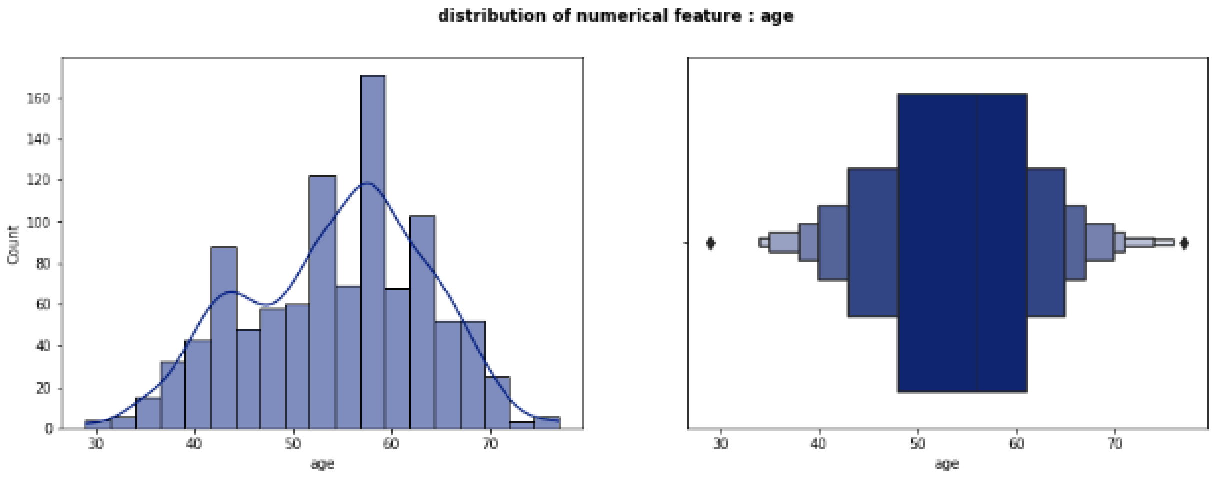
If the skewness is between -0.5 and 0.5, the data are fairly symmetrical

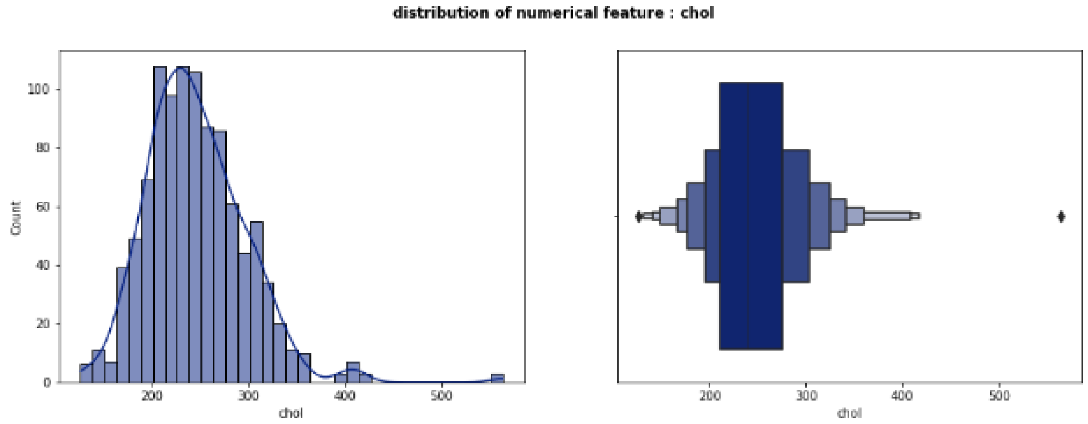
```
1  ## skewness
2  df.skew()
```

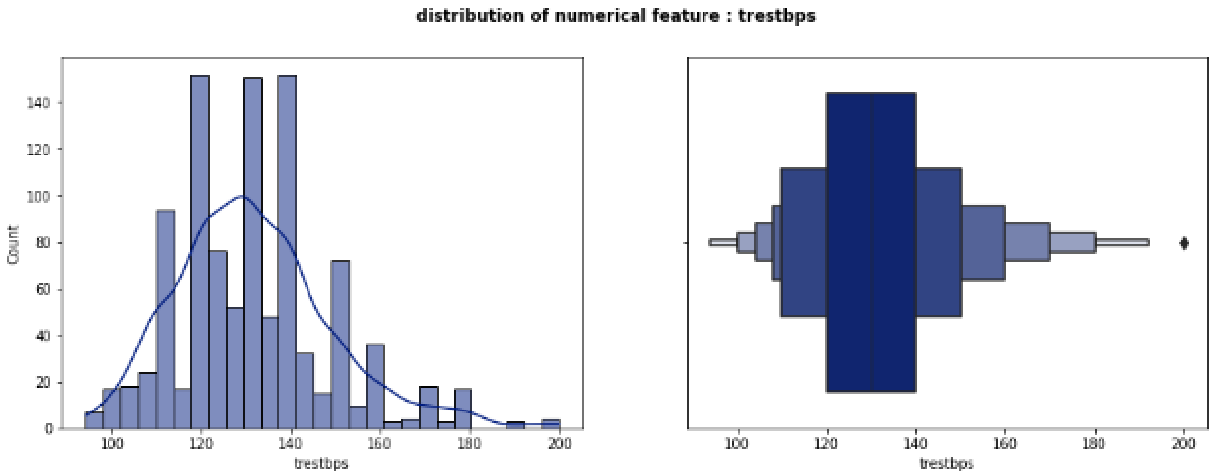
```
age        -0.248866
sex        -0.851449
cp          0.529455
trestbps    0.739768
chol        1.074073
fbs         1.971339
restecg     0.180440
thalach    -0.513777
exang       0.692655
oldpeak     1.210899
slope      -0.479134
ca          1.261189
thal       -0.524390
target     -0.052778
dtype: float64
```

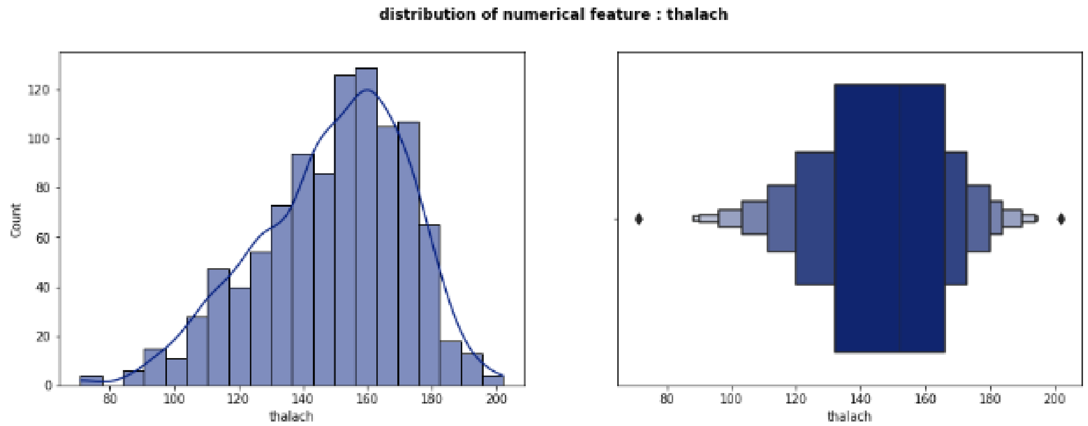
distribution of numerical features



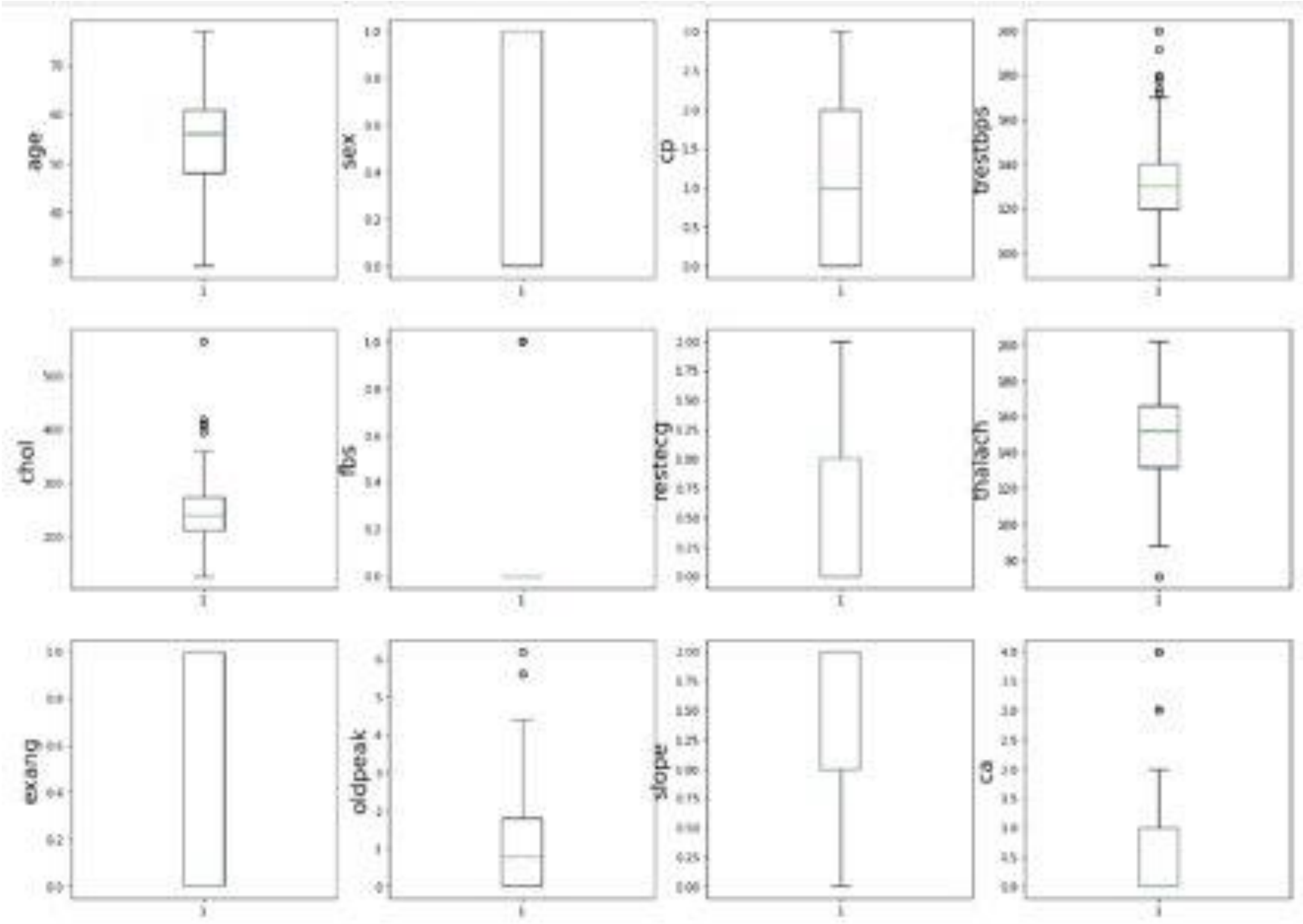








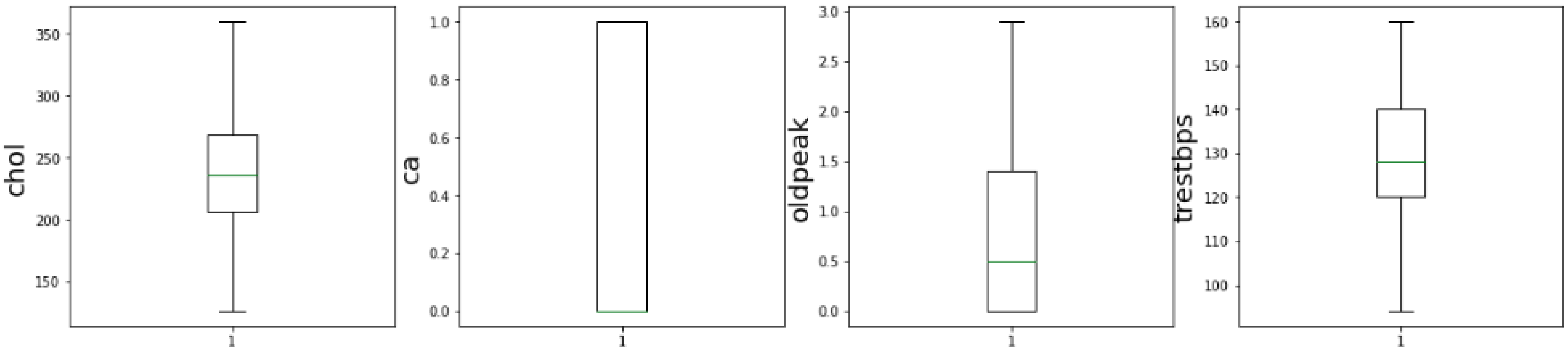
Check outliers

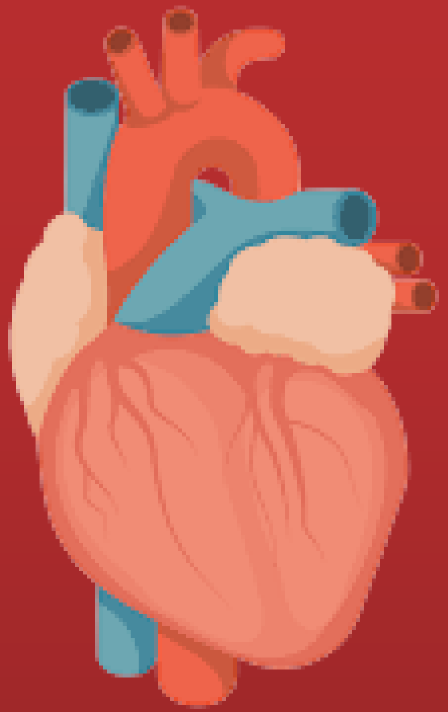


we can see that there are many outliers in the column trestbps,chol,ca,thalach,oldpeak

remove outliers

```
#Removing outliers

outlier=['chol','ca','oldpeak','trestbps']
for i in outlier:
    q3=df[i].quantile(q=0.75)
    q1=df[i].quantile(q=0.25)
    IQR=q3-q1
    IQR_lower_limit=int(q1-1.5*IQR)
    IQR_upper_limit=int(q3+1.5*IQR)
    k=df[df[i]>IQR_upper_limit]
    df=df[df[i]<IQR_upper_limit]
```





Modelling and Evaluation

- We will try 3 models :
  - Random Cut Forest
  - Support Vector Machine
  - Logistic Regression

Split data

split the target column

```
x = df.drop(columns = ["target"])
y = df["target"]
```

[25] ✓ 0.9s

split the data into train and test set

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20, random_state = 1234)
```

[26] ✓ 0.1s

Scaling data

You want to scale data when you're using methods based on measures of how far apart data points are, like support vector machines (SVM), logistic regression .

scaling data

```
x = df.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled)
```

✓ 0.6s

```
x_scaled_train, x_scaled_test, y_train, y_test = train_test_split(x_scaled, y, test_size = 0.20, random_state = 1234)
```

✓ 0.4s

Random Cut Forest

```
RF= RandomForestClassifier()
RF.fit(x_train,y_train)
y_pred = RF.predict(x_test)
rf_accuracy = accuracy_score(y_test,y_pred)
c_mat= confusion_matrix(y_test,y_pred)
print(classification_report(y_test,y_pred))
```

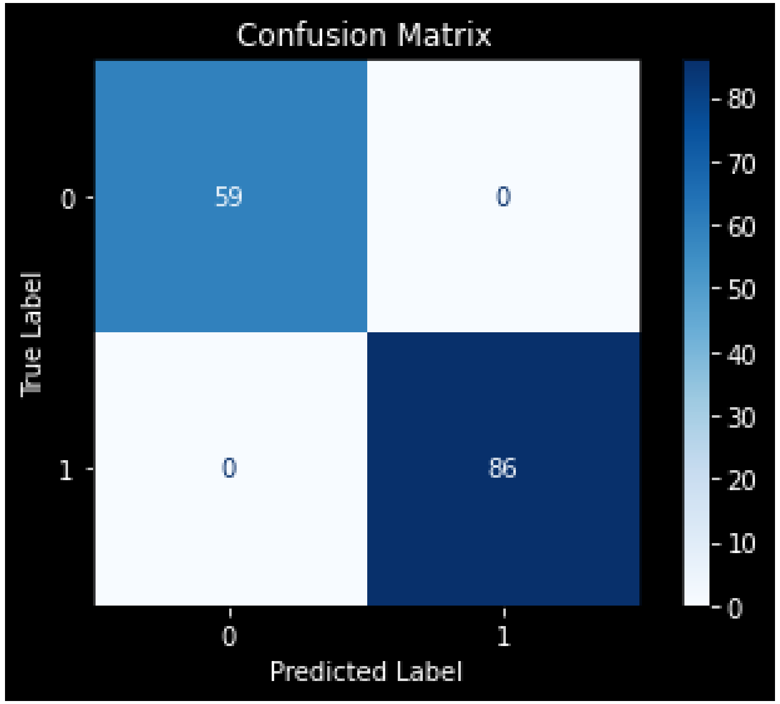
✓ 0.5s

	precision	recall	f1-score	support
0	1.00	1.00	1.00	59
1	1.00	1.00	1.00	86
accuracy			1.00	145
macro avg	1.00	1.00	1.00	145
weighted avg	1.00	1.00	1.00	145

```
print("Accuracy: %.2f%%" % (rf_accuracy*100))
```

✓ 0.5s

Accuracy: 100.00%



Support Vector Machine

```
model= SVC()
model.fit(x_scaled_train,y_train)
y_pred = model.predict(x_scaled_test)
sv_accuracy = accuracy_score(y_test,y_pred)
confusion_matrix = confusion_matrix(y_test,y_pred)
print(classification_report(y_test,y_pred))
```

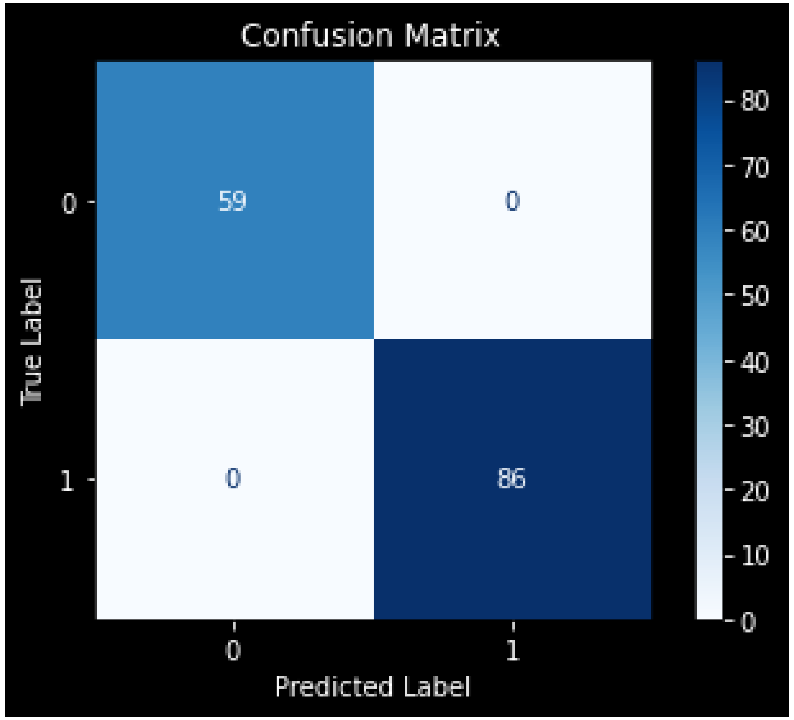
[32] ✓ 0.7s

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        59
           1       1.00      1.00      1.00        86

    accuracy                           1.00       145
   macro avg       1.00      1.00      1.00       145
weighted avg       1.00      1.00      1.00       145
```

```
print("Accuracy: %.2f%%" % (sv_accuracy*100))
```

[33] ✓ 0.4s

```
Accuracy: 100.00%
```



Logistic regression

```
log_reg = LogisticRegression()
log_reg.fit(x_scaled_train, y_train)
y_pred = log_reg.predict(x_scaled_test)
lr_acc = accuracy_score(y_test, y_pred)
```

[35] ✓ 0.7s

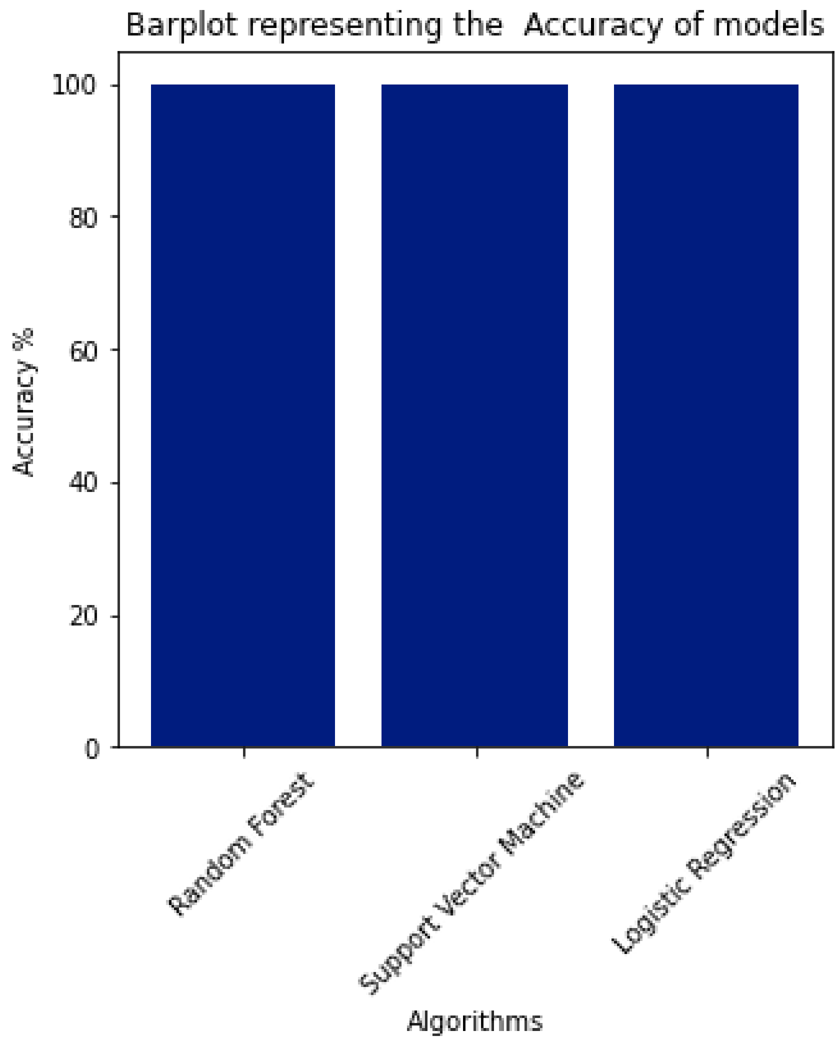
```
print("Accuracy: %.2f%%" % (lr_acc*100))
```

[36] ✓ 0.4s

```
Accuracy: 100.00%
```

Evaluation

	Model	Accuracy
0	Random Forest	100.0
1	Support Vector Machine	100.0
2	Logistic Regression	100.0



Thanks...