MACHINE LEARNING
PROJECT PHASE 2

1. PROBLEM DEFINITION

It is essential to create a system that can forecast heart diseases precisely and effectively given the rise in fatalities caused by heart diseases. Machine learning can be used to detect whether a person is suffering from a cardiovascular disease by considering certain attributes like chest pain, cholesterol level, age of the person and some other attributes.

2. DATASETS

Our datasets consists of the following attributes to predict the output:

- **age** - age in years
- **sex** - (1 = male; 0 = female)
    a. 0: Typical angina: chest pain related decrease blood supply to the heart
    b. 1: Atypical angina: chest pain not related to heart
    c. 2: Non-anginal pain: typically esophageal spasms (non heart related)
    d. 3: Asymptomatic: chest pain not showing signs of disease
- **cp** - chest pain type
- **trestbps** - resting blood pressure (in mm Hg on admission to the hospital)
    a. Anything above 130-140 is typically cause for con
- **chol** - serum cholestoral in mg/dl
    a. serum = LDL + HDL + .2 * triglycerides
    b. above 200 is cause for concern
- **fbs** - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
    a. '>126' mg/dL signals diabetes

- **restecg** - resting electrocardiographic results
    a. 0: Nothing to note
    b. 1: ST-T Wave abnormality
        i. can range from mild symptoms to severe problems
        ii. signals non-normal heart beat
    c. 2: Possible or definite left ventricular hypertrophy
        i. Enlarged heart's main pumping chamber

- **thalach** - maximum heart rate achieved
- **exang** - exercise induced angina (1 = yes; 0 = no)
- **oldpeak** - ST depression induced by exercise relative to rest
    a. looks at stress of heart during excercise
    b. unhealthy heart will stress more
- **slope** - the slope of the peak exercise ST segment
    a. 0: Upsloping: better heart rate with excercise (uncommon)
    b. 1: Flatsloping: minimal change (typical healthy heart)
    c. 2: Downslopins: signs of unhealthy heart
- **ca** - number of major vessels (0-3) colored by flourosopy
    a. colored vessel means the doctor can see the blood passing through
    b. the more blood movement the better (no clots)
- **thal** - thalium stress result
    a. 1,3: normal
    b. 6: fixed defect: used to be defect but ok now
    c. 7: reversable defect: no proper blood movement when excercising
- **target** - have disease or not (1=yes, 0=no) (= the predicted attribute)

3. PREPARE DATA

DATA PREPROCESSING:

IMPORTING NECESSARY LIBRARIES FOR LOADING AND VISUALIZING THE DATASET:

In [47]:
```
# Regular EDA and plotting libraries
import numpy as np # np is short for numpy
import pandas as pd # pandas is so commonly used, it's shortened to pd
import matplotlib.pyplot as plt
import time
import seaborn as sns # seaborn gets shortened to sns

# We want our plots to appear in the notebook
%matplotlib inline

## Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

## Model evaluators
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import plot_roc_curve, accuracy_score
```

LOADING THE DATASET:

```
data=pd.read_csv('heart_disease_data.csv')
```

```
data.head(5)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Exploration (exploratory data analysis or EDA)

Once you've imported a dataset, the next step is to explore. There's no set way of doing this. But what you should be trying to do is become more and more familiar with the dataset.

Compare different columns to each other, compare them to the target variable. Refer back to your **data dictionary** and remind yourself of what different columns mean.

Your goal is to become a subject matter expert on the dataset you're working with. So if someone asks you a question about it, you can give them an explanation and when you start building models, you can sound check them to make sure they're not performing too well (**overfitting**) or why they might be performing poorly (**underfitting**).

Since EDA has no real set methodolgy, the following is a short check list you might want to walk through:

1. What question(s) are you trying to solve (or prove wrong)?

2. What kind of data do you have and how do you treat different types?

3. What's missing from the data and how do you deal with it?

4. Where are the outliers and why should you care about them?

5. How can you add, change or remove features to get more out of your data?

Once of the quickest and easiest ways to check your data is with the `head()` function. Calling it on any dataframe will print the top 5 rows, `tail()` calls the bottom 5. You can also pass a number to them like `head(10)` to show the top 10 rows.

'value_counts()' allows you to show how many times each of the values of a categorical column appear.

```
data.target.value_counts()
```

```
1    165
0    138
Name: target, dtype: int64
```

Since these two values are close to even, our `target` column can be considered "balanced". An "unbalanced" target column, meaning some classes have far more samples, can be harder to model than a balanced set. Ideally, all your target classes have the same number of samples.

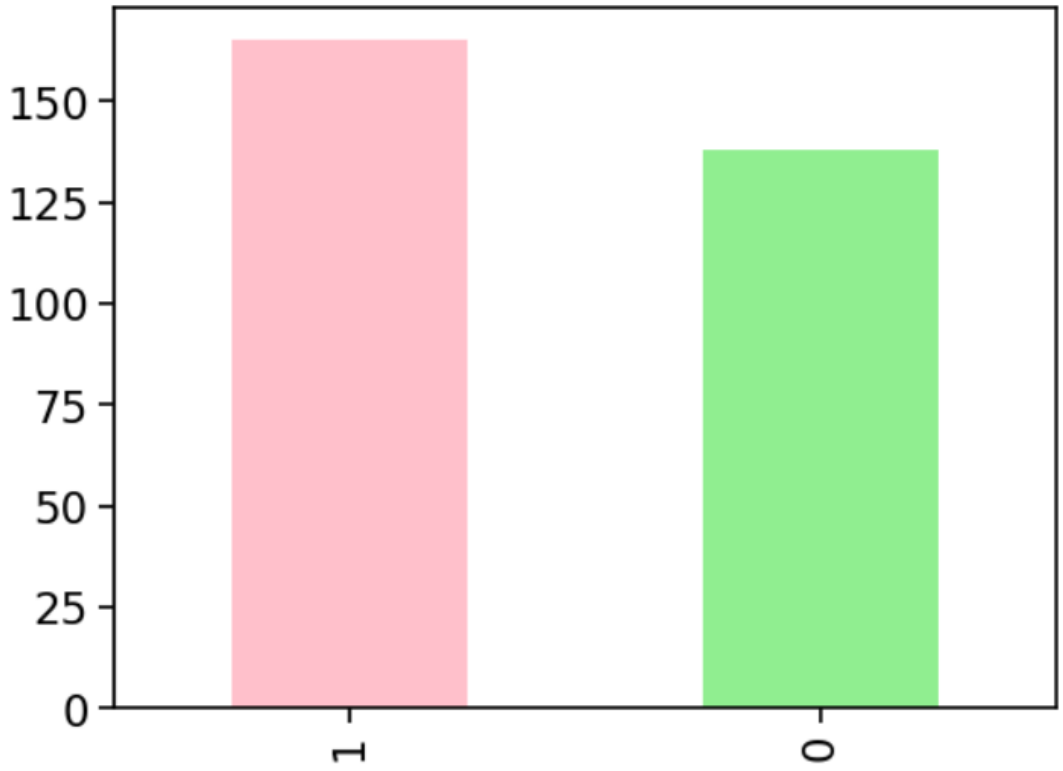
If you'd prefer these values in percentages, 'value_counts()' takes a parameter, `normalize` which can be set to true.

```
data.target.value_counts(normalize=True)
```

```
1    0.544554
0    0.455446
Name: target, dtype: float64
```

We can plot the target column value counts by calling the `plot()` function and telling it what kind of plot we'd like, in this case, bar is good.

```
data.target.value_counts().plot(kind="bar", color=["pink", "lightgreen"]);
```



‘data.info()’ shows a quick insight to the number of missing values you have and what type of data your working with.

```
data.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        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Another way to get some quick insights on your dataframe is to use ‘data.describe()’. It shows a range of different metrics about your numerical columns such as mean, max and standard deviation.

```
data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.3
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.6
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.0
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.0
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.0
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.0

DATA STANDARDIZATION:

SPLITTING THE DATA FOR STANDARDIZATION:

```
X=data.drop(columns='target',axis=1)
y=data['target']
print(X,y)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   3       145   233    1        0      150      0      2.3
1     37    1   2       130   250    0        1      187      0      3.5
2     41    0   1       130   204    0        0      172      0      1.4
3     56    1   1       120   236    0        1      178      0      0.8
4     57    0   0       120   354    0        1      163      1      0.6
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal
0        0   0     1
1        0   0     2
2        2   0     2
3        2   0     2
4        2   0     2
..     ...  ..   ...
298      1   0     3
299      1   0     3
300      1   2     3
301      1   1     3
302      1   1     2

[303 rows x 13 columns] 0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

STANDARDIZING THE SPLIT DATA USING THE FIT TRANSFORM METHOD:

```
scaler=StandardScaler()
standardized_data=scaler.fit_transform(X)
print(standardized_data)
```

```
[[ 0.9521966   0.68100522  1.97312292 ... -2.27457861 -0.71442887
  -2.14887271]
 [-1.91531289  0.68100522  1.00257707 ... -2.27457861 -0.71442887
  -0.51292188]
 [-1.47415758 -1.46841752  0.03203122 ...  0.97635214 -0.71442887
  -0.51292188]
 ...
 [ 1.50364073  0.68100522 -0.93851463 ... -0.64911323  1.24459328
   1.12302895]
 [ 0.29046364  0.68100522 -0.93851463 ... -0.64911323  0.26508221
   1.12302895]
 [ 0.29046364 -1.46841752  0.03203122 ... -0.64911323  0.26508221
  -0.51292188]]
```

This standardized data is then stored into the X in an array format. In further steps (for summarization and visualization).

We will use the same dataset i.e., the table stored in the variable “data” itself.

CHECKING MISSING DATA:

```
data.isnull()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	False	False	False	False	False	False	False	False	False	False	False	False	False	False
299	False	False	False	False	False	False	False	False	False	False	False	False	False	False
300	False	False	False	False	False	False	False	False	False	False	False	False	False	False
301	False	False	False	False	False	False	False	False	False	False	False	False	False	False
302	False	False	False	False	False	False	False	False	False	False	False	False	False	False

303 rows × 14 columns

```
data.isna().sum().sum()
```

```
0
```

The dataset does not contain any null values.

UNDERSTANDING THE DATA:

```
data.head(5)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
data.tail(5)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

```
data.shape
```

```
(303, 14)
```

```
data.dtypes
```

```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

```
data.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        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

DATA SUMMARIZATION:

A way of understanding the given data is by using describe() which shows a range of different metrics about your

numerical columns such as mean, max and standard deviation. This helps us get a statistical insight.

```
data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0(
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.31
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.61
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.0(
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.0(
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.0(
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.0(
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.0(

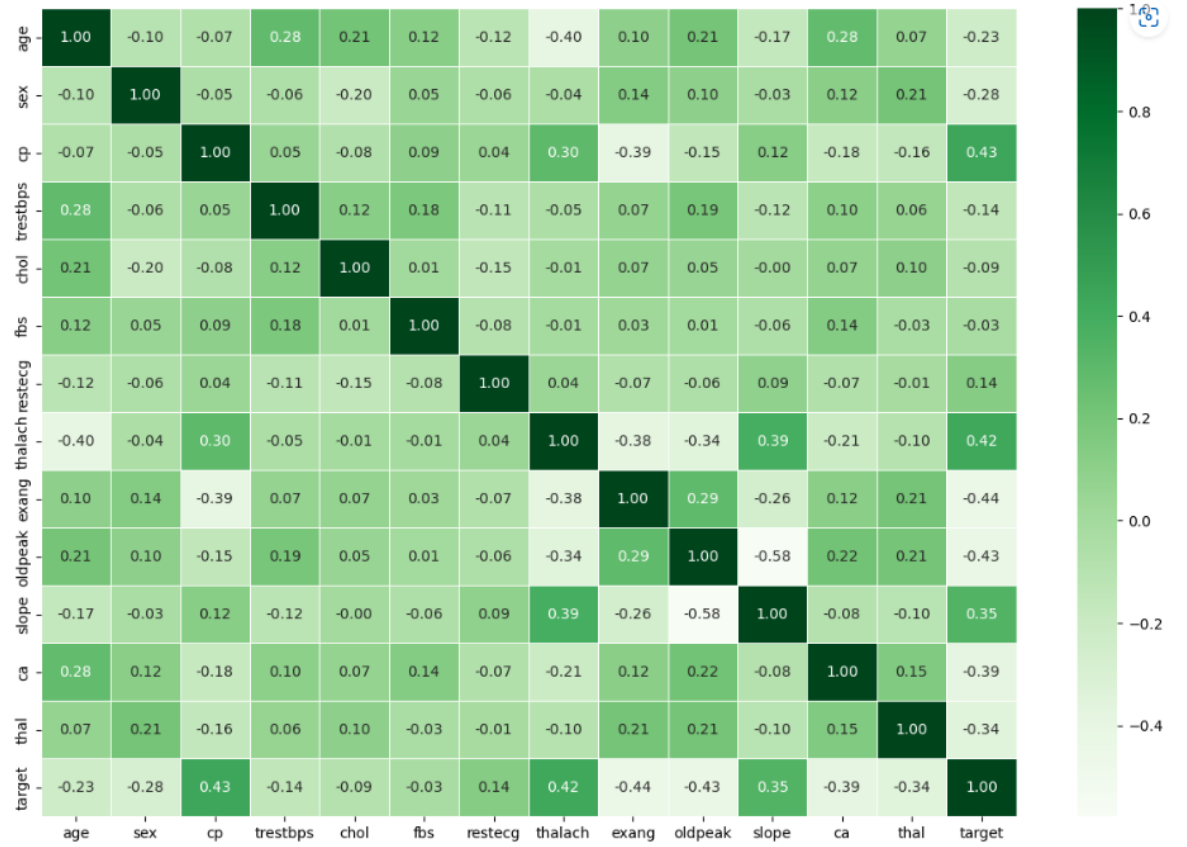
FINDING THE CORRELATION BETWEEN INDEPENDENT VARIABLES USING CORRELATION MATRIX:

```
corr_matrix = data.corr()
corr_matrix
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276326	0.068001	-0.225439
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118261	0.210041	-0.280937
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181053	-0.161736	0.433798
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101389	0.062210	-0.144931
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070511	0.098803	-0.085239
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137979	-0.032019	-0.028046
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072042	-0.011981	0.137230
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213177	-0.096439	0.421741
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115739	0.206754	-0.436757
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222682	0.210244	-0.430696
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080155	-0.104764	0.345877
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000000	0.151832	-0.391724
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151832	1.000000	-0.344029
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.391724	-0.344029	1.000000

FINDING A WAY TO REPRESENT IT VISUALLY:

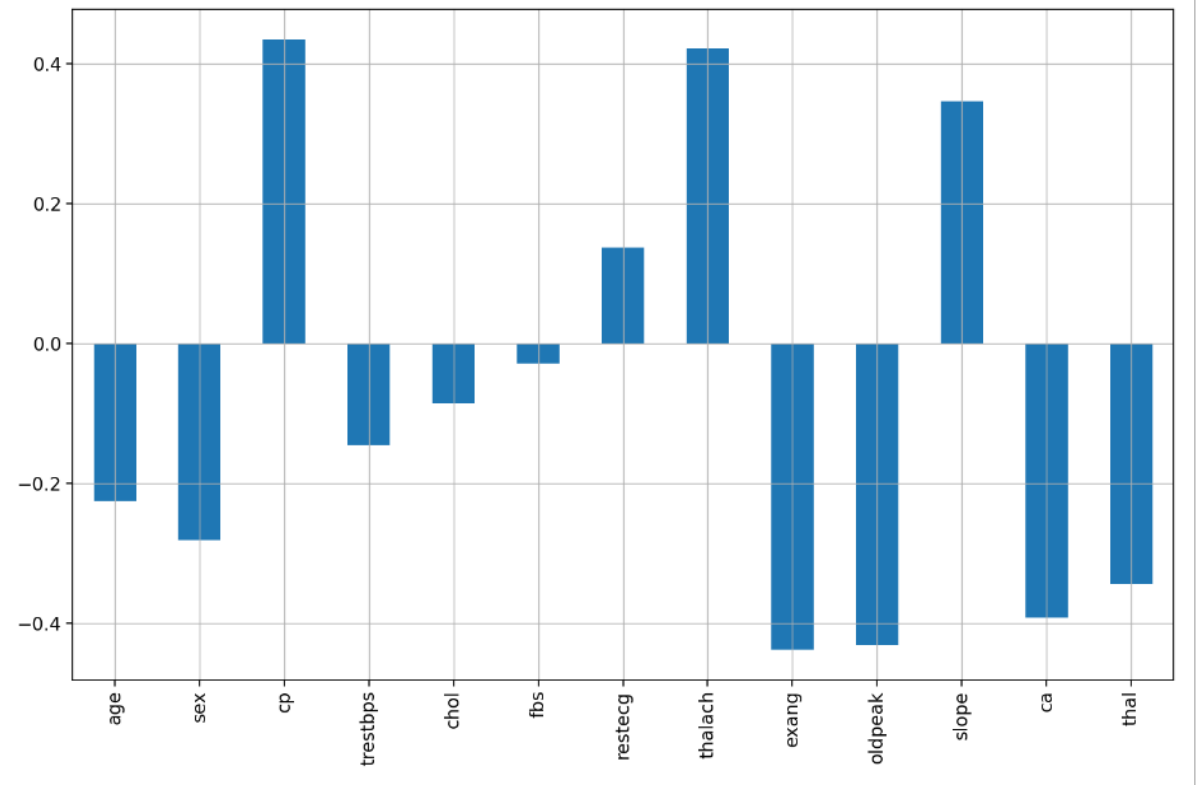
```
corr_matrix = data.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix,
            annot=True,
            linewidths=0.5,
            fmt= ".2f",
            cmap="Greens");
```



Here, a higher positive value means a potential positive correlation (increase) and a higher negative value means a potential negative correlation (decrease).

CORRELATION OF ALL FEATURES WITH THE TARGET VARIABLE:

```
sns.set_context('notebook',font_scale = 1.5)
h=data.drop('target', axis=1)
h.corrwith(data['target']).plot(kind='bar', grid=True, figsize=(15, 10))
plt.tight_layout()
```



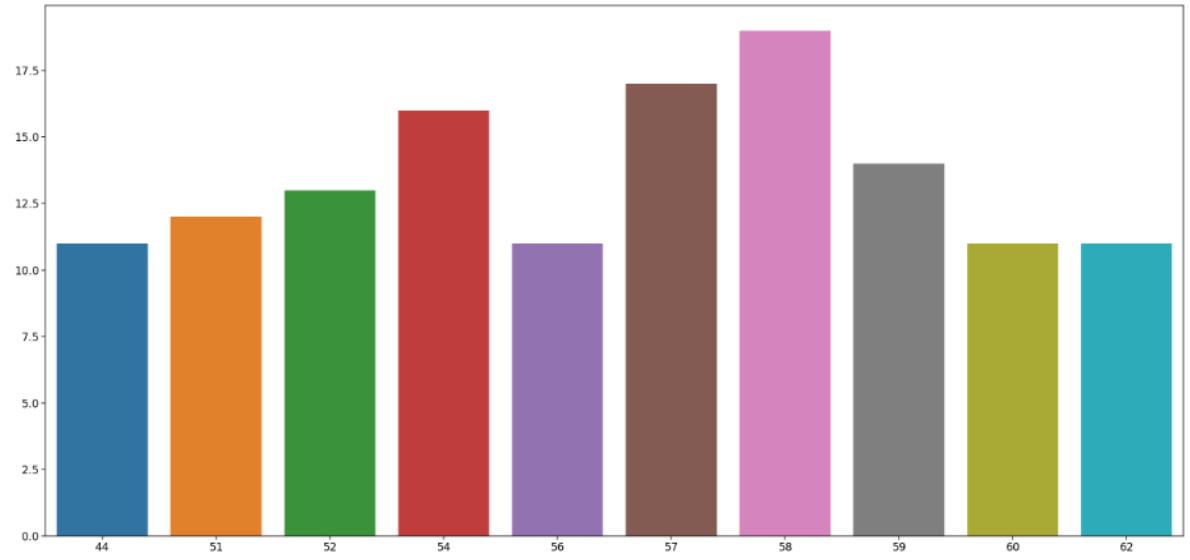
The target feature has negative correlation with attributes like “age”, “sex”, “trestbps”, “chol”, “fbs”, “exang”, “oldpeak”, “ca”, “thal”. All other features have a positive correlation with the target value.

DATA VISUALIZATION:

Analyzing each feature by visualizing for a better representation.

PLOTTING AGE WITH THE COUNT OF EACH 10 CATEGORIES:

```
plt.figure(figsize=(25,12))
sns.set_context('notebook',font_scale = 1.5)
sns.barplot(x=data['age'].value_counts()[:10].index,y=data['age'].value_counts()[:10].values)
plt.tight_layout()
```



From the graph below we can understand that age 58 has the highest frequency. From this we can understand that in this given dataset the patients near to age 58 are more likely to appear in this study.

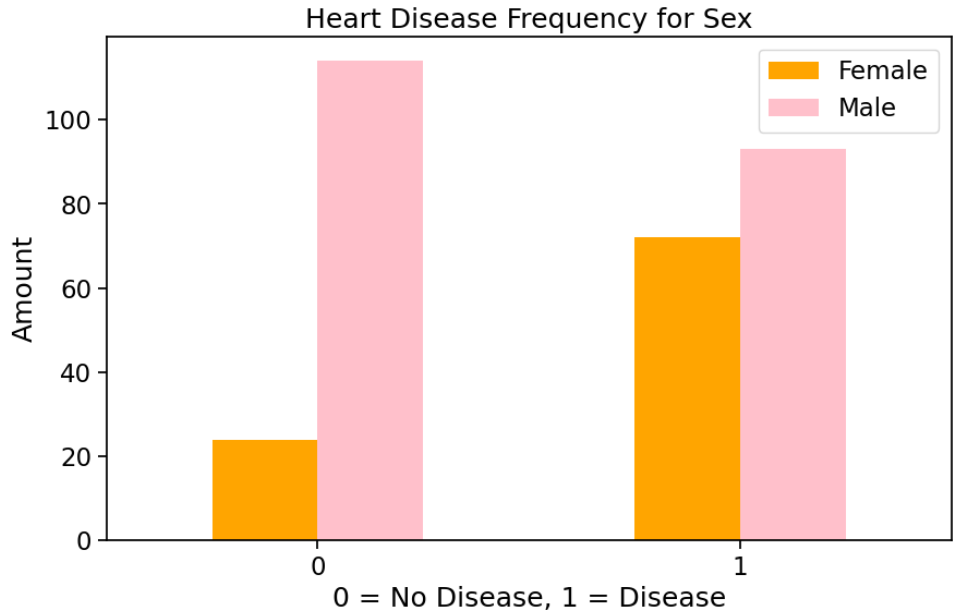
CHECKING THE RANGE OF AGES OF THE PATIENTS:

```
minAge=min(data['age'])
maxAge=max(data['age'])
meanAge=data['age'].mean()
print('Minimum Age :',minAge)
print('Maximum Age :',maxAge)
print('Mean Age :',meanAge)
```

```
Minimum Age : 29
Maximum Age : 77
Mean Age : 54.366336633663366
```

PLOTTING THE RELATION B/W SEX AND SICK PATIENTS:

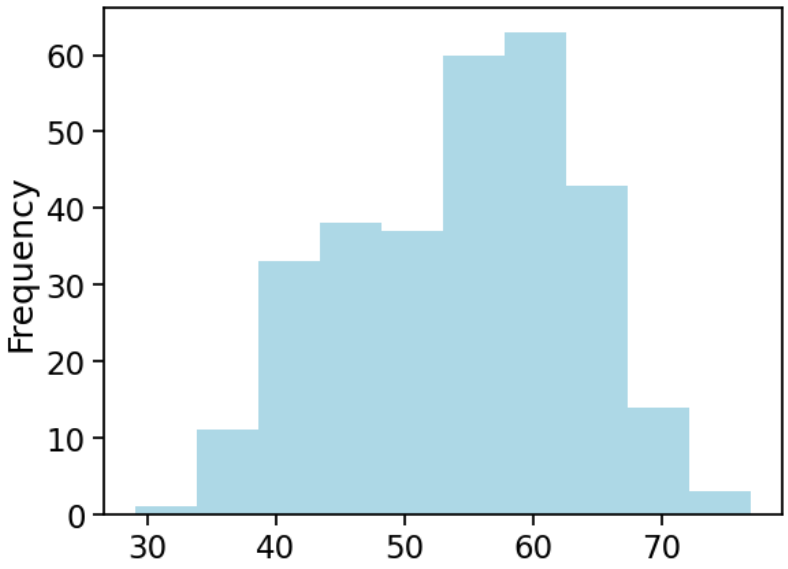
```
pd.crosstab(data.target, data.sex).plot(kind="bar", figsize=(10,6), color=["orange", "pink"])
plt.title("Heart Disease Frequency for Sex")
plt.xlabel("0 = No Disease, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female", "Male"])
plt.xticks(rotation=0);
```



Here, we can see the number of females with no disease is low. This shows that in this dataset the proportion of women who have heart disease is much higher than that of men. But the number of men who have heart disease is more because there are more men in this survey than women.

PLOTTING A HISTOGRAM FOR AGE:

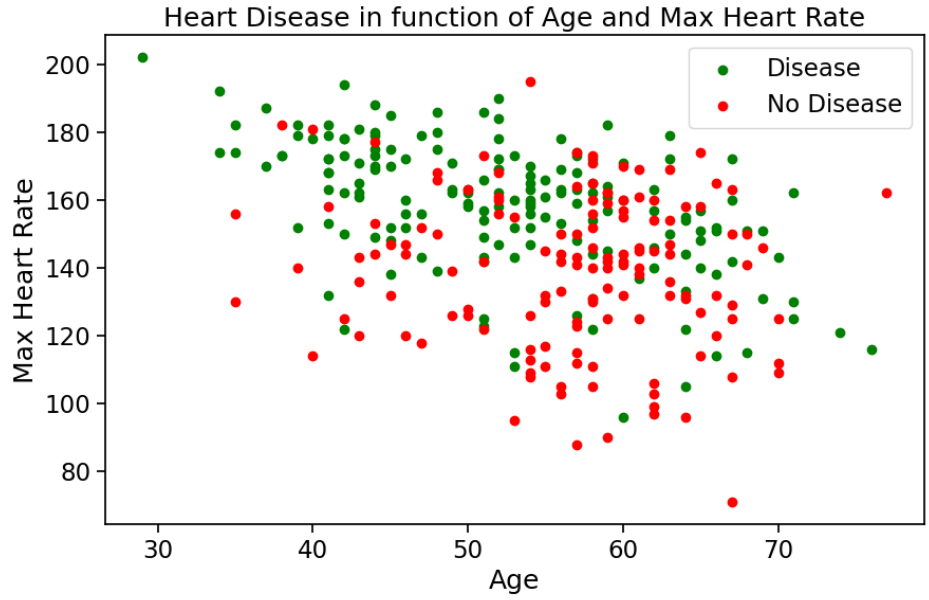
```
data.age.plot.hist(color='lightblue');
```



People near the age category 50-60 is more in this survey.

PLOTTING A SCATTERPLOT WITH MAX HEART RATE AND AGE:

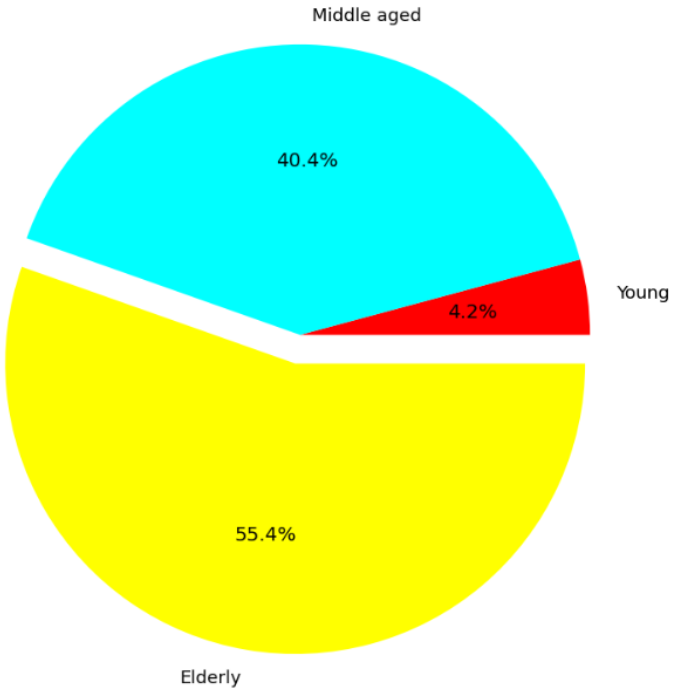
```
plt.figure(figsize=(10,6))
plt.scatter(data.age[data.target==1],
            data.thalach[data.target==1],
            c="green")
plt.scatter(data.age[data.target==0],
            data.thalach[data.target==0],
            c="red")
plt.title("Heart Disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.legend(["Disease", "No Disease"])
plt.ylabel("Max Heart Rate");
```



Here, it seems the younger someone is, the higher their max heart rate (dots are higher on the left of the graph) and the older someone is, the more green dots there are. But this may be because there are more dots all together on the right side of the graph (older participants).

PLOTTING A PIE CHART ON AGE:

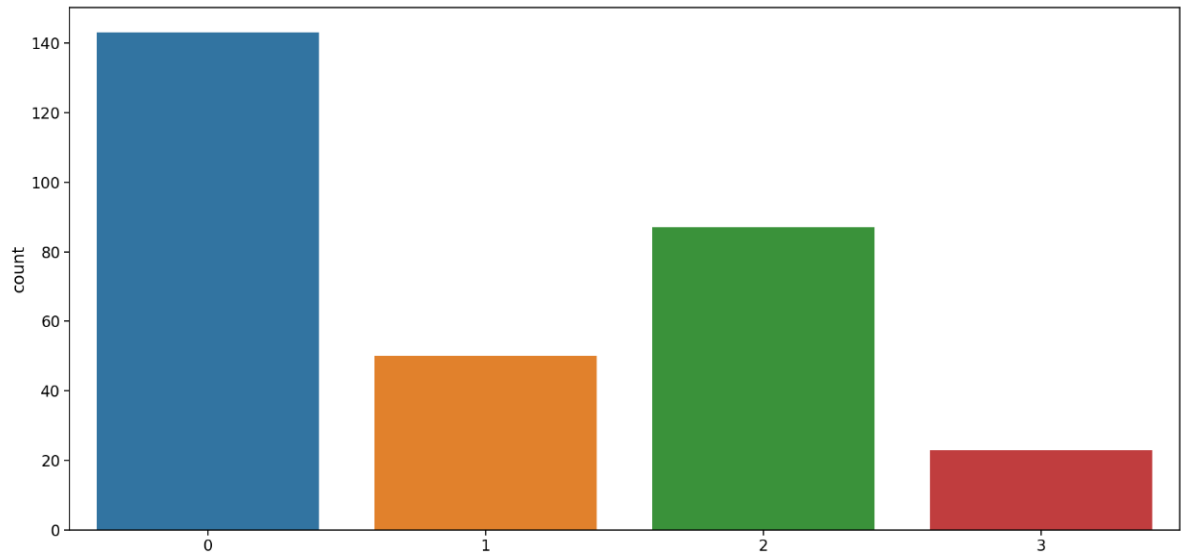
```
Young = data[(data['age']>=29)&(data['age']<39)]
Middle = data[(data['age']>=39)&(data['age']<54)]
Elder = data[(data['age']>54)]
colors = ['red','cyan','yellow']
explode = [0,0,0.1]
plt.figure(figsize=(8,8))
sns.set_context('notebook',font_scale = 1.2)
plt.pie([len(Young),len(Middle),len(Elder)],labels=['Young','Middle aged','Elderly'],explode=explode,colors=colors, autopct='%1.1
plt.tight_layout()
```



PLOTTING A COUNTPLOT ON CHEST PAIN TYPE:

```
plt.figure(figsize=(18,9))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data['cp'])
plt.tight_layout()
```

```
C:\Users\Laksh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```



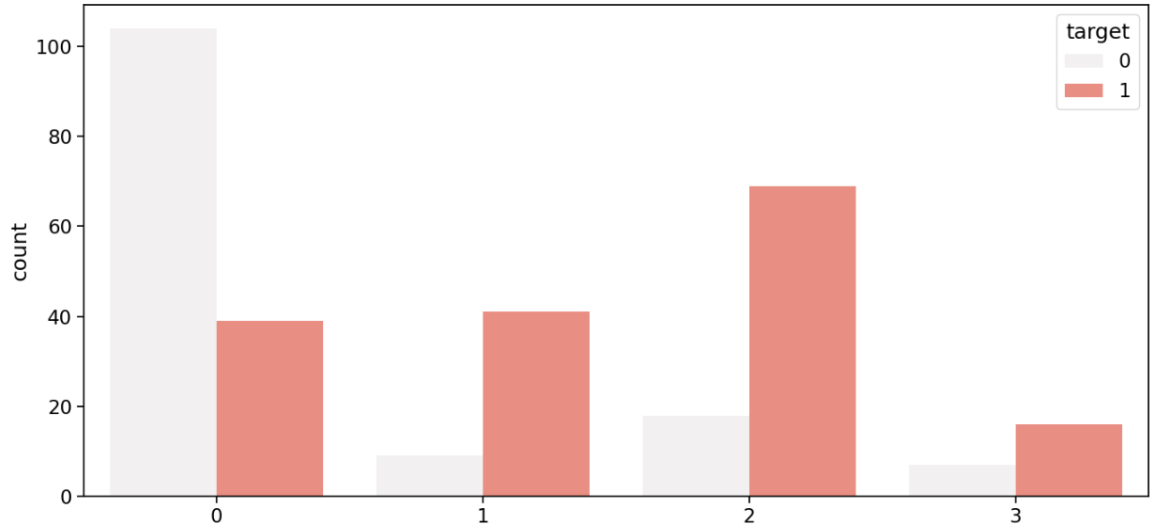
People with the chest pain type - 1 are more likely to come up with a heart disease compared to all other types.

This is also brought out in the graph below.

COUNTPLOT SPLIT INTO TARGET VALUES 0 AND 1:

```
plt.figure(figsize=(14,7))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data['cp'],hue=data["target"],color='salmon')
plt.tight_layout()
```

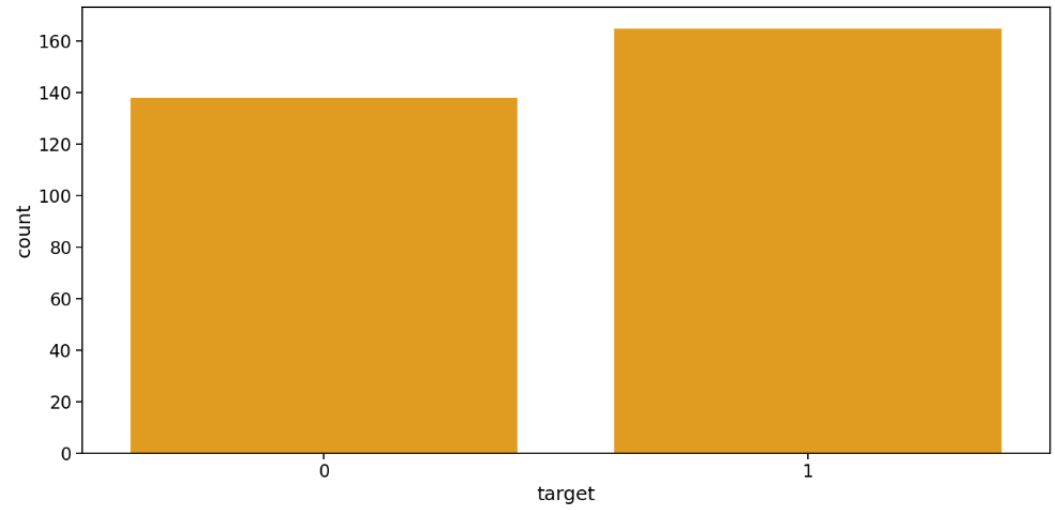
```
C:\Users\Laksh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```



ANALYING THE TARGET VARIABLE:

```
plt.figure(figsize=(14,7))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data['target'],color='orange')
plt.tight_layout()
```

```
C:\Users\Laksh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```



This graph correctly shows that the given dataset is not imbalanced as the ratio b/w 1 and 0 is less than 1.5.

Hence, we can use this data for creating a machine learning model as it is well balanced and contains relevant data for creating a relationship with the causes for a heart disease in patients.

Each relation we have found from the graphs are specified under the graph itself.

4. Python Packages:

The python packages that we have used are :

- **Pandas Library** : It is used for data cleaning and analysis.
- **NumPy** : to perform a wide variety of mathematical operations on arrays.
- **Matplotlib/seaborn** : Matplotlib and Seaborn act as the backbone of data visualization through Python.
- **Scikit-Learn** : It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

5. LEARNING ALGORITHMS:

We have used 3 algorithms for this model, which are :

- **KNN Algorithm**

The k-nearest neighbours algorithm, is a supervised learning classifier that employs proximity to produce classifications or predictions about the grouping of a single data point. We chose this algorithm as it makes highly accurate predictions, therefore we can use KNN algorithm for programs that require high accuracy but that do not require a human-readable model.

- **Logistic Regression**

This Classification algorithm is one of the most efficient technique for solving the classification problems. This algorithm is easier to implement, interpret and very efficient to train. It is also very fast at classifying the unknown records or data's.

- **Random Forest**

A Random Forest Algorithm is a supervised machine learning algorithm which is extremely popular and is used for Classification and Regression problems in Machine Learning. This algorithm's ability to handle data sets with both continuous variables, as in regression, and categorical variables, as in classification, is one of its most crucial qualities. In terms of classification issues, it delivers superior outcomes.

- Splitting data into training, validation, and testing :

```
# Everything except target variable
X = data.drop("target", axis=1)

# Target variable
y = data.target.values
```

```
X.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

```
y
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

```
from sklearn.model_selection import train_test_split
np.random.seed(42)

# Split into train & test set
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.2)
```

```
X_train.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
132	42	1	1	120	295	0	1	162	0	0.0	2	0	2
202	58	1	0	150	270	0	0	111	1	0.8	2	0	3
196	46	1	2	150	231	0	1	147	0	3.6	1	0	2
75	55	0	1	135	250	0	0	161	0	1.4	1	0	2
176	60	1	0	117	230	1	1	160	1	1.4	2	2	3

```
y_train, len(y_train)
```

```
(array([1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
        1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
        0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
        1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
        1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
        0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1,
        1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
        1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1],
       dtype=int64),
 242)
```

```
X_test.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
179	57	1	0	150	276	0	0	112	1	0.6	1	1	1
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3
111	57	1	2	150	126	1	1	173	0	0.2	2	1	3
246	56	0	0	134	409	0	0	150	1	1.9	1	2	3
60	71	0	2	110	265	1	0	130	0	0.0	2	1	2

```
y_test, len(y_test)
```

```
(array([0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
        0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64),
 61)
```

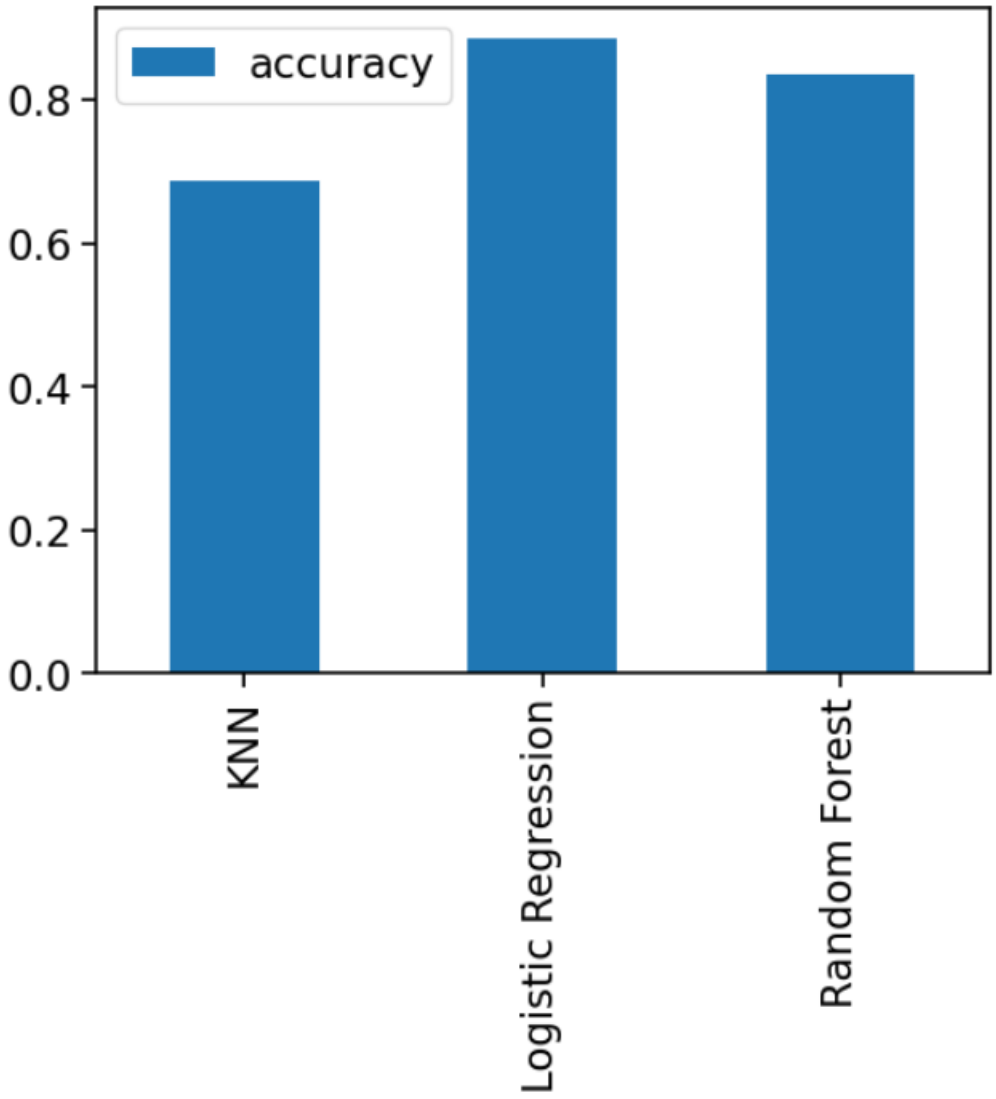
- Creating the models and estimating their accuracy on unseen data using the specified ML algorithms

```
ti = []
models = {"KNN": KNeighborsClassifier(n_neighbors=5),
          "KNN1": KNeighborsClassifier(n_neighbors=9),
          "KNN2": KNeighborsClassifier(n_neighbors=11)}

def fit_and_score1(models, X_train, X_test, y_train, y_test):


    np.random.seed(42)
    # Make a list to keep model scores
    model_scores = {}
    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(X_train, y_train)
        # Evaluate the model and append its score to model_scores
        starttime=time.time()
        model_scores[name] = model.score(X_test, y_test)
        exectime=time.time()-starttime
        ti.append(exectime)
        print("Predicted by ",model)
        print(accuracy_score(y_test, model.predict(X_test)))
        print(" ")


    return model_scores.values()
```

```
model_scores = fit_and_score1(models=models,
                              X_train=X_train,
                              X_test=X_test,
                              y_train=y_train,
                              y_test=y_test)

plt.scatter([5, 9, 11], ti)
plt.xlabel('K')
plt.ylabel('Time')



model_scores
```

```
Predicted by  KNeighborsClassifier()
0.6885245901639344

Predicted by  KNeighborsClassifier(n_neighbors=9)
0.6885245901639344

Predicted by  KNeighborsClassifier(n_neighbors=11)
0.7540983606557377
```

```
model_compare = pd.DataFrame(model_scores, index=['accuracy'])
model_compare.T.plot.bar();
```



- Making predictions on dataset :

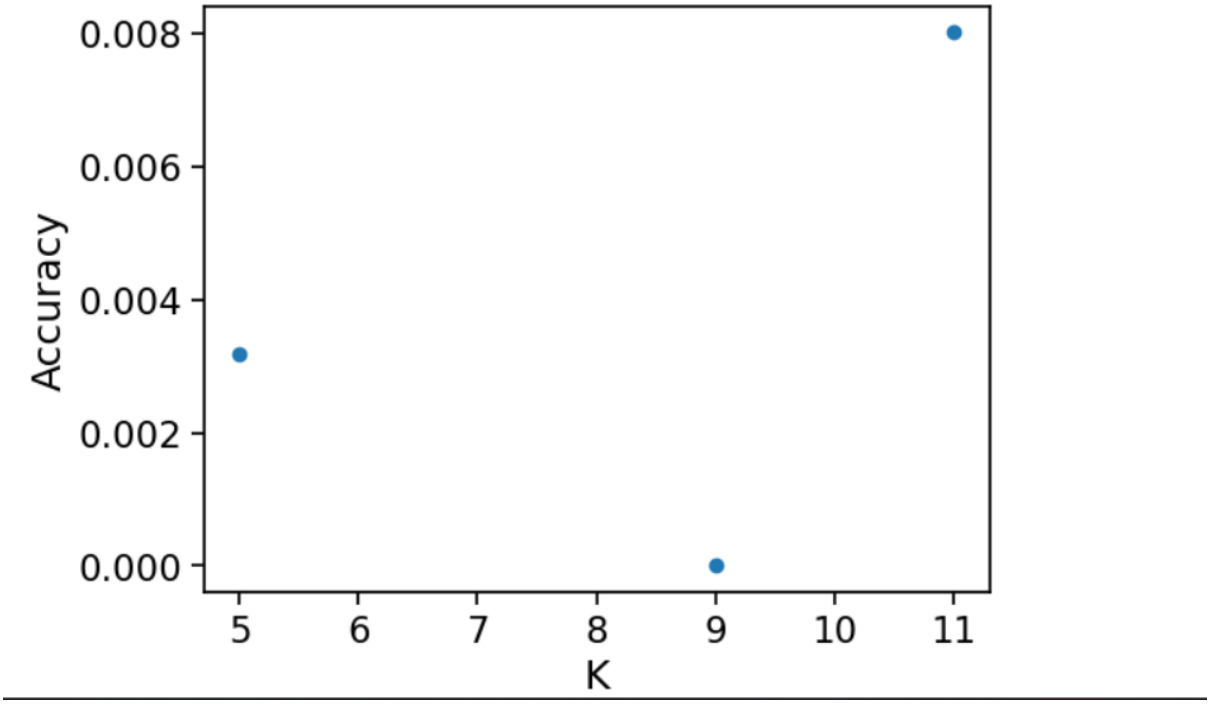
```
{'KNN': 0.6885245901639344,
 'Logistic Regression': 0.8852459016393442,
 'Random Forest': 0.8360655737704918}
```

```
models = {"KNN": KNeighborsClassifier(n_neighbors=5),
          "Logistic Regression": LogisticRegression(),
          "Random Forest": RandomForestClassifier()}

def fit_and_score(models, X_train, X_test, y_train, y_test):


    np.random.seed(42)
    # Make a list to keep model scores
    model_scores = {}
    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(X_train, y_train)
        # Evaluate the model and append its score to model_scores
        model_scores[name] = model.score(X_test, y_test)

        print("Predicted by ",model)
        print(accuracy_score(y_test, model.predict(X_test)))
        print(" ")

    return model_scores.values()
```

```
model_scores = fit_and_score1(models=models,
                              X_train=X_train,
                              X_test=X_test,
                              y_train=y_train,
                              y_test=y_test)

plt.scatter([5, 9, 11], ti)
plt.xlabel('K')
plt.ylabel('Accuracy')



model_scores
```

```
Predicted by  KNeighborsClassifier()
0.6885245901639344

Predicted by  LogisticRegression()
0.8852459016393442

Predicted by  RandomForestClassifier()
0.8360655737704918
```

```
dict_values([0.6885245901639344, 0.8852459016393442, 0.8360655737704918])
```

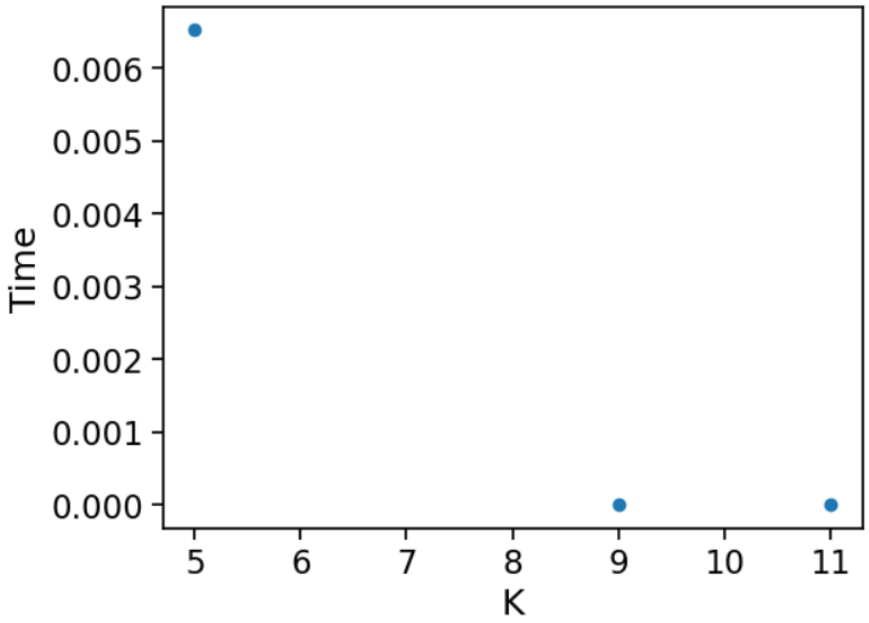



```
ti = []
models = {"KNN": KNeighborsClassifier(n_neighbors=5),
          "KNN1": KNeighborsClassifier(n_neighbors=9),
          "KNN2": KNeighborsClassifier(n_neighbors=11)}

def fit_and_score1(models, X_train, X_test, y_train, y_test):


    np.random.seed(42)
    # Make a list to keep model scores
    model_scores = {}
    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(X_train, y_train)
        # Evaluate the model and append its score to model_scores
        starttime=time.time()
        model_scores[name] = model.score(X_test, y_test)
        exectime=time.time()-starttime
        ti.append(exectime)
        print("Predicted by ",model)
        print(accuracy_score(y_test, model.predict(X_test)))
        print(" ")


    return model_scores.values()
```

```
model_scores = fit_and_score1(models=models,
                              X_train=X_train,
                              X_test=X_test,
                              y_train=y_train,
                              y_test=y_test)

plt.scatter([5, 9, 11], ti)
plt.xlabel('K')
plt.ylabel('Time')



model_scores
```

```
Predicted by  KNeighborsClassifier()
0.6885245901639344

Predicted by  KNeighborsClassifier(n_neighbors=9)
0.6885245901639344

Predicted by  KNeighborsClassifier(n_neighbors=11)
0.7540983606557377
```

```
dict_values([0.6885245901639344, 0.6885245901639344, 0.7540983606557377])
```



- Use k-fold cross validation to evaluate your ML algorithm:

1. FOR LOGISTIC REGRESSION

```
train_scores1=cross_val_score(LogisticRegression(), X_train, y_train, scoring='accuracy', cv=5)
```

```
train_scores1.mean()
```

```
0.8180272108843537
```

2. FOR KNN ALGORITHM

```
train_scores2=cross_val_score(KNeighborsClassifier(), X_train, y_train, scoring='accuracy', cv=5)
train_scores2.mean()
```

```
0.6488095238095238
```

3. FOR RANDOM FOREST

```
train_scores3=cross_val_score(RandomForestClassifier(), X_train, y_train, scoring='accuracy', cv=5)
train_scores3.mean()
```

```
0.7934523809523809
```

SUBMITTED BY

KRISHNAPRIYA V S