Predicting heart disease using machine learning

This notebook looks into using various `Python-based machine learning and data science libraries` in an attempt to build a machine learning model capable of predicting whether or not someone has heart disease based on their medical attributes

We are going to take the following approach

1. Problem definition
2. Data
3. Evaluation
4. Features
5. Modelling
6. Experimentation

1. Problem Definition

In a statement,

> Given clinical parameters about a patient, can we predict whether or not they have heart disease?

2. Data

The original data if from the Cleveland data from the UCI machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/heart+disease

There is also a version on Kaggle. https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

3. Evaluation

> If we can reach 95% accuracy at predicting whether or not a patient has heart disease during the proof of concept, we will pursue the project.

4. Features

This where you get different information about each of the features in your data

Create data dictionary

1. age

    > age in years

2. sex

> (1 = male; 0 = female)

3. cp

> chest pain type

4. trestbps

> resting blood pressure (in mm Hg on admission to the hospital)

5. chol

> serum cholestoral in mg/dl

6. fbs

> (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

7. restecg

> resting electrocardiographic results

8. thalach

> maximum heart rate achieved

9. exang

> exercise induced angina (1 = yes; 0 = no)

10. oldpeak

> ST depression induced by exercise relative to rest

11. slope

> the slope of the peak exercise ST segment

12. ca

> number of major vessels (0-3) colored by flourosopy

13. thal

> 1 = normal; 2 = fixed defect; 3 = reversable defect

14. target

> 1 or 0

Preparing the tools

We are going to use Pandas, Numpy and Matplotlib for data analysis and manipulation

In [2]:
```
# Import all the tools we need
```

```
# Regular EDA and plotting libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# So that our plot will appear inside the notebook
%matplotlib inline



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

#Model Evaluations
from sklearn.model_selection import train_test_split, GridSearchCV,cross_val_score,R
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import precision_score, recall_score, f1_score,plot_roc_curve
```

Load data

In [4]:

```
df =pd.read_csv("heart-disease.csv")
df.shape # (Rows, Columns)
```

Out[4]:

```
(303, 14)
```

Data Exploratory Analysis (EDA)

The goal here is to find our more anout the data and become a subject matter expert on the dataset you are working with.

1. What questions are you trying to solve
2. What kind of data do you have and how do we treat different types?
3. What is 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?

In [5]:

```
df.head()
```

Out[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

In [6]:

```
df.tail()
```

Out[6]:

	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

In [10]:

```
df.target.value_counts() # method 1 of referencing columns
```

Out[10]:

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

In [11]:

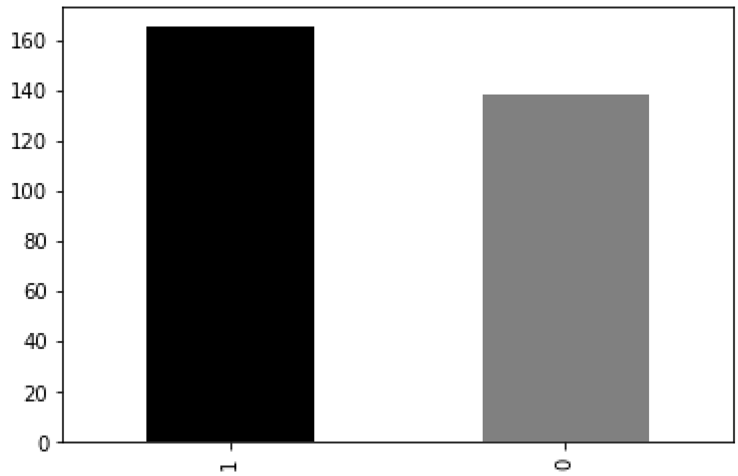
```
df['target'].value_counts() # method 2 of referencing columns
```

Out[11]:

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

In [40]:

```
df.target.value_counts().plot(kind='bar',color=['black','gray']);
```



In [15]:

```
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        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
```

In [17]:

```
df.isna().sum()
```

Out[17]:

```
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
```

In [18]:

```
df.describe()
```

Out[18]:

	age	sex	cp	trestbps	chol	fbs	restecg	thala
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.6468
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.9051
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.0000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.5000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.0000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.0000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.0000

Heart Disease Frequency according to Sex

In [19]:

```
new =pd.DataFdf.sex.value_counts()
```

Out[19]:

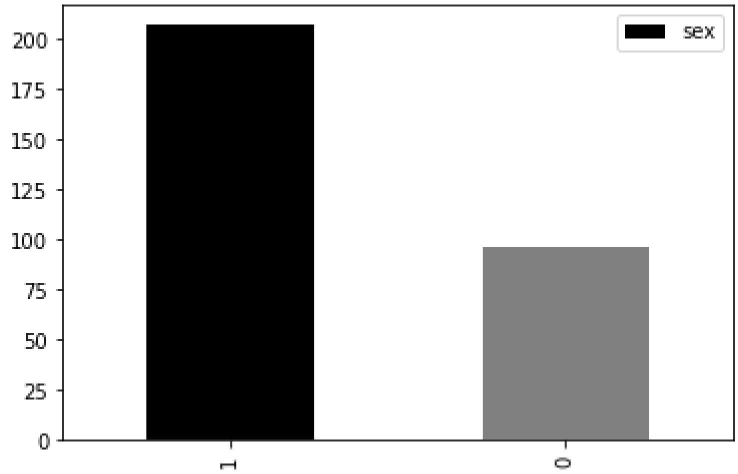
```
1    207
0     96
Name: sex, dtype: int64
```

In [29]:

```
df.sex.value_counts().plot(kind='bar',color=['black','gray']);
# plt.legend()
```

Out[29]:

```
<matplotlib.legend.Legend at 0x1b762fb5120>
```



In [30]:

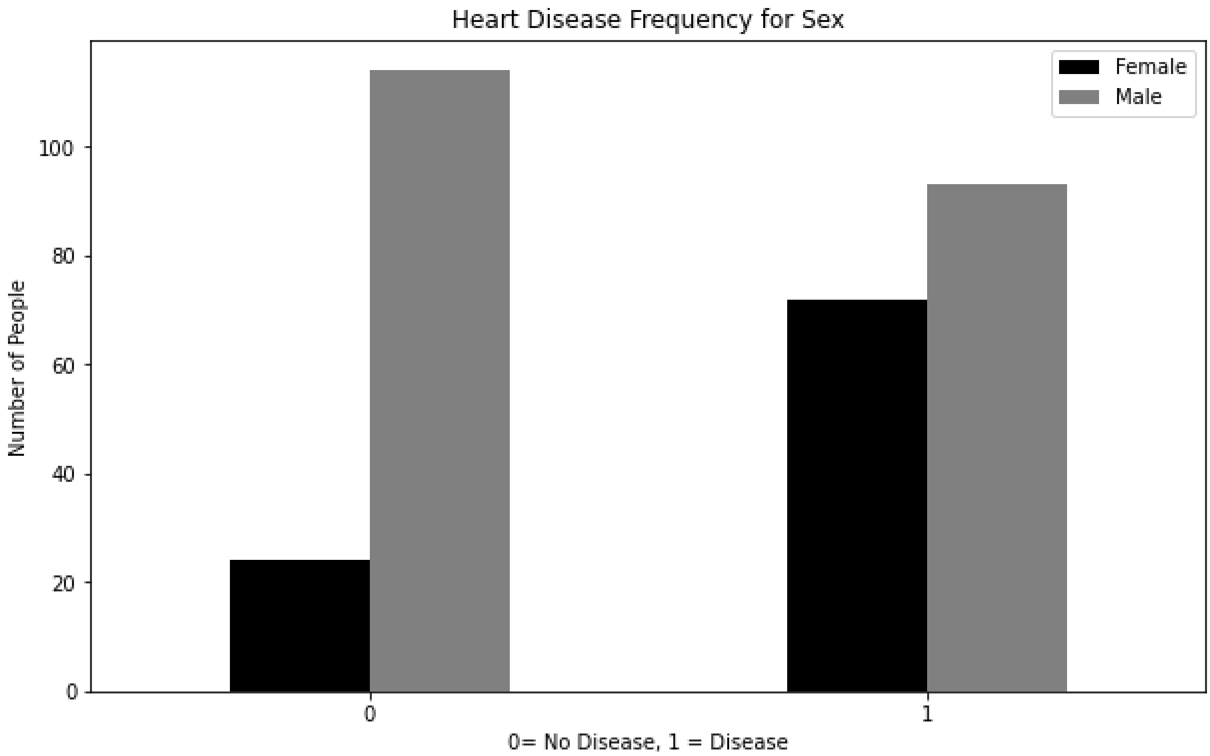
```
# compare target column with the sex column
pd.crosstab(df.target,df.sex)
```

Out[30]:

sex	0	1
target		
0	24	114
1	72	93

In [39]:

```
#create a plot of the crosstab
pd.crosstab(df.target,df.sex).plot(kind='bar',
                                   figsize=(10,6),
                                   color=['black','gray']);

plt.title('Heart Disease Frequency for Sex')
plt.xlabel('0= No Disease, 1 = Disease')
plt.ylabel ('Number of People')
plt.legend(['Female','Male']);
plt.xticks(rotation=0);
```



In [41]:

```
df.thalach.value_counts()
```

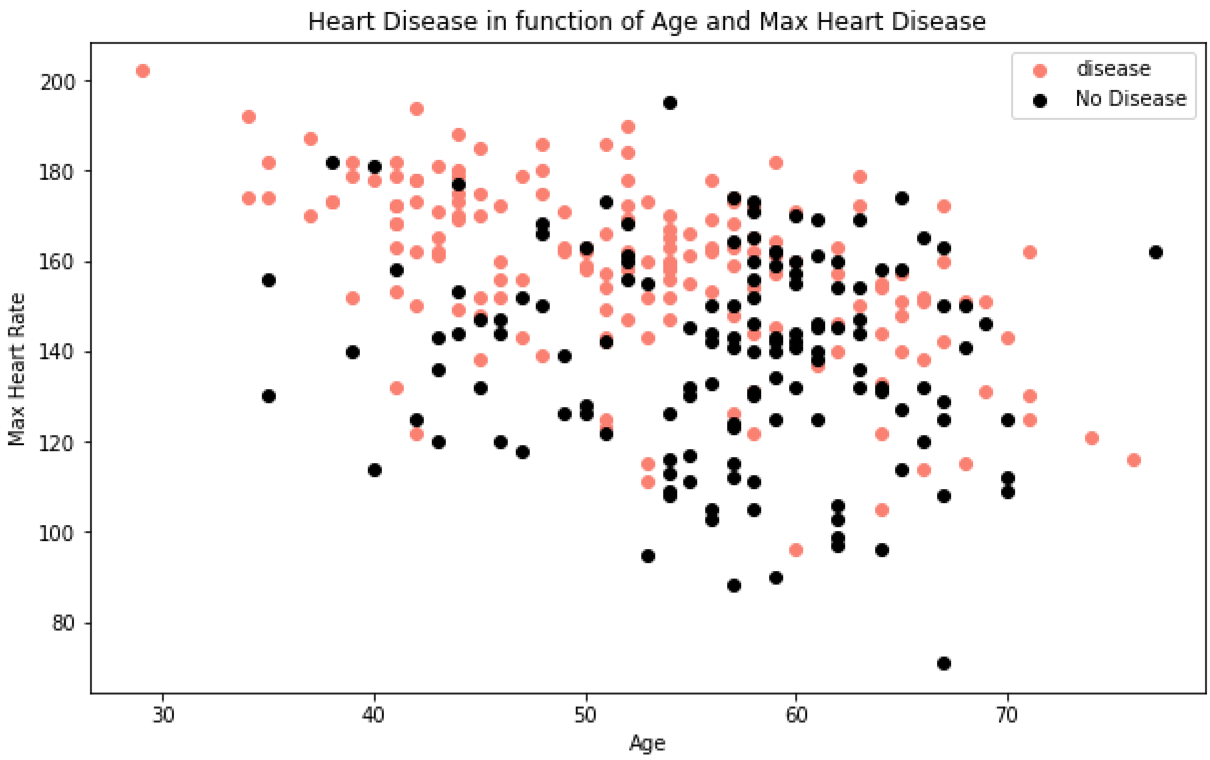
Out[41]:

```
162    11
160     9
163     9
152     8
173     8
       ..
202     1
184     1
121     1
192     1
90      1
Name: thalach, Length: 91, dtype: int64
```

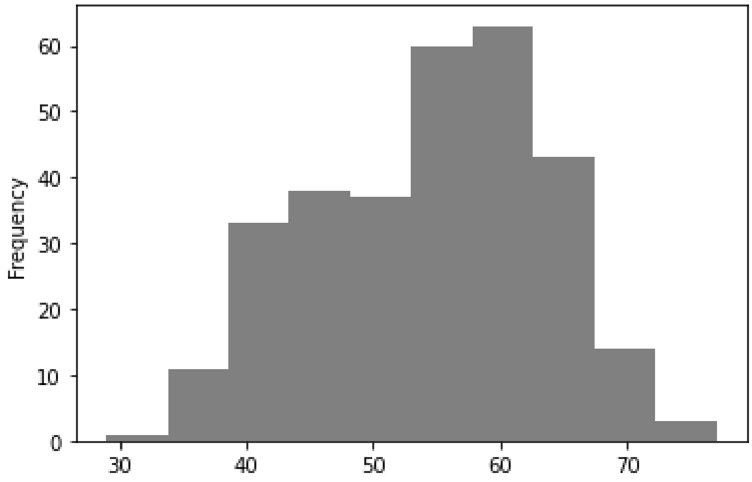
Age Vs Max Heart Rate for Heart Disease

In [52]:

```
# Create another figure
plt.figure(figsize=(10,6))

# scatter with positive example
plt.scatter(df.age[df.target==1],
            df.thalach[df.target==1],
            color=['Salmon'])

# scatter with negative example
plt.scatter(df.age[df.target==0],
            df.thalach[df.target==0],
            color='black');

#Add some helpful info

plt.title('Heart Disease in function of Age and Max Heart Disease')
plt.xlabel('Age')
plt.ylabel('Max Heart Rate')
plt.legend(['disease','No Disease']);
```



In [71]:
```
# Check the distribution of the Age column (Spread of the data)
df.age.plot.hist(color='gray');
```



Compare the Chest Pain Type to Target

Heart disease frequency per chest pain type

cp: chest pain type

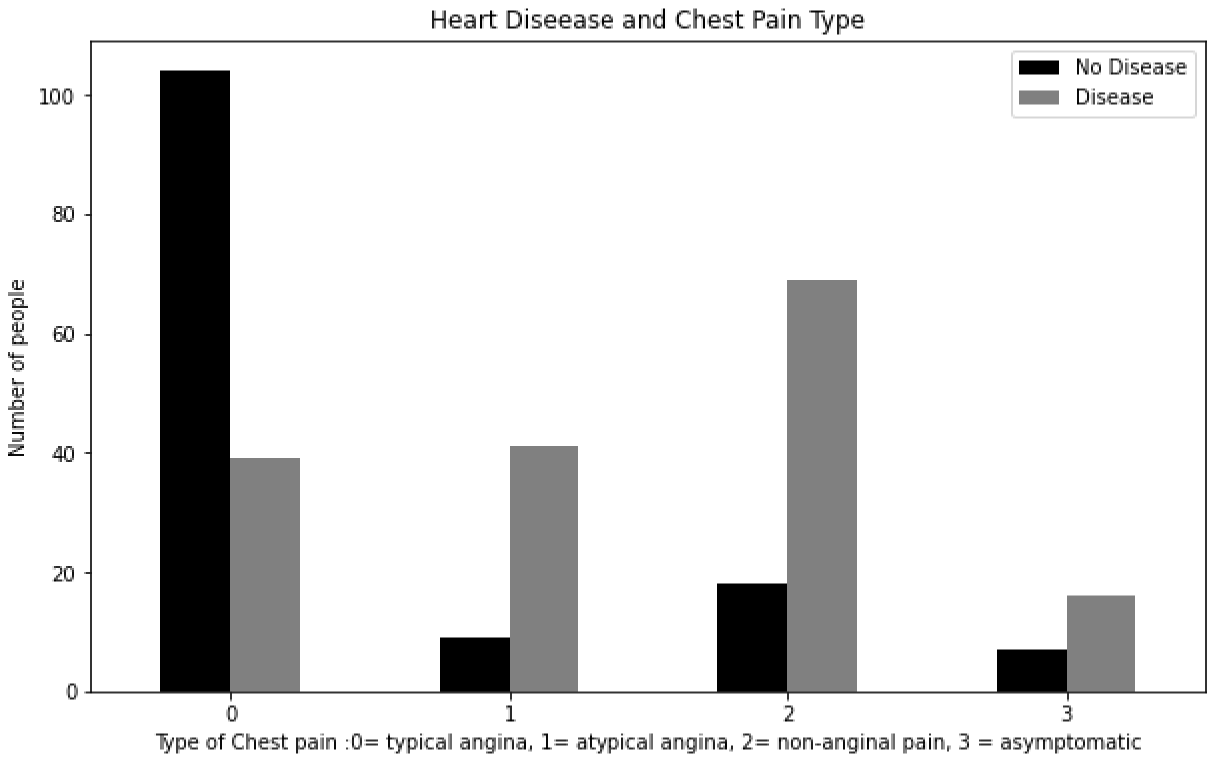
- Value 1: typical angina
- Value 2: atypical angina
- Value 3: non-anginal pain
- Value 4: asymptomatic

In [56]:
```
pd.crosstab(df.cp,df.target)
```

Out[56]:

target	0	1
cp		
0	104	39
1	9	41
2	18	69
3	7	16

In [66]:
```
## make the crosstab more visual

pd.crosstab(df.cp,df.target).plot(kind='bar',color=['black','gray'],
                                  figsize=(10,6))

# Add some communication
plt.title('Heart Diseease and Chest Pain Type')
plt.xlabel('Type of Chest pain :0= typical angina, 1= atypical angina, 2= non-angina
plt.ylabel('Number of people')
plt.legend(['No Disease','Disease']);
plt.xticks(rotation=0);
```



In [67]:

```
# Make a correlation matrix of all the features
df.corr()
```

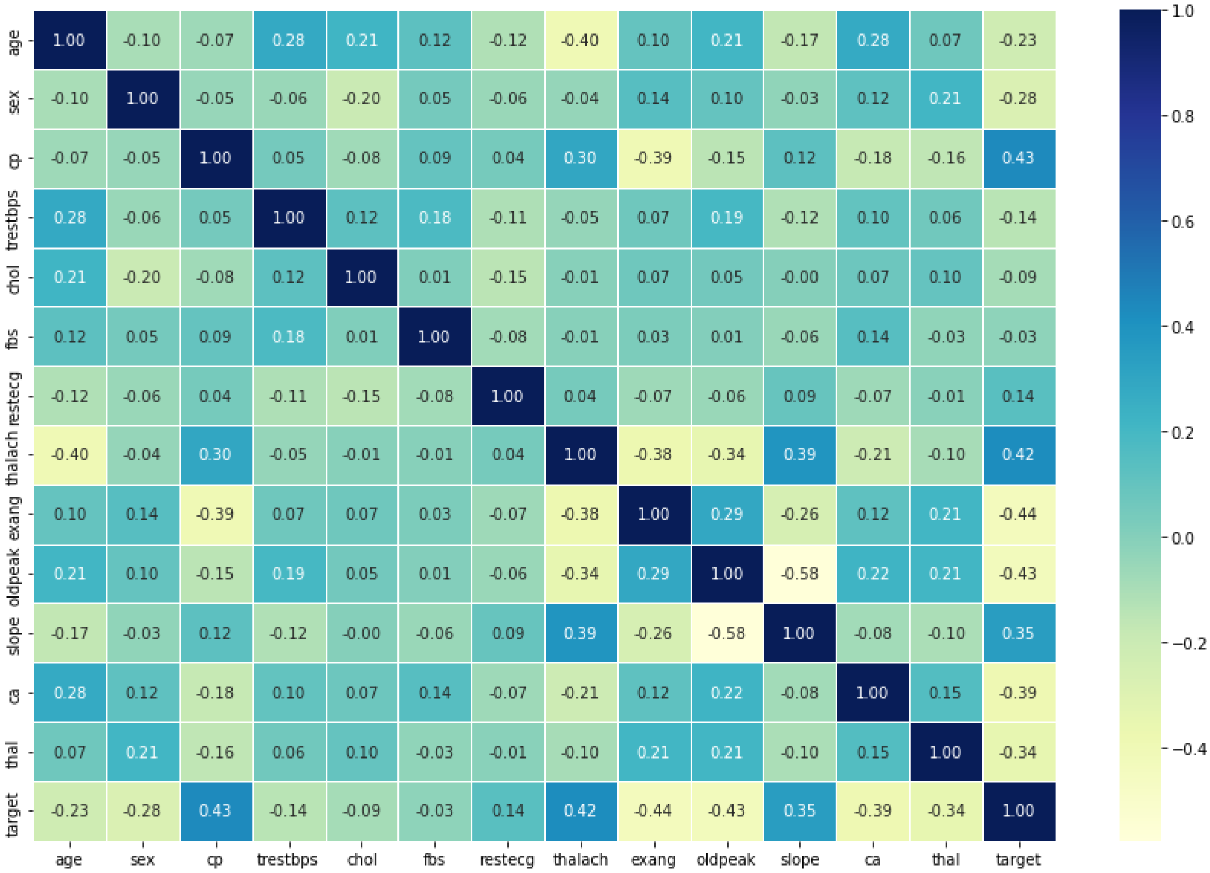
Out[67]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	e
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.09
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.14
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.39
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.06
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.06
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.02
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.07
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.37
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.00
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.28
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.25
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.11
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.20
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.43

In [69]:

```
# Let's make our correlation matrix a little better
corr_matrix=df.corr()
fig,ax =plt.subplots(figsize=(15,10))

ax =sns.heatmap(corr_matrix,
```

```
                    annot=True,
                    linewidth=0.5,
                    fmt='.2f',
                    cmap='YlGnBu')
```



5. Modelling

In [72]:

```
df.head()
```

Out[72]:

	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

In [73]:

```
# Split the data into X and y
X= df.drop('target',axis=1)
X
```

Out[73]:

	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

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

303 rows × 13 columns

In [75]:

```
y = df.target
y
```

Out[75]:

```
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
```

In [77]:

```
# Split the data into train and test sets
np.random.seed(42)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Out[77]:

```
((242, 13), (242,), (61, 13), (61,))
```

Now that we have the data into training and test sets, it's time to build a machine learning model

We will train it (find the patterns) on the training set And we will test it (use the patterns) on the test set

We are going to try out 3 different machine learning models

1. Logistic Regression
2. K-Nearest Neighbours Classifier
3. Random Forest Classifier

In [95]:

```
# Put models in a dictionary
models = {'Logistic Regression': LogisticRegression(),
          'KNN':KNeighborsClassifier(),
          'Randon Forest':RandomForestClassifier()}
```

```
# Create a function to fit and score models
def fit_and_score (models,X_Train,X_test,y_train,y_test):
    '''
    Fits and evaluates given machine learning models
    models: a dict of differeny Scikit-Learn machine learning models
    X_train: training data (no labels)
    X_test: test data (no labels)
    y_train: training labels
    y_test: test labels

    '''
    # set random seed
    np.random.seed(42)

    #make a dictionary to keep model scores
    model_scores={}

    # Loop through the models'
    for name,model in models.items():
        #Fit the model to the data
        model.fit(X_train,y_train)
        #Evaluate the model and append the score to model_score
        model_scores[name]=model.score(X_test,y_test)
    return model_scores
```

In [98]:

```
model_scores = fit_and_score(models=models,
                             X_Train=X_train,
                            X_test=X_test,
                            y_train=y_train,
                            y_test=y_test)


model_scores
```

```
C:\Users\USER\OneDrive\Documents\PERSONAL\PERSONAL DEVELOPMENT\DATA SCIENCE\Personal
Projects\heart-diesease-project\env\lib\site-packages\sklearn\linear_model\_logisti
c.py:444: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[98]:

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

Model Comparison

In [107…

```
model_compare=pd.DataFrame(model_scores,index=['accuracy'])
model_compare
```

Out[107…

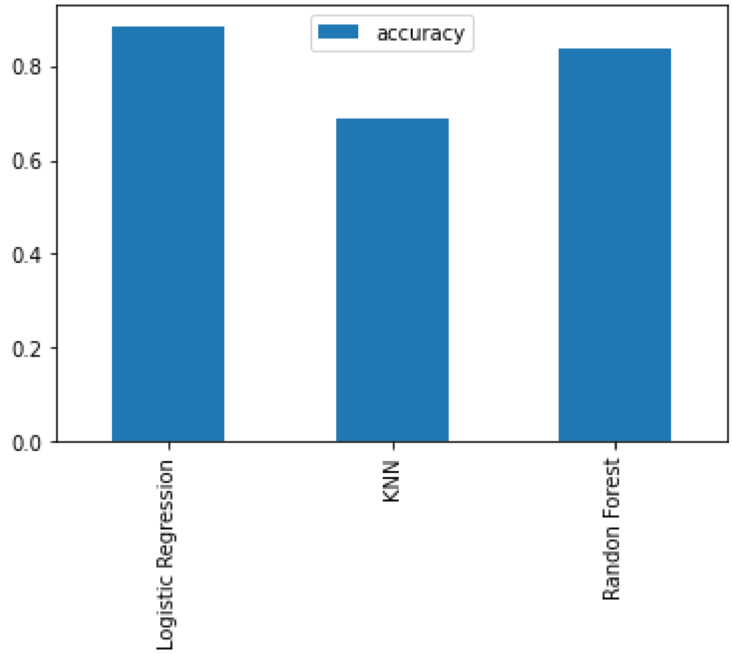
	Logistic Regression	KNN	Randon Forest
accuracy	0.885246	0.688525	0.836066

In [105…

```
model_compare.T.plot(kind='bar')
```

Out[105…

```
<AxesSubplot:>
```



Now we have got a baseline model and know a model's first predictions aren't always what we should base our next steps off. What should i do?

let's look at the following:

- Hyperparameter tuning (*all types of models*) *Feature importance (*all types of models*)*Confusion matrix (*classification models only*) *Cross-validation (*classification models only*???)*Precision (*classification models only*) *Recall (*classification models only*)*F1 Score (*classification models only*) *Classification report (*classification models only*)*ROC curve (*classification models only*) *Area under the curve (AUC) (*classification models only*)

Hyperparameter tuning (by hand)

In [110…

```
# Let's tune KNN
train_scores =[]
test_scores =[]

# Create a list if different values for n_neighbors
neighbors = range(1,21)

#Setup KNN instance
knn = KNeighborsClassifier()

# Loop through different n_neighbors
for i in neighbors:
    knn.set_params(n_neighbors=i)

    #fit the algorithm
    knn.fit(X_train,y_train)

    #update the training scores list
    train_scores.append(knn.score(X_train,y_train))

    #update the test scores list
    test_scores.append(knn.score(X_test,y_test))
```

In [111…

```
train_scores
```

Out[111…

```
[1.0,
 0.8099173553719008,
 0.7727272727272727,
 0.743801652892562,
 0.7603305785123967,
 0.7520661157024794,
 0.743801652892562,
 0.7231404958677686,
 0.71900826446281,
 0.6942148760330579,
 0.7272727272727273,
 0.6983471074380165,
 0.6900826446280992,
 0.6942148760330579,
 0.6859504132231405,
 0.6735537190082644,
 0.6859504132231405,
 0.6652892561983471,
 0.6818181818181818,
 0.6694214876033058]
```

In [112…

```
test_scores
```

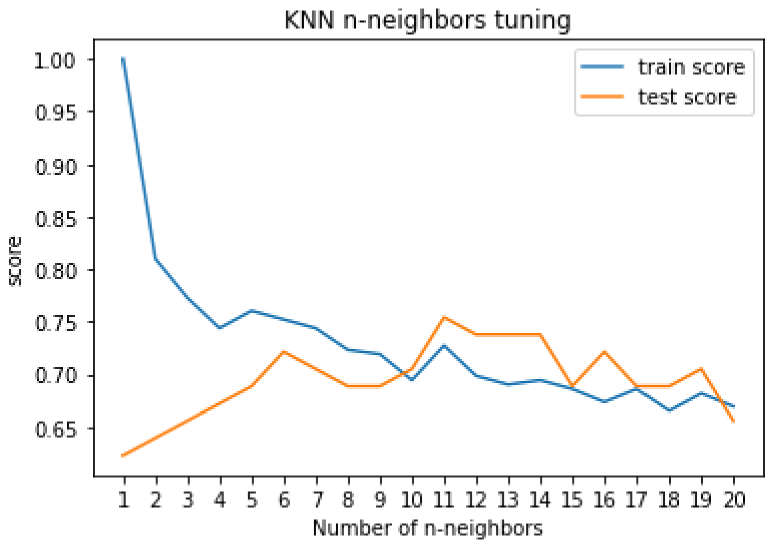
Out[112…

```
[0.6229508196721312,
 0.639344262295082,
 0.6557377049180327,
 0.6721311475409836,
 0.6885245901639344,
 0.7213114754098361,
 0.7049180327868853,
 0.6885245901639344,
 0.6885245901639344,
 0.7049180327868853,
 0.7540983606557377,
 0.7377049180327869,
 0.7377049180327869,
 0.7377049180327869,
 0.6885245901639344,
 0.7213114754098361,
 0.6885245901639344,
 0.6885245901639344,
 0.7049180327868853,
 0.6557377049180327]
```

In [118…

```
plt.plot(neighbors,train_scores,label='train score')
plt.plot(neighbors,test_scores,label='test score');


#Add some communications
plt.title("KNN n-neighbors tuning")
plt.xlabel('Number of n-neighbors')
plt.xticks(np.arange(1,21,1))
plt.ylabel('score')
plt.legend(['train score','test score'])

print (f'Maximum KNN score on the test data: {max(test_scores)*100:.2f}%');
```

```
Maximum KNN score on the test data: 75.41%
```



Hyperparameter tuning (by RandomizedSearchCV)

We are going to tune:

- LogisticRegression()
- RandomForestClassifier() ....RandomizedSearchCv

In [130…

```
# Create a hyperparameter grid for LogisticRegression()
log_reg_grid = {"C":np.logspace(-4,4,20),
               'solver':['liblinear']}

# Create a hyperparameter grid for RandomForestClassifier()
rf_grid= {'n_estimators':np.arange(10,1000,50),
         'max_depth':[None,3,5,10],
         'min_samples_split':np.arange(2,20,2),
         'min_samples_leaf':np.arange(1,20,2)}
```

Now we have hyperparameter grids setup for each of our models, lets tune them using RandomizedSearchCV...

In [122…

```
# Tune LogisticRegression

np.random.seed(42)

# setup random hyperparameter search for LogisticRegression
rs_log_reg =RandomizedSearchCV(LogisticRegression(),
                              param_distributions=log_reg_grid,
                              cv=5,
                              n_iter=20,
                              verbose=True)

# Fit random hyperparameter search model for LogisticRegression
rs_log_reg.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[122…

```
▸ RandomizedSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [123…

```
rs_log_reg.best_params_
```

Out[123…

```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

In [124…

```
rs_log_reg.score(X_test,y_test)
```

Out[124…

```
0.8852459016393442
```

Now we have tuned LogisticRegression(), let's do the same for RandomForestClassifier()

In [131…

```
# Setup random seed
np.random.seed(42)

# setup random hyperparameter search for RandomForestClassifier
rf =RandomizedSearchCV(RandomForestClassifier(),
                              param_distributions=rf_grid,
                              cv=5,
                              n_iter=20,
                              verbose=True)

# Fit random hyperparameter search model for RandomForestClassifier
rf.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[131…

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [132…

```
# Find the best hyperparameters
rf.best_params_
```

Out[132…

```
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

In [133…

```
# Evaluate the randomized search RandomForestClassifier model
rf.score(X_test,y_test)
```

Out[133…

```
0.8688524590163934
```

Tuning hyperparameter (Use the GridSearchCV for the LogisticRegression() model)

Hyperparameter Tuning with GridSearchCV

Since our logisticRegression model provides the best score so far, we'll try and improve them again using GridSearchCV...

In [136…

```
# Different parameters for our LogisticRegression model
```

```
log_reg_grid= {'C': np.logspace(-4,4,30),
               'solver':['liblinear']}

#Setup grid hyperparameter search for LogisticRegression
gs_log_reg =GridSearchCV(LogisticRegression(),
                         param_grid=log_reg_grid,
                         cv=5,
                         verbose=True)

#Fit grid hyperparameter search model
gs_log_reg.fit(X_train,y_train)
```

Fitting 5 folds for each of 30 candidates, totalling 150 fits

Out[136…

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

In [137…

```
gs_log_reg.best_params_
```

Out[137…

```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [138…

```
# Evaluate the grid search LogisticRegression() model
gs_log_reg.score(X_test,y_test)
```

Out[138…

```
0.8852459016393442
```

Evaluating our tuned machine learning classifier, beyond accuracy

- ROC curve and AUC
- Confusion matrix
- Classification report
- Precision
- Recall
- F1-score

...and it would be great if cross-validation was used where possible.

To make comparisons and evaluate our trained model, first we need to make predictions

In [164…

```
# Make predictions with tuned model
y_preds =gs_log_reg.predict(X_test)
```

In [165…

```
y_preds
```

Out[165…

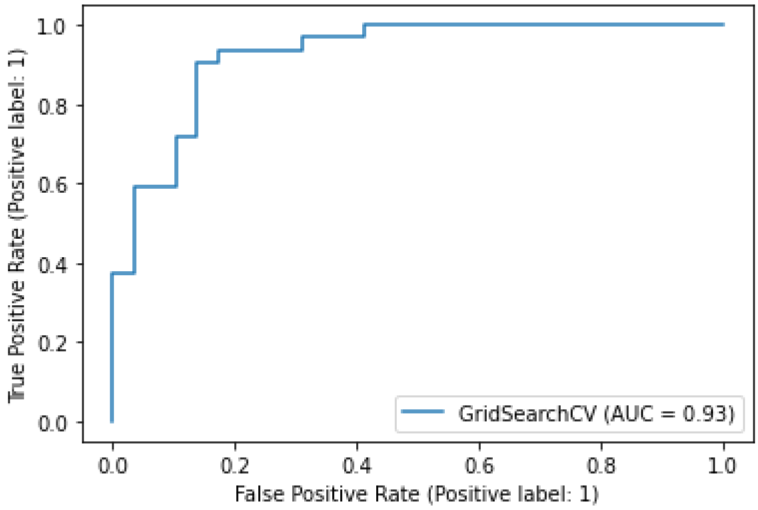
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [166…

```
#Plot the ROC curve and calculate the AUC metric
```

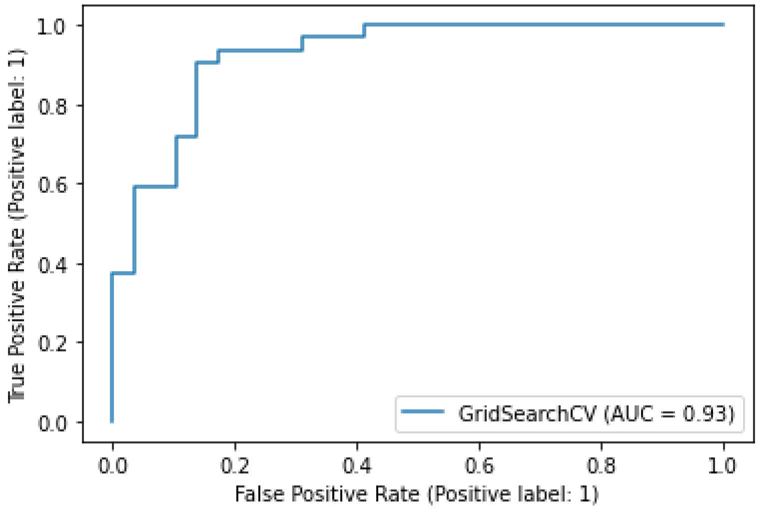
```
plot_roc_curve(gs_log_reg,X_test,y_test);
```

```
C:\Users\USER\OneDrive\Documents\PERSONAL\PERSONAL DEVELOPMENT\DATA SCIENCE\Personal
Projects\heart-diesease-project\env\lib\site-packages\sklearn\utils\deprecation.py:8
7: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_cu
rve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
:meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.metric.Roc
CurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



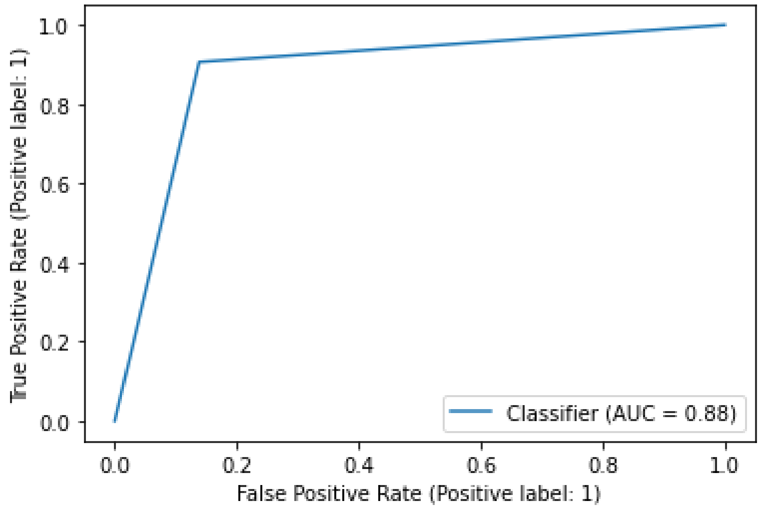
In [167…

```
# Use the latest method for ROC Curve
#import the required function
from sklearn.metrics import RocCurveDisplay


RocCurveDisplay.from_estimator(gs_log_reg, X_test, y_test);
```



In [169…

```
RocCurveDisplay.from_predictions(y_test,y_preds);
```



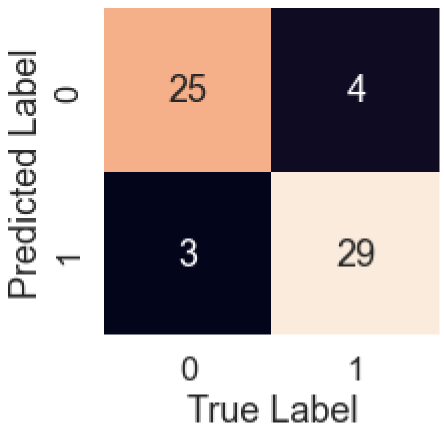
In [170…

```
# Confusion matrix
print (confusion_matrix(y_test,y_preds))
```

```
[[25  4]
 [ 3 29]]
```

In [174…

```
#Visualise using seaborn
sns.set(font_scale=1.5)

def plot_conf_mat(y_test,y_preds):
    '''
    Plot a nice looking confusion matrix using seaborn's heatmap()
    '''
    fig,ax =plt.subplots(figsize=(3,3))
    ax =sns.heatmap(confusion_matrix(y_test,y_preds),
                    annot=True,
                    cbar=False)
    plt.xlabel('True Label')
    plt.ylabel('Predicted Label')

plot_conf_mat(y_test,y_preds)
```



Now we've got the ROC curve, an AUC metric, a confusion matrix, let's get a classification report as well as cross-validated precision, recall and f1-score

In [175…

```
print(classification_report(y_test,y_preds))
```

```
              precision    recall  f1-score   support
```

```
           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61
```

Calulate evaluation metrics (precision ,recall and f1-score) using cross-validation

We are going to calculate precision, recall and f1-score of our model using cross-validation and to do so we will be using cross_val_score().

In [176…

```
#check the best hyperparameters
gs_log_reg.best_params_
```

Out[176…

```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [179…

```
# Create a new classifier with best parameters
clf =LogisticRegression(C=0.20433597178569418,solver ='liblinear')
```

In [183…

```
#cross-validated accuracy
cv_acc=cross_val_score(clf,
                       X,
                       y,
                       cv=5,
                       scoring ='accuracy')
cv_acc
```

Out[183…

```
array([0.81967213, 0.90163934, 0.86885246, 0.88333333, 0.75      ])
```

In [184…

```
cv_acc=np.mean(cv_acc)
cv_acc
```

Out[184…

```
0.8446994535519124
```

In [188…

```
#cross-validated precision
cv_precision=cross_val_score(clf,
                       X,
                       y,
                       cv=5,
                       scoring ='precision')
cv_precision
```

Out[188…

```
array([0.775     , 0.88571429, 0.85714286, 0.86111111, 0.725     ])
```

In [189…

```
cv_precision=np.mean(cv_precision)
cv_precision
```

Out[189…

```
0.8207936507936507
```

In [190…

```
#cross-validated recall
```

```
cv_recall =cross_val_score(clf,
                           X,
                           y,
                           cv=5,
                           scoring ='recall')
cv_recall=np.mean(cv_recall)
cv_recall
```

Out[190…

```
0.9212121212121213
```

In [191…

```
#cross-validated f1-score
cv_f1 =cross_val_score(clf,
                       X,
                       y,
                       cv=5,
                       scoring ='f1')
cv_f1=np.mean(cv_f1)
cv_f1
```

Out[191…

```
0.8673007976269721
```

In [196…

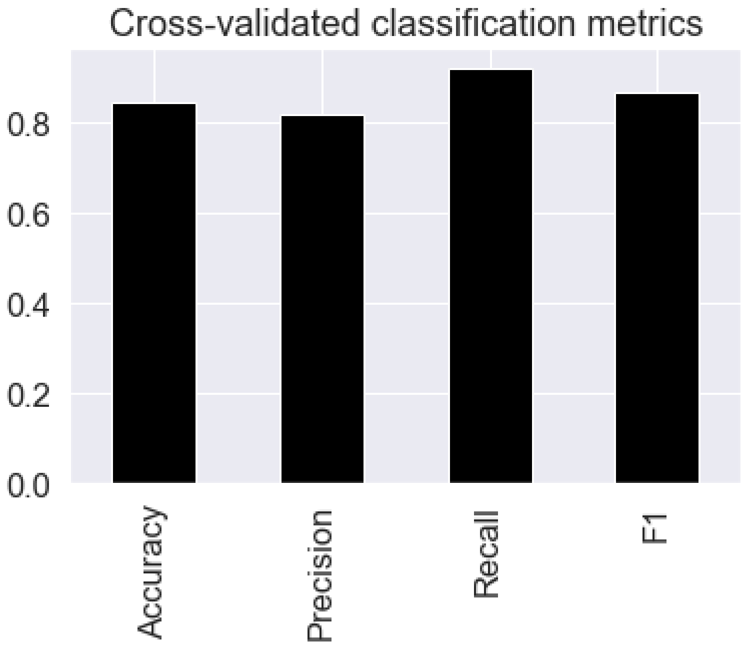
```
# Visualise the cross-validated metrics
cv_metrics =pd.DataFrame({'Accuracy':cv_acc,
                          'Precision':cv_precision,
                          'Recall':cv_recall,
                          'F1':cv_f1},index =[0])
cv_metrics
```

Out[196…

	Accuracy	Precision	Recall	F1
0	0.844699	0.820794	0.921212	0.867301

In [202…

```
cv_metrics.T.plot(kind='bar',color='black',title='Cross-validated classification met
```



Feature Importance

Feature importance is another way of asking, "which features contributed most to the outcomes of the model and how did they contribute?"

Finding feature importance is different for each machine learning model.One way to find feature importance is to searcg for ('MODEL NAME') feature importance

Let's find the feature importance for our LogisticRegression model

In [205…

```
# Fit an instance of LogisticRegression

clf= LogisticRegression(C=0.20433597178569418,solver ='liblinear')

clf.fit(X_train,y_train)
```

Out[205…

```
▾ LogisticRegression
LogisticRegression(C=0.20433597178569418, solver='liblinear')
```

In [206…

```
# Check coef_
clf.coef_
```

Out[206…

```
array([[ 0.00316728, -0.86044651,  0.66067041, -0.01156993, -0.00166374,
         0.04386107,  0.31275847,  0.02459361, -0.6041308 , -0.56862804,
         0.45051628, -0.63609897, -0.67663373]])
```

In [207…

```
# Match coef's of features to columns
feature_dict=dict(zip(df.columns,list(clf.coef_[0])))
feature_dict
```

Out[207…

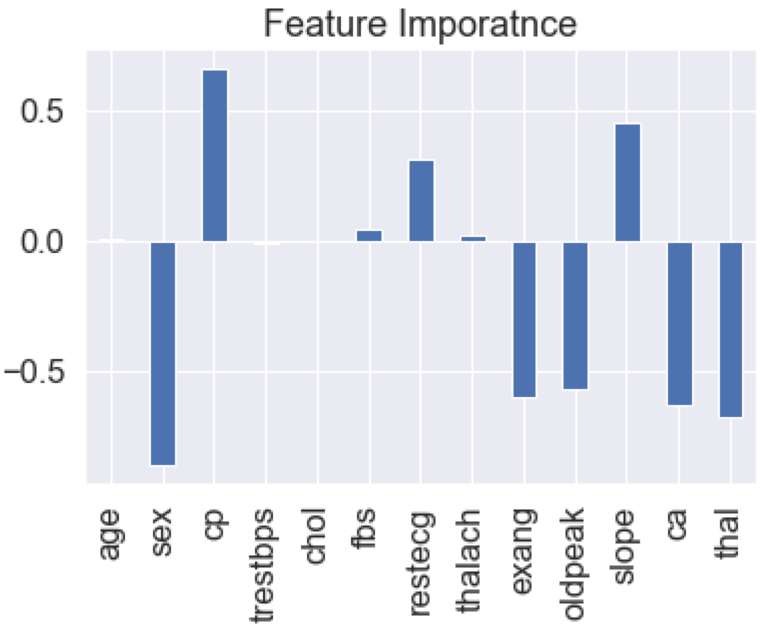
```
{'age': 0.0031672801993431563,
 'sex': -0.8604465072345515,
 'cp': 0.6606704082033799,
 'trestbps': -0.011569931680808875,
 'chol': -0.0016637445047768871,
 'fbs': 0.043861071652469864,
 'restecg': 0.31275846822418324,
 'thalach': 0.024593613737779126,
 'exang': -0.6041308000615746,
 'oldpeak': -0.5686280368396555,
 'slope': 0.4505162797258308,
 'ca': -0.6360989676086223,
 'thal': -0.6766337263029825}
```

In [208…

```
# Visualise feature importances
feature_df =pd.DataFrame(feature_dict,index=[0])
feature_df.T.plot.bar(title='Feature Imporatnce',legend=False)
```

Out[208…

```
<AxesSubplot:title={'center':'Feature Imporatnce'}>
```



In [209…

```
pd.crosstab(df['sex'],df['target'])
```

Out[209…

target	0	1
sex		
0	24	72
1	114	93

In [210…

```
pd.crosstab(df['slope'],df['target'])
```

Out[210…

target	0	1
slope		
0	12	9
1	91	49
2	35	107

slope-the slope of the peak exercise ST segment

- 0: Upsloping: better heart rate with exercise (uncommon)
- 1: Flatsloping: minimal change (typical healthy heart)
- 2: Downsloping signs of unhealthy heart

6. Experimentation

If you haven't hit your evaluation metric target yet......ask yourself...

- Could you collect more data?
- Could you try a better model? Like CatBoost or XGBoost?
- Could you improve the current models ? (beyond what is done so far)

If model is good enough (you have hit your evaluation metric) how would you export it and share it with others?

In [ ]: