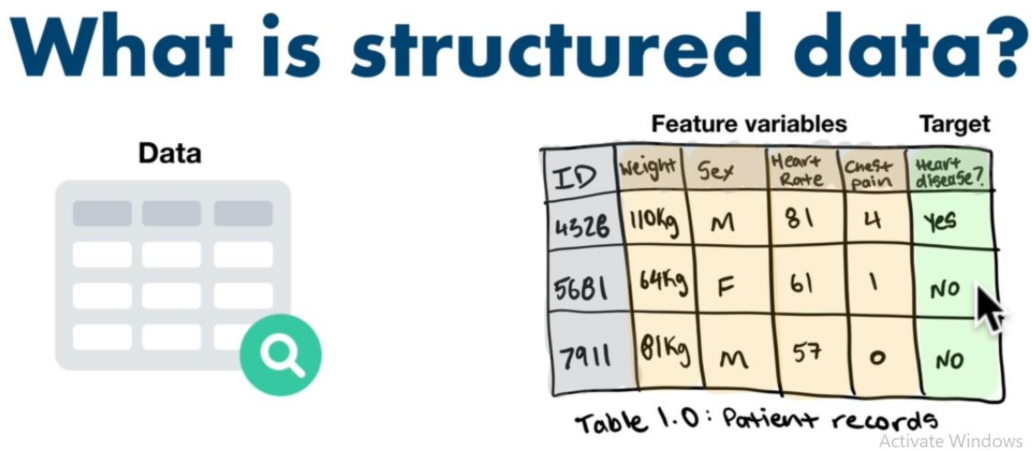
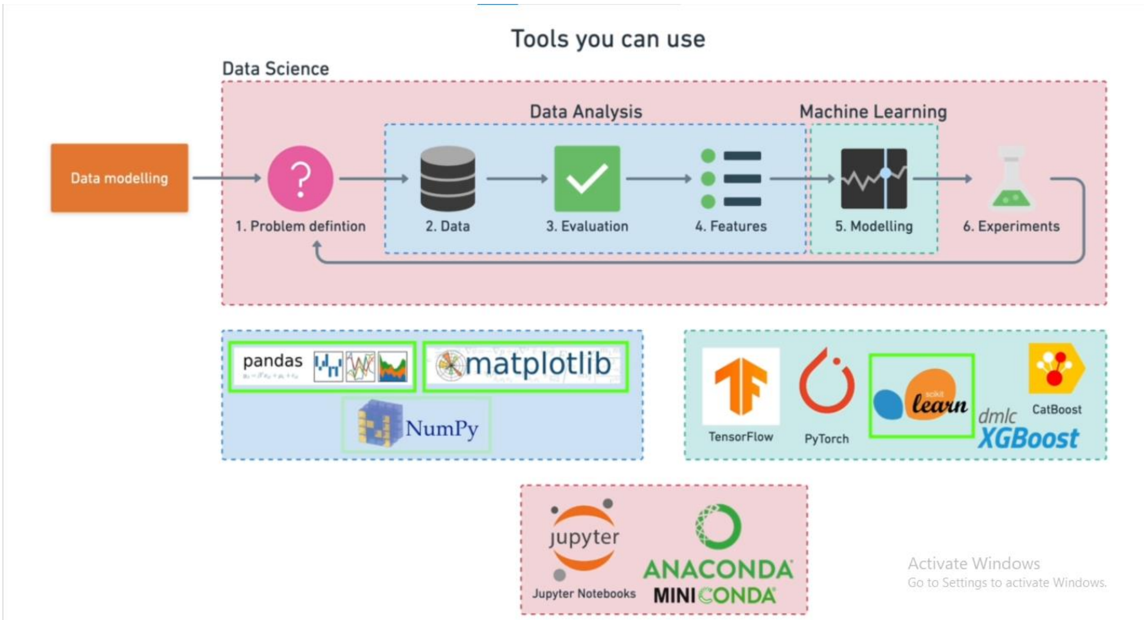
ML-7

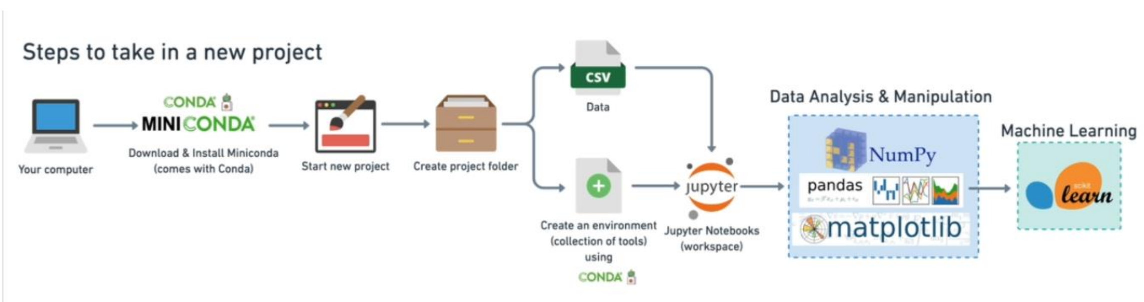
PROJECT 1

Muhammad Naqeeb | Machine-Learning / project 1 | August 18, 2021


What is structured data?
Data
Feature variables
Target
ID
Weight
Sex
Heart Rate
Chest pain
Heart disease?
4328
110Kg
M
81
4
Yes
5681
64Kg
F
61
1
NO
7911
81Kg
M
57
0
NO
Table 1.0: Patient records
Activate Windows


Steps in a full machine learning project
Data collection
What we're going to cover
Data modelling
Deployment
What problem are we trying to solve?
What data do we have?
What defines success?
What features should we model?
What kind of model should we use?
What have we tried/ what else can we try?
1. Problem defintion
2. Data
3. Evaluation
4. Features
5. Modelling
6. Experiments
Iterative proces


Tools you can use
Data Science
Data Analysis
Machine Learning
Data modelling
1. Problem defintion
2. Data
3. Evaluation
4. Features
5. Modelling
6. Experiments
pandas
matplotlib
NumPy
TensorFlow
PyTorch
learn
dmlc
XGBoost
CatBoost
jupyter
Jupyter Notebooks
ANACONDA
MINICONDA
Activate Windows
Go to Settings to activate Windows.


Steps to take in a new project
Your computer
CONDA
MINICONDA
Download & Install Miniconda
(comes with Conda)
Start new project
Create project folder
CSV
Data
Create an environment
(collection of tools)
using
CONDA
jupyter
Jupyter Notebooks
(workspace)
Data Analysis & Manipulation
NumPy
pandas
matplotlib
Machine Learning
learn

Predicting heart disease using machine learning

This notebook looks into using various Python based machine learning and data science libraries which are our tools 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're going to take the following approach.

1. Problem defination
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 came from the Cleavland database from UCI machine learning repository There is also a version of it available on Kaggle.

3. Evaluation

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

4. Features

This is where you'll get different information about each of the features in your data.

Preparing the tools.

We're going to use Panda's matplotlib and Numpy for data analysis and manipulation.

```
# Import all the tools we need

# Regular EDA (exporatory data analysis) and plotting libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#we want our plots to appear inside the notebook

%matplotlib inline

#Model from Scikit-learn
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# Model Evaluation
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
```

Load data

```
df = pd.read_csv("11.2 heart-disease.csv")
df
```

```
In [10]:  df = pd.read_csv("11.2 heart-disease.csv")
          df
```

Out[10]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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

303 rows × 14 columns

Data Explortion (exploratory data analysis or EDA)

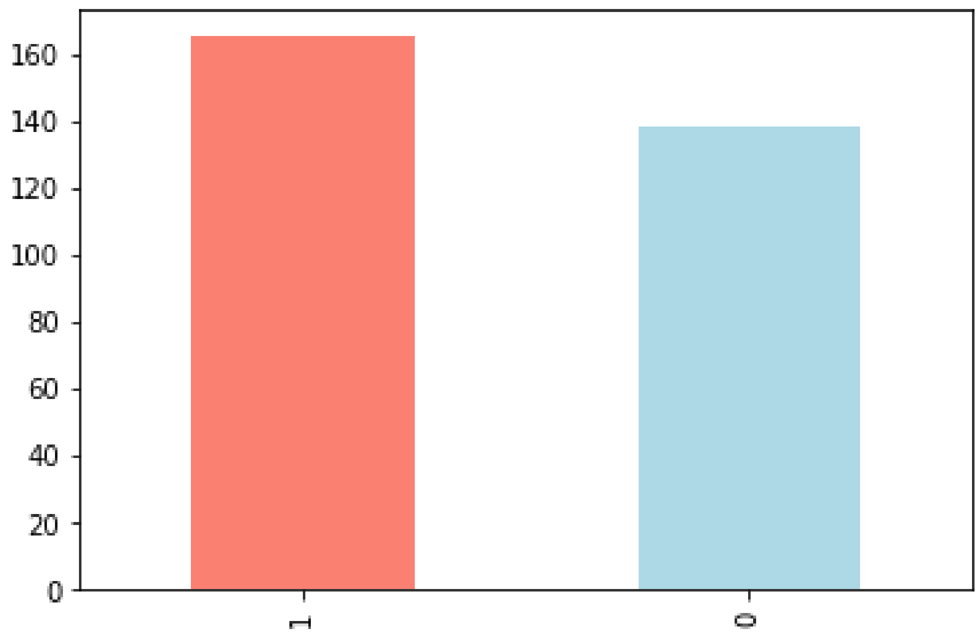
So the goal here is to find out more about the data and become a subject matter expert on the data set you're working with

1. what questions are you trying to solve.
2. what kind of data do we have. And how do we 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.

```
# let's find out how many of each class there
df["target"].value_counts()
```

```
In [16]:  # let's find out how many of each class there
          df["target"].value_counts()

Out[16]:  1    165
          0    138
          Name: target, dtype: int64
```

```
df["target"].value_counts().plot(kind ="bar", color=["salmon",
"lightblue"] )
```




```
df.shape
```

```
df.info()
```

```
In [14]: df.shape
Out[14]: (303, 14)
```

```
In [18]: 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
```

```
df.isna().sum() # checking missing
```

```
In [20]: df.isna().sum() # checking missing
Out[20]: 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
```

```
df.describe()
```

```
In [21]: df.describe()
```

Out[21]:

	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
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
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
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
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
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
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
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

Heart Disease Frequency according to Sex

```
df.sex.value_counts()
```

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

```
In [23]: df.sex.value_counts()

Out[23]: 1    207
         0     96
         Name: sex, dtype: int64
```

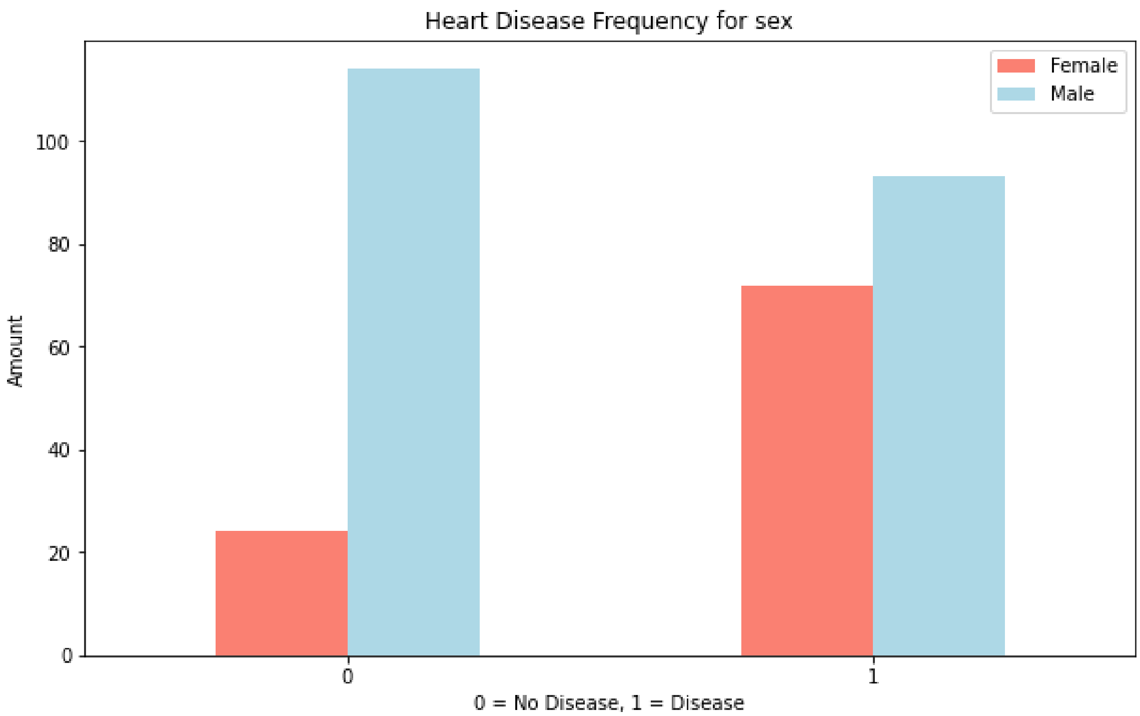
```
In [25]: # compare target column with sex column
         pd.crosstab(df.target, df.sex)
```

Out[25]:

sex	0	1
target		
0	24	114
1	72	93

```
# Create a plot of crostab
pd.crosstab(df.target, df.sex).plot(kind="bar",
                                    figsize=(10,6),
                                    color=["salmon", "lightblue"])

plt.title("Heart Disease Frequency for sex")
```

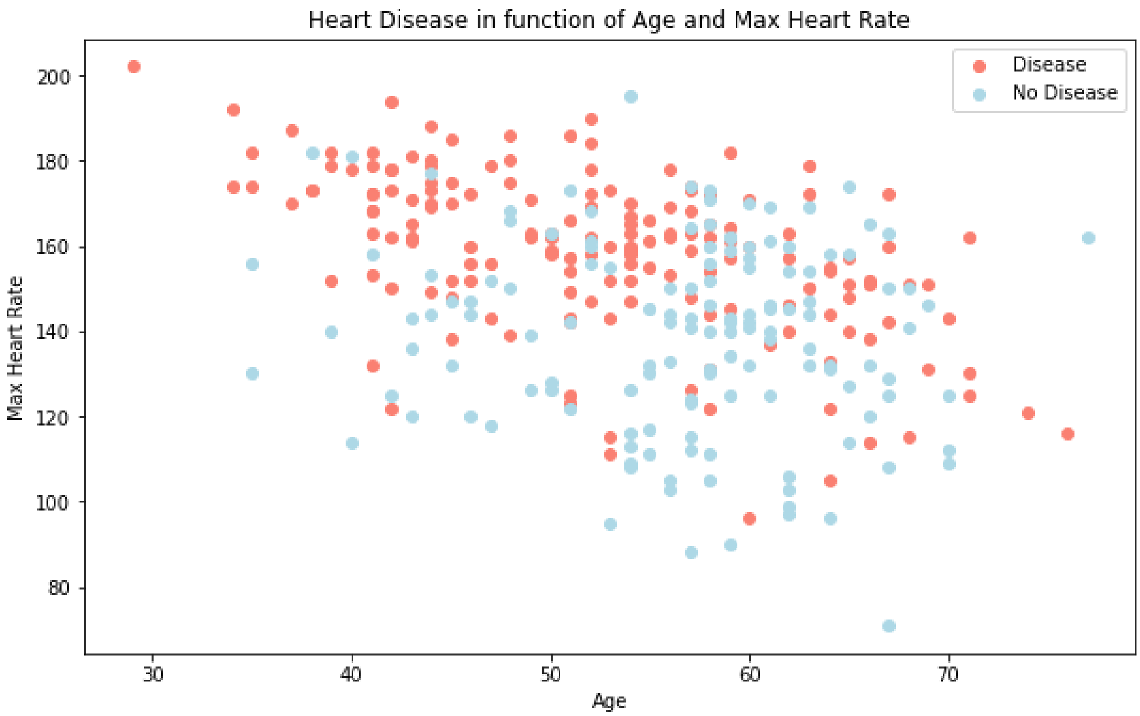
```
plt.xlabel("0 = No Disease, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female","Male"])
plt.xticks(rotation = 0);
```



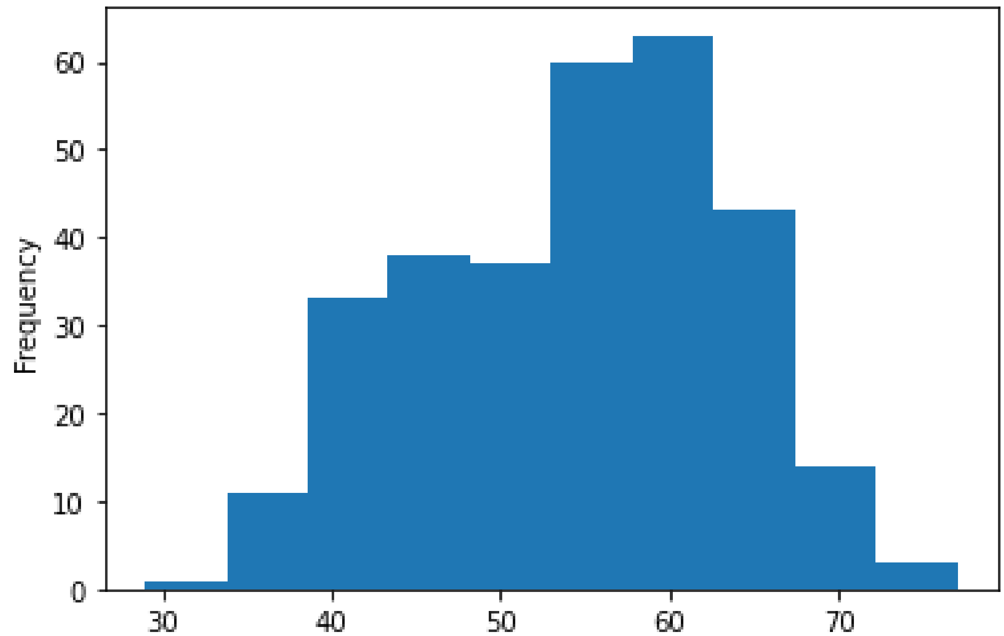
Age vs. Max Heart Rate for Heart Disease

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

# Scatter with positive example
plt.scatter(df.age[df.target==1],
            df.thalach[df.target==1],
            c="salmon")

# Scatter with negative example
plt.scatter(df.age[df.target==0],
            df.thalach[df.target==0],
            c="lightblue")
```

```
# Add some helpful info
plt.title("Heart Disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease","No Disease"])
```

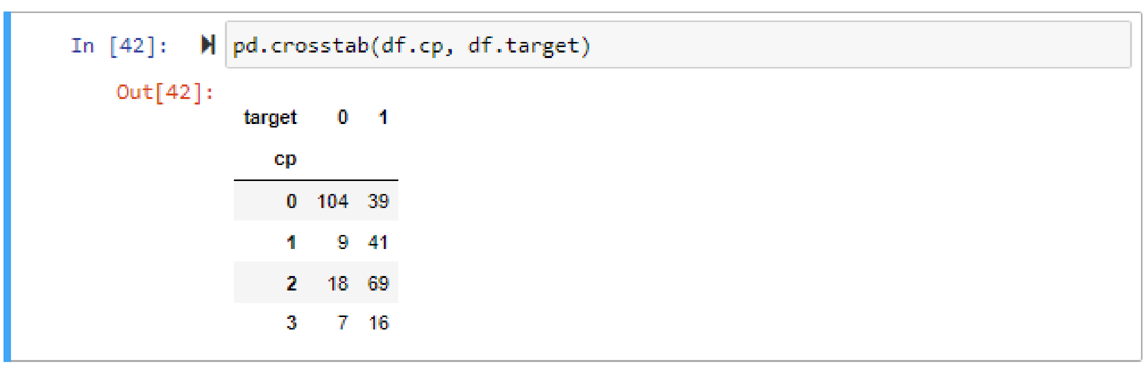



```
# Chech the distribution of the age column with a histogram
df.age.plot.hist();
```



Heart Disease Frequency per Chest pain Type

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

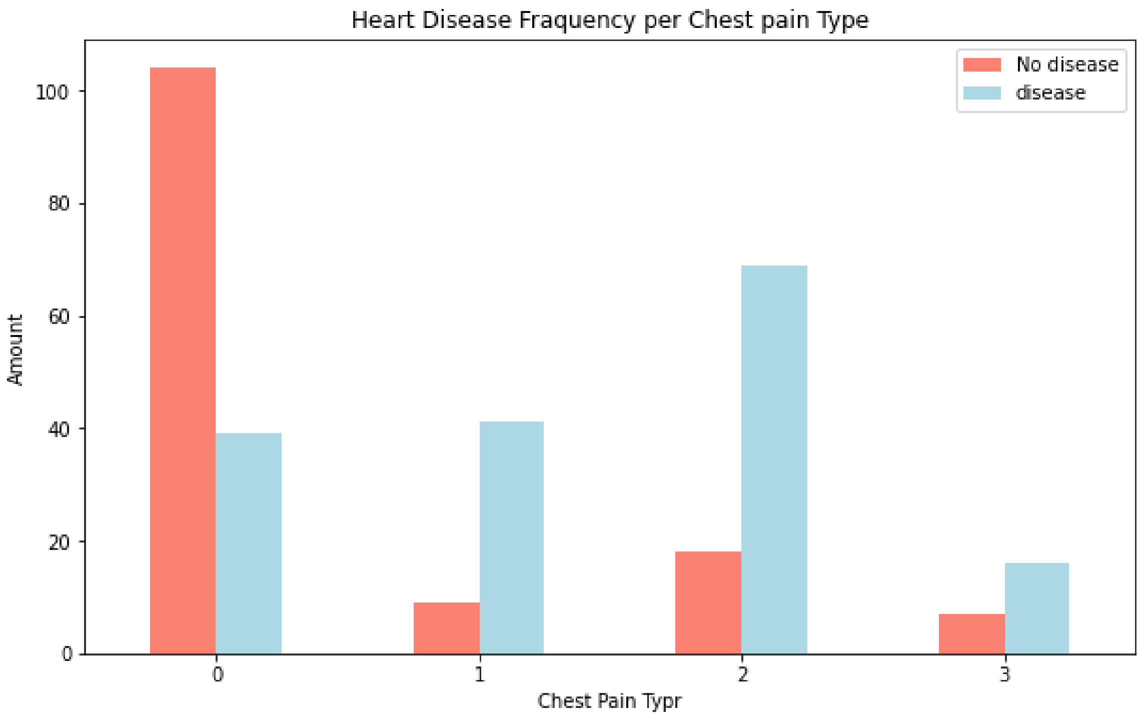

In [42]: pd.crosstab(df.cp, df.target)

Out[42]:

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

```
# Make the crosstab more visual
# Create a plot of crostab
pd.crosstab(df.cp, df.target).plot(kind="bar",
                                   figsize=(10,6),
                                   color=["salmon", "lightblue"])

plt.title("Heart Disease Fraquency per Chest pain Type")
plt.xlabel("Chest Pain Typr")
plt.ylabel("Amount")
```

```
plt.legend(["No disease","disease"])
plt.xticks(rotation = 0);
```




```
# Make a correlation matrix
df.corr()
```

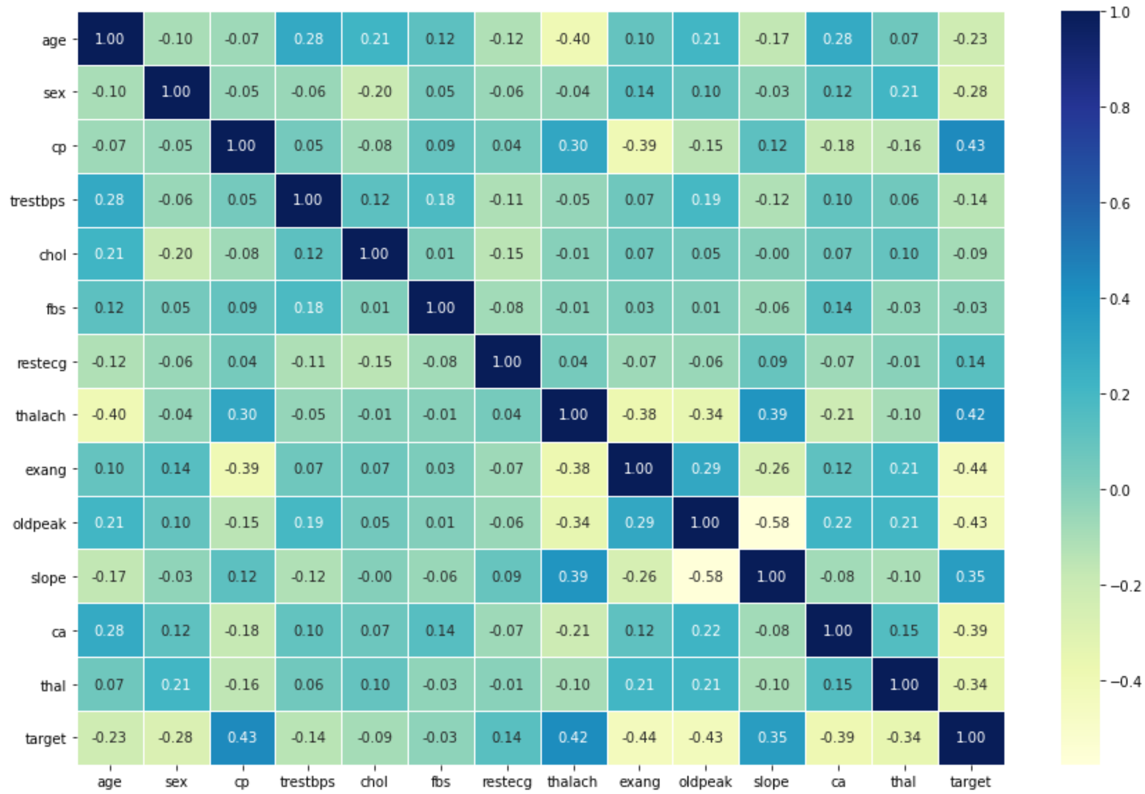
In [44]:

```
# Make a correlation matrix
df.corr()
```

Out[44]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	tar
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	-0.2254
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	-0.2809
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	0.4337
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	-0.1449
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	-0.0852
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	-0.0280
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	0.1372
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	0.4217
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	-0.4367
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	-0.4306
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	0.3458
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	-0.3917
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	-0.3440
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	1.0000

```
# Let's make our correlation matrix a little prettier

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")
```



A higher positive value means a potential positive correlation and a higher negative value means a potential negative correlation or a decrease. this is saying is as C.P. goes up the target value also increases

Correction: Negative correlation = a relationship between two variables in which one variable increases as the other decreases

6. Modelling

```
# Split data into x and y
X = df.drop("target", axis = 1)
y=df["target"]
```

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

# # split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
```

Now we've got our data split into training and test sets, it's time to build a machine learning model.

We'll train it (find the patterns) on the training set.

And we'll test it (use the patents) on the test set.

We're going to try 3 different machine learning models:

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

```
# Put models in a dictionary
models = {"Logistic Regression":LogisticRegression(),
          "KNN":KNeighborsClassifier(),
          "Random 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 different scikit-Learn machine learning
models
    X_train : training data (no labels)
```

```
    X_test : trsting data (no labels)
    y_train : training labels
    y_test : test labels
    """

    # Set random seed
    np.random.seed(42)
    # Make a dictionary to keep model score
    model_scores={}

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

    return model_scores
```

```
# let's see how each model perform
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
```

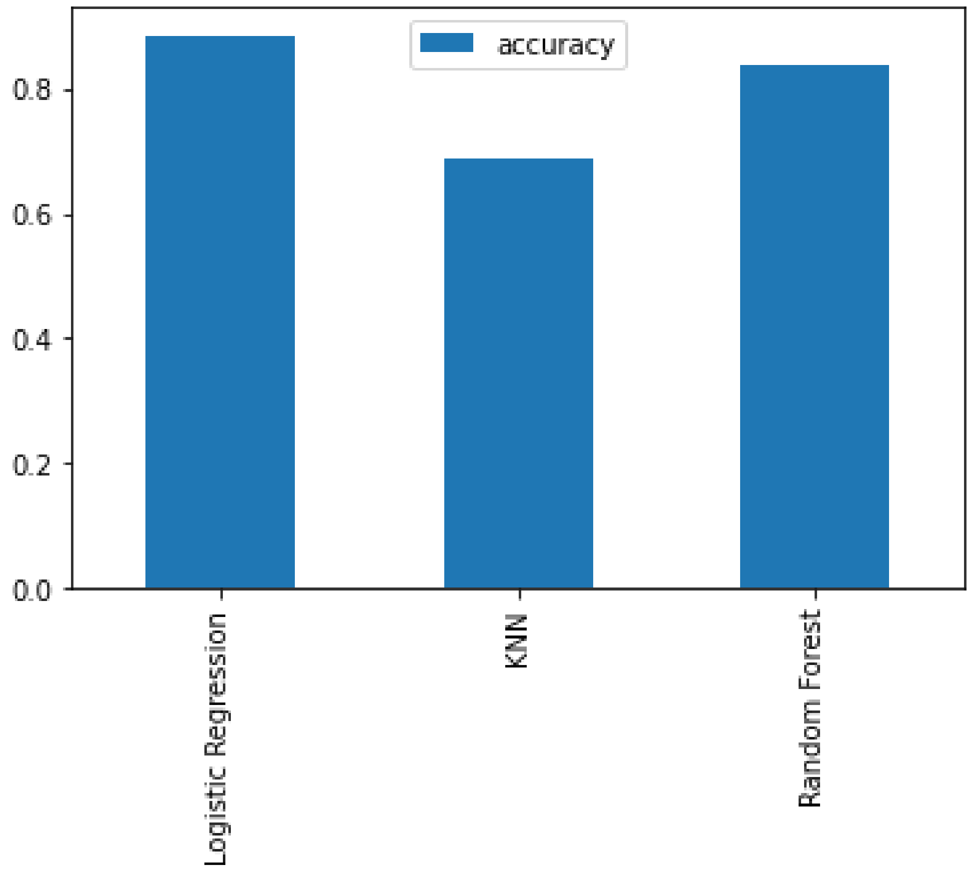
```
In [24]:  # Let's see how each model perform
          model_scores = fit_and_score(models = models,
                                       X_train = X_train
                                     , X_test = X_test
                                     , y_train = y_train
                                     , y_test = y_test)

          model_scores
```

```
C:\Users\toshiba c55t-a\desktop\sample_project\env\lib\site-packages\sk
learn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed t
o 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[24]: {'Logistic Regression': 0.8852459016393442,
          'KNN': 0.6885245901639344,
          'Random Forest': 0.8360655737704918}
```

Model Comparison

```
model_compare  = pd.DataFrame(model_scores, index=["accuracy"])
model_compare.T.plot.bar()
```



Now we've got a baseline model... and we know a models first predictions aren't always what we should base our next steps off.

What should we do?

let's look at the following

- Hyperparameter tuning
- Feature importance
- Confusion matrix
- Cross-validation
- Precision
- Recall
- F1 score
- Classification report
- ROC curve
- Area under the curve (AUC)

Classification and Regression metrics

Classification	Regression
Accuracy	**R^2 (r-squared)**
Precision	Mean absolute error (MAE)
Recall	Mean squared error (MSE)
F1	Root mean squared error (RMSE)

Bold = default evaluation in Scikit-Learn

HYPERPARAMETER TUNING

```
# Let's tune KNN

train_scores = []
test_scores = []

# Create a list of 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 algorithms
    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))
```

```
train_scores
```

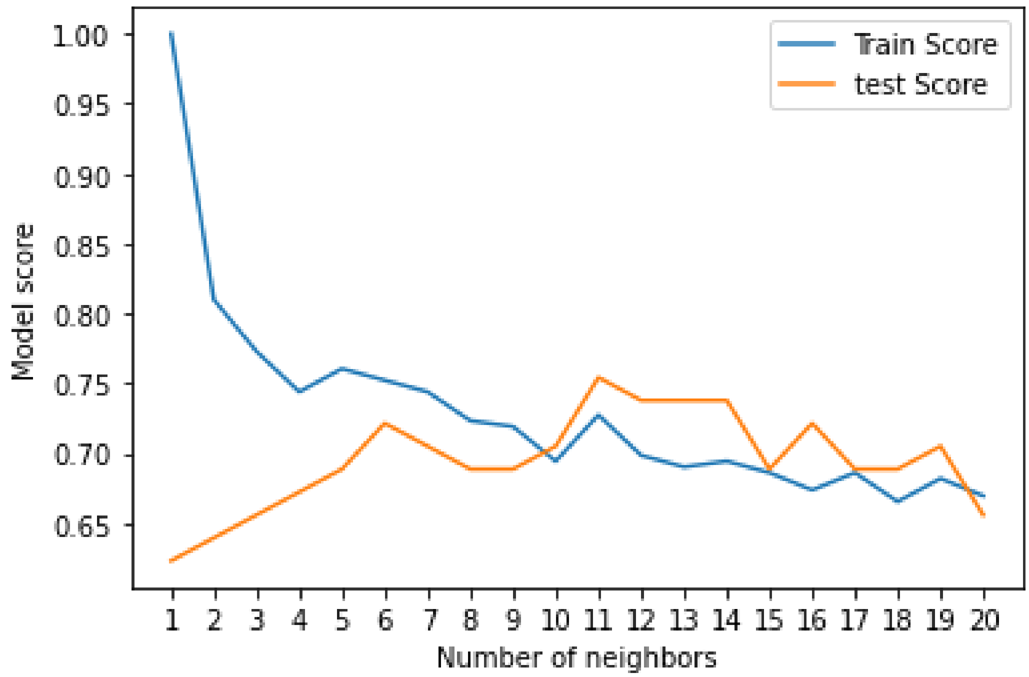
```
In [32]:  train_scores

Out[32]: [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]
```

```
test_scores
```

```
In [33]:  test_scores
```

```
Out[33]: [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]
```

```
plt.plot(neighbors, train_scores, label="Train Score")

plt.plot(neighbors, test_scores, label="test Score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()

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



HYPERPARAMETER TUNING WITH RANDOMIZEDSEARCHCV

We're going tp tune

- LogisticRegression()
- RandomForestClassifier()

... using RandomizedSearchCV

```
# create a hyper parameter 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've got hyperparameter grid setup for each of our models, let's tune them using RandomizedSearchCV

```
# 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 for logisticRegression
rs_log_reg.fit(X_train, y_train)
```

```
In [42]:  # 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 for logisticRegression
          rs_log_reg.fit(X_train, y_train)

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

Out[42]:  RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_iter=20,
                             param_distributions={'C': array([1.00000000e-04, 2.6
          3665090e-04, 6.95192796e-04, 1.83298071e-03,
                 4.83293024e-03, 1.27427499e-02, 3.35981829e-02, 8.85866790e-02,
                 2.33572147e-01, 6.15848211e-01, 1.62377674e+00, 4.28133240e+00,
                 1.12883789e+01, 2.97635144e+01, 7.84759970e+01, 2.06913808e+02,
                 5.45559478e+02, 1.43844989e+03, 3.79269019e+03, 1.00000000e+0
          4]),
                                                  'solver': ['liblinear']},
                             verbose=True)
```

```
rs_log_reg.best_params_
```

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

```
In [43]:  rs_log_reg.best_params_
Out[43]: {'solver': 'liblinear', 'C': 0.23357214690901212}

In [45]:  rs_log_reg.score(X_test, y_test)
Out[45]: 0.9344262295081968
```

Now we've tuned logisticRegression(), let's do the same for RandomForestClassifier().....

```
np.random.seed(42)
# Setup random hyperparameter search for RandomForestClassifier()
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                               param_distributions=rf_grid,
                               cv=5,
                               n_iter = 20,
                               verbose = True)

# fit random hyperparameter search for RandomForestClassifier()
rs_rf.fit(X_train, y_train)
```

```
In [48]:  np.random.seed(42)
           # Setup random hyperparameter search for RandomForestClassifier()
           rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                                      param_distributions=rf_grid,
                                      cv=5,
                                      n_iter = 20,
                                      verbose = True)

           # fit random hyperparameter search for RandomForestClassifier()
           rs_rf.fit(X_train, y_train)

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

Out[48]:   RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=20,
                              param_distributions={'max_depth': [None, 3, 5, 10],
                                                   'min_samples_leaf': array([ 1,
           3,  5,  7,  9, 11, 13, 15, 17, 19]),
                                                   'min_samples_split': array([ 2,
           4,  6,  8, 10, 12, 14, 16, 18]),
                                                   'n_estimators': array([ 10,  6
           0, 110, 160, 210, 260, 310, 360, 410, 460, 510, 560, 610,
                  660, 710, 760, 810, 860, 910, 960])},
                              verbose=True)
```

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

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

```
In [49]:  # Find the best hyperparameters
           rs_rf.best_params_

Out[49]:   {'n_estimators': 210,
            'min_samples_split': 4,
            'min_samples_leaf': 19,
            'max_depth': 3}

In [50]:  # Evaluate the randomized search RandomForestClassifier model
           rs_rf.score(X_test, y_test)

Out[50]:   0.8688524590163934
```

HYPERPARAMETERS TUNING WITH GRIDSEARCHCV

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

```
# different hyperparameters for our LogisticRegression model
log_reg_grid = {"C": np.logspace(-4, 4, 30),
               "solver":["liblinear"]}

# Setup grid hyperparameter search for LogisticRegression model
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)
```

```
In [53]: # different hyperparameters for our LogisticRegression model
         log_reg_grid = {"C": np.logspace(-4, 4, 30),
                        "solver":["liblinear"]}

         # Setup grid hyperparameter search for LogisticRegression model
         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[53]: GridSearchCV(cv=5, estimator=LogisticRegression(),
                      param_grid={'C': array([1.00000000e-04, 1.88739182e-04, 3.
         56224789e-04, 6.72335754e-04,
                1.26896100e-03, 2.39502662e-03, 4.52035366e-03, 8.53167852e-03,
                1.61026203e-02, 3.03919538e-02, 5.73615251e-02, 1.08263673e-01,
                2.04335972e-01, 3.85662042e-01, 7.27895384e-01, 1.37382380e+00,
                2.59294380e+00, 4.89390092e+00, 9.23670857e+00, 1.74332882e+01,
                3.29034456e+01, 6.21016942e+01, 1.17210230e+02, 2.21221629e+02,
                4.17531894e+02, 7.88046282e+02, 1.48735211e+03, 2.80721620e+03,
                5.29831691e+03, 1.00000000e+04]),
                                  'solver': ['liblinear']},
                      verbose=True)
```

```
# best hyperparameters
gs_log_reg.best_params_
```

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

```
In [54]:  # best hyperparameters
          gs_log_reg.best_params_

Out[54]:  {'C': 0.20433597178569418, 'solver': 'liblinear'}

In [55]:  # Evaluate the grid search LogisticRegression  model
          gs_log_reg.score(X_test, y_test)

Out[55]:  0.8852459016393442
```

Evaluting our tuned machine learning classifier, beyond accuracy

- ROC curve and AUC score
- 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.

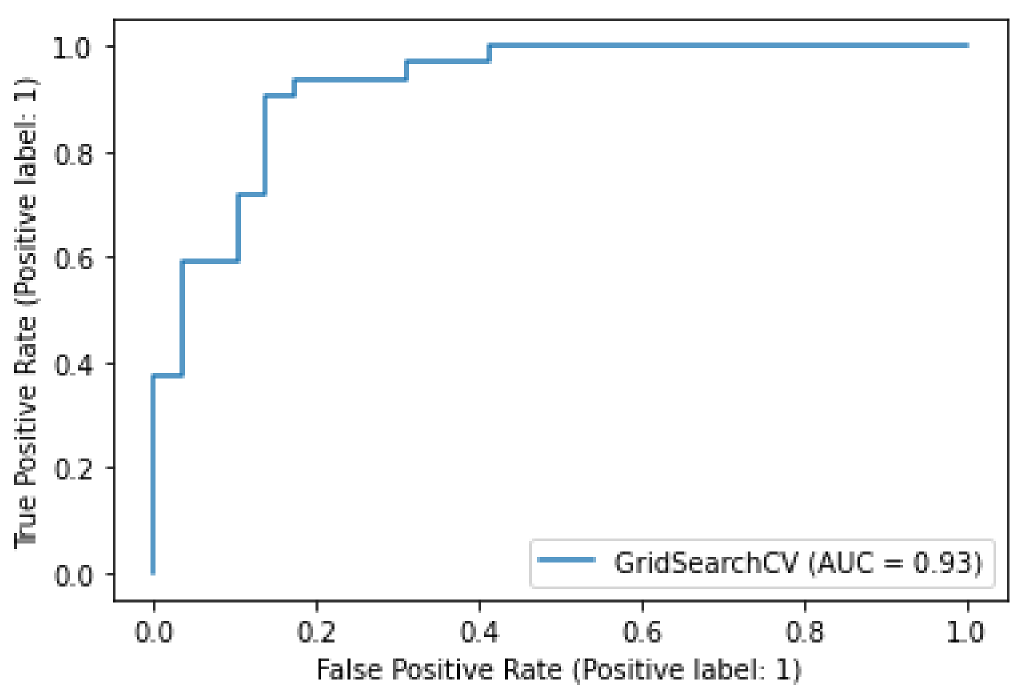
```
# Make predictions with tuned model
y_preds = gs_log_reg.predict(X_test)
```

```
y_preds
```

```
y_test
```

```
In [56]: # Make predictions with tuned model
         y_preds = gs_log_reg.predict(X_test)

In [57]: y_preds

Out[57]: 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)
```

```
# Plot ROC curve and calculate and caculate AUC metric
plot_roc_curve(gs_log_reg, X_test, y_test)
```

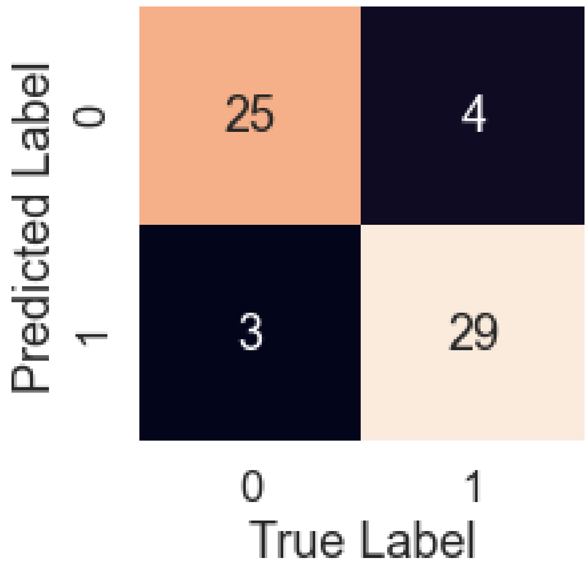



```
# Confution matrix
```

```
print(confusion_matrix(y_test, y_preds))
```

```
In [60]:  # Confution matrix
          print(confusion_matrix(y_test, y_preds))

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

```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, y_preds):
    """
    Plots a nice looking confusion metrix 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 a ROC curve and AUC metric and a confusion matrix, Let's get a classification report as well as cross validated precision, recall and f1-score

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

```
In [67]:  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
```

Calculate evaluation metrics using cross validation

we're going to calculate accuracy, precision, recall and f1 score of our model using cross-validation, and to do so we'll be using `cross_val_score()`

```
# Check best hyperparameters
gs_log_reg.best_params_
```

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

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

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

```
In [63]: # Check best hyperparameters
         gs_log_reg.best_params_
Out[63]: {'C': 0.20433597178569418, 'solver': 'liblinear'}

In [65]: # Create a new classifier with best parameters
         clf = LogisticRegression(C=0.20433597178569418,
                                  solver= "liblinear")

In [70]: # Cross-validated accuracy
         cv_acc = cross_val_score(clf,
                                 X,
                                 y,
                                 cv=5,
                                 scoring = "accuracy")
         cv_acc
Out[70]: array([0.81967213, 0.90163934, 0.86885246, 0.88333333, 0.75      ])

In [72]: cv_acc = np.mean(cv_acc)
         cv_acc
Out[72]: 0.8446994535519124
```

```
# Cross-validated precision
cv_precision = cross_val_score(clf,
                               X,
                               y,
                               cv=5,
                               scoring = "precision")
cv_precision = np.mean(cv_precision)
cv_precision
```

```
# Cross-validated recall
cv_recall = cross_val_score(clf,
                            X,
                            y,
                            cv=5,
                            scoring = "recall")
cv_recall = np.mean(cv_recall)
cv_recall
```

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

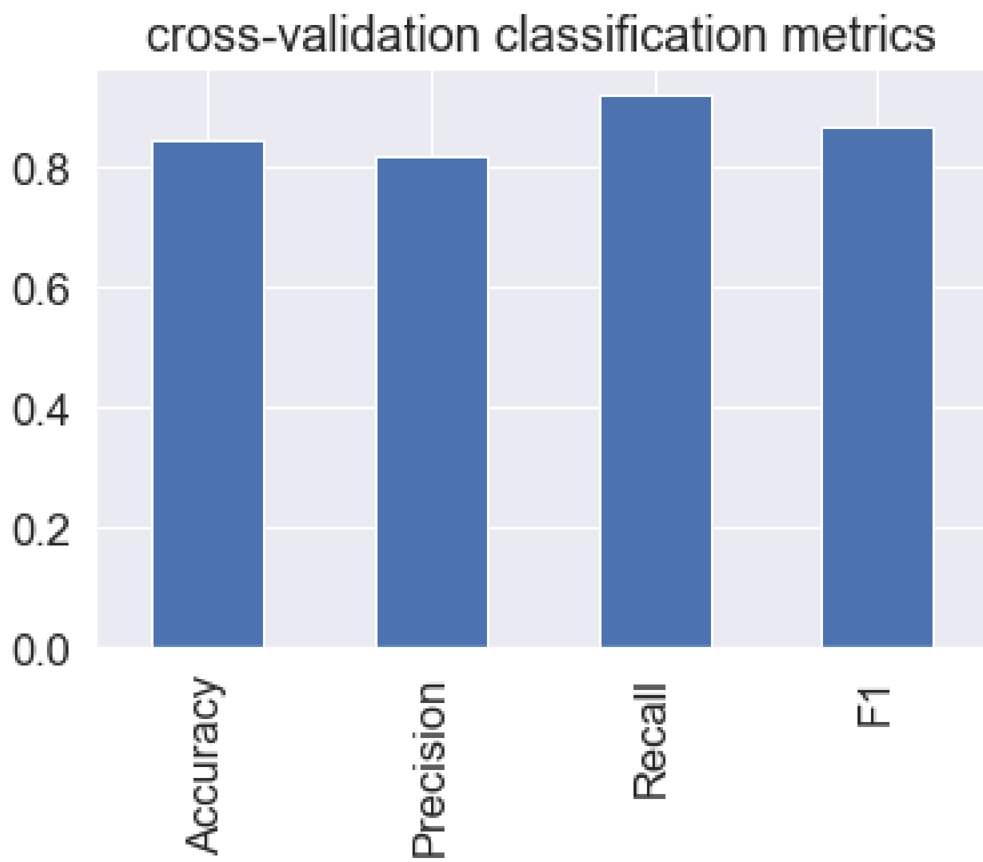
In [73]:
```python
# Cross-validated precision
cv_precision = cross_val_score(clf,
                               X,
                               y,
                               cv=5,
                               scoring = "precision")
cv_precision = np.mean(cv_precision)
cv_precision
```
Out[73]: 0.8207936507936507

In [74]:
```python
# Cross-validated recall
cv_recall = cross_val_score(clf,
                            X,
                            y,
                            cv=5,
                            scoring = "recall")
cv_recall = np.mean(cv_recall)
cv_recall
```
Out[74]: 0.9212121212121213

In [75]:
```python
# 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[75]: 0.8673007976269721

```python
# Visualize cross-validation metrics
cv_metrics = pd.DataFrame({"Accuracy":cv_acc,
                           "Precision":cv_precision,
                           "Recall":cv_recall,
                           "F1":cv_f1},
                          index=[0])


cv_metrics.T.plot.bar(title="cross-validation classification
metrics",
legend = False)
```



Feature Importance

Feature Importance is another as 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 impoetance is to search for **(MODEL NAME) feature importance**

let's find the feature important for now logisticRegression model...

```
# Fit an instance of logisticRegression

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

clf.fit(X_train, y_train)
```

```
# check coef_
clf.coef_
```

```
In [83]:  # Fit an instance of LogisticRegression

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

          clf.fit(X_train, y_train)

Out[83]:  LogisticRegression(C=0.20433597178569418, solver='liblinear')
```

```
In [84]:  # check coef_
          clf.coef_

Out[84]:  array([[ 0.00316728, -0.86044652,  0.6606704 , -0.01156993, -0.0016637
          5,
                   0.04386107,  0.31275848,  0.02459362, -0.60413081, -0.5686280
          3,
                   0.45051628, -0.63609898, -0.67663373]])
```

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

```
In [85]:  # Match coef's of features to columns
          feature_dict = dict(zip(df.columns, list(clf.coef_[0])))
          feature_dict

Out[85]:  {'age': 0.0031672806268220445,
           'sex': -0.8604465226286001,
           'cp': 0.6606703996492814,
           'trestbps': -0.011569930743501303,
           'chol': -0.001663745833540806,
           'fbs': 0.043861067871676124,
           'restecg': 0.3127584791782968,
           'thalach': 0.024593615091850 37,
           'exang': -0.6041308102637141,
           'oldpeak': -0.5686280255489925,
           'slope': 0.4505162810238786,
           'ca': -0.6360989756865822,
           'thal': -0.6766337272356 1}
```

```
#Visualize feature importance
feature_df = pd.DataFrame(feature_dict, index=[0])
feature_df.T.plot.bar(title="Feature Importance", legend=False)
```


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

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

In [89]:
```
pd.crosstab(df["sex"], df["target"])
```
Out[89]:

target	0	1
sex		
0	24	72
1	114	93

In [90]:
```
pd.crosstab(df["slope"], df["target"])
```
Out[90]:

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

6. Experimentation

If you haven't hit your evaluation metric 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 we're done so far)
- If your model is good enough (you have hit your evaluation metric) how would you export it and share it with other?