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

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 came from the Cleavland data from the UCI Machine Learning Repository.https://archive.ics.uci.edu/dataset/45/heart+disease

There is also a version of it available in Kaggle.https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data

3. Evaluation

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

4. Features

Create a data dictionary

- id (Unique id for each patient)
- age (Age of the patient in years)
- origin (place of study)
- sex (Male/Female)
- origin (place of study)
- sex (Male/Female)
- cp chest pain type

- trestbps resting blood pressure
- chol (serum cholesterol in mg/dl)
- fbs (if fasting blood sugar > 120 mg/dl)
- restecg (resting electrocardiographic results)
- thalach: maximum heart rate achieved
- exang: exercise-induced angina (True/ False)
- oldpeak: ST depression induced by exercise relative to rest
- slope: the slope of the peak exercise ST segment
- ca: number of major vessels (0-3) colored by fluoroscopy
- target 1 or 0

Preparing the tools

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

In [202…

```
# Import all the tools

# Regular EDA(exploratory 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

# Models 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 RocCurveDisplay
```

Load Data

In [96]:

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

Out[96]:

	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 Exploration (exploratory data analysis EDA)

The goal is to study more about the data and become a subject matter expert on the dataset you are working with.

> 1. What question(s) are you trying to solve?
> 2. What kind of data do we have and how do we treat different types?
> 3. What is missing from the data and how do you deal with it?
> 4. What are the outliers and why should you care about them?
> 5. How can you add, change or remove features to get more out of data?

In [97]:

```
# Let's find how many of each classes are there
df["target"].value_counts()
```

Out[97]:

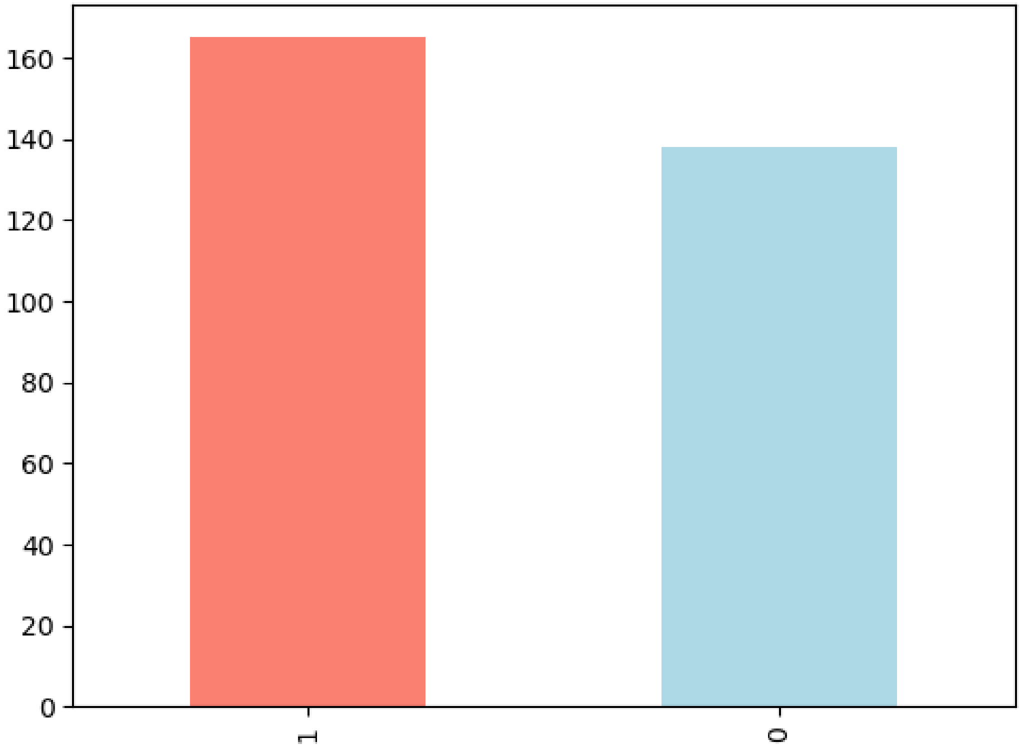
```
1    165
0    138
Name: target, dtype: int64
```

In [98]:

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

Out[98]:

```
<Axes: >
```



In [99]:
```
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 [100…
```
# Are there any missing values
df.isna().sum()
```

Out[100]:
```
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 [101…
```
df.describe()
```

Out[101]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
count	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
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000

Heart Disease Frequency according to sex

In [102…
```
df.sex.value_counts()
```

Out[102]:
```
1    207
0     96
Name: sex, dtype: int64
```

1 = male & 2 = female

In [103…
```
# Compare target column with sex column
pd.crosstab(df.target, df.sex)
```

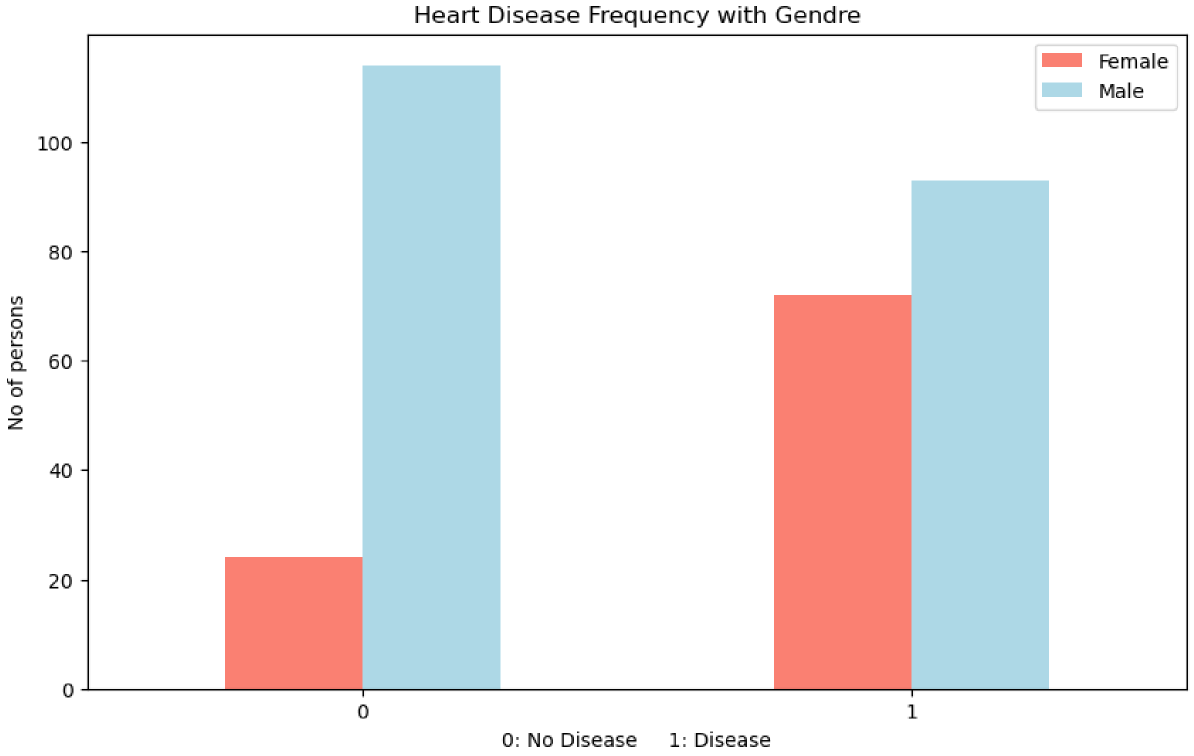
Out[103]:

sex	0	1
target		
0	24	114
1	72	93

In [104…
```
#   Create a plot of crosstab

pd.crosstab(df.target, df.sex).plot(kind = "bar",
                                    figsize = (10, 6),
                                    color = ["salmon", "lightblue"],
                                    ylabel = "No of persons")
plt.title("Heart Disease Frequency with Gendre")
plt.xlabel("0: No Disease     1: Disease")
```

```
plt.legend(["Female", "Male"])
plt.xticks(rotation = 0);
```



In [105…

```
df["thalach"].value_counts()
```

Out[105]:

```
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

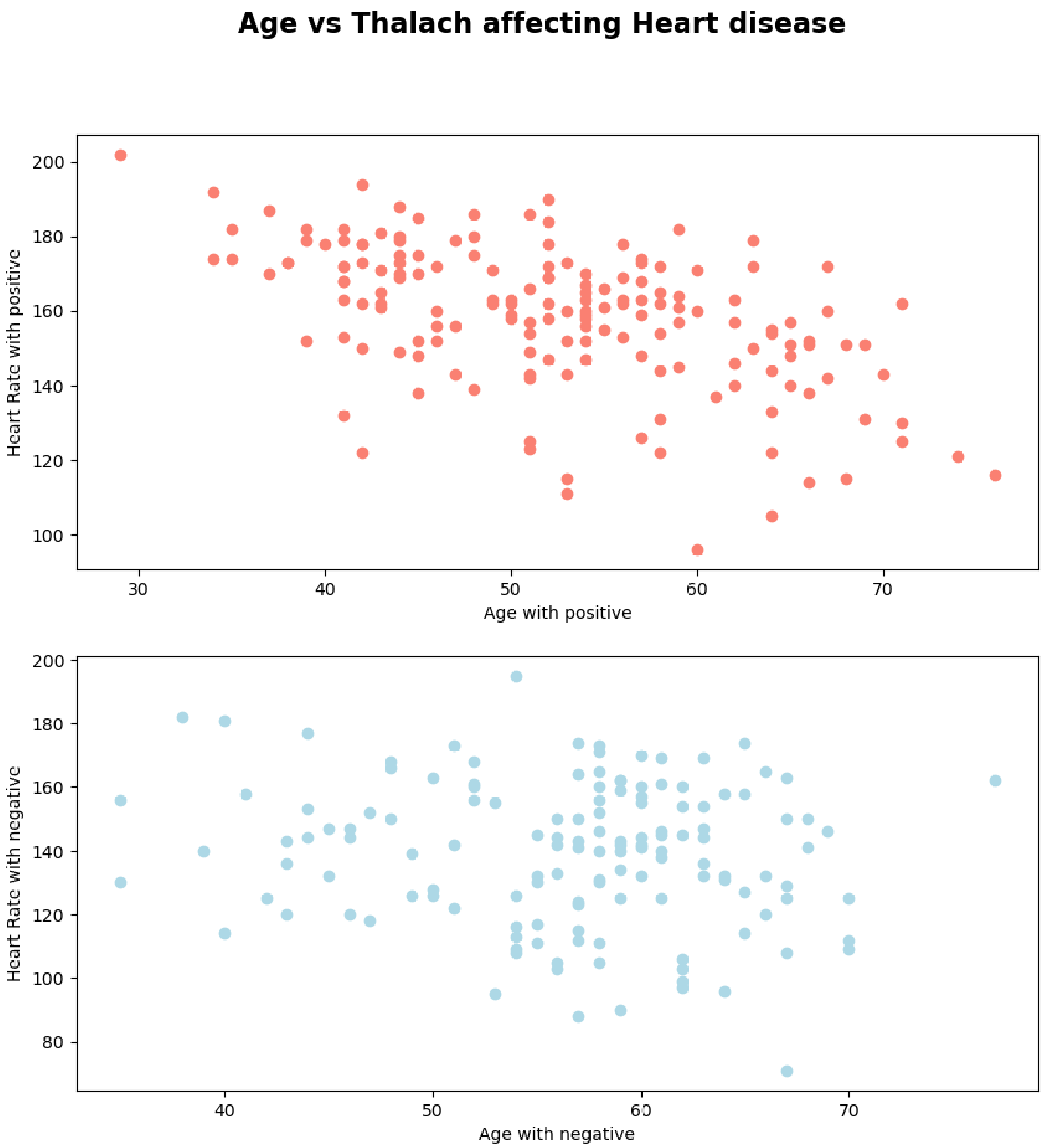
In [106…

```
fig, ax = plt.subplots(figsize = (10, 10),
                      nrows = 2,
                       ncols = 1
                      )
# Scatter with positive
ax[0].scatter(x = df.age[df.target == 1],
           y = df.thalach[df.target == 1],
           c = "salmon"
          )
ax[0].set( xlabel = "Age with positive",
       ylabel = "Heart Rate with positive"
      )

# Scatter with negative
ax[1].scatter(x = df.age[df.target == 0],
           y = df.thalach[df.target == 0],
           c = "lightblue"
                )
ax[1].set(
       xlabel = "Age with negative",
       ylabel = "Heart Rate with negative"
```

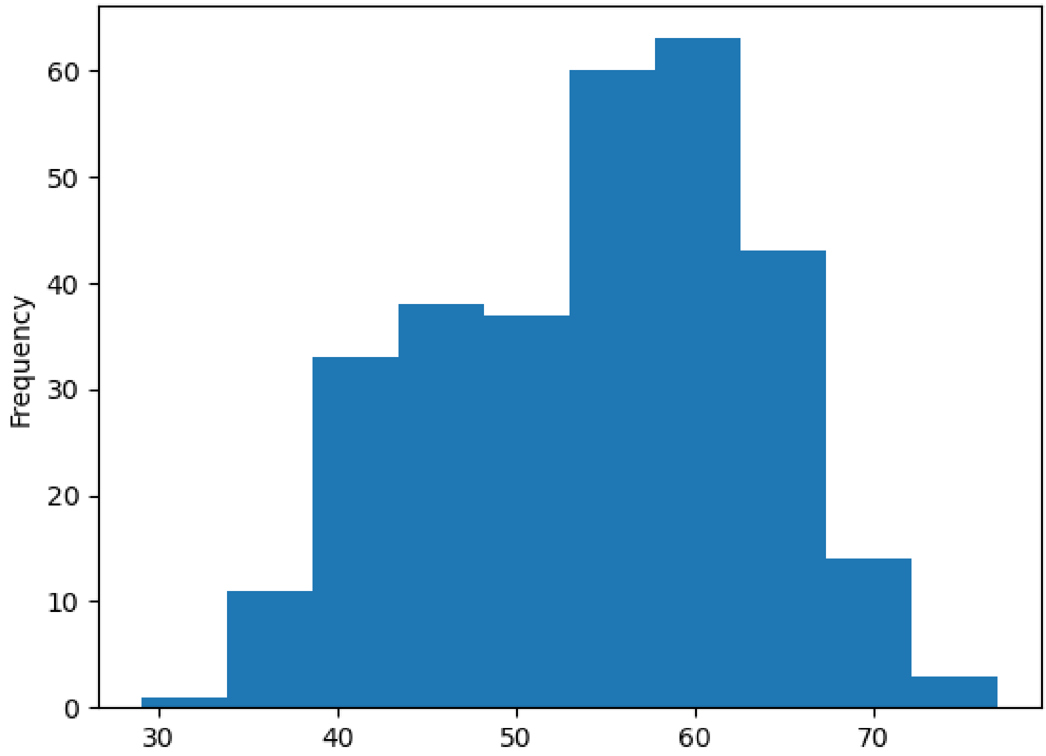
```
)

fig.suptitle("Age vs Thalach affecting Heart disease", fontsize = 16, fontweight = "bold
```



In [107…

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



Heart Disease Frequency per heart pain type

In [108…

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

Out[108]:

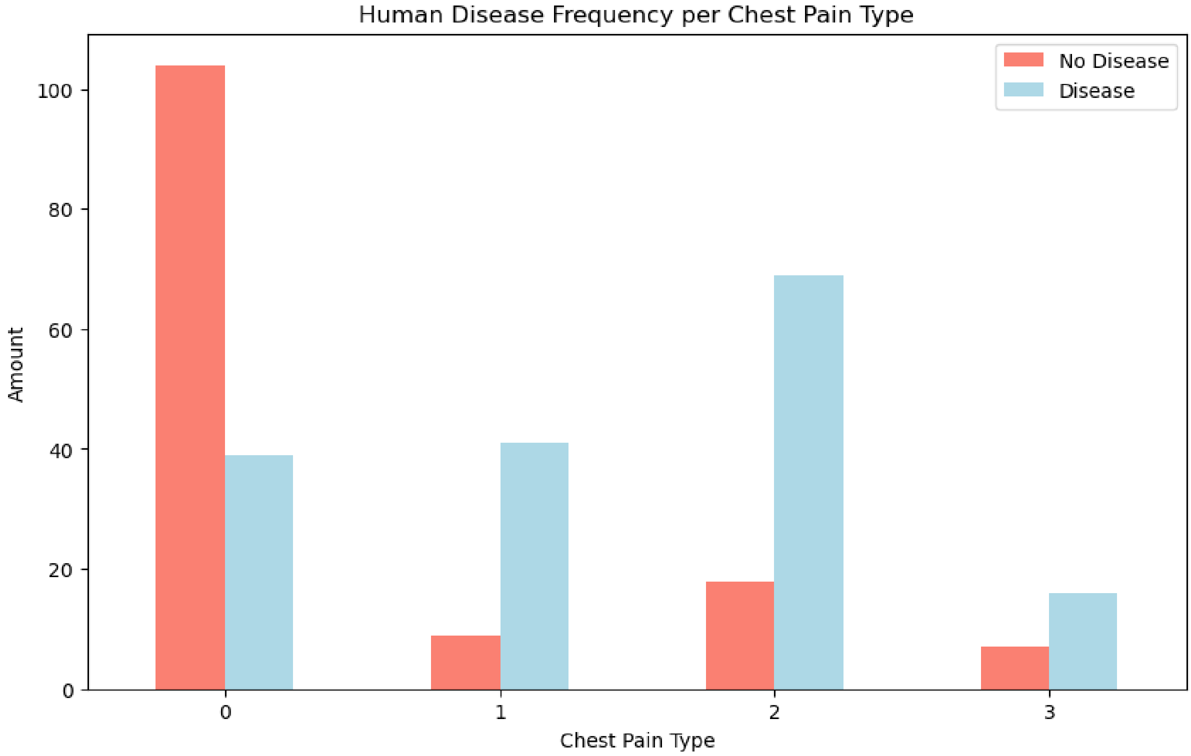
target	0	1
cp		
0	104	39
1	9	41
2	18	69
3	7	16

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

In [109…

```
pd.crosstab(df.cp, df.target).plot(kind = "bar",
                                  figsize = (10, 6),
                                  color = ["salmon", "lightblue"])

plt.title("Human Disease Frequency per Chest Pain Type")
plt.xlabel("Chest Pain Type")
plt.ylabel("Amount")
plt.legend(["No Disease", "Disease"])
plt.xticks(rotation = 0);
```



In [110…

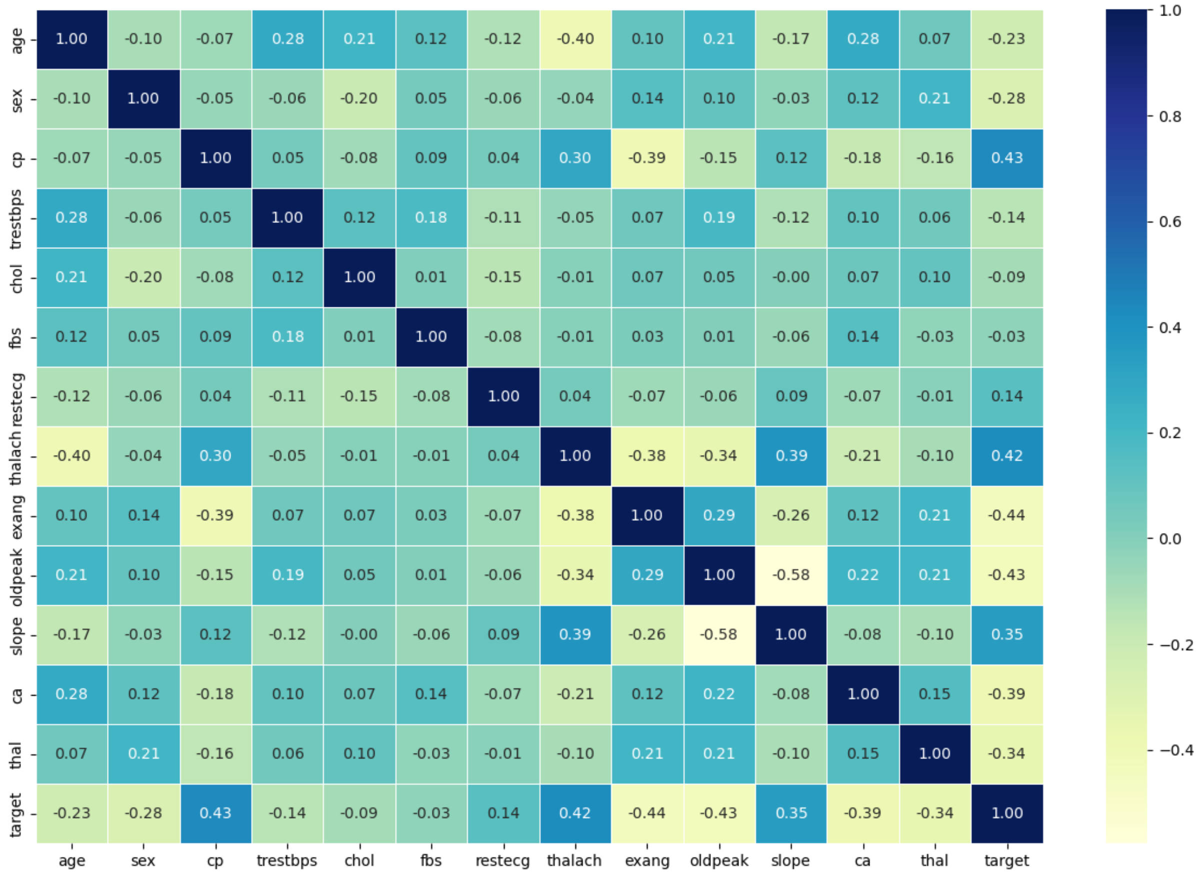
```
# Make a correlation matrix
df.corr()
```

Out[110]:

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

In [111…

```
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize = (15, 10))
ax = sns.heatmap(corr_matrix,
                 annot = True,
                 linewidths = 0.5,
                 fmt = ".2f",
                 cmap = "YlGnBu");
#bottom, top = ax.get_ylim()
#ax.set_ylim(bottom + 0.5, top - 0.5);
```



5. Modelling

In [112…

```
df.head()
```

Out[112]:

	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 [115…

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

X.shape, y.shape, df.shape
```

Out[115]:

```
((303, 13), (303,), (303, 14))
```

In [116…

```
# Split into training set and test set
np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

In [117…

```
X_train
```

Out[117]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
188	50	1	2	140	233	0	1	163	0	0.6	1	1	3
71	51	1	2	94	227	0	1	154	1	0.0	2	1	3
106	69	1	3	160	234	1	0	131	0	0.1	1	1	2
270	46	1	0	120	249	0	0	144	0	0.8	2	0	3
102	63	0	1	140	195	0	1	179	0	0.0	2	2	2

242 rows × 13 columns

In [118…

```
y_train
```

Out[118]:

```
132    1
202    0
196    0
75     1
176    0
      ..
188    0
71     1
106    1
270    0
102    1
Name: target, Length: 242, dtype: int64
```

Now we have got our data split into train and test set and we have to decide upon which Machine-Learning model we should use.

We will train (find the pattern) the model on training set and test (use the pattern) the model on test set.

We are trying 3 different machine learning models:

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

In [121…

```
# 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 evaluate a givem machine learning models.
    models: a dict of different Scikit-Learn machine learning models.
    X_train: training data (no labels)
    X_test: testing 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 model in models:
        # Fit the model to the data
        models[model].fit(X_train, y_train)
        # Evaluate the model and append its score to model_scores
        model_scores[model] = models[model].score(X_test, y_test)
    return model_scores
```

In [123…

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

```
D:\DataSci\Study\Project\Project_1\env\Lib\site-packages\sklearn\linear_model\_logistic.
py:458: 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[123]:

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

Model comparison

In [124…

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

In [125…

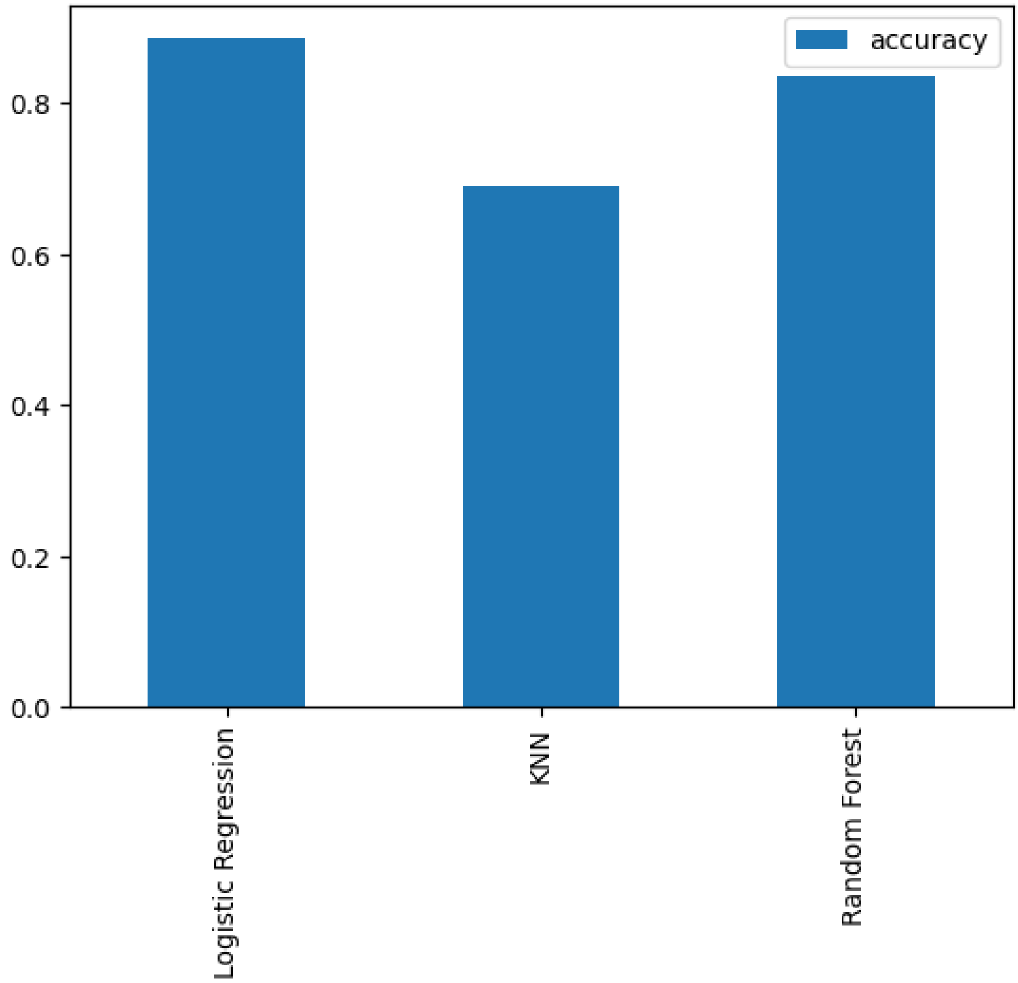
```
model_compare
```

Out[125]:

	Logistic Regression	KNN	Random Forest
accuracy	0.885246	0.688525	0.836066

In [129…

```
model_compare.T.plot(kind = "bar");
```



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

Hyperparameter Tuning

Tuning KNN (By hand)

In [131…

```
# Tuning KNN

train_score = []
test_score = []

# Create a list of different values of n_neighbors
neighbors = range(1, 21)
```

```python
# Setup KNN instance
knn = KNeighborsClassifier()

for i in neighbors:
    knn.set_params(n_neighbors = i)
    # Fit the algorithm
    knn.fit(X_train, y_train)
    # Update training scores list
    train_score.append(knn.score(X_train, y_train))
    # Update test scores list
    test_score.append(knn.score(X_test, y_test))
```

In [133…

```python
train_score
```

Out[133]:

```
[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 [134…

```python
test_score
```

Out[134]:

```
[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 [135…

```python
max(train_score), max(test_score)
```

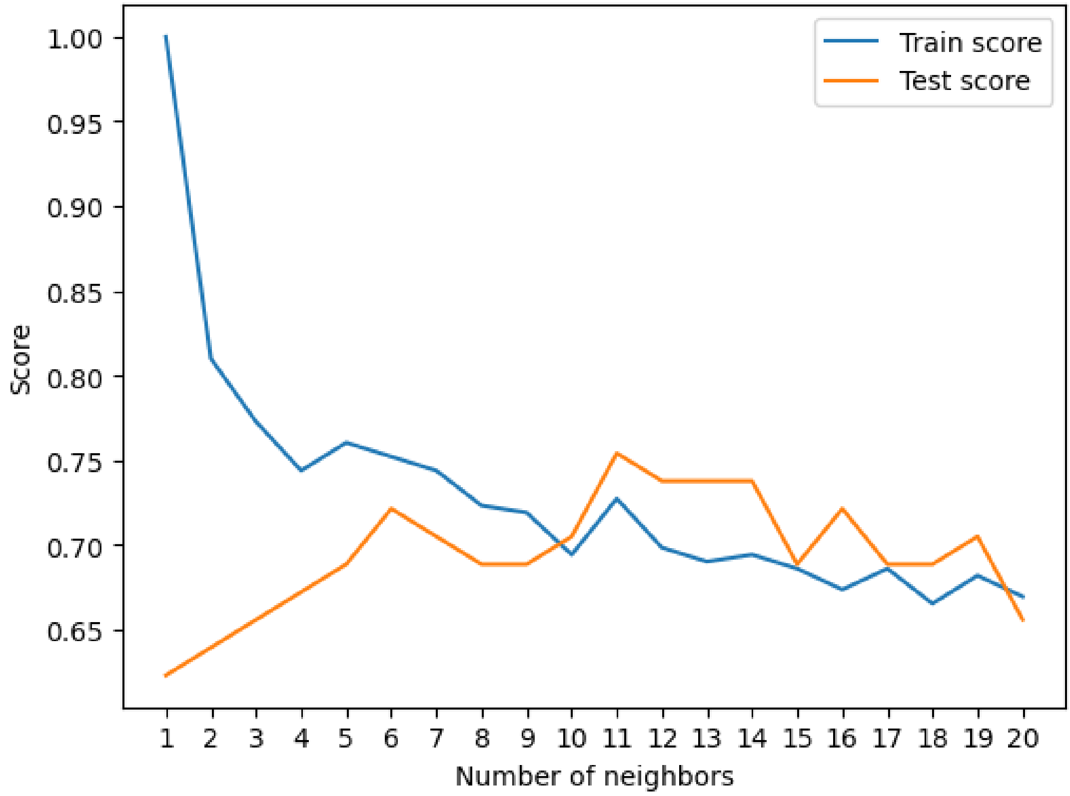
Out[135]:

```
(1.0, 0.7540983606557377)
```

In [141…

```python
plt.plot(neighbors, train_score, label="Train score")
plt.plot(neighbors, test_score, label = "Test score")
plt.xlabel("Number of neighbors")
plt.ylabel("Score")
```

```
plt.legend()
plt.xticks(np.arange(1, 21, 1));
```



Hyperparameter Tuning

Tuning Logistic Regression and Random Forest Classifier (RandomizesSearchCV)

In [150…

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

# Create a hyperparameter grid for RandomForestClassifier
grid_RFC = {"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)}
```

Tune Logistic Regression

In [145…

```
np.random.seed(42)

# Setup random hyperparameter search for LogisticRegression
rs_log_reg = RandomizedSearchCV(LogisticRegression(),
                                param_distributions = grid_LR,
                                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[145]:

```
RandomizedSearchCV
estimator: LogisticRegression
LogisticRegression
```

In [157…

```python
# Finding best hyperparameters
rs_log_reg.best_params_
```

Out[157]:

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

In [154…

```python
# Evaluate the ransomized search LogisticRegression model
rs_log_reg.score(X_test, y_test)
```

Out[154]:

```
0.8852459016393442
```

Tune RandomForestClassifier

In [151…

```python
np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier
rs_rfc = RandomizedSearchCV(RandomForestClassifier(),
                           param_distributions = grid_RFC,
                            cv = 5,
                            n_iter = 20,
                            verbose = True)

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

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

Out[151]:

```
RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [158…

```python
# Finding best hyperparameters
rs_rfc.best_params_
```

Out[158]:

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

In [155…

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

Out[155]:

```
0.8688524590163934
```

Hyperparameter Tuning

By GridSearchCV for LogisticRegression`

Since the LogisticRegression model provides the best score so far, we will try and improve them again using GridSearchCV

In [159…

```python
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[159]:

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

In [161…

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

Out[161]:

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

In [162…

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

Out[162]:

```
0.8852459016393442
```

Evaluating out tuned machine learning classifier, beyond accuracy

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

use cross-validation where possible

In [164…

```
# Make predictions with tuned model

y_preds = gs_log_reg.predict(X_test)
y_preds
```

Out[164]:

```
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 [165…

```
check = pd.DataFrame()
check["True"] = y_test
check["Predicted"] = y_preds
check
```

Out[165]:

	True	Predicted
179	0	0
228	0	1
111	1	1
246	0	0
60	1	1
...	...	...
249	0	0
104	1	1
300	0	0
193	0	0
184	0	0

61 rows × 2 columns

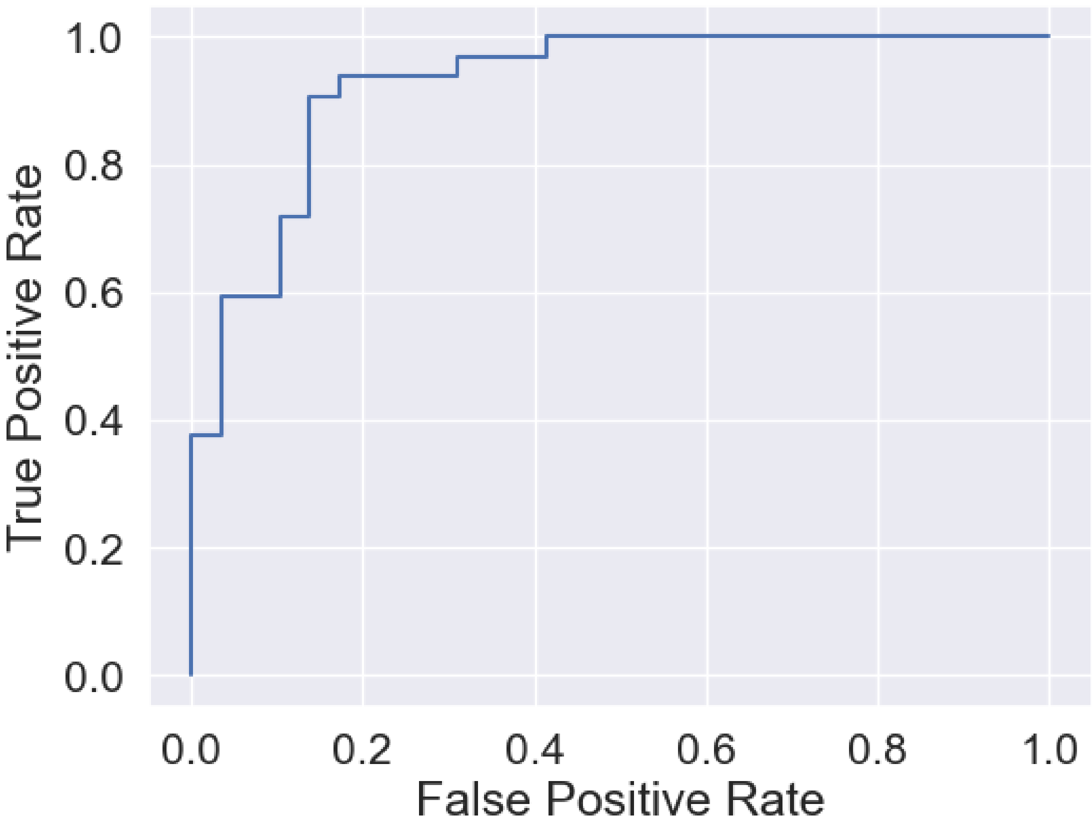
In [171…

```
from sklearn.metrics import roc_curve
```

In [191…

```
#  Plot ROC curve and calculate the AUC metric
y_probs = gs_log_reg.predict_proba(X_test)
y_probs_positive = y_probs[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_probs_positive)
RocCurveDisplay(fpr = fpr, tpr = tpr).plot()
```

Out[191]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x267926a12d0>



In [184…

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

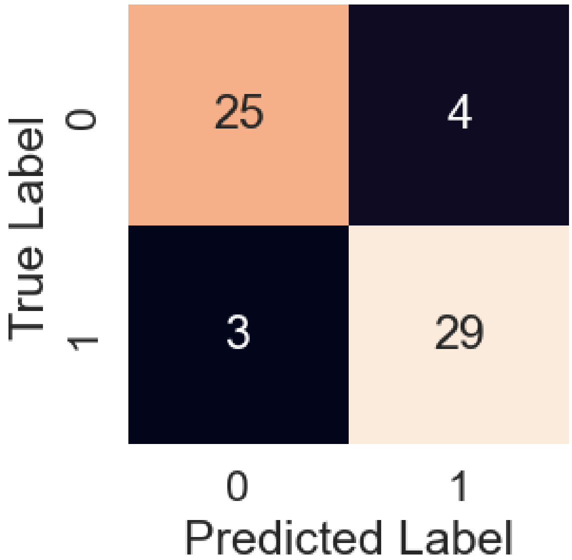
```
[[25  4]
 [ 3 29]]
```

In [189…

```
sns.set(font_scale = 1.5)

def plot_conf_matrix(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("Predicted Label")
    plt.ylabel("True Label")

plot_conf_matrix(y_test, y_preds)
```



Now we got a ROC curve, an AUC metric and a confusion matrix, lets get a classification report as well as cross-validated precision, recall and f1 score.

In [192…

```
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 are going to calculate precision, recall and f1-score of our model using cross-validation and to do so we will bw using `cross_val_score`

In [193…

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

Out[193]:

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

In [195…
```
# Create a new classifier with best parameters
clf = LogisticRegression(C = 0.20433597178569418,
                         solver = "liblinear")
```

In [209…
```
# Cross-validated Accuracy
cv_acc = cross_val_score(clf,
                         X,
                         y,
                         cv = 5,
                         scoring = "accuracy")
cv_acc
```

Out[209]:
```
array([0.81967213, 0.90163934, 0.86885246, 0.88333333, 0.75      ])
```

In [210…
```
cv_acc = np.mean(cv_acc)
cv_acc
```

Out[210]:
```
0.8446994535519124
```

In [212…
```
# Cross-calidated Precision
cv_precision = cross_val_score(clf,
                               X,
                               y,
                               cv = 5,
                               scoring = "precision")
cv_precision
```

Out[212]:
```
array([0.775     , 0.88571429, 0.85714286, 0.86111111, 0.725     ])
```

In [214…
```
cv_precision = np.mean(cv_precision)
cv_precision
```

Out[214]:
```
0.8207936507936507
```

In [215…
```
# Cross-validated Recall
cv_recall = cross_val_score(clf,
                            X,
                            y,
                            cv = 5,
                            scoring = "recall")
cv_recall
```

Out[215]:
```
array([0.93939394, 0.93939394, 0.90909091, 0.93939394, 0.87878788])
```

In [217…
```
cv_recall = np.mean(cv_recall)
cv_recall
```

Out[217]:
```
0.9212121212121213
```

In [218…
```
# Cross-validated f1-score
cv_f1 = cross_val_score(clf,
                        X,
                        y,
                        cv = 5,
                        scoring = "f1")
cv_f1
```

Out[218]:
```
array([0.84931507, 0.91176471, 0.88235294, 0.89855072, 0.79452055])
```

In [220…
```
cv_f1 = np.mean(cv_f1)
cv_f1
```

Out[220]:
```
0.8673007976269721
```

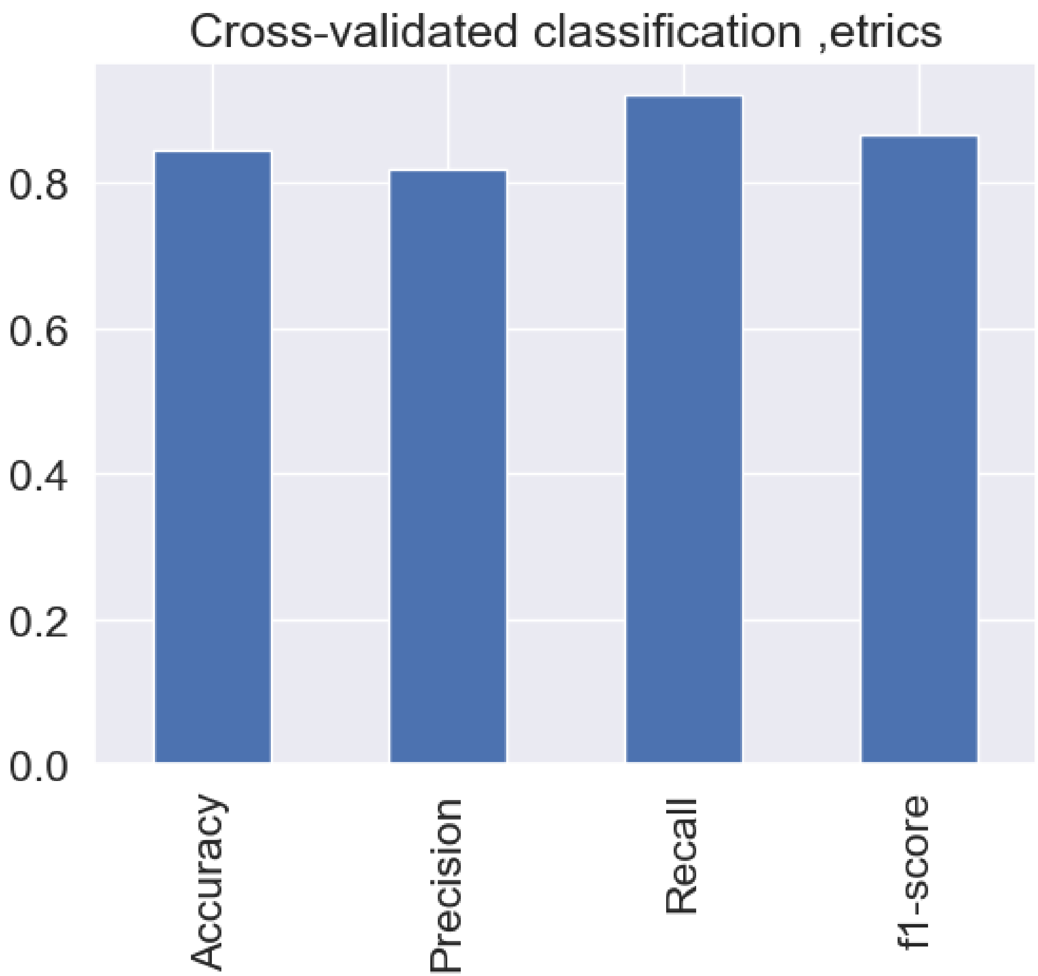
In [223…
```
# Visualise cross-validated metrics
cv_metrics = pd.DataFrame({"Accuracy": cv_acc,
                           "Precision": cv_precision,
                           "Recall": cv_recall,
                           "f1-score": cv_f1}, index = [0])
cv_metrics
```

Out[223]:

	Accuracy	Precision	Recall	f1-score
0	0.844699	0.820794	0.921212	0.867301

In [224…
```
cv_metrics.T.plot.bar(title = "Cross-validated classification ,etrics",
                      legend = False)
```

Out[224]:
```
<Axes: title={'center': 'Cross-validated classification ,etrics'}>
```



Feature Importance

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

In [225…
```
df.head()
```

Out[225]:

	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

Finding feature importance is different for different machine learning models

Let's find the feature importance for LogisticRegression model

In [226…

```
gs_log_reg.best_params_
```

Out[226]:

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

In [229…

```
# Fit an instance of LogisticRegression
clf = LogisticRegression(C = 0.20433597178569418,
                         solver = "liblinear")
clf.fit(X_train, y_train)
```

Out[229]:

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

In [230…

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

Out[230]:

```
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 [231…

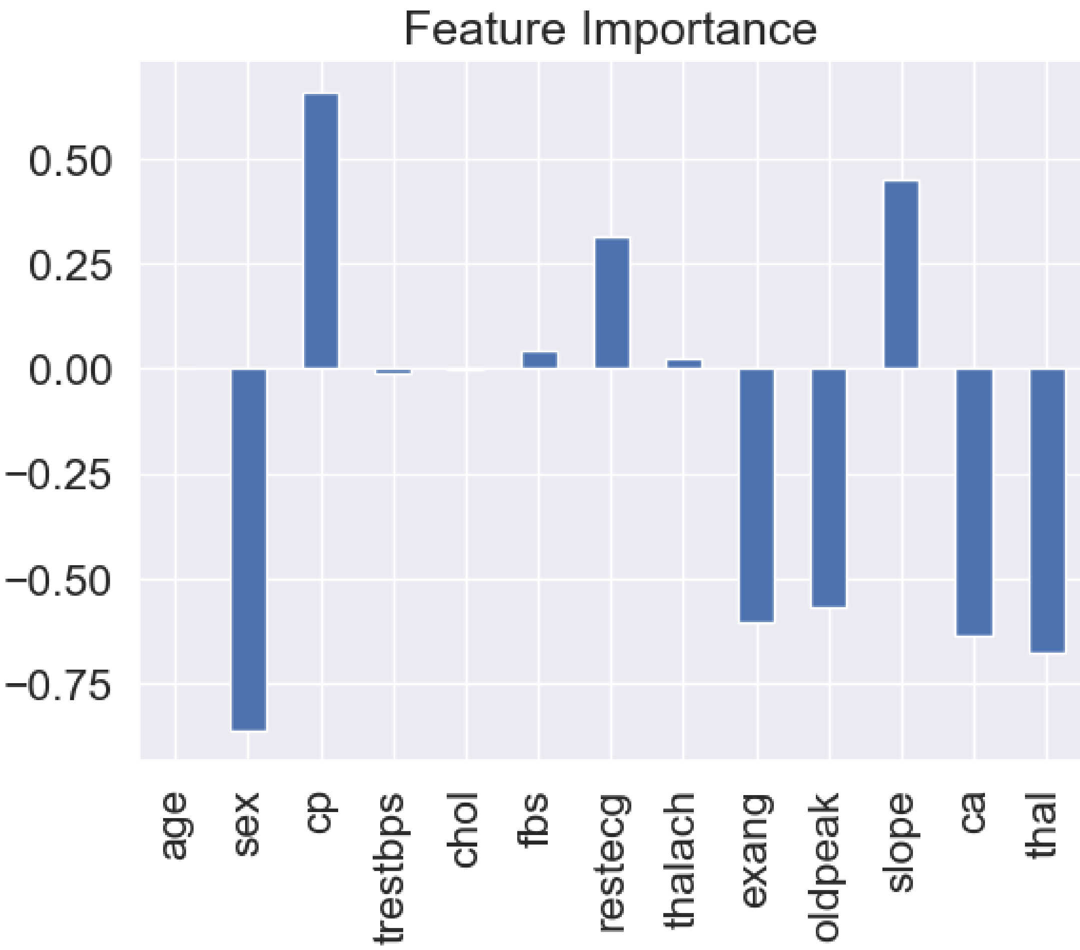
```
# Match coef_'s of the features to columns
feature_dict = dict(zip(df.columns, list(clf.coef_[0])))
feature_dict
```

Out[231]:

```
{'age': 0.0031672801993431563,
 'sex': -0.8604465072345515,
 'cp': 0.6606704082033799,
 'trestbps': -0.01156993168080875,
 'chol': -0.001663744504776871,
 '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 [236…

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



In [237…

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

Out[237]:

target	0	1
sex		
0	24	72
1	114	93

In [239…

```
72/24 , 93/114
```

Out[239]: (3.0, 0.8157894736842105)

In [240…

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

Out[240]:

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

In [241…

```
9/12, 49/91, 107/35
```

Out[241]: (0.75, 0.5384615384615384, 3.057142857142857)

6. Experimentation

- Could you collect more data
- Could you try a better model? Like CatBoost or XGBoost?
- Could we improve the current model?