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 a 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 came from the Cleavland data from the UCI Machine Learning Repository.

There is also a version of it available on Kaggle https://www.kaggle.com/c/heart-disease-uci (https://www.kaggle.com/c/heart-disease-uci)

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 is where you will get different information about each of the features in your data.

Create data dictionary

- age. The age of the patient.
- sex. The gender of the patient. (1 = male, 0 = female).
- cp. Type of chest pain. (1 = typical angina, 2 = atypical angina, 3 = non — anginal pain, 4 = asymptotic).
- trestbps. Resting blood pressure in mmHg.
- chol. Serum Cholestero in mg/dl.
- fbs. Fasting Blood Sugar. (1 = fasting blood sugar is more than 120mg/dl, 0 = otherwise).
- restecg. Resting ElectroCardioGraphic results (0 = normal, 1 = ST-T wave abnormality, 2 = left ventricular hyperthrophy).
- thalach. Max heart rate achieved.

- exang. Exercise induced angina (1 = yes, 0 = no).
- oldpeak. ST depression induced by exercise relative to rest.
- slope. Peak exercise ST segment (1 = upsloping, 2 = flat, 3 = downsloping).
- ca. Number of major vessels (0–3) colored by flourosopy.
- thal. Thalassemia (3 = normal, 6 = fixed defect, 7 = reversible defect).
- num. Diagnosis of heart disease (0 = absence, 1, 2, 3, 4 = present).

Preparing the tools

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

In [2]:

```
# Import all the tools we need

# Regular EDA(Exploring Data Analysis) and plotting librarires
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 Evaluations
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

In [3]:

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

Out[3]:

```
(303, 14)
```

EDA(Data Exploration)

The goal here is to find out more about the data and become a subject matter expert on the data set in which work is being done

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 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 [4]:

```
df.head()
```

Out[4]:

	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 [5]:

```
df.tail()
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targ
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	

In [6]:

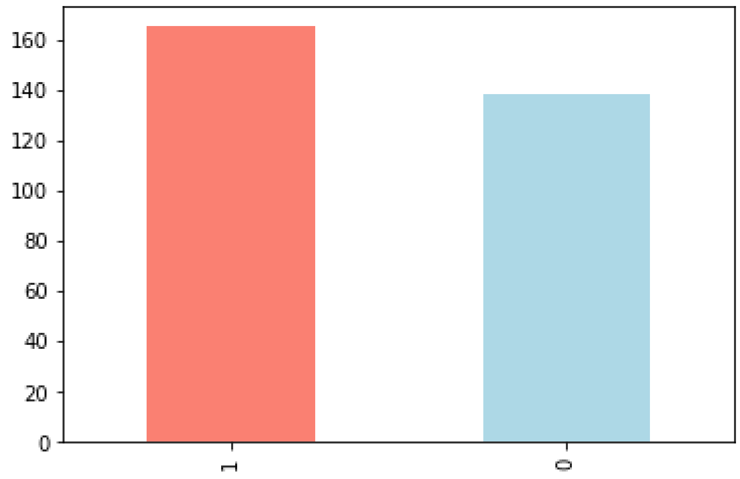
```
# Lets find out how many of each class there are
df["target"].value_counts()
```

Out[6]:

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

In [7]:

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



In [8]:

```
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 [9]:

```
# Are there any missing values?
df.isna().sum()
```

Out[9]:

```
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 [10]:

```
df.describe()
```

Out[10]:

	age	sex	cp	trestbps	chol	fbs	restecg	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	30
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	14
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	2
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	7
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	13
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	15
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	16
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	20

Heart Disease Frequency according to Sex

In [11]:

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

Out[11]:

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

```
In [12]:
```

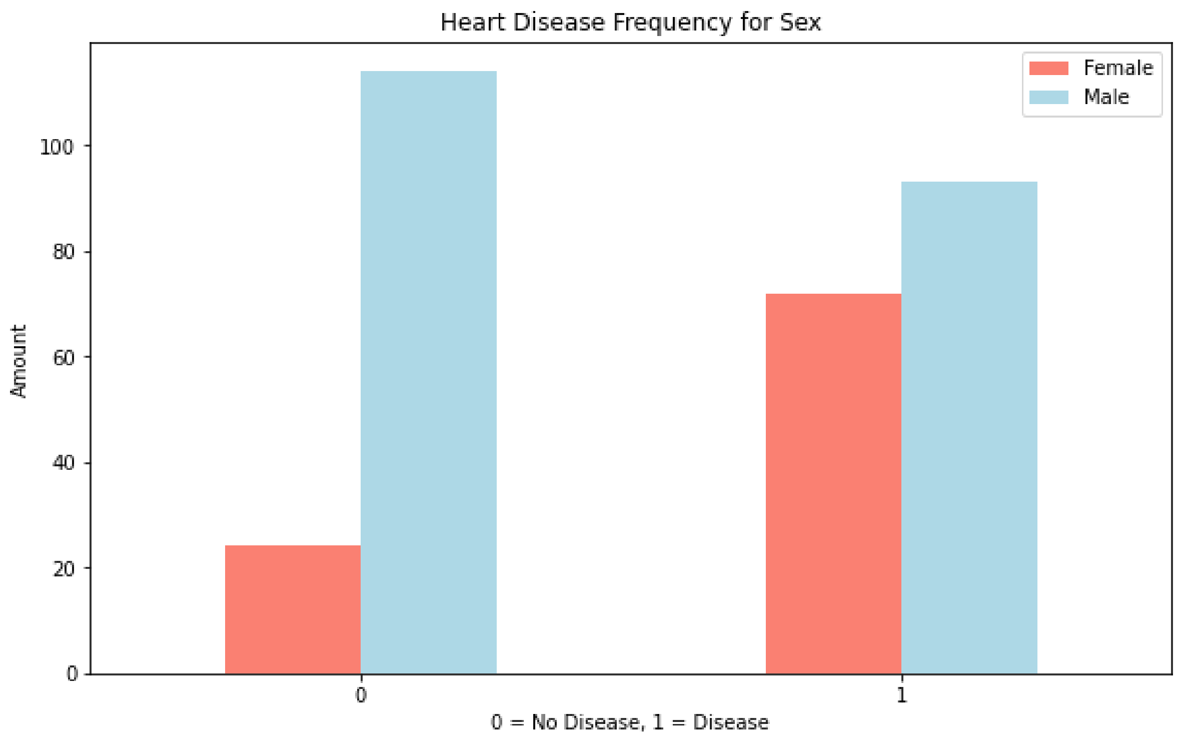
```
# Comapare target column with sex column
pd.crosstab(df.target, df.sex)
```

```
Out[12]:
```

sex	0	1
target		
0	24	114
1	72	93

```
In [13]:
```

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



In [14]:

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

Out[14]:

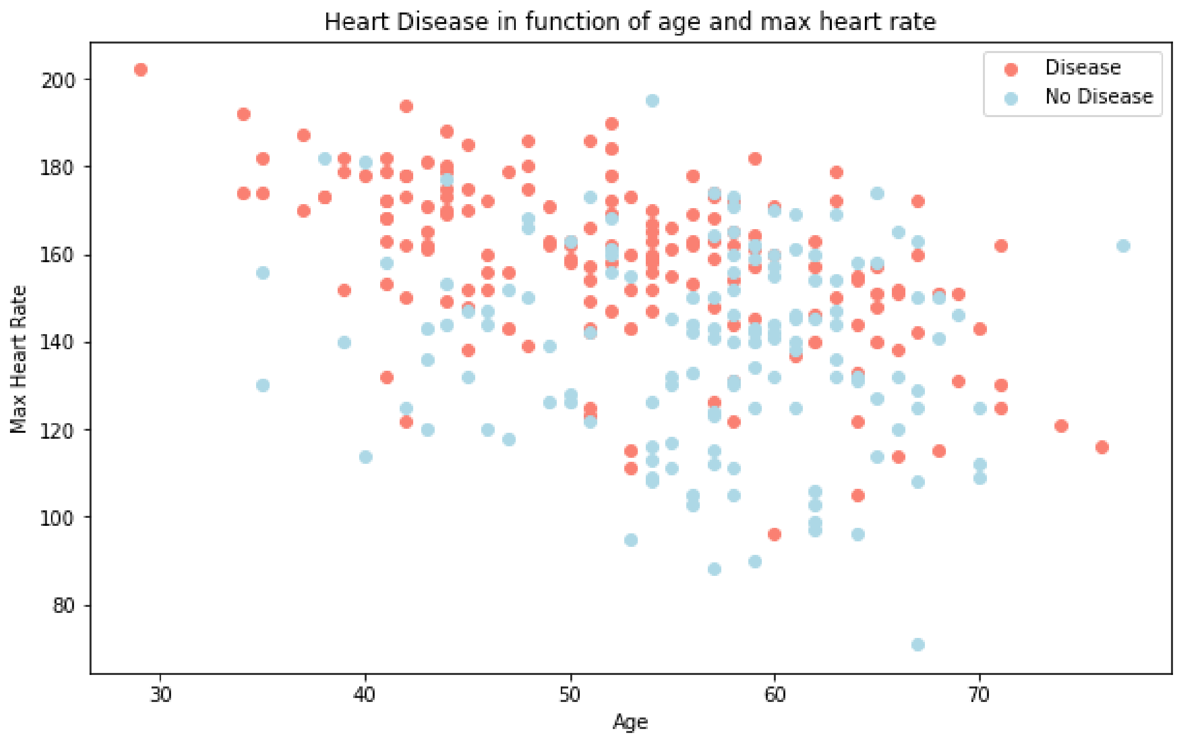
```
162    11
160     9
163     9
152     8
173     8
       ..
202     1
184     1
121     1
192     1
90      1
Name: thalach, Length: 91, dtype: int64
```

Type *Markdown* and LaTeX: α^2

Age vs Max heart rate for heart disease

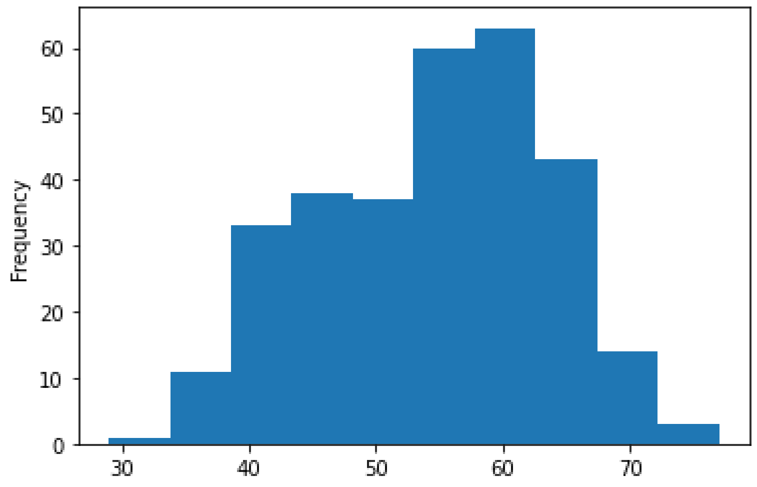
In [15]:

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

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

# Scanner with negative examples
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"]);
```



In [16]:

```
# Check the distribution of age with histogram

df.age.plot.hist();
```



Heart Disease freq. per chest pain type

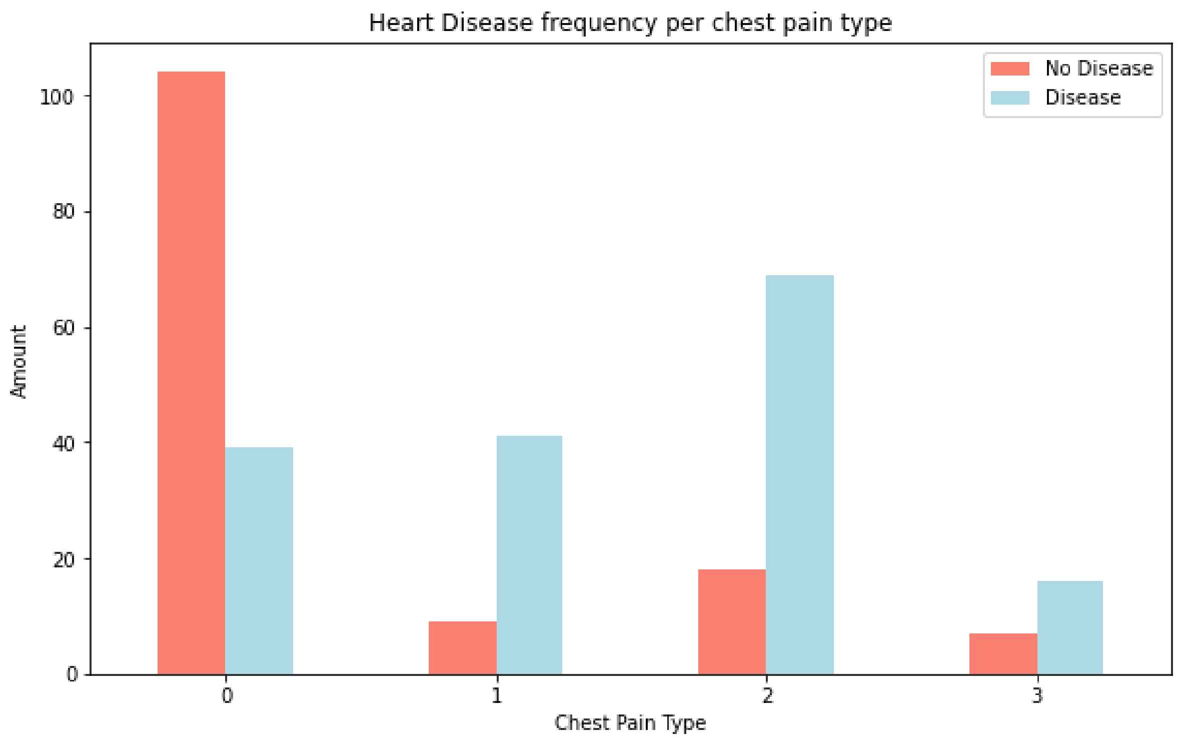
- cp. Type of chest pain. (1 = typical angina, 2 = atypical angina, 3 = non — anginal pain, 4 = asymptotic).

In [17]:

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

In [18]:

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

plt.title("Heart Disease frequency per chest pain type")
plt.xlabel("Chest Pain Type")
plt.ylabel("Amount")
plt.legend(["No Disease", "Disease"])
plt.xticks(rotation=0);
```



In [19]:

```
df.head()
```

Out[19]:

	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 [20]:

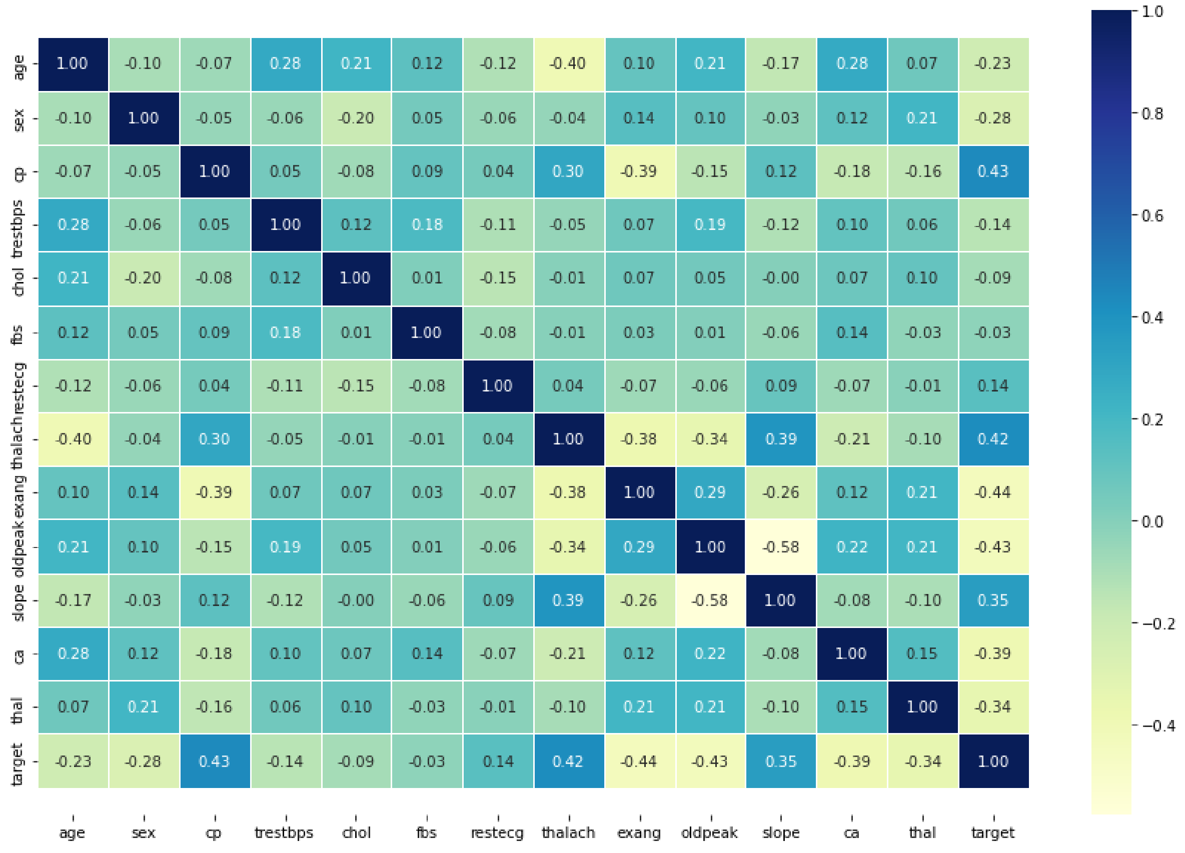
```
# Make a correlation matrix
df.corr()
```

Out[20]:

	age	sex	cp	trestbps	chol	fbs	restecg	thala
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.3985
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.0440
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.2957
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.0466
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.0099
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.0085
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.0441
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.0000
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.3788
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.3441
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.3867
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.2131
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.0964
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.4217

In [21]:

```
# Lets 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");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5);
```



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
age	1.00	-0.10	-0.07	0.28	0.21	0.12	-0.12	-0.40	0.10	0.21	-0.17	0.28	0.07	-0.23
sex	-0.10	1.00	-0.05	-0.06	-0.20	0.05	-0.06	-0.04	0.14	0.10	-0.03	0.12	0.21	-0.28
cp	-0.07	-0.05	1.00	0.05	-0.08	0.09	0.04	0.30	-0.39	-0.15	0.12	-0.18	-0.16	0.43
trestbps	0.28	-0.06	0.05	1.00	0.12	0.18	-0.11	-0.05	0.07	0.19	-0.12	0.10	0.06	-0.14
chol	0.21	-0.20	-0.08	0.12	1.00	0.01	-0.15	-0.01	0.07	0.05	-0.00	0.07	0.10	-0.09
fbs	0.12	0.05	0.09	0.18	0.01	1.00	-0.08	-0.01	0.03	0.01	-0.06	0.14	-0.03	-0.03
restecg	-0.12	-0.06	0.04	-0.11	-0.15	-0.08	1.00	0.04	-0.07	-0.06	0.09	-0.07	-0.01	0.14
thalach	-0.40	-0.04	0.30	-0.05	-0.01	-0.01	0.04	1.00	-0.38	-0.34	0.39	-0.21	-0.10	0.42
exang	0.10	0.14	-0.39	0.07	0.07	0.03	-0.07	-0.38	1.00	0.29	-0.26	0.12	0.21	-0.44
oldpeak	0.21	0.10	-0.15	0.19	0.05	0.01	-0.06	-0.34	0.29	1.00	-0.58	0.22	0.21	-0.43
slope	-0.17	-0.03	0.12	-0.12	-0.00	-0.06	0.09	0.39	-0.26	-0.58	1.00	-0.08	-0.10	0.35
ca	0.28	0.12	-0.18	0.10	0.07	0.14	-0.07	-0.21	0.12	0.22	-0.08	1.00	0.15	-0.39
thal	0.07	0.21	-0.16	0.06	0.10	-0.03	-0.01	-0.10	0.21	0.21	-0.10	0.15	1.00	-0.34
target	-0.23	-0.28	0.43	-0.14	-0.09	-0.03	0.14	0.42	-0.44	-0.43	0.35	-0.39	-0.34	1.00

5. Modeling

In [22]:

```
df.head()
```

Out[22]:

	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 [23]:

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

y = df["target"]
```

In [24]:

```
x, y
```

Out[24]:

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

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

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

In [25]:

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

# Split into train and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

In [26]:

```
x_train
```

Out[26]:

	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 [27]:

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

Out[27]:

```
(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,
 242)
```

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

We will training it to find the patterns on the training set, and test it using the pattern found on the test set

We are going to try three different machine learning models:

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

In [28]:

```
# 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 differetn 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 scores
    model_scores = {}
    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(x_train, y_train)
        # Evaluate the model and append its score to model_scores
        model_scores[name] = model.score(x_test, y_test)
    return model_scores
```

In [29]:

```
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
```

```
E:\ml_project\heart-disease-project\env\lib\site-packages\sklearn\linear_mod
el\_logistic.py:814: 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 (https://scik
it-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-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

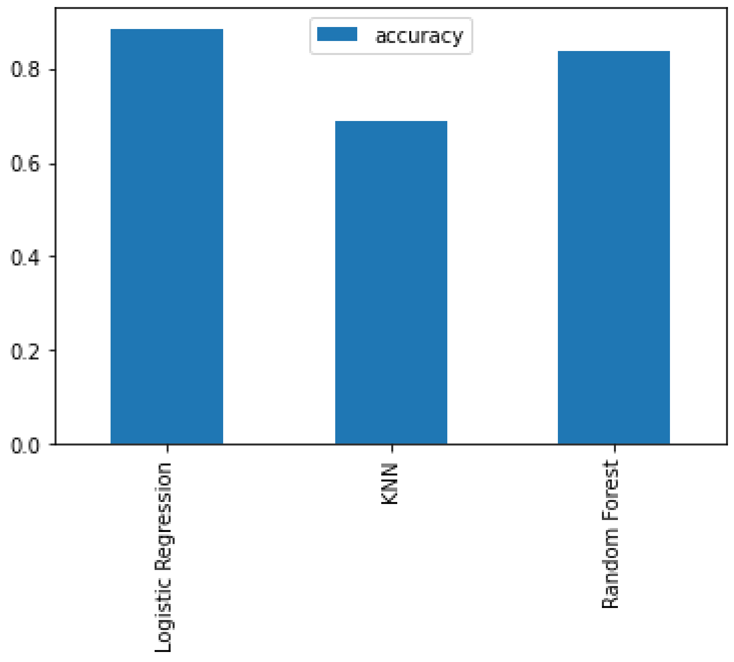
Out[29]:

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

Model Comparison

In [30]:

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



Now we've got a base line model... and we knowa model's first prediction 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
- ROC Curve
- Area under the curve (AUC)

Hyperparameter tuning(by hand)

In [31]:

```
# Let's train KNN

train_scores = []
test_scores = []

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

knn = KNeighborsClassifier()

# Loop through different n_neightbors
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 [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]
```

In [33]:

```
test_scores
```

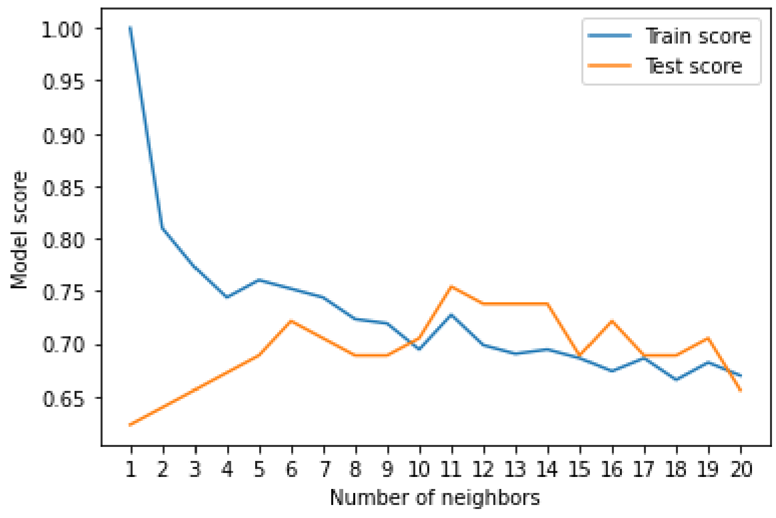
Out[33]:

```
[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]
```

In [34]:

```
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}%")
```

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



Hyperparameter tuning with RandomizedSearchCV

We're going to tune:

- LogisticRegression()
- RandomForestClassifier()

.. using RandomizedSearchCV

In [35]:

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

# Create 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 grids for each of our models, let's tune them using RandomizedSearchCV...

In [36]:

```
# 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[36]:

```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_iter=20,
                   param_distributions={'C': array([1.00000000e-04, 2.636650
90e-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+04]),
                                        'solver': ['liblinear']},
                   verbose=True)
```

In [37]:

```
rs_log_reg.best_params_
```

Out[37]:

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

In [38]:

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

Out[38]:

```
0.8852459016393442
```

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

In [39]:

```
# Setup random seed
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 model for RandomForestClassifier()
rs_rf.fit(x_train, y_train)
```

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

Out[39]:

```
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,  60, 11
0, 160, 210, 260, 310, 360, 410, 460, 510, 560, 610,
       660, 710, 760, 810, 860, 910, 960])},
                   verbose=True)
```

In [40]:

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

Out[40]:

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

In [41]:

```
# evaluate the Randomized search RandomForestClassifier model
rs_rf.score(x_test, y_test)
```

Out[41]:

```
0.8688524590163934
```

Hyperparameter Tuning with GridSearchCV

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

In [42]:

```
# Different hyperparameters 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
```

In [43]:

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

Out[43]:

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

In [44]:

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

Out[44]:

```
0.8852459016393442
```

Evaluating our tuned machine learning classifier, beyond accuracy

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

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

TO make comparisions and evaluate our trained model, we need first to make predictions

In [45]:

```
# Make predictions with tuned model

y_preds = gs_log_reg.predict(x_test)
```

In [46]:

```
y_preds
```

Out[46]:

```
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 [47]:

```
y_test
```

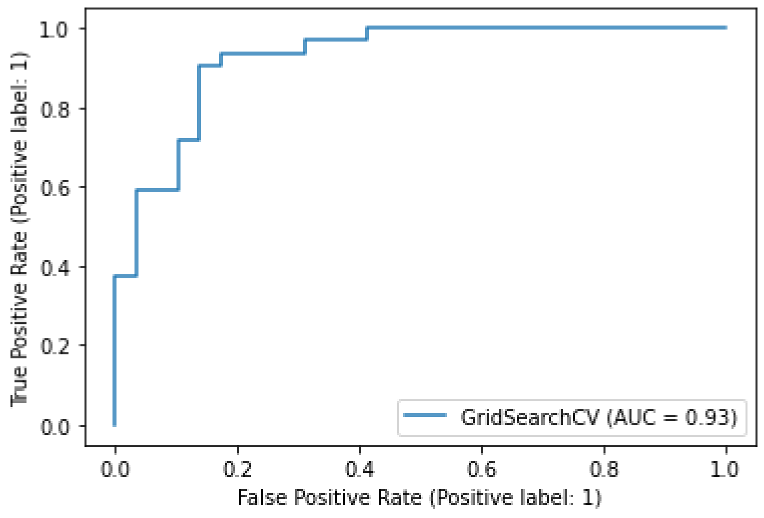
Out[47]:

```
179    0
228    0
111    1
246    0
60     1
      ..
249    0
104    1
300    0
193    0
184    0
Name: target, Length: 61, dtype: int64
```

In [48]:

```
# Plot ROC curve and calculate AUC metrics
plot_roc_curve(gs_log_reg, x_test, y_test);
```

```
E:\ml_project\heart-disease-project\env\lib\site-packages\sklearn\utils\depr
ecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Functio
n :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Us
e one of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predi
ctions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



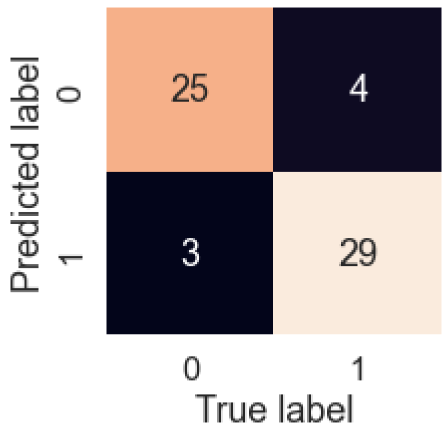
In [49]:

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

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

In [50]:

```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, y_preds):

    """
    Plots a 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")

#     bottom, top = ax.get_ylim()
#     ax.set_ylim(bottom+0.5, top-0.5)

plot_conf_mat(y_test, y_preds);
```



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

In [51]:

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

In [52]:

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

Out[52]:

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

In [53]:

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

In [54]:

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

Out[54]:

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

In [55]:

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

Out[55]:

```
0.8446994535519124
```

In [56]:

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

Out[56]:

```
0.8207936507936507
```

In [57]:

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

Out[57]:

```
0.9212121212121213
```

In [58]:

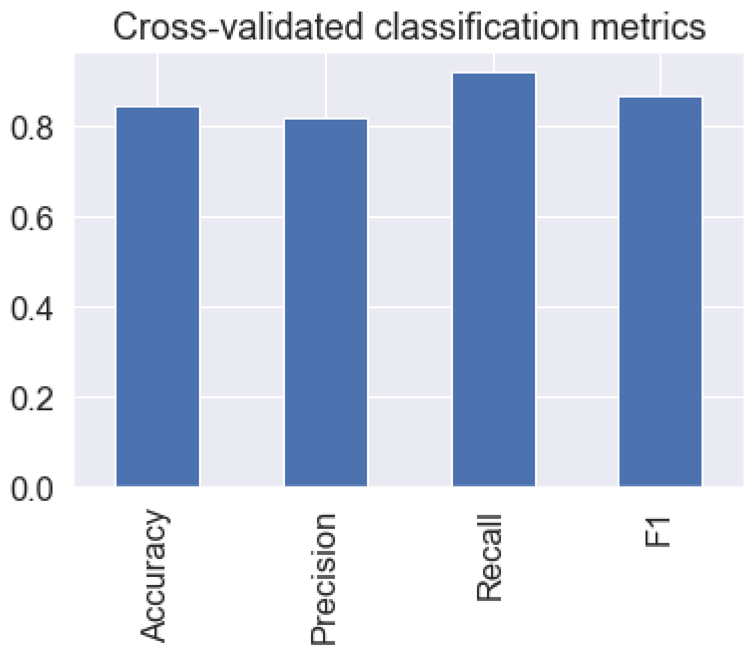
```
# Cross-validated f1-score
cv_f1 = cross_val_score(clf,
                               x,
                               y,
                               cv=5,
                               scoring="f1")
cv_f1 = np.mean(cv_f1)
cv_f1
```

Out[58]:

```
0.8673007976269721
```

In [59]:

```
# Visualise our cross validated 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-validated classification metrics",
                      legend=False);
```



Feature importance

Which features contributed most to the outcome of the model and how did they contribute?

Finding feature importance is different for each ML model. One way to find feature importance is to search for model name feature importance.

Let's find the feature importance for our LogisticRegression model:

In [63]:

```
gs_log_reg.best_params_

# fit an instance of LogisticRegression
clf = LogisticRegression(C = 0.20433597178569418, solver = 'liblinear')

clf.fit(x_train, y_train)
```

Out[63]:

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

In [64]:

```
# Check
clf.coef_
```

Out[64]:

```
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 [65]:

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

In [66]:

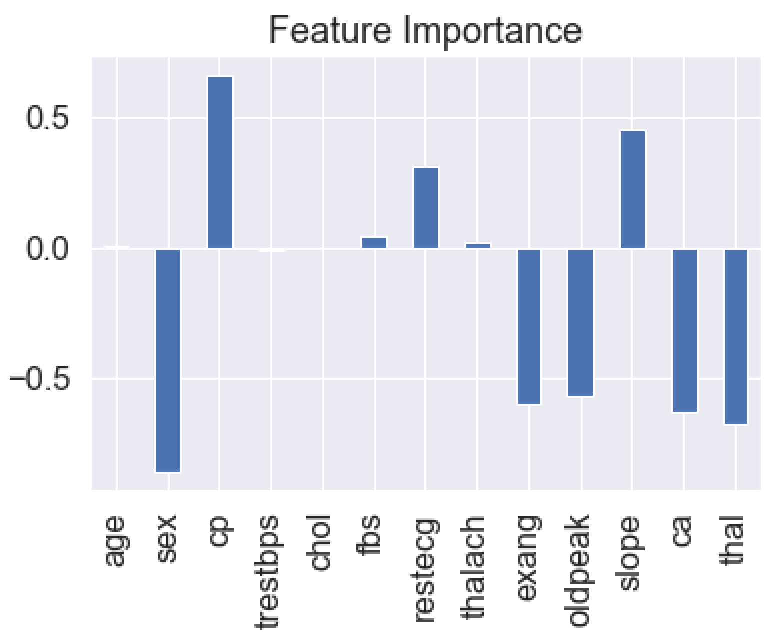
```
feature_dict
```

Out[66]:

```
{'age': 0.0031672801993431563,
 'sex': -0.8604465072345515,
 'cp': 0.6606704082033799,
 'trestbps': -0.01156993168080875,
 '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 [68]:

```
# Visualize feature importance
feature_df = pd.DataFrame(feature_dict, index=[0])
feature_df.T.plot.bar(title='Feature Importance', legend=False);
```



In [69]:

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

Out[69]:

target	**0**	**1**
sex		
0	24	72
1	114	93

In [70]:

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

Out[70]:

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

6. Expermimentation

If you haven't hit your evaluation metric yet then you might:

- Collect more data
- Try a better model like CatBoost or XGBoost
- Improve the current models
- If model is good enough then you may export and share it with others