Predicting heart disease using Machine learning

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

We're 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. https://archive.icss.uci.edu/ml/datasets/heart+Disease (https://archive.icss.uci.edu/ml/datasets/heart+Disease)

There is also a version of it available on kaggle. https://www.kaggle.com/ronitf/heart-disease-uci (https://www.kaggle.com/ronitf/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. Wecan do this via doing our own research (such as looking at the links above) or by talking to a subject matter expert(someone who knows about the dataset).

Create data dictionary

1. age - age in years
2. sex - (1 = male; 0 = female)
3. cp - chest pain type
    - 0:Typical angina: chest pain related decrease blood supply to the heart
    - 1:Atypical angina: chest pain not related to heart
    - 2:Non-anginal pain: typically esophageal spasms(non heart related)
    - 3:Asymptomatic: chest pain not showing sign of disease
4. trestbps-resting blood pressure (in mm Hg on admission to the hospital) anything above 130-140 is typically cause for concern
5. chol - serum cholestoral in mg/dl
    - serum=LDL + HDL + .2*triglycerides
    - above 200 is cause for concern
6. fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
    - '>126' mg/dl signals diabetes
7. restecg - resting electrocardiographic results
    - 0:Nothing to note
    - 1:ST-T Wave abnormality
        - can range from mild symptoms to severe problems
        - signals non-normal heart beat
    - 2:Possible or definite left ventricular hypertrophy
        - Enlarged heart's main pumping chamber
8. thalach - maximum heart rate achieved
9. exang - exercise induced angina (1 = yes; 0 = no)
10. oldpeak - ST depression induced by exercise relative to rest looks at stress of heart during exercise unhealthy heart will stress more
11. slope - the slope of the peak exercise ST segment
    - 0:Upsloping:better heart rate with exercise (uncommon)
    - 1:Flatsloping:minimal change (typical healthy heart)
    - 2:Downsloping:signs of Unhealthy heart
12. ca - number of major vessels (0-3) colored by flourosopy
    - colored vessel means the doctor can see the blood passing through
    - the more blood movement the better (no clots)
13. thal - thalium stress result
    - 1,3 = normal
    - 6 = fixed defect: used to be defect but ok now
    - 7 = reversable defect: no proper blood movement when exercising
14. target - have disease or not (1=yes, 0=no) (=the predicted attribute)

Preparing the tools

We're going to use pandas, Matplotlib, and NumPy for data analysis and manipulation.

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

# 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 plot 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 plot_roc_curve
```

Load Data

In [5]:
```python
df=pd.read_csv("heart-disease.csv")
df.shape # (rows, columns)
```

Out[5]:
```
(303, 14)
```

Data Exploration (exploratory data analysis or EDA)

The goal here is to find out more about the data and become a subject matter expert on the dataset we are working with.

1. what questions are we 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 we deal with it?
4. where are the outliers and why should we care about them?
5. How can we add, change or remove features to get more out of our data?

In [6]:
```python
df.head()
```

Out[6]:

	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 [7]:
```python
df.tail()
```

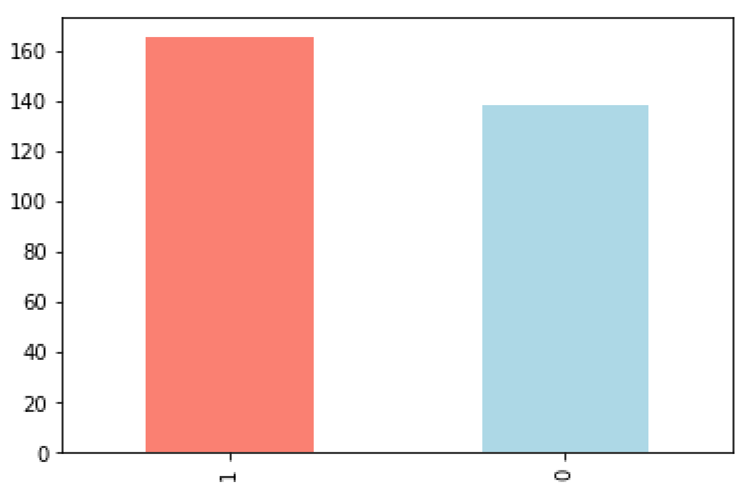
Out[7]:

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

In [9]:
```python
#  Let's find out how many of each class there
df["target"].value_counts()
```

Out[9]:
```
1    165
0    138
Name: target, dtype: int64
```

In [10]:
```python
df["target"].value_counts().plot(kind="bar",  color=["salmon","lightblue"]);
```



In [11]:
```
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 [15]:
```
df.isna().sum()
```

Out[15]:
```
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 [16]:
```
df.describe()
```

Out[16]:

	age	sex	cp	trestbps	chol	fbs	restecg	tha
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.64
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.90
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.00
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.50
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.00
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.00
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.00

Heart disease frequency according to sex

In [18]:
```
df.sex.value_counts()
```

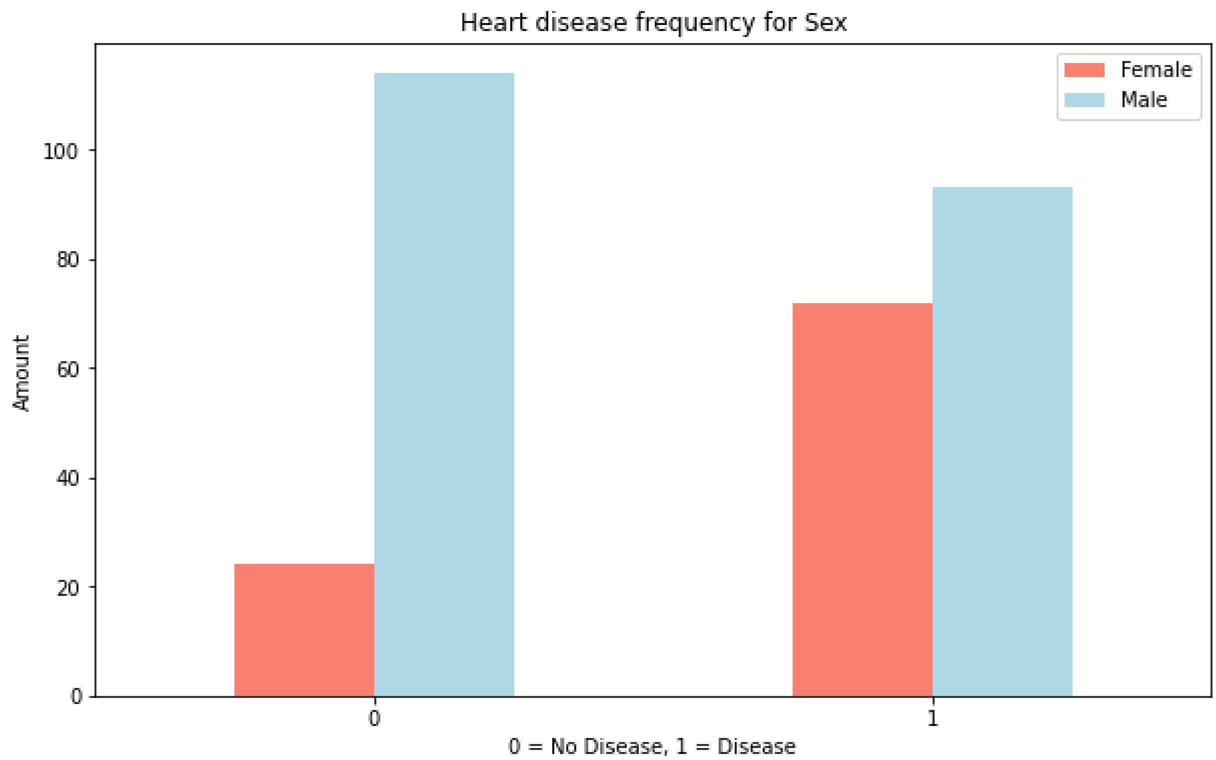
Out[18]:
```
1    207
0     96
Name: sex, dtype: int64
```

In [19]:
```
# Compare target columns with sex columns
pd.crosstab(df.target, df.sex)
```

Out[19]:

sex	0	1
target		
0	24	114
1	72	93

In [26]:
```
#  Create a plot of crosstab
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);
```

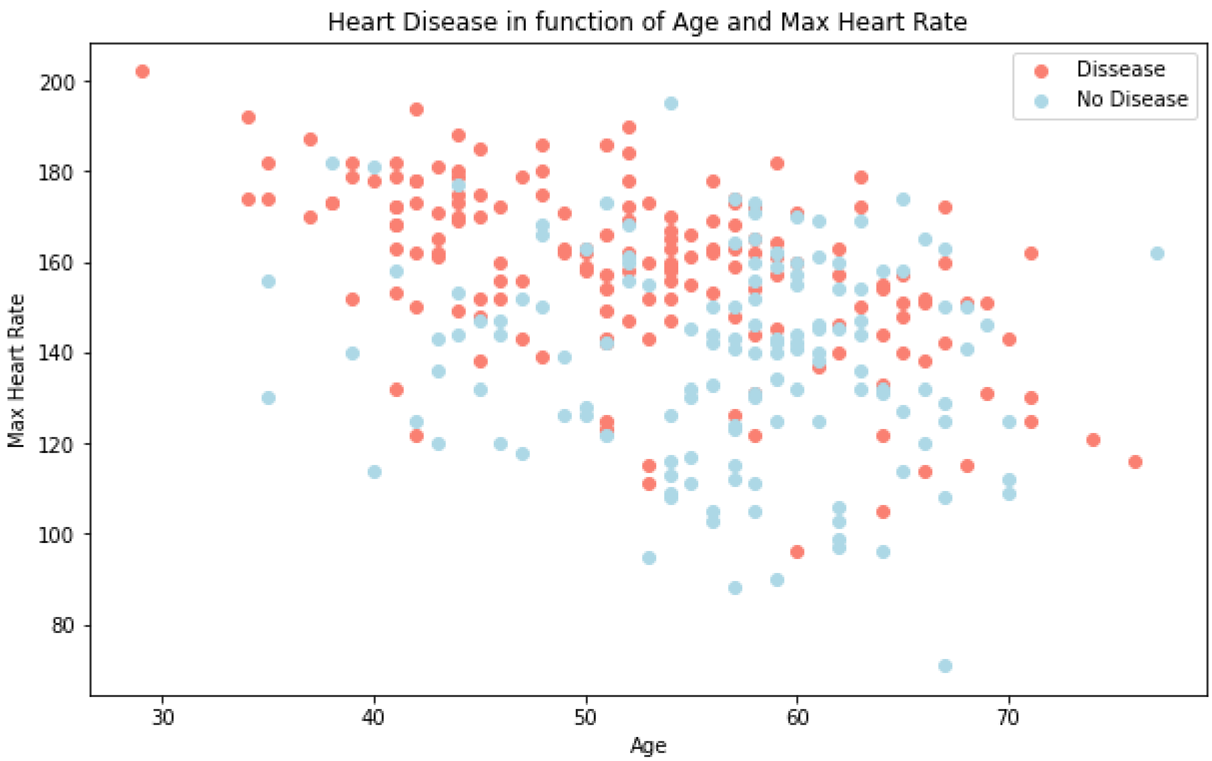


Age vs Max Heart rate for Heart Disease

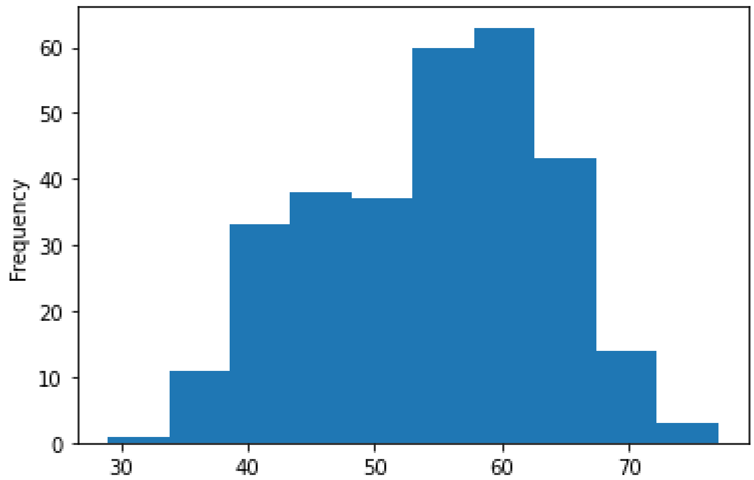
In [31]:
```
df.thalach.value_counts()
```

Out[31]:
```
162    11
163     9
160     9
152     8
173     8
       ..
128     1
129     1
134     1
137     1
202     1
Name: thalach, Length: 91, dtype: int64
```

In [30]:
```python
# Create 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")

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

# Add some helpfull information
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 [33]:
```
# Check the  distribution of the age column with a histogram
df.age.plot.hist();
```



Heart Disease frequency per Chest Pain

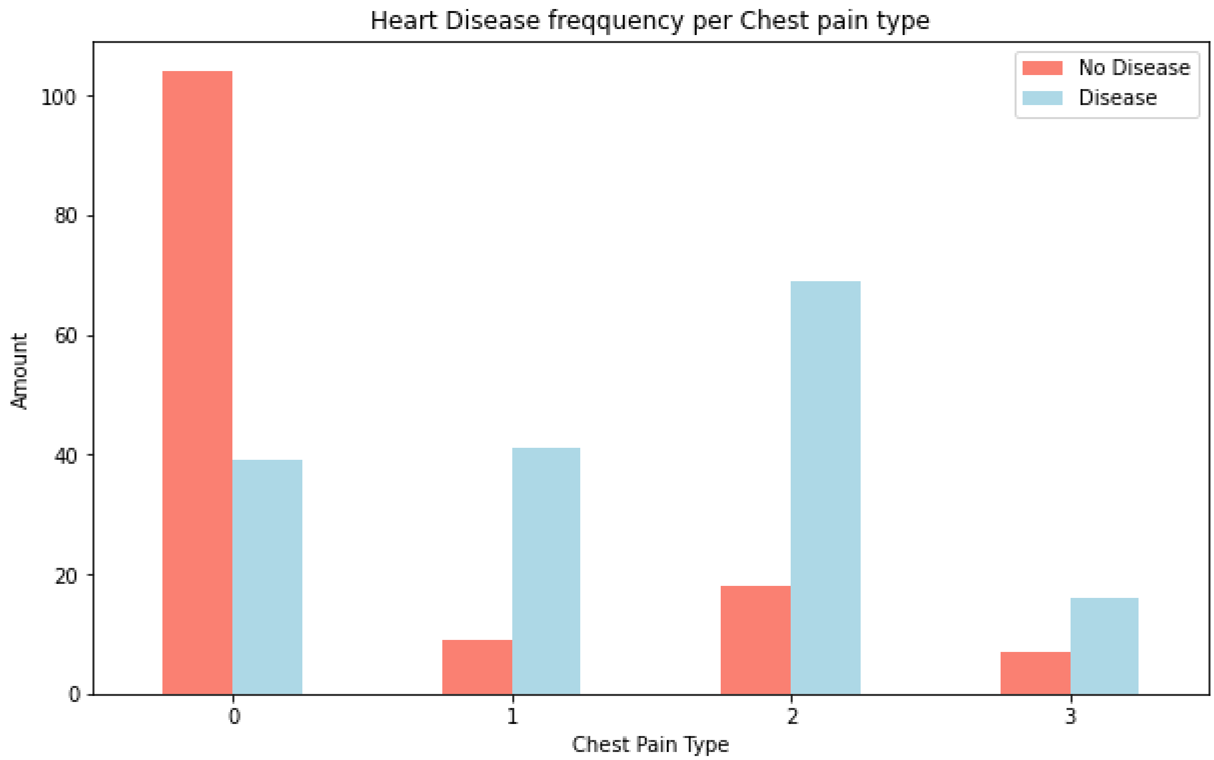
3. cp - chest pain type
    - 0:Typical angina: chest pain related decrease blood supply to the heart
    - 1:Atypical angina: chest pain not related to heart
    - 2:Non-anginal pain: typically esophageal spasms(non heart related)
    - 3:Asymptomatic: chest pain not showing sign of disease

In [35]:
```
pd.crosstab(df.cp, df.target)
```

Out[35]:

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

In [41]:
```
# Make the crosstab more visual
pd.crosstab(df.cp, df.target).plot(kind="bar",
                                   figsize=(10, 6),
                                   color=["salmon", "lightblue"])

# Add more useful information
plt.title("Heart Disease freqquency per Chest pain type")
plt.xlabel("Chest Pain Type")
plt.ylabel("Amount")
plt.legend(["No Disease", "Disease"])
plt.xticks(rotation=0);
```



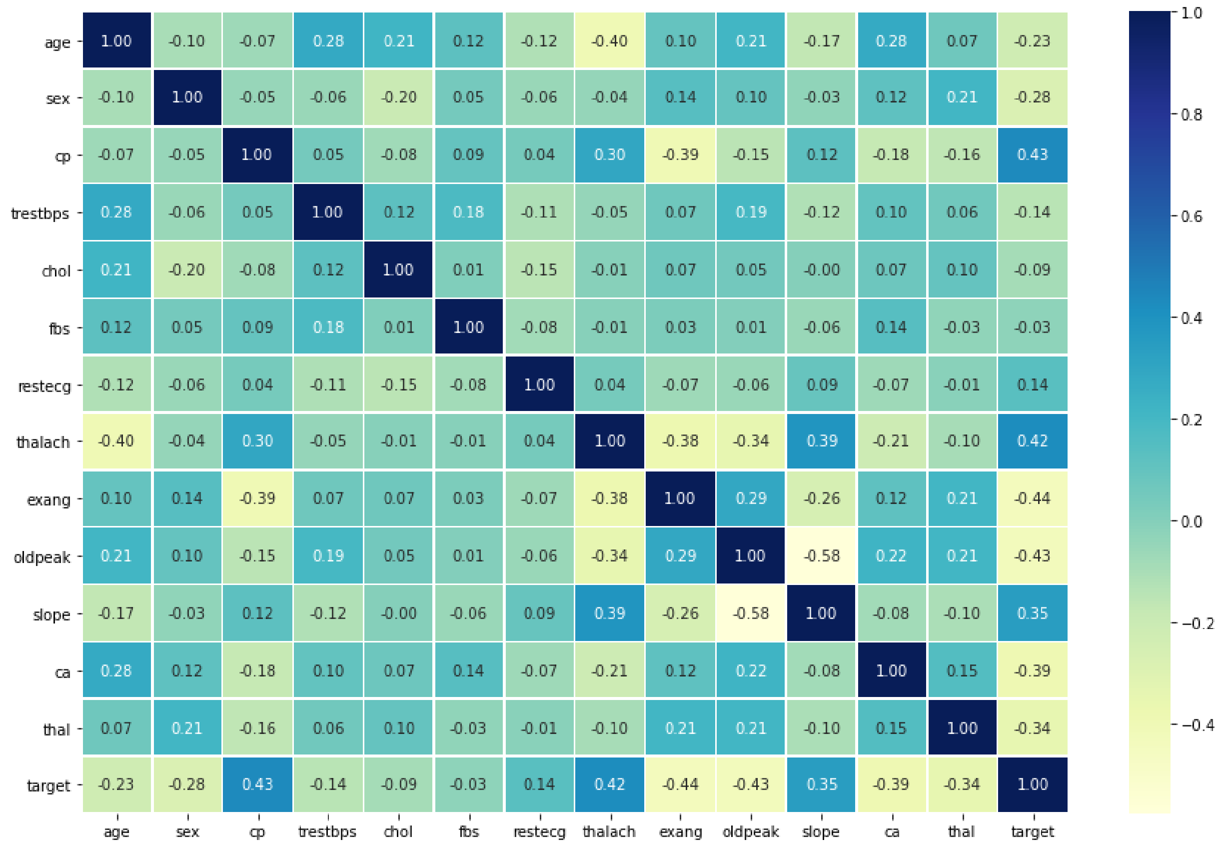
In [43]:

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

Out[43]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784

In [48]:
```python
# 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");
```



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

In one case, strangely enough, according to the correlation value here, if someone gets chest pain during exercise(exang=1), their chance of having heart disease goes down(target=0).

5. Modelling

In [49]:
```
df.head()
```

Out[49]:

	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 [50]:
```
# Split data into X and y
X=df.drop("target",axis=1)
y=df["target"]
```

In [52]:
```
X
```

Out[52]:

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

303 rows × 13 columns

In [53]:
```
y
```

Out[53]:
```
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 [54]:
```
# Split data into train and test sets
np.random.seed(42)

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

In [55]:
```
X_train
```

Out[55]:

	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 [58]:
```
y_train
```

Out[58]:
```
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 training and test sets, it's time to build a machinee learning model.

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

And we'll test it(use the patterns) 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

In [59]:
```
# 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 : Testing data (no labels)
    y_train : Training labels
    y_test :Testing labels
    """
    # Set random seed
    np.random.seed(42)
    # Make a dictionary tto keep model scores
    model_score={}
    # 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 model score
        model_score[name]=model.score(X_test, y_test)
    return  model_score
```

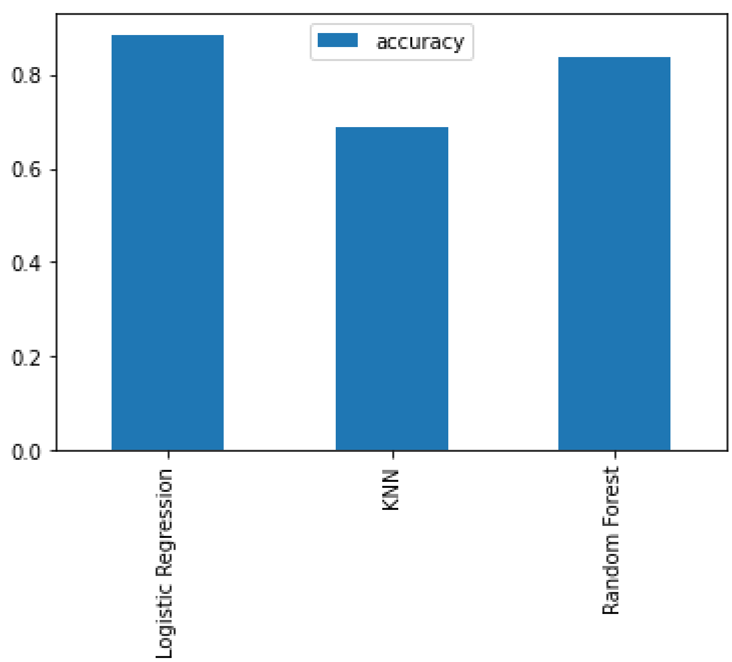
In [60]:
```
model_score=fit_and_score(models=models,
                          X_train=X_train,
                          X_test=X_test,
                          y_train=y_train,
                          y_test=y_test)
model_score
```

```
F:\Data science projects\heart-disease-project\env\lib\site-packages\sklearn\li
near_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (stat
us=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://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-regressi
on (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on)
  n_iter_i = _check_optimize_result(
```

Out[60]:
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

Comapre the models

In [63]:
```
model_compare=pd.DataFrame(model_score, index=["accuracy"])
model_compare.T.plot.bar();
```



Now we've got a baseline model... and we know a model's first prediction aren't always what we should based our next steps off. what should do?

Let's took at the following:

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

Hyperparameter tuning by hand

In [65]:
```
# Let's tune KNN

train_score=[]
test_score=[]

# 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 algorithm
    knn.fit(X_train, y_train)

    # Update the training score list
    train_score.append(knn.score(X_train, y_train))

    # pdate the test score list
    test_score.append(knn.score(X_test, y_test))
```

In [66]:
```
train_score
```

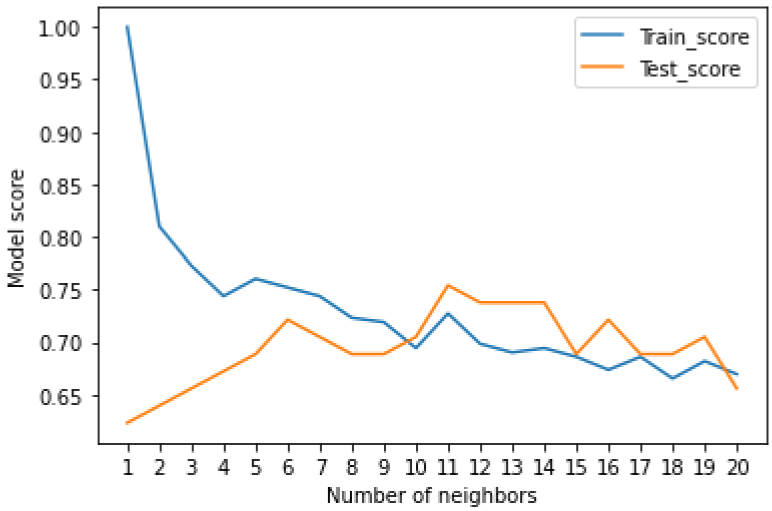
Out[66]:
```
[1.0,
 0.8099173553719008,
 0.7727272727272727,
 0.743801652892562,
 0.7603305785123967,
 0.7520661157024794,
 0.743801652892562,
 0.7231404958677686,
 0.71900826446281,
 0.6942148760330579,
 0.7272727272727273,
 0.6983471074380165,
 0.6900826446280992,
 0.6942148760330579,
 0.6859504132231405,
 0.6735537190082644,
 0.6859504132231405,
 0.6652892561983471,
 0.6818181818181818,
 0.6694214876033058]
```

In [67]:
```
test_score
```

Out[67]:
```
[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 [69]:
```
plt.plot(neighbors, train_score, label="Train_score")
plt.plot(neighbors, test_score, 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_score)*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 [75]:
```python
# Create a hyperparameter grid for LogissticRegression
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 grids setup for each of our models, let's tune them using RandomizedSearchCV...

In [71]:
```python
# Tune Logistic Regression

np.random.seed(42)

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

# Fit Random hyperparameter search model for Logistic Regression
rs_log_reg.fit(X_train, y_train)
```

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

Out[71]:
```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_iter=20,
                   param_distributions={'C': array([1.00000000e-04, 2.63665090e
-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 [72]:
```python
rs_log_reg.best_params_
```

Out[72]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

In [73]:
```python
rs_log_reg.score(X_test, y_test)
```

Out[73]:
```
0.8852459016393442
```

Now we've tune the logistic Regression, let's do the same for RandomForestClassifier()...

In [76]:
```
# Tune Logistic Regression

# Setp random seed
np.random.seed(42)

# Setup random hyperparameter search for Random Forest Classifier
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                                param_distributions=rf_grid,
                                cv=5,
                                n_iter=20,
                                verbose=True)

# Fit Random hyperparameter search model for Random Forest Classifier
rs_rf.fit(X_train, y_train)
```

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

Out[76]:
```
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, 110, 1
60, 210, 260, 310, 360, 410, 460, 510, 560, 610,
       660, 710, 760, 810, 860, 910, 960])},
                   verbose=True)
```

In [77]:
```
#Finding the best hyperparameter
rs_rf.best_params_
```

Out[77]:
```
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

In [78]:
```
# Evaluate the randomized search RandomForestClassifier model
rs_rf.score(X_test, y_test)
```

Out[78]:
```
0.8688524590163934
```

Hyperparameter tuning with GridSearchCV

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

In [82]:
```
# Different hyperparameters for our Logistic regression model
log_reg_grid={"C":np.logspace(-4, 4, 30),
              "solver":["liblinear"]}

# Setup grid hyperparameter search for Logistic Regression
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[82]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.00000000e-04, 1.88739182e-04, 3.56224789
e-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)
```

In [83]:
```
# Check the best hyper parameters
gs_log_reg.best_params_
```

Out[83]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [84]:
```
# Evaluate the grid search Logistic Regression model
gs_log_reg.score(X_test, y_test)
```

Out[84]:
```
0.8852459016393442
```

Evaluating 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.

In [85]:
```
# Make predictions with tuned model
y_preds=gs_log_reg.predict(X_test)
```

In [100]:
```
y_preds
```

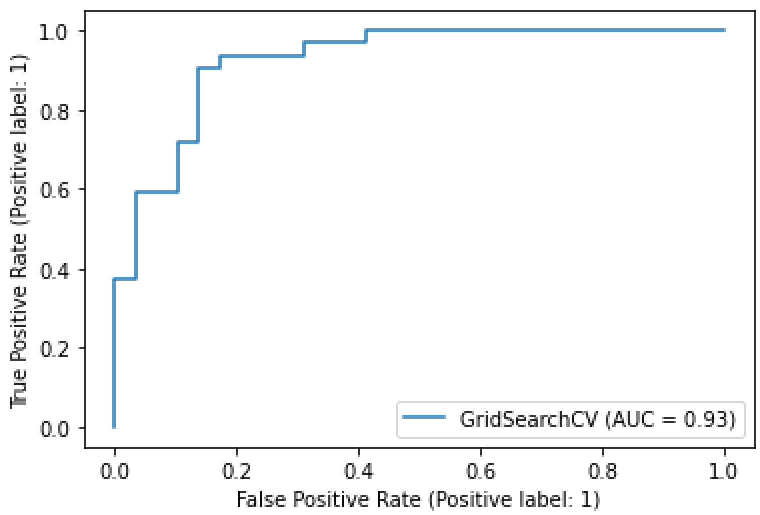
Out[100]:
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [98]:
```
y_test.value_counts()
```

Out[98]:
```
1    32
0    29
Name: target, dtype: int64
```

In [88]:
```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(gs_log_reg, X_test, y_test)
```

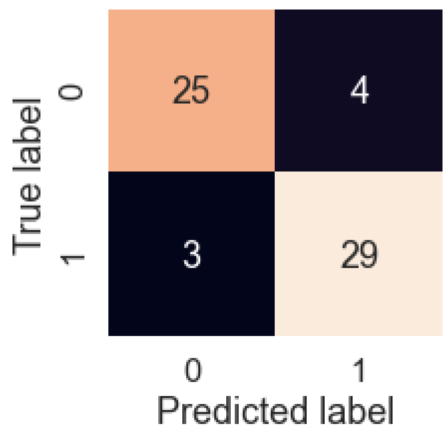
Out[88]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x19c83471160>
```



In [89]:
```
# Confusion matrix
print(confusion_matrix(y_test, y_preds))
```

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

In [101]:
```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, y_preds):
    """
    Plots 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_mat(y_test, y_preds)
```



Now we've got 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 [102]:
```
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 sing 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().

In [104]:
```
# Check best hyperparameters
gs_log_reg.best_params_
```

Out[104]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [105]:
```
# Create a new classifier with best parameters
clf = LogisticRegression(C=0.20433597178569418,
                         solver="liblinear")
```

In [106]:
```
# Cross-validated accuracy
cv_acc = cross_val_score(clf,
                         X,
                         y,
                         cv=5,
                         scoring="accuracy")
cv_acc=np.mean(cv_acc)
cv_acc
```

Out[106]:
```
0.8446994535519124
```

In [107]:
```
# Cross-validated precision
cv_precision = cross_val_score(clf,
                         X,
                         y,
                         cv=5,
                         scoring="precision")
cv_precision=np.mean(cv_precision)
cv_precision
```

Out[107]:
```
0.8207936507936507
```

In [108]:
```
# Cross-validated recall
cv_recall = cross_val_score(clf,
                         X,
                         y,
                         cv=5,
                         scoring="recall")
cv_recall=np.mean(cv_recall)
cv_recall
```

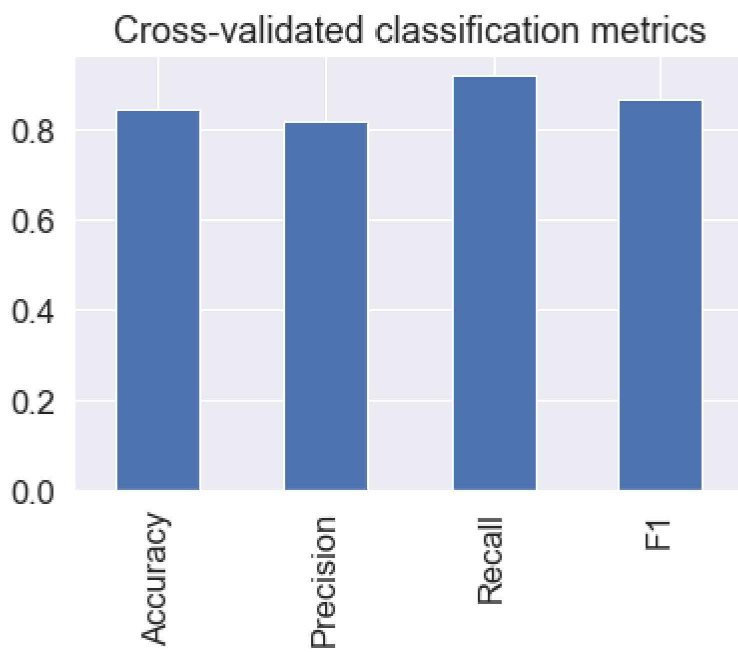
Out[108]:
```
0.9212121212121213
```

In [109]:
```
# 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[109]:
```
0.8673007976269721
```

In [113]:
```
# Visualize 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)
```

Out[113]: <AxesSubplot:title={'center':'Cross-validated classification metrics'}>



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 importance is to search for "(MODEL NAME) feature importance

Let's Find the feature importance for our LogisticRegression model...

In [115]:
```
# Fit an instance of Logistic Regression
clf=LogisticRegression(C=0.20433597178569418,
                       solver="liblinear")
clf.fit(X_train, y_train)
```

Out[115]: LogisticRegression(C=0.20433597178569418, solver='liblinear')

In [116]:
```
# check coef_
clf.coef_
```

Out[116]:
```
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 [117]:
```
df.head()
```

Out[117]:

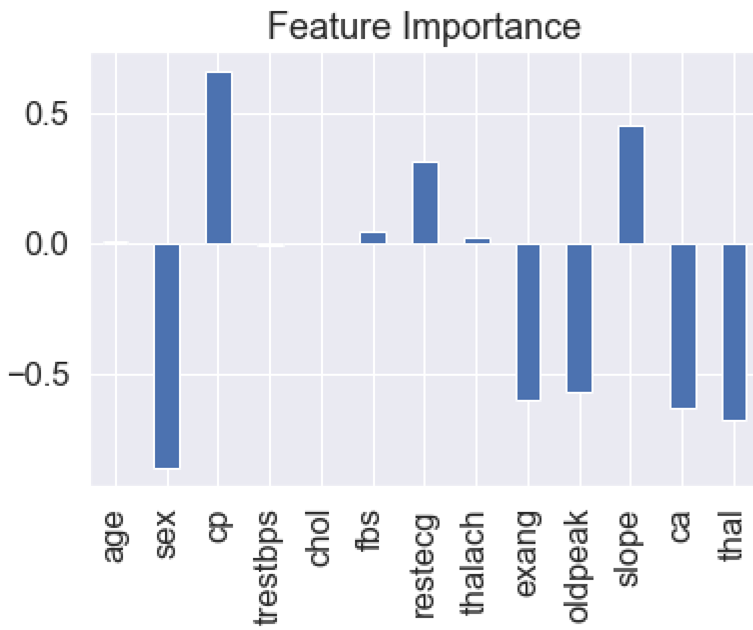
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [118]:
```
# Match coef's of features to columns
feature_dict = dict(zip(df.columns, list(clf.coef_[0])))
feature_dict
```

Out[118]:
```
{'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 [119]:
```
# Visualize feature importance
featuure_df = pd.DataFrame(feature_dict, index=[0])
featuure_df.T.plot.bar(title="Feature Importance", legend=False)
```

Out[119]: <AxesSubplot:title={'center':'Feature Importance'}>



In [122]:
```
pd.crosstab(df["sex"], df["target"])
```

Out[122]:

target	0	1
sex		
0	24	72
1	114	93

In [121]:
```
pd.crosstab(df["slope"], df["target"])
```

Out[121]:

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

slope - the slope of the peak exercise ST segment

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

Experimentation

If you haven't hit our evaluation metric yet.....ask ourselves...

- Could we collect more data?
- Could we try a better model? Like CatBoost or XGBoost
- Could we improve the current models?(beyond what we've done so far)
- If our model is good enough(we have hit our evaluation metric) how would we export it and share it with others?