heart-disease-classification

June 14, 2021

1 Predicting heart disease using machine learning

This notebook uses various Python based machine learning and data science libraries in an attempt to predict whether or not someone has heart disease based on their medical attributes.

We are going to take the following approach: 1. Problem Definition 2. Data 3. Evaluation 4. Features 5. Modelling 6. Experimentation

1.1 1. Problem Definition

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

1.2 2. Data

The original data came from the Cleaveland data from UCI machine learning repository.

There is also another version of it available on Kaggle.

1.3 3. Evaluation

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

1.4 4. Features

This is where we will get different infromation about each of the features of our data.

- age - age in years
- sex - (1 = male; 0 = female)
- 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 signs of disease
- trestbps - resting blood pressure (in mm Hg on admission to the hospital) anything above 130-140 is typically cause for concern
- chol - serum cholestoral in mg/dl serum = LDL + HDL + .2 * triglycerides above 200 is cause for concern
- fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false) '>126' mg/dL signals diabetes
- 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

- thalach - maximum heart rate achieved
- exang - exercise induced angina (1 = yes; 0 = no)
- oldpeak - ST depression induced by exercise relative to rest looks at stress of heart during excercise unhealthy heart will stress more
- slope - the slope of the peak exercise ST segment 0: Upsloping: better heart rate with excercise (uncommon) 1: Flatsloping: minimal change (typical healthy heart) 2: Downslopins: signs of unhealthy heart
- 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)
- 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 excercising
- target - have disease or not (1=yes, 0=no) (= the predicted attribute)

1.4.1 Preparing the tools

We are going to use: 1. Pands 2. Matplotlib 3. Pandas for data analysis and manipulation

[1]:
```
# Import all tools 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

# to plot all graphs within the jupyter 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
```

1.4.2 Load Data

[2]:
```
df = pd.read_csv("heart-disease.csv")
df
```

[2]:
```
     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  target
0        0   0     1       1
1        0   0     2       1
2        2   0     2       1
3        2   0     2       1
4        2   0     2       1
..     …  ..   …     …
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0

[303 rows x 14 columns]
```

1.4.3 Data exploration (Explaratory Data Analysis or EDA)

The goal here is to become familiar with our dataset and become a subject matter expert of the dataset that we are working on.

1. What question(s) 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 your data?

[3]:
```
df.head()
```

[3]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
```

```
2   41    0    1         130    204    0          0        172        0        1.4        2
3   56    1    1         120    236    0          1        178        0        0.8        2
4   57    0    0         120    354    0          1        163        1        0.6        2

   ca   thal   target
0   0      1        1
1   0      2        1
2   0      2        1
3   0      2        1
4   0      2        1
```

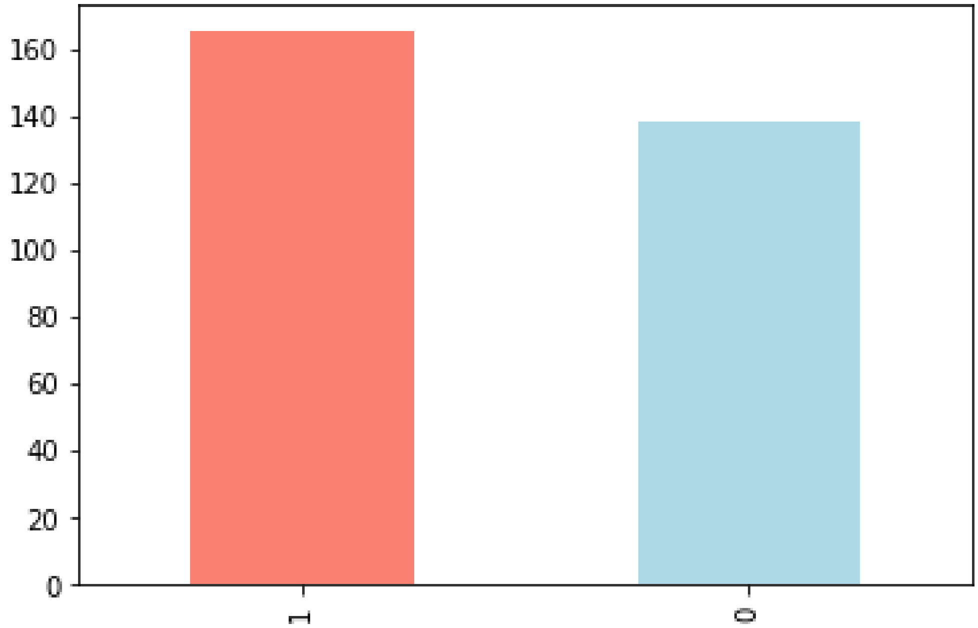
[4]:
```
df.tail()
```

[4]:
```
     age   sex   cp   trestbps   chol   fbs   restecg   thalach   exang   oldpeak  \
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   target
298      1    0      3        0
299      1    0      3        0
300      1    2      3        0
301      1    1      3        0
302      1    1      2        0
```

[5]:
```
# Let's see how many of each class do we have
df["target"].value_counts()
```

[5]:
```
1    165
0    138
Name: target, dtype: int64
```

[6]:
```
df["target"].value_counts().plot(kind="bar", color=["salmon", "lightblue"]);
```



[7]:

```
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
```

[8]:

```
# Checking for missing values
df.isna().sum()
```

[8]:
```
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
```

[9]:
```python
df.describe()
```

[9]:

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

	restecg	thalach	exang	oldpeak	slope	ca
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373
std	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606
min	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000
50%	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000
75%	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

	thal	target
count	303.000000	303.000000
mean	2.313531	0.544554
std	0.612277	0.498835
min	0.000000	0.000000
25%	2.000000	0.000000
50%	2.000000	1.000000
75%	3.000000	1.000000
max	3.000000	1.000000

1.4.4 Heart disease frequency according to sex

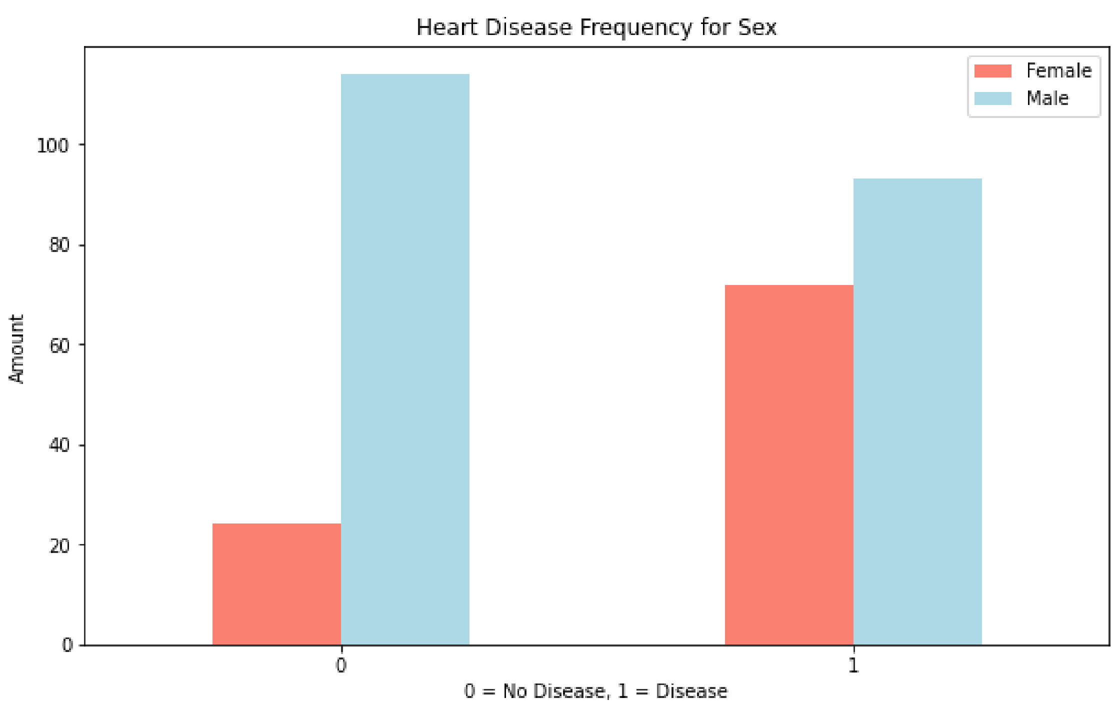
[10]:
```
df["sex"].value_counts()
```

[10]:
```
1    207
0     96
Name: sex, dtype: int64
```

[11]:
```
# Compare target column with sex colum
pd.crosstab(df["target"], df["sex"])
```

[11]:
```
sex      0    1
target
0       24  114
1       72   93
```

[12]:
```
# Plotting the 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);
```

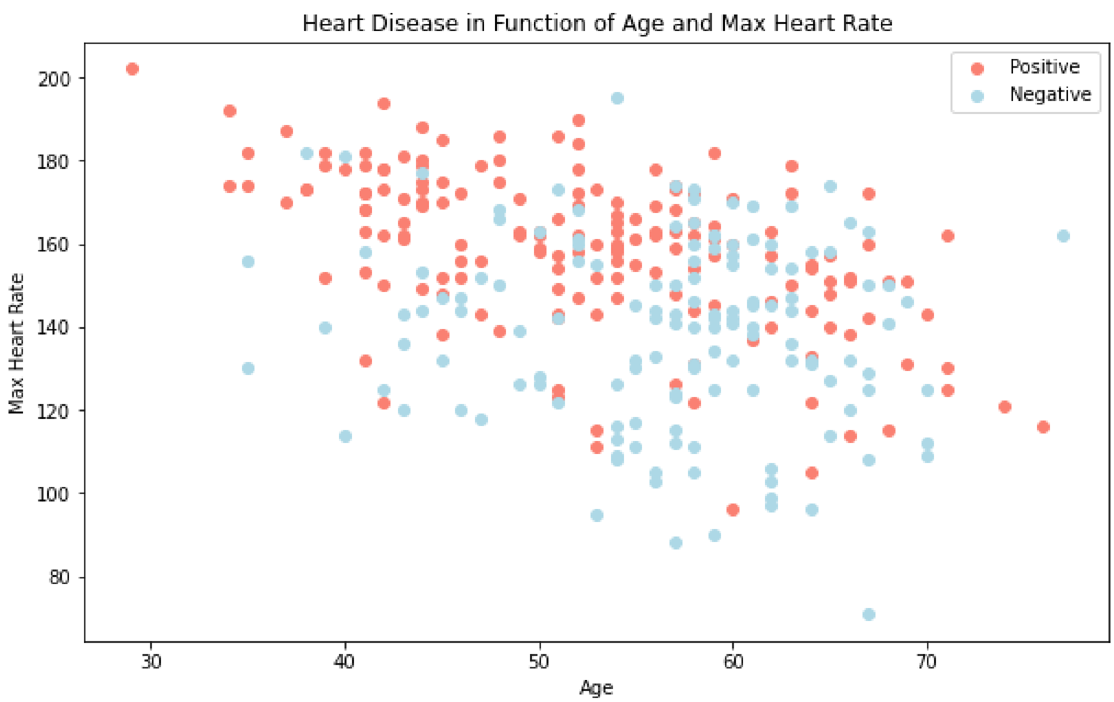


[13]:
```
df["thalach"].value_counts()
```

[13]:
```
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
```

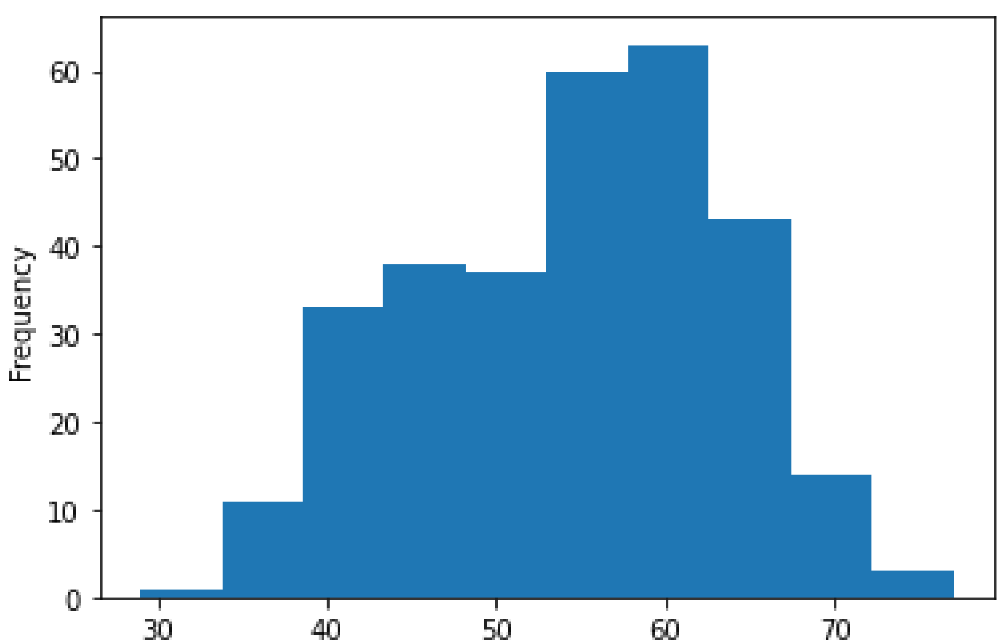
1.4.5 Age Vs. Max Heart Rate for Heart Disease

[19]:
```
# Create a figure
plt.figure(figsize=(10,6))

# PLot with positive examples
plt.scatter(df["age"][df["target"] ==1], df["thalach"][df["target"]==1],␣
 ↪color="salmon")

# Plot with negative examples
plt.scatter(df["age"][df["target"]==0], df["thalach"][df["target"]==0],␣
 ↪color="lightblue")

# Adding info to the plot

plt.title("Heart Disease in Function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Positive", "Negative"]);
```



[22]:
```
# Checking the distribution of age column
df["age"].plot.hist()
```

[22]: <AxesSubplot:ylabel='Frequency'>

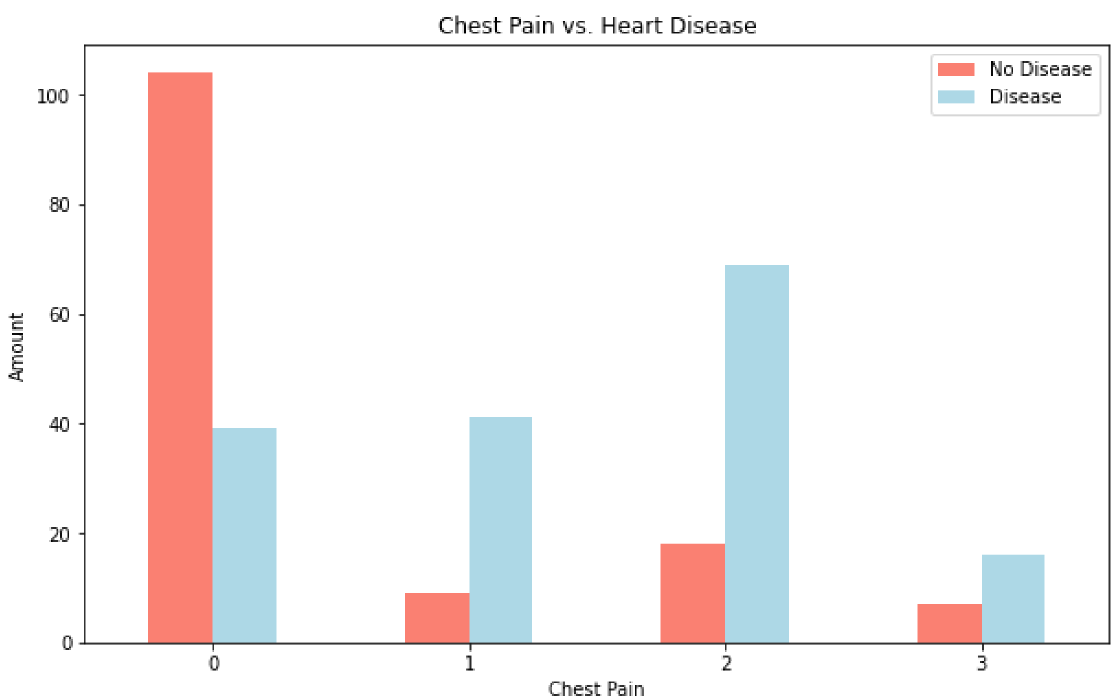


1.4.6 Heart Disease Frequency per Chest Pain type

[23]:
```
pd.crosstab(df["cp"], df["target"])
```

[23]:
```
target    0   1
cp
0       104  39
1         9  41
2        18  69
3         7  16
```

[28]:
```
# Plot a visual of the crosstab
pd.crosstab(df["cp"], df["target"]).plot(kind="bar",
                                        figsize=(10,6),
                                        color=["salmon", "lightblue"])

# Adding info
plt.title("Chest Pain vs. Heart Disease")
plt.xlabel("Chest Pain")
plt.ylabel("Amount")
plt.legend(["No Disease", "Disease"])
plt.xticks(rotation=0);
```

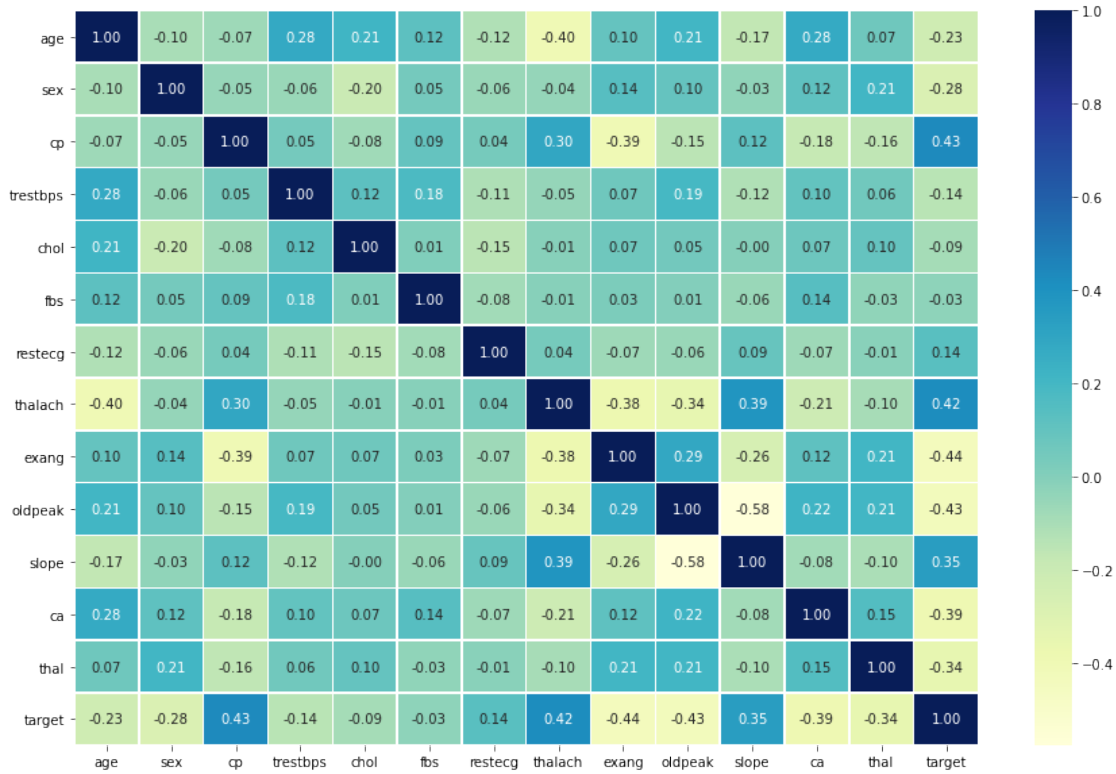


[29]:
```
# Make a correlation matrix
df.corr()
```

[29]:
```
               age       sex        cp  trestbps      chol       fbs  \
age       1.000000 -0.098447 -0.068653  0.279351  0.213678  0.121308
sex      -0.098447  1.000000 -0.049353 -0.056769 -0.197912  0.045032
cp       -0.068653 -0.049353  1.000000  0.047608 -0.076904  0.094444
trestbps  0.279351 -0.056769  0.047608  1.000000  0.123174  0.177531
chol      0.213678 -0.197912 -0.076904  0.123174  1.000000  0.013294
fbs       0.121308  0.045032  0.094444  0.177531  0.013294  1.000000
restecg  -0.116211 -0.058196  0.044421 -0.114103 -0.151040 -0.084189
thalach  -0.398522 -0.044020  0.295762 -0.046698 -0.009940 -0.008567
exang     0.096801  0.141664 -0.394280  0.067616  0.067023  0.025665
oldpeak   0.210013  0.096093 -0.149230  0.193216  0.053952  0.005747
slope    -0.168814 -0.030711  0.119717 -0.121475 -0.004038 -0.059894
ca        0.276326  0.118261 -0.181053  0.101389  0.070511  0.137979
thal      0.068001  0.210041 -0.161736  0.062210  0.098803 -0.032019
target   -0.225439 -0.280937  0.433798 -0.144931 -0.085239 -0.028046

           restecg   thalach     exang   oldpeak     slope        ca  \
age      -0.116211 -0.398522  0.096801  0.210013 -0.168814  0.276326
sex      -0.058196 -0.044020  0.141664  0.096093 -0.030711  0.118261
cp        0.044421  0.295762 -0.394280 -0.149230  0.119717 -0.181053
trestbps -0.114103 -0.046698  0.067616  0.193216 -0.121475  0.101389
chol     -0.151040 -0.009940  0.067023  0.053952 -0.004038  0.070511
fbs      -0.084189 -0.008567  0.025665  0.005747 -0.059894  0.137979
restecg   1.000000  0.044123 -0.070733 -0.058770  0.093045 -0.072042
thalach   0.044123  1.000000 -0.378812 -0.344187  0.386784 -0.213177
exang    -0.070733 -0.378812  1.000000  0.288223 -0.257748  0.115739
oldpeak  -0.058770 -0.344187  0.288223  1.000000 -0.577537  0.222682
slope     0.093045  0.386784 -0.257748 -0.577537  1.000000 -0.080155
ca       -0.072042 -0.213177  0.115739  0.222682 -0.080155  1.000000
thal     -0.011981 -0.096439  0.206754  0.210244 -0.104764  0.151832
target    0.137230  0.421741 -0.436757 -0.430696  0.345877 -0.391724

              thal    target
age       0.068001 -0.225439
sex       0.210041 -0.280937
cp       -0.161736  0.433798
trestbps  0.062210 -0.144931
chol      0.098803 -0.085239
fbs      -0.032019 -0.028046
restecg  -0.011981  0.137230
thalach  -0.096439  0.421741
exang     0.206754 -0.436757
oldpeak   0.210244 -0.430696
slope    -0.104764  0.345877
```

```
ca         0.151832 -0.391724
thal       1.000000 -0.344029
target    -0.344029  1.000000
```

[31]:
```
# Let's make the correlation matrix more visible
corr_mat = df.corr()

fig, ax = plt.subplots(figsize=(15,10))
ax = sns.heatmap(corr_mat, annot=True,linewidths=0.5, fmt=".2f", cmap="YlGnBu");
```



1.5 5. Modelling

[32]:
```
df.head()
```

[32]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
```

```
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

[33]:
```python
# Split the data into X and y
X = df.drop("target", axis=1)
y = df["target"]
```

[34]:
```python
X
```

[34]:
```
     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]
```

[35]:
```python
y
```

[35]:
```
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
```

[36]:
```
# Split the data into train and test sets

np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[37]:
```
X_train
```

[37]:
```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
132   42    1   1       120   295    0        1      162      0      0.0
202   58    1   0       150   270    0        0      111      1      0.8
196   46    1   2       150   231    0        1      147      0      3.6
75    55    0   1       135   250    0        0      161      0      1.4
176   60    1   0       117   230    1        1      160      1      1.4
..   …  …  ..       …   …  …       …      …    …      …
188   50    1   2       140   233    0        1      163      0      0.6
71    51    1   2        94   227    0        1      154      1      0.0
106   69    1   3       160   234    1        0      131      0      0.1
270   46    1   0       120   249    0        0      144      0      0.8
102   63    0   1       140   195    0        1      179      0      0.0

     slope  ca  thal
132      2   0     2
202      2   0     3
196      1   0     2
75       1   0     2
176      2   2     3
..     …  ..   …
188      1   1     3
71       2   1     3
106      1   1     2
270      2   0     3
102      2   2     2

[242 rows x 13 columns]
```

[38]:
```
y_train
```

[38]:
```
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 that we have got our data split into train and test set,

We will train (find patterns) it on training set.

We will test it(use the patterns) on test set.

We are going to use three machine learning models for our classification problem: 1. Logistic Regression 2. K-Nearest Neighbors Classifier 3. Random Forest Regressor

[40]:
```python
# 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 Scores machine learning models with the given data.
    models: A dictionary containing models to be tested.
    X_train: training data(no labels)
    X_test: test data(no labels)
    y_train: training labels
    y_test: test labels
    """

    # Set up a random seed
    np.random.seed(42)

    # Create a dictionary to store our model scores
    model_scores = {}

    # Loop through the models dictionary
    for name, model in models.items():
        # Fit training data into the model
        model.fit(X_train, y_train)
```

```
        # Evaluate the model on test data and store the score in dictionary
        model_scores[name] = model.score(X_test, y_test)

    return model_scores
```

[41]:
```
model_scores = fit_and_score(models, X_train, X_test, y_train, y_test)
model_scores
```

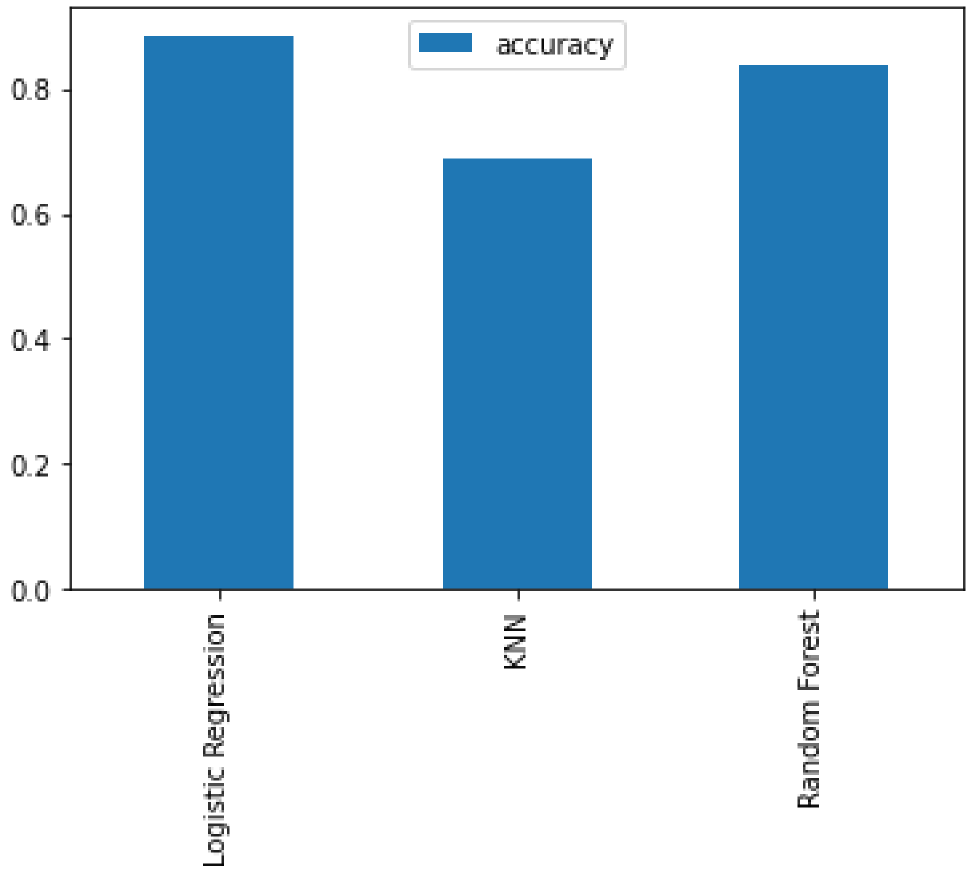
```
D:\ds_and_ml_projects\heart-disease-project\env\lib\site-
packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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(
```

[41]:
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

1.5.1 Model Comparison

[43]:
```
model_compare = pd.DataFrame(model_scores, index=["accuracy"])
model_compare.T.plot.bar()
```

[43]:
```
<AxesSubplot:>
```



Now we should look at the following steps to evaluate our models:

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

1.5.2 Hyperparameter Tuning (by hand)

[45]:
```python
# Let's tune KNN

train_scores = []
test_scores = []
```

```python
# Set up a list of n_neighbors values
neighbors = range(1,21)

# Instantiate KNN
knn = KNeighborsClassifier()

# Loop through different values of n_neighbors
for i in neighbors:
    # Set parameter of KNN
    knn.set_params(n_neighbors=i)

    # Fit the training set on the model
    knn.fit(X_train, y_train)

    # Update train scores list
    train_scores.append(knn.score(X_train, y_train))

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

[46]:
```python
train_scores
```

[46]:
```
[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]
```

[47]:
```python
test_scores
```

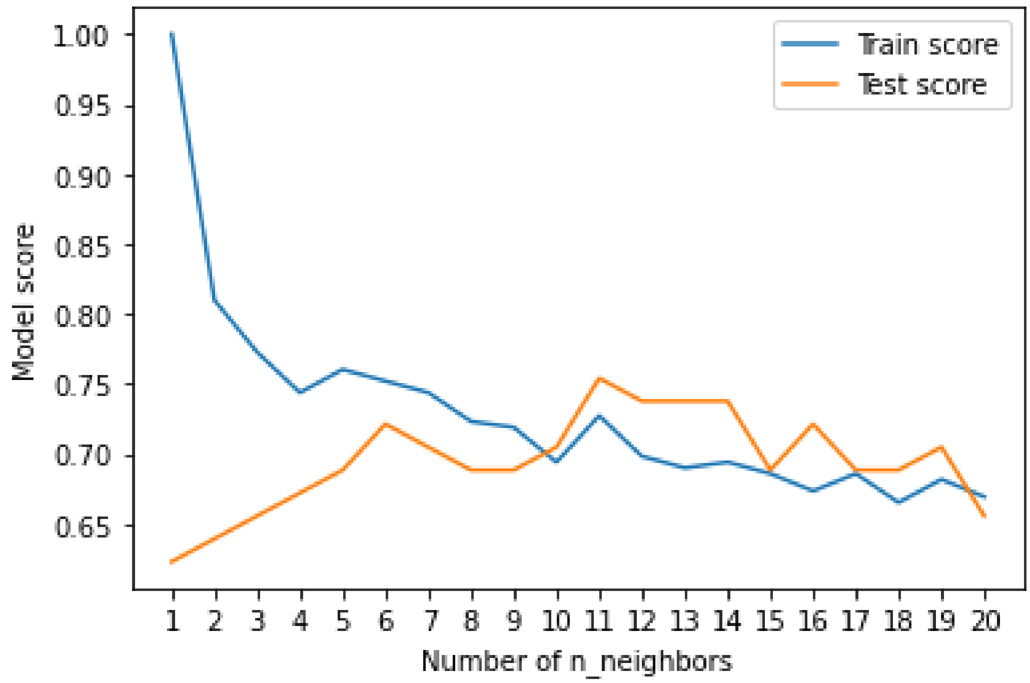
[47]:
```
[0.6229508196721312,
 0.639344262295082,
```

```
 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]
```

[54]:

```
# Let's visualize the train and test scores

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 n_neighbors")
plt.ylabel("Model score")
plt.legend()

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

```
Max KNN score on test data: 75.41 %
```



1.6 Hyperparameter tuning with RandomizedSearchCV

We are going to tune:

- LogisticRegression()
- RandomForestClassifier()

… using RandomizedSearchCV

[55]:
```
# Create a hyper parameter for LogisticRegression

log_reg_grid = {
    "C": np.logspace(-4,4,20),
    "solver":["liblinear"],
}

# Create a hyper parameter 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),
}
```

[56]:
```python
# Tune logistic regression

np.random.seed(42)

# Setup hyperparameter for LogisticRegression

rs_log_reg = RandomizedSearchCV(LogisticRegression(),
                                param_distributions=log_reg_grid,
                                cv=5,
                                n_iter=20,
                                verbose=True)

rs_log_reg.fit(X_train, y_train)
```

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

[56]:
```
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)
```

[57]:
```python
rs_log_reg.best_params_
```

[57]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

[58]:
```python
rs_log_reg.score(X_test, y_test)
```

[58]:
```
0.8852459016393442
```

[62]:
```python
# Tune RandomForestClassifier

np.random.seed(42)

# Setup hyperparameter for RandomForestClassifier

rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                           param_distributions=rf_grid,
                           cv=5,
                           n_iter=50,
                           verbose=True)

rs_rf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

```
[62]: RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=50,
                         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,
160, 210, 260, 310, 360, 410, 460, 510, 560, 610,
       660, 710, 760, 810, 860, 910, 960])},
                         verbose=True)
```

```
[63]: rs_rf.best_params_
```

```
[63]: {'n_estimators': 260,
       'min_samples_split': 16,
       'min_samples_leaf': 17,
       'max_depth': 3}
```

```
[65]: rs_rf.score(X_test, y_test)
```

```
[65]: 0.8688524590163934
```

1.6.1 Hyperparameter tuning using GridSearchCV

Our LogisticRegression model is still better than the hyperparameter tuned RandomForestClassifier.

Now will use GridSearchCV in our LogisticRegression model to again tune it.

```
[72]: # Different hyperparameters for our LogisticRegression model

np.random.seed(42)

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

# Setup hyperparameter for GridSearchCV

gs_log_reg = GridSearchCV(LogisticRegression(),
                          param_grid= log_reg_grid,
                          cv=5,
                          verbose=True)

# Fit our data in the hyperparameter tuned model

gs_log_reg.fit(X_train, y_train)
```

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

[72]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.00000000e-04, 1.88739182e-04,
3.56224789e-04, 6.72335754e-04,
       1.26896100e-03, 2.39502662e-03, 4.52035366e-03, 8.53167852e-03,
       1.61026203e-02, 3.03919538e-02, 5.73615251e-02, 1.08263673e-01,
       2.04335972e-01, 3.85662042e-01, 7.27895384e-01, 1.37382380e+00,
       2.59294380e+00, 4.89390092e+00, 9.23670857e+00, 1.74332882e+01,
       3.29034456e+01, 6.21016942e+01, 1.17210230e+02, 2.21221629e+02,
       4.17531894e+02, 7.88046282e+02, 1.48735211e+03, 2.80721620e+03,
       5.29831691e+03, 1.00000000e+04]),
                         'solver': ['liblinear']},
             verbose=True)
```

[73]:
```
gs_log_reg.best_params_
```

[73]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

[74]:
```
gs_log_reg.score(X_test, y_test)
```

[74]:
```
0.8852459016393442
```

1.6.2 Evaluating our tuned machine learning classifier, beyond accuracy

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

To make comparisons and evaluate our training model, firs we need to make predictions

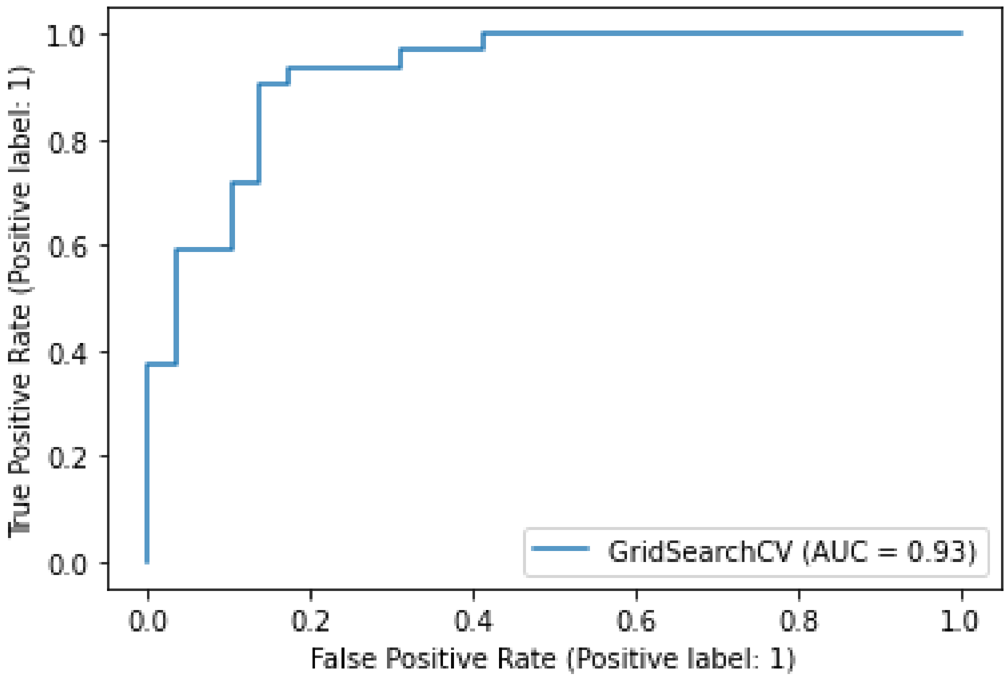
[76]:
```
# Make predictions using our tuned model
y_preds = gs_log_reg.predict(X_test)
```

[77]:
```
y_preds
```

[77]:
```
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)
```

[78]:
```
# Plot ROC curve and AUC metric
plot_roc_curve(gs_log_reg, X_test, y_test)
```

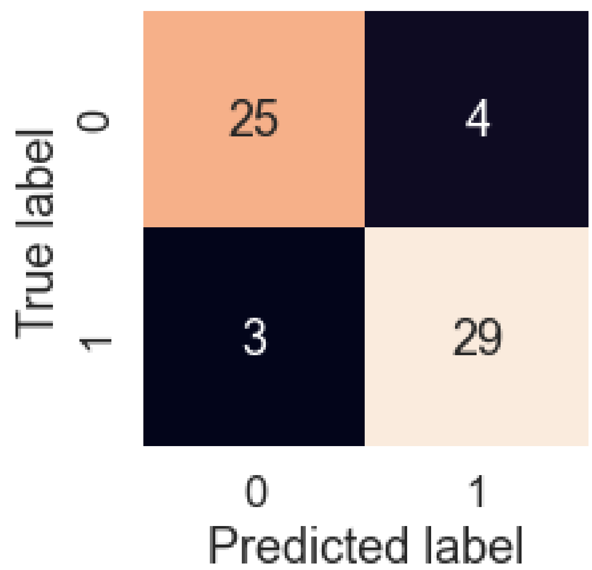
[78]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1cb3b80e070>
```



[79]:
```
# Confusion matrix
print(confusion_matrix(y_test, y_preds))
```

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

[82]:
```
# Plot confusion matrix using seaborn's heatmap
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("Predicted label")
    plt.ylabel("True label")

plot_conf_mat(y_test, y_preds)
```



Now let's see the classification report and cross validated precision, recall and f1 score

[83]:
```
# Classification report
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
```

1.6.3 Calculate evaluation metrics using cross validation

We are going to calculate accuracy, precision, recall and f1 score using `cross_val_score()`

[84]:
```
# Checking the best hyperparameters

gs_log_reg.best_params_
```

[84]: {'C': 0.20433597178569418, 'solver': 'liblinear'}

[85]:
```
# Setup a new classification model with the best hyperparameters
clf = LogisticRegression(C=0.20433597178569418,
                         solver="liblinear")
```

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

```
[86]: 0.8446994535519124
```

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

```
[87]: 0.8207936507936507
```

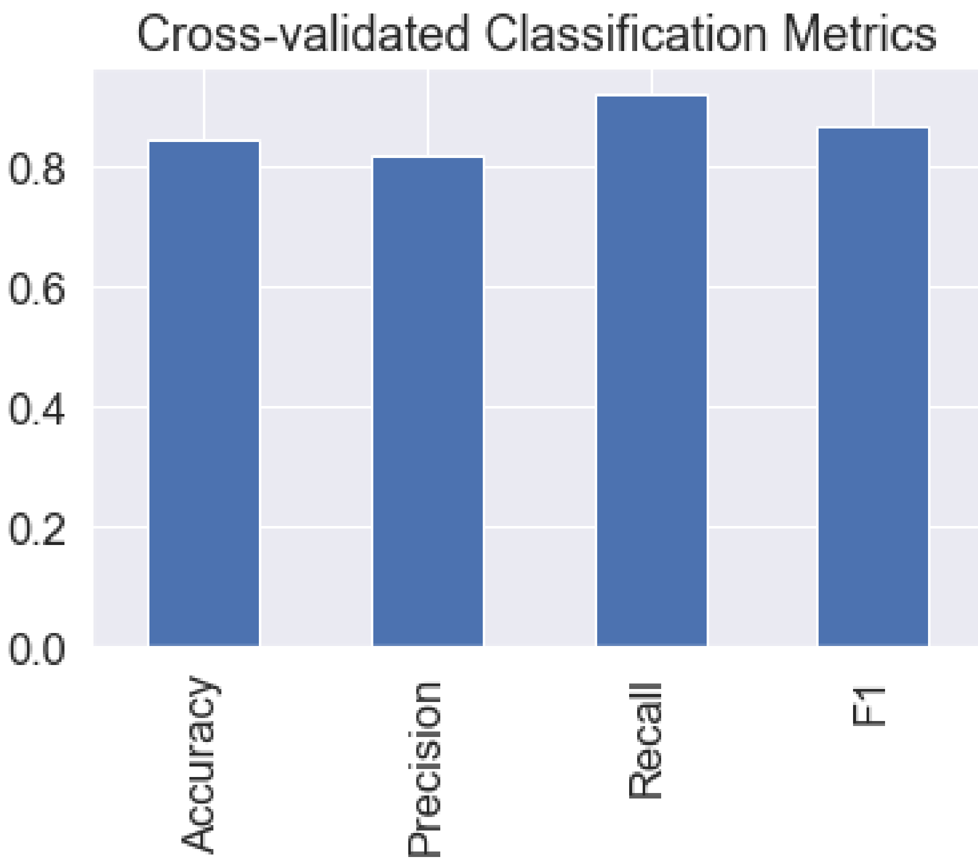
```
[90]: # Cross-validated recall
cv_rec = cross_val_score(clf, X, y, cv=5, scoring="recall")
cv_rec = np.mean(cv_rec)
cv_rec
```

```
[90]: 0.9212121212121213
```

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

```
[91]: 0.8673007976269721
```

```
[92]: # Visualize cross-validated score

cv_metrics = pd.DataFrame({"Accuracy": cv_acc,
                          "Precision": cv_prec,
                          "Recall": cv_rec,
                          "F1": cv_f1},
                         index=[0])

cv_metrics.T.plot.bar(title="Cross-validated Classification Metrics",
                     legend=False);
```



1.6.4 Feature Importance

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

Let's find feature importance for our LogisticRegression model.

[96]:
```
# Fit an instance of logistic regression

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

clf.fit(X_train, y_train);
```

[97]:
```
# Check coefficent

clf.coef_
```

[97]:
```
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]])
```

[102]:
```
# Match coef's of features to columns
feature_dict = dict(zip(df.columns, list(clf.coef_[0])))
feature_dict
```

[102]:
```
{'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}
```

[104]:
```
# Visualize feature dict
feature_df = pd.DataFrame(feature_dict, index=[0])
feature_df.T.plot.bar(title="Feature Importance", legend=False);
```