end-to-end-heart-disease-classification

February 11, 2026

1 Predicting heart disease using machine learning

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

We're 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 a patient, can we predict whether or not the have heart disease?

1.2 2. Data

The original data come from: `https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data`

1.3 3. Evaluation

> If we can reach 95% accuracy at predicting whether or not a patient has heart disease during the proof of concept we'll pursue the project

1.4 4. Features

This is where you will get diffrent information about each of the features of your data.

Create Data Dictionary

```
1. id (Unique id for each patient)
2. age (Age of the patient in years)
3. origin (place of study)
4. sex (Male/Female)
5. cp chest pain type ([typical angina, atypical angina, non-anginal, asymptomatic])
6. trestbps resting blood pressure (resting blood pressure (in mm Hg on admission to the hospita
7. chol (serum cholesterol in mg/dl)
8. fbs (if fasting blood sugar > 120 mg/dl)
9. restecg (resting electrocardiographic results)
 -- Values: [normal, stt abnormality, lv hypertrophy]
10. thalach: maximum heart rate achieved
11. exang: exercise-induced angina (True/ False)
```

```
12. oldpeak: ST depression induced by exercise relative to rest
13. slope: the slope of the peak exercise ST segment
14. ca: number of major vessels (0-3) colored by fluoroscopy
15. thal: [normal; fixed defect; reversible defect]
16. target: the predicted attribute
```

1.5 Preparing the tools

We are going to use pandas, matplotlib and NumPy for data analysis and manipulation

[72]:
```python
# Import all the tools

# Regular EDA (exploring data analysis) and plotting libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 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, f1_score, recall_score
from sklearn.metrics import RocCurveDisplay
```

1.6 Load Data

[2]:
```python
df = pd.read_csv("./data/heart-disease.csv")
df.shape # rows, columns
```

[2]: (303, 14)

[3]:
```python
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
```

1.7 Data Exploration (exploratory data analysis or EDA)

The goal here is to find out more about the data and become a aubject matter export on the dataset you are working with.

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

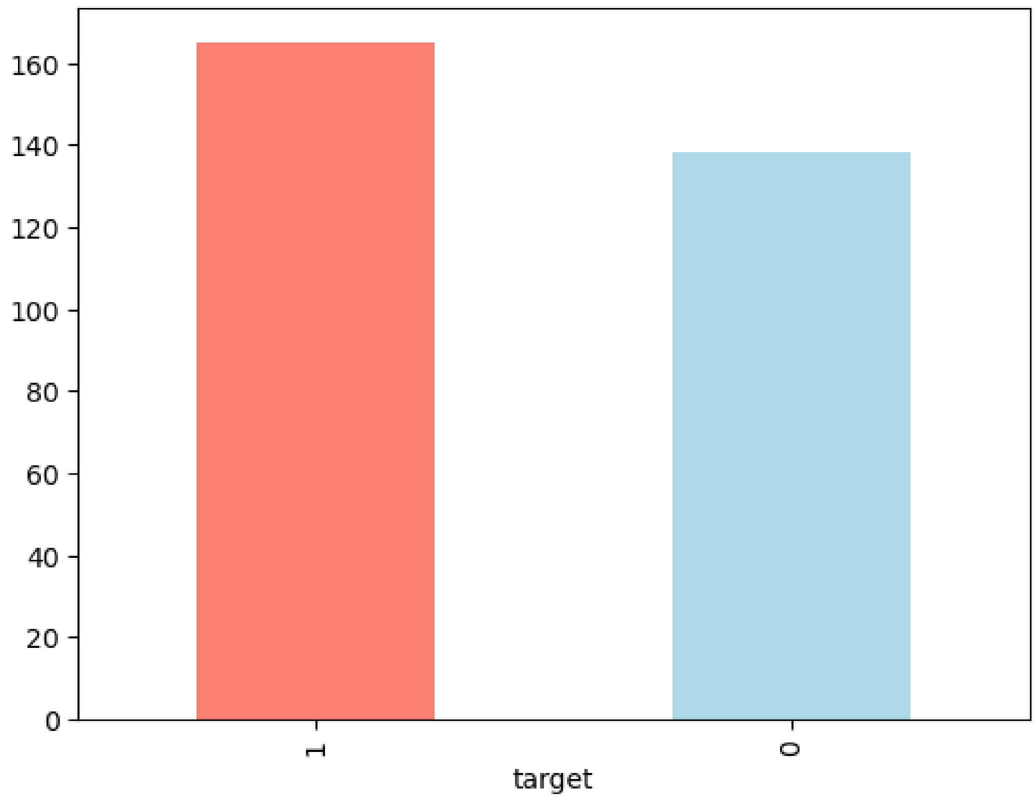
[4]:
```python
df.tail()
```

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

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

[6]:
```python
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]:
```python
#Are there any 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
```

1.7.1 Heart Disease Freuquency according to Sex

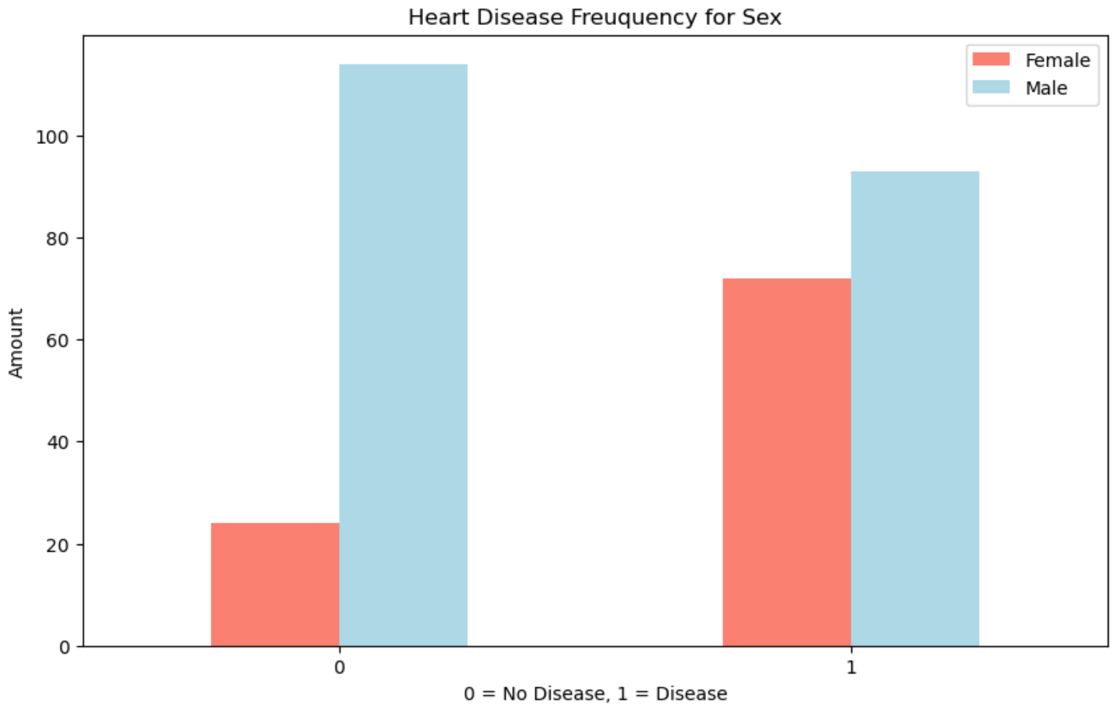
[10]:
```
df.sex.value_counts()
```

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

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

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

[12]:
```
# Create a plot of crosstab
pd.crosstab(df.target, df.sex).plot(kind = "bar", figsize=(10,6),␣
 ↪color=["salmon","lightblue"])

plt.title("Heart Disease Freuquency for Sex")
plt.xlabel("0 = No Disease, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female","Male"])

plt.xticks(rotation=0);
```



[13]:
```
df.head()
```

[13]:
```
   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
```

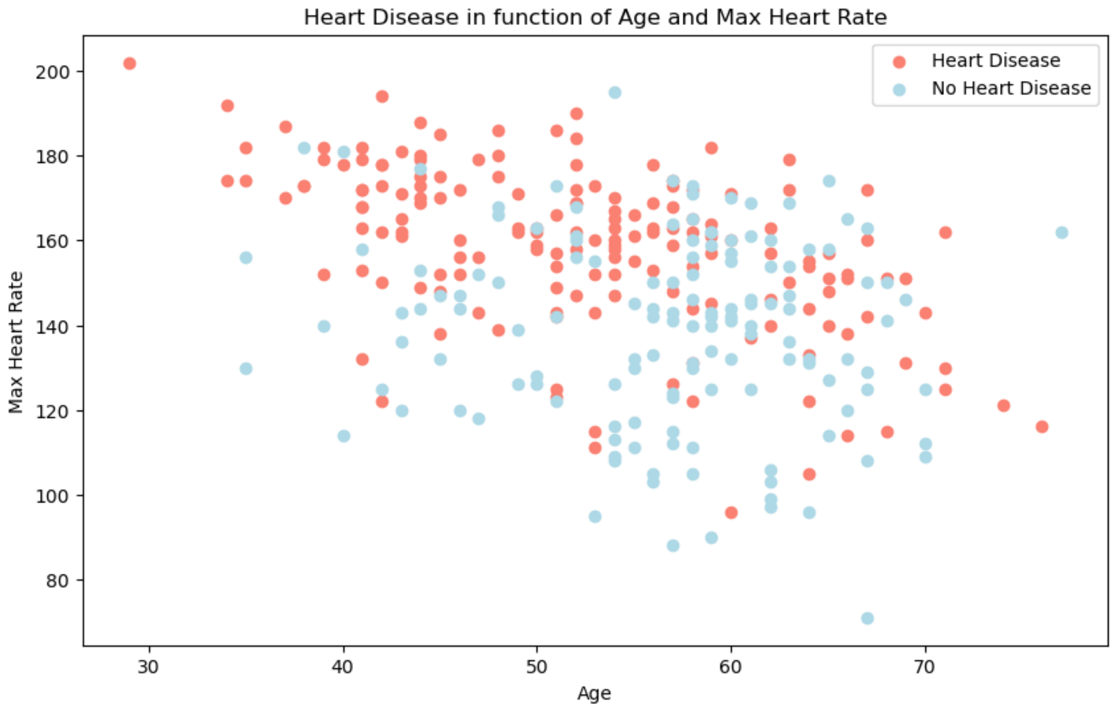
[14]:
```
df["thalach"].value_counts()
```

[14]:
```
thalach
162    11
160     9
163     9
152     8
173     8
```

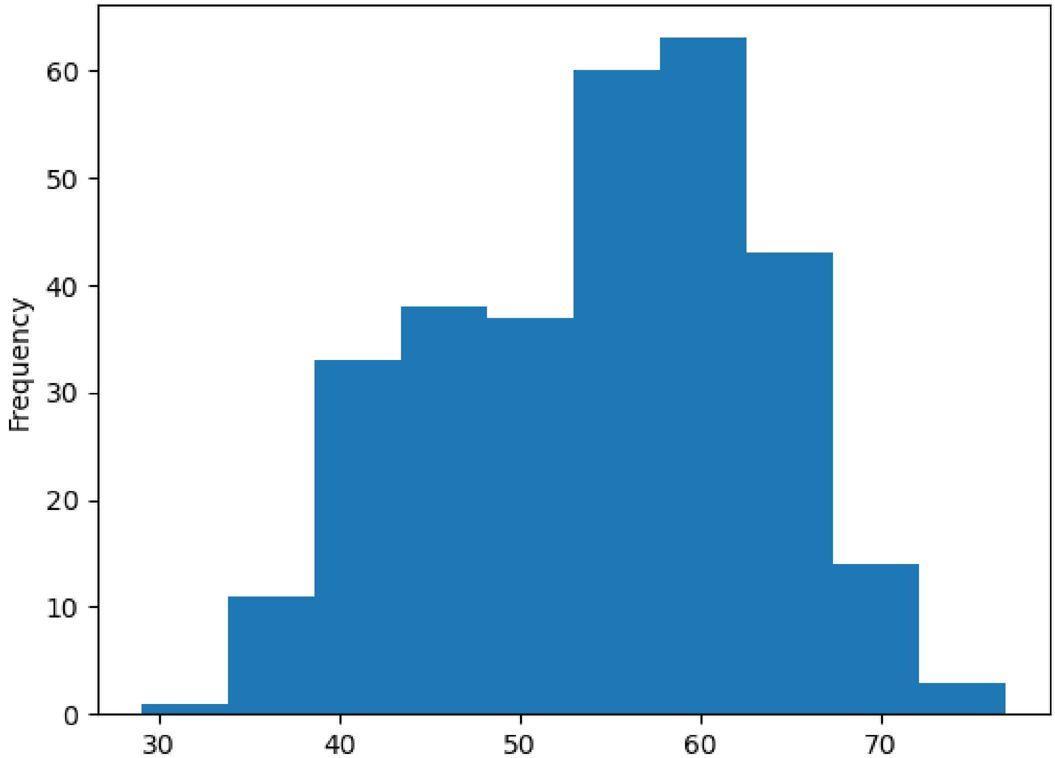
```
        ..
202     1
184     1
121     1
192     1
90      1
Name: count, Length: 91, dtype: int64
```

1.8 Age vs Max Heart Rate or Heart Disease

[15]:
```python
# Create another figure
plt.figure(figsize= (10,6))

# Scatter with possitive 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");

plt.legend(["Heart Disease","No Heart Disease"]);
plt.title("Heart Disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate");
```



[16]:
```
# Check the distribution of the age column with a histogram
df.age.plot.hist();
```



1.9 Heart Disease Frequency per Chest Pain Type

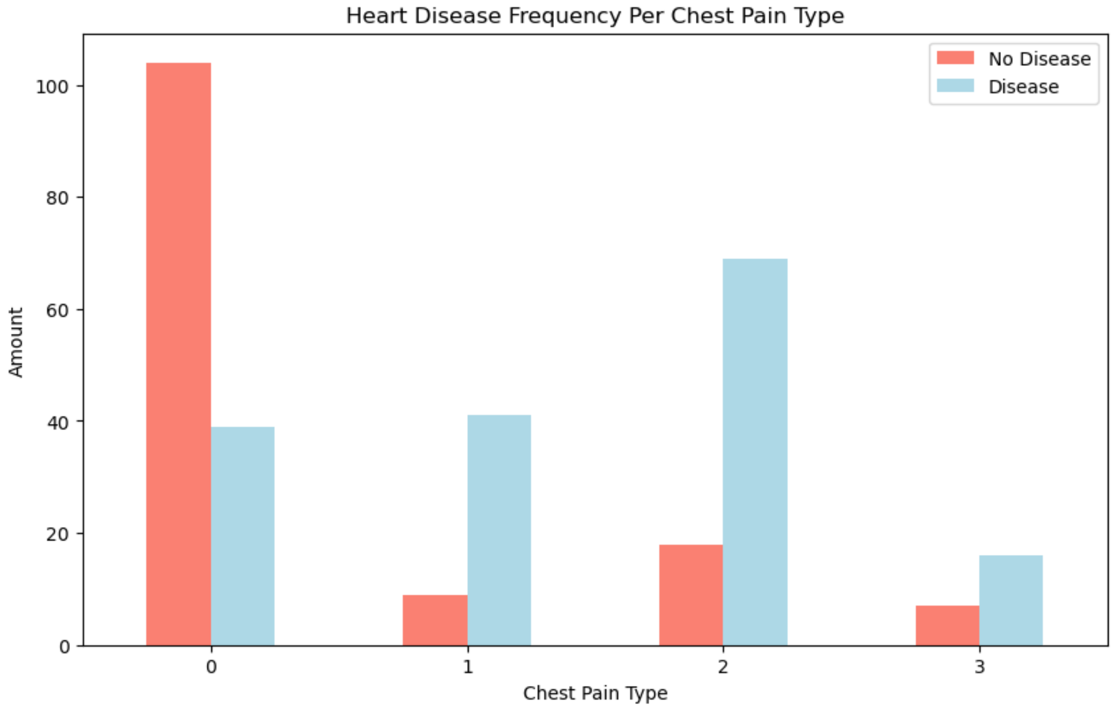
5. cp chest pain type ([typical angina, atypical angina, non-anginal, asymptomatic])

[17]:
```
pd.crosstab(df.cp,df.target)
```

[17]:
```
target    0   1
cp
0       104  39
1         9  41
2        18  69
3         7  16
```

[18]:
```
# Make the crosstab more visual

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

# Add some communication
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);
```



[19]:
```
df.head()
```

[19]:

	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	

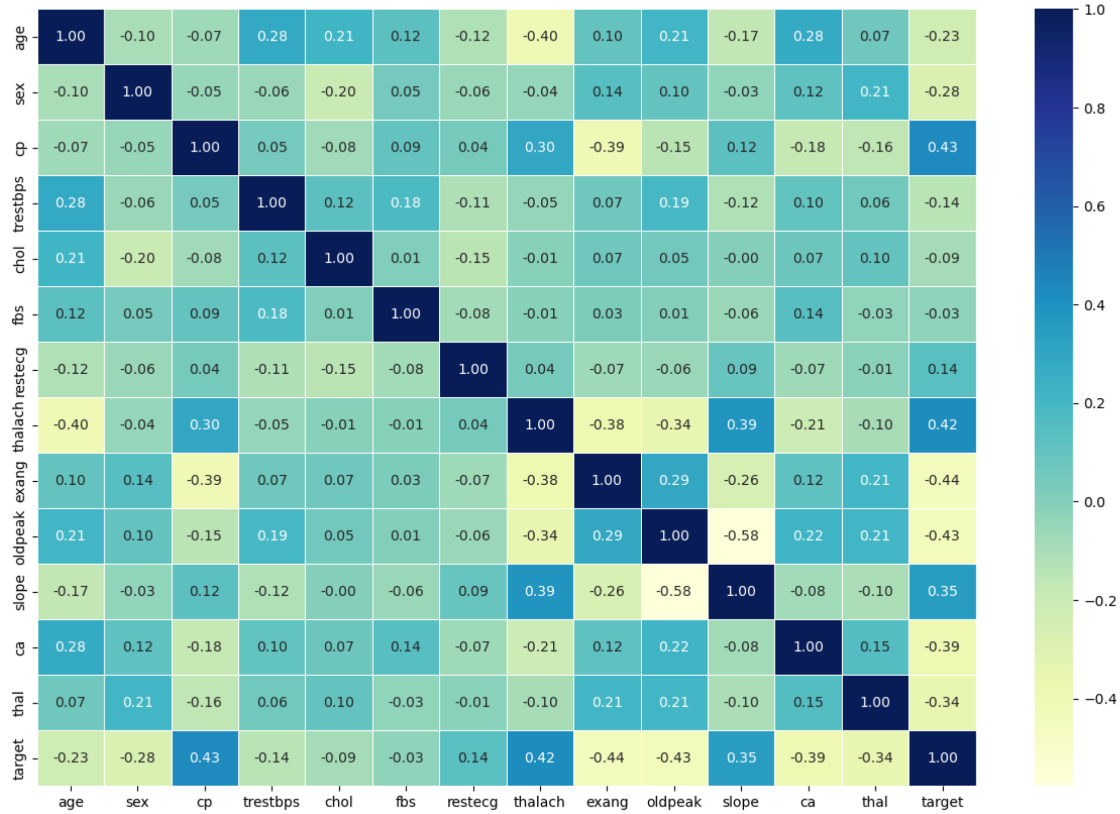
	ca	thal	target
0	0	1	1
1	0	2	1
2	0	2	1
3	0	2	1
4	0	2	1

[20]:
```
# Make a correlation matrix
df.corr()
```

[20]:

```
               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
```

[21]:
```python
# Let's make the correlation matrix a little bit prettier

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



2 5. Modeling

[22]:
```python
from sklearn.pipeline import Pipeline
```

[23]:
```python
# Split the data
X = df.drop("target",axis = 1)
y = df["target"]

#reproduce the results
np.random.seed(42)
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,shuffle=True, test_size = 0.
↪2)
```

[24]:
```
X_train.shape, X_test.shape
```

[24]: ((242, 13), (61, 13))

Now we have got our datat split into training and test sets, it's time to build a machine learning model. We will train it (find the patterns) on the training set. And we will test it (use the patterns) on the test set.

We are going to try 3 diffrent machine learning models: 1. Logistic Regression 2. K-Nearest Neighbours Classifier 3. Random Forest Classifier

[25]:
```
#Put models into a Dict

models = { "Logistic Regression": LogisticRegression(),
           "KNN": KNeighborsClassifier(),
           "Random Forest": RandomForestClassifier()}

# Create a function to fit and score our 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 diffrent Scikit.Learn machine learning models
    X_train: training data (no labels)
    X_test: test data (no labels)
    y_train: training data (labels)
    y_test: test data (labels)

    """
    np.random.seed(42)

    # Make a dict 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 it to the model_scores dict
        model_scores[name] = model.score(X_test,y_test)
    return model_scores
```

[26]:
```
models_scores = fit_and_score(models,X_train,X_test,y_train,y_test)
```

```
C:\Users\RomaM\anaconda3\Lib\site-
```

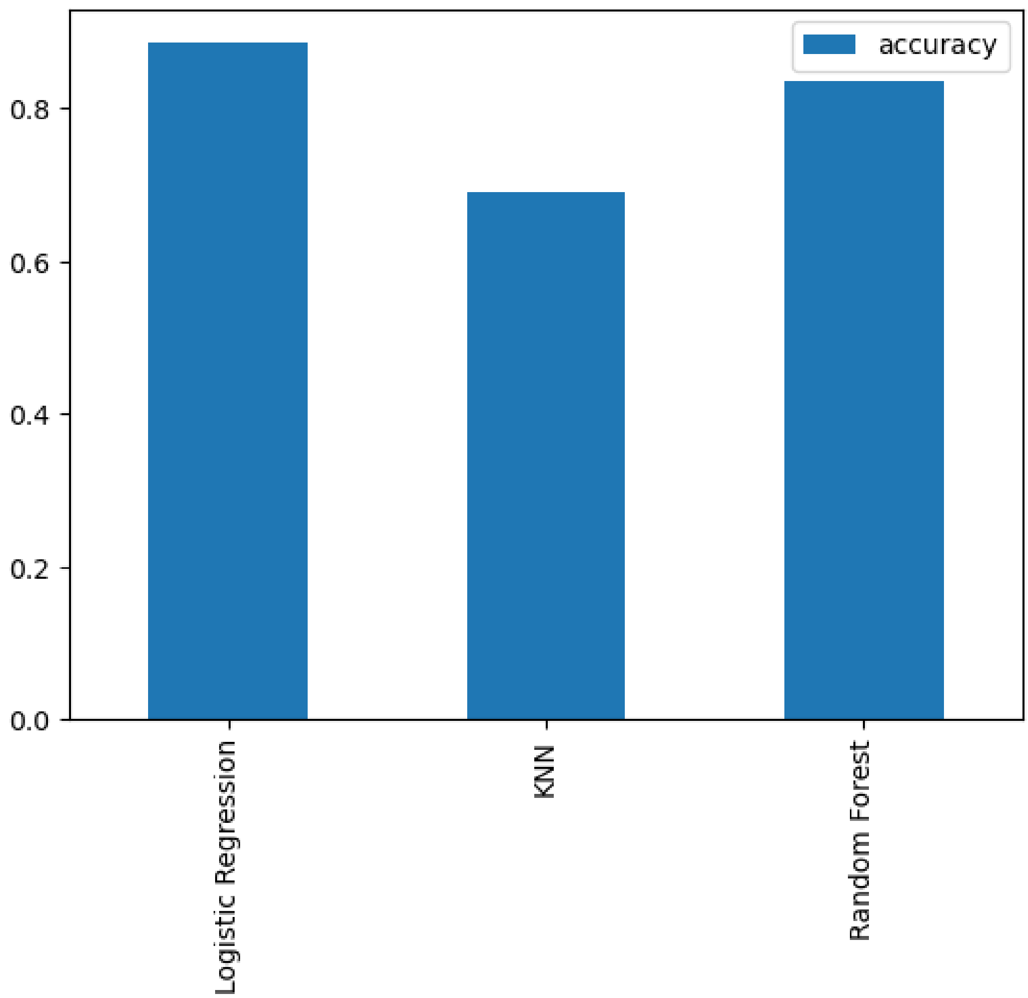
```
packages\sklearn\linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed
to converge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=100).
You might also want to 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(
```

[27]:
```python
models_scores
```

[27]:
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

2.1 Model Comparison

[28]:
```python
model_compare = pd.DataFrame(models_scores, index = ["accuracy"])
model_compare.T.plot.bar();
```



[29]:
```
model_compare
```

[29]:
```
          Logistic Regression       KNN  Random Forest
accuracy             0.885246  0.688525       0.836066
```

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

Let's look at the following:

- Hyperparameter tuning
- Feature importance
- Confusion Matrix
- Cross_Validation
- Precision
- Recall
- F1 score

- Classification report
- ROC curve
- Area under the curve (AUC)

2.2 Hyperparameter Tuning (by hand)

[30]:
```python
# Let's tune KNN

train_scores = []
test_scores = []

# Create a list of diffrent values for n neighbors
neighbors = range (1,21)

# Setup KNN instance
knn = KNeighborsClassifier();

#Loop through diffrent n_neigbors

for i in neighbors:
    knn.set_params(n_neighbors=i)

    # Fit the alg
    knn.fit(X_train,y_train)

    # Updating 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))
```

[31]:
```python
train_scores
```

[31]:
```
[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,
```

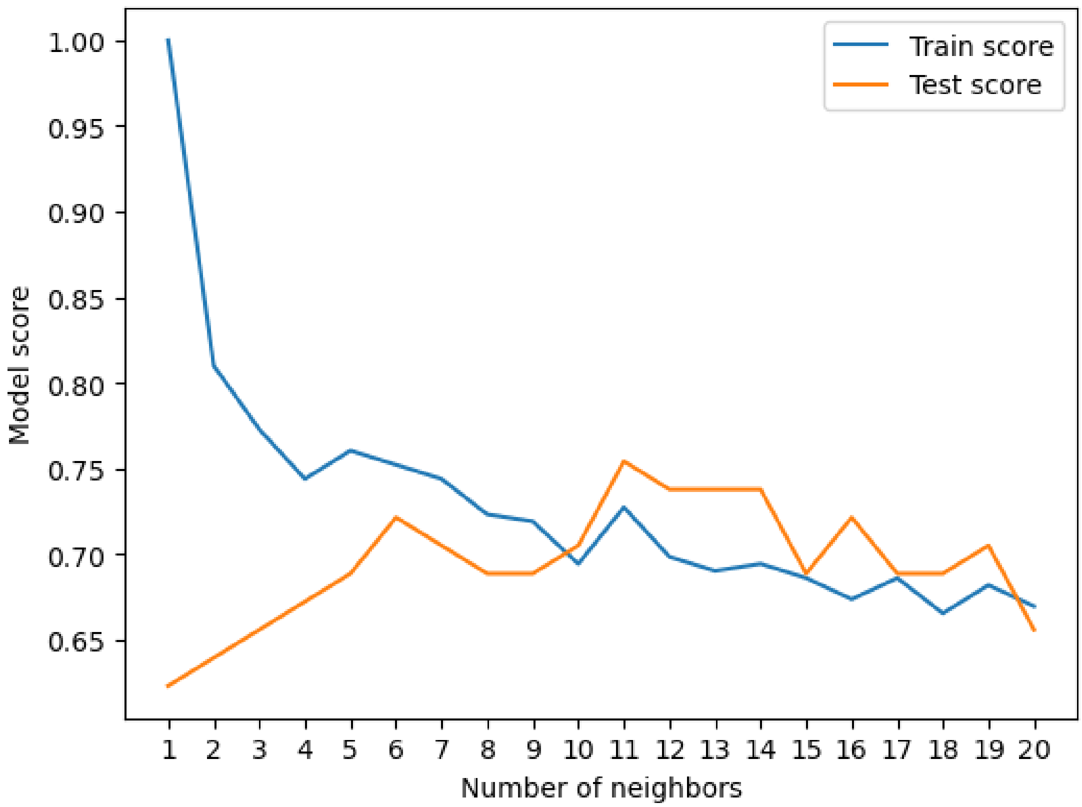
```
 0.6942148760330579,
 0.6859504132231405,
 0.6735537190082644,
 0.6859504132231405,
 0.6652892561983471,
 0.6818181818181818,
 0.6694214876033058]
```

[32]:
```python
test_scores
```

[32]:
```
[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]
```

[33]:
```python
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%
```



[34]:
```
#KNN is not the right algorithm for this problem because the other algorithms␣
↪perform far better
```

2.3 Hyperparameter tuning with RandomizedSearchCV

We are going to tune: * LogisticRegression() * RandomForestClassifier()

. . . using RSCV

[35]:
```
# Create a hyperparmeter grid for logisitcregression model
log_reg_grid={
    "C":np.logspace(-4,4,20),
    "solver":["liblinear"]
}

# Create a HP grid fpr RandomForestClass.
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 have got hyperparameters grids for each of our models, let's tune them using RandomizedSearchCV

[36]:
```
# tune LR Model

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
```

[36]:
```
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)
```

[37]:
```
rs_log_reg.best_params_
```

[37]:
```
{'solver': 'liblinear', 'C': np.float64(0.23357214690901212)}
```

[38]:
```
rs_log_reg.score(X_test,y_test)
```

[38]:
```
0.8852459016393442
```

Now we have tuned LogisticRegression(), let's do the same for RandomForest()

[39]:
```
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 LogisticRegression
```

```
rs_rf.fit(X_train,y_train)
```

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

[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, 110,
160, 210, 260, 310, 360, 410, 460, 510, 560, 610,
       660, 710, 760, 810, 860, 910, 960])},
                   verbose=True)
```

[40]:
```
rs_rf.best_params_
```

[40]:
```
{'n_estimators': np.int64(210),
 'min_samples_split': np.int64(4),
 'min_samples_leaf': np.int64(19),
 'max_depth': 3}
```

[41]:
```
rs_rf.score(X_train,y_train)
```

[41]:
```
0.8553719008264463
```

2.4 Hyperparameter Tuning with GridSearchCV

Since our LogisticRegression model provides the best scores so far, we will try and improve the HP by GridSearchCV

[42]:
```
# Diffrent 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 the model
gs_log_reg.fit(X_train,y_train)
```

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

[42]:
```
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)
```

[43]:
```
gs_log_reg.score(X_test,y_test)
```

[43]:
```
0.8852459016393442
```

2.5 Evaluating our tuned machine learning classifier, beyond accuracy

- ROC Curve and AUC Score
- Confusuin Matrix
- Classification Report
- Precision
- Recall
- F1-Score

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

To make compariosons and evaluate our trained model, first we need to make predictions.

[45]:
```
# Make Predictions with tuned model
y_preds = gs_log_reg.predict(X_test)
```

[46]:
```
y_preds
```

[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])
```

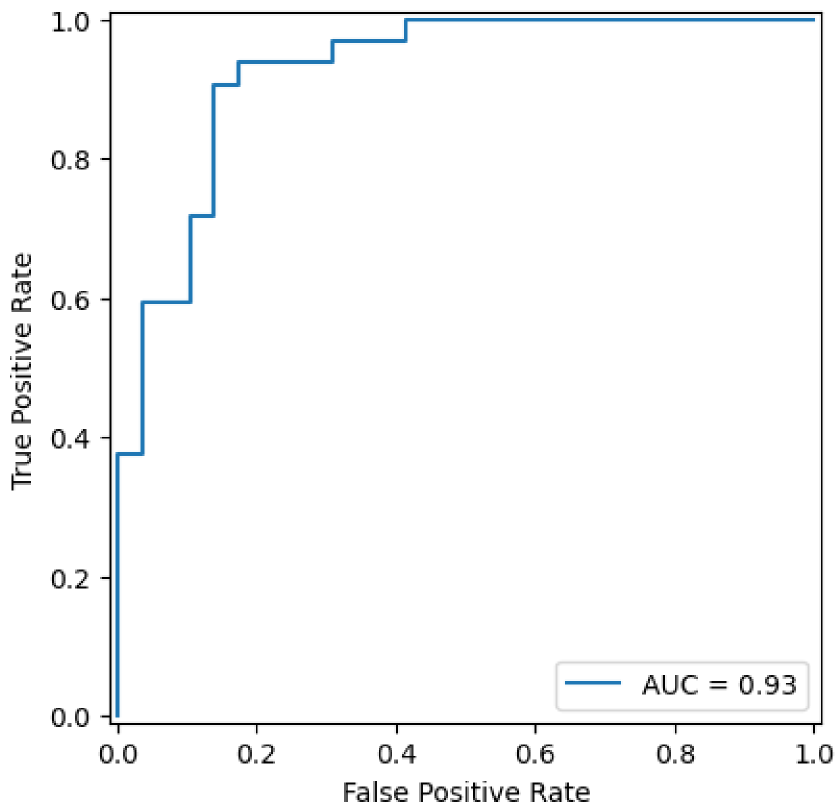
[47]:
```
y_test
```

[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
```

[58]:
```python
from sklearn import metrics

y_score = gs_log_reg.predict_proba(X_test)[:, 1]   # Wahrscheinlichkeit für␣
 ↪Klasse 1

fpr, tpr, thresholds = metrics.roc_curve(y_test, y_score)
roc_auc = metrics.auc(fpr, tpr)

disp = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc)
disp.plot()
```

[58]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x223bc39f790>
```



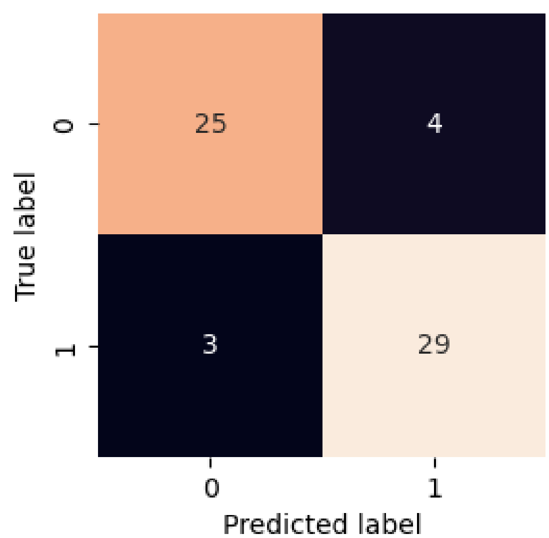
[59]:
```python
y_score
```

[59]:
```
array([0.13274739, 0.75591518, 0.81452426, 0.05469225, 0.88453364,
       0.87070287, 0.60512182, 0.00435981, 0.01376378, 0.56138489,
       0.7172079 , 0.11904138, 0.88730101, 0.06005038, 0.96750057,
```

```
0.93181128, 0.96404698, 0.09452756, 0.01769764, 0.0264636 ,
0.71543156, 0.02727603, 0.14274667, 0.71660794, 0.88198278,
0.69480581, 0.8423425 , 0.69335569, 0.01830941, 0.87782365,
0.07150765, 0.06684154, 0.01510285, 0.14314483, 0.60229465,
0.12640328, 0.6633502 , 0.85079097, 0.81898326, 0.84121548,
0.54515856, 0.79250831, 0.77817602, 0.70538842, 0.83243213,
0.02113004, 0.73216369, 0.93234387, 0.10276671, 0.06440756,
0.13470669, 0.03554564, 0.80441973, 0.95312794, 0.31714658,
0.00309601, 0.08734727, 0.93823575, 0.02813797, 0.01309874,
0.06291747])
```

[63]:

```
gs_log_reg.best_params_

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, # Annotate the boxes
                     cbar=False)
    plt.xlabel("Predicted label") # predictions go on the x-axis
    plt.ylabel("True label") # true labels go on the y-axis

plot_conf_mat(y_test, y_preds)
```



[61]:

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

[61]: array([[25, 4],
       [3, 29]])

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

[64]:
```
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
```

2.5.1 Calculating evaluation metrics using cross-validation

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

[65]:
```
# Check best Hyperparameters
gs_log_reg.best_params_
```

[65]: {'C': np.float64(0.20433597178569418), 'solver': 'liblinear'}

[66]:
```
# Create a new classifier with best parameters
clf = LogisticRegression(C= 0.20433597178569418, solver = "liblinear")
```

[73]:
```
# Cross-validated accuracy
cv_acc = cross_val_score(clf,X,y,cv = 5, scoring = "accuracy")

cv_acc.mean()
```

[73]: np.float64(0.8446994535519124)

[74]:
```
#Cross-validated precision
cv_prec = cross_val_score(clf,X,y,cv = 5, scoring = "precision")
cv_prec.mean()
```

[74]: np.float64(0.8207936507936507)

[75]:
```
#Cross-validated recall
cv_rec = cross_val_score(clf,X,y,cv = 5, scoring = "recall")
cv_rec.mean()
```

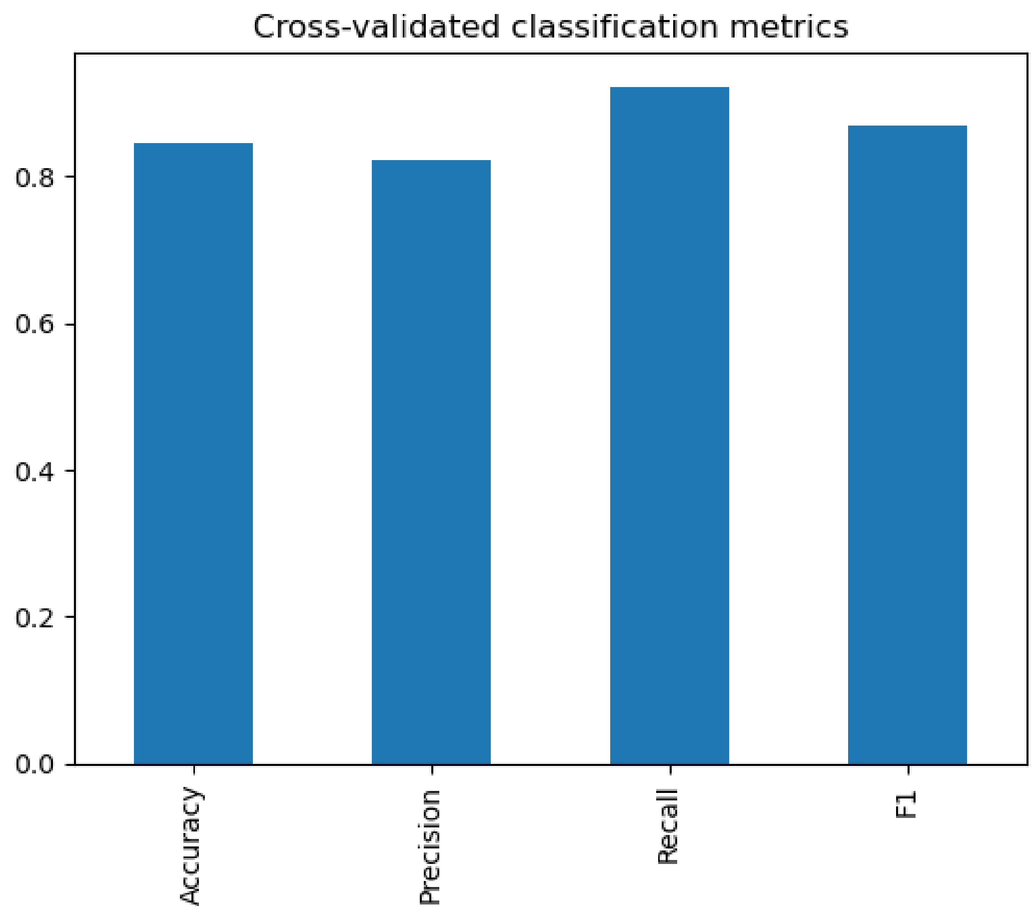
[75]: np.float64(0.9212121212121213)

[78]:
```python
#Cross-validates f1-score
cv_f1 = cross_val_score(clf,X,y,cv = 5, scoring = "f1")
cv_f1.mean()
```

[78]:
```
np.float64(0.8673007976269721)
```

[85]:
```python
# Visualize cross-validated metrics
cv_metrics = pd.DataFrame({"Accuracy": cv_acc.mean(),
                           "Precision":cv_prec.mean(),
                           "Recall":cv_rec.mean(),
                           "F1":cv_f1.mean()},
                          index = [0])
cv_metrics.T.plot.bar(title="Cross-validated classification metrics", legend=False)
```

[85]:
```
<Axes: title={'center': 'Cross-validated classification metrics'}>
```



2.5.2 Feature Importance

Feature importance is another way of asking which features contributed most to tthe outcomes of the model and how did the contribute

Finding feature importance is diffrent do each machine learning model.

Let's find the feature importance for our LR model...

[86]:
```
df.head()
```

[86]:
```
   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
```

[88]:
```
# Fit an instance of Logistic Regression
# Create a new classifier with best parameters
clf = LogisticRegression(C= 0.20433597178569418, solver = "liblinear")

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

[89]:
```
#Check coef_
clf.coef_
```

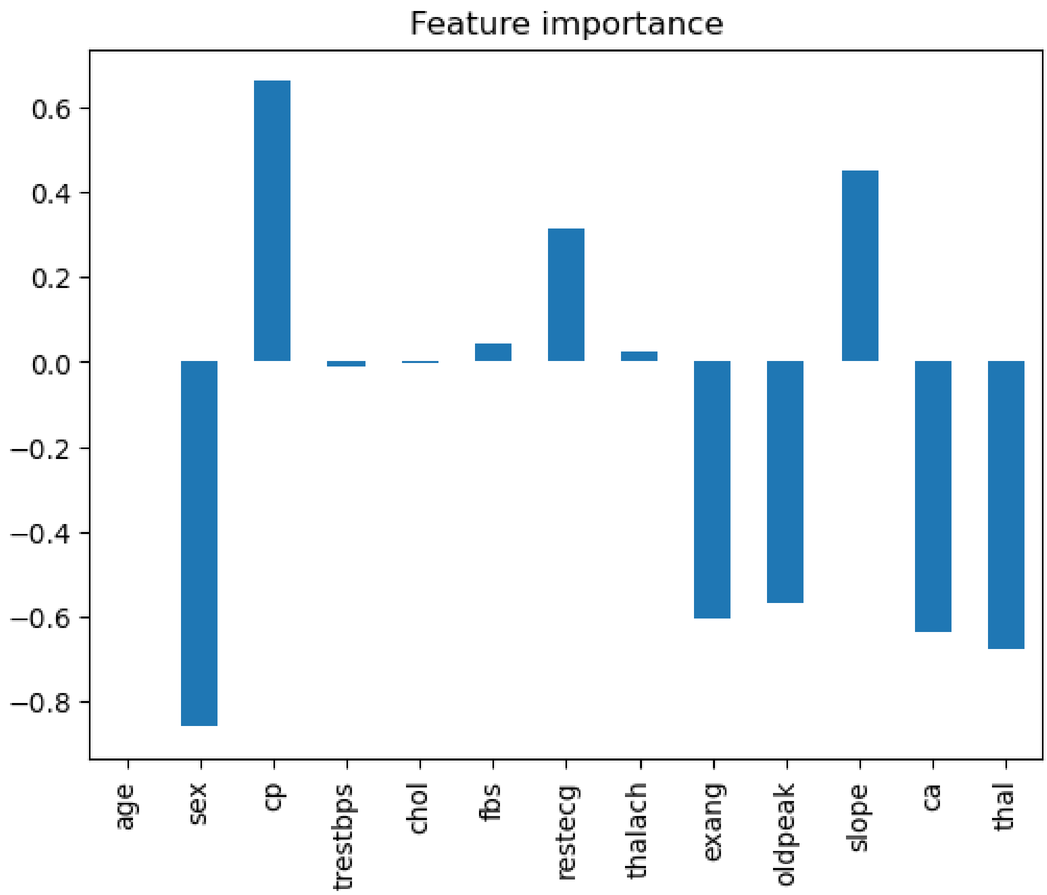
[89]:
```
array([[ 0.00316727, -0.86044582,  0.66067073, -0.01156993, -0.00166374,
         0.04386131,  0.31275787,  0.02459361, -0.60413038, -0.56862852,
         0.45051617, -0.63609863, -0.67663375]])
```

[90]:
```
feature_dict = dict(zip(df.columns,list(clf.coef_[0])))
feature_dict
```

[90]:
```
{'age': np.float64(0.0031672721856887734),
 'sex': np.float64(-0.860445816920919),
 'cp': np.float64(0.6606707303492849),
 'trestbps': np.float64(-0.011569930902919925),
 'chol': np.float64(-0.001663741604035976),
 'fbs': np.float64(0.04386130751482091),
 'restecg': np.float64(0.3127578715206996),
 'thalach': np.float64(0.02459360818122666),
```

```
 'exang': np.float64(-0.6041303799858143),
 'oldpeak': np.float64(-0.5686285194546157),
 'slope': np.float64(0.4505161679452401),
 'ca': np.float64(-0.6360986316921434),
 'thal': np.float64(-0.6766337521354281)}
```

[91]:
```
# Visualize Feature Importance
feature_df = pd.DataFrame(feature_dict, index =[0])
feature_df.T.plot.bar(title="Feature importance", legend = False);
```



[92]:
```
pd.crosstab(df["sex"],df["target"])
```

[92]:

target	0	1
sex		
0	24	72
1	114	93

[94]:
```
pd.crosstab(df["slope"],df["target"]) #  slope: the slope of the peak exercise␣
 ↪ST segment
```

[94]:
```
target   0    1
slope
0       12    9
1       91   49
2       35  107
```

2.6 6. Experimenations

If you haven't hit your evaluation metric yet… ask yourself…

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

[ ]:
```

```