Predicating 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 are going to take following approch:

1. Problem definition
2. Data
3. Evaluation
4. Features
5. Modelling
6. Experimentation

1. Problem Defination

in a statement,

> Given clinical parameters about a patient, can we predict wether or not they have heart disease?

2. Data

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

There is also a version of it avilable on keggle.

3. Evaluation

> If we can 95% accuracy at predicating whether or not a paitent has heart disease during proof of concept we'll purpose the project

4. Features

This is where you'll get diffrent information about each of the features in your data.

Create data dictionary

- ageage 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 heat related)
  - 3: Asymptomatic: chest pain not showing signs of disease:
- trestbps -resting blood pressure (in mm Hg on admission to the hospital) anythings
- cholserum cholestoral in mg/dl
- fbs (fasting blood sugar > 120 mg/dl) (1 true; 0= false) 38 restecgresting electrocardiographic results

- thalachmaximum heart rate achieved
- exangexercise induced angina (1 = yes; 0 = no) *
- oldpeakST depression induced by exercise relative to rest
- 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
- canumber of major vessels (0-3) colored by flourosopy 13
- thal3 normal; 6 fixed defect; 7 reversable defect
- target1 or 0

Preparing the tools

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

In [1]:

```
# Import all the tools we need

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

# we want our plots to apper 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 [2]:

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

Out[2]:

```
(303, 14)
```

Data Exploration (exploratory data analysis or EDA)

The goal here is to find out more about data and become a subject matter export on the datasett you're working with

1. what question(s) are you tring 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 yuo deal with it?
4. How can you add, change or remove features to get more out of your data

In [3]:

```
df.head()
```

Out[3]:

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

In [4]:

```
df.tail()
```

Out[4]:

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

In [5]:

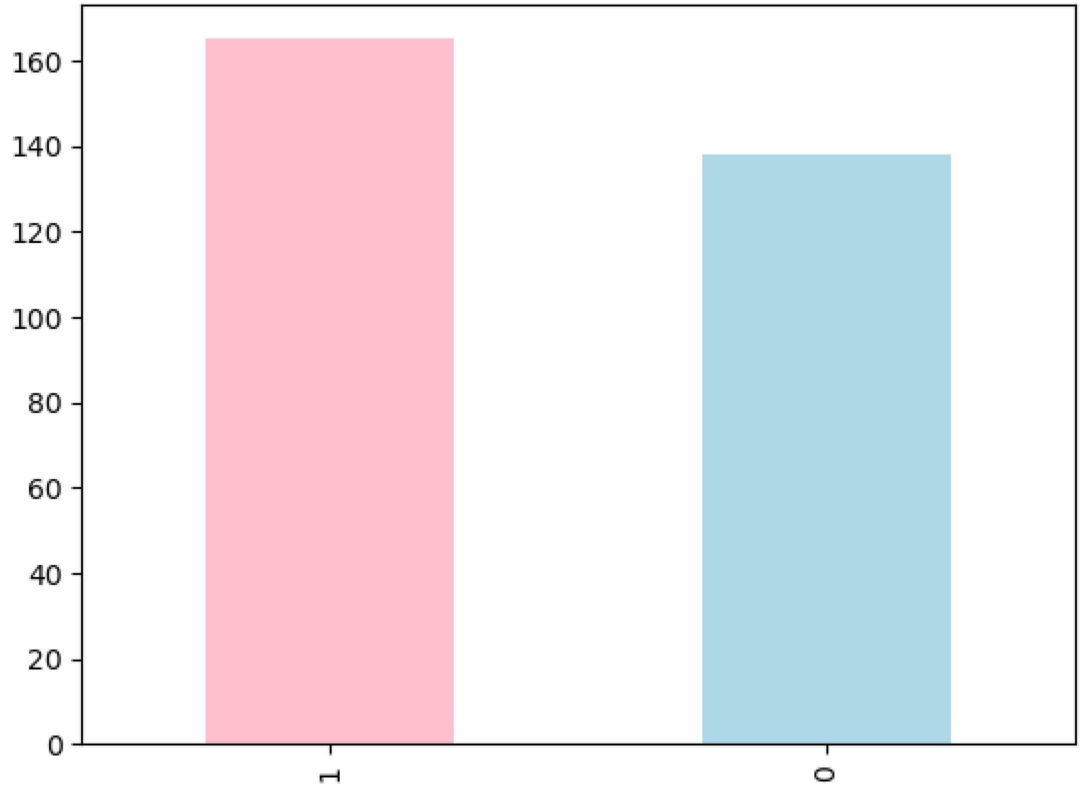
```
# Let's find out how many of each class there
df["target"].value_counts()
```

Out[5]:

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

In [6]:

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



In [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
```

In [8]:

```
# are ther any missing value
df.isna().sum()
```

Out[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
```

In [9]:

```
df.describe()
```

Out[9]:

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

In [10]:

```
# Heart Disease Frequency according to sex
df.sex.value_counts()
```

Out[10]:

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

In [11]:

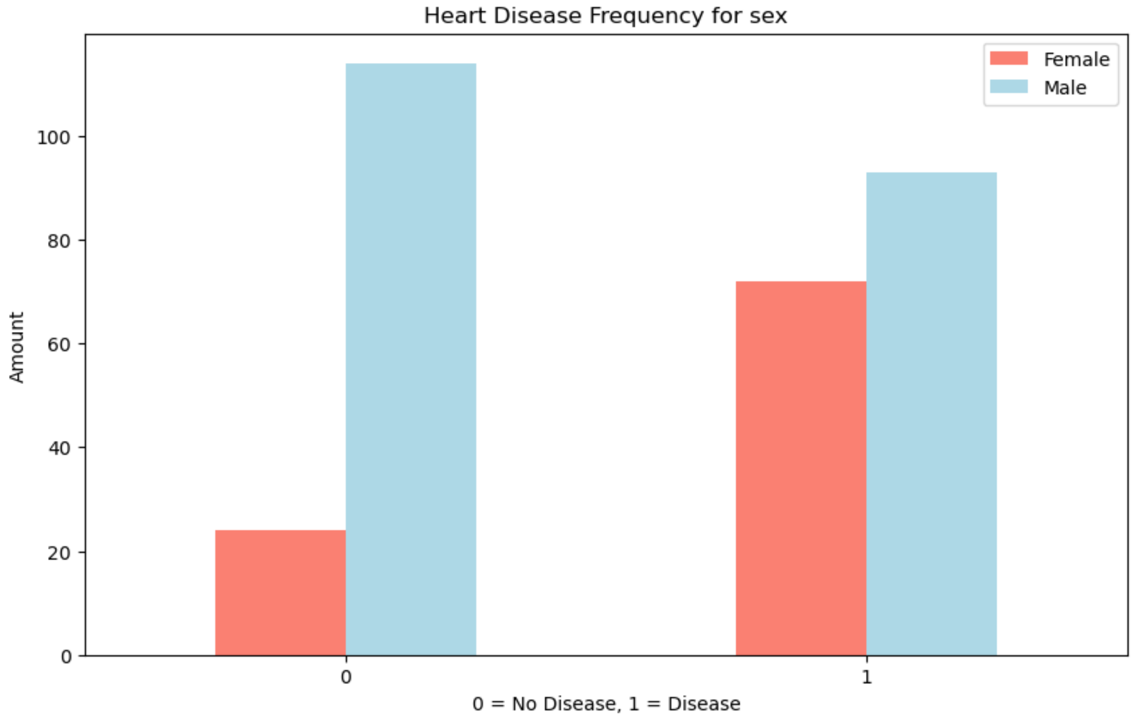
```
# Compare target column with sex column
pd.crosstab(df.target, df.sex)
```

Out[11]:

sex	0	1
target		
0	24	114
1	72	93

In [12]:

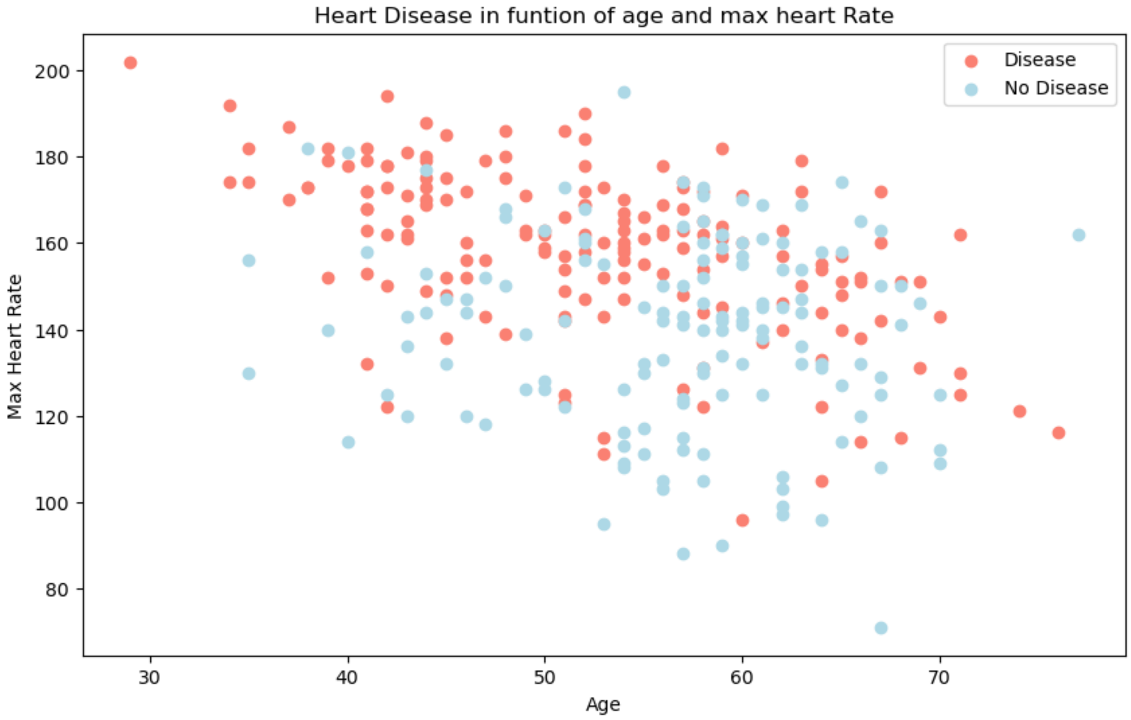
```
# create a plot of crosstab
pd.crosstab(df.target, df.sex).plot(kind="bar",
                                    figsize=(10, 6),
                                    color=["salmon", "lightblue"]);
plt.title("Heart Disease Frequency for sex")
plt.xlabel("0 = No Disease, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female", "Male"]);
plt.xticks(rotation=0);
```



Age vs Max Heart Rate for disease

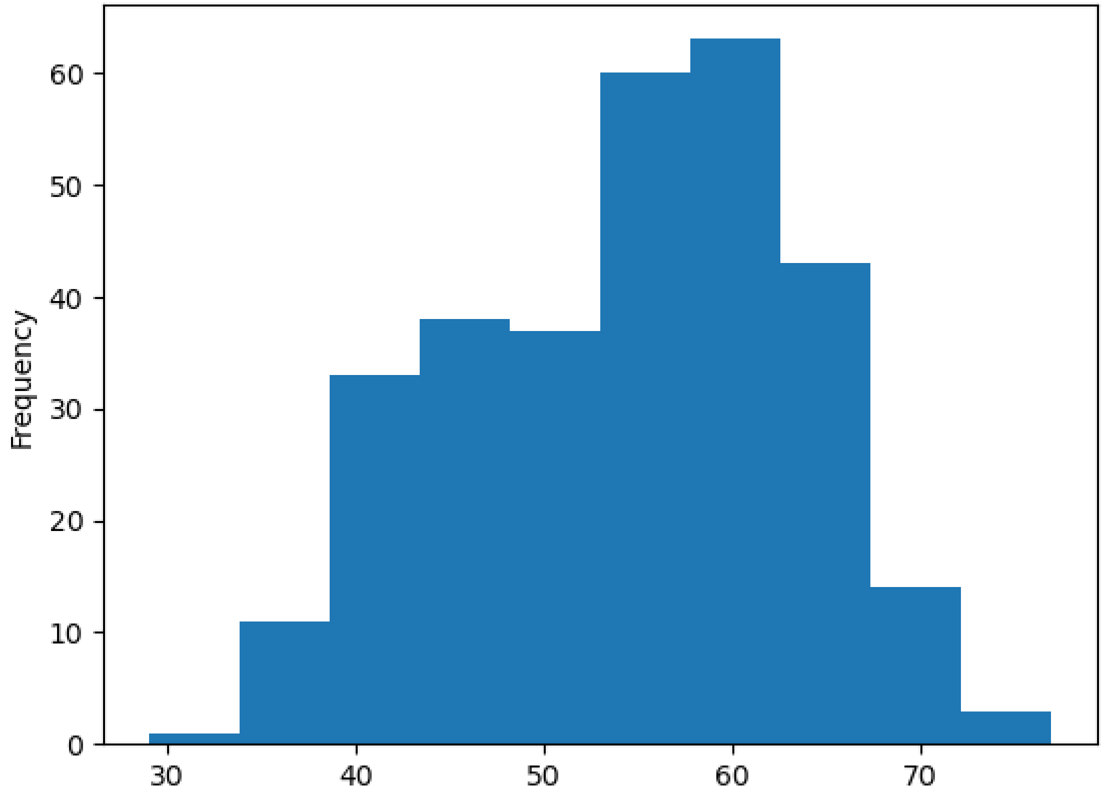
In [13]:

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

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

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

#Add some helpful info
plt.title("Heart Disease in funtion of age and max heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease", "No Disease"]);
```



In [14]:

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



Heart Disease frequency per 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 heat related)
- 3: Asymptomatic: chest pain not showing signs of disease:

In [15]:

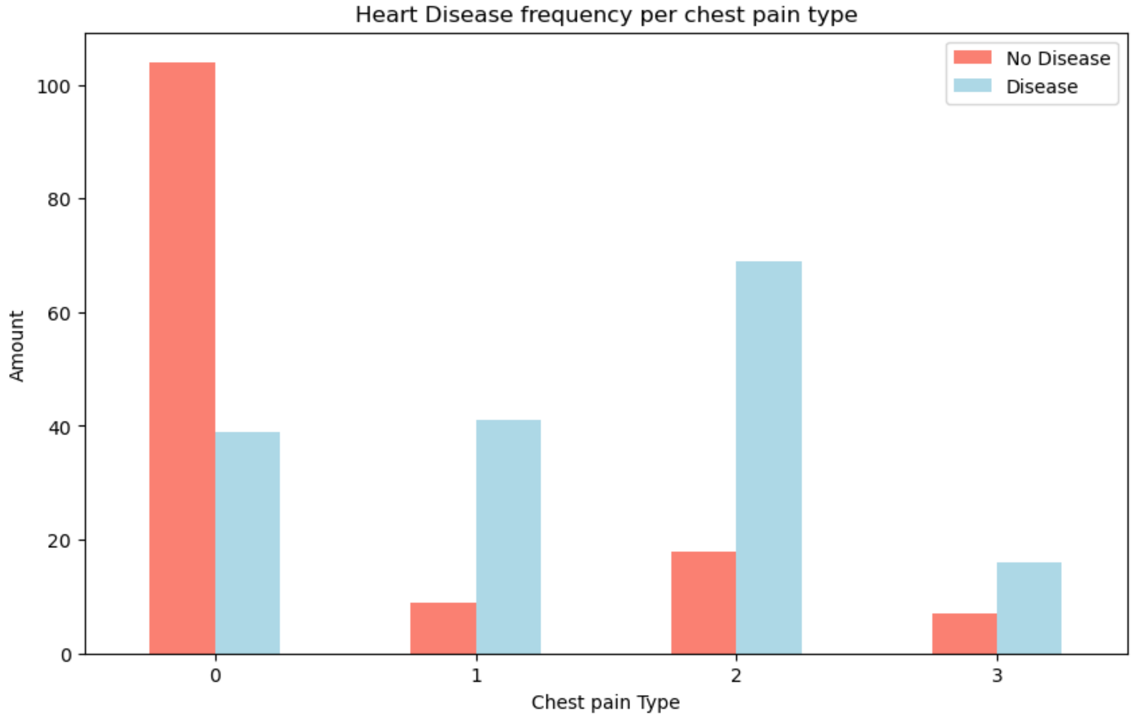
```
pd.crosstab(df.cp, df.target)
```

Out[15]:

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

In [16]:

```
# 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);
```



In [17]:

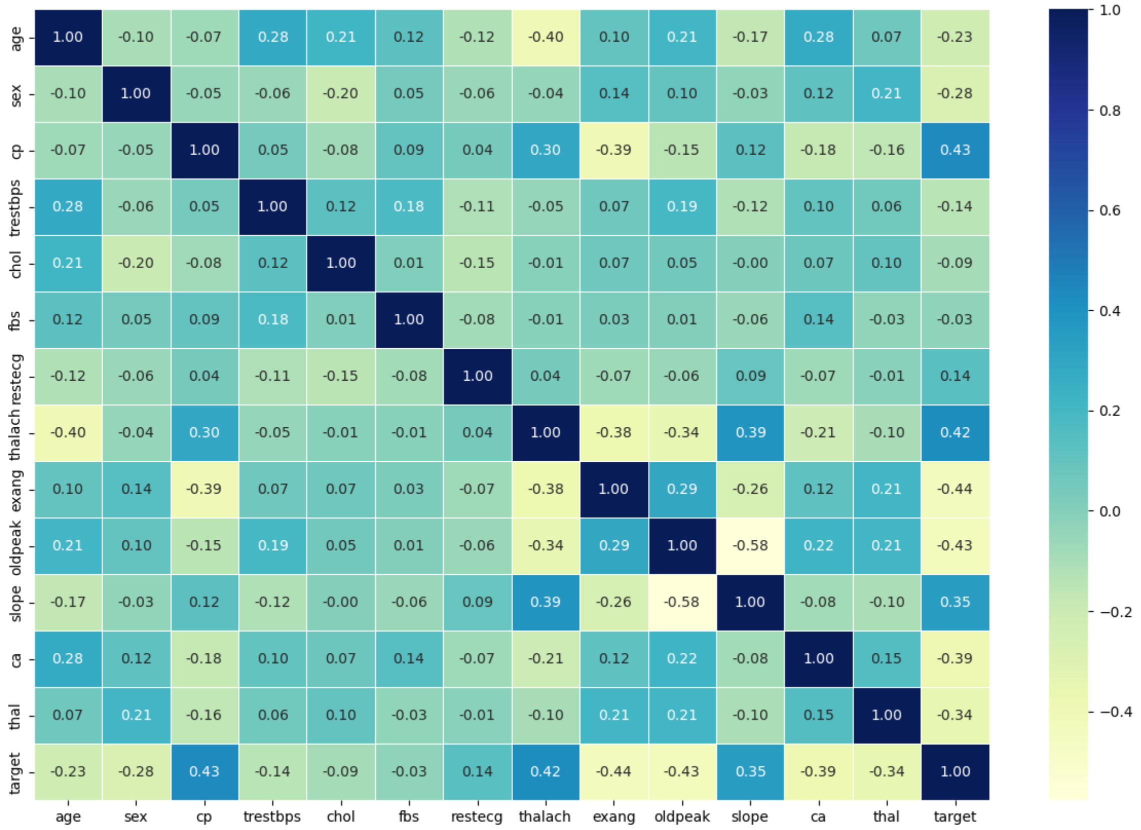
```
# Make correlation matrix
df.corr()
```

Out[17]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalac
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.39852
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.04402
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.29576
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.04669
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.00994
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.00856
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.04412
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.00000
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.37881
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.34418
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.38678
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.21317
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.09643
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.42174

In [18]:

```
# let's make our correlation a little 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")
```



Modelling

In [19]:

```
df.head()
```

Out[19]:

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

In [20]:

```
# split the data into x and y
x= df.drop("target", axis=1);
y = df["target"]
```

In [21]:

```
x
```

Out[21]:

	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 [22]:

```
y
```

Out[22]:

```
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 [23]:

```
# split data into train and test set
np.random.seed(42)
# split  into train and test set
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.2)
```

In [24]:

```
x_train
```

Out[24]:

	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 [25]:

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

Out[25]:

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

now we've got our data split into traning and test sets, it's time to build a machine learning model.

we'll train it(find the pattern) on traing sets

And we'll test it(use the pattern) on test set

we're going to try 3 diffrent machine learning models

1. Logistic Regression
2. K-Nearest Neighbours classifier
3. Random forest classifier

In [26]:

```
# put model in dictionary
models = {"Logistic Regression": LogisticRegression(),
          "KNM": KNeighborsClassifier(),
          "Random Forest": RandomForestClassifier()}
#create a funtion to fit and score models
def fit_and_score(models, x_train, x_test, y_train, y_test):
    """
    Fits and evaluate given machine models.
    models : a dict of diffrent sckite-Learn machine learning models
    x_train : training data (no labels)
    x_test : tasting data(no labels)
    y_train : training labels
    y_test : test labels
    """
    # set random seed
    np.random.seed(42)
    # Make a dictionary to keep model score
    model_scores ={}
    # Loop through models
    for name, model in models.items():
        # Fit the model to data
        model.fit(x_train, y_train)
        # Evaluate the model and append its score to the model scores
        model_scores[name] = model.score(x_test, y_test)
    return model_scores
```

In [27]:

```
model_scores = fit_and_score(models=models,
                             x_train=x_train,
                             x_test=x_test,
                             y_train=y_train,
                             y_test=y_test)
model_scores
```

```
C:\Users\alokr\Machine_learning\heart_disease_project\env\lib\site-package
s\sklearn\linear_model\_logistic.py:444: 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 i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-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-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(
```

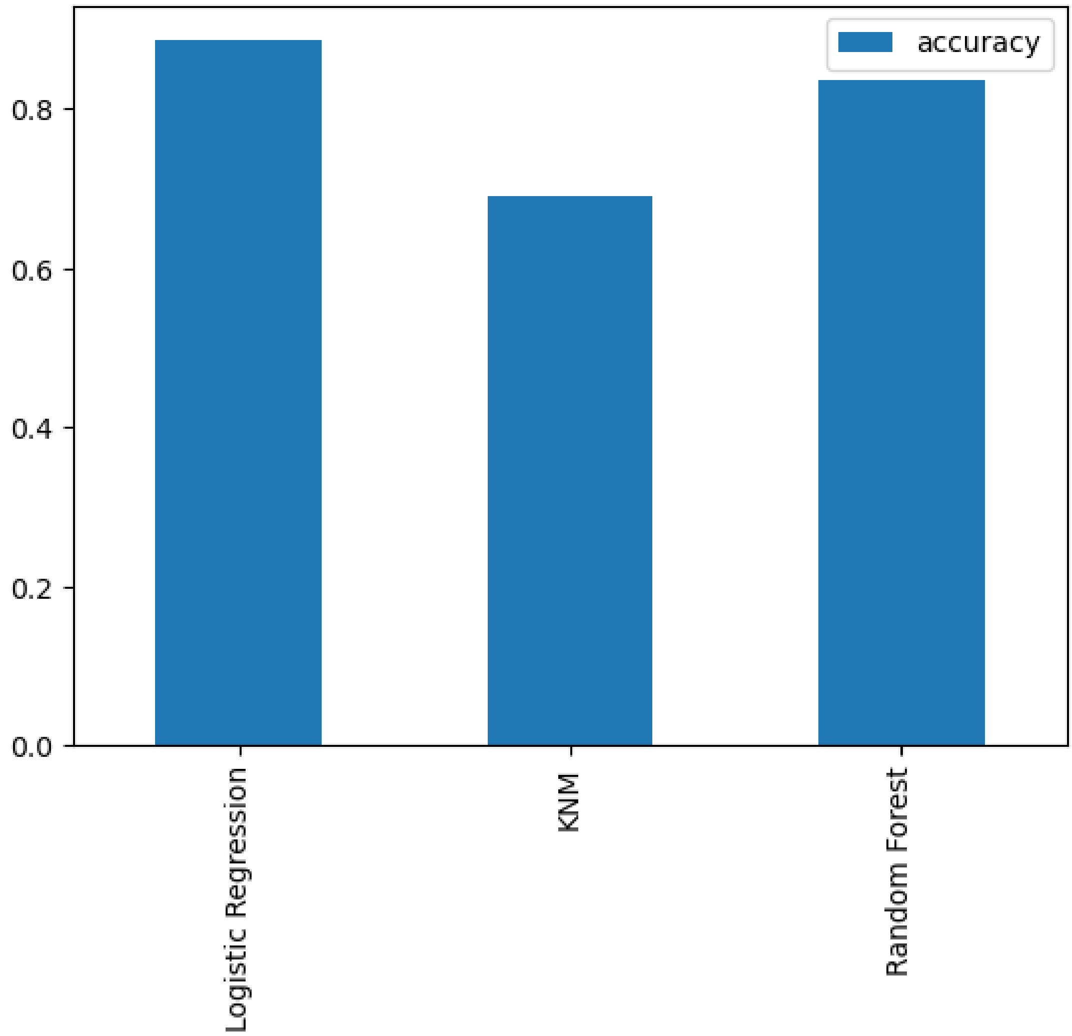
Out[27]:

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

Model Comparison

In [28]:

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



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

Let's look at the following:

* Hypyterparameter tuning
* Feature importance
* Confusion matrix
* Cross-validation
* Precision
* Recall
* F1 score
* Classification report
* ROC curve
* Area under curve

Hyperparameter tuning

In [29]:

```
# 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()

# Loops through diffrent n_neighbors
for i in neighbors:
    knn.set_params(n_neighbors=i)

    # fit the algorithm
    knn.fit(x_train, y_train)

    # Update the trainig scores list
    train_scores.append(knn.score(x_train, y_train))

    # update the test scores list
    test_scores.append(knn.score(x_test, y_test))
```

In [30]:

```
train_scores
```

Out[30]:

```
[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 [31]:

```
test_scores
```

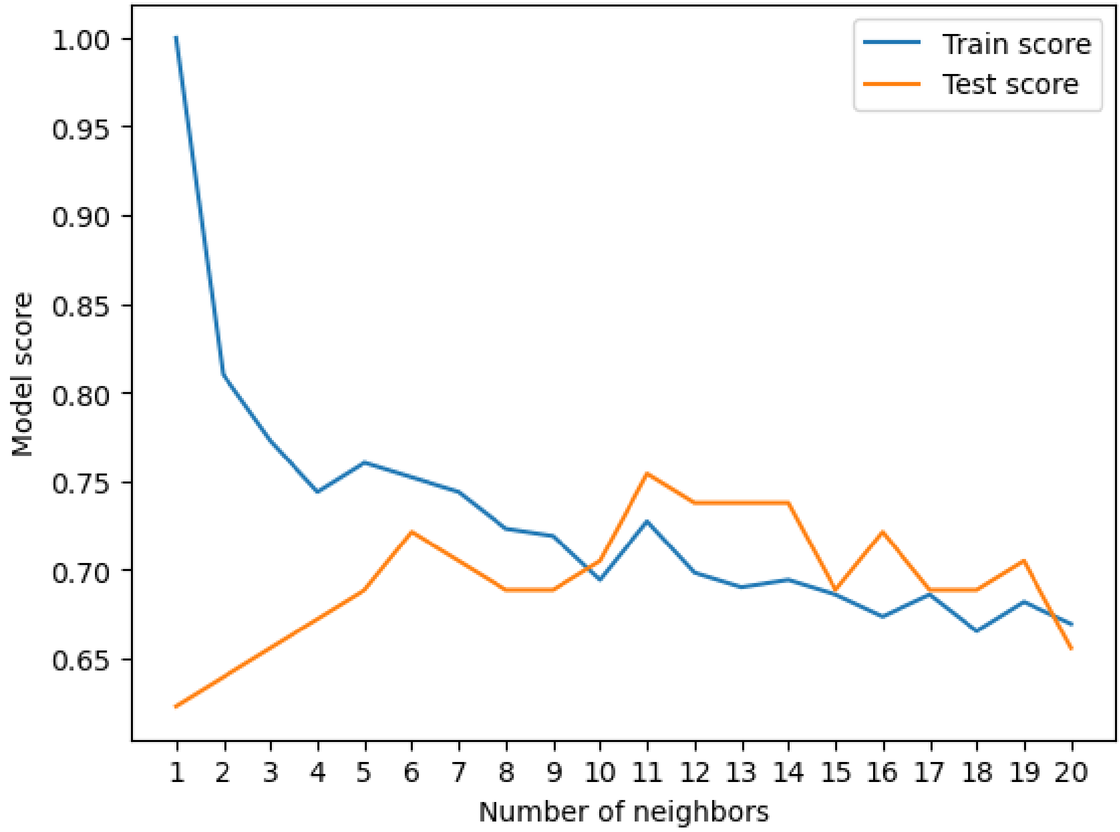
Out[31]:

```
[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 [32]:

```
plt.plot(neighbors, train_scores, label="Train score")
plt.plot(neighbors, test_scores, label="Test score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()
print(f"Maximum KNN score on the test data: {max(test_scores)*100:.2f}%")
```

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



Hyperparameter tuning with RandomizedSearchCV

we'r going to tune:

- LogisticRegression()
- RandomForestClassifier()

using RandomizedSearchCV

In [33]:

```
# create a hyperparameter grid for LogisticRegression
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 using RandomizedSearchCV

In [34]:

```
# Tune LongisticRegression
np.random.seed(42)

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

#fit random hyperparameter search model for logisticRegression
rs_log_reg.fit(x_train, y_train)
```

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

Out[34]:

```
▸ RandomizedSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [35]:

```
rs_log_reg.best_params_
```

Out[35]:

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

In [36]:

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

Out[36]:

```
0.8852459016393442
```

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

In [37]:

```
# setup random seed
np.random.seed(42)

#setup random hyperparameter search for RandomforestClassifier
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                           param_distributions=rf_grid,
                           cv=5,
                           n_iter=20,
                           verbose=True)

#fit random hyperparameter search model
rs_rf.fit(x_train, y_train)
```

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

Out[37]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [38]:

```
rs_rf.best_params_
```

Out[38]:

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

In [39]:

```
# evaluate the randomized search RandomforestClassifier model
rs_rf.score(x_test, y_test)
```

Out[39]:

```
0.8688524590163934
```

In [40]:

```
model_scores
```

Out[40]:

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

Hyperparameter tuning with GridSearchCV

since our LogisticRegression model provide the best scores so far we'll try to imporve then using gridSearchCV

In [41]:

```
# Diiffrent hyperParameter for our LogisticRegression model
log_reg_grid = {"C": np.logspace(-4, 4, 30),
                "solver": ["liblinear"]}
#setup grid hyperparameter search for LogisticRegression
gs_log_reg = GridSearchCV(LogisticRegression(),
                          param_grid=log_reg_grid,
                          cv=5,
                          verbose=True)

#fit grid hyperparameter search model
gs_log_reg.fit(x_train, y_train);
```

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

In [42]:

```
gs_log_reg.best_params_
```

Out[42]:

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

In [43]:

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

Out[43]:

```
0.8852459016393442
```

Evaluating our tuned machine learning Classifier, beyond accuracy

- Roc curve and auc curve
- Confusion Matrix
- classificatin report
- precision
- recall
- f1 score and cross-validation where possible

In [44]:

```
# Make prediciton with tuned model
y_preds =gs_log_reg.predict(x_test)
```

In [45]:

```
y_test
```

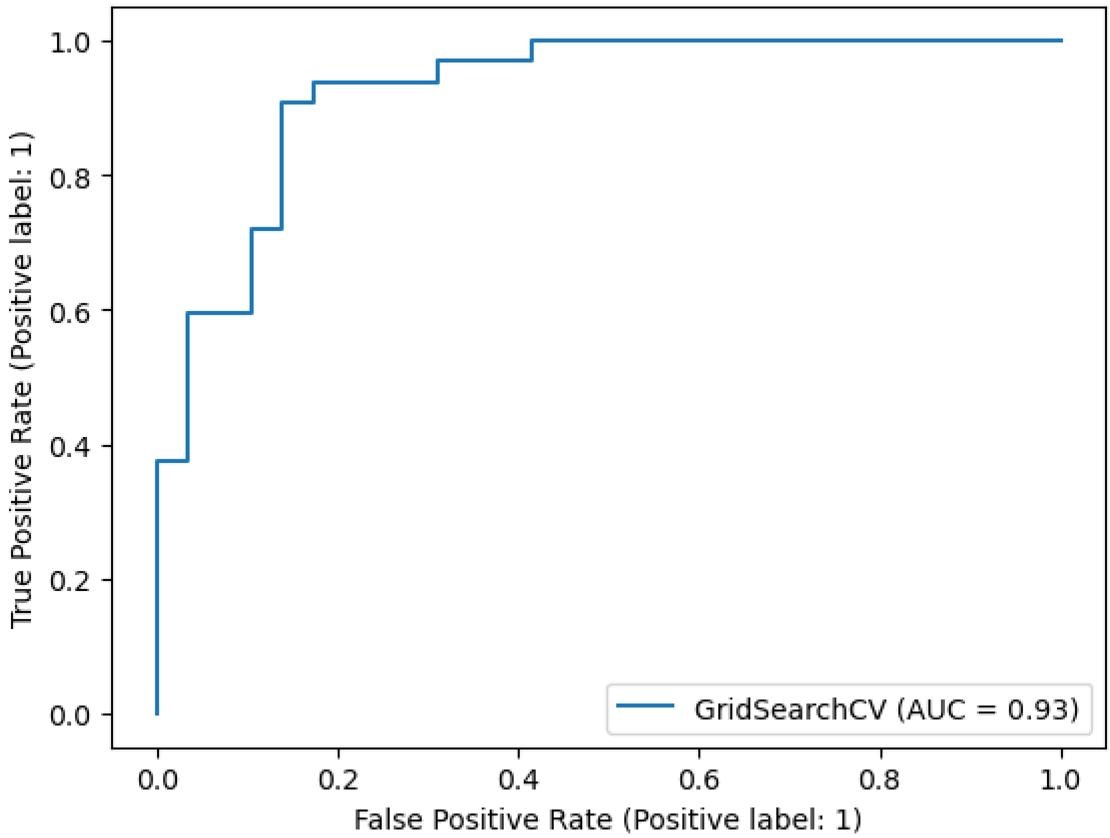
Out[45]:

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

In [46]:

```
# plot ROC curve and calculate and calculate AUC metric
plot_roc_curve(gs_log_reg, x_test, y_test);
```

```
C:\Users\alokr\Machine_learning\heart_disease_project\env\lib\site-package
s\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve
is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and wi
ll be removed in 1.2. Use one of the class methods: :meth:`sklearn.metric
s.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisp
lay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



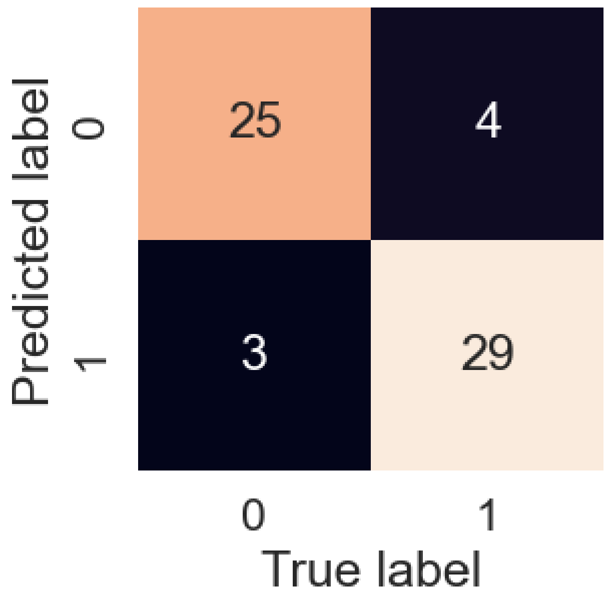
In [47]:

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

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

In [48]:

```
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("True label")
    plt.ylabel("Predicted 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 [49]:

```
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61
```

Calculate evaluation metrics using cross-validation

We're going to calculate accuracy precision, recall and 11-score of our model using cross-validation and to do so we'll be using cross_val_score().

In [50]:

```
#check best hyperparameters
gs_log_reg.best_params_
```

Out[50]:

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

In [51]:

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

In [52]:

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

Out[52]:

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

In [53]:

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

Out[53]:

```
0.8446994535519124
```

In [54]:

```
# cross validated precision
cv_precision = cross_val_score(clf,
                               x,
                               y,
                               scoring="precision")
cv_precision=np.mean(cv_precision)
cv_precision
```

Out[54]:

```
0.8207936507936507
```

In [55]:

```
# cross validation recall
cv_recall = cross_val_score(clf,
                            x,
                            y,
                            scoring="recall")
cv_recall=np.mean(cv_recall)
cv_recall
```

Out[55]:

```
0.9212121212121213
```

In [56]:

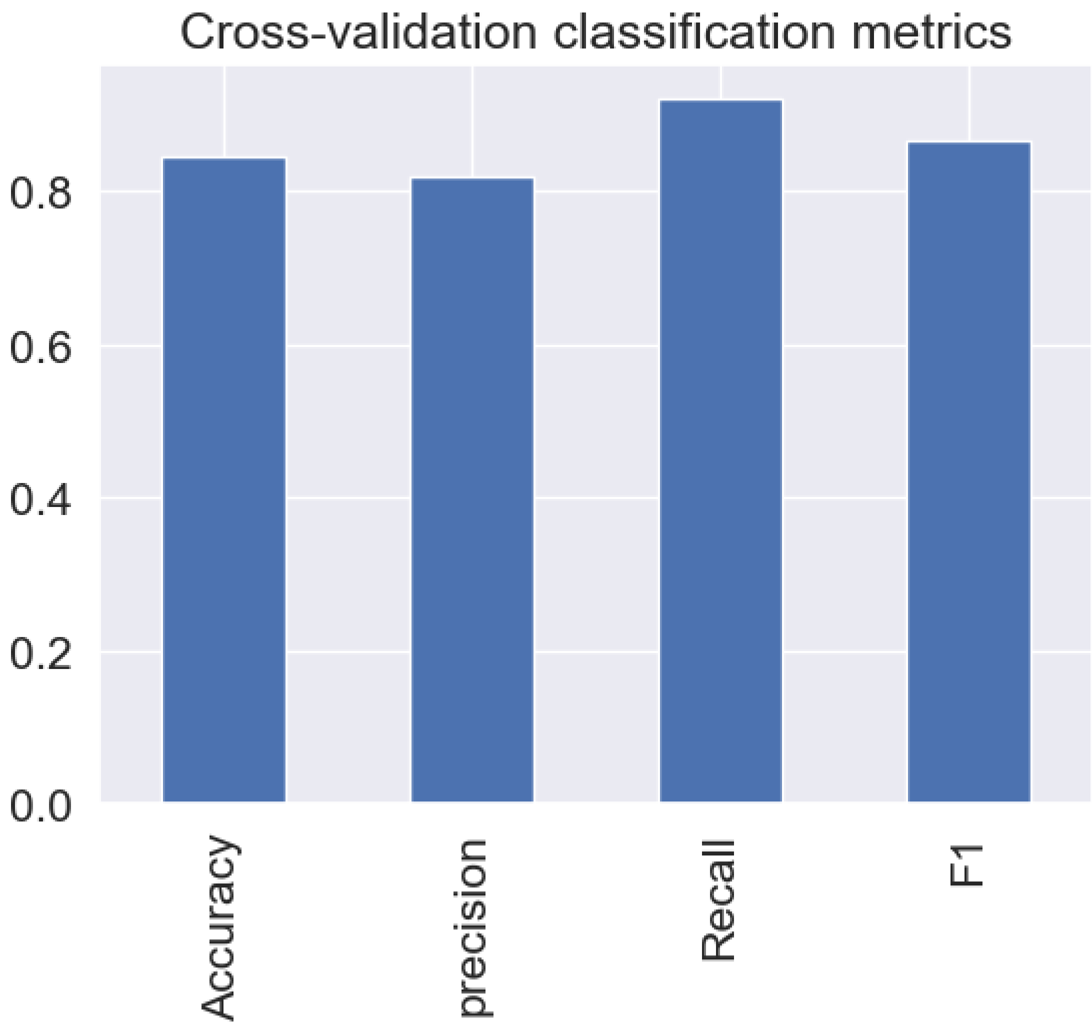
```
# cross validation f1 score
cv_f1 = cross_val_score(clf,
                        x,
                        y,
                        scoring="f1")
cv_f1=np.mean(cv_f1)
cv_f1
```

Out[56]:

```
0.8673007976269721
```

In [57]:

```
# visulize cross validation 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-validation classification metrics",
                      legend=False);
```



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 [58]:

```
# fit an instance of LogisticRegression

clf = LogisticRegression(C=0.20433597178569418,
                         solver="liblinear")
clf.fit(x_train, y_train);
```

In [59]:

```
#chek coef
clf.coef_
```

Out[59]:

```
array([[ 0.00316728, -0.86044652,  0.6606704 , -0.01156993, -0.00166375,
         0.04386107,  0.31275848,  0.02459362, -0.60413081, -0.56862803,
         0.45051628, -0.63609898, -0.67663373]])
```

In [60]:

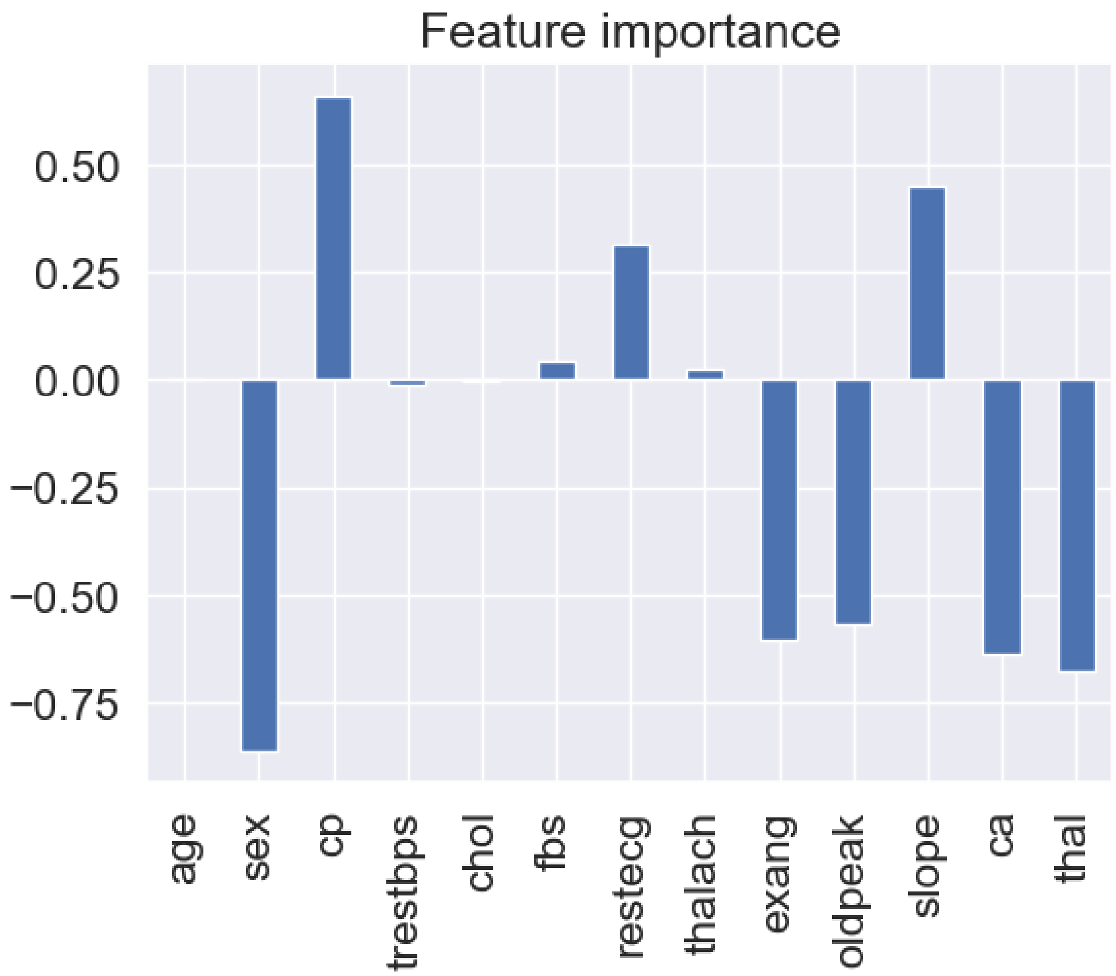
```
# match coef's of features to columns
feature_dict = dict(zip(df.columns, list(clf.coef_[0])))
feature_dict
```

Out[60]:

```
{'age': 0.0031672806268220445,
 'sex': -0.8604465226286001,
 'cp': 0.6606703996492814,
 'trestbps': -0.011569930743501303,
 'chol': -0.0016637458335408806,
 'fbs': 0.043861067871676124,
 'restecg': 0.3127584791782968,
 'thalach': 0.02459361509185037,
 'exang': -0.6041308102637141,
 'oldpeak': -0.5686280255489925,
 'slope': 0.4505162810238786,
 'ca': -0.6360989756865822,
 'thal': -0.67663372723561}
```

In [61]:

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



In [62]:

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

Out[62]:

target	0	1
sex		
0	24	72
1	114	93

In [63]:

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

Out[63]:

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

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

In [ ]:

```

```