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. mentation

We're going to take the following approach:

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

1. Problem Definition

In a statement,

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

2. Data

The original data came from the Cleavland data from the UCI Machine Learning Repository.
https://archive.ics.uci.edu/ml/datasets/heart+Disease

There is also a version of it available on Kaggle. https://www.kaggle.com/datasets/sumaiyatasmeem/heart-disease-classification-dataset

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.

4. Features

This is where you'll get different information about each of the features in your data. You can do this via doing your own research (such as looking at the links above) or by talking to a subject matter expert (someone who knows about the dataset)

. Create a Dictionary

1. age - age in years
2. sex - (1 = male; 0 = female)
3. cp - chest pain type

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

4. trestbps - resting blood pressure (in mm Hg on admission to the hospital) anything above 130-140 is typically cause for concern
5. chol - serum cholestoral in mg/dl

- serum = LDL + HDL + .2*triglycerides
- above 200 is cause for concern

6. fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

- '>126' mg/dL signals diabetes

7. restecg - resting electrocardiographic results

- 0: Nothing to note
- 1: ST-T Wave abnormality
  - can range from mild symptoms to severe problems
  - signals non-normal heart beat
- 2: Possible or definite left ventricular hypertrophy
  - Enlarged heart's main pumping chamber

8. thalach - maximum heart rate achieved
9. exang - exercise induced angina (1 = yes; 0 = no)
10. oldpeak - ST depression induced by exercise relative to rest looks at stress of heart during excercise unhealthy heart will stress more
11. 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

12. ca - number of major vessels (0-3) colored by flourosopy

- colored vessel means the doctor can see the blood passing through
- the more blood movement the better (no clots)

13. thal - thalium stress result

- 1,3: normal
- 6: fixed defect: used to be defect but ok now
- 7: reversable defect: no proper blood movement when excercising

14. target - have disease or not (1=yes, 0=no) (= the predicted attribute)

Preparing the tools

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

In [1]:
```
# Import all the 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
```

In [2]:
```
# Models from Scikit-Learn
from sklearn.model_selection import train_test_split,cross_val_score,RandomizedSearchCV,GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

# Model Evaluations
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report,precision_score,recall_score,f1_score
from sklearn.metrics import RocCurveDisplay
```

Load the data

In [3]:
```
df=pd.read_csv("heart-disease.csv")
```

Data Exploration (exploratory data analysis or EDA)

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

1. What question(s) are you 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 you deal with it?
4. Where are the outliers and why should you care about them?
5. How can you add, change or remove features to get more out of your data?

In [4]:
```
df.head()
```

Out[4]:

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

In [5]:
```
df.tail()
```

Out[5]:

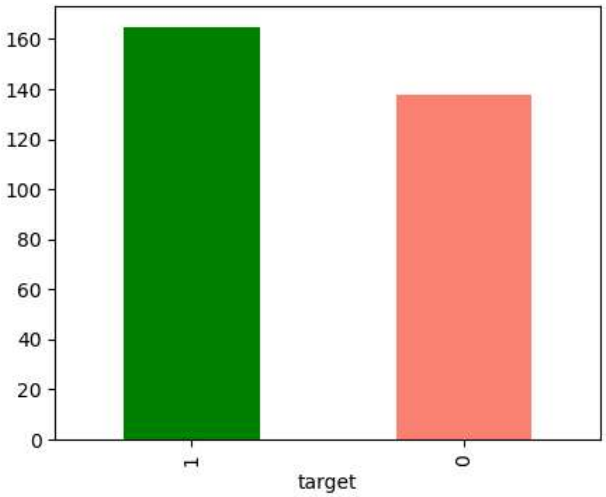
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

In [6]:
```
df["target"].value_counts()
```

Out[6]:
```
target
1    165
0    138
Name: count, dtype: int64
```

Plotting the data

In [7]:
```
df.target.value_counts().plot(kind="bar",color=["green","salmon"],figsize=(5,4));
```



In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

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

Out[9]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.3
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.6
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.0
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.0
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.0
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.0

In [10]:
```
# Are there any missing values?
df.isna().sum()
```

Out[10]:
```
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
```

Heart Disease Frequency according to Sex

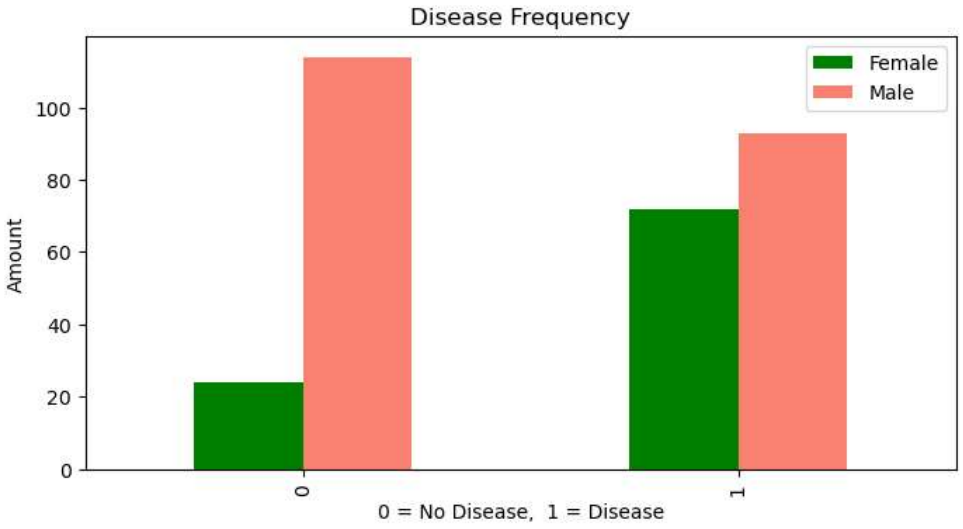
In [11]:
```
df.sex.value_counts()
```

Out[11]:
```
sex
1    207
0     96
Name: count, dtype: int64
```

In [12]:
```
pd.crosstab(df.target,df.sex).plot(kind="bar",color=["green","salmon"],figsize=(8,4));

plt.title("Disease Frequency")
plt.xlabel("0 = No Disease,  1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female","Male"])
```

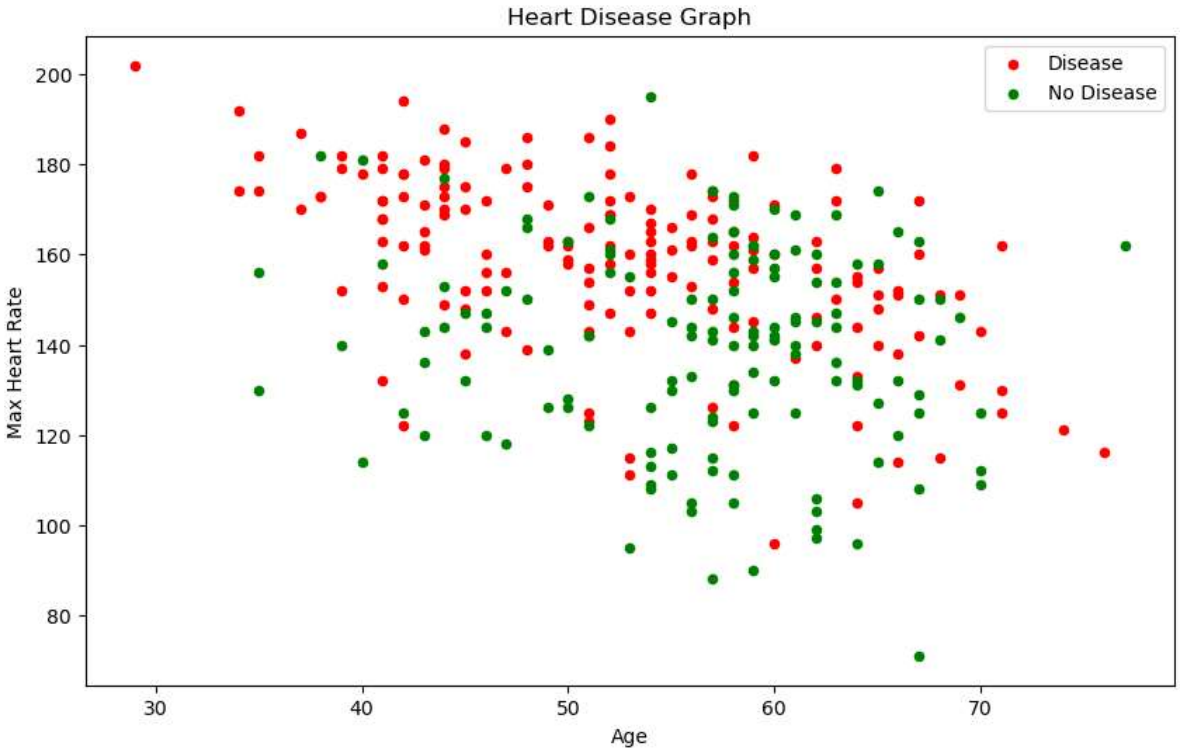
Out[12]:
```
<matplotlib.legend.Legend at 0x22b2a5a2e90>
```



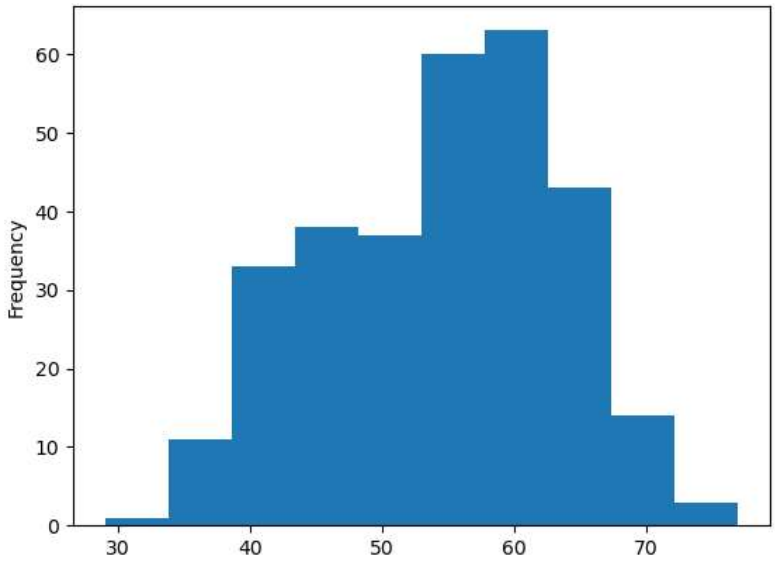
Age vs. Max Heart Rate for Heart Disease

In [13]:
```
# Create another figure
plt.figure(figsize=(10,6));
# Scatter with postivie examples
```

```
plt.scatter(df.age[df.target==1],df.thalach[df.target==1],20,c="red");
# Scatter with negative examples
plt.scatter(df.age[df.target==0],df.thalach[df.target==0],20,c="green");
# Add some helpful info
plt.title("Heart Disease Graph")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease","No Disease"]);
```



In [14]:
```
# Check the distribution of the age column with a histogram
df.age.plot.hist();
```



Heart Disease Frequency per Chest pain

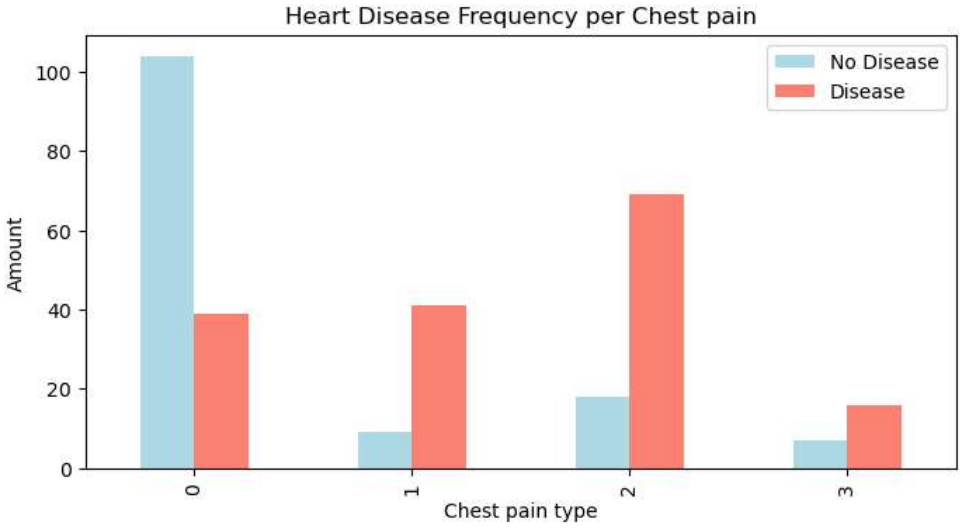
3. cp - chest pain type
   - 0: Typical angina: chest pain related decrease blood supply to the hea * rt
   - 1: Atypical angina: chest pain not related to he * art
   - 2: Non-anginal pain: typically esophageal spasms (non heart rela * ted)
   - 3: Asymptomatic: chest pain not showing signs of di4. sease

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",color=["lightblue","salmon"],figsize=(8,4));
# Add some communication
plt.title("Heart Disease Frequency per Chest pain")
plt.xlabel("Chest pain type")
plt.ylabel("Amount")
plt.legend(["No Disease","Disease"]);
```

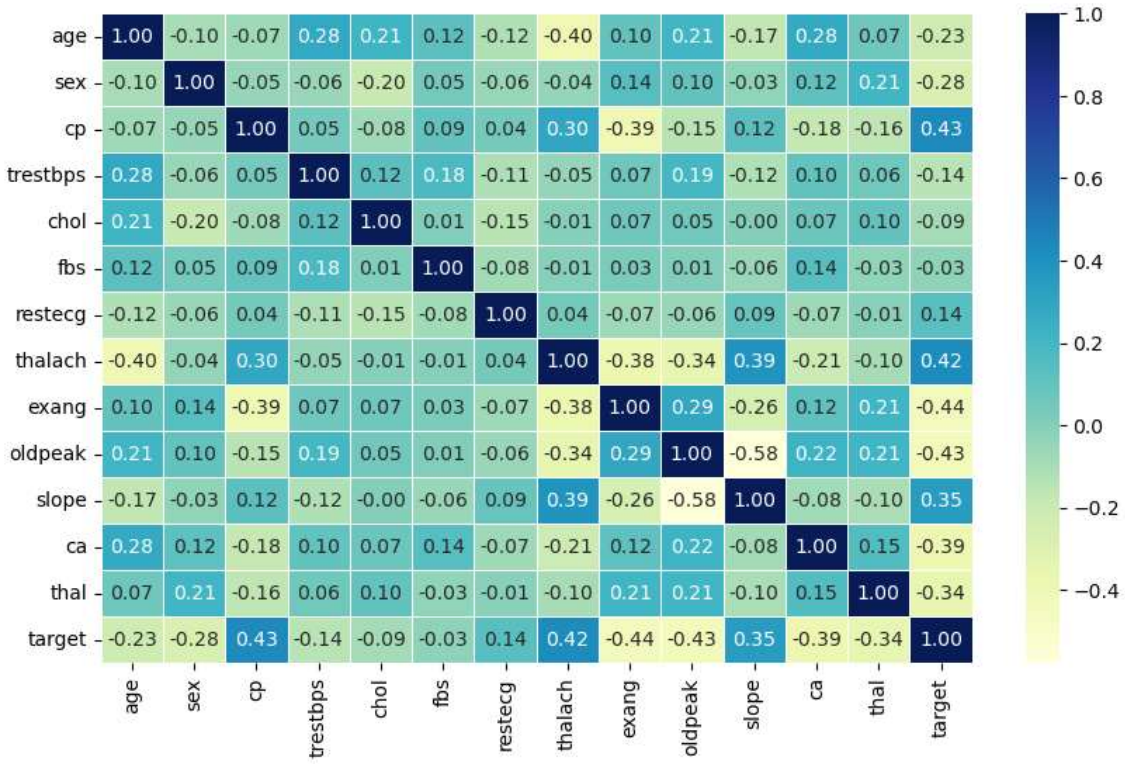


In [17]:
```
df.corr()
```

Out[17]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.391

In [18]:
```
# Let's make our correlation matrix a little prettier
corr_mat=df.corr()
fig,ax=plt.subplots(figsize=(10,6))
ax=sns.heatmap(corr_mat,
               annot=True,
               linewidths=0.5,
               fmt=".2f",
               cmap="YlGnBu");
```



5. Modelling

In [19]:
```
#splitting data into X and y
X=df.drop("target",axis=1)
y=df["target"]
```

In [20]:
```
X.head()
```

Out[20]:

	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

In [21]:
```
y
```

Out[21]:
```
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
```

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

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

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

We're going to try 3 different machine learning models:

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

In [22]:
```
#splitting data in train and test dataset
np.random.seed(42)
```

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

In [23]:
```
models={"Logical Regression":LogisticRegression(),
        "KNN":KNeighborsClassifier(),
        "Random Forest":RandomForestClassifier()}
def fit_and_score(models,X_train,X_test,y_train,y_test):
    np.random.seed(42)
    model_score={}
    for name,model in models.items():
        model.fit(X_train,y_train)
        model_score[name]=model.score(X_test,y_test)
    return model_score
```

In [24]:
```
model_score=fit_and_score(models=models,
                          X_train=X_train,
                          X_test=X_test,
                          y_train=y_train,
                          y_test=y_test)
model_score
```

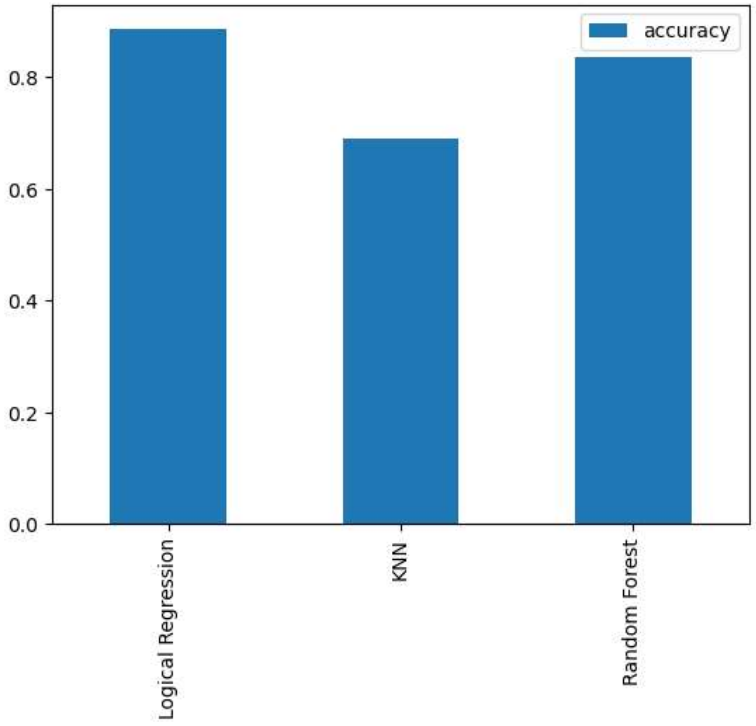
```
C:\Users\DELL\Desktop\heart_project\env\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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(
```

Out[24]:
```
{'Logical Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

Model Comparison

In [25]:
```
model_compare=pd.DataFrame(model_score,index=["accuracy"])
model_compare.T.plot.bar();
```



Now we got a baseline model.... and we know model's First predictons are'nt always what we should based our step-off. What shoud we do? Let's ook at the following-

- Hyperparameter tuning
- Feature imporatance
- cross validation
- Precision
- F1 score
- Classification Report
- ROC Curve
- Area under Curve (AUC)

Hyperparameter Tuning

In [26]:
```
#Let's tune KNN
train_scores=[]
test_scores=[]

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

#setup KNN neighbors
knn=KNeighborsClassifier()

#Loop through diffrent neighbors values
for i in neighbors:
    knn.set_params(n_neighbors=i)
    #fit the algorithm
    knn.fit(X_train,y_train)
    #update the training score list
    train_scores.append(knn.score(X_train,y_train))
    #update the test score list
    test_scores.append(knn.score(X_test,y_test))
```

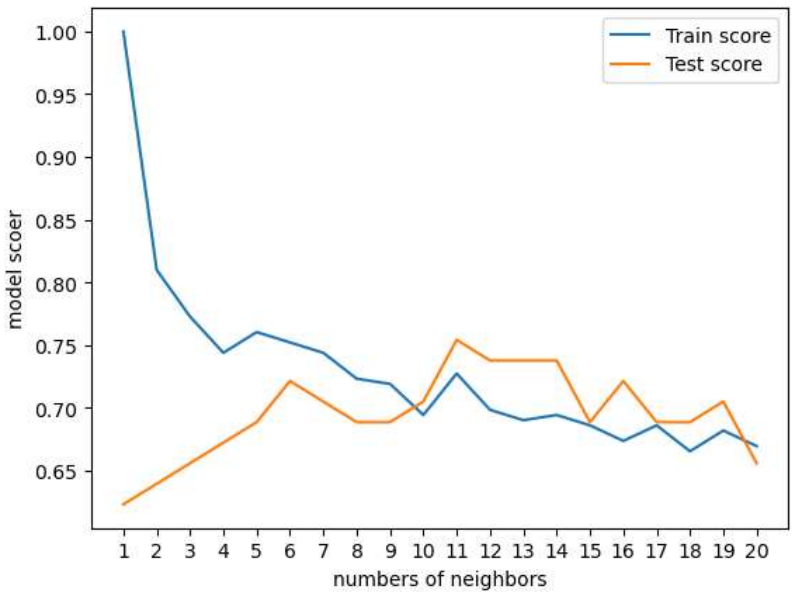
In [27]:
```
train_scores
```

Out[27]:
```
[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 [28]:
```
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("numbers of neighbors")
plt.ylabel("model scoer")
plt.legend();

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

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



Hyperparameter tuning with RandomizedSearchCV

we are going to tune with

- LogisticRegression()
- RandomForestClassifier()

....using RandomizedSearchCV

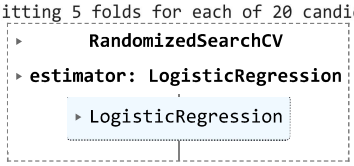
In [29]:
```
#creating hyperparameter grid for LogisticRegression()
log_grid={"C":np.logspace(-4,4,20),
          "solver":["liblinear"]}
#creating a hyperparameter grid for RandomizedForestClassifier()
rf_grid={"n_estimators":np.arange(10,1000,50),
         "max_depth":[None,3,5,10],
         "min_samples_split":np.arange(2,20,2),
         "min_samples_leaf":np.arange(1,20,2)}
```

Now we've got hyperparameter grids setup for each of our models.Lets tune them using RandomizedSearchCV().

In [30]:
```
#tune LogisticRegression()
np.random.seed(42)
#setup random hyperparameter search fore LogisticRegression
rs_log_reg=RandomizedSearchCV(LogisticRegression(),
                              param_distributions=log_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
```

Out[30]:



In [31]:
```
#checking best hyperparameter
rs_log_reg.best_params_
```

Out[31]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

In [32]:
```
rs_log_reg.score(X_test,y_test)
```

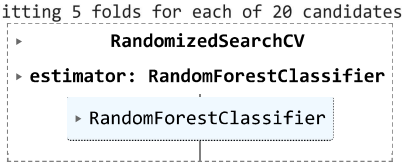
Out[32]:
```
0.8852459016393442
```

Now we have tuned LogisticRegression(). Lets's do the same for RandomForestClassifier()

In [33]:
```
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)
rs_rf.fit(X_train,y_train)
```

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

Out[33]:



In [34]:
```
#to get best hyperparameter
rs_rf.best_params_
```

Out[34]:
```
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

In [35]:
```
rs_rf.score(X_test,y_test)
```

Out[35]:
```
0.8688524590163934
```

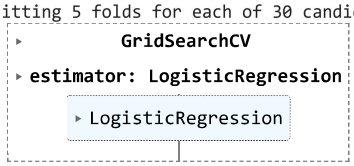
Hyperparameter tuning with GridSearchCV

Since our LogisticRegression() provides the best score so far , we will try to improve them using GridSearchCV()

In [36]:
```
# Disfferent hyperparameter for or LogisticRegression Model
log_reg_grid={'C':np.logspace(-4,4,30),
              'solver':["liblinear"]}
# Setup Grid Hyperparameter search for logistic regression
gs_log_reg=GridSearchCV(LogisticRegression(),
                        param_grid=log_reg_grid,
                        cv=5,
                        verbose=True)
#fit the hyperparameter model
gs_log_reg.fit(X_train,y_train)
```

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

Out[36]:



In [37]:
```
gs_log_reg.best_params_
```

Out[37]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

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

Out[38]:
```
0.8852459016393442
```

Evaluating our tuned machine learning classifier ,beyond accuracy

- ROC & AUC Curve
- Confusion Matrix
- Classification Report
- Precision
- Recall
- F1-score

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

To make Comparision and evaluate our training model First we need to make predictions

In [39]:
```
y_preds=gs_log_reg.predict(X_test)
```

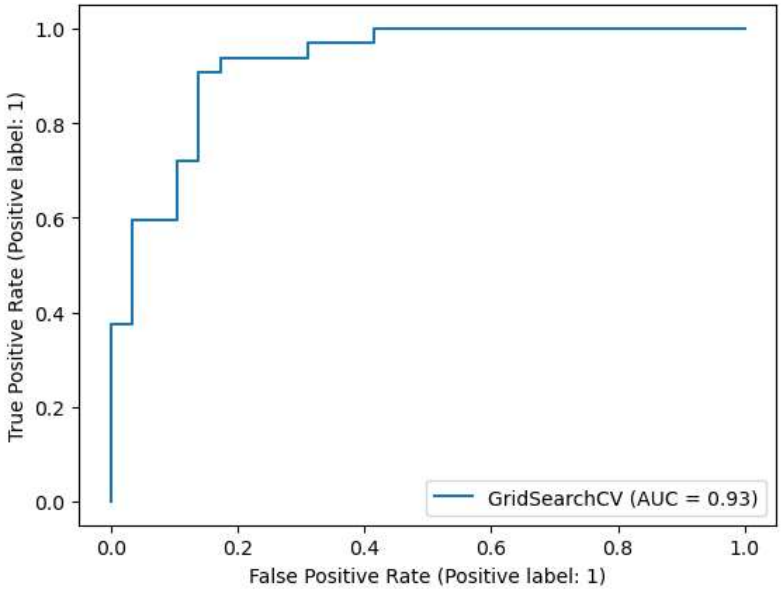
In [40]:
```
y_preds
```

Out[40]:
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [41]:
```
y_test
```

Out[41]:
```
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 [42]:
```
# plot ROC Curve and calculate the AUC metric
RocCurveDisplay.from_estimator(gs_log_reg,X_test,y_test);
```



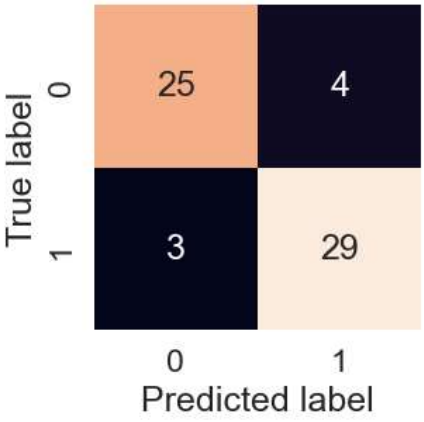
In [43]:
```
#cofusion matrix
print(confusion_matrix(y_test,y_preds))
```

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

In [44]:
```
# Increase font size
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, # 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)
```



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

In [45]:
```
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 f1-score of our model using cross-validation and to do so we'll be using cross_val_score().

In [46]:
```
#checking best hyperparameter
gs_log_reg.best_params_
```

Out[46]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [47]:
```
clf=LogisticRegression(C= 0.20433597178569418, solver= 'liblinear')
```

In [48]:
```
#cross validated accuracy
cv_acc=cross_val_score(clf,
                       X,
                       y,
                       cv=5,
                       scoring="accuracy")
cv_acc
```

Out[48]:
```
array([0.81967213, 0.90163934, 0.86885246, 0.88333333, 0.75      ])
```

In [49]:
```
cv_acc=np.mean(cv_acc)
cv_acc
```

Out[49]:
```
0.8446994535519124
```

In [50]:
```
#cross validated precision
cv_precision=cross_val_score(clf,
                       X,
                       y,
                       cv=5,
                       scoring="precision")
cv_precision=np.mean(cv_precision)
cv_precision
```

Out[50]:
```
0.8207936507936507
```

In [51]:
```
#cross validated recall
cv_recall=cross_val_score(clf,
                       X,
                       y,
                       cv=5,
                       scoring="recall")
cv_recall=np.mean(cv_recall)
cv_recall
```

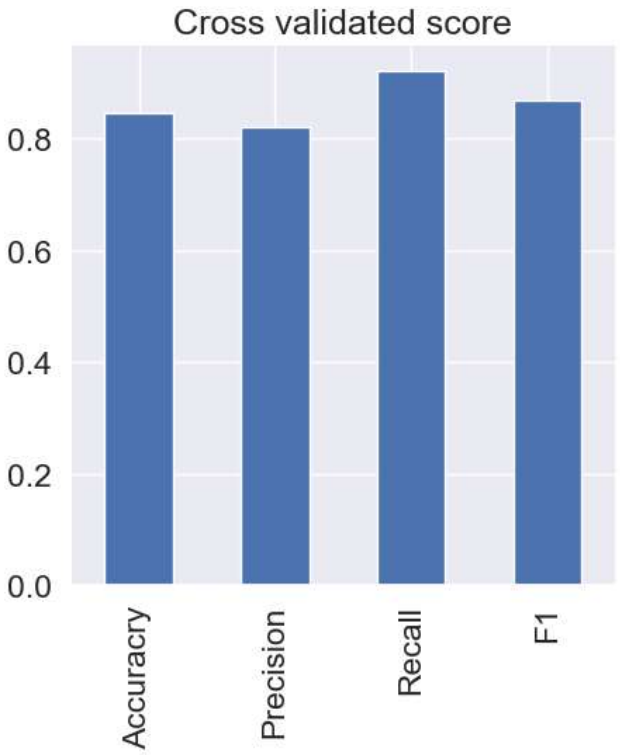
Out[51]:
```
0.9212121212121213
```

In [52]:
```
#cross validated f1
cv_f1=cross_val_score(clf,
                       X,
                       y,
                       scoring="f1",cv=5)
cv_f1=np.mean(cv_f1)
cv_f1
```

Out[52]:
```
0.8673007976269721
```

In [53]:
```
#Visualize Cross Validated Score
cv_dict=pd.DataFrame({"Accuracry":cv_acc,
                      "Precision":cv_precision,
                      "Recall":cv_recall,
                      "F1":cv_f1},index=[0])
```

In [54]:
```
cv_dict.T.plot.bar(title="Cross validated score",
                   legend=False,
                  figsize=(5,5));
```



Feature Importance

It is another way of asking,"which feature contributed most to the outcome of the model and how did they contribute?" Finding feature importance is different for each machine larning model. One way to Find Feature Importance is to search for ("MODEL NAME") Feature Importance...

Let's find the feature importance for our Logistic Refression Model...

In [55]:
```
#fit the instance of Logistic Regression
clf=LogisticRegression(C=0.20433597178569418,solver="liblinear")
clf.fit(X_train,y_train)
```

Out[55]:
```
LogisticRegression(C=0.20433597178569418, solver='liblinear')
```

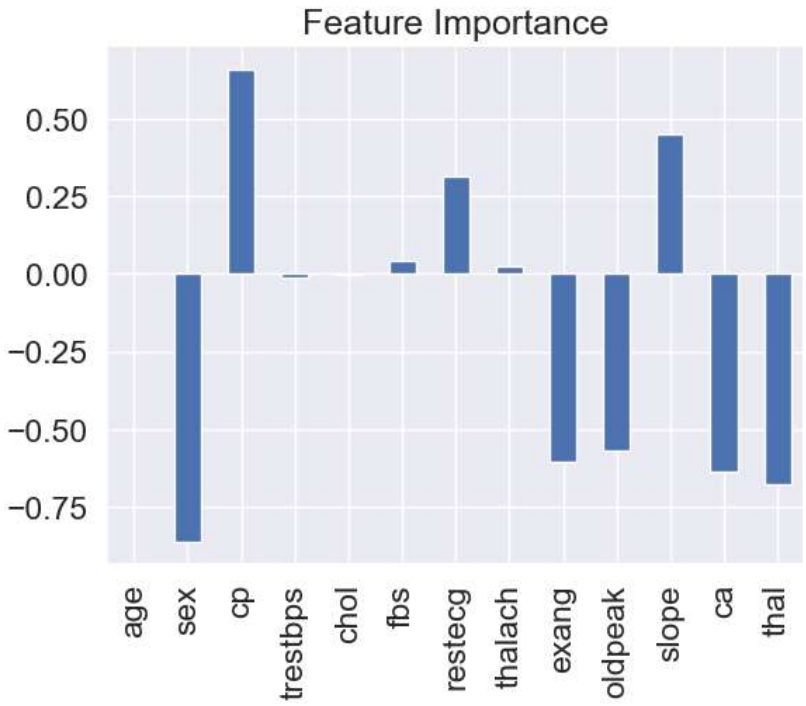
In [56]:
```
#check coef_
clf.coef_
```

Out[56]:
```
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]])
```

In [57]:
```
#Match the coef's of features to columns
feature_dict=dict(zip(df.columns,list(clf.coef_[0])))
feature_dict
```

Out[57]:
```
{'age': 0.0031672721856887734,
 'sex': -0.860445816920919,
 'cp': 0.6606707303492849,
 'trestbps': -0.011569930902919925,
 'chol': -0.001663741604035976,
 'fbs': 0.04386130751482091,
 'restecg': 0.3127578715206996,
 'thalach': 0.02459360818122666,
 'exang': -0.6041303799858143,
 'oldpeak': -0.5686285194546157,
 'slope': 0.4505161679452401,
 'ca': -0.6360986316921434,
 'thal': -0.6766337521354281}
```

In [58]:
```
#visualize the Feature importance
feature_df=pd.DataFrame(feature_dict,index=[0])
feature_df.T.plot.bar(title="Feature Importance",legend=False);
```



6. Experimentation

If we haven't hit our evaluation metric yet... we can ask ourself...

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