Predict heart disease using ML

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

1. Problem Definination

> Given clinical parameters , can we predict , heart disease or not

2. Data

The original data is from Cleaveland data from UCI Machine Learning Repositittory.
https://archive.ics.uci.edu/dataset/45/heart+disease

3. Evaluation

If we reach 95% accuracy during proof of conceot , we'll pursue.

4. Features

Only 14 attributes used:

```
1. #3  (age)
2. #4  (sex)
3. #9  (cp)
4. #10 (trestbps)
5. #12 (chol)
6. #16 (fbs)
7. #19 (restecg)
8. #32 (thalach)
9. #38 (exang)
10. #40 (oldpeak)
11. #41 (slope)
12. #44 (ca)
13. #51 (thal)
14. #58 (num)       (the predicted attribute)
```

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)

In [1]:
```
import sklearn
sklearn.__version__
```

Out[1]:
```
'1.2.1'
```

Prepping tools

We will use Pandas Matplotlib Numpy for data analysis and manipulation

In [2]:
```
#Import all tools
#Regular EDA (Explanantory Data Analysis) and plotting libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
#For making plots appear inside notebook

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

Load Data

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

Out[3]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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

303 rows × 14 columns

In [4]:
```
df.shape #(rows,columns)
```

Out[4]:
```
(303, 14)
```

Data Exploration (EDA)

1. Problem Statement
2. Data type
3. Missing data
4. Outliers : outlandish data point
5. Add , Change , Remove

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

Out[5]:

	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 [6]:
```
df.tail()
```

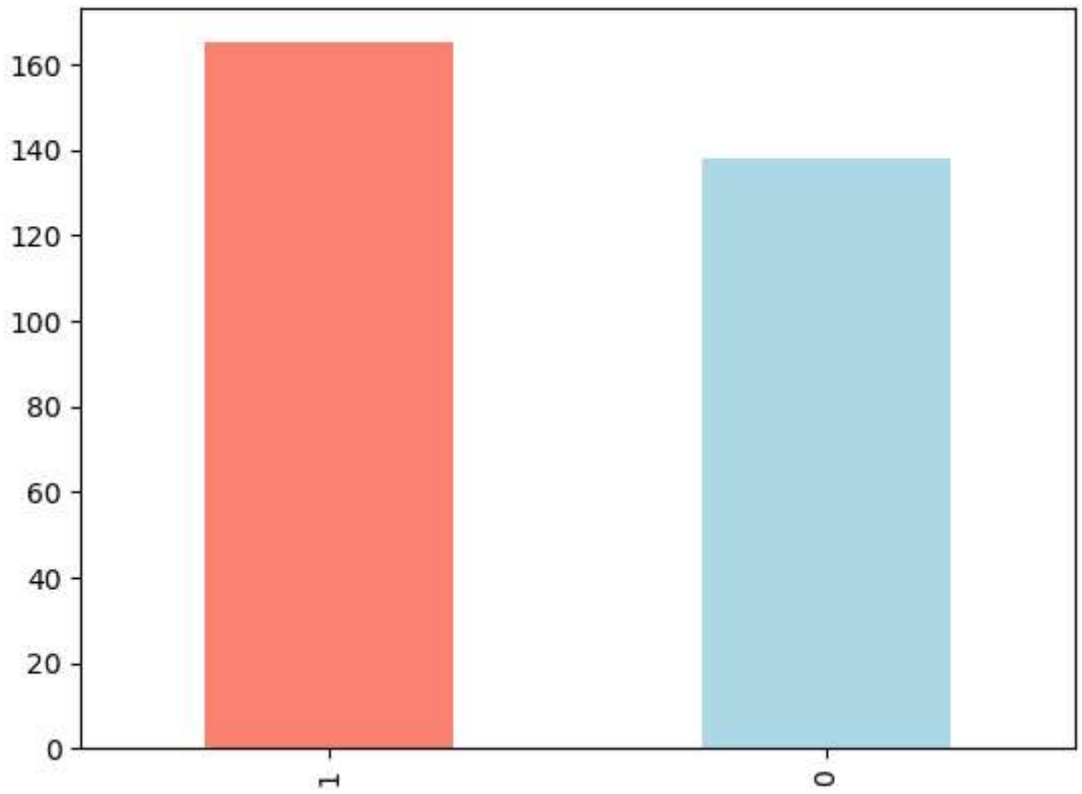
Out[6]:

	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 [7]:
```
df["target"].value_counts()  #shows no of unique data
```

Out[7]:
```
1    165
0    138
Name: target, dtype: int64
```

In [8]:
```
df["target"].value_counts().plot(kind="bar", color=["salmon","lightblue"]);
```



In [9]:
```
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 [10]:
```
df.isna().sum() #check missing data
```

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
```

In [11]:
```
df.describe() #numerical details about features
```

Out[11]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
count	303.000000	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	149.646865
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000

Heart Disease Frequency according to Sex

In [12]:
```
df.sex.value_counts()
```

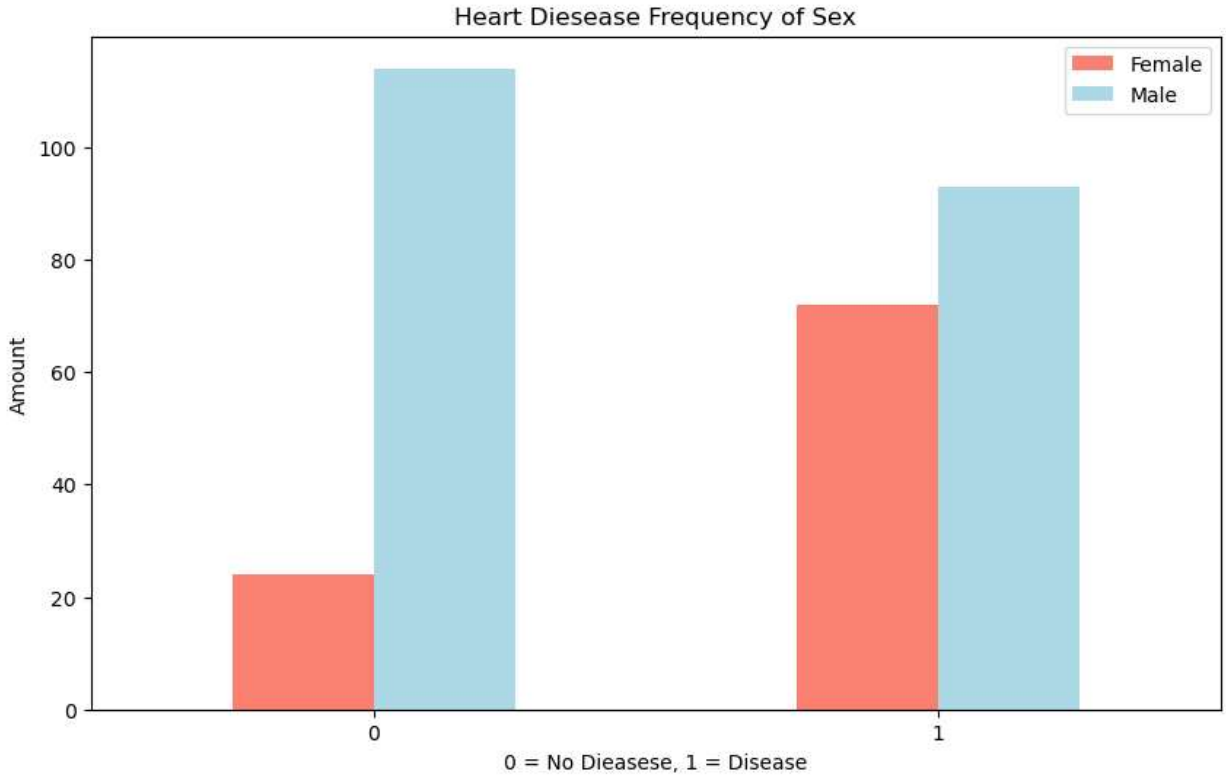
Out[12]:
```
1    207
0     96
Name: sex, dtype: int64
```

In [13]:
```
# Compare target column with sex column
pd.crosstab(df.target, df.sex)
```

Out[13]:

sex	0	1
target		
0	24	114
1	72	93

In [14]:
```
# Create a plot of crosstab
pd.crosstab(df.target,df.sex).plot(kind="bar",
                                   figsize=(10,6), ##?? TODO : why ?
                                   color=["salmon","lightblue"]);

plt.title("Heart Diesease Frequency of Sex")
plt.xlabel("0 = No Dieasese, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female","Male"]);
plt.xticks(rotation=0);
```



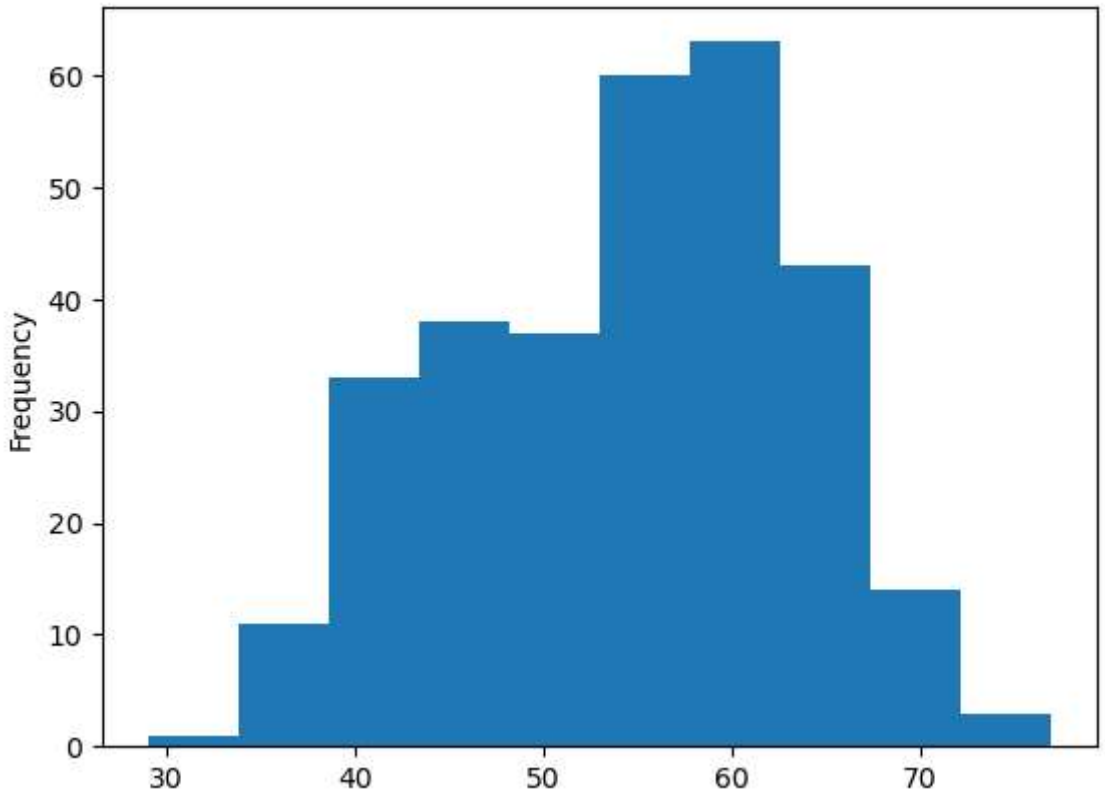
In [15]:
```
df["thalach"].value_counts()
```

Out[15]:
```
162    11
160     9
163     9
152     8
173     8
       ..
202     1
184     1
121     1
192     1
90      1
Name: thalach, Length: 91, dtype: int64
```

Age vs. Max Heart Rate for Heart Disease

In [16]:
```
#Create another figure
plt.figure(figsize=(10,6))

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

#Scatter with negetive examples
plt.scatter(df.age[df.target==0],
            df.thalach[df.target==0],
            c="lightblue");
plt.title("Heart Diesease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease","No Disease"]);
```

In [17]:
```
# Check the distribution of the age column with histogram
df.age.plot.hist();
```



Heart Disease Frequency per chest pain type

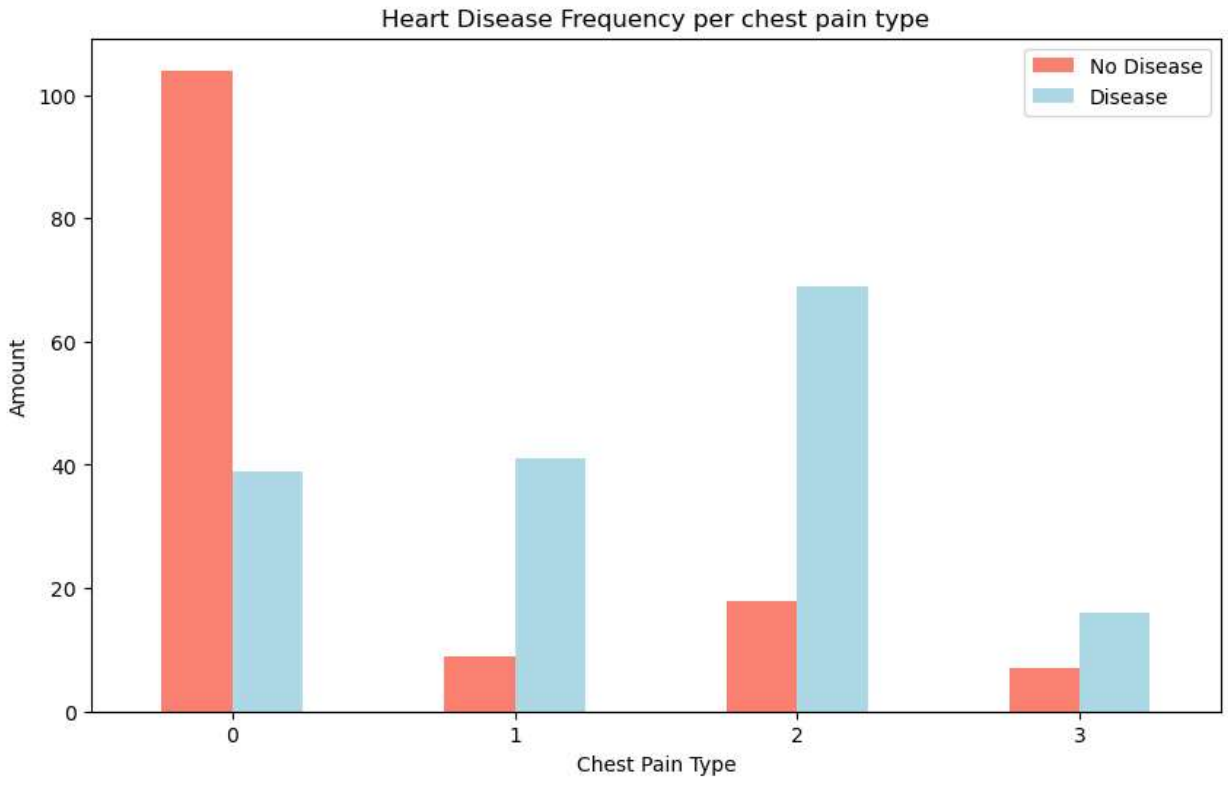
cp - chest pain type 0: Typical angina: chest pain related decrease blood supply to the heart 1: Atypical angina: chest pain not related to heart 2: Non-anginal pain: typically esophageal spasms (non heart related) 3: Asymptomatic: chest pain not showing signs of disease

In [18]:
```
pd.crosstab(df.cp,df.target)
```

Out[18]:

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

In [19]:
```
pd.crosstab(df.cp,df.target).plot(kind="bar",
                                  figsize=(10,6),
                                  color=["salmon","lightblue"])

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); #TODO what is this ?
```



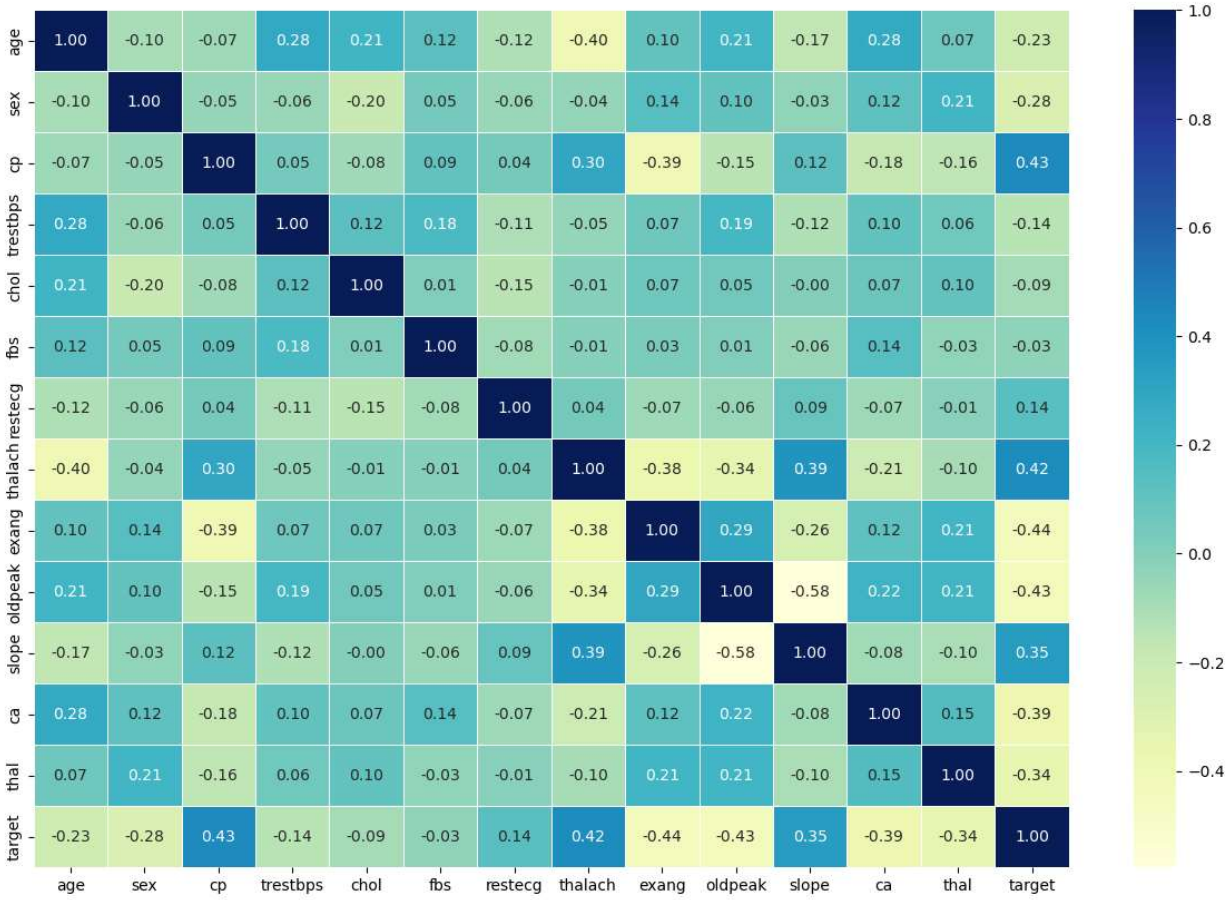
In [20]:
```
#Correlation Matrix

df.corr()
```

Out[20]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exa
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436

In [21]:

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



In [22]:
```
#Split data into X and y
X = df.drop("target", axis=1)
y = df["target"]

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)
```

We are going to try 3 diff models:

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

In [23]:
```
models = {"Logistic Regression" : LogisticRegression(),
          "KNN": KNeighborsClassifier(),
          "Random Forest" : RandomForestClassifier()}

#Create a function to fit and score models
def fit_and_acore(models, X_train, X_test, y_train, y_test):
    """
    Fits and evaluates ML models
    models : dict of Scikit Learn ML models
    X_train : training data (no labels)
    X_test : testing data (no_labels)
    y_train : traning labels
    y_test : test labels
    """
```

```
    #set random seed
    np.random.seed(42)
    #make dict to keep model scores
    model_scores = {}
    #Loop through models
    for name, model in models.items():
        #Fit model to data
        model.fit(X_train, y_train)
        #Evaluate
        model_scores[name] = model.score(X_test, y_test)
    return model_scores
```

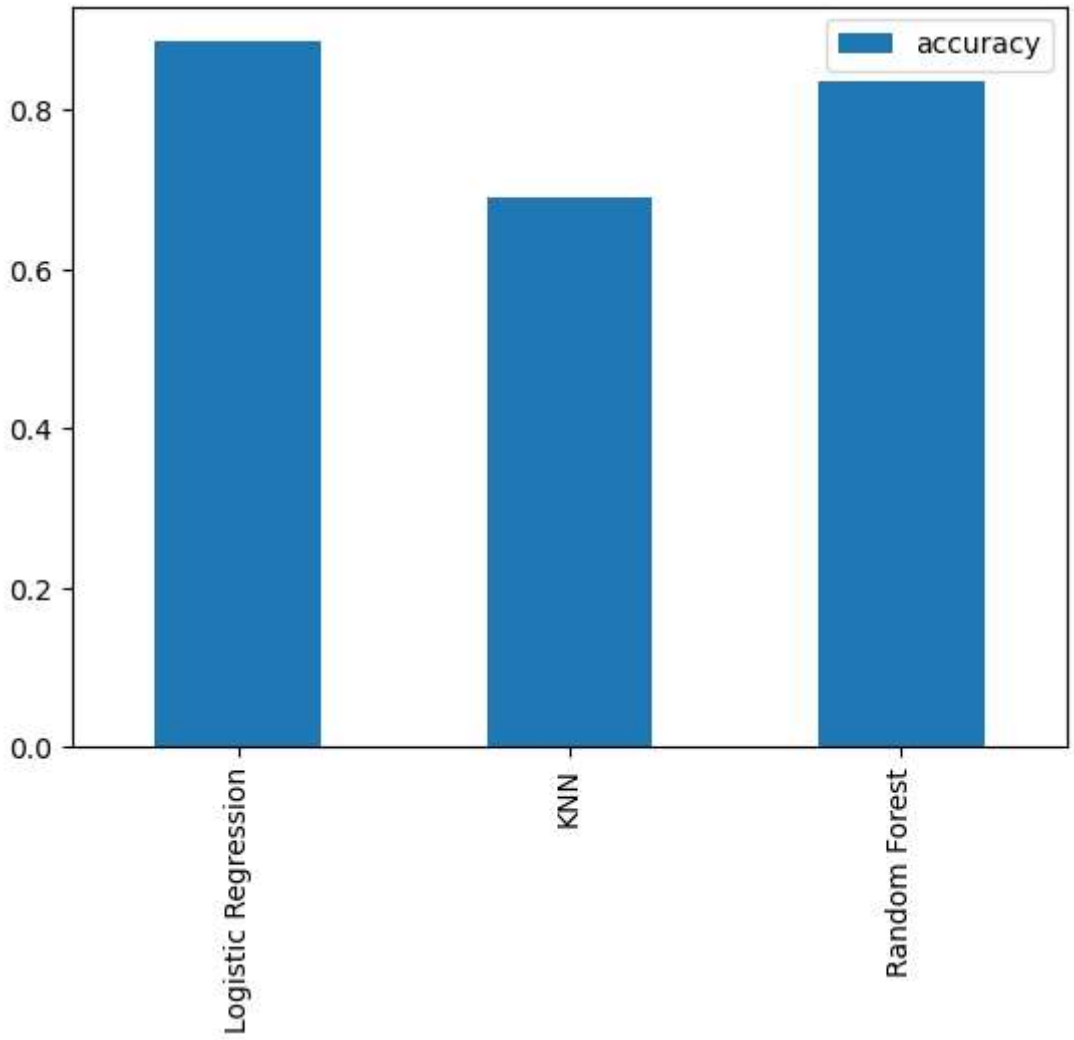
In [24]:
```
model_scores = fit_and_acore(models=models,
                             X_train= X_train,
                             X_test= X_test,
                             y_train= y_train,
                             y_test= y_test)

model_scores
```

```
C:\Users\HP\AI_bootcamp\sample_project_1\env\lib\site-packages\sklearn\linear_model\_
logistic.py:458: 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]:
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

Model Comparison

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



We'll explore the below :

- Hyperparameter tuning
- Feature importance
- Confusion Matrix
- Cross Validation
- Precision
- Recall
- F1 Score
- Classification Report
- ROC Curve
- Area Under Curve

Hyperparameter tuning

In [26]:
```
#Tune KNN

train_scores = []
test_scores = []
```

```
neighbors = range(1,21)

knn = KNeighborsClassifier()

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

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

    train_scores.append(knn.score(X_train, y_train))

    test_scores.append(knn.score(X_test, y_test))
```

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

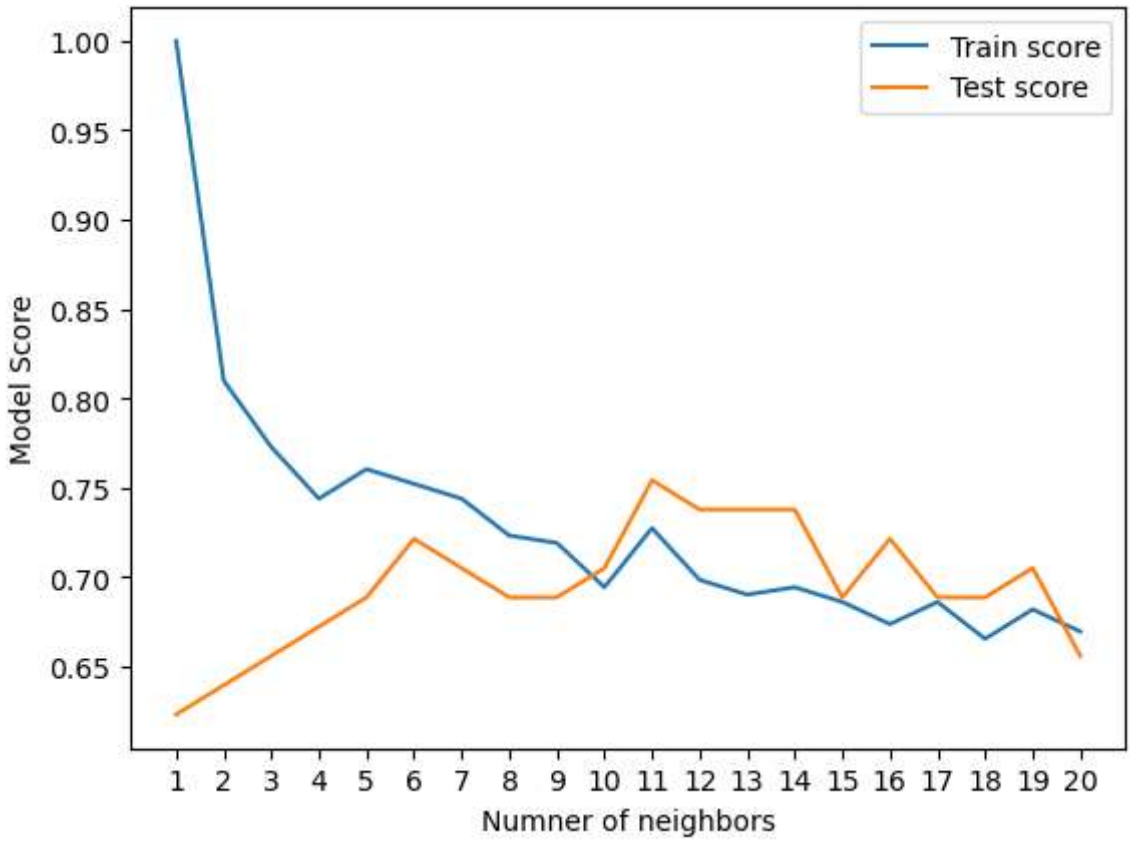
Out[27]:
```
[1.0,
 0.8099173553719008,
 0.7727272727272727,
 0.743801652892562,
 0.7603305785123967,
 0.7520661157024794,
 0.743801652892562,
 0.7231404958677686,
 0.71900826446281,
 0.6942148760330579,
 0.7272727272727273,
 0.6983471074380165,
 0.6900826446280992,
 0.6942148760330579,
 0.6859504132231405,
 0.6735537190082644,
 0.6859504132231405,
 0.6652892561983471,
 0.6818181818181818,
 0.6694214876033058]
```

In [28]:
```
test_scores
```

Out[28]:
```
[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 [29]:
```
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("Numner 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 for RandomizedSearchCV

Tuning :

LogisticRegression() RandomForestClassifier()

using RandomizedSearchCV

In [30]:
```
#For Logistic Regression

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

In [31]:
```
np.logspace(-4, 4,20)
```

Out[31]:
```
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])
```

In [32]:
```
#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)}
```

In [33]:
```
#Tune Logistic Regression Model

np.random.seed(42)

#random hyperparameter search for Logistic Regression

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

#Fit random hyperparameter sarch model for Logistic Regression

rs_log_reg.fit(X_train,y_train)
```

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

Out[33]:

▸ RandomizedSearchCV

▸ estimator: LogisticRegression

▸ LogisticRegression

In [34]:
```
rs_log_reg.best_params_
```

Out[34]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

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

Out[35]:
```
0.8852459016393442
```

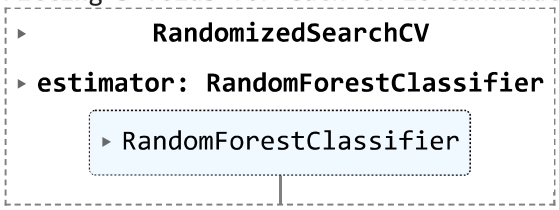
Now turning RandomForest Classifier

In [36]:
```
np.random.seed(42)

#setup random hyperarameter

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[36]:



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

Out[37]:
```
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

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

Out[38]:
```
0.8688524590163934
```

In [39]:
```
model_scores
```

Out[39]:
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest': 0.8360655737704918}
```

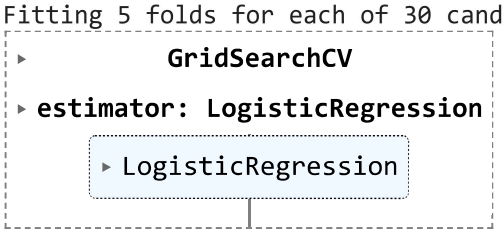
Hyperparameter Tuning with GridSearchCV

Since LogisticRegression is best here so tune it using GridSearchCV

In [40]:
```
log_reg_grid = {"C":np.logspace(-4, 4, 30),
                "solver":["liblinear"]}

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

gs_log_reg.fit(X_train,y_train)
```

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

Out[40]:



In [41]:
```
#Check best hyperparameter

gs_log_reg.best_params_
```

Out[41]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [54]:
```
#Evaluate gridsearch Logistic Regression
gs_log_reg.score(X_test, y_test)
```

Out[54]:
```
0.8852459016393442
```

Evaluating Models

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

we want to use cross-validation where ever applicable

In [42]:
```
y_preds = gs_log_reg.predict(X_test)
```

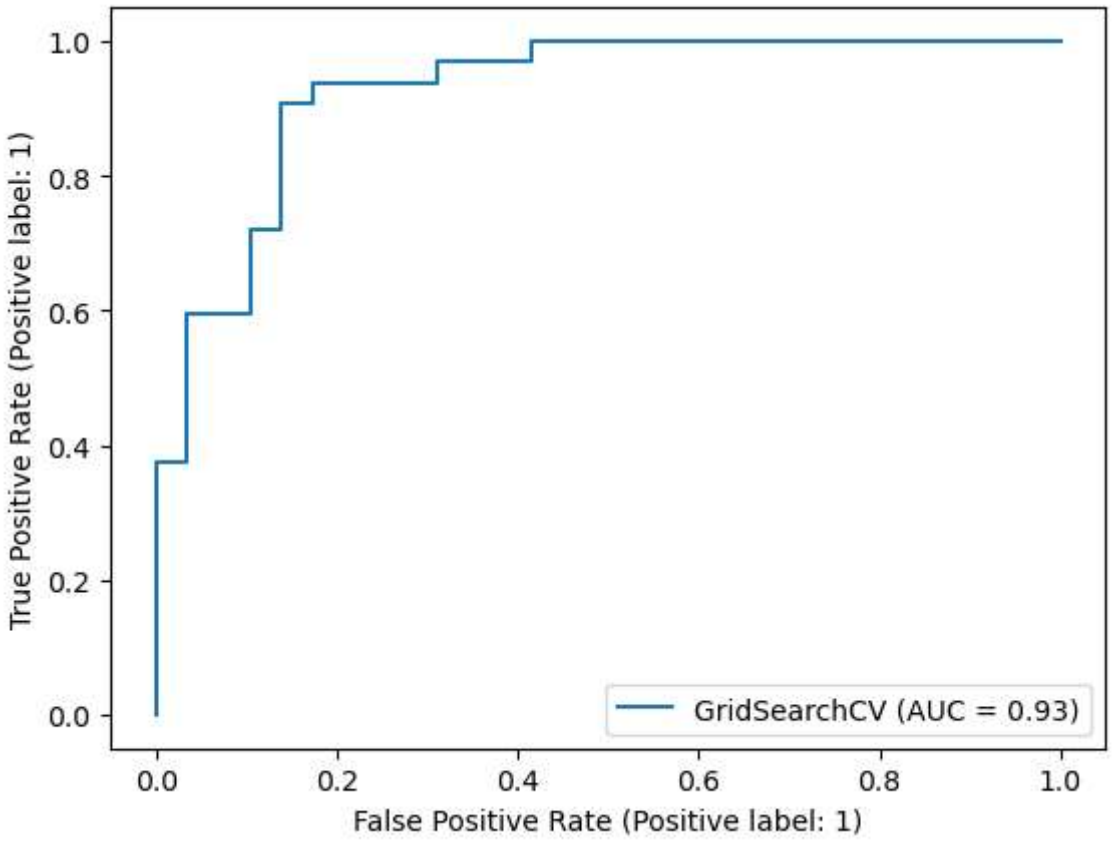
In [43]:
```
y_preds
```

Out[43]:
```
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 [44]:
```
y_test
```

Out[44]:
```
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 [48]:
```
# ROC Curve and AUC Score
#Plot ROC cruve and calculate AUC Metric

#THIS WONT WORK # plot_roc_curve(gs_log_reg, X_test, y_test)
# Chat GPT helped here :
roc_display = RocCurveDisplay.from_estimator(gs_log_reg, X_test, y_test)
#roc_display.plot()
```



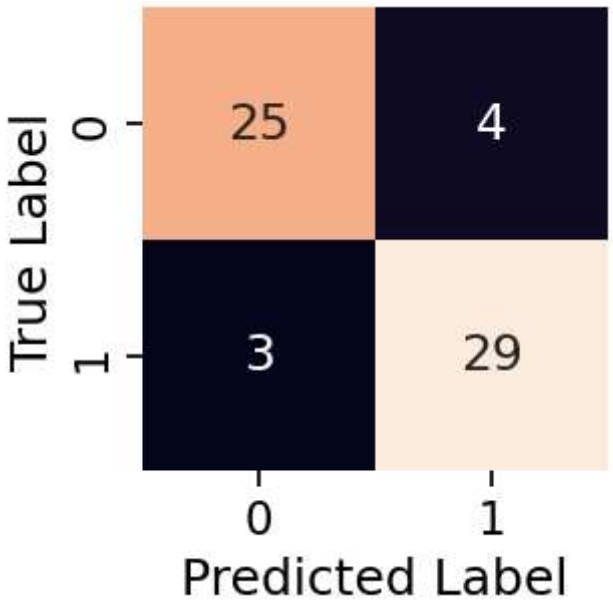
In [49]:
```
# COnfusion Matrix

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

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

In [52]:
```
sns
```

Out[52]:
```
<module 'seaborn' from 'C:\\Users\\HP\\AI_bootcamp\\sample_project_1\\env\\lib\\site-packages\\seaborn\\__init__.py'>
```

In [58]:
```
# usinf seaborn for better visualization
#sns.set(font_size=1.5)  <- didnt work again Chat GPT helped

sns.set_context("notebook", font_scale=1.5)

def plot_conf_mat(y_test, y_preds):
    """
    Plots a nice looking conf 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 get classification Report and cross validated , precision , recall and F1 Score

In [59]:
```
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

In [62]:
```
#Cross_val_score
#Check best hyper parameter
gs_log_reg.best_params_
```

Out[62]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [63]:
```
#Create new classifier with best parameters
clf = LogisticRegression(C=0.20433597178569418, solver="liblinear")
```

In [64]:
```
#cross-validated accuracy
cv_acc = cross_val_score(clf,
                         X,
                         y,
                         cv=5,
                         scoring="accuracy")
cv_acc
```

Out[64]:
```
array([0.81967213, 0.90163934, 0.86885246, 0.88333333, 0.75      ])
```

In [65]:
```
cv_acc = np.mean(cv_acc)
cv_acc
```

Out[65]:
```
0.8446994535519124
```

In [66]:
```
#Cross-validated precision
cv_precision = cross_val_score(clf,
                           X,
                           y,
                           cv=5,
                           scoring="precision")
cv_precision = np.mean(cv_precision)
cv_precision
```

Out[66]:
```
0.8207936507936507
```

In [67]:
```
#Cross-validated recall
cv_recall = cross_val_score(clf,
                           X,
                           y,
                           cv=5,
                           scoring="recall")
cv_recall = np.mean(cv_recall)
cv_recall
```

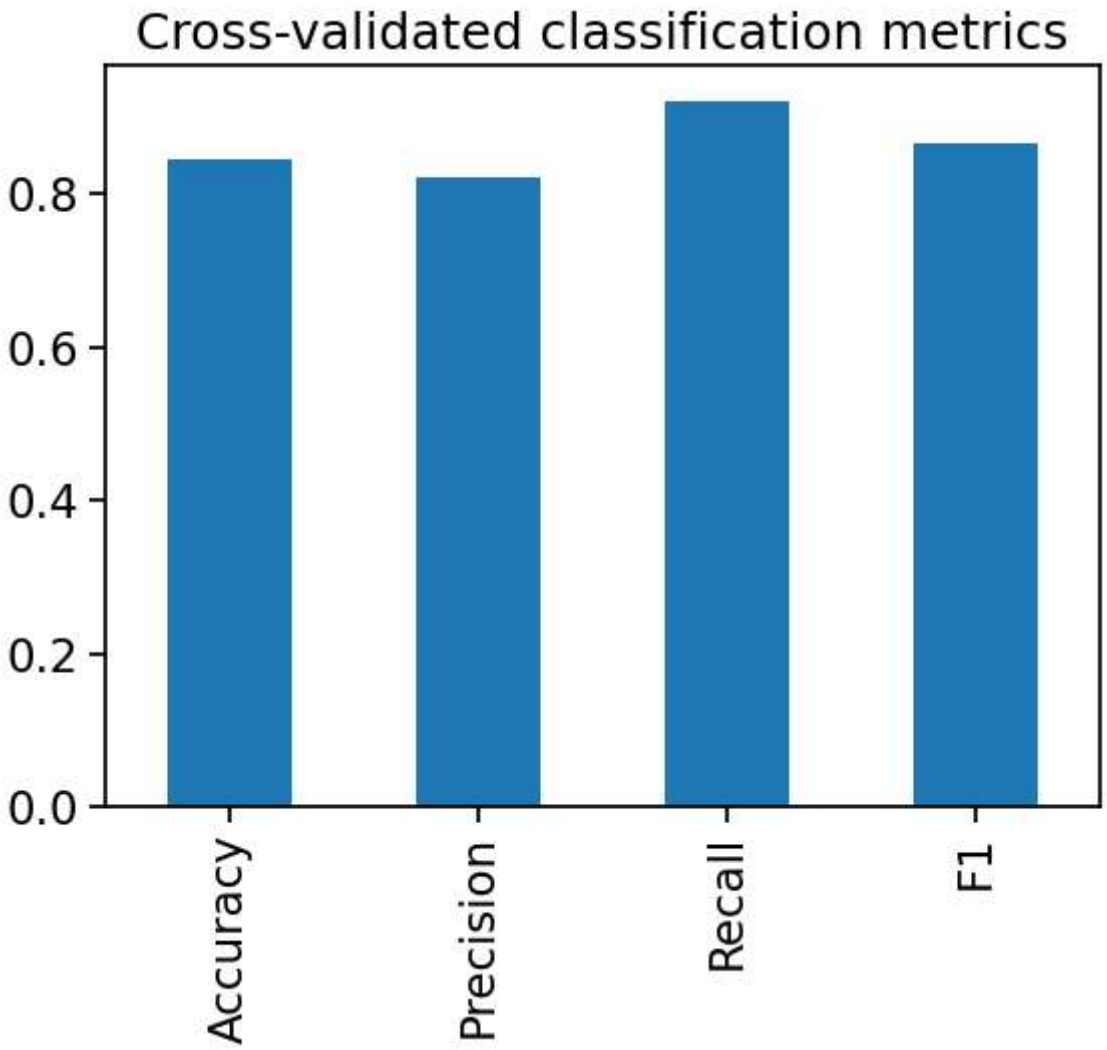
Out[67]:
```
0.9212121212121213
```

In [68]:
```
#Cross-validated f1 score
cv_f1 = cross_val_score(clf,
                           X,
                           y,
                           cv=5,
                           scoring="f1")
cv_f1 = np.mean(cv_f1)
cv_f1
```

Out[68]:
```
0.8673007976269721
```

In [71]:
```
#Visualize cross validated metrics

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

cv_metrics.T.plot.bar(title="Cross-validated classification metrics",legend= False)
```

Out[71]:
```
<Axes: title={'center': 'Cross-validated classification metrics'}>
```



Feature Importance

Which features contribued most to the outcomes of the model

In [73]:
```
clf = LogisticRegression(C=0.20433597178569418, solver="liblinear")

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

In [74]:
```
clf.coef_
```

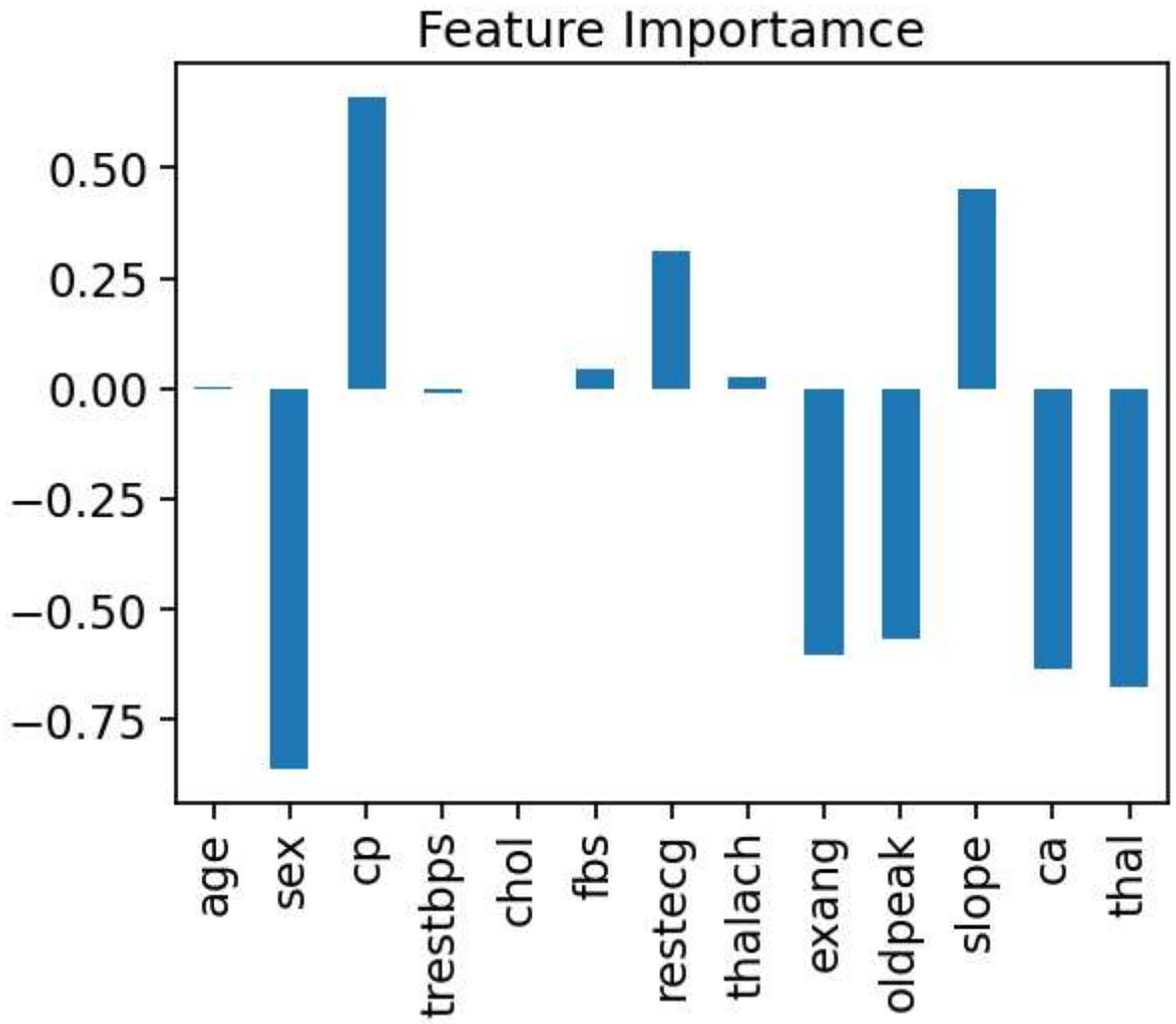
Out[74]:
```
array([[ 0.00316728, -0.86044651,  0.66067041, -0.01156993, -0.00166374,
         0.04386107,  0.31275847,  0.02459361, -0.6041308 , -0.56862804,
         0.45051628, -0.63609897, -0.67663373]])
```

In [75]:
```
#Match coef of features to columns

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

Out[75]:
```
{'age': 0.0031672801993431563,
 'sex': -0.8604465072345515,
 'cp': 0.6606704082033799,
 'trestbps': -0.01156993168080875,
 'chol': -0.001663744504776871,
 'fbs': 0.043861071652469864,
 'restecg': 0.31275846822418324,
 'thalach': 0.024593613737779126,
 'exang': -0.6041308000615746,
 'oldpeak': -0.5686280368396555,
 'slope': 0.4505162797258308,
 'ca': -0.6360989676086223,
 'thal': -0.6766337263029825}
```

In [76]:
```
# Visualize feature importance
feature_df = pd.DataFrame(feature_dict,index=[0])
feature_df.T.plot.bar(title="Feature Importamce",legend=False)
```

Out[76]:
```
<Axes: title={'center': 'Feature Importamce'}>
```



In [77]:
```
pd.crosstab(df["sex"],df["target"])
```

Out[77]:

target	0	1
sex		
0	24	72
1	114	93

In [78]:
```
pd.crosstab(df["slope"],df["target"])
```

Out[78]:

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

In [ ]: