Prediciting Heart-Disease using machine learning

This notebook looks into using various Python-based machine learning and Data science libearies 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 approach.

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

1. Problem Defination

In a statement,

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

2. Data

The original data came from the claveland from the UCI Machine learning Repository. https://archive.ics.uci.edu/ml/datasets/Heart+Disease (https://archive.ics.uci.edu/ml/datasets/Heart+Disease) There is also a version of it on Kaggle. https://www.kaggle.com/ronitf/heart-disease-uci/metadata (https://www.kaggle.com/ronitf/heart-disease-uci/metadata)

3. Evaluation

> If we can reach 95% accuracy at predicting whether or not a patient has heart disease during the proof of the concept , we will persue the project.

4. Features

This is where you will get information about each of the features in your data. **Create data dictionary**

- age in years
- sex(1=Male,0=Female)
- chest pain type (4 values)
- resting blood pressure
- serum cholestoral in mg/dl
- fasting blood sugar > 120 mg/dl(1=true,0=false)

- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina(1=yes,0=no)
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by flourosopy
- thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
- target 1 or 0

Preparing the tools

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

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

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

 #we want out plots to apear inside the notebook.
%matplotlib inline


#Import models from Sci-kit Learn
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# 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 The Data

In [2]:
```
df=pd.read_csv("heart-disease.csv")
df.shape #rows and columns
```

Out[2]:
```
(303, 14)
```

Data Exploration (EDA)

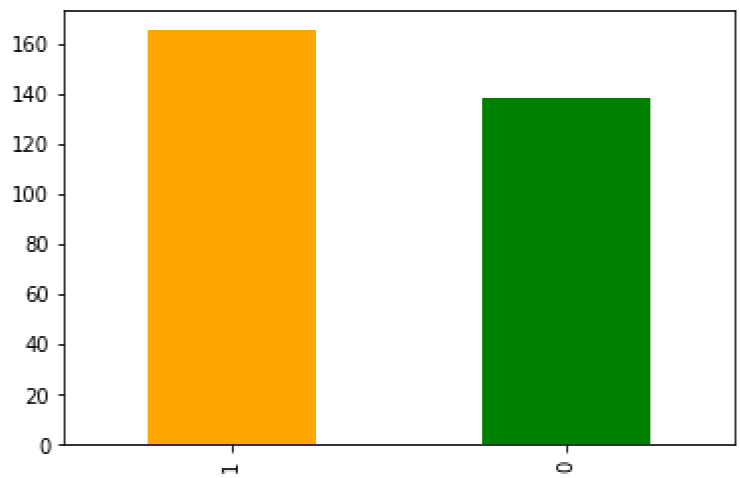
The goal here is to find out more about the data and become the subject matter expert on the dataset you are working with.

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

In [3]:
```
# Lets find out how many of each class there
df["target"].value_counts()
```

Out[3]:
```
1    165
0    138
Name: target, dtype: int64
```

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



In [5]:
```
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 [6]:
```
#Are there any missing values
df.isna().sum()
```

Out[6]:
```
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 [7]:
```
df.describe()
```

Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202

Heart Disease Frequency according to sex

In [8]:
```
df.sex.value_counts()
```

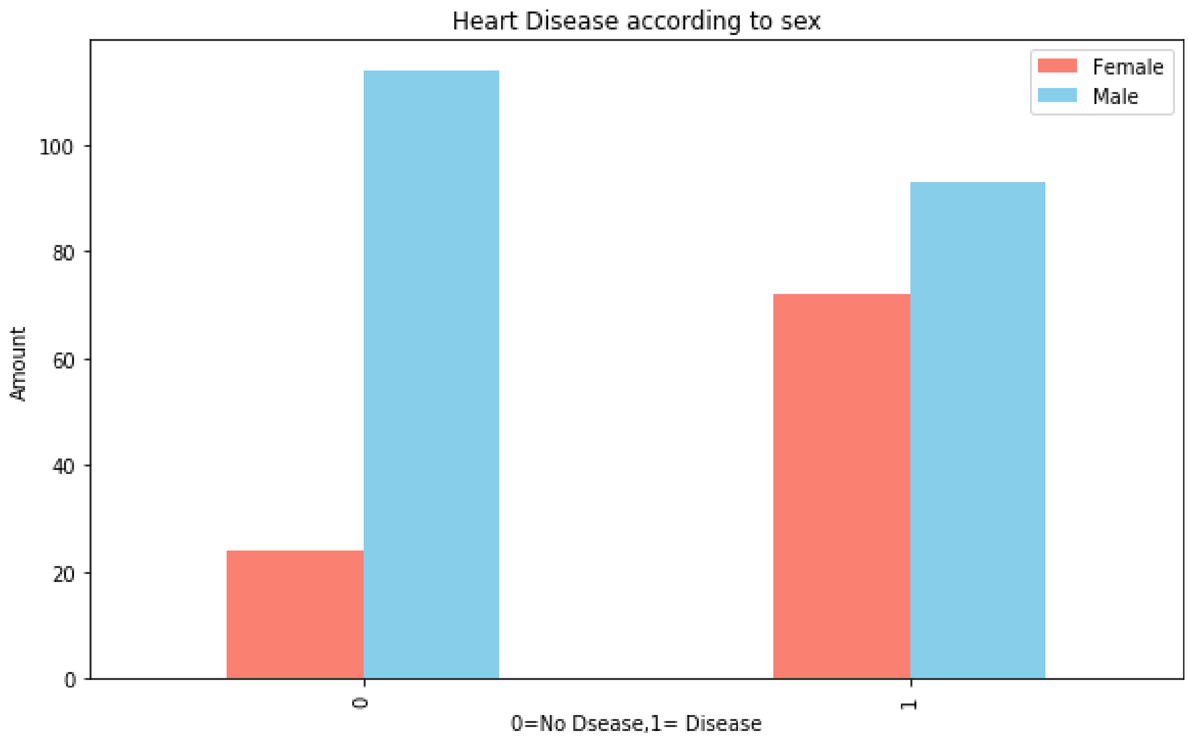
Out[8]:
```
1    207
0     96
Name: sex, dtype: int64
```

In [9]:
```
# Compare Sex column with Target
pd.crosstab(df.target,df.sex)
```

Out[9]:

sex	0	1
target		
0	24	114
1	72	93

In [10]:
```
#create a plot of crosstab
pd.crosstab(df.target,df.sex).plot(kind="bar",figsize=(10,6),color=["salmon",
"skyblue"])
plt.title("Heart Disease according to sex")
plt.xlabel("0=No Dsease,1= Disease")
plt.ylabel("Amount")
plt.legend(["Female","Male"]);
```



Age vs Max heart rate for heart disease

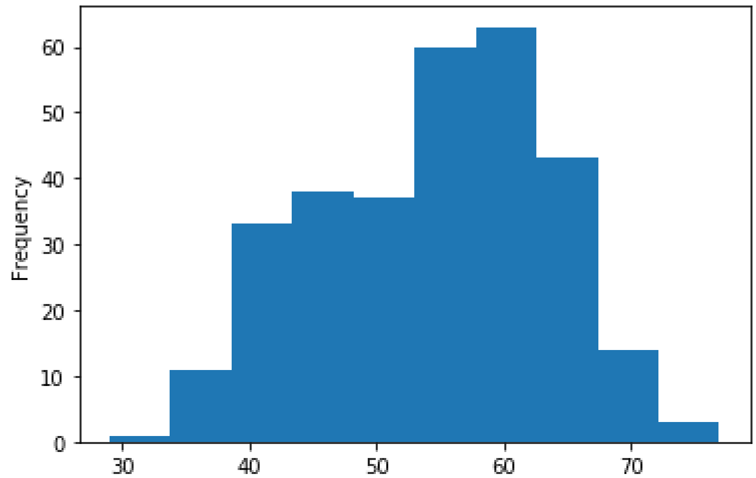
In [11]:
```
#Craeting another plot

plt.figure(figsize=(10,6))

#Scatter for positive
plt.scatter(df.age[df.target==1],
            df.thalach[df.target==1],
            c="orange")
# Scatter for negative
plt.scatter(df.age[df.target==0],
            df.thalach[df.target==0],
            c="green")

#Cutomise it
plt.title("Heart disease according to age and thalach")
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate (Thalach) ")
plt.legend(["Age","Thalach"]);
```

In [12]:
```
df.age.plot.hist();
```

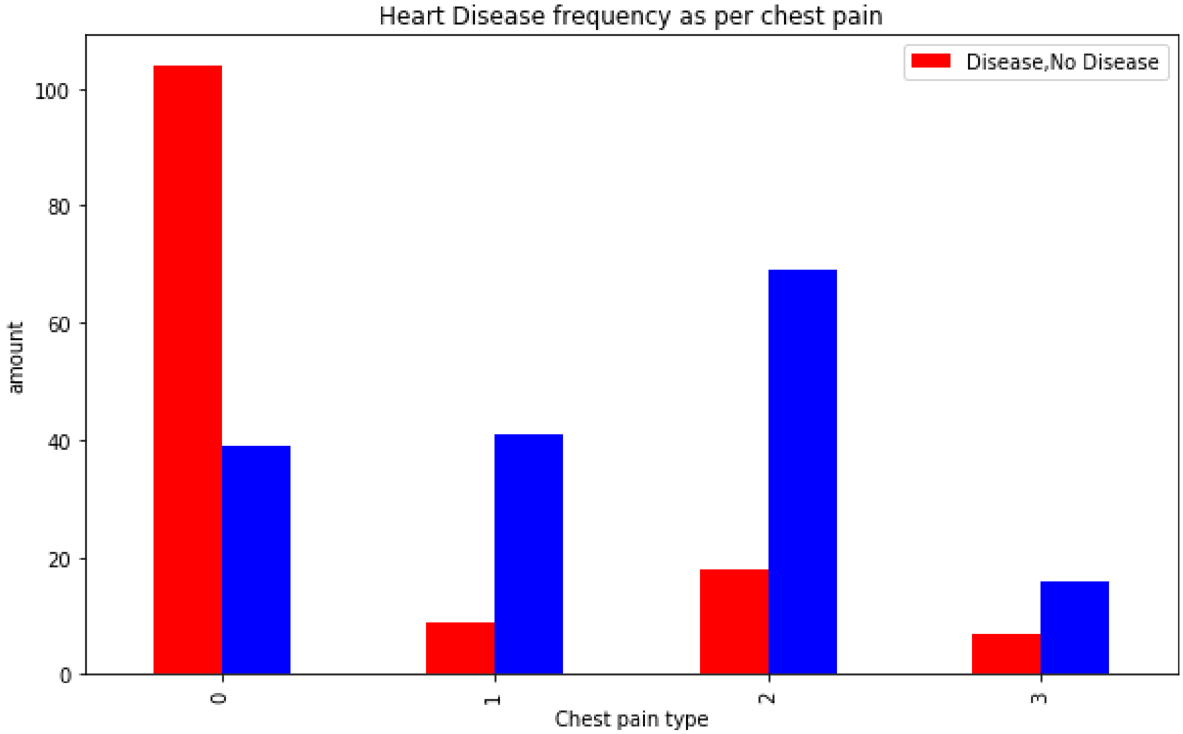


In [13]:
```
### Heart disease frequency as per the chest pain with respect to the target.
pd.crosstab(df.cp,df.target)
```

Out[13]:

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

In [14]:
```
pd.crosstab(df.cp,df.target).plot(kind="bar",figsize=(10,6),color=["red","blu
e"])
plt.title("Heart Disease frequency as per chest pain")
plt.xlabel("Chest pain type")
plt.ylabel("amount")
plt.legend(["Disease,No Disease"]);
```



In [15]:
```
df.head()
```

Out[15]:

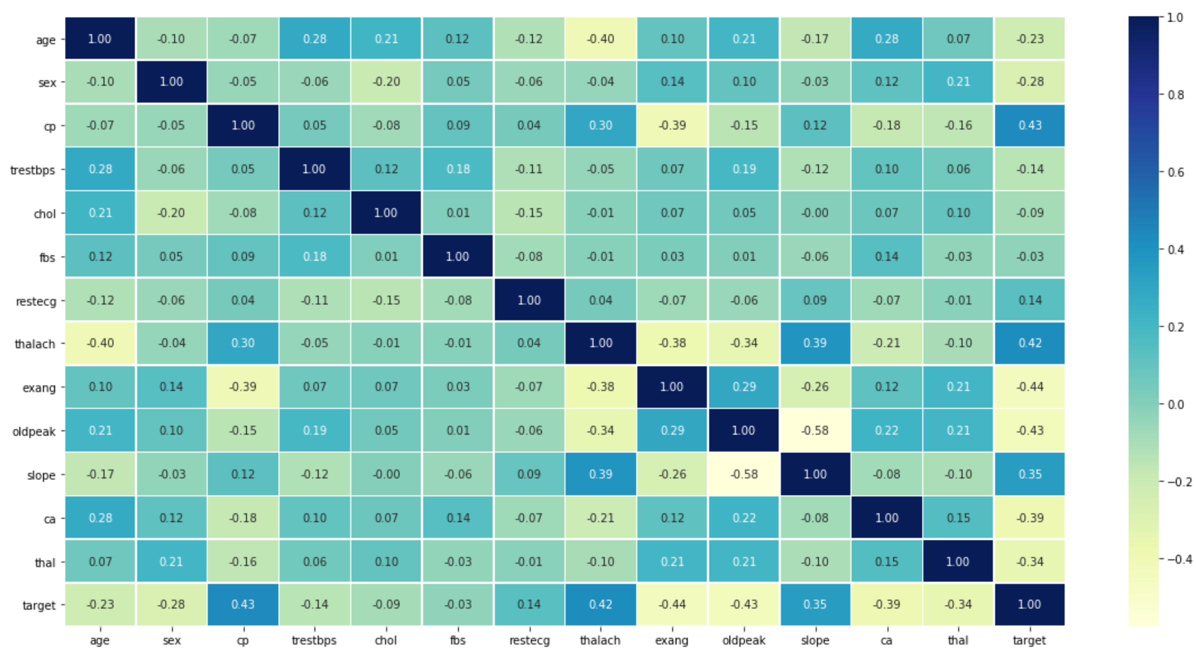
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [16]:
```
# Make a CoRealation matrix
df.corr()
```

Out[16]:

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

In [17]:
```
cor_matrix=df.corr()
fig,ax=plt.subplots(figsize=(20,10))
ax=sns.heatmap(cor_matrix,
               annot=True,
               linewidth=0.5,
               fmt=".2f",
               cmap="YlGnBu");
```



5. Modeling

In [18]:
```
df.head()
```

Out[18]:

	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 [19]:
```
# Split data into x and y

x=df.drop("target",axis=1)
y=df["target"]
```

In [20]:
```
#split the data into train and test
np.random.seed(42)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

5.1 Choosing a right model

1. Logistic regression
2. Kneighbors
3. Random forest classifier

In [21]:
```
#put the data into dictionary
models={"Logistic Regression":LogisticRegression(),
       "KNN":KNeighborsClassifier(),
       "Random Forest Classifier":RandomForestClassifier()}

#Create a funtion to fit and score a model
def fit_and_score(models,x_train,x_test,y_train,y_test):
    """
    Fits and evaluate the given machine learning models.
    models: A dict of different machine learning models in the classifier.
    x_train: Training data(no labels).
    x_test:testing data(no labels)
    y_training:training labels
    y_testign:testing labels
    """

    #Set random seed
    np.random.seed(42)
    #make a dict to keep model scores.
    model_scores={}
    #loops throgh the models
    for name,model in models.items():
        #fit the model to the data
        model.fit(x_train,y_train)
        #evaluate the model and append its score to model_score
        model_scores[name]=model.score(x_test,y_test)
    return model_scores
```

In [22]:
```
model_scores=fit_and_score(models,
                          x_train=x_train,
                          x_test=x_test,
                          y_train=y_train,
                          y_test=y_test)
model_scores
```

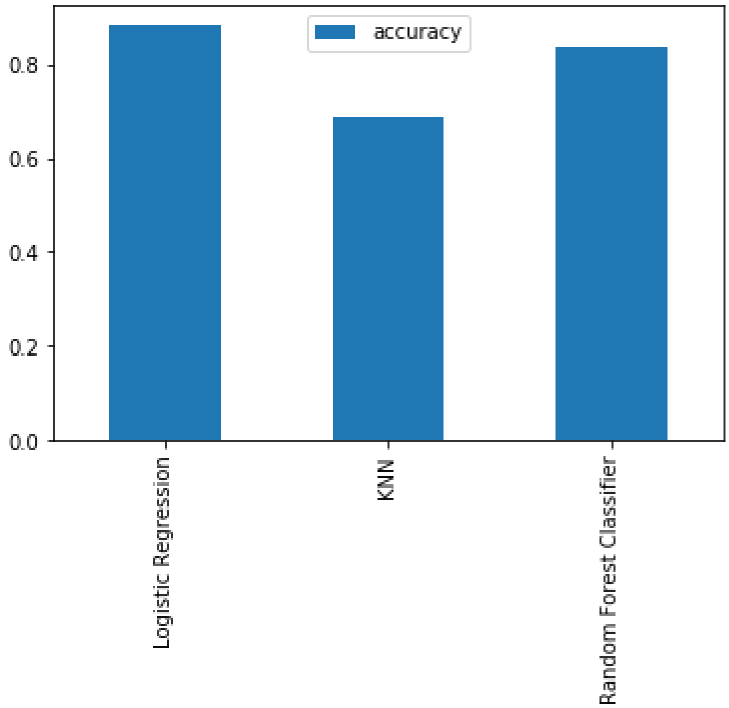
```
C:\Users\Akash\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[22]:
```
{'Logistic Regression': 0.8852459016393442,
 'KNN': 0.6885245901639344,
 'Random Forest Classifier': 0.8360655737704918}
```

Model comparision

In [23]:
```
model_compare=pd.DataFrame(model_scores,index=["accuracy"])
model_compare.T.plot.bar();
```



- Now we have got baseline model and we knows a models first predictiction are not always what we should based our next steps off. what should we do? ### Looks at following
- Hyper parameter
- feature importance
- confusion matrix
- cross validation
- precision
- recall
- f1 score
- classification report
- roc curve
- auc curve ### Hyper parameter tunning

In [24]:
```
#K Neighbors
train_scores=[]
test_scores=[]

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

#Set up KNN Instance
knn=KNeighborsClassifier()

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

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

    #Update training scores
    train_scores.append(knn.score(x_train,y_train))

    #update the testing score
    test_scores.append(knn.score(x_test,y_test))
```

In [25]:
```
train_scores
```

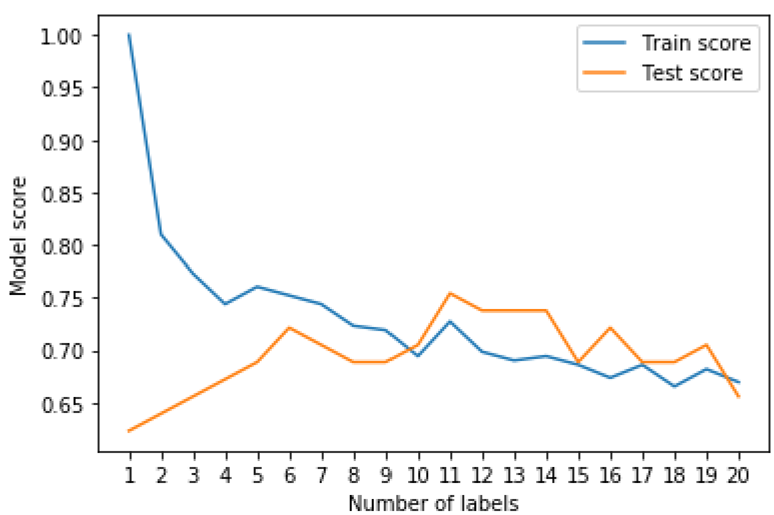
Out[25]:
```
[1.0,
 0.8099173553719008,
 0.7727272727272727,
 0.743801652892562,
 0.7603305785123967,
 0.7520661157024794,
 0.743801652892562,
 0.7231404958677686,
 0.71900826446281,
 0.6942148760330579,
 0.7272727272727273,
 0.6983471074380165,
 0.6900826446280992,
 0.6942148760330579,
 0.6859504132231405,
 0.6735537190082644,
 0.6859504132231405,
 0.6652892561983471,
 0.6818181818181818,
 0.6694214876033058]
```

In [26]:
```
test_scores
```

Out[26]:
```
[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 [27]:
```
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 labels")
plt.ylabel("Model score")
plt.legend()
print(f"Maximum KNN score on test data:{max(test_scores)*100:.2f}%");
```

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



Hyper paramter tunning using RandomizedSearchCV

we are going to tune our model logesticRegression and RandomForestClassifier using RandomSearchCV

In [28]:
```
# Create our hyper parameter grid for logistic regression
log_reg_grid={"C":np.logspace(-4,4,20),
             "solver":["liblinear"]}
#Create hyper parameter grid for random forest classifier
ran_for_grid={"n_estimators":np.arange(10,1000,50),
             "max_depth":[None,3,5,10],
#               "max_features":
             "min_samples_split":np.arange(2,20,2),
             "min_samples_leaf":np.arange(1,20,2)}
```

- Now we have got hyper parameter grid setup for each of our model, lets tune them using RandomizedSearchCV

In [29]:
```
#Tune LogisticRegression
np.random.seed(42)
#setup 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 search model for logistic regression
rs_log_reg.fit(x_train,y_train)
```

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

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.4s finished
```

Out[29]:
```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=LogisticRegression(C=1.0, class_weight=None,
                                                dual=False, fit_intercept=Tru
e,
                                                intercept_scaling=1,
                                                l1_ratio=None, max_iter=100,
                                                multi_class='auto', n_jobs=No
ne,
                                                penalty='l2', random_state=No
ne,
                                                solver='lbfgs', tol=0.0001,
                                                verbose=0, warm_start=False),
                   iid='deprecated', n_iter=20, n_jobs=None,
                   param_distributions={'C':...
       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']},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=True)
```

In [30]:
```
rs_log_reg.best_params_
```

Out[30]:
```
{'solver': 'liblinear', 'C': 0.23357214690901212}
```

In [31]:
```
rs_log_reg.score(x_test,y_test)
```

Out[31]:
```
0.8852459016393442
```

- Now we have tunned logistic regression. lets do the same for the random forest classifier.

In [32]:
```
#setup random seed
np.random.seed(9)

#setup random hyperparameter search for random forest classifier
rs_ran_for=RandomizedSearchCV(RandomForestClassifier(),
                             param_distributions=ran_for_grid,
                             cv=5,
                              n_iter=20,
                             verbose=2)
# fit the random hyper parameter search model for random forest classifier
rs_ran_for.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=Non
e

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.

[CV]  n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=No
ne, total=   0.7s
[CV] n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=Non
e

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.6s remaining:    0.
0s
```

```
[CV]  n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=No
ne, total=   0.7s
[CV] n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=Non
e
[CV]  n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=No
ne, total=   0.6s
[CV] n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=Non
e
[CV]  n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=No
ne, total=   0.6s
[CV] n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=Non
e
[CV]  n_estimators=110, min_samples_split=4, min_samples_leaf=5, max_depth=No
ne, total=   0.6s
[CV] n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=5
[CV]  n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=
5, total=   1.2s
[CV] n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=5
[CV]  n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=
5, total=   1.1s
[CV] n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=5
[CV]  n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=
5, total=   1.2s
[CV] n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=5
[CV]  n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=
5, total=   1.2s
[CV] n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=5
[CV]  n_estimators=210, min_samples_split=14, min_samples_leaf=9, max_depth=
5, total=   1.1s
[CV] n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=1
0
[CV]  n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=
10, total=   0.6s
[CV] n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=1
0
[CV]  n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=
10, total=   0.6s
[CV] n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=1
0
[CV]  n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=
10, total=   0.6s
[CV] n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=1
0
[CV]  n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=
10, total=   0.6s
[CV] n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=1
0
[CV]  n_estimators=110, min_samples_split=14, min_samples_leaf=15, max_depth=
10, total=   0.7s
[CV] n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=Non
e
[CV]  n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=No
ne, total=   0.4s
[CV] n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=Non
e
[CV]  n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=No
ne, total=   0.4s
```

```
[CV] n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=Non
e
[CV]  n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=No
ne, total=   0.4s
[CV] n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=Non
e
[CV]  n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=No
ne, total=   0.4s
[CV] n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=Non
e
[CV]  n_estimators=60, min_samples_split=16, min_samples_leaf=5, max_depth=No
ne, total=   0.4s
[CV] n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=10
[CV]  n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=1
0, total=   0.1s
[CV] n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=10
[CV]  n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=1
0, total=   0.1s
[CV] n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=10
[CV]  n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=1
0, total=   0.1s
[CV] n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=10
[CV]  n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=1
0, total=   0.1s
[CV] n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=10
[CV]  n_estimators=10, min_samples_split=10, min_samples_leaf=9, max_depth=1
0, total=   0.1s
[CV] n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=N
one
[CV]  n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=
None, total=   1.9s
[CV] n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=N
one
[CV]  n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=
None, total=   1.9s
[CV] n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=N
one
[CV]  n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=
None, total=   1.9s
[CV] n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=N
one
[CV]  n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=
None, total=   2.0s
[CV] n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=N
one
[CV]  n_estimators=360, min_samples_split=18, min_samples_leaf=19, max_depth=
None, total=   2.0s
[CV] n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=5
[CV]  n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=
5, total=   4.2s
[CV] n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=5
[CV]  n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=
5, total=   4.0s
[CV] n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=5
[CV]  n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=
5, total=   3.8s
[CV] n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=5
```

```
[CV]  n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=
5, total=   3.8s
[CV] n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=5
[CV]  n_estimators=710, min_samples_split=16, min_samples_leaf=15, max_depth=
5, total=   3.6s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=No
ne
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=N
one, total=   2.5s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=No
ne
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=N
one, total=   2.5s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=No
ne
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=N
one, total=   2.5s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=No
ne
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=N
one, total=   2.4s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=No
ne
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=13, max_depth=N
one, total=   2.5s
[CV] n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=
5, total=   3.6s
[CV] n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=
5, total=   3.8s
[CV] n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=
5, total=   3.6s
[CV] n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=
5, total=   3.5s
[CV] n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=660, min_samples_split=18, min_samples_leaf=11, max_depth=
5, total=   3.5s
[CV] n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=10
[CV]  n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=1
0, total=   4.4s
[CV] n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=10
[CV]  n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=1
0, total=   3.9s
[CV] n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=10
[CV]  n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=1
0, total=   3.9s
[CV] n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=10
[CV]  n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=1
0, total=   3.9s
[CV] n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=10
[CV]  n_estimators=710, min_samples_split=8, min_samples_leaf=3, max_depth=1
0, total=   4.3s
[CV] n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=
```

```
5, total=   3.5s
[CV] n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=
5, total=   3.3s
[CV] n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=
5, total=   3.3s
[CV] n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=
5, total=   3.5s
[CV] n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=5
[CV]  n_estimators=610, min_samples_split=4, min_samples_leaf=11, max_depth=
5, total=   3.9s
[CV] n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=10
[CV]  n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=1
0, total=   4.0s
[CV] n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=10
[CV]  n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=1
0, total=   4.0s
[CV] n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=10
[CV]  n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=1
0, total=   4.2s
[CV] n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=10
[CV]  n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=1
0, total=   4.2s
[CV] n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=10
[CV]  n_estimators=760, min_samples_split=6, min_samples_leaf=15, max_depth=1
0, total=   3.9s
[CV] n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=10
[CV]  n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=1
0, total=   4.3s
[CV] n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=10
[CV]  n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=1
0, total=   4.7s
[CV] n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=10
[CV]  n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=1
0, total=   4.6s
[CV] n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=10
[CV]  n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=1
0, total=   4.6s
[CV] n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=10
[CV]  n_estimators=810, min_samples_split=2, min_samples_leaf=1, max_depth=1
0, total=   4.5s
[CV] n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5,
total=   4.2s
[CV] n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5,
total=   3.9s
[CV] n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5,
total=   3.9s
[CV] n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5,
total=   3.9s
[CV] n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=710, min_samples_split=6, min_samples_leaf=3, max_depth=5,
```

```
total=   4.2s
[CV] n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=3
[CV]  n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=
3, total=   2.5s
[CV] n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=3
[CV]  n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=
3, total=   2.5s
[CV] n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=3
[CV]  n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=
3, total=   2.5s
[CV] n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=3
[CV]  n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=
3, total=   2.5s
[CV] n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=3
[CV]  n_estimators=460, min_samples_split=14, min_samples_leaf=5, max_depth=
3, total=   2.5s
[CV] n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5,
total=   4.0s
[CV] n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5,
total=   3.4s
[CV] n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5,
total=   3.3s
[CV] n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5,
total=   3.3s
[CV] n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5
[CV]  n_estimators=610, min_samples_split=6, min_samples_leaf=3, max_depth=5,
total=   3.6s
[CV] n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=No
ne
[CV]  n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=N
one, total=   3.6s
[CV] n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=No
ne
[CV]  n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=N
one, total=   3.5s
[CV] n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=No
ne
[CV]  n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=N
one, total=   3.7s
[CV] n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=No
ne
[CV]  n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=N
one, total=   3.6s
[CV] n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=No
ne
[CV]  n_estimators=660, min_samples_split=4, min_samples_leaf=19, max_depth=N
one, total=   3.8s
[CV] n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=10
[CV]  n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=1
0, total=   3.0s
[CV] n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=10
[CV]  n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=1
0, total=   3.1s
```

```
[CV] n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=10
[CV]  n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=1
0, total=   3.0s
[CV] n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=10
[CV]  n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=1
0, total=   3.1s
[CV] n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=10
[CV]  n_estimators=560, min_samples_split=2, min_samples_leaf=13, max_depth=1
0, total=   3.3s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=3
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=
3, total=   2.5s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=3
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=
3, total=   2.4s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=3
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=
3, total=   2.5s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=3
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=
3, total=   2.5s
[CV] n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=3
[CV]  n_estimators=460, min_samples_split=8, min_samples_leaf=19, max_depth=
3, total=   2.5s
[CV] n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=10
[CV]  n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=1
0, total=   4.9s
[CV] n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=10
[CV]  n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=1
0, total=   4.6s
[CV] n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=10
[CV]  n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=1
0, total=   4.5s
[CV] n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=10
[CV]  n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=1
0, total=   4.9s
[CV] n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=10
[CV]  n_estimators=860, min_samples_split=4, min_samples_leaf=17, max_depth=1
0, total=   4.6s
```

```
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  4.6min finished
```

Out[32]:
```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.
0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=
0.0,
                                                    n_estimators=100,
                                                    n_jobs...
                   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])},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

In [33]:
```
rs_ran_for.best_params_
```

Out[33]:
```
{'n_estimators': 110,
 'min_samples_split': 14,
 'min_samples_leaf': 15,
 'max_depth': 10}
```

In [34]:
```
rs_ran_for.score(x_test,y_test)
```

Out[34]:
```
0.8688524590163934
```

Hyper parameter tunning using GridSearchCV:

Since logistic regression model provieds best score so far, we will try and impove them again using GridSearchCV

In [35]:
```
# Different hyper parameter for our logistic regression mmodel
log_reg_grid={"C":np.logspace(-4,4,30),
             "solver":["liblinear"]}

#setup grid hyper parameter search for logistic regression
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

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:    2.1s finished
```

Out[35]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fals
e,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([1.00000000e-04, 1.8...
       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']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)
```

In [36]:
```
gs_log_reg.best_params_
```

Out[36]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [37]:
```
gs_log_reg.score(x_test,y_test)
```

Out[37]:
```
0.8852459016393442
```

Evaluating our tunned machine learning classifier, beyond accuracy

- ROC and AUC curve
- Confusion matrix
- classification report
- Precision
- Recall
- F1 score
- ....And it would be great if cross validation was used where possible ### To make comparision and evaluate our trained model, first we need to make few predictions

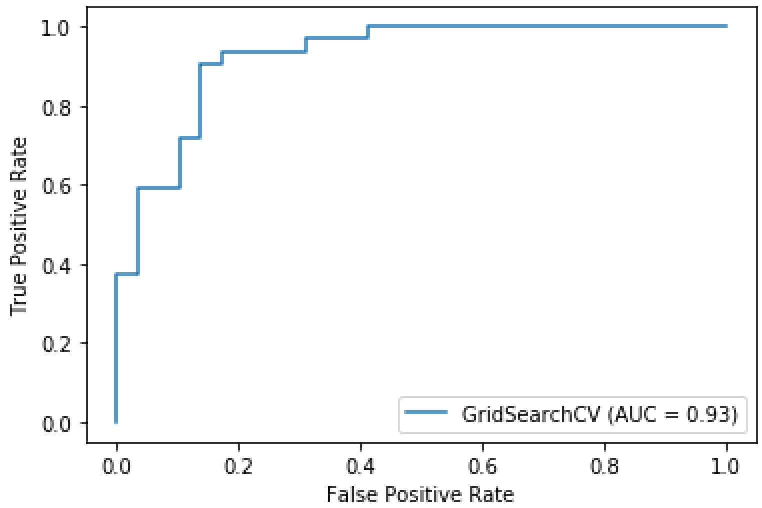
In [38]:
```
#make predictions with tunned model
y_preds=gs_log_reg.predict(x_test)
y_preds
```

Out[38]:
```
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 [39]:
```
y_test
```

Out[39]:
```
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 [40]:
```
#plot roc curve and calculate the auc metric
plot_roc_curve(gs_log_reg,x_test,y_test);
```

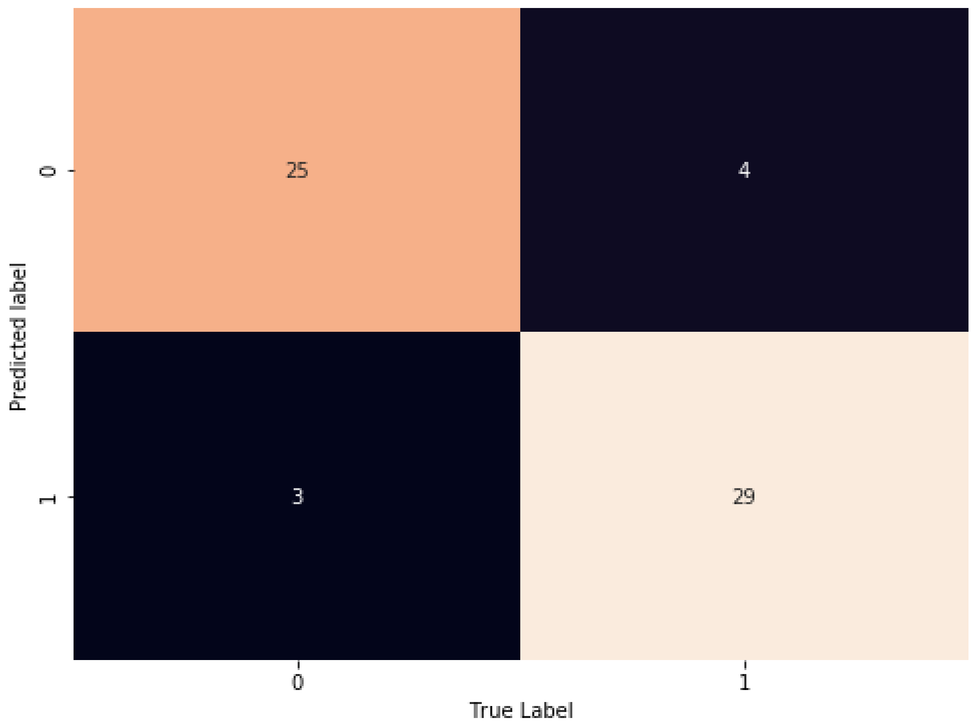


In [41]:
```
# Confusion matrix
print(confusion_matrix(y_test,y_preds))
```

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

In [42]:
```
def plot_conf_mat(y_test,y_preds):
    """
    A nice looking confusion matrix using sea born heatmap
    """
    fig,ax=plt.subplots(figsize=(8,6))
    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 have got a ROC curve, an AUC metric and confusion matrix, lets get a classification report as well as cross validated precision, racall and f1 score.

In [43]:
```
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 matrix using cross validation.

we are going to calculate the accuracy,precision, recall and f1_score using cross validation and to do so we will be using cross_val_score.

In [44]:
```
# Check best Hyper parameters
gs_log_reg.best_params_
```

Out[44]: {'C': 0.20433597178569418, 'solver': 'liblinear'}

In [45]:
```
#Create a new classifier with best params
clf=LogisticRegression(C=0.20433597178569418,
                      solver="liblinear")
```

In [46]:
```
# Cross validated accuracy
cv_acc=cross_val_score(clf,
                      x,
                      y,
                      cv=5,
                      scoring="accuracy")

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

Out[46]: 0.8446994535519124

In [47]:
```
# Cross validated precision
cv_precision=cross_val_score(clf,
                      x,
                      y,
                      cv=5,
                      scoring="precision")

cv_precision=np.mean(cv_precision)
cv_precision
```

Out[47]: 0.8207936507936507

In [48]:
```
# Cross validated Recall
cv_recall = cross_val_score(clf,
                      x,
                      y,
                      cv=5,
                      scoring= "recall")

cv_recall = np.mean(cv_recall)
cv_recall
```

Out[48]: 0.9212121212121213

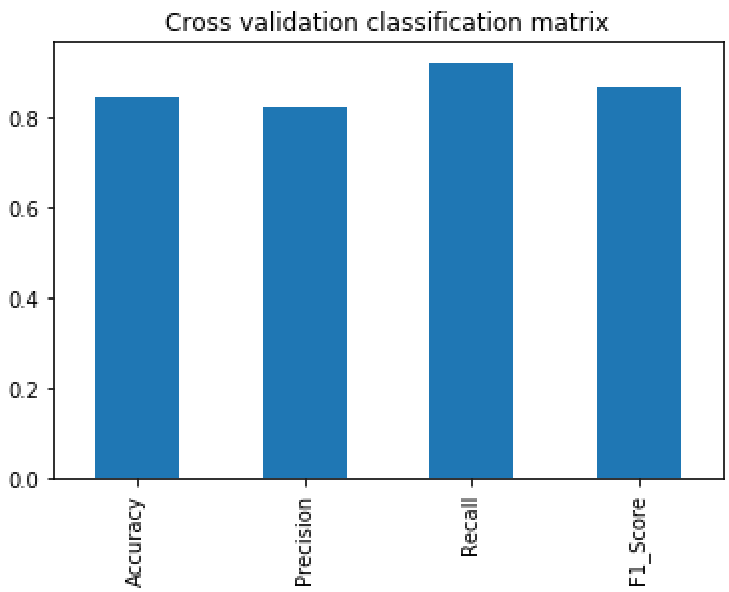
In [49]:
```
# Cross validated F1 Score
cv_f1=cross_val_score(clf,
                      x,
                      y,
                      cv=5,
                      scoring="f1")

cv_f1=np.mean(cv_f1)
cv_f1
```

Out[49]: 0.8673007976269721

In [50]:
```
### Visualise our cross valid metrics

cv_metrics=pd.DataFrame({"Accuracy":cv_acc,
                       "Precision":cv_precision,
                       "Recall":cv_recall,
                       "F1_Score":cv_f1},
                       index=[0])
cv_metrics.T.plot.bar(title="Cross validation classification matrix",
                     legend=False);
```



feature Imporatance:

- Feature imporatance is another way of asking," WHich feature contributed the most to the outcomes of the model and how did they contributed?
- Finding feature importance for each machine learning model. one way to find feature imporatance is to search for MODEL name feature importance
- lets find out the feature importance for our logistic regression model.

In [51]:
```
# Fit an intance of logistics regression
gs_log_reg.best_params_
```

Out[51]:
```
{'C': 0.20433597178569418, 'solver': 'liblinear'}
```

In [52]:
```
clf=LogisticRegression(C=0.20433597178569418,
                       solver="liblinear")
clf.fit(x_train,y_train);
```

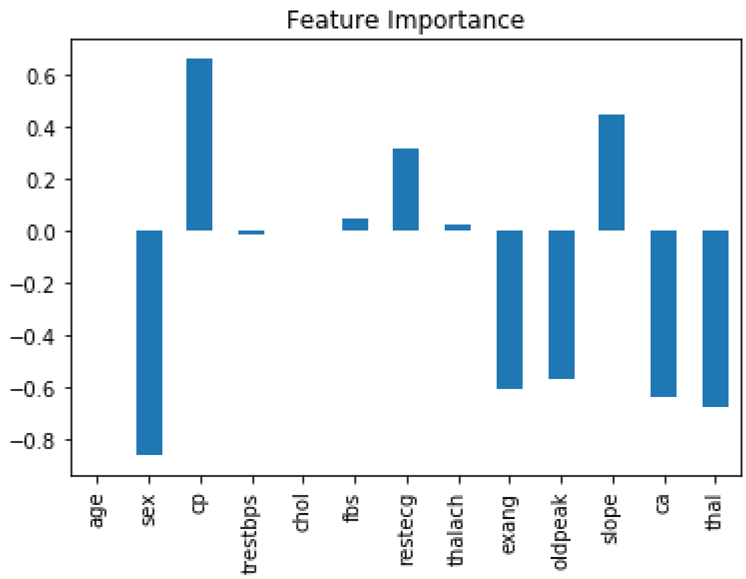
In [53]:
```
# Check coef
clf.coef_
```

Out[53]:
```
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 [54]:
```
# Match coefs features to columns
features_dict=dict(zip(df.columns,list(clf.coef_[0])))
features_dict
```

Out[54]:
```
{'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 [61]:
```
# Visualise feature importance
feature_df=pd.DataFrame(features_dict,index=[0])
feature_df.T.plot.bar(title="Feature Importance",legend=False);
```



6. Experimentation

Well we've completed all the metrics your boss requested. You should be able to put together a great report containing a confusion matrix, a handful of cross-valdated metrics such as precision, recall and F1 as well as which features contribute most to the model making a decision.

1. But after all this you might be wondering where step 6 in the framework is, experimentation.1ell the secret here is, as you might've guessed, the whole thing is experimentation.
2. From trying different models, to tuning different models to figuring out which hyperparameters were best.
3. What we've worked through so far has been a series of experiments.
4. And the truth is, we could keep going. But of course, things can't go on forever.
5. So by this stage, after trying a few different things, we'd ask ourselves did we meet the evaluation metric?

- Remember we defined one in step 3.

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

In this case, we didn't. The highest accuracy our model achieved was below 90%.

What next? You might be wondering, what happens when the evaluation metric doesn't get hit?

Is everything we've done wasted?

No.

It means we know what doesn't work. In this case, we know the current model we're using (a tuned version of LogisticRegression) along with our specific data set doesn't hit the target we set ourselves.

This is where step 6 comes into its own.

1. A good next step would be to discuss with your team or research on your own different options of going forward.
2. Could you collect more data?
3. Could you try a better model? If you're working with structured data, you might want to look into CatBoost or XGBoost.
4. Could you improve the current models (beyond what we've done so far)?
5. If your model is good enough, how would you export it and share it with others? (Hint: check out Scikit-Learn's documentation on model persistance)
6. The key here is to remember, your biggest restriction will be time. Hence, why it's paramount to minimise your times between experiments.
7. The more you try, the more you figure out what doesn't work, the more you'll start to get a hang of what does.

In [ ]: