1. Problem Definition

Predict whether or not patient have heart disease?

Heart Disease Data Dictionary

The following are the features we'll use to predict our target variable (heart disease or no heart disease).

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 Tools

In [1]:
```
# EDA & Plotting Tools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# Model Evaluators
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
```

Load Data

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

Out[2]: (303, 14)

In [3]:
```
### Data Exploration(EDA)
df.head(10)
```

Out[3]:

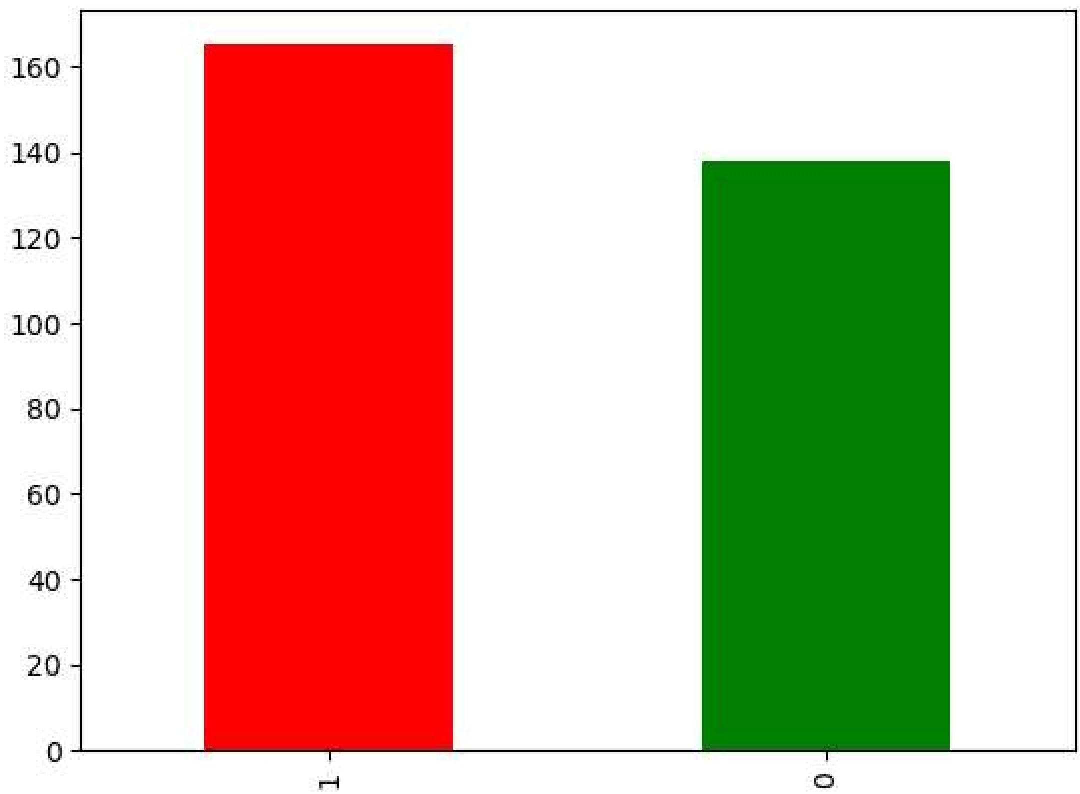
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1

In [4]:
```
df.target.value_counts()
```

Out[4]:
```
1    165
0    138
Name: target, dtype: int64
```

- **165 has disease and 138 havn't**

In [5]:
```
# Ploting the values with bar graph
df.target.value_counts().plot(kind="bar", color=["red","green"]);
```



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

Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	tl
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	149.6
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.9
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	133.5
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.(

Heat Disease Frequency according to Gender

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

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

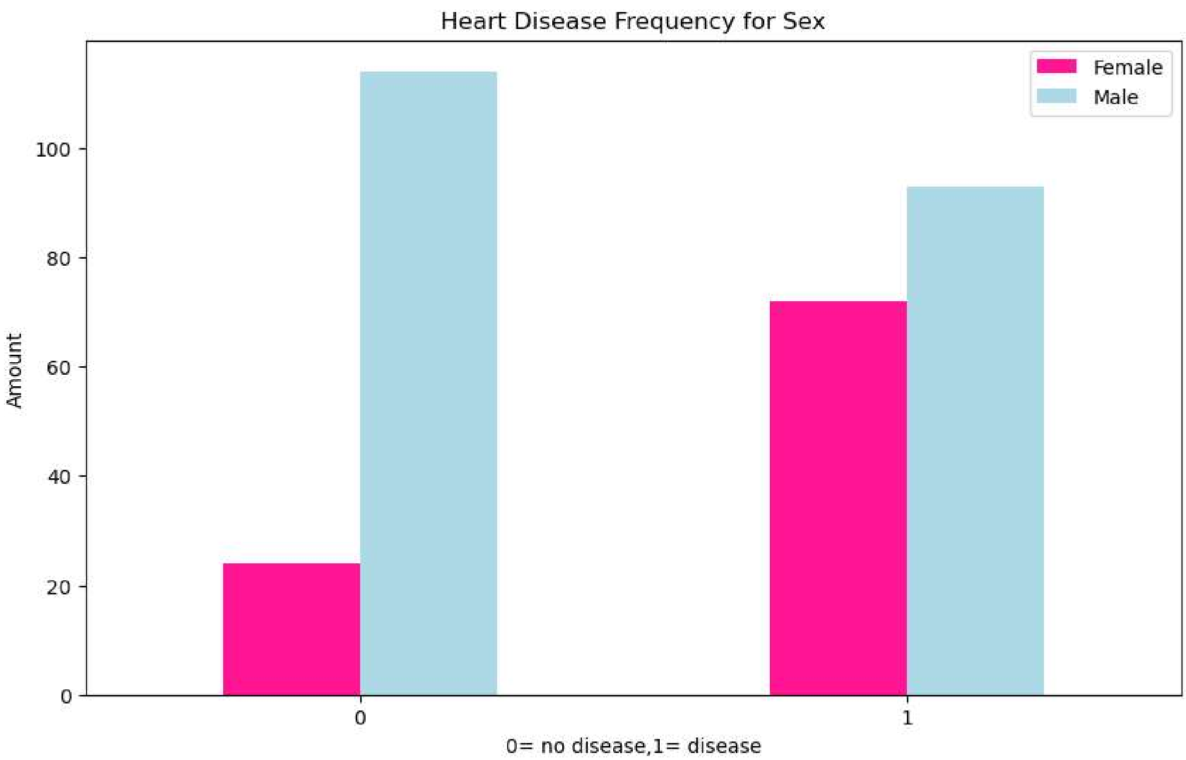
In [9]:
```
#compare target column with sex column
pd.crosstab(df.target,df.sex)
```

Out[9]:

sex	0	1
target		
0	24	114
1	72	93

- **As per comparison about 100 women and 72 of them have positive value of heart disease and there are about 200 male and about 50% have positive value of heart**

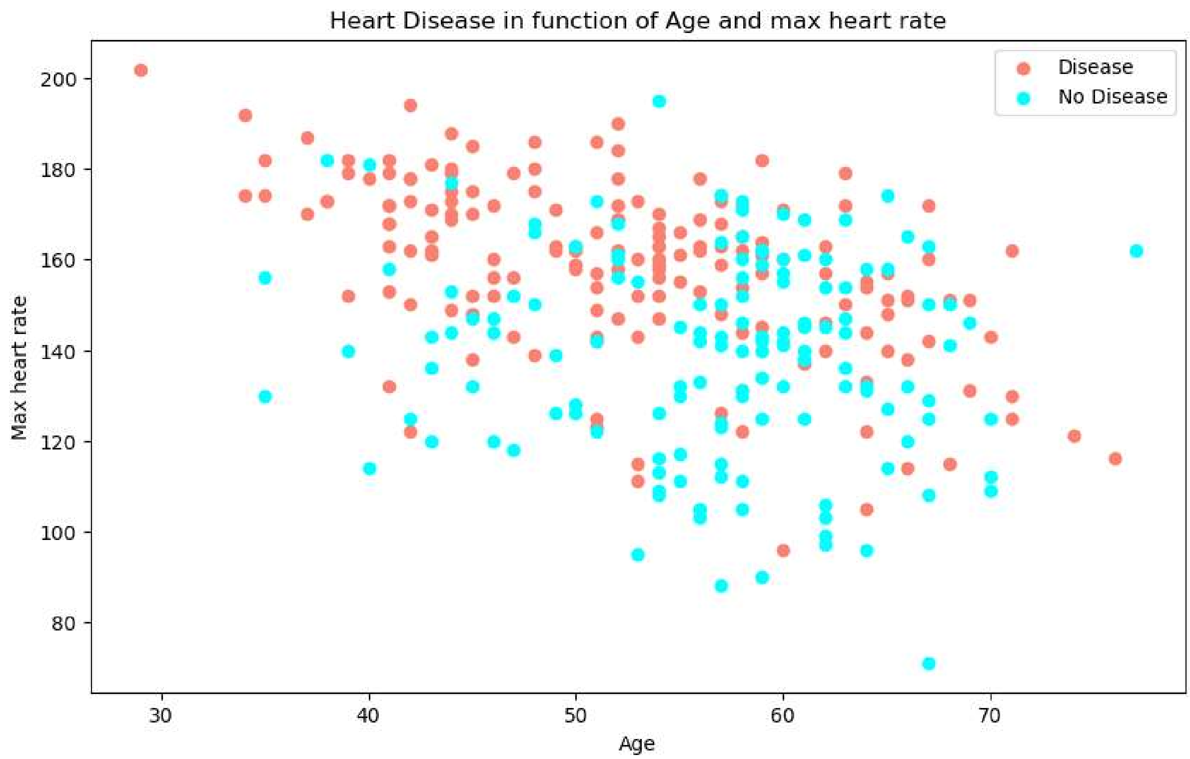
In [10]:
```
# Visualizing the Crosstab
pd.crosstab(df.target,df.sex).plot(kind="bar",
                                   figsize=(10,6),
                                   color=["deeppink","lightblue"]);
plt.title("Heart Disease Frequency for Sex")
plt.xlabel("0= no disease,1= disease")
plt.ylabel("Amount")
plt.legend(["Female","Male"])
plt.xticks(rotation=0);
```



In [11]:
```
### Age vs Max Heart rate for Heart Disease

plt.figure(figsize=(10,6))
# Initialize with positive example
plt.scatter(df.age[df.target==1],
            df.thalach[df.target==1],
            c="salmon")

# now for negative example
plt.scatter(df.age[df.target==0],
            df.thalach[df.target==0],
            c="cyan")
plt.title("Heart Disease in function of Age and max heart rate")
plt.xlabel("Age")
plt.ylabel("Max heart rate")
plt.legend(["Disease","No Disease"]);
```



- **We infer from this that younger ones have higher heart rate compare to olders and age beteween 40 to 65 have positive value of disease**

In [12]:
```
df.head()
```

Out[12]:

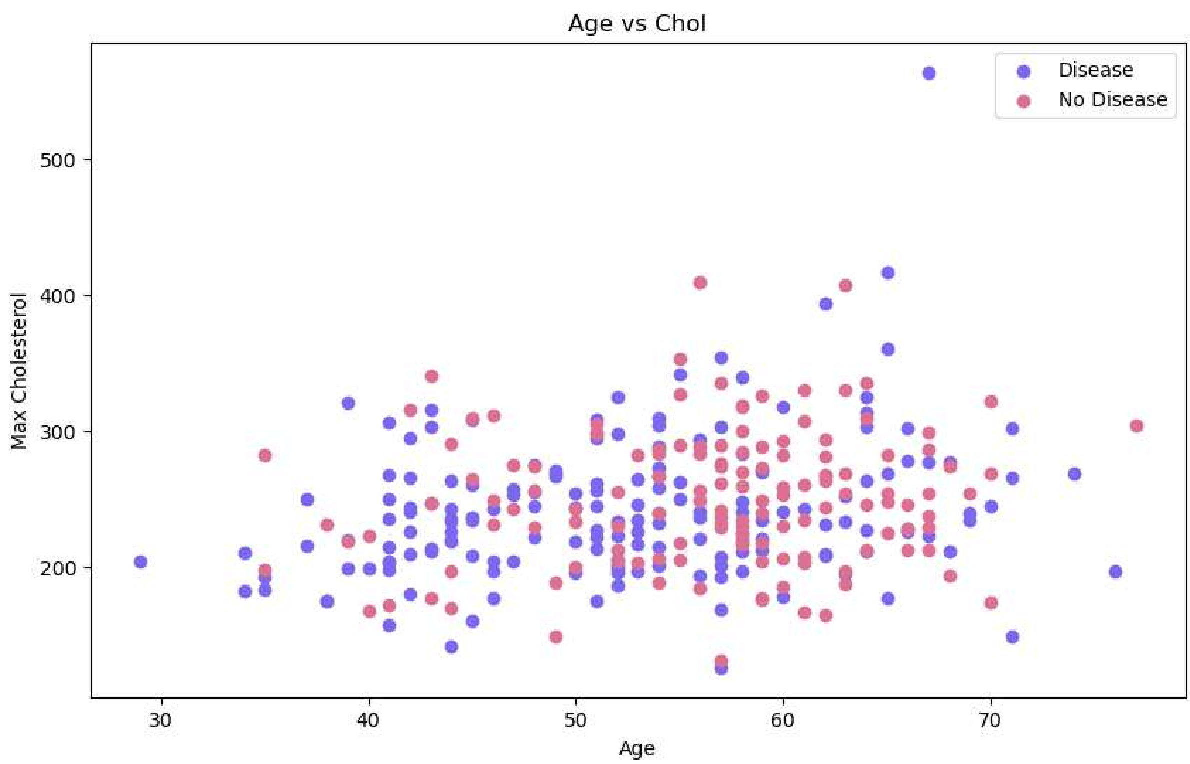
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [13]:
```
df.chol.value_counts()
```

Out[13]:
```
204    6
197    6
234    6
269    5
254    5
      ..
284    1
224    1
167    1
276    1
131    1
Name: chol, Length: 152, dtype: int64
```

In [14]:
```
# Age vs Chol
plt.figure(figsize=(10,6))
# initialize with positive example
plt.scatter(df.age[df.target==1],
            df.chol[df.target==1],
            c="mediumslateblue")
# with negative example
plt.scatter(df.age[df.target==0],
            df.chol[df.target==0],
            c="palevioletred")
plt.title("Age vs Chol")
plt.xlabel("Age")
plt.ylabel("Max Cholesterol")
plt.legend(["Disease","No Disease"])
```

Out[14]:
```
<matplotlib.legend.Legend at 0x1d453c06950>
```



- **Cholesterol levle betwwen 200 - 300 have most positive disease values and age between 50 to 68 have higher cholesterol value and positive disease**

Heart Disease Frequency per Chest pain type

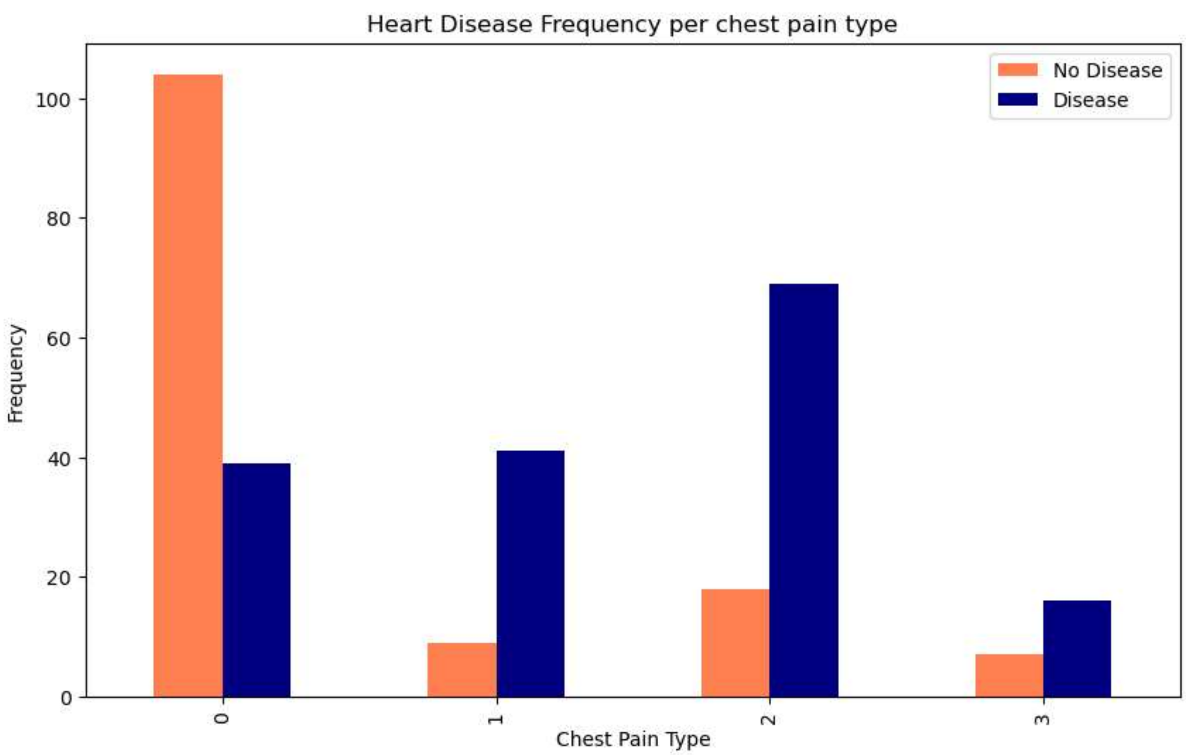
In [15]:
```
pd.crosstab(df.cp,df.target)
```

Out[15]:

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

In [16]:
```
# plotting the above result
pd.crosstab(df.cp,df.target).plot(kind="bar",
                                  figsize=(10,6),
                                  color=["coral","navy"])
plt.title("Heart Disease Frequency per chest pain type")
plt.xlabel("Chest Pain Type")
plt.ylabel("Frequency")
plt.legend(["No Disease","Disease"])
```

Out[16]: <matplotlib.legend.Legend at 0x1d453c9cf90>



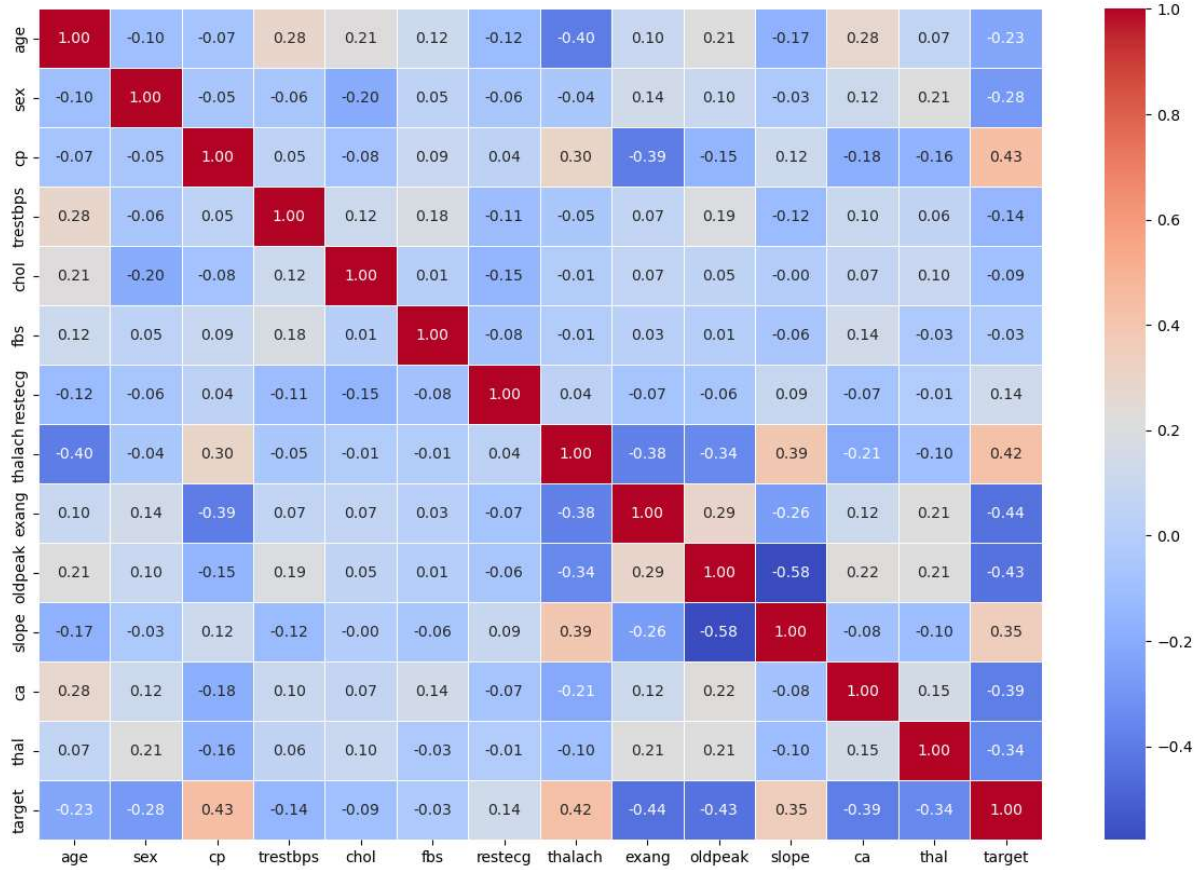
In [17]:
```
# correlation between our independent variables
corr_matrix=df.corr()
corr_matrix
```

Out[17]:

	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	0.
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.
slope	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0
ca	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0
thal	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.
target	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0

In [18]:

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



- **Correlation not giving the clear picture of Higher or lower coorelation**

Modeling

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

Out[19]:

	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 [20]:
```
# Splitting the target variable from rest

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

In [21]:
```
x
```

Out[21]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

303 rows × 13 columns

In [22]:
```
y
```

Out[22]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [23]:
```
# Random seed for reproducibility
np.random.seed(42)

# Splitting into Train & Test set
x_train,x_test,y_train,y_test=train_test_split(x,y,
                                               test_size=0.2# 20 % of data usec
                                              )
```

In [24]:
```
x_train
```

Out[24]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
132	42	1	1	120	295	0	1	162	0	0.0	2	0	2
202	58	1	0	150	270	0	0	111	1	0.8	2	0	3
196	46	1	2	150	231	0	1	147	0	3.6	1	0	2
75	55	0	1	135	250	0	0	161	0	1.4	1	0	2
176	60	1	0	117	230	1	1	160	1	1.4	2	2	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
188	50	1	2	140	233	0	1	163	0	0.6	1	1	3
71	51	1	2	94	227	0	1	154	1	0.0	2	1	3
106	69	1	3	160	234	1	0	131	0	0.1	1	1	2
270	46	1	0	120	249	0	0	144	0	0.8	2	0	3
102	63	0	1	140	195	0	1	179	0	0.0	2	2	2

242 rows × 13 columns

In [25]:
```
x_test
```

Out[25]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
179	57	1	0	150	276	0	0	112	1	0.6	1	1	1
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3
111	57	1	2	150	126	1	1	173	0	0.2	2	1	3
246	56	0	0	134	409	0	0	150	1	1.9	1	2	3
60	71	0	2	110	265	1	0	130	0	0.0	2	1	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
249	69	1	2	140	254	0	0	146	0	2.0	1	3	3
104	50	1	2	129	196	0	1	163	0	0.0	2	0	2
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
193	60	1	0	145	282	0	0	142	1	2.8	1	2	3
184	50	1	0	150	243	0	0	128	0	2.6	1	0	3

61 rows × 13 columns

In [26]:
```
y_test
```

Out[26]:
```
array([0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [27]:
```
# put models in dictionary
models= {"KNN":KNeighborsClassifier(),
         "Logistic Regression":LogisticRegression(),
         "Random Forest":RandomForestClassifier()}
```

In [28]:
```
# Put models in a dictionary
models = {"KNN": KNeighborsClassifier(),
          "Logistic Regression": LogisticRegression(),
          "Random Forest": RandomForestClassifier()}

# Create function to fit and score models
def fit_and_score(models, x_train, x_test, y_train, y_test):
    """
    Fits and evaluates given machine learning models.
    models : a dict of different Scikit-Learn machine learning models
    x_train : training data
    x_test : testing data
    y_train : labels assosciated with training data
    y_test : labels assosciated with test data
    """
    # Random seed for reproducible results
    np.random.seed(42)
    # Make a list to keep model scores
    model_scores = {}
    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(x_train, y_train)
        # Evaluate the model and append its score to model_scores
        model_scores[name] = model.score(x_test, y_test)
    return model_scores
```

In [29]:
```
model_scores = fit_and_score(models=models,
                             x_train=x_train,
                             x_test=x_test,
                             y_train=y_train,
                             y_test=y_test)
model_scores
```

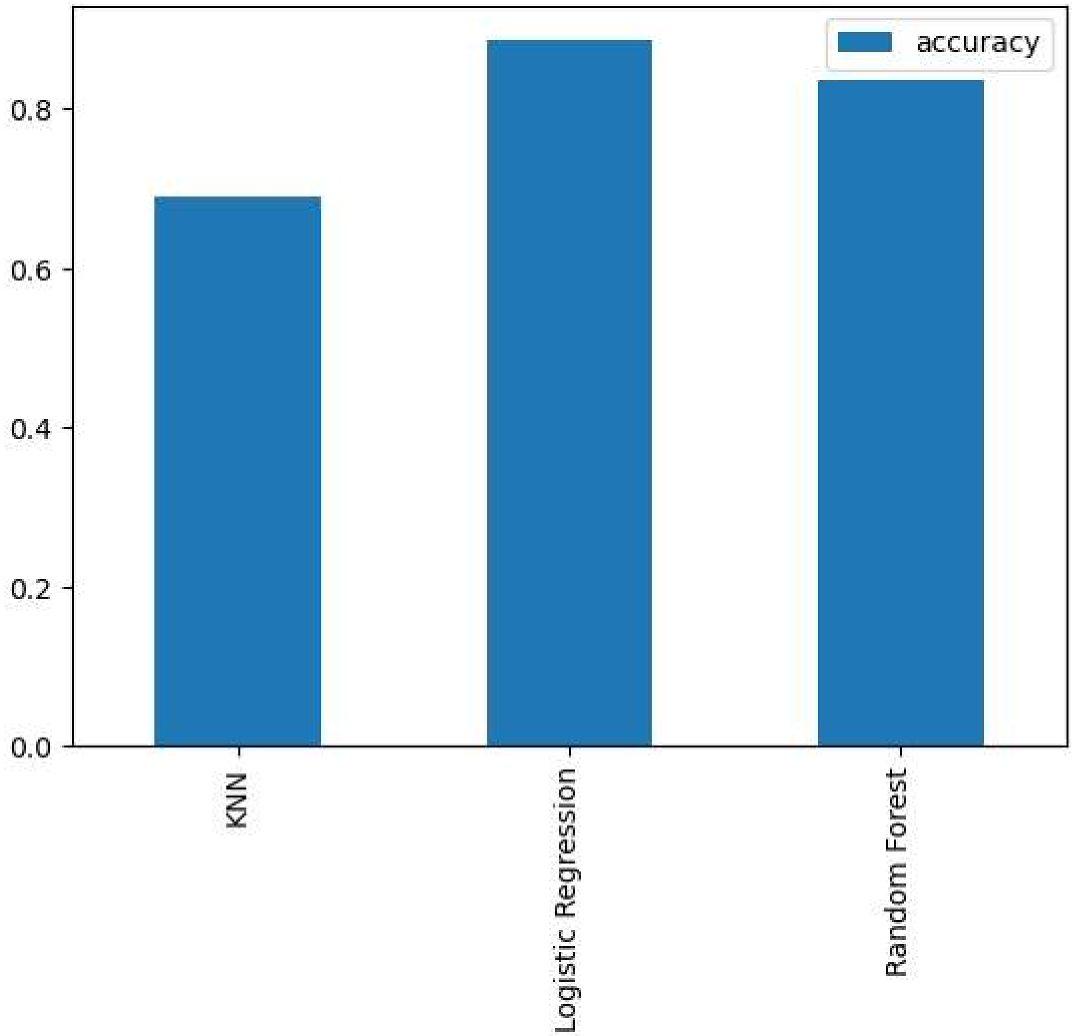
```
C:\Users\RDS Kolkata\Desktop\sample_project\env\Lib\site-packages\sklearn\lin
ear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (sta
tus=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

Out[29]:
```
{'KNN': 0.6885245901639344,
 'Logistic Regression': 0.8852459016393442,
 'Random Forest': 0.8360655737704918}
```

In [64]:
```
# Plotting Model comparison
```

In [31]:
```
model_compare = pd.DataFrame(model_scores, index=['accuracy'])
model_compare.T.plot.bar();
```



Hyperparameter tuning and Cross-Validation

Tuning KNeghborsClassfier(KNN or K-Nearest)

In [32]:
```
# Create a list of train scores
train_scores = []

# Create a list of test scores
test_scores = []

# Create a list of different values for n_neighbors
neighbors = range(1, 25) # 1 to 20

# Setup algorithm
knn = KNeighborsClassifier()

# Loop through different neighbors values
for i in neighbors:
    knn.set_params(n_neighbors = i) # set neighbors value

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

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

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

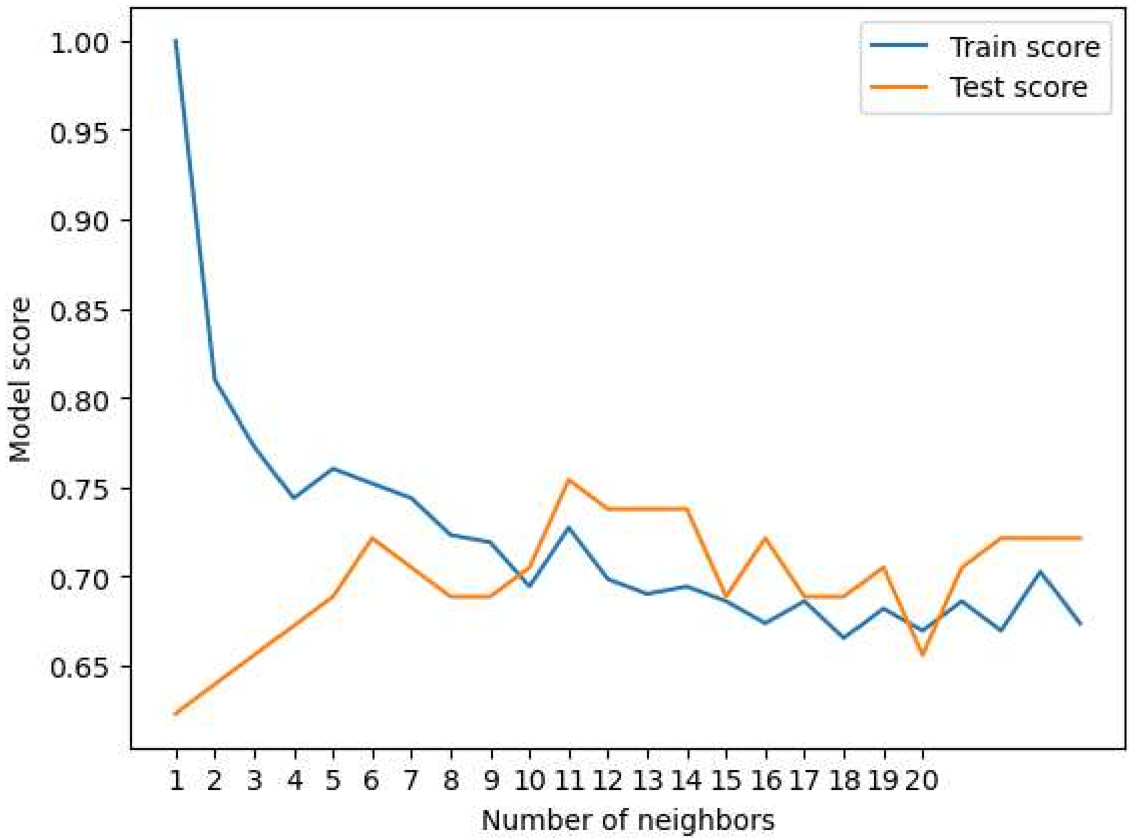
In [33]:
```
train_scores
```

Out[33]:
```
[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,
 0.6859504132231405,
 0.6694214876033058,
 0.7024793388429752,
 0.6735537190082644]
```

In [34]:
```
plt.plot(neighbors, train_scores, label="Train score")
plt.plot(neighbors, test_scores, label="Test score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()

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

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



- **N-neighbors at 11 look best.But though its even below of logistic regression I will discard it.**

Tuning model with RandomizedSearchCV

In [35]:
```
# Different LogisticRegression hyperparameters
log_reg_grid = {"C": np.logspace(-4, 4, 25),
                "solver": ["liblinear"]}

# Different RandomForestClassifier hyperparameters
rf_grid = {"n_estimators": np.arange(10, 1000, 50),
           "max_depth": [None, 3, 5, 10],
           "min_samples_split": np.arange(2, 25, 2),
           "min_samples_leaf": np.arange(1, 25, 2)}
```

We'll pass it the different hyperparameters from log_reg_grid as well as set n_iter = 23

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

# Setup random hyperparameter search for LogisticRegression
rs_log_reg = RandomizedSearchCV(LogisticRegression(),
                                param_distributions=log_reg_grid,
                                cv=5,
                                n_iter=25,
                                verbose=True)

# Fit random hyperparameter search model
rs_log_reg.fit(x_train, y_train);
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
```

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

Out[37]:
```
{'solver': 'liblinear', 'C': 0.21544346900318823}
```

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

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

Tuning with RandomForestClassifier

In [39]:
```
# Setup random seed
np.random.seed(42)

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

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

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

In [40]:
```
# Finding the best parameters
rs_rf.best_params_
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

Out[40]:
```
{'n_estimators': 210,
 'min_samples_split': 4,
 'min_samples_leaf': 19,
 'max_depth': 3}
```

In [42]:
```
# Evaluation of  the randomized search random forest model
rs_rf.score(x_test, y_test)
```

Out[42]:
```
0.8688524590163934
```

- ***Logistic regression is still highest so I will tune it by GridSearchCV***

In [61]:
```
# Different LogisticRegression hyperparameters
log_reg_grid = {"C": np.logspace(-4, 4, 50),
                "solver": ["liblinear"]}

# Setup grid hyperparameter search for LogisticRegression
gs_log_reg = GridSearchCV(LogisticRegression(),
                          param_grid=log_reg_grid,
                          cv=3,
                          verbose=True)

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

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

In [62]:
```
# Checking the best parameters
gs_log_reg.best_params_
```

Out[62]:
```
{'C': 0.3906939937054613, 'solver': 'liblinear'}
```

In [63]:
```
# Evaluating the model
gs_log_reg.score(x_test, y_test)
```

Out[63]:
```
0.8852459016393442
```

- **Hence Final Model is Logistic Regression with 88 % Accuracy**

In [ ]: