In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
df = pd.read_csv('heart.csv')
```

In [3]:
```python
df.head()
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

In [4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

- We can see that there are no null values

Column names and descriptions

- age - age in years

- age - age in years
- sex - (1 = male; 0 = female)
- cp - chest pain type-----> Value 1: typical angina Value 2: atypical angina Value 3: non-anginal pain Value 4: asymptomatic
- trestbps - resting blood pressure (in mm Hg on admission to the hospital)
- chol - serum cholestoral in mg/dl
- fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- restecg - resting electrocardiographic results
- thalach - maximum heart rate achieved
- exang - exercise induced angina (1 = yes; 0 = no)
- oldpeak - ST depression induced by exercise relative to rest
- slope - the slope of the peak exercise ST segment
- ca - number of major vessels (0-3) colored by flourosopy
- thal - 3 = normal; 6 = fixed defect; 7 = reversable defect
- target - have disease or not (1=yes, 0=no)

In [5]:
```
df['target'].value_counts()
```

Out[5]:
```
1    526
0    499
Name: target, dtype: int64
```

we can see that the target column is slightly imbalanced. We can ignore this amount of imbalance

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

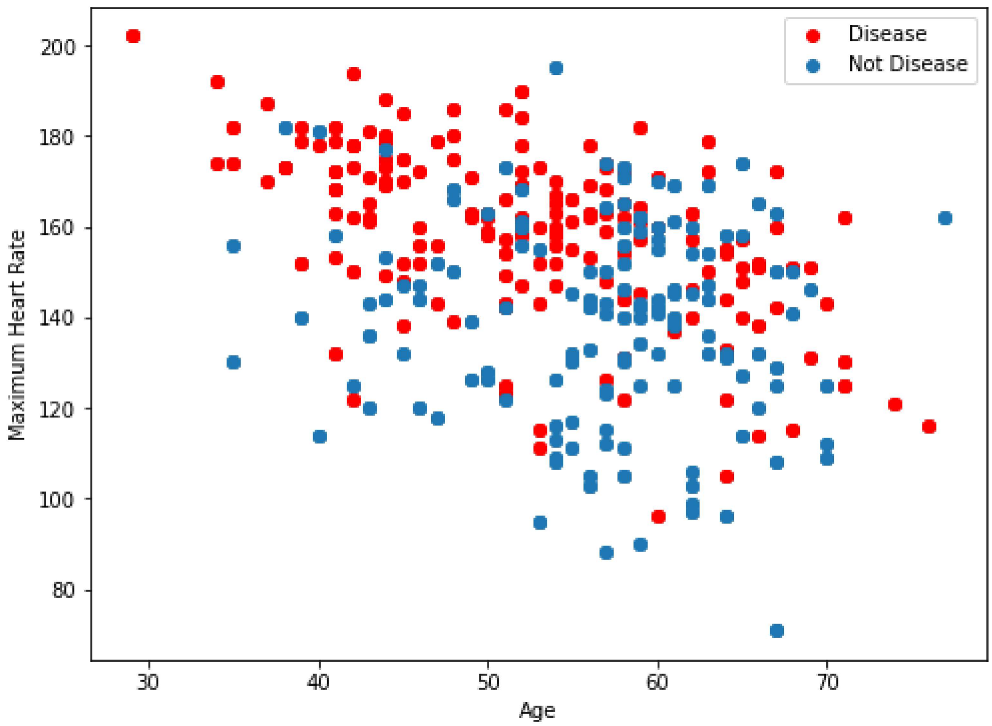
Out[6]:

	age	sex	cp	trestbps	chol	fbs	restec
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.00000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.52975
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.52787
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.00000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.00000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.00000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.00000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.00000

we can see that the median value for most of the features is very close to the mean value. so we can conclude that the data in each feature is normall distributed and does not have any kind of skewness (or to be precise, very minutely skewed)

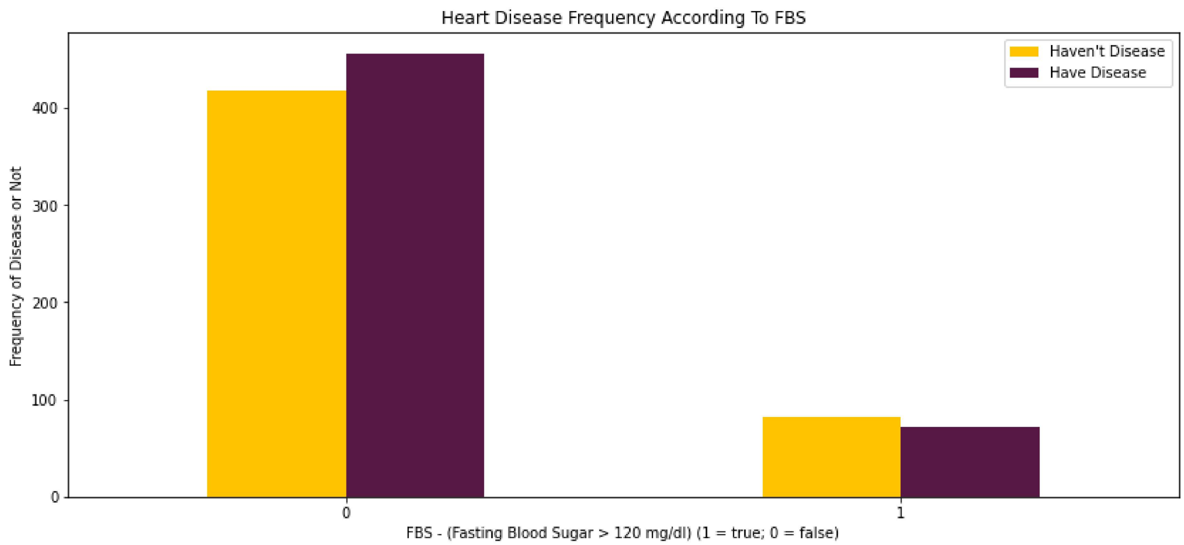
Let us now try to bring some insights out the available data

In [7]:
```
plt.figure(figsize=(8,6))
plt.scatter(x=df.age[df.target==1], y=df.thalach[(df.target==1)], c="red")
plt.scatter(x=df.age[df.target==0], y=df.thalach[(df.target==0)])
plt.legend(["Disease", "Not Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```



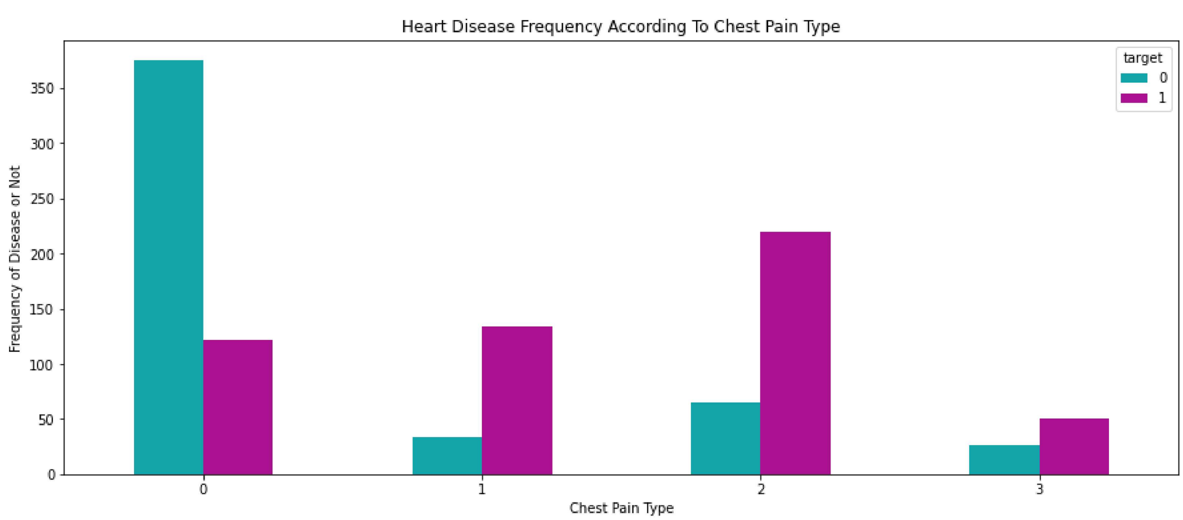
We can see that as the Heart rate increases, the patient is more to be having heart related disease

In [8]:
```
pd.crosstab(df.fbs,df.target).plot(kind="bar",figsize=(14,6),color=['#FFC300',
plt.title('Heart Disease Frequency According To FBS')
plt.xlabel('FBS - (Fasting Blood Sugar > 120 mg/dl) (1 = true; 0 = false)')
plt.xticks(rotation = 0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



It is shocking to see that the population with controlled sugar levels are more prone to heart related diseases as compared to population with diabetes. Maybe, this is because once a person is diagnosed with diabetes, he/she controls the food consumption and starts exercising more regularly and focusses on eating healthy food.

In [9]:
```
pd.crosstab(df.cp,df.target).plot(kind="bar",figsize=(15,6),color=['#11A5AA','
plt.title('Heart Disease Frequency According To Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.xticks(rotation = 0)
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



People with chest pain of type: 1,2,3 are highly prone to heart related diseases

In [10]:
```
#seeing the relation between different parameters
sns.pairplot(df, hue = 'target')
plt.show()
```

In [11]:
```
df.corr()
```

Out[11]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
age	1.000000	-0.103240	-0.071966	0.271121	0.219823	0.121243	-0.132696	-0.390227
sex	-0.103240	1.000000	-0.041119	-0.078974	-0.198258	0.027200	-0.055117	-0.049365
cp	-0.071966	-0.041119	1.000000	0.038177	-0.081641	0.079294	0.043581	0.306839
trestbps	0.271121	-0.078974	0.038177	1.000000	0.127977	0.181767	-0.123794	-0.039264
chol	0.219823	-0.198258	-0.081641	0.127977	1.000000	0.026917	-0.147410	-0.021772
fbs	0.121243	0.027200	0.079294	0.181767	0.026917	1.000000	-0.104051	-0.008866
restecg	-0.132696	-0.055117	0.043581	-0.123794	-0.147410	-0.104051	1.000000	0.048411
thalach	-0.390227	-0.049365	0.306839	-0.039264	-0.021772	-0.008866	0.048411	1.000000
exang	0.088163	0.139157	-0.401513	0.061197	0.067382	0.049261	-0.065606	-0.380281
oldpeak	0.208137	0.084687	-0.174733	0.187434	0.064880	0.010859	-0.050114	-0.349796
slope	-0.169105	-0.026666	0.131633	-0.120445	-0.014248	-0.061902	0.086086	0.395308
ca	0.271551	0.111729	-0.176206	0.104554	0.074259	0.137156	-0.078072	-0.207888
thal	0.072297	0.198424	-0.163341	0.059276	0.100244	-0.042177	-0.020504	-0.098068
target	-0.229324	-0.279501	0.434854	-0.138772	-0.099966	-0.041164	0.134468	0.422895

the pair plot was fairly big to conclude any points. So, checking the correlation value was easier to conclude the impact of each feature on the target.

Splitting the data

In [12]:
```
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [13]:
```
x.shape
```

Out[13]: (1025, 13)

In [14]:
```
y.shape
```

Out[14]: (1025,)

Splitting into training and testing data

In [15]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.2, random_
```

1 - Predicting the data using KNN

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(xtrain, ytrain)
ypred_knn = knn.predict(xtest)
```

- Evaluating the model

In [17]:
```
from sklearn.metrics import accuracy_score, classification_report
```

In [18]:
```
ac_knn = accuracy_score(ytest, ypred_knn)
print(ac_knn)
```

```
0.7463414634146341
```

In [19]:
```
print(classification_report(ytest, ypred_knn))
```

```
              precision    recall  f1-score   support

           0       0.74      0.78      0.76       104
           1       0.76      0.71      0.73       101

    accuracy                           0.75       205
   macro avg       0.75      0.75      0.75       205
weighted avg       0.75      0.75      0.75       205
```

we have achieved an average accuracy of 75 % which isnt that great. Lets see if we can increase this accuracy by hyper tuning

- Hyper Parameter Tuning

In [20]:
```
ac_list_knn = []
for k in range(1,50,1):
    knn_hpt = KNeighborsClassifier(n_neighbors=k)
    knn_hpt.fit(xtrain, ytrain)
    ypred_knn_hpt = knn_hpt.predict(xtest)
    ac = accuracy_score(ytest, ypred_knn_hpt)
    ac_list_knn.append(ac)
```

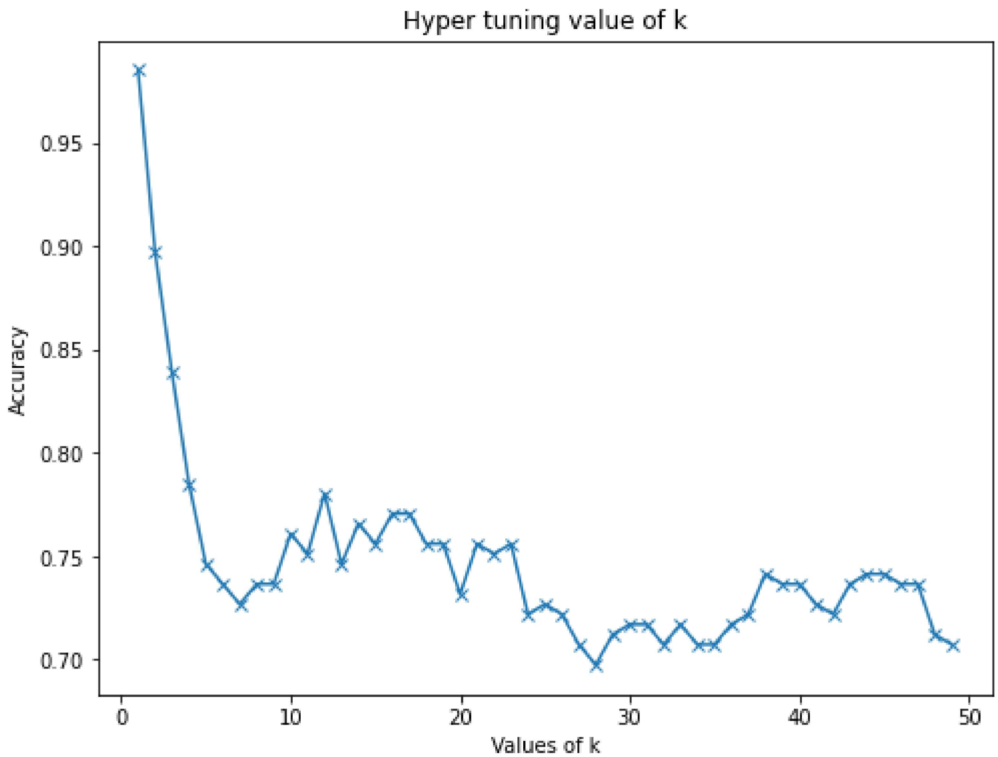
In [21]:
```
ac_list_knn
```

...

In [22]:
```
plt.figure(figsize = (8,6))
plt.plot(range(1,50), ac_list_knn, marker = "x")

plt.title("Hyper tuning value of k")
plt.xlabel("Values of k")
plt.ylabel("Accuracy")

plt.show()
```



at k = 12, we are getting an accuracy of 78 %

In [23]:
```
knn_final = KNeighborsClassifier(n_neighbors = 12)
knn_final.fit(xtrain, ytrain)
ypred_knn_final = knn_final.predict(xtest)
accuracy_score(ytest, ypred_knn_final)
```

Out[23]: 0.7804878048780488

In [24]:
```
knn_final.score(xtrain, ytrain)
```

Out[24]: 0.7585365853658537

In [25]:
```
knn_final.score(xtest, ytest)
```

Out[25]: 0.7804878048780488

From the above values, we can conclude that the model is not over fitting. However it a very low accuracy value since we are dealing with life and death situation in prediction of a disease

In [26]:
```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
```

In [27]:
```
xtrain_mms = mms.fit_transform(xtrain)
```

In [28]:
```
xtest_mms = mms.fit_transform(xtest)
```

In [30]:
```
knn_mms = KNeighborsClassifier(n_neighbors = 12)
knn_mms.fit(xtrain_mms, ytrain)
ypred_mms = knn_mms.predict(xtest_mms)
accuracy_score(ytest, ypred_mms)
```

Out[30]:
```
0.8390243902439024
```

After doing feature scaling, it can be seen that we were able to increase the accuracy score to 83.9 %

2 - Predicting the data using Logistic Regression Classifier

In [31]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(xtrain, ytrain)
ypred_logreg = logreg.predict(xtest)
```

- Evaluating the model

In [32]:
```
ac_logreg = accuracy_score(ytest, ypred_logreg)
print(ac_logreg)
```

```
0.8780487804878049
```

Using Logistic regression classifier we are able to achieve an accuracy score of 87.8 %. Lets see if the scaling of the values increases the accuracy or not

In [34]:
```
logreg.fit(xtrain_mms, ytrain)
ypred_mms_logreg = logreg.predict(xtest)
ac_mms_logreg = accuracy_score(ytest, ypred_mms_logreg)
```

In [35]:
```
print(ac_mms_logreg)
```

```
0.5121951219512195
```

3 - Predicting the data using Support Vector Machines

In [36]:
```
from sklearn.svm import SVC
svm = SVC()
svm.fit(xtrain, ytrain)
ypred_svm = svm.predict(xtest)
acc_svm = accuracy_score(ytest, ypred_svm)
```

In [37]:
```
print(acc_svm)
```

```
0.7024390243902439
```

In [38]:
```
svm_1 = SVC(kernel = 'linear')
svm_1.fit(xtrain, ytrain)
ypred_svm_1 = svm_1.predict(xtest)
acc_svm_1 = accuracy_score(ytest, ypred_svm_1)
print(acc_svm_1)
```

```
0.8780487804878049
```

In [39]:
```
svm_2 = SVC(kernel = 'poly')
svm_2.fit(xtrain, ytrain)
ypred_svm_2 = svm_2.predict(xtest)
acc_svm_2 = accuracy_score(ytest, ypred_svm_2)
print(acc_svm_2)
```

```
0.7024390243902439
```

In [40]:
```
svm_3 = SVC(kernel = 'sigmoid')
svm_3.fit(xtrain, ytrain)
ypred_svm_3 = svm_3.predict(xtest)
acc_svm_3 = accuracy_score(ytest, ypred_svm_3)
print(acc_svm_3)
```

```
0.5317073170731708
```

we will go ahead with SVM with Linear kernel to get the highest accuracy

4 - Predicting the data using Decision Trees

In [41]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(xtrain, ytrain)
ypred_dt = dt.predict(xtest)
acc_dt = accuracy_score(ytest, ypred_dt)
```

In [42]:
```
print(acc_dt)
```

```
1.0
```

In [43]:
```
print(classification_report(ytest, ypred_dt))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       104
           1       1.00      1.00      1.00       101

    accuracy                           1.00       205
   macro avg       1.00      1.00      1.00       205
weighted avg       1.00      1.00      1.00       205
```

In [44]:
```
dt.score(xtrain, ytrain)
```

Out[44]:
```
1.0
```

In [45]:
```
dt.score(xtest, ytest)
```

Out[45]:
```
1.0
```

Use of Decision Tree will lead to an overfit model

Conclusion

Based on the above accuracy scores, we should go ahead with Support Vector Machines algorithm with Linear Kernel

In [ ]:
```

```