Introduction

Dataset - heart.csv

Predictions - patients have heart disease(1) or not(0)

In [1]:

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

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

In [2]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import precision_score,recall_score,accuracy_score
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

In [3]:

```
df = pd.read_csv("../input/heart.csv")
```

In [4]:

```
df.head()
```

Out[4]:

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

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
target      303 non-null int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

EDA

In [6]:

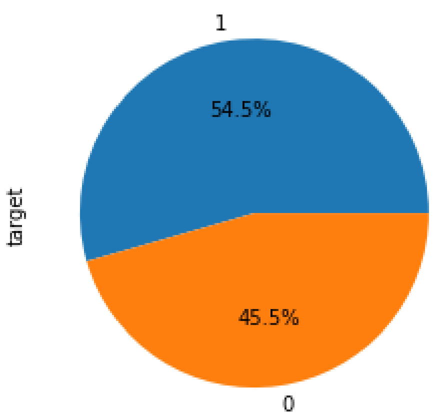
```
df.target.value_counts()
```

Out[6]:

```
1    165
0    138
Name: target, dtype: int64
```

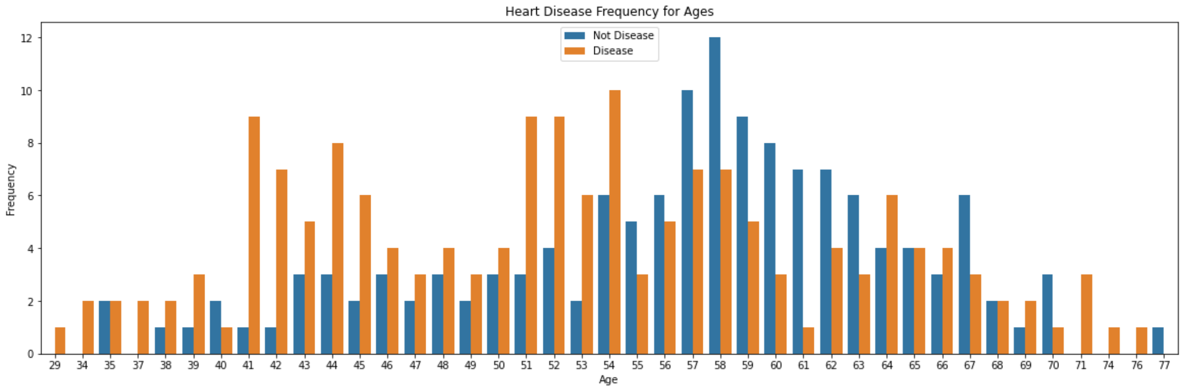
In [6]:

```
plt.figure()
df['target'].value_counts().plot(kind='pie',autopct='%1.1f%%')
plt.show()
```



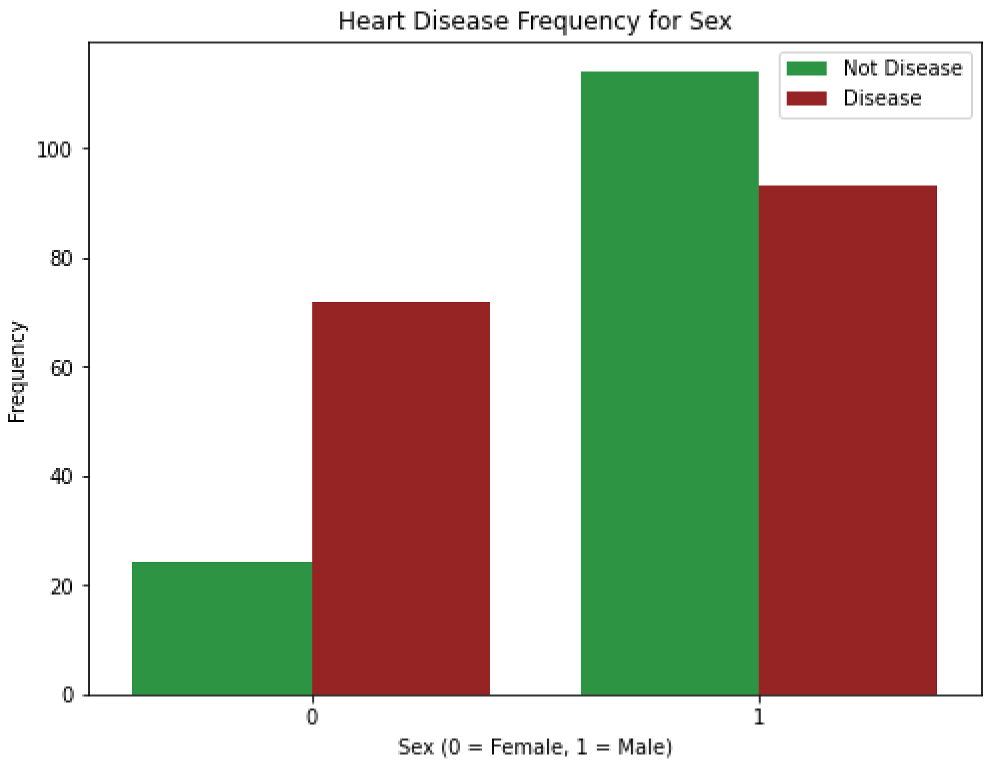
In [14]:

```
plt.figure(figsize=(20,6))
sns.countplot(data=df,x='age',hue='target')
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.legend([ "Not Disease","Disease"])
plt.show()
```



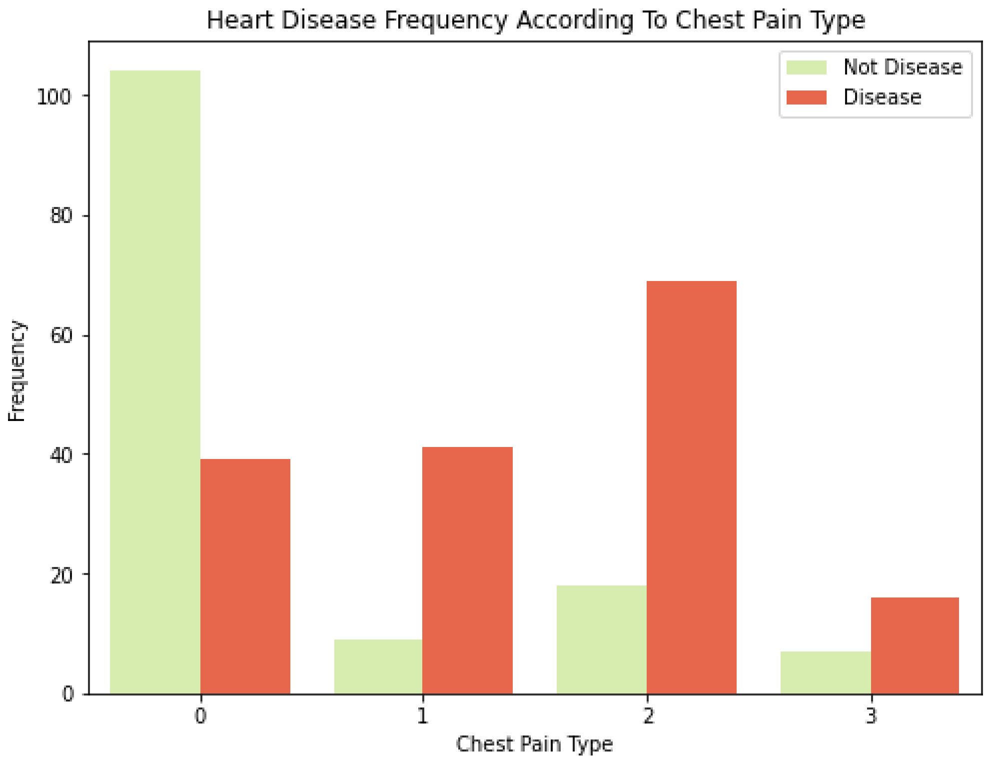
In [20]:

```
plt.figure(figsize=(8,6))
sns.countplot(data=df,x='sex',hue='target',palette=['#1CA53B','#AA1111' ])
plt.title('Heart Disease Frequency for Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.ylabel('Frequency')
plt.legend([ "Not Disease","Disease"])
plt.show()
```



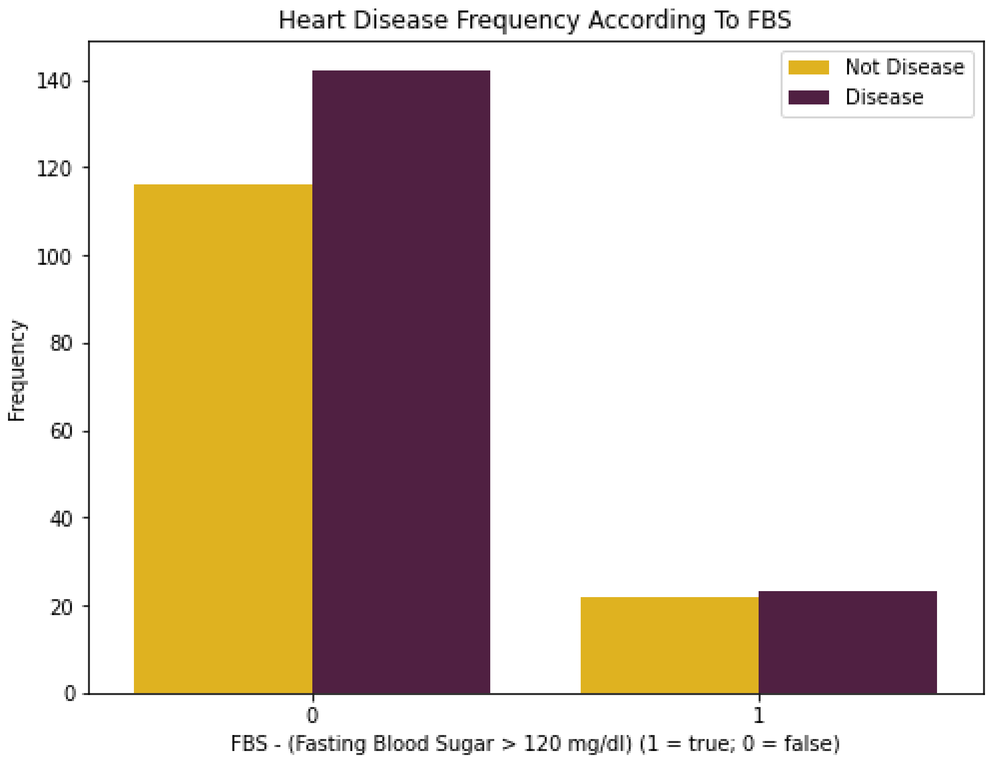
In [21]:

```
plt.figure(figsize=(8,6))
sns.countplot(data=df,x='cp',hue='target',palette=['#DAF7A6','#FF5733' ])
plt.title('Heart Disease Frequency According To Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.ylabel('Frequency')
plt.legend([ "Not Disease","Disease"])
plt.show()
```



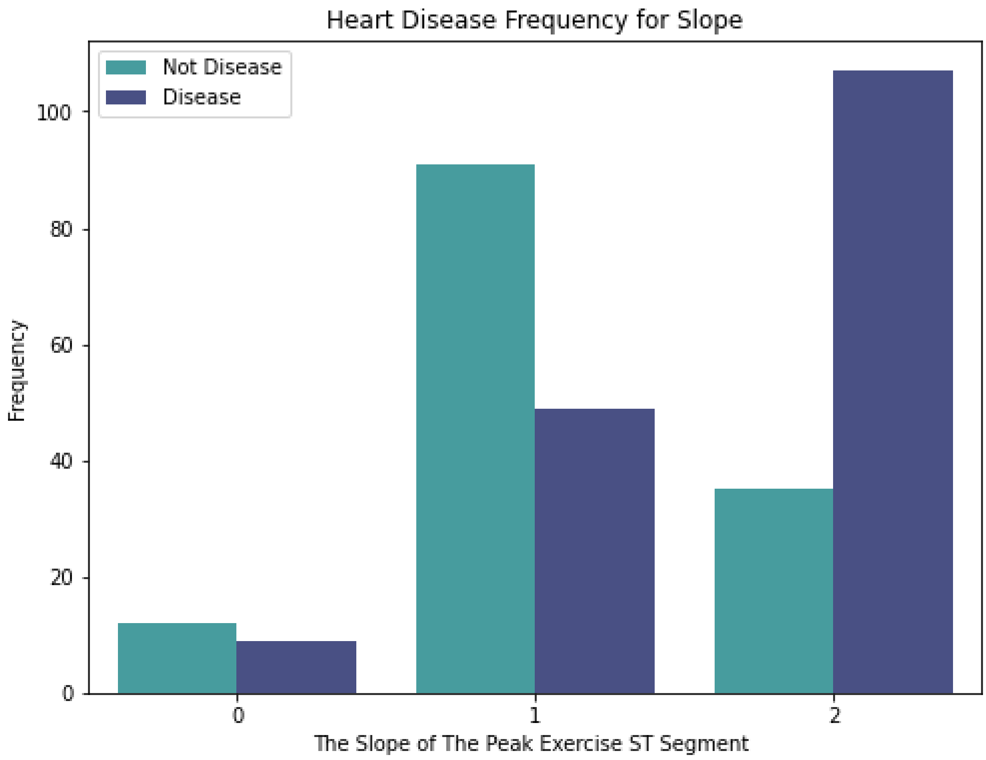
In [23]:

```
plt.figure(figsize=(8,6))
sns.countplot(data=df,x='fbs',hue='target',palette=['#FFC300','#581845' ])
plt.title('Heart Disease Frequency According To FBS')
plt.xlabel('FBS - (Fasting Blood Sugar > 120 mg/dl) (1 = true; 0 = false)')
plt.ylabel('Frequency')
plt.legend([ "Not Disease","Disease"])
plt.show()
```



In [22]:

```
plt.figure(figsize=(8,6))
sns.countplot(data=df,x='slope',hue='target',palette='mako_r')
plt.title('Heart Disease Frequency for Slope')
plt.xlabel('The Slope of The Peak Exercise ST Segment ')
plt.ylabel('Frequency')
plt.legend([ "Not Disease","Disease"])
plt.show()
```



In [34]:

```
plt.figure(figsize=(8,6))
sns.scatterplot(data=df,x='age',y='thalach',hue='target',palette=['#AA1111','#1CA53B'])
plt.title('Heart Disease Frequency for Slope')
plt.xlabel("Age")

plt.ylabel("Maximum Heart Rate")
plt.legend([ "Not Disease","Disease"])
plt.show()
```

In [27]:

```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [28]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=1)
```

In [29]:

```
accuracies = []
precision = []
recall = []
```

In [30]:

```
def create_model(model):
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    accuracies.append((accuracy_score(y_test,y_pred))*100)
    precision.append((precision_score(y_test,y_pred))*100)
    recall.append((recall_score(y_test,y_pred))*100)
    print(classification_report(y_test,y_pred))
    return model
```

Baseline model

In [31]:

```
log = LogisticRegression()
create_model(log)
```

```
              precision    recall  f1-score   support

           0       0.81      0.73      0.77        41
           1       0.80      0.86      0.83        50

    accuracy                           0.80        91
   macro avg       0.80      0.80      0.80        91
weighted avg       0.80      0.80      0.80        91
```

Out[31]:

```
LogisticRegression()
```

Random Forest

In [32]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [33]:

```
rf = RandomForestClassifier(n_estimators=200,random_state=1,min_samples_leaf=50)
create_model(rf)
```

```
              precision    recall  f1-score   support

           0       0.83      0.71      0.76        41
           1       0.79      0.88      0.83        50

    accuracy                           0.80        91
   macro avg       0.81      0.79      0.80        91
weighted avg       0.81      0.80      0.80        91
```

Out[33]:

```
RandomForestClassifier(min_samples_leaf=50, n_estimators=200, random_state=
1)
```

KNN

Standardize the Variables

In [34]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

In [35]:

```
scaler = StandardScaler()
```

In [36]:

```
scaler.fit(df.drop('target',axis=1))
```

Out[36]:

```
StandardScaler()
```

In [37]:

```
scaled_features = scaler.transform(df.drop('target',axis=1))
```

In [38]:

```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=1)
```

Choosing a K Value

In [39]:

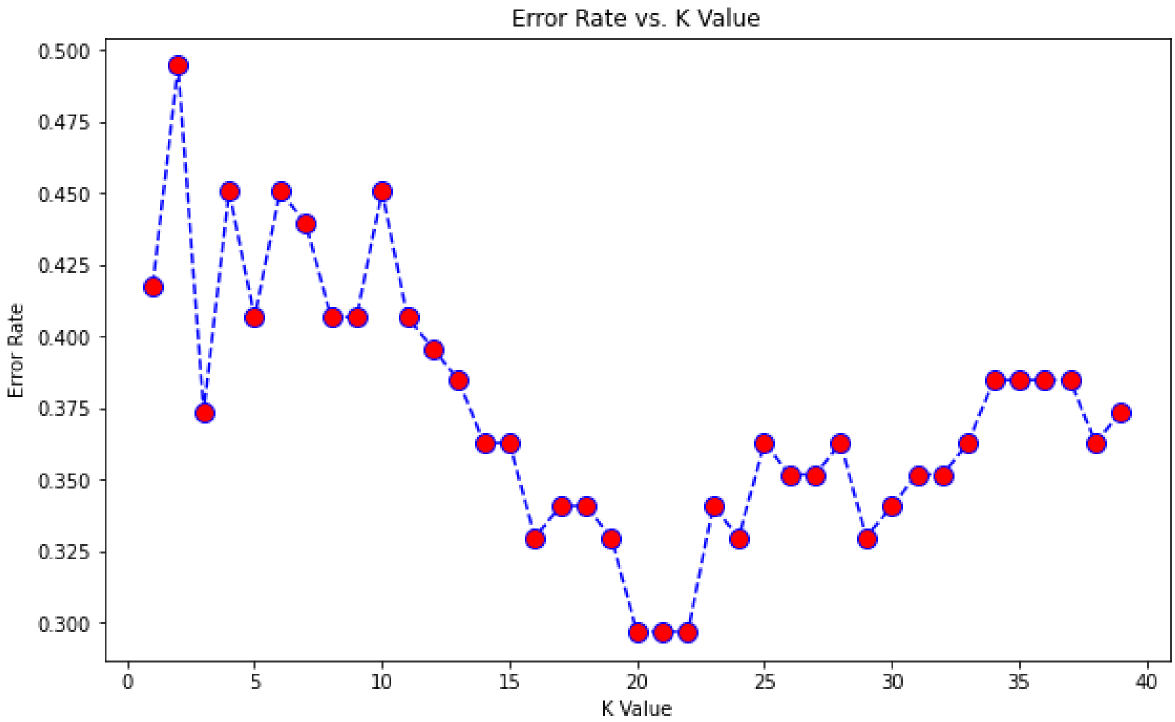
```
error_rate = []

for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_test)
    error_rate.append(np.mean(y_pred != y_test))
```

In [40]:

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K Value')
plt.ylabel('Error Rate')
```

Out[40]:

```
Text(0, 0.5, 'Error Rate')
```



In [41]:

```
knn = KNeighborsClassifier(n_neighbors=20)
create_model(knn)
```

```
              precision    recall  f1-score   support

           0       0.66      0.71      0.68        41
           1       0.74      0.70      0.72        50

    accuracy                           0.70        91
   macro avg       0.70      0.70      0.70        91
weighted avg       0.71      0.70      0.70        91
```

Out[41]:

```
KNeighborsClassifier(n_neighbors=20)
```

SVM Models

1.Simple Linear SVM

In [42]:

```
# Hard margin
svc1 = LinearSVC(random_state=1)
create_model(svc1)
```

```
              precision    recall  f1-score   support

           0       0.85      0.68      0.76        41
           1       0.78      0.90      0.83        50

    accuracy                           0.80        91
   macro avg       0.81      0.79      0.80        91
weighted avg       0.81      0.80      0.80        91
```

Out[42]:

```
LinearSVC(random_state=1)
```

In [43]:

```
# Soft Margin
svc2 = LinearSVC(random_state=1,C=0.5)
create_model(svc2)
```

```
              precision    recall  f1-score   support

           0       0.85      0.68      0.76        41
           1       0.78      0.90      0.83        50

    accuracy                           0.80        91
   macro avg       0.81      0.79      0.80        91
weighted avg       0.81      0.80      0.80        91
```

Out[43]:

```
LinearSVC(C=0.5, random_state=1)
```

SVM Kernel Trick

1. Polynomial

In [44]:

```
poly_svc = SVC(random_state=1,kernel="poly",C=0.4)
create_model(poly_svc)
```

```
              precision    recall  f1-score   support

           0       0.59      0.46      0.52        41
           1       0.63      0.74      0.68        50

    accuracy                           0.62        91
   macro avg       0.61      0.60      0.60        91
weighted avg       0.61      0.62      0.61        91
```

Out[44]:

```
SVC(C=0.4, kernel='poly', random_state=1)
```

2. Radial Bais

In [45]:

```
radial_svc = SVC(random_state=1,kernel="rbf")
create_model(radial_svc)
```

```
              precision    recall  f1-score   support

           0       0.59      0.46      0.52        41
           1       0.63      0.74      0.68        50

    accuracy                           0.62        91
   macro avg       0.61      0.60      0.60        91
weighted avg       0.61      0.62      0.61        91
```

Out[45]:

```
SVC(random_state=1)
```

Grid Search

In [46]:

```
param_grid = {'C':[0.1,1,10,100,1000],'gamma':[1,0.1,0.01,0.001,0.0001]}
```

In [47]:

```
grid = GridSearchCV(SVC(),param_grid,verbose=3)
```

In [48]:

```
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.535, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.535, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.548, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.548, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.548, total=   0.0s
[CV] C=0.1, gamma=0.1 ................................................
[CV] .................... C=0.1, gamma=0.1, score=0.535, total=   0.0s
[CV] C=0.1, gamma=0.1 ................................................
[CV] .................... C=0.1, gamma=0.1, score=0.535, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
```

In [49]:

```
grid.best_params_
```

Out[49]:

```
{'C': 1000, 'gamma': 0.0001}
```

In [50]:

```
grid.best_estimator_
```

Out[50]:

```
SVC(C=1000, gamma=0.0001)
```

In [51]:

```
create_model(grid)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.535, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.535, total=   0.1s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.548, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.548, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.548, total=   0.0s
[CV] C=0.1, gamma=0.1 ................................................
[CV] .................... C=0.1, gamma=0.1, score=0.535, total=   0.0s
[CV] C=0.1, gamma=0.1 ................................................
[CV] .................... C=0.1, gamma=0.1, score=0.535, total=   0.0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wo
rkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
```

Comparing Models

In [52]:

```
accuracies
```

Out[52]:

```
[80.21978021978022,
 80.21978021978022,
 70.32967032967034,
 80.21978021978022,
 80.21978021978022,
 61.53846153846154,
 61.53846153846154,
 72.52747252747253]
```

In [53]:

```
precision
```

Out[53]:

```
[79.62962962962963,
 78.57142857142857,
 74.46808510638297,
 77.58620689655173,
 77.58620689655173,
 62.71186440677966,
 62.71186440677966,
 75.51020408163265]
```

In [54]:

```
recall
```

Out[54]:

```
[86.0, 88.0, 70.0, 90.0, 90.0, 74.0, 74.0, 74.0]
```

In [55]:

```
models = ["Logistic Regression","Random Forest","KNN","Linear SVM Hard Margin","Linear SVM
values = [precision,recall,accuracies]
result = {}
```

In [56]:

```
values
```

Out[56]:

```
[[79.62962962962963,
  78.57142857142857,
  74.46808510638297,
  77.58620689655173,
  77.58620689655173,
  62.71186440677966,
  62.71186440677966,
  75.51020408163265],
 [86.0, 88.0, 70.0, 90.0, 90.0, 74.0, 74.0, 74.0],
 [80.21978021978022,
  80.21978021978022,
  70.32967032967034,
  80.21978021978022,
  80.21978021978022,
  61.53846153846154,
  61.53846153846154,
  72.52747252747253]]
```

In [57]:

```
def Extract(values,x):
    return ([i[x] for i in values])
```

In [58]:

```
x = 0
for key in models:
    result[key] = Extract(values,x)
    x += 1
```

In [59]:

```
result
```

Out[59]:

```
{'Logistic Regression': [79.62962962962963, 86.0, 80.21978021978022],
 'Random Forest': [78.57142857142857, 88.0, 80.21978021978022],
 'KNN': [74.46808510638297, 70.0, 70.32967032967034],
 'Linear SVM Hard Margin': [77.58620689655173, 90.0, 80.21978021978022],
 'Linear SVM Soft Margin': [77.58620689655173, 90.0, 80.21978021978022],
 'Polynomial Kernel': [62.71186440677966, 74.0, 61.53846153846154],
 'Radial Bias Kernel': [62.71186440677966, 74.0, 61.53846153846154],
 'Grid Seach': [75.51020408163265, 74.0, 72.52747252747253]}
```

In [62]:

```
cost_function = ["Precision","Recall","Accuracy"]
score_df = pd.DataFrame(result,index=cost_function)
score_df
```

Out[62]:

	Logistic Regression	Random Forest	KNN	Linear SVM Hard Margin	Linear SVM Soft Margin	Polynomial Kernel	Radial Bias Kernel	S
Precision	79.62963	78.571429	74.468085	77.586207	77.586207	62.711864	62.711864	75.51
Recall	86.00000	88.000000	70.000000	90.000000	90.000000	74.000000	74.000000	74.00
Accuracy	80.21978	80.219780	70.329670	80.219780	80.219780	61.538462	61.538462	72.52

Conclusion: Our models work fine but best of them are Linear SVM Soft and Hard margin ,KNN and Random Forest with high accuracy,precision and recall.

In [ ]: