Importing Libraries

In [1]:

```
%matplotlib inline
import numpy as np
import scipy as sp
import matplotlib as mpl
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import pandas as pd
#from pandas.tools.plotting import scatter_matrix
pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 100)
pd.set_option('display.notebook_repr_html', True)
import seaborn as sns
sns.set(style="whitegrid")
import warnings
warnings.filterwarnings('ignore')
import string
import math
import sys
from sklearn import preprocessing

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import sklearn
from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast_node_interactivity = "all"

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score ,precision_score, recall_scor
e,f1_score

from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegressionCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier






from astropy.table import Table, Column
import matplotlib.pyplot as plt
```

Importing Data Set

In [2]:

```
import pandas as pd
```

In [3]:

```
train = pd.read_csv("Titanic Disaster Dataset.csv")
```

In [4]:

```
train
```

Out[4]:

	PassengerId	PClass	Gender	Sibling	Embarked	Survived
0	1	3	male	1	S	0
1	2	1	female	1	C	1

2	3	3	female	0	S	1
3	4	1	female	1	S	1
4	5	3	male	0	S	0
5	6	3	male	0	Q	0
6	7	1	male	0	S	0
7	8	3	male	3	S	0
8	9	3	female	0	S	1
9	10	2	female	1	C	1
10	11	3	female	1	S	1
11	12	1	female	0	S	1
12	13	3	male	0	S	0
13	14	3	male	1	S	0
14	15	3	female	0	S	0
15	16	2	female	0	S	1
16	17	3	male	4	Q	0
17	18	2	male	0	S	1
18	19	3	female	1	S	0
19	20	3	female	0	C	1
20	21	2	male	0	S	0
21	22	2	male	0	S	1
22	23	3	female	0	Q	1
23	24	1	male	0	S	1
24	25	3	female	3	S	0
25	26	3	female	1	S	1
26	27	3	male	0	C	0
27	28	1	male	3	S	0
28	29	3	female	0	Q	1
29	30	3	male	0	S	0
...	...	...	...	...	...	...
1279	1280	3	male	0	Q	0
1280	1281	3	male	3	S	0
1281	1282	1	male	0	S	0
1282	1283	1	female	0	S	1
1283	1284	3	male	0	S	0
1284	1285	2	male	0	S	0
1285	1286	3	male	3	S	0
1286	1287	1	female	1	S	1
1287	1288	3	male	0	Q	0
1288	1289	1	female	1	C	1
1289	1290	3	male	0	S	0
1290	1291	3	male	0	Q	0
1291	1292	1	female	0	S	1
1292	1293	2	male	1	S	0
1293	1294	1	female	0	C	1
1294	1295	1	male	0	S	0
1295	1296	1	male	1	C	0
1296	1297	2	male	0	C	0
1297	1298	2	male	1	S	0
1298	1299	1	male	1	C	0
1299	1300	3	female	0	Q	1
1300	1301	3	female	1	S	1
1301	1302	3	female	0	Q	1
1302	1303	1	female	1	Q	1
1303	1304	3	female	0	S	1
1304	1305	3	male	0	S	0

1305	1306	1	female	0	C	1
1306	1307	3	male	0	S	0
1307	1308	3	male	0	S	0
1308	1309	3	male	1	C	0

1309 rows × 6 columns

In [5]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 6 columns):
PassengerId    1309 non-null int64
PClass         1309 non-null int64
Gender         1309 non-null object
Sibling        1309 non-null int64
Embarked       1307 non-null object
Survived       1309 non-null int64
dtypes: int64(4), object(2)
memory usage: 61.4+ KB
```

COUNT

In [6]:

```
train.count()
```

Out[6]:

```
PassengerId    1309
PClass         1309
Gender         1309
Sibling        1309
Embarked       1307
Survived       1309
dtype: int64
```

Missing values of attributes

In [7]:

```
print('Train columns with numm values:\n' , train.isnull().sum()  )
print("-"*42)
```

```
Train columns with numm values:
 PassengerId    0
PClass         0
Gender         0
Sibling        0
Embarked       2
Survived       0
dtype: int64
------------------------------------------
```

A handy code train.describe(include='all') describes the whole data set in terms of count, the unique values, mean and etc. But it may not be very meaningful now since there are a lot of missing values. Also, some categorical variables are not useful at all. This code is just for your information - which may come in handy when you do other projects.

In [8]:

```
train.describe(include = 'all')
```

Out[8]:

	PassengerId	PClass	Gender	Sibling	Embarked	Survived
count	1309.000000	1309.000000	1309	1309.000000	1307	1309.000000
unique	NaN	NaN	2	NaN	3	NaN
top	NaN	NaN	male	NaN	S	NaN
freq	NaN	NaN	843	NaN	914	NaN
mean	655.000000	2.294882	NaN	0.498854	NaN	0.377387
std	378.020061	0.837836	NaN	1.041658	NaN	0.484918
min	1.000000	1.000000	NaN	0.000000	NaN	0.000000
25%	328.000000	2.000000	NaN	0.000000	NaN	0.000000
50%	655.000000	3.000000	NaN	0.000000	NaN	0.000000
75%	982.000000	3.000000	NaN	1.000000	NaN	1.000000
max	1309.000000	3.000000	NaN	8.000000	NaN	1.000000

Exploratory Data Analysis

Survival Count

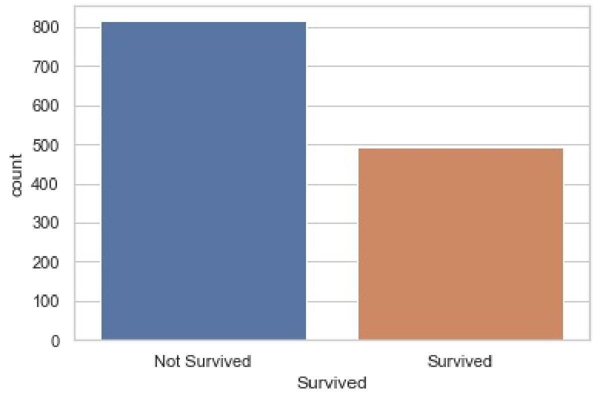
In [9]:

```
print("Total n.o of Survived and not survived in train data set")
survived_bar=sns.countplot("Survived",data=train)
survived_bar.set_xticklabels(['Not Survived', 'Survived'])
```

```
Total n.o of Survived and not survived in train data set
```

Out[9]:

```
[Text(0, 0, 'Not Survived'), Text(0, 0, 'Survived')]
```



Passenger Class Count

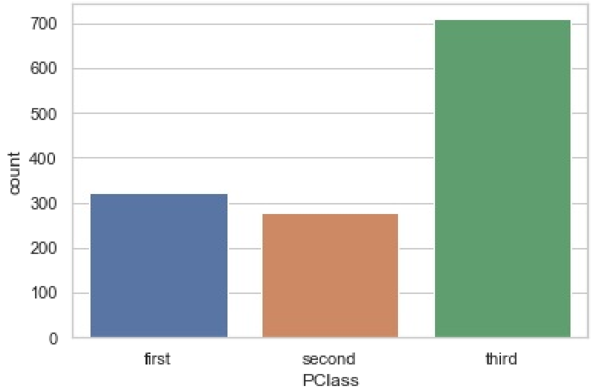
In [10]:

```
print("Total n.o of Classes in train data set")
Pclass=sns.countplot("PClass",data=train)
Pclass.set_xticklabels(['first', 'second', 'third'])
```

```
Total n.o of Classes in train data set
```

Out[10]:

```
[Text(0, 0, 'first'), Text(0, 0, 'second'), Text(0, 0, 'third')]
```



Siblings Count

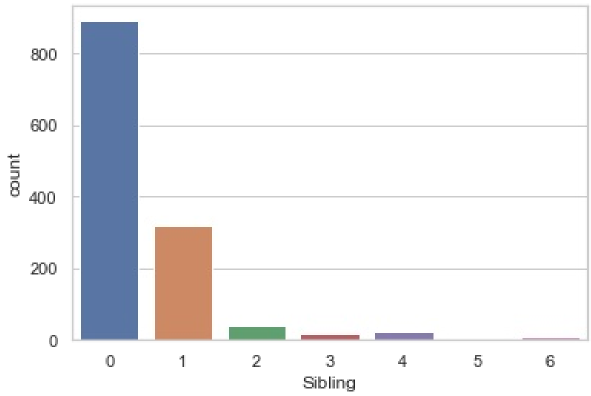
In [11]:

```
print("Total n.o of Siblings in train data set")
Sib_bar=sns.countplot("Sibling",data=train)
Sib_bar.set_xticklabels(['0', '1', '2','3','4','5','6'])
```

```
Total n.o of Siblings in train data set
```

Out[11]:

```
[Text(0, 0, '0'),
 Text(0, 0, '1'),
 Text(0, 0, '2'),
 Text(0, 0, '3'),
 Text(0, 0, '4'),
 Text(0, 0, '5'),
 Text(0, 0, '6')]
```



Embarked Count

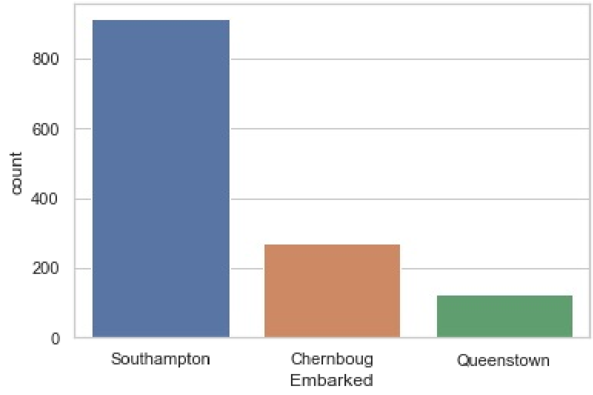
In [12]:

```
print("Total n.o of Classes in train data set")
Emb_bar=sns.countplot("Embarked",data=train)
Emb_bar.set_xticklabels(['Southampton', 'Chernboug', 'Queenstown'])
```

```
Total n.o of Classes in train data set
```

Out[12]:

```
[Text(0, 0, 'Southampton'), Text(0, 0, 'Chernboug'), Text(0, 0, 'Queenstown')]
```



Gender Count

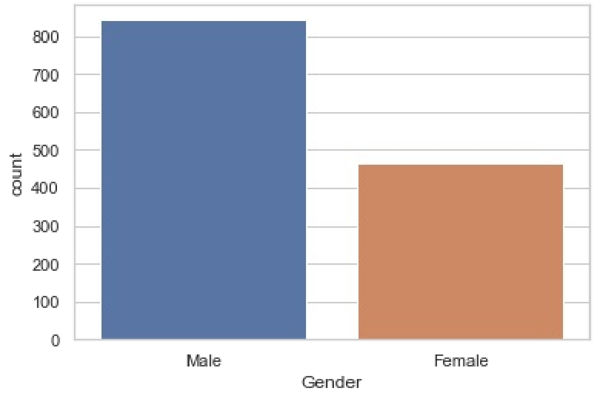
In [13]:

```
print("Total n.o of Classes in train data set")
gender_bar=sns.countplot("Gender",data=train)
gender_bar.set_xticklabels(['Male', 'Female'])
```

```
Total n.o of Classes in train data set
```

Out[13]:

```
[Text(0, 0, 'Male'), Text(0, 0, 'Female')]
```



Treating missing values in Embarked before Label Encoding

In [14]:

```
print('Amount of missing data in Embarked for train:', train.Embarked.isnull().sum())
```

```
Amount of missing data in Embarked for train: 2
```

In [15]:

```
train[train['Embarked'].isnull()]
```

Out[15]:

	PassengerId	PClass	Gender	Sibling	Embarked	Survived
61	62	1	female	0	NaN	1
829	830	1	female	0	NaN	1

In [16]:

```
train['Embarked'] = train['Embarked'].fillna('S')
```

Label Encoding for Train Data Set

In [17]:

```
le_gender = preprocessing.LabelEncoder()
le_Pclass = preprocessing.LabelEncoder()
le_Siblings = preprocessing.LabelEncoder()
le_Embarked = preprocessing.LabelEncoder()
le_Survival = preprocessing.LabelEncoder()
```

In [18]:

```
le_gender =  preprocessing.LabelEncoder()
```

In [19]:

```
train_data_en = train.copy()
```

In [20]:

```
train_data_en['Sibling'] = le_Siblings.fit_transform(train_data_en['Sibling'])

print("Gender attribute encoding in train dataset  :\n")
print(train_data_en[["Sibling","Sibling"]].head())
```

```
Gender attribute encoding in train dataset  :

   Sibling  Sibling
0        1        1
1        1        1
2        0        0
3        1        1
4        0        0
```

In [21]:

```
train_data_en['PClass'] = le_Pclass.fit_transform(train_data_en['PClass'])

print("Gender attribute encoding in train dataset  :\n")
print(train_data_en[["PClass","PClass"]])
```

```
Gender attribute encoding in train dataset  :

      PClass  PClass
0          2       2
1          0       0
2          2       2
3          0       0
4          2       2
5          2       2
6          0       0
7          2       2
8          2       2
9          1       1
10         2       2
11         0       0
12         2       2
13         2       2
14         2       2
15         1       1
16         2       2
17         1       1
18         2       2
19         2       2
20         1       1
21         1       1
22         2       2
23         0       0
24         2       2
25         2       2
26         2       2
27         0       0
28         2       2
29         2       2
...      ...     ...
1279       2       2
1280       2       2
1281       0       0
1282       0       0
1283       2       2
1284       1       1
1285       2       2
1286       0       0
1287       2       2
1288       0       0
1289       2       2
1290       2       2
1291       0       0
1292       1       1
1293       0       0
1294       0       0
1295       0       0
1296       1       1
1297       1       1
1298       0       0
1299       2       2
1300       2       2
1301       2       2
1302       0       0
1303       2       2
1304       2       2
1305       0       0
1306       2       2
1307       2       2
1308       2       2

[1309 rows x 2 columns]
```

In [22]:

```
train_data_en['Embarked'] = le_Embarked.fit_transform(train_data_en['Embarked'])

print("Gender attribute encoding in train dataset  :\n")
print(train_data_en[["Embarked","Embarked"]])
```

```
Gender attribute encoding in train dataset  :

      Embarked  Embarked
0            2         2
1            0         0
2            2         2
3            2         2
4            2         2
5            1         1
6            2         2
7            2         2
8            2         2
9            0         0
10           2         2
11           2         2
12           2         2
13           2         2
14           2         2
15           2         2
16           1         1
17           2         2
18           2         2
19           0         0
20           2         2
21           2         2
22           1         1
23           2         2
24           2         2
25           2         2
26           0         0
27           2         2
28           1         1
29           2         2
...        ...       ...
1279         1         1
1280         2         2
1281         2         2
1282         2         2
1283         2         2
1284         2         2
1285         2         2
1286         2         2
1287         1         1
1288         0         0
1289         2         2
1290         1         1
1291         2         2
1292         2         2
1293         0         0
1294         2         2
1295         0         0
1296         0         0
1297         2         2
1298         0         0
1299         1         1
1300         2         2
1301         1         1
1302         1         1
1303         2         2
1304         2         2
1305         0         0
1306         2         2
1307         2         2
1308         0         0

[1309 rows x 2 columns]
```

In [23]:

```
train_data_en['Gender'] = le_gender.fit_transform(train_data_en['Gender'])

print("Gender attribute encoding in train dataset  :\n")
print(train_data_en[["Gender","Gender"]])
```

```
Gender attribute encoding in train dataset  :

      Gender  Gender
0          1       1
1          0       0
2          0       0
3          0       0
4          1       1
5          1       1
6          1       1
7          1       1
8          0       0
9          0       0
10         0       0
11         0       0
12         1       1
13         1       1
14         0       0
15         0       0
16         1       1
17         1       1
18         0       0
19         0       0
20         1       1
21         1       1
22         0       0
23         1       1
24         0       0
25         0       0
26         1       1
27         1       1
28         0       0
29         1       1
...      ...     ...
1279       1       1
1280       1       1
1281       1       1
1282       0       0
1283       1       1
1284       1       1
1285       1       1
1286       0       0
1287       1       1
1288       0       0
1289       1       1
1290       1       1
1291       0       0
1292       1       1
1293       0       0
1294       1       1
1295       1       1
1296       1       1
1297       1       1
1298       1       1
1299       0       0
1300       0       0
1301       0       0
1302       0       0
1303       0       0
1304       1       1
1305       0       0
1306       1       1
1307       1       1
1308       1       1

[1309 rows x 2 columns]
```

Gender Attribute Encoding in train dataset

In [24]:

```
drop_column = ['PassengerId']
train_data_en.drop(drop_column, axis=1, inplace = True)
```

In [25]:

```
train_data_en
```

Out[25]:

	PClass	Gender	Sibling	Embarked	Survived
0	2	1	1	2	0
1	0	0	1	0	1
2	2	0	0	2	1
3	0	0	1	2	1
4	2	1	0	2	0
5	2	1	0	1	0
6	0	1	0	2	0
7	2	1	3	2	0
8	2	0	0	2	1
9	1	0	1	0	1
10	2	0	1	2	1
11	0	0	0	2	1
12	2	1	0	2	0
13	2	1	1	2	0
14	2	0	0	2	0
15	1	0	0	2	1
16	2	1	4	1	0
17	1	1	0	2	1
18	2	0	1	2	0
19	2	0	0	0	1
20	1	1	0	2	0
21	1	1	0	2	1
22	2	0	0	1	1
23	0	1	0	2	1
24	2	0	3	2	0
25	2	0	1	2	1
26	2	1	0	0	0
27	0	1	3	2	0
28	2	0	0	1	1
29	2	1	0	2	0
...	...	...	...	...	...
1279	2	1	0	1	0
1280	2	1	3	2	0
1281	0	1	0	2	0
1282	0	0	0	2	1
1283	2	1	0	2	0
1284	1	1	0	2	0
1285	2	1	3	2	0
1286	0	0	1	2	1
1287	2	1	0	1	0
1288	0	0	1	0	1
1289	2	1	0	2	0
1290	2	1	0	1	0
1291	0	0	0	2	1
1292	1	1	1	2	0
1293	0	0	0	0	1
1294	0	1	0	2	0
1295	0	1	1	0	0
1296	1	1	0	0	0

1297	1	1	1	2	0
1298	0	1	1	0	0
1299	2	0	0	1	1
1300	2	0	1	2	1
1301	2	0	0	1	1
1302	0	0	1	1	1
1303	2	0	0	2	1
1304	2	1	0	2	0
1305	0	0	0	0	1
1306	2	1	0	2	0
1307	2	1	0	2	0
1308	2	1	1	0	0

1309 rows × 5 columns

In [26]:

```
train_data_en['PClass'] = le_Pclass.fit_transform(train_data_en['PClass'])
train_data_en['Gender'] = le_gender.fit_transform(train_data_en['Gender'])
train_data_en['Sibling'] = le_Siblings.fit_transform(train_data_en['Sibling'])
train_data_en['Embarked'] = le_Embarked.fit_transform(train_data_en['Embarked'])

train_data_en['Survived'] = le_Survival.fit_transform(train_data_en['Survived'])
```

In [27]:

```
train_data_en
```

Out[27]:

	PClass	Gender	Sibling	Embarked	Survived
0	2	1	1	2	0
1	0	0	1	0	1
2	2	0	0	2	1
3	0	0	1	2	1
4	2	1	0	2	0
5	2	1	0	1	0
6	0	1	0	2	0
7	2	1	3	2	0
8	2	0	0	2	1
9	1	0	1	0	1
10	2	0	1	2	1
11	0	0	0	2	1
12	2	1	0	2	0
13	2	1	1	2	0
14	2	0	0	2	0
15	1	0	0	2	1
16	2	1	4	1	0
17	1	1	0	2	1
18	2	0	1	2	0
19	2	0	0	0	1
20	1	1	0	2	0
21	1	1	0	2	1
22	2	0	0	1	1
23	0	1	0	2	1
24	2	0	3	2	0
25	2	0	1	2	1
26	2	1	0	0	0
27	0	1	3	2	0
28	2	0	0	1	1
29	2	1	0	2	0

...	...	...	...	...	...
1279	2	1	0	1	0
1280	2	1	3	2	0
1281	0	1	0	2	0
1282	0	0	0	2	1
1283	2	1	0	2	0
1284	1	1	0	2	0
1285	2	1	3	2	0
1286	0	0	1	2	1
1287	2	1	0	1	0
1288	0	0	1	0	1
1289	2	1	0	2	0
1290	2	1	0	1	0
1291	0	0	0	2	1
1292	1	1	1	2	0
1293	0	0	0	0	1
1294	0	1	0	2	0
1295	0	1	1	0	0
1296	1	1	0	0	0
1297	1	1	1	2	0
1298	0	1	1	0	0
1299	2	0	0	1	1
1300	2	0	1	2	1
1301	2	0	0	1	1
1302	0	0	1	1	1
1303	2	0	0	2	1
1304	2	1	0	2	0
1305	0	0	0	0	1
1306	2	1	0	2	0
1307	2	1	0	2	0
1308	2	1	1	0	0

1309 rows × 5 columns

Spliting of dataset in train and test

In [28]:

```
dataset = train_data_en
```

In [29]:

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
```

In [30]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)
```

Classification Model

LogisticRegression

In [31]:

```
ls = LogisticRegression()

pred_ls = ls.fit(X_train, y_train)
y_pred_ls= pred_ls.predict(X_test)
```

In [32]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_ls))
print(" recall : ",recall_score(y_test, y_pred_ls))
print(" precision : ",precision_score(y_test, y_pred_ls))
print(" f1 : ",f1_score(y_test, y_pred_ls))
```

```
 accuracy :  0.8740458015267175
 recall :  0.7830188679245284
 precision :  0.8924731182795699
 f1 :  0.8341708542713568
```

SGDClassifier

In [33]:

```
SGD = SGDClassifier()

pred_SGD = SGD.fit(X_train, y_train)
y_pred_SGD= pred_SGD.predict(X_test)
```

In [34]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_SGD))
print(" recall : ",recall_score(y_test, y_pred_SGD))
print(" precision : ",precision_score(y_test, y_pred_SGD))
print(" f1 : ",f1_score(y_test, y_pred_SGD))
```

```
 accuracy :  0.8740458015267175
 recall :  0.7641509433962265
 precision :  0.9101123595505618
 f1 :  0.8307692307692307
```

BernoulliNB

In [35]:

```
ber = BernoulliNB()

pred_ber = ber.fit(X_train, y_train)
y_pred_ber= pred_ber.predict(X_test)
```

In [36]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_ber))
print(" recall : ",recall_score(y_test, y_pred_ber))
print(" precision : ",precision_score(y_test, y_pred_ber))
print(" f1 : ",f1_score(y_test, y_pred_ber))
```

```
 accuracy :  0.8664122137404581
 recall :  0.7830188679245284
 precision :  0.8736842105263158
 f1 :  0.8258706467661692
```

LogisticRegressionCV

In [37]:

```
LR = LogisticRegressionCV()

pred_LR = LR.fit(X_train, y_train)
y_pred_LR= pred_LR.predict(X_test)
```

In [38]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_LR))
print(" recall : ",recall_score(y_test, y_pred_LR))
print(" precision : ",precision_score(y_test, y_pred_LR))
print(" f1 : ",f1_score(y_test, y_pred_LR))
```

```
 accuracy :  0.8740458015267175
 recall :  0.7830188679245284
 precision :  0.8924731182795699
 f1 :  0.8341708542713568
```

KNeighborsClassifier

In [39]:

```
KN = KNeighborsClassifier(n_neighbors=3, metric = 'euclidean')
pred_KN= KN.fit(X_train , y_train)
y_pred_KN = pred_KN.predict(X_test)
```

In [40]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_KN))
print(" recall : ",recall_score(y_test, y_pred_KN))
print(" precision : ",precision_score(y_test, y_pred_KN))
print(" f1 : ",f1_score(y_test, y_pred_KN))
```

```
 accuracy :  0.8282442748091603
 recall :  0.6792452830188679
 precision :  0.8674698795180723
 f1 :  0.7619047619047619
```

RandomForestClassifier

In [41]:

```
RC = RandomForestClassifier()

pred_RC = RC.fit(X_train, y_train)
y_pred_RC= pred_RC.predict(X_test)
```

In [42]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_RC))
print(" recall : ",recall_score(y_test, y_pred_RC))
print(" precision : ",precision_score(y_test, y_pred_RC))
print(" f1 : ",f1_score(y_test, y_pred_RC))
```

```
 accuracy :  0.8587786259541985
 recall :  0.7264150943396226
 precision :  0.9058823529411765
 f1 :  0.8062827225130891
```

LinearSVC

In [43]:

```
SVC = LinearSVC()

pred_SVC = SVC.fit(X_train, y_train)
y_pred_SVC= pred_SVC.predict(X_test)
```

In [44]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_SVC))
print(" recall : ",recall_score(y_test, y_pred_SVC))
print(" precision : ",precision_score(y_test, y_pred_SVC))
print(" f1 : ",f1_score(y_test, y_pred_SVC))
```

```
 accuracy :  0.8664122137404581
 recall :  0.7830188679245284
 precision :  0.8736842105263158
 f1 :  0.8258706467661692
```

NuSVC

In [45]:

```
NU = NuSVC()

pred_NU = NU.fit(X_train, y_train)
y_pred_NU= pred_NU.predict(X_test)
```

In [46]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_NU))
print(" recall : ",recall_score(y_test, y_pred_NU))
print(" precision : ",precision_score(y_test, y_pred_NU))
print(" f1 : ",f1_score(y_test, y_pred_NU))
```

```
 accuracy :  0.8740458015267175
 recall :  0.7641509433962265
 precision :  0.9101123595505618
 f1 :  0.8307692307692307
```

DecisionTreeClassifier

In [47]:

```
DT = DecisionTreeClassifier()

pred_DT = DT.fit(X_train, y_train)
y_pred_DT= pred_DT.predict(X_test)
```

In [48]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_DT))
print(" recall : ",recall_score(y_test, y_pred_DT))
print(" precision : ",precision_score(y_test, y_pred_DT))
print(" f1 : ",f1_score(y_test, y_pred_DT))
```

```
 accuracy :  0.8587786259541985
 recall :  0.7264150943396226
 precision :  0.9058823529411765
 f1 :  0.8062827225130891
```

GradientBoostingClassifier

In [49]:

```
GB = GradientBoostingClassifier()

pred_GB = GB.fit(X_train, y_train)
y_pred_GB= pred_GB.predict(X_test)
```

In [50]:

```
print(" accuracy : ",accuracy_score(y_test, y_pred_GB))
print(" recall : ",recall_score(y_test, y_pred_GB))
print(" precision : ",precision_score(y_test, y_pred_GB))
print(" f1 : ",f1_score(y_test, y_pred_GB))
```

```
 accuracy :  0.8740458015267175
 recall :  0.7641509433962265
 precision :  0.9101123595505618
 f1 :  0.8307692307692307
```

Cross fold Validation

In [51]:

```
from sklearn.model_selection import cross_val_score
```

In [52]:

```
classifier1 =  LogisticRegression()
classifier2 =  SGDClassifier()
classifier3 = BernoulliNB()
classifier4 =  LogisticRegressionCV()
classifier5 = KNeighborsClassifier()
classifier6 = RandomForestClassifier()
classifier7 =  LinearSVC()
classifier8 =  NuSVC()
classifier9 = DecisionTreeClassifier()
classifier10 = GradientBoostingClassifier()
```

In [53]:

```
k_dataset = train_data_en
X = k_dataset.iloc[:, :-1]
y = k_dataset.iloc[:, 4]
```

LogisticRegression

In [54]:

```
c1_score1 = cross_val_score(classifier1,X,y,cv=10,scoring='accuracy')
c1_score1
```

Out[54]:

```
array([0.8030303 , 0.8030303 , 0.83333333, 0.75      , 0.77099237,
       0.78461538, 0.83076923, 0.99230769, 0.98461538, 1.        ])
```

In [55]:

```
print(c1_score1.mean())
```

```
0.8552693998113845
```

In [56]:

```
c1_score2 = cross_val_score(classifier1,X,y,cv=10,scoring='recall')
c1_score2
```

Out[56]:

```
array([0.74      , 0.7       , 0.8       , 0.6       , 0.65306122,
       0.57142857, 0.73469388, 0.97959184, 0.95918367, 1.        ])
```

In [57]:

```
c1_score2.mean()
```

Out[57]:

```
0.773795918367347
```

In [58]:

```
c1_score3 = cross_val_score(classifier1,X,y,cv=10,scoring='precision')
c1_score3
```

Out[58]:

```
array([0.74      , 0.76086957, 0.76923077, 0.69767442, 0.71111111,
       0.8       , 0.8       , 1.        , 1.        , 1.        ])
```

In [59]:

```
c1_score3.mean()
```

Out[59]:

```
0.8278885864163922
```

In [60]:

```
c1_score4 = cross_val_score(classifier1,X,y,cv=10,scoring='f1')
c1_score4
```

Out[60]:

```
array([0.74      , 0.72916667, 0.78431373, 0.64516129, 0.68085106,
       0.66666667, 0.76595745, 0.98969072, 0.97916667, 1.        ])
```

In [61]:

```
c1_score4.mean()
```

Out[61]:

```
0.798097424810056
```

SGDClassifier

In [62]:

```
c2_score1 = cross_val_score(classifier2,X,y,cv=10,scoring='accuracy')
c2_score1
```

Out[62]:

```
array([0.81060606, 0.79545455, 0.83333333, 0.75      , 0.80152672,
       0.78461538, 0.82307692, 1.        , 0.98461538, 1.        ])
```

In [63]:

```
print(c2_score1.mean())
```

```
0.8583228349258883
```

In [64]:

```
c2_score2 = cross_val_score(classifier2,X,y,cv=10,scoring='recall')
c2_score2
```

Out[64]:

```
array([0.7       , 0.68      , 0.8       , 0.52      , 0.65306122,
       0.42857143, 0.73469388, 0.97959184, 0.95918367, 0.73469388])
```

In [65]:

```
c2_score2.mean()
```

Out[65]:

```
0.7189795918367347
```

In [66]:

```
c2_score3 = cross_val_score(classifier2,X,y,cv=10,scoring='precision')
c2_score3
```

Out[66]:

```
array([0.95652174, 0.74468085, 0.48421053, 0.69767442, 0.71111111,
       0.8       , 0.81818182, 1.        , 1.        , 1.        ])
```

In [67]:

```
c2_score3.mean()
```

Out[67]:

```
0.8212380464407634
```

In [68]:

```
c2_score4 = cross_val_score(classifier2,X,y,cv=10,scoring='f1')
c2_score4
```

Out[68]:

```
array([0.74226804, 0.72164948, 0.78431373, 0.64516129, 0.71111111,
       0.66666667, 0.75789474, 1.        , 1.        , 0.97029703])
```

In [69]:

```
c2_score4.mean()
```

Out[69]:

```
0.7999362085908825
```

BernoulliNB

In [70]:

```
c3_score1 = cross_val_score(classifier3,X,y,cv=10,scoring='accuracy')
c3_score1
```

Out[70]:

```
array([0.8030303 , 0.79545455, 0.83333333, 0.75      , 0.77099237,
       0.77692308, 0.82307692, 1.        , 1.        , 1.        ])
```

In [71]:

```
print(c3_score1.mean())
```

```
0.8552810548230395
```

In [72]:

```
c3_score2 = cross_val_score(classifier3,X,y,cv=10,scoring='recall')
c3_score2
```

Out[72]:

```
array([0.74      , 0.7       , 0.8       , 0.6       , 0.65306122,
       0.57142857, 0.73469388, 1.        , 1.        , 1.        ])
```

In [73]:

```
c3_score2.mean()
```

Out[73]:

```
0.7799183673469388
```

In [74]:

```
c3_score3 = cross_val_score(classifier3,X,y,cv=10,scoring='precision')
c3_score3
```

Out[74]:

```
array([0.74      , 0.74468085, 0.76923077, 0.69767442, 0.71111111,
       0.77777778, 0.7826087 , 1.        , 1.        , 1.        ])
```

In [75]:

```
c3_score3.mean()
```

Out[75]:

```
0.8223083623440314
```

In [76]:

```
c3_score4 = cross_val_score(classifier3,X,y,cv=10,scoring='f1')
c3_score4
```

Out[76]:

```
array([0.74      , 0.72164948, 0.78431373, 0.64516129, 0.68085106,
       0.65882353, 0.75789474, 1.        , 1.        , 1.        ])
```

In [77]:

```
c3_score4.mean()
```

Out[77]:

```
0.7988693830432516
```

LogisticRegressionCV

In [78]:

```
c4_score1 = cross_val_score(classifier4,X,y,cv=10,scoring='accuracy')
c4_score1
```

Out[78]:

```
array([0.8030303 , 0.79545455, 0.83333333, 0.75      , 0.77099237,
       0.78461538, 0.83076923, 0.99230769, 0.98461538, 1.        ])
```

In [79]:

```
print(c4_score1.mean())
```

```
0.8545118240538088
```

In [80]:

```
c4_score2 = cross_val_score(classifier4,X,y,cv=10,scoring='recall')
c4_score2
```

Out[80]:

```
array([0.74      , 0.68      , 0.8       , 0.6       , 0.65306122,
       0.57142857, 0.73469388, 0.97959184, 0.95918367, 1.        ])
```

In [81]:

```
c4_score2.mean()
```

Out[81]:

```
0.771795918367347
```

In [82]:

```
c4_score3 = cross_val_score(classifier4,X,y,cv=10,scoring='precision')
c4_score3
```

Out[82]:

```
array([0.74      , 0.75555556, 0.76923077, 0.69767442, 0.71111111,
       0.8       , 0.8       , 1.        , 1.        , 1.        ])
```

In [83]:

```
c4_score3.mean()
```

Out[83]:

```
0.8273571854502088
```

In [84]:

```
c4_score4 = cross_val_score(classifier4,X,y,cv=10,scoring='f1')
c4_score4
```

Out[84]:

```
array([0.74      , 0.71578947, 0.78431373, 0.64516129, 0.68085106,
       0.66666667, 0.76595745, 0.98969072, 0.97916667, 1.        ])
```

In [85]:

```
c4_score4.mean()
```

Out[85]:

```
0.7967597055118103
```

KNeighborsClassifier

In [86]:

```
c5_score1 = cross_val_score(classifier5,X,y,cv=10,scoring='accuracy')
c5_score1
```

Out[86]:

```
array([0.76515152, 0.8030303 , 0.82575758, 0.76515152, 0.79389313,
       0.76923077, 0.79230769, 0.86923077, 0.87692308, 0.84615385])
```

In [87]:

```
print(c5_score1.mean())
```

```
0.8106830192708057
```

In [88]:

```
c5_score2 = cross_val_score(classifier5,X,y,cv=10,scoring='recall')
c5_score2
```

Out[88]:

```
array([0.62      , 0.68      , 0.66      , 0.58      , 0.63265306,
       0.59183673, 0.71428571, 0.85714286, 0.89795918, 0.79591837])
```

In [89]:

```
c5_score2.mean()
```

Out[89]:

```
0.7029795918367348
```

In [90]:

```
c5_score3 = cross_val_score(classifier5,X,y,cv=10,scoring='precision')
c5_score3
```

Out[90]:

```
array([0.72093023, 0.77272727, 0.84615385, 0.74358974, 0.775     ,
       0.74358974, 0.72916667, 0.80769231, 0.8       , 0.79591837])
```

In [91]:

```
c5_score3.mean()
```

Out[91]:

```
0.7734768180324658
```

In [92]:

```
c5_score4 = cross_val_score(classifier5,X,y,cv=10,scoring='f1')
c5_score4
```

Out[92]:

```
array([0.66666667, 0.72340426, 0.74157303, 0.65168539, 0.69662921,
       0.65909091, 0.72164948, 0.83168317, 0.84615385, 0.79591837])
```

In [93]:

```
c5_score4.mean()
```

Out[93]:

```
0.7334454337879863
```

RandomForestClassifier

In [94]:

```
c6_score1 = cross_val_score(classifier6,X,y,cv=10,scoring='accuracy')
c6_score1
```

Out[94]:

```
array([0.8030303 , 0.81060606, 0.84090909, 0.77272727, 0.78625954,
       0.78461538, 0.83076923, 0.99230769, 0.98461538, 0.99230769])
```

In [95]:

```
print(c6_score1.mean())
```

```
0.8598147653872845
```

In [96]:

```
c6_score2 = cross_val_score(classifier6,X,y,cv=10,scoring='recall')
c6_score2
```

Out[96]:

```
array([0.7       , 0.68      , 0.8       , 0.6       , 0.65306122,
       0.57142857, 0.73469388, 0.97959184, 0.95918367, 0.97959184])
```

In [97]:

```
c6_score2.mean()
```

Out[97]:

```
0.7657551020408163
```

In [98]:

```
c6_score3 = cross_val_score(classifier6,X,y,cv=10,scoring='precision')
c6_score3
```

Out[98]:

```
array([0.76086957, 0.79069767, 0.78431373, 0.75      , 0.74418605,
       0.8       , 0.8       , 1.        , 1.        , 1.        ])
```

In [99]:

```
c6_score3.mean()
```

Out[99]:

```
0.8430067011637821
```

In [100]:

```
c6_score4 = cross_val_score(classifier6,X,y,cv=10,scoring='f1')
c6_score4
```

Out[100]:

```
array([0.72164948, 0.7311828 , 0.79207921, 0.66666667, 0.69565217,
       0.66666667, 0.77419355, 0.98969072, 0.97916667, 0.98969072])
```

In [101]:

```
c6_score4.mean()
```

Out[101]:

```
0.8006638653754908
```

LinearSVC

In [102]:

```
c7_score1 = cross_val_score(classifier7,X,y,cv=10,scoring='accuracy')
c7_score1
```

Out[102]:

```
array([0.8030303 , 0.8030303 , 0.83333333, 0.75      , 0.77099237,
       0.77692308, 0.82307692, 1.        , 0.99230769, 1.        ])
```

In [103]:

```
print(c7_score1.mean())
```

```
0.8552693998113845
```

In [104]:

```
c7_score2 = cross_val_score(classifier7,X,y,cv=10,scoring='recall')
c7_score2
```

Out[104]:

```
array([0.74      , 0.7       , 0.8       , 0.6       , 0.65306122,
       0.57142857, 0.73469388, 1.        , 0.97959184, 1.        ])
```

In [105]:

```
c7_score2.mean()
```

Out[105]:

```
0.7778775510204082
```

In [106]:

```
c7_score3 = cross_val_score(classifier7,X,y,cv=10,scoring='precision')
c7_score3
```

Out[106]:

```
array([0.74      , 0.76086957, 0.76923077, 0.69767442, 0.71111111,
       0.77777778, 0.7826087 , 1.        , 1.        , 1.        ])
```

In [107]:

```
c7_score3.mean()
```

Out[107]:

```
0.8239272337593875
```

In [108]:

```
c7_score4 = cross_val_score(classifier7,X,y,cv=10,scoring='f1')
c7_score4
```

Out[108]:

```
array([0.74      , 0.72916667, 0.78431373, 0.64516129, 0.68085106,
       0.65882353, 0.75789474, 1.        , 0.98969072, 1.        ])
```

In [109]:

```
c7_score4.mean()
```

Out[109]:

```
0.7985901734212585
```

NuSVC()

In [110]:

```
c8_score1 = cross_val_score(classifier8,X,y,cv=10,scoring='accuracy')
c8_score1
```

Out[110]:

```
array([0.81060606, 0.8030303 , 0.83333333, 0.76515152, 0.80152672,
       0.77692308, 0.83846154, 0.98461538, 0.98461538, 0.99230769])
```

In [111]:

```
print(c8_score1.mean())
```

```
0.8590571006601542
```

In [112]:

```
c8_score2 = cross_val_score(classifier8,X,y,cv=10,scoring='recall')
c8_score2
```

Out[112]:

```
array([0.7       , 0.68      , 0.78      , 0.6       , 0.65306122,
       0.55102041, 0.73469388, 0.95918367, 0.95918367, 0.97959184])
```

In [113]:

```
c8_score2.mean()
```

Out[113]:

```
0.7596734693877552
```

In [114]:

```
c8_score3 = cross_val_score(classifier8,X,y,cv=10,scoring='precision')
c8_score3
```

Out[114]:

```
array([0.77777778, 0.77272727, 0.78      , 0.73170732, 0.7804878 ,
       0.79411765, 0.81818182, 1.        , 1.        , 1.        ])
```

In [115]:

```
c8_score3.mean()
```

Out[115]:

```
0.8454999637696912
```

In [116]:

```
c8_score4 = cross_val_score(classifier8,X,y,cv=10,scoring='f1')
c8_score4
```

Out[116]:

```
array([0.73684211, 0.72340426, 0.78      , 0.65934066, 0.71111111,
       0.65060241, 0.77419355, 0.97916667, 0.97916667, 0.98969072])
```

In [117]:

```
c8_score4.mean()
```

Out[117]:

```
0.7983518144042546
```

DecisionTreeClassifier

In [118]:

```
c9_score1 = cross_val_score(classifier9,X,y,cv=10,scoring='accuracy')
c9_score1
```

Out[118]:

```
array([0.8030303 , 0.81060606, 0.84090909, 0.77272727, 0.78625954,
       0.78461538, 0.83846154, 0.99230769, 0.98461538, 0.99230769])
```

In [119]:

```
print(c9_score1.mean())
```

```
0.8605839961565153
```

In [120]:

```
c9_score2 = cross_val_score(classifier9,X,y,cv=10,scoring='recall')
c9_score2
```

Out[120]:

```
array([0.7       , 0.68      , 0.8       , 0.6       , 0.65306122,
       0.57142857, 0.73469388, 0.97959184, 0.95918367, 0.97959184])
```

In [121]:

```
c9_score2.mean()
```

Out[121]:

```
0.7657551020408163
```

In [122]:

```
c9_score3 = cross_val_score(classifier9,X,y,cv=10,scoring='precision')
c9_score3
```

Out[122]:

```
array([0.76086957, 0.79069767, 0.78431373, 0.75      , 0.74418605,
       0.8       , 0.81818182, 1.        , 1.        , 1.        ])
```

In [123]:

```
c9_score3.mean()
```

Out[123]:

```
0.8448248829819638
```

In [124]:

```
c9_score4 = cross_val_score(classifier9,X,y,cv=10,scoring='f1')
c9_score4
```

Out[124]:

```
array([0.72916667, 0.7311828 , 0.79207921, 0.66666667, 0.69565217,
       0.66666667, 0.77419355, 0.98969072, 0.97916667, 0.98969072])
```

In [125]:

```
c9_score4.mean()
```

Out[125]:

```
0.8014155835885493
```

GradientBoostingClassifier

In [126]:

```
c10_score1 = cross_val_score(classifier10,X,y,cv=10,scoring='accuracy')
c10_score1
```

Out[126]:

```
array([0.8030303 , 0.8030303 , 0.84090909, 0.77272727, 0.78625954,
       0.78461538, 0.83076923, 0.99230769, 0.98461538, 0.99230769])
```

In [127]:

```
print(c10_score1.mean())
```

```
0.8590571896297087
```

In [128]:

```
c10_score2 = cross_val_score(classifier10,X,y,cv=10,scoring='recall')
c10_score2
```

Out[128]:

```
array([0.7       , 0.68      , 0.8       , 0.6       , 0.65306122,
       0.57142857, 0.73469388, 0.97959184, 0.95918367, 0.97959184])
```

In [129]:

```
c10_score2.mean()
```

Out[129]:

```
0.7657551020408163
```

In [130]:

```
c10_score3 = cross_val_score(classifier10,X,y,cv=10,scoring='precision')
c10_score3
```

Out[130]:

```
array([0.76086957, 0.77272727, 0.78431373, 0.75      , 0.74418605,
       0.8       , 0.8       , 1.        , 1.        , 1.        ])
```

In [131]:

```
c10_score3.mean()
```

Out[131]:

```
0.8412096609946488
```

In [132]:

```
c10_score4 = cross_val_score(classifier10,X,y,cv=10,scoring='f1')
c10_score4
```

Out[132]:

```
array([0.72916667, 0.72340426, 0.79207921, 0.66666667, 0.69565217,
       0.66666667, 0.76595745, 0.98969072, 0.97916667, 0.98969072])
```

In [133]:

```
c10_score4.mean()
```

Out[133]:

```
0.7998141193927131
```

Table

TRAIN TEST SPLIT APPROACH

In [134]:

```
print("Detailed performance of all the models")
print("===================================+")
print("+-----------------------------------+")
print("|        Model               |       Accuracy       |   Precision        | Recall            |   f1
|")
print("+-----------------------------------+")

print("|LogisticRegression        | ",accuracy_score(y_test, y_pred_ls), "|",precision_score(y_test, y_pred_ls),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("|SGDClassifier             | ",accuracy_score(y_test, y_pred_SGD),"|",precision_score(y_test, y_pred_SGD),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("|BernoulliNB               | ",accuracy_score(y_test, y_pred_ber),"|",precision_score(y_test, y_pred_ber),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("|LogisticRegressionCV      | ",accuracy_score(y_test,y_pred_LR),  "|",precision_score(y_test,y_pred_LR),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("|KNeighborsClassifier      | ",accuracy_score(y_test, y_pred_KN), "|",precision_score(y_test, y_pred_KN),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("|RandomForestClassifier    | ",accuracy_score(y_test, y_pred_RC), "|",precision_score(y_test, y_pred_RC),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("|LinearSVC                 | ",accuracy_score(y_test, y_pred_SVC),"|",precision_score(y_test, y_pred_SVC),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("|NuSVC                     | ",accuracy_score(y_test, y_pred_NU), "|",precision_score(y_test, y_pred_NU),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("|DecisionTreeClassifier    | ",accuracy_score(y_test, y_pred_DT), "|",precision_score(y_test, y_pred_DT),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("|GradientBoostingClassifier| ",accuracy_score(y_test, y_pred_GB), "|",precision_score(y_test, y_pred_GB),
" |",recall_score(y_test, y_pred_ls), "|",f1_score(y_test, y_pred_ls)," |")
print("+-----------------------------------+")
print("")
print("                Best Model          ")
print("===================================+")
print("+-----------------------------------+")
print("|        Models       |     Accuracy    |")
print("+-----------------------------------+")
print("| DecisionTreeClassifier      |     ",accuracy_score(y_test, y_pred_DT),"     |")
print("| GradientBoostingClassifier |     ",accuracy_score(y_test, y_pred_GB),"     |")
print("+-----------------------------------+")
```

```
Detailed performance of all the models
=====================================+
+-------------------------------------+
|        Model               |       Accuracy       |   Precision          | Recall             |   f1
|
+-------------------------------------+
|LogisticRegression         |   0.8740458015267175 | 0.8924731182795699   | 0.7830188679245284 | 0.8341
708542713568   |
|SGDClassifier              |   0.8740458015267175 | 0.9101123595505618   | 0.7830188679245284 | 0.8341
708542713568   |
|BernoulliNB                |   0.8664122137404581 | 0.8736842105263158   | 0.7830188679245284 | 0.8341
708542713568   |
|LogisticRegressionCV       |   0.8740458015267175 | 0.8924731182795699   | 0.7830188679245284 | 0.8341
708542713568   |
|KNeighborsClassifier       |   0.8282442748091603 | 0.8674698795180723   | 0.7830188679245284 | 0.8341
708542713568   |
|RandomForestClassifier     |   0.8587786259541985 | 0.9058823529411765   | 0.7830188679245284 | 0.8341
708542713568   |
|LinearSVC                  |   0.8664122137404581 | 0.8736842105263158   | 0.7830188679245284 | 0.8341
708542713568   |
|NuSVC                      |   0.8740458015267175 | 0.9101123595505618   | 0.7830188679245284 | 0.8341
708542713568   |
|DecisionTreeClassifier     |   0.8587786259541985 | 0.9058823529411765   | 0.7830188679245284 | 0.8341
708542713568   |
|GradientBoostingClassifier|   0.8740458015267175 | 0.9101123595505618   | 0.7830188679245284 | 0.8341
708542713568   |
+-------------------------------------+

                Best Model
=====================================+
+-------------------------------------+
|        Models         |    Accuracy    |
+-------------------------------------+
| DecisionTreeClassifier       |      0.8587786259541985       |
| GradientBoostingClassifier |      0.8740458015267175       |
+-------------------------------------+
```

K fold Cross Validation Approach

In [135]:

```
print("Detailed performance of all the models")
print("====================================+")
print("+-----------------------------------+")
print("|        Model             |       Accuracy       |   Precision        | Recall             |   f1
|")
print("+-----------------------------------+")

print("|LogisticRegression        | ",c1_score1.mean(),  "|",c1_score2.mean()," |",c1_score3.mean(), "|",c1_score
4.mean()," |")
print("|SGDClassifier             | ",c2_score1.mean(),  "|",c2_score2.mean()," |",c2_score3.mean(), "|",c2_score
4.mean()," |")
print("|BernoulliNB               | ",c3_score1.mean(),  "|",c3_score2.mean()," |",c3_score3.mean(), "|",c2_score
4.mean()," |")
print("|LogisticRegressionCV      | ",c4_score1.mean(),  "|",c4_score2.mean()," |",c4_score3.mean(), "|",c2_score
4.mean(), "|")
print("|KNeighborsClassifier      | ",c5_score1.mean(),  "|",c5_score2.mean()," |",c5_score3.mean(), "|",c2_score
4.mean()," |")
print("|RandomForestClassifier    | ",c6_score1.mean(),  "|",c6_score2.mean()," |",c6_score3.mean(), "|",c2_score
4.mean()," |")
print("|LinearSVC                 | ",c7_score1.mean(),  "|",c7_score2.mean()," |",c7_score3.mean(), "|",c2_score
4.mean(), "|")
print("|NuSVC                     | ",c8_score1.mean(),  "|",c8_score2.mean()," |",c8_score3.mean(), "|",c2_score
4.mean(), "|")
print("|DecisionTreeClassifier    | ",c9_score1.mean(),  "|",c9_score2.mean()," |",c9_score3.mean(), "|",c2_score
4.mean(), "|")
print("|GradientBoostingClassifier| ",c10_score1.mean(), "|",c10_score2.mean(),"|",c10_score3.mean(),"|",c2_score
4.mean(), "|")
print("+-----------------------------------+")
print("")
print("                Best Model           ")
print("====================================+")
print("+-----------------------------------+")
print("|        Model       |    Accuracy    |")
print("+-----------------------------------+")
print("| DecisionTreeClassifier |     ",c9_score1.mean(),"     |")
print("+-----------------------------------+")
```

```
Detailed performance of all the models
====================================+
+-----------------------------------+
|        Model             |       Accuracy       |   Precision        | Recall             |   f1
|
+-----------------------------------+
|LogisticRegression        |  0.8552693998113845 | 0.773795918367347  | 0.8278885864163922 | 0.79809
7424810056  |
|SGDClassifier             |  0.8583228349258883 | 0.7189795918367347  | 0.8212380464407634 | 0.7999
362085908825  |
|BernoulliNB               |  0.8552810548230395 | 0.7799183673469388  | 0.8223083623440314 | 0.7999
362085908825  |
|LogisticRegressionCV      |  0.8545118240538088 | 0.771795918367347  | 0.8273571854502088 | 0.79993
62085908825 |
|KNeighborsClassifier      |  0.8106830192708057 | 0.7029795918367348  | 0.7734768180324658 | 0.7999
362085908825  |
|RandomForestClassifier    |  0.8598147653872845 | 0.7657551020408163  | 0.8430067011637821 | 0.7999
362085908825  |
|LinearSVC                 |  0.8552693998113845 | 0.7778775510204082  | 0.8239272337593875 | 0.7999
362085908825 |
|NuSVC                     |  0.8590571006601542 | 0.7596734693877552  | 0.8454999637696912 | 0.7999
362085908825 |
|DecisionTreeClassifier    |  0.8605839961565153 | 0.7657551020408163  | 0.8448248829819638 | 0.7999
362085908825 |
|GradientBoostingClassifier|  0.8590571896297087 | 0.7657551020408163 | 0.8412096609946488 | 0.79993
62085908825 |
+-----------------------------------+

                Best Model
====================================+
+-----------------------------------+
|        Model       |    Accuracy    |
+-----------------------------------+
| DecisionTreeClassifier |      0.8605839961565153      |
+-----------------------------------+
```

GUI

In [136]:

```
dataset_new = train_data_en
```

In [137]:

```
X_train = dataset_new[['PClass','Gender','Sibling','Embarked']]
Y_train = dataset_new[['Survived']]
```

In [138]:

```
RC.fit(X_train, np.ravel(Y_train))
```

Out[138]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=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=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [139]:

```
import pickle
import pandas as pd
```

In [140]:

```
model_saved = pickle.dumps(RC)
```

In [141]:

```
BestClassifier = pickle.loads(model_saved)
```

In [142]:

```
from tkinter import*
```

In [143]:

```
root = Tk()
root.title("TITANIC PROJECT")
```

Out[143]:

```
''
```

In [144]:

```
e1 = Entry(root, width =35, borderwidth = 5)
e1.grid(row= 1, column=2, columnspan = 3, padx = 20 , pady = 10)

e2 = Entry(root, width =35, borderwidth = 5)
e2.grid(row= 2, column=2, columnspan = 3, padx = 20 , pady = 10)

e3 = Entry(root, width =35, borderwidth = 5)
e3.grid(row= 3, column=2, columnspan = 3, padx = 20 , pady = 10)

e4 = Entry(root, width =35, borderwidth = 5)
e4.grid(row= 4, column=2, columnspan = 3, padx = 20 , pady = 10)
```

In [145]:

```
def myClick():
        global PClass
        PClass = e1.get()
        global Gender
        Gender = e2.get()
        global Sibling
        Sibling = e3.get()
        global Embarked
        Embarked = e4.get()
        user_data = {
            'PClass' : [PClass],
            'Gender' : [Gender],
            'Sibling': [Sibling],
            'Embarked': [Embarked]
        }
        user_input = pd.DataFrame(user_data)
        print("User input in actual DataFrame format:\n")
        user_input
        user_input_encoded = user_input.copy()
        Embarked  = ['S','C','Q']
        Gender    = ["male","female"]
        le_Embarked.fit(Embarked)
        le_gender.fit(Gender)

        user_input_encoded['Embarked'] = le_Embarked.transform(user_input['Embarked'])
        user_input_encoded['Gender']   = le_gender.transform(user_input['Gender'])
        ud = pd.DataFrame(user_data)
        print("User input in actual DataFrame format : \n")
        print(ud,"\n")
        print("User input in encoded DataFrame format : \n")
        print(user_input_encoded)
        global prediction
        prediction = BestClassifier.predict(user_input_encoded)
        print(prediction)
        root.destroy()
```

In [146]:

```
myLabel = Label(root, text="ENTER YOUR PREDICTIONS")
myLabel1 = Label(root, text="Enter the Passenger class ")
myLabel2 = Label(root, text="Enter the Gender ")
myLabel3 = Label(root, text="Enter the Siblings")
myLabel4 = Label(root, text="Enter Embarked ")


myLabel5 = Label(root, text=" 1, 2,3, --")
myLabel6 = Label(root, text=" male, female")
myLabel7 = Label(root, text=" 1, 2, 3 ---")
myLabel8 = Label(root, text=" S, C, Q ")

myLabel.grid(row=0 , column = 2, padx= 10, pady =20)
myLabel1.grid(row=1 , column = 0, padx= 10, pady =20)
myLabel2.grid(row=2 , column = 0,padx= 10, pady =20)
myLabel3.grid(row=3 , column = 0,padx= 10, pady =20)
myLabel4.grid(row=4 , column = 0,padx=10, pady =20)

myLabel5.grid(row=1 , column = 4, padx= 10, pady =20)
myLabel6.grid(row=2 , column = 4,padx= 10, pady =20)
myLabel7.grid(row=3 , column = 4,padx= 10, pady =20)
myLabel8.grid(row=4 , column = 4,padx=10, pady =20)

myButton1 = Button(root, text="Predict", command = myClick,fg= "blue")

myButton1.grid(row=6 , column = 3, padx= 50, pady =20)

print("this is my function")

root.mainloop()
```

```
this is my function
User input in actual DataFrame format:

User input in actual DataFrame format :

  PClass Gender Sibling Embarked
0      2   male       3        Q

User input in encoded DataFrame format :

  PClass  Gender Sibling  Embarked
0      2       1       3         1
[0]
```

In [ ]:

```
if prediction == 0:
      root2 = Tk()
      root2.title("NOT SURVIVED")
      button = Button(root2, text = "NOT SURVIVED", padx =50, pady =50)
      button.pack()
      root2.mainloop()
else:
      root1 = Tk()
      root1.title("SURVIVED")
      button = Button(root2, text = "NOT SURVIVED", padx =50, pady =50)
      button.pack()
      root1.mainloop()
```

Out[ ]:

```
''
```

In [ ]:

```

```

In [ ]:

```

```