Predicting the Survival of Titanic Passengers

RMS TITANIC:

RMS Titanic was a British passenger liner operated by the White Star Line that sank in the North Atlantic Ocean on 15 April 1912, after striking an iceberg during her maiden voyage from Southampton to New York City.

- In this blog,I will experiment with different machine learning algorithms and build a program that can predict whether a given passenger would have survived this disaster or not according to Pclass, sex, age etc

Importing necessary library

```
import pandas as pd
import numpy as np

#Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns

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

Getting the Data

```
df=pd.read_csv('titanic.csv')
df.head()
```

Checking shape of dataset.

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
df.shape
(891, 12)
```

In this dataset 891 rows & 12 columns are present,which is describes below:

1.PassengerID:Unique ID of passenger.

2.survived: Survival

(0 = no; 1 = yes)

3.Pclass: Passenger class

(1 = first; 2 = second; 3 = third)

4.Name: Name

5.sex: Sex

6.Age: Age in years

7.sibsp: Number of siblings/spouses aboard

8.parch: Number of parents/children aboard

9.ticket: Ticket number

10.fare: Passenger fare

11.cabin: Cabin number

12.embarked: Port of embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

Data Exploration/Analysis

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Observation:

1.It has 11 features & 1 target column (survived).

2.Age & Fare are floats

3.PassengerID,Survived,Pclass,sibsp,parch are integers

4.Name,sex,Ticket,cabin,embarked are objects.

Now we check description of numerical column

```
df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Some Observation:

1.There are a total of 891 passengers in our dataset

2. there are very less mean value for the survived ,it means very less people survived

3.the mean of age are around 30,it means maxm middle age people were travelling

4.For age minm value is 0.42,it means some infants were also travelling (of few months).

5.We also notice that age contains some missing values.

Column names:

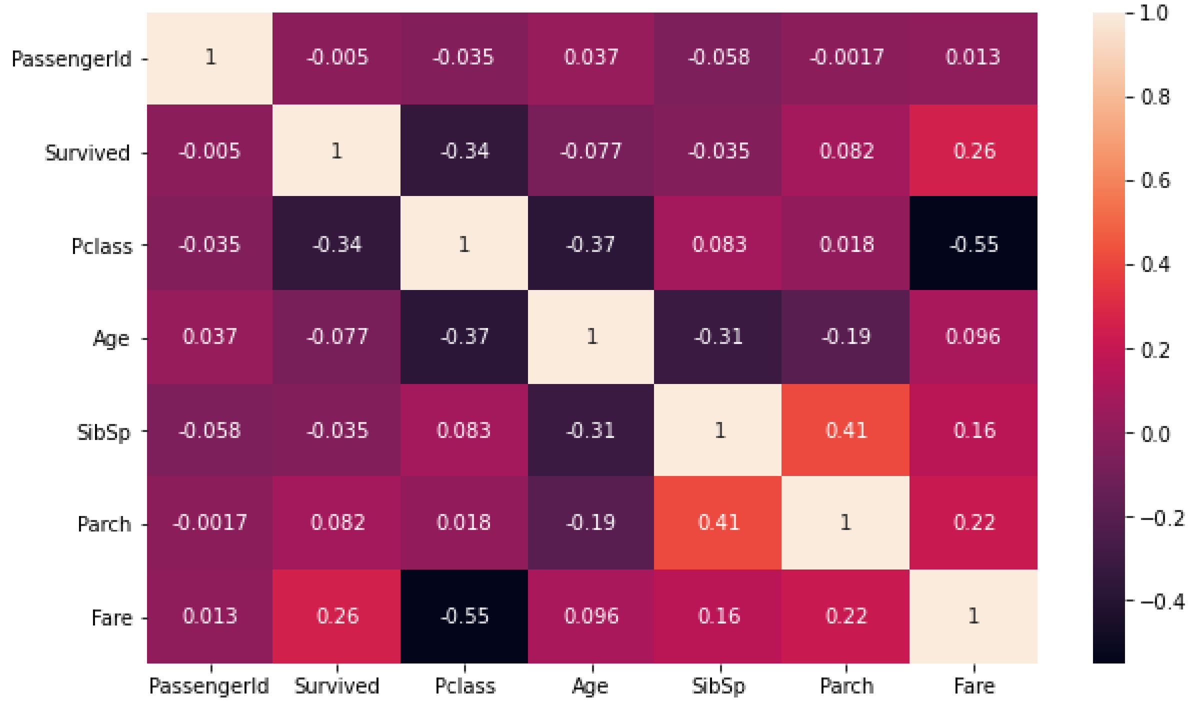
['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']

*Now we will try to find out, **What features could contribute to high survival rate?***

First we check correlation between the columns.

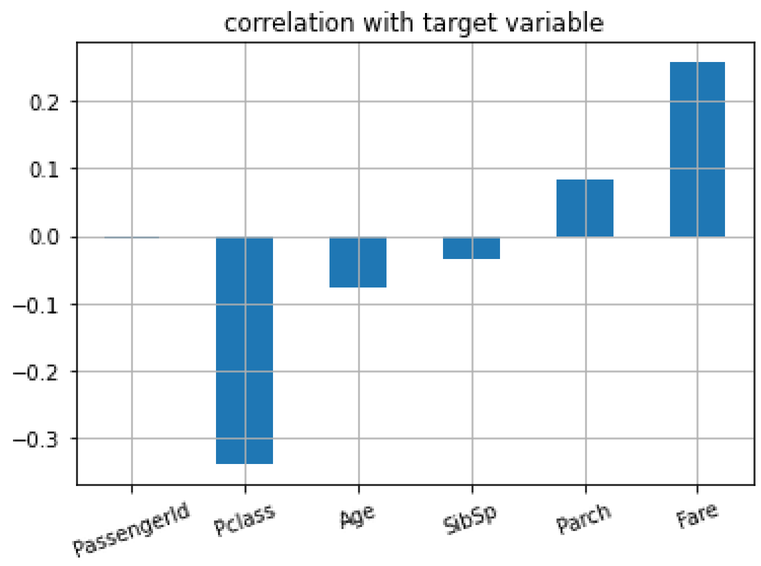
check corelation

```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(),annot=True)
```



Now we check correlation with the target variable .ie survived.

```
#Checking correlation with the taget variable .ie num
plt.figure(figsize=(8,6))
df.drop('Survived',axis=1).corrwith(df['Survived']).plot(kind='bar',grid=True)
plt.xticks(rotation=20)
```



Observation:

1.survived is very very less correlated with passengerId

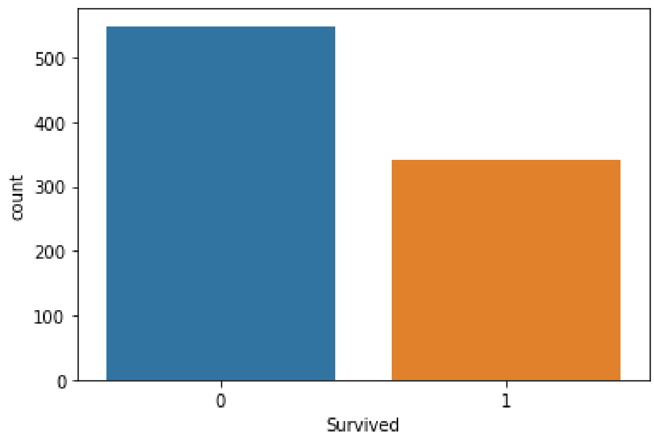
2.Survived is positively correlated with Parch & Fare

3.Survived is negatively correlated with Pclass,Age,Sibsp

- Now we perform some **DATA VISUALIZATION**

Survived.

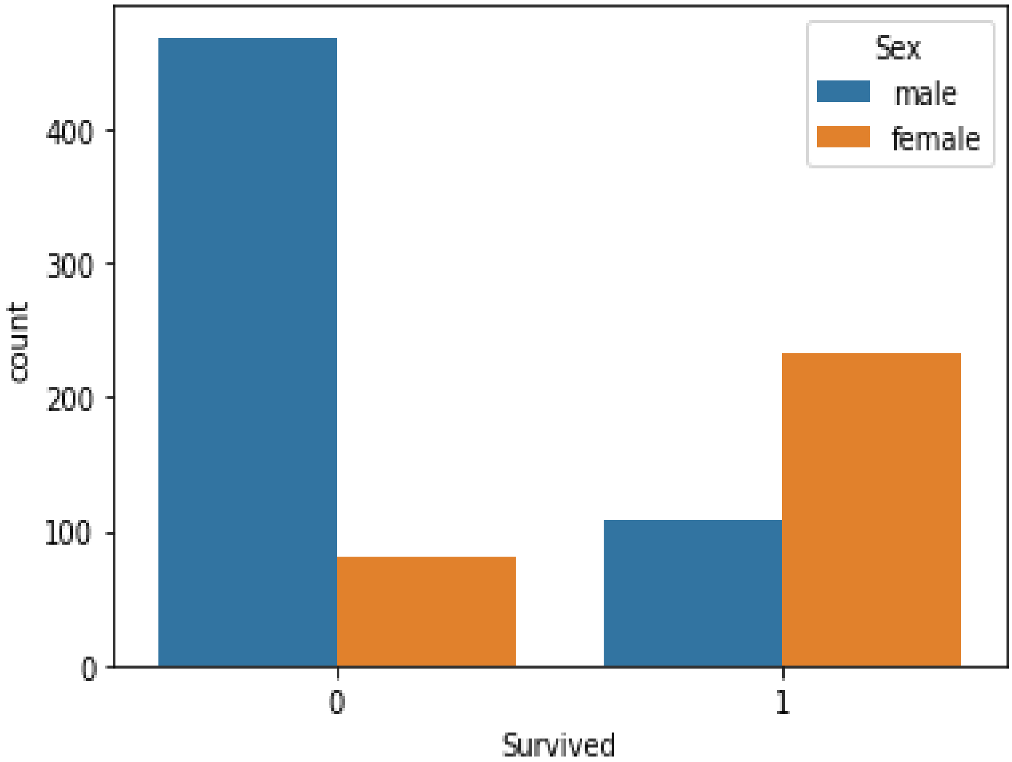
```
sns.countplot(x='Survived',data=df)
```



From above graph we find that no. of survived people is very less than that of not survived

Survived vs sex

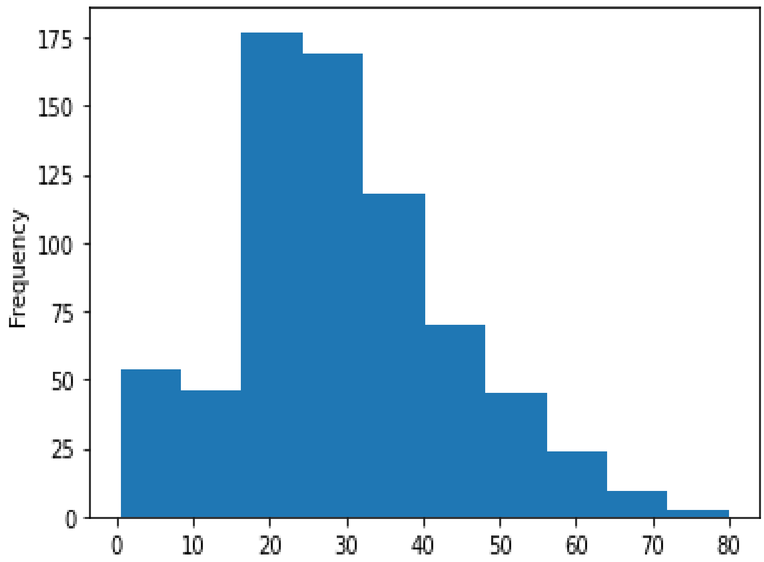
```
sns.countplot(x='Survived',data=df,hue='Sex')
```



From above plot is is clear that out of the total male very few survived,while on the other hand most of the female survived.

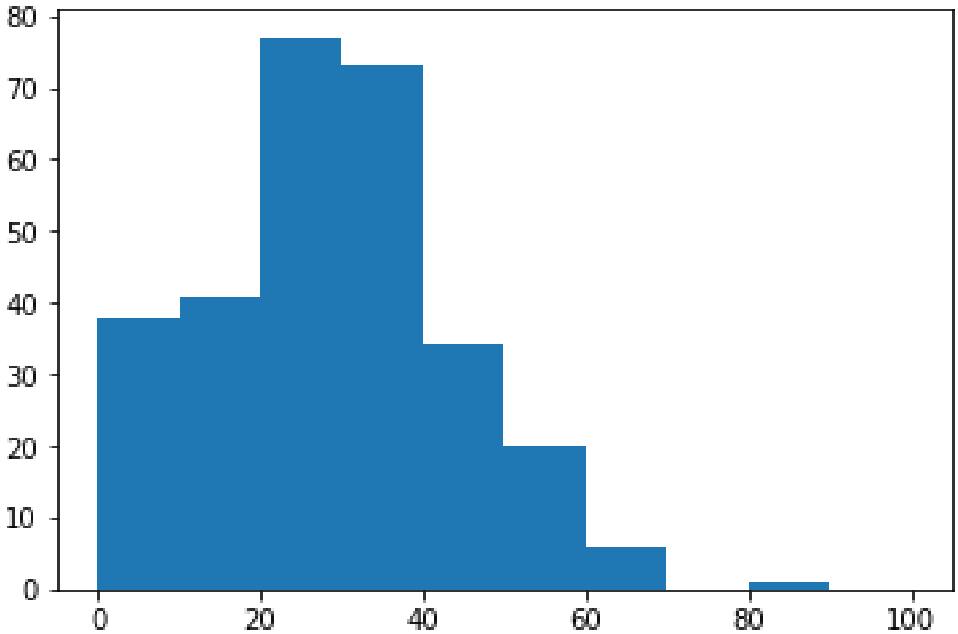
Checking Age columns

```
df['Age'].plot.hist()
```



It shows more middle age people are travelling on titanic.

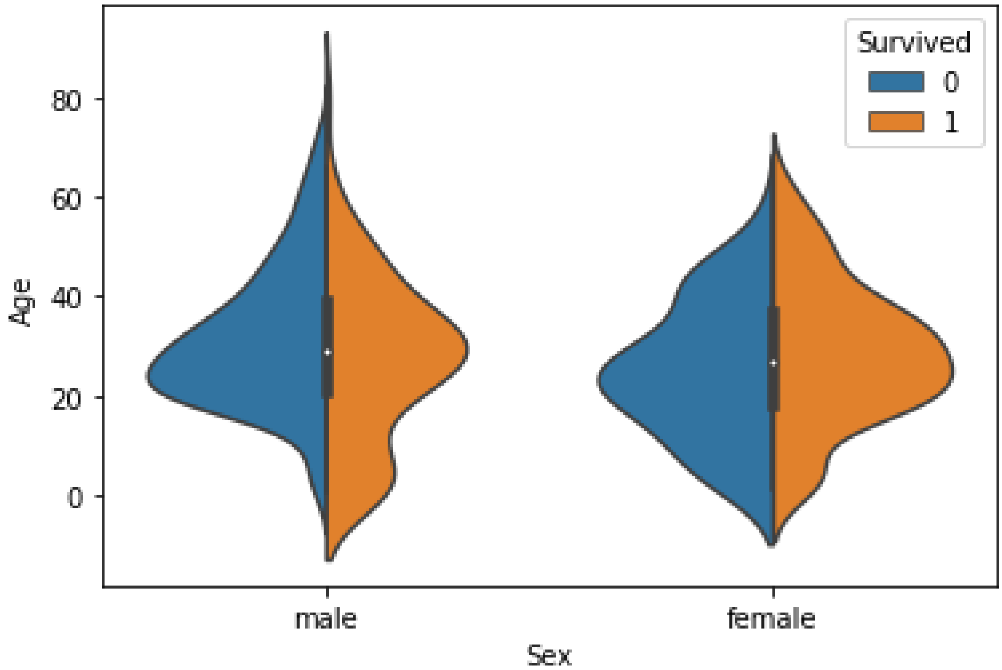
```
#checking people of which age range maximum
 survived
plt.hist(x='Age',bins=range(0,110,10),data=df.loc[df[
'Survived']==1])
plt.show()
```



It shows around 78 people of age b/w 20-30 survived

Age vs survived

sns.violinplot(x='Sex',y='Age',hue='Survived',data=df,split=True)

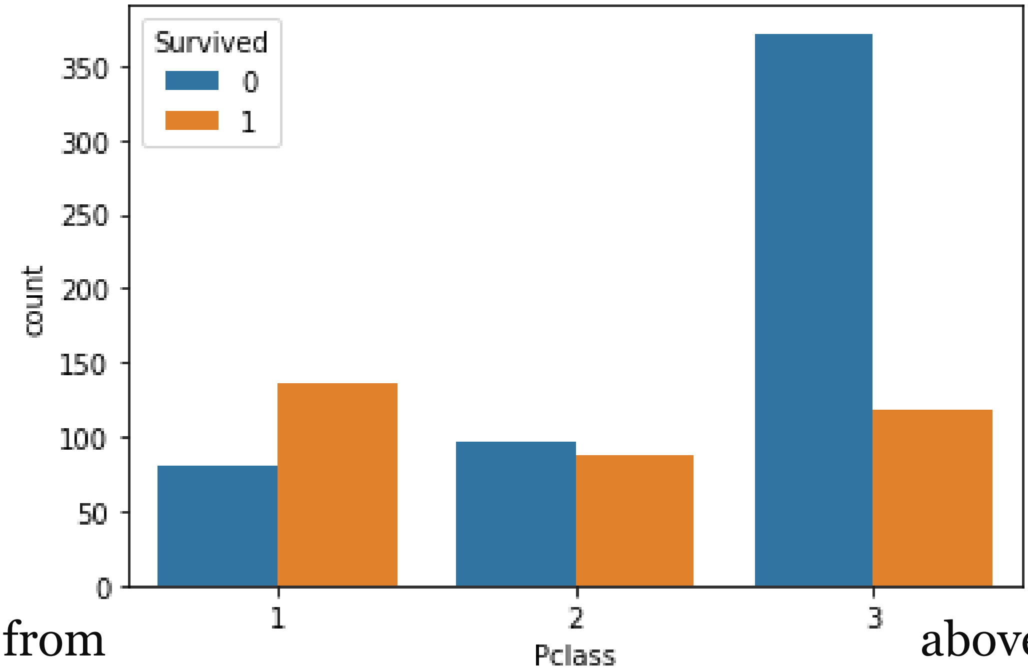


This graph gives a summary of the age range of men,women & children who were saved.
The survival rate is-

1.Good for children
2.High for women in the age range 20-50
3.Less for men as the age increases

Pclass vs survived

```
sns.barplot(x='Pclass',y='Survived',data=df,hue='Sex')
```



from above graph we find that:-

1.In 1st class ,more people survived than dying

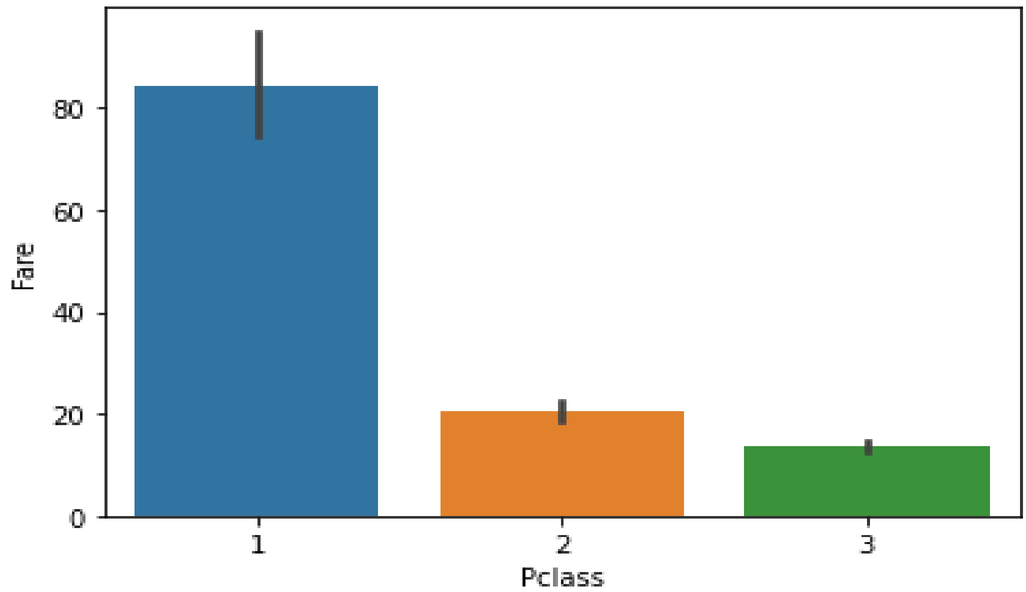
2.In 2nd class no. of people who survived were less but almost equal to not survived

3.In 3rd class no. of people who survived were far less than that who did not survived.

*** lets check what was the avg fare price for 1st 2nd & 3rd clss people**

Pclass vs Fare

```
sns.barplot(x='Pclass',y='Fare',data=df)
```



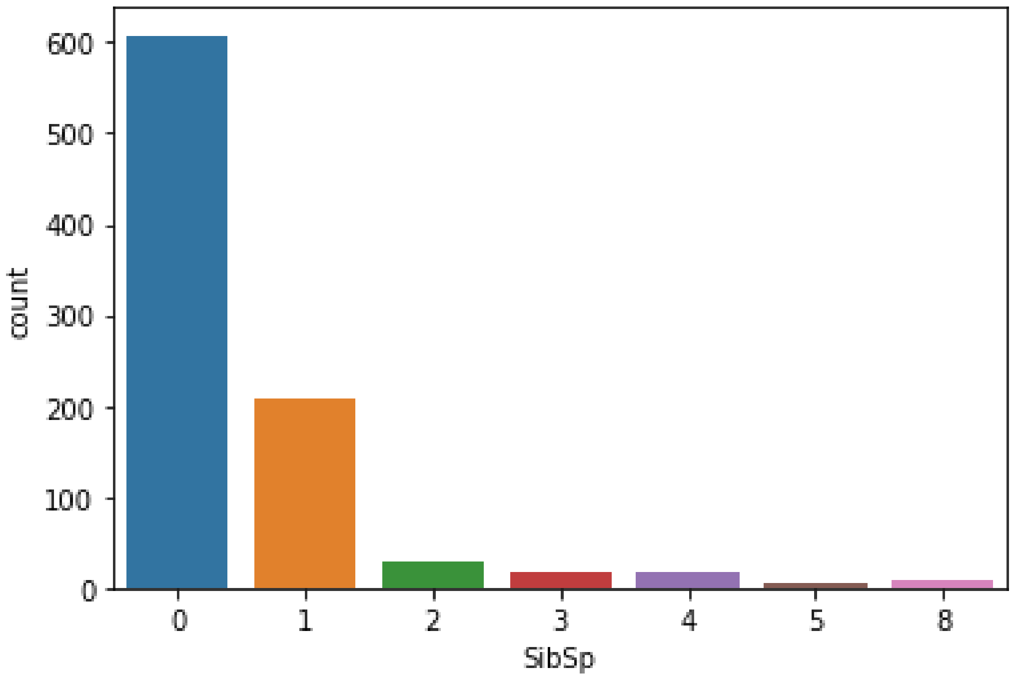
We observe that:

1.For the 1st class which is wealthier class ,the fare is quite higher

2.For 2nd class, fare is low & for 3rd class fare is very very low.

Checking sibsp column:

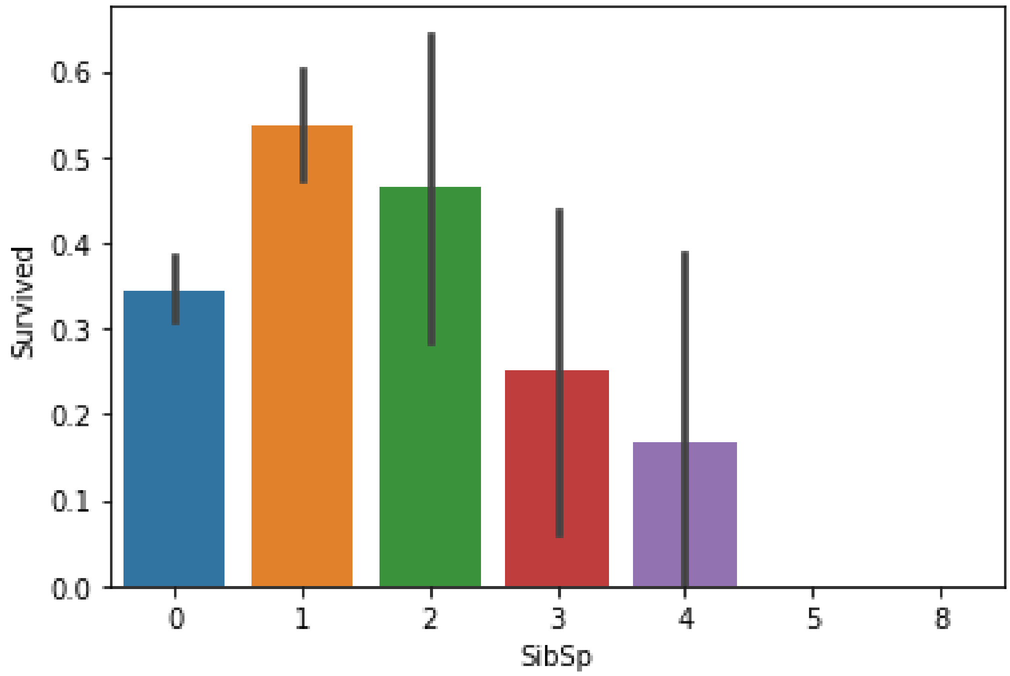
```
sns.countplot(x='SibSp',data=df)
```



Here sibsp features refers to the number of siblings or spouse the person was accompanied with.
We see that most of the people came alone.

SibSp vs Survived

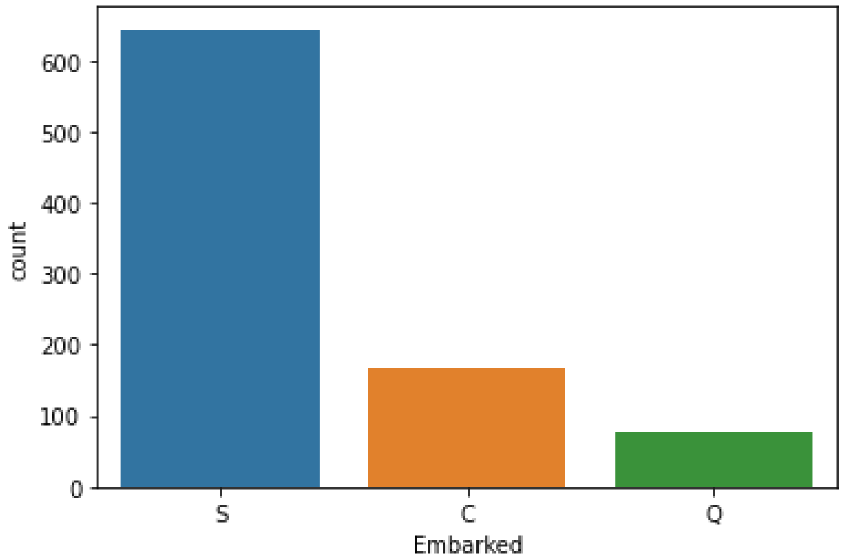
```
sns.barplot(x='SibSp',y='Survived',data=df)
```



We observe that ,with 1 or 2 relative ,the chance of survival is more.

Embarked :

```
sns.countplot(x='Embarked',data=df)
```

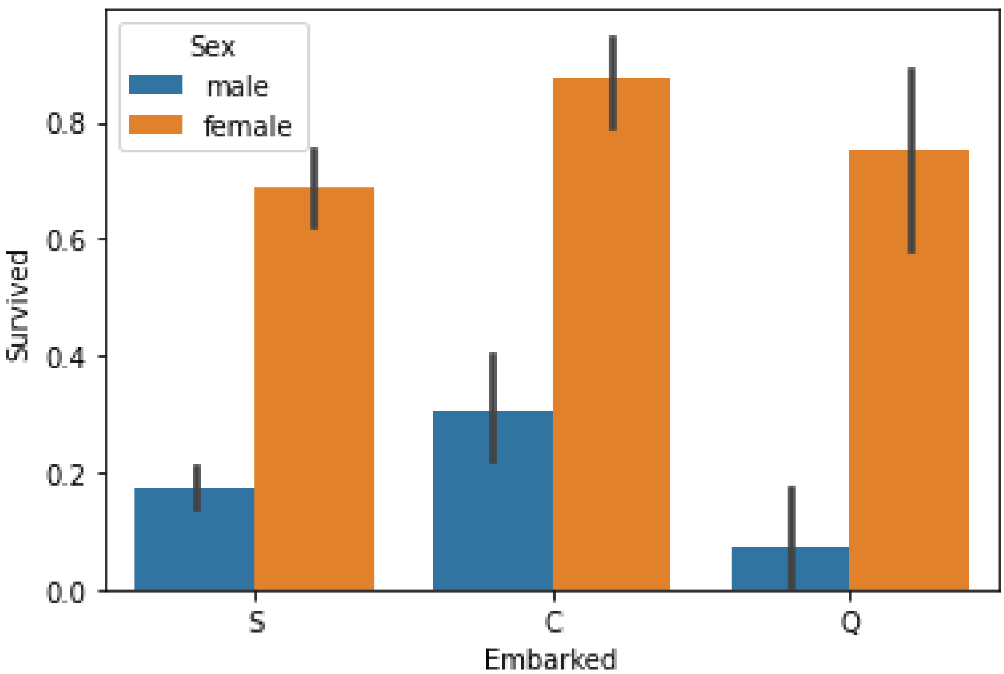


Most of the passenger boarded the ship from port S.

Few passenger boarded from port Q

Embarked vs survived w.r.t Sex

sns.barplot(x='Embarked',y='Survived',hue='Sex',data=df)



We observe that:

1.Women on all the port have a higher chance of survival.

2. Men have a high survival probability if they are on port C, but a low probability if they are on port Q or S.

Data Preprocessing

Checking the missing value:

```
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
#checking the missing percentage

df.isnull().sum()*100/len(df)
```

```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
dtype: float64
```

Observation:
1.The Embarked feature has only 2 missing values, which can easily be filled
2.Age features has 177 missing values,which we will fill
3.The ‘Cabin’ feature has 77 % of missing value.So,we can drop it.

We can also drop PassengerID,Ticket & Name as it is not of much use.

```
df.drop(['PassengerId','Name','Ticket',’Cabin’],axis=1,inplace=True)
Imputing null value

#SimpleImputer works forimputing null values in object or categorical data

from sklearn.impute import SimpleImputer

imp=SimpleImputer(strategy='most_frequent')
df['Embarked']=imp.fit_transform(df['Embarked'].values.reshape(-1,1))

im=SimpleImputer(strategy='mean')
df['Age']=im.fit_transform(df['Age'].values.reshape(-1,1))
```

Converting Features:

Sex & Embarked features are of object type,which we will convert into numerical datatype

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()


list=['Sex','Embarked']
for i in list:
    df[i]=le.fit_transform(df[i].astype(str))
df.head()
```

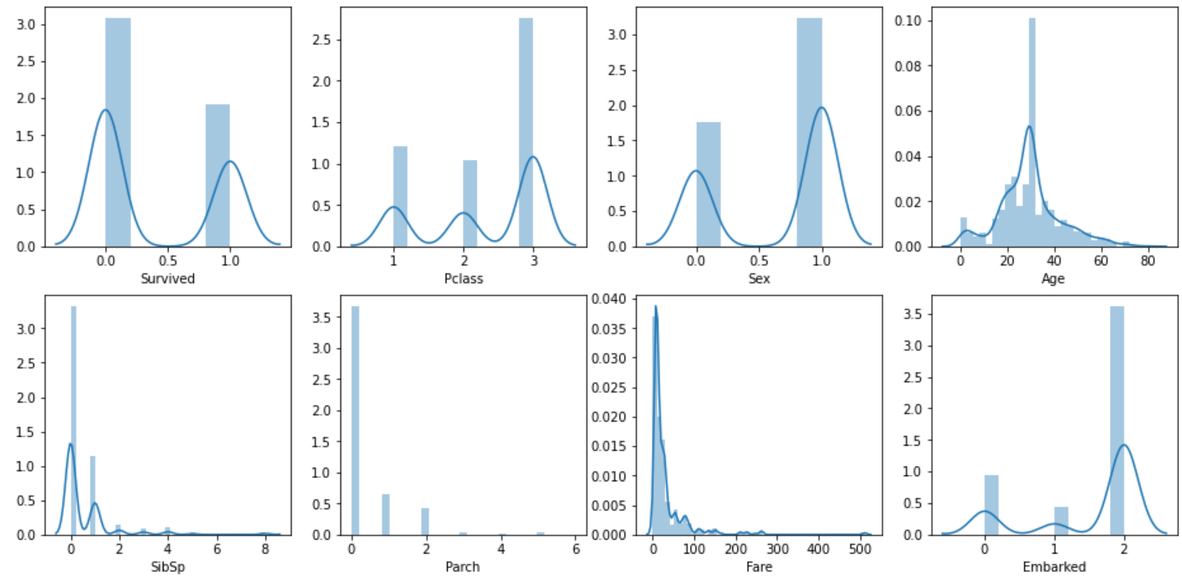
Checking skewness & outliers

Checking skewness

The skewness is a measure of how asymmetrical our data is distributed

If distribution is between -0.5 & 0.5,the distribution is approximately symmetric.

```
collist=df.columns.values
nrows=4
ncol=4

plt.figure(figsize=(16,16))
for i in range (0,len(collist)):
    plt.subplot(nrows,ncol,i+1)
    sns.distplot(df[collist[i]])
```




```
df.skew()
```

```
Survived     0.478523
Pclass      -0.630548
Sex         -0.618921
Age          0.434488
SibSp        3.695352
Parch        2.749117
Fare         4.787317
Embarked    -1.264823
dtype: float64
```

we see that the data is skewed,which we need to remove

```
##treating skewness via squareroot method and cube root method

#treating skewness via squareroot method and cube root method

df.skew()

for col in df.skew().index:

  if col in df.describe().columns:

      if df[col].skew()>0.55:

         df[col]=np.sqrt(df[col])

      if df[col].skew()<-0.55:

         df[col]=np.cbrt(df[col])
```

- Again checking skewness

```
df.skew()
```

```
Survived     0.478523
Pclass      -0.776838
Sex         -0.618921
Age          0.434488
SibSp        1.436526
Parch        1.529799
Fare         2.085004
Embarked    -1.520662
dtype: float64
```

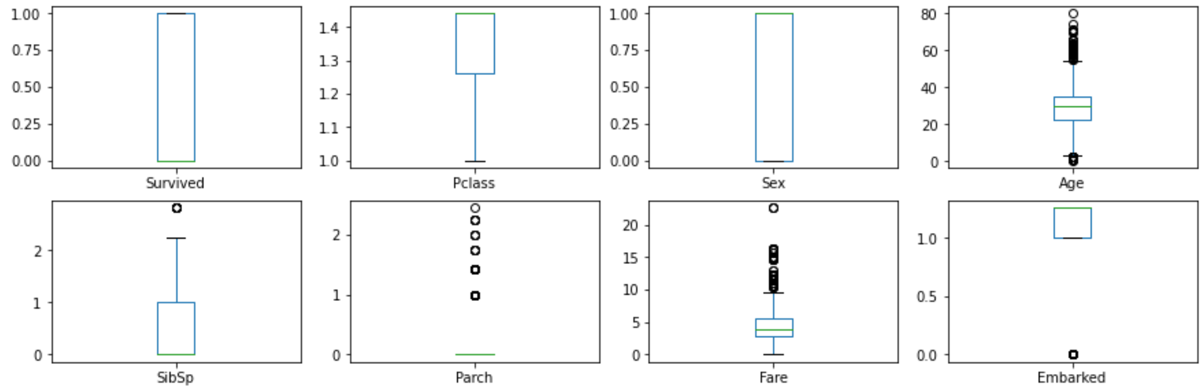
Skewness has been removed

plotting outliers

Now by using boxplot,we check if outliers are present or not.

Outliers are nothing ,but the abnormal data present in the dataset,that deviates from other observation in dataset.

```
df.plot(kind='box',subplots=True,layout=(4,4),figsize=(15,10))
```



We observe that some outliers are present,which needs to be removed.

Removing Outliers

```
from scipy.stats import zscore
z_score=abs(zscore(df))
print(df.shape)

df_new=df.loc[(z_score<3).all(axis=1)]
print(df_new.shape)
```

```
(891, 8)
(843, 8)
```

Outliers has been removed.

Now our dataset is ready to be used as input to a machine learning model.

We see that outliers have been removed.Before, dataset consist of 891 rows & 8 columns.Dataset after removal of outliers contains 843 rows & 8 columns.

```
#spliting the data into input and output variable
x=df_new.iloc[:,1:]
x.shape
```

```
(843, 7)
```

```
y=pd.DataFrame(df_new['Survived'])
y.shape
```

```
(843, 1)
```

Scaling the input variable

We apply standard scaling to make sure that all features are on same scale so that each feature is equally importa nt & make it easier to process by most ML algorithm.

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)
```

Train_Test_Split

Now let us divide the data into train & test set.
In this project ,I have divided the data into 80:20 ratio .ie training size is 80% & testing size is 20% of the whole data.

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=42)
```

```
print('x_train_shape:',x_train.shape)
print('x_test_shape:',x_test.shape)
print('y_train_shape:',y_train.shape)
print('y_test_shape:',y_test.shape
```

```
x_train_shape: (674, 7)
x_test_shape: (169, 7)
y_train_shape: (674, 1)
y_test_shape: (169, 1)
```

```
#importing our models library

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

#importing metrics
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

logistic regression

```
max_accuracy_score=0
for r_state in range(30,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=r_state,test_size=.20)
    lg=LogisticRegression()
    lg.fit(x_train,y_train)
    lg_pred=lg.predict(x_test)
    accuracy_scr=accuracy_score(y_test,lg_pred)
    if accuracy_scr>max_accuracy_score:
        max_accuracy_score=accuracy_scr
        final_r_state=r_state

print('max accuracy score corresponding to ',final_r_state,'is ',max_accuracy_score)
```

max accuracy score corresponding to 89 is 0.8402366863905325

KNeighborsClassifier

```
#using GridsearchCV to find the best parmeter in KNeighborsClassifier
```

```
parameters={'n_neighbors':range(22,30)}
knn=KNeighborsClassifier()
clf=GridSearchCV(knn,parameters)
clf.fit(x,y)

print(clf.best_params_)
```

```
{'n_neighbors': 26}
```

DecisionTreeClassifier

```
#using GridsearchCV to find the best parmeter in
DecisionTreeClassifier

parameters={'criterion':['gini','entropy'],'random_state':range(
42,100)}
dtc=DecisionTreeClassifier()
clf=GridSearchCV(dtc,parameters)
clf.fit(x,y)

print(clf.best_params_)
```

```
{'criterion': 'entropy', 'random_state': 59}
```

SVC

In [428]:

```
#gridsearchcv for svc
parameters={'kernel':['linear','rbf'],'C':[1,10],'random_state':ra
nge(35,100)}
svc=SVC()
clf=GridSearchCV(svc,parameters)
clf.fit(x,y)

print(clf.best_params_)
```

```
{'C': 1, 'kernel': 'rbf', 'random_state': 35}
```

```
#models with is best parameters
lg=LogisticRegression(random_state=89)
```

```
knn=KNeighborsClassifier(n_neighbors=26)
svc=SVC(kernel='rbf',C=1,random_state=35)
dtc=DecisionTreeClassifier(criterion='entropy',random_state=
59)


#all Algorithm by using for loop
model=[lg,knn,svc,dtc]
for m in model:
    m.fit(x_train,y_train)
    m.score(x_train,y_train)
    predm=m.predict(x_test)
    print('Accuracy score of
',m,'is:',accuracy_score(y_test,predm))
    print('\n')
    print(confusion_matrix(y_test,predm))
    print('\n')
    print(classification_report(y_test,predm))

print('******************************************************
*********************************')
    print('\n')
```

```
Accuracy score of  LogisticRegression(random_st
ate=89) is: 0.8402366863905325


[[94 10]
 [17 48]]


              precision    recall  f1-score   support

           0       0.85      0.90      0.87       104
           1       0.83      0.74      0.78        65

    accuracy                           0.84       169
   macro avg       0.84      0.82      0.83       169
weighted avg       0.84      0.84      0.84       169

***************************************************************************
**********
```

Accuracy score of **KNeighborsClassifier**(n_neighbors=26) is: **0.8461538461538461**

```
[[97  7]
 [19 46]]
```

	precision	recall	f1-score	support
0	0.84	0.93	0.88	104
1	0.87	0.71	0.78	65
accuracy			0.85	169
macro avg	0.85	0.82	0.83	169
weighted avg	0.85	0.85	0.84	169

****

Accuracy score of **SVC**(C=1, random_state=35) is: **0.8461538461538461**

```
[[96  8]
 [18 47]]
```

	precision	recall	f1-score	support
0	0.84	0.92	0.88	104
1	0.85	0.72	0.78	65
accuracy			0.85	169
macro avg	0.85	0.82	0.83	169
weighted avg	0.85	0.85	0.84	169

****

```
Accuracy score of  DecisionTreeClassifier(crite
rion='entropy', random_state=59) is: 0.75147928
99408284
```

```
[[84 20]
 [22 43]]
```

```
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       104
           1       0.68      0.66      0.67        65

    accuracy                           0.75       169
   macro avg       0.74      0.73      0.74       169
weighted avg       0.75      0.75      0.75       169
```

Cross Validation

Cross validation helps to find out the over fitting and under fitting of the model. In the cross validation the model is made to run on different subsets of the dataset which will get multiple measures of the model. If we take 5 folds, the data will be divided into 5 pieces where each part being 20% of full dataset. While running the Cross validation the 1st part (20%) of the 5 parts will be kept out as a hold out set for validation and everything else is used for training data. This way we will get the first estimate of the model quality of the dataset. In the similar way further iterations are made for the second 20% of the dataset is held as a hold out set and remaining 4 parts are used for training data during process. This way we will get the second estimate of the model quality of the dataset. These steps are repeated during the cross validation process to get the remaining estimate of the model quality.

#cross validate the models

```
models=[lg,knn,svc,dtc]
for m in models:
    score=cross_val_score(m,x,y,cv=5,scoring='accuracy')
    print('Model:',m)
    print('\n')
    print('score:',score)
    print('mean_score:',score.mean())
    print('standard deviation:',score.std())
print('******************************************************************************************')
    print('\n')
```

Model: LogisticRegression(random_state=89)

```
score: [0.78106509 0.77514793 0.76923077 0.78571429 0
.79761905]
mean_score: 0.7817554240631164
standard deviation: 0.009678116247374377
```

```
*****************************************************
************************************

Model: KNeighborsClassifier(n_neighbors=26)

score: [0.81065089 0.80473373 0.78698225 0.79166667 0
.82142857]
mean_score: 0.8030924204001127
standard deviation: 0.012538933263509954
*****************************************************
************************************

Model: SVC(C=1, random_state=35)

score: [0.82840237 0.80473373 0.80473373 0.81547619 0
.85714286]
mean_score: 0.8220977740208509
standard deviation: 0.019569217215636692
*****************************************************
************************************

Model: DecisionTreeClassifier(criterion='entrop
y', random_state=59)

score: [0.75147929 0.76331361 0.85207101 0.75595238 0
.80357143]
mean_score: 0.7852775429698508
standard deviation: 0.03815949253582776
*****************************************************
************************************
```

Using ensemble technique to boost up our score RandomForestClassifier

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

```
rf=RandomForestClassifier(n_estimators=50,random_stat
e=42)

rf.fit(x_train,y_train)
predrf=rf.predict(x_test)
print(accuracy_score(y_test,predrf))
print(confusion_matrix(y_test,predrf))
print(classification_report(y_test,predrf,labels=[0,1
]))
```

```
0.8224852071005917
[[95  9]
 [21 44]]
              precision    recall  f1-score   support

           0       0.82      0.91      0.86       104
           1       0.83      0.68      0.75        65

    accuracy                           0.82       169
   macro avg       0.82      0.80      0.80       169
weighted avg       0.82      0.82      0.82       169
```

AdaBoost Classifier

```
from sklearn.ensemble import AdaBoostClassifier

ad=AdaBoostClassifier(n_estimators=50,algorithm='SAMM
E.R')
ad.fit(x_train,y_train)
ad_pred=ad.predict(x_test)
print(accuracy_score(y_test,ad_pred))
print(confusion_matrix(y_test,ad_pred))
print(classification_report(y_test,ad_pred))
```

```
0.8106508875739645
[[90 14]
 [18 47]]
              precision    recall  f1-score   support

           0       0.83      0.87      0.85       104
```

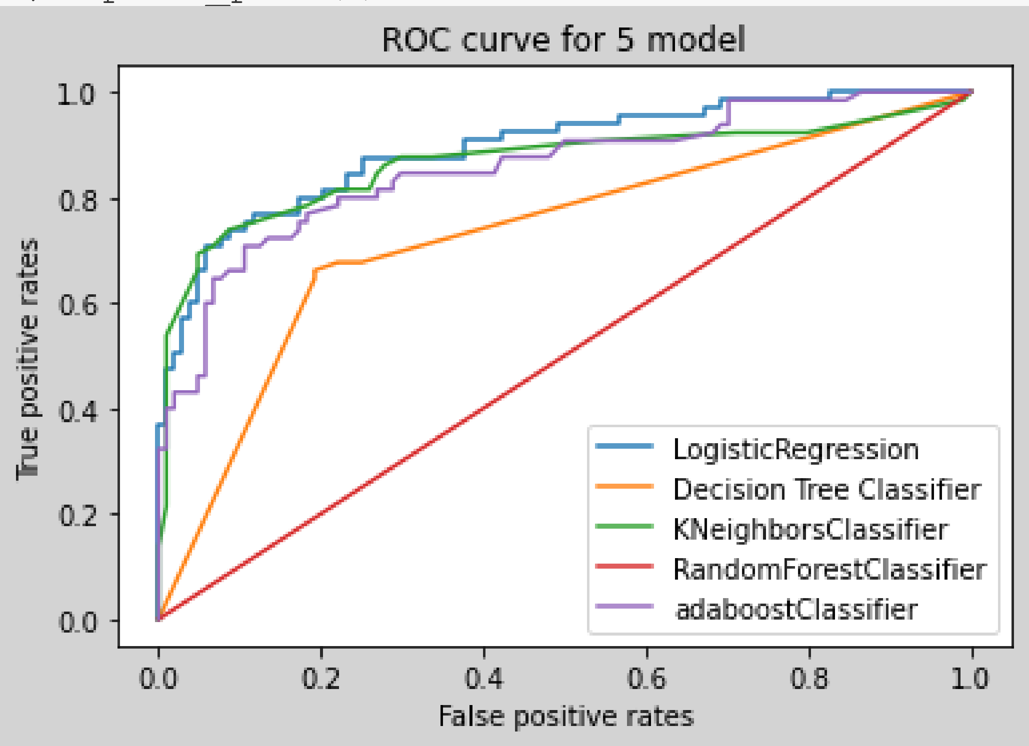
```
           1       0.77      0.72      0.75        65

    accuracy                           0.81       169
   macro avg       0.80      0.79      0.80       169
weighted avg       0.81      0.81      0.81       169
```

AUC ROC CURVE

ROC curve is nothing but a graph displaying the perfor mance of classification model.
AUC ROC plot is used to visualise the performace of a binary classifier.
More the aea is under the curve ,better the model is working.

```
lgpred_prob=lg.predict_proba(x_test)[:,1]
dtcpred_prob=dtc.predict_proba(x_test)[:,1]
knnpred_prob=knn.predict_proba(x_test)[:,1]
rfpred_prob=rf.predict_proba(x_test)[:,1]
adpred_prob=ad.predict_proba(x_test)[:,1]


from sklearn.metrics import roc_curve
lg_tpr,lg_fpr,lg_thresholds=roc_curve(y_test,lgpred_p
rob)
dtc_tpr,dtc_fpr,dtc_thresholds=roc_curve(y_test,dtcpr
ed_prob)
knn_tpr,knn_fpr,knn_thresholds=roc_curve(y_test,knnpr
ed_prob)
rf_tpr,rf_fpr,rf_threshold=roc_curve(y_test,rfpred_pr
ob)
ad_tpr,ad_fpr,ad_threshold=roc_curve(y_test,adpred_pr
ob)

plt.plot(lg_tpr,lg_fpr,label='LogisticRegression')
plt.plot(dtc_tpr,dtc_fpr,label ='Decision Tree Classi
fier')
plt.plot(knn_tpr,knn_fpr,label='KNeighborsClassifier'
)
```

```
plt.plot(rf_tpr,rf_tpr,label='RandomForestClassifier'
)
plt.plot(ad_tpr,ad_fpr,label='adaboostClassifier')

plt.xlabel('False positive rates')
plt.ylabel('True positive rates')
plt.title('ROC curve for 5 model')
plt.legend(loc='best')
plt.show()

from sklearn.metrics import roc_auc_score
print('LG AUC score',roc_auc_score(y_test,lgpre
d_prob))
print('DTC AUC SCORE',roc_auc_score(y_test,dtcp
red_prob))
print('KNN auc score',roc_auc_score(y_test,knnp
red_prob))
print('Random forest classifier',roc_auc_score(
y_test,rfpred_prob))
print('Adaboost classifier',roc_auc_score(y_tes
t,adpred_prob))
```



```
LG AUC score 0.8926035502958579
DTC AUC SCORE 0.7298076923076923
KNN auc score 0.8656065088757398
Random forest classifier 0.8610946745562131
Adaboost classifier 0.8560650887573964
```

Higher the AUC ,better the model is working

LogisticRegression can be used to classify ,who survived or not ,or what factors make people more likely to survive.