In this project, we are going to work with titanic dataset which is obtained from kaggle website and predict using machine learning model whether the passenger onboard the ship will survive the disaster or not. The main focus of this article is on the step by step procedure that will be needed to be executed in order to accomplish the goal.

**1. Problem Definition**

1. **Project Overview**

The sinking of the titanic is regarded as one of the most infamous incident of the 20th century. During the ship wreckage, many people had to lose their life due to shortage of lifeboats onboard. It was later found that, apart from luck, there were several other factors that contributed to the passenger’s survival.

1. **Problem Statement**

The goal of this project is to predict whether a passenger will survive or die in the titanic ship sinking. This project is approached as a standard supervised classification problem, where the output label y is a binary variable i.e. 1 (if the passenger survives sinking) and 0 (if the passenger fails to survive).

**2. Data Analysis**

Data analysis is the process that is used to discover patterns and trends in any given dataset by summarizing the information/data using the statistical table and visual graphs. We can begin this process by importing all the essential libraries that will be needed to undertake this project.

1. **Importing Libraries**

import warnings

warnings.filterwarnings('ignore')

import pandas as pd

import numpy as np

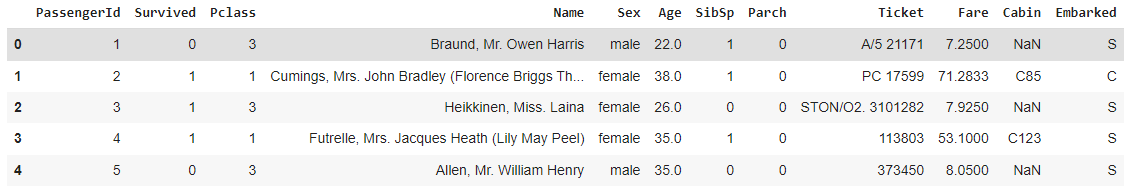
import matplotlib.pyplot as plt

import seaborn as sns

1. **Importing Dataset**

df=pd.read\_csv("C:\Users\RedPr\Downloads\titanic\_train.csv")

df.head()



By looking at the dataset, it is now clear that the ‘**Survived**’ column is our target feature (i.e. the Y label) that will be used in model prediction.

### Data Overview

### df.shape

****

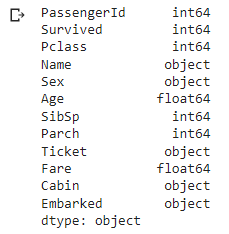
The dataset contains 891 rows and 12 columns. Next we observe the list of columns in the dataset.

df.columns



Since there are no missing column names in the entire list, we proceed ahead to figure out the data type of each column present in the dataset.

df.dtypes

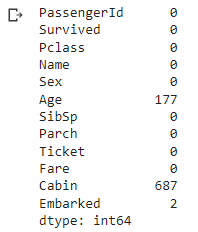


It can be observed that our dataset contains three different types of data namely,

* object type (which are usually called the categorical type),
* integer type (int64) and
* float type (float64)

1. **Handling Missing Values**

df.isnull().sum()



The number of null values that are detected in the columns: Age, Cabin and Embarked are 177, 687 and 2 respectively.

Once the null values are located, we need to replace it with an alternative value; either using mean, if the data type of the column is integer/float type or using mode if the column is object type. Scikit-learn class such as simple imputer can be used to perform this task.

from sklearn.impute import SimpleImputer

imp=SimpleImputer(missing\_values=np.nan,strategy='mean')

df["Age"]=imp.fit\_transform(df["Age"].values.reshape(-1,1))

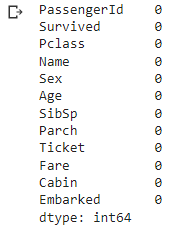
imp1=SimpleImputer(strategy='most\_frequent')

df["Cabin"]=imp1.fit\_transform(df["Cabin"].values.reshape(-1,1))

df["Embarked"]=imp1.fit\_transform(df["Embarked"].values.reshape(-1,1))

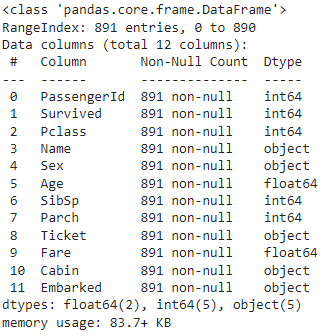
Once the operation is executed on the missing values, we can cross check the implementation that has taken place using isnull().sum() method.

df.isnull().sum()



Sometimes the columns are labeled with incorrect data types or the symbol such as “?” are present in the column instead of numeric value, which makes it difficult for isnull() method to detect such values. In such scenario info() method helps us get clear picture of the dataset.

df.info()



There are no symbol value (such as “?”) present in the numeric column and all the columns are labeled with correct data types.

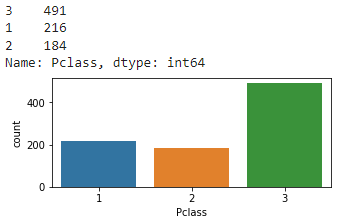
1. **Visualization**

Data Visualization is a technique that is widely used to analyze the data in the form of graphs or maps, so that it can be easier to understand and interpret the underlying data trends and patterns.

**Univariate Analysis** - In this type of analysis, only a single variable is used to analyze the data. It is the easiest form of analysis, since it does test any relationship with other variables.

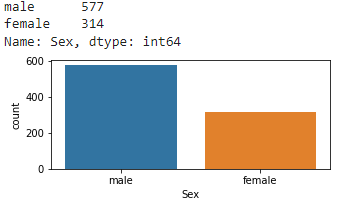
sns.countplot(df["Pclass"])

print(df["Pclass"].value\_counts())



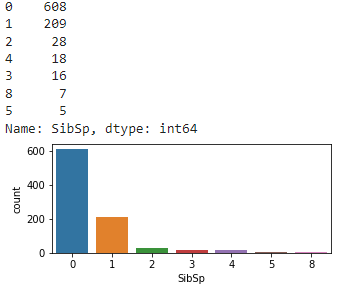
sns.countplot(df["Sex"])

print(df["Sex"].value\_counts())



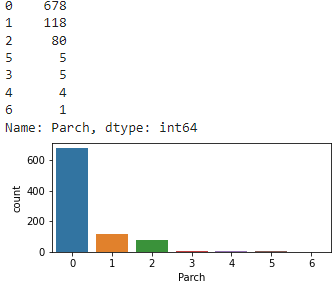
sns.countplot(df["SibSp"])

print(df["SibSp"].value\_counts())



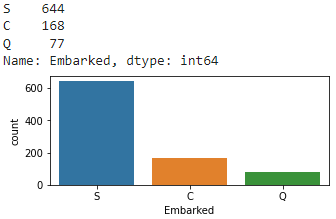
sns.countplot(df["Parch"])

print(df["Parch"].value\_counts())



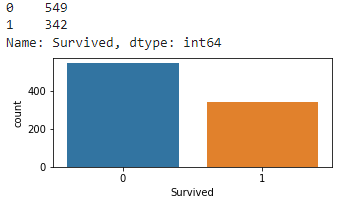
sns.countplot(df["Embarked"])

print(df["Embarked"].value\_counts())



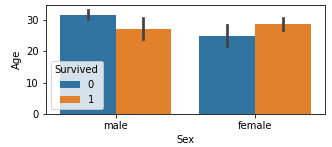
sns.countplot(df["Survived"])

print(df["Survived"].value\_counts())

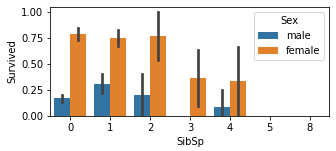


**Bivariate Analysis** - In this type of analysis, two variables are used to analyze the data and test if any relationship with other variables exists.

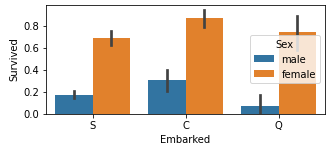
sns.barplot(y=df["Age"],x=df["Sex"],hue=df["Survived"])



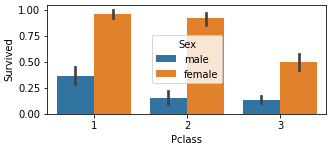
sns.barplot(x=df["SibSp"],y=df["Survived"],hue=df["Sex"])



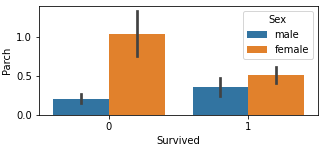
sns.barplot(x=df["Embarked"],y=df["Survived"],hue=df["Sex"])



sns.barplot(x=df["Pclass"],y=df["Survived"],hue=df["Sex"])



sns.barplot(y=df["Parch"],x=df["Survived"],hue=df["Sex"])



sns.pairplot(df)

If needed, we can also use pair plot seaborn method to visualize the distribution of both the single variable and multiple variables that are having relationship, all together under a single visual frame.

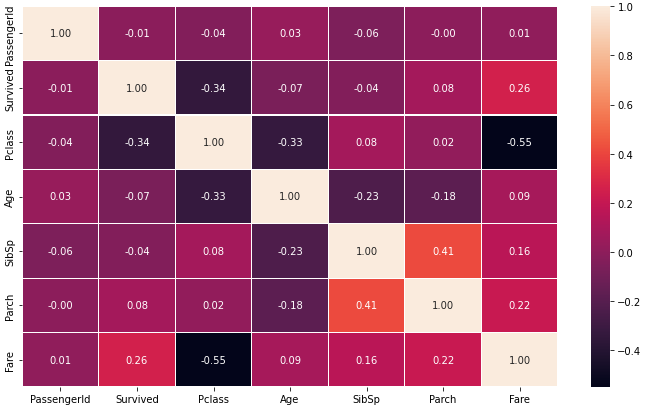
1. **Data Correlation**

Data correlation is a technique that is used to understand how one set of features corresponds to another set of features in the dataset.

Using heatmap (a seaborn visualization method), we can quickly learn how much each feature (of numeric datatype) is correlated with the target feature.

plt.figure(figsize=(12,7))

sns.heatmap(df.corr(),annot=True,linewidths=0.1,linecolor="White",fmt=".2f")

****

It is observed that,

- Parch has 8 percent correlation with target column which is considered as a very weak +ve bond.

- Fare has 26 percent correlation with target column which is considered as a weak +ve bond.

- SibSp has -4 percent correlation with target column which is considered as a very weak -ve bond.

- Age has -7 percent correlation with target column which is considered as a very weak -ve bond.

- Pclass has -34 percent correlation with target column which is considered as a weak -ve bond.

- PassengerId has -1 percent correlation with target column which is considered as a very weak -ve bond

Furthermore we can also use bar graph to plot correlation of each feature in the dataset with the target column (Survived), in descending order.

plt.figure(figsize=(15,7))

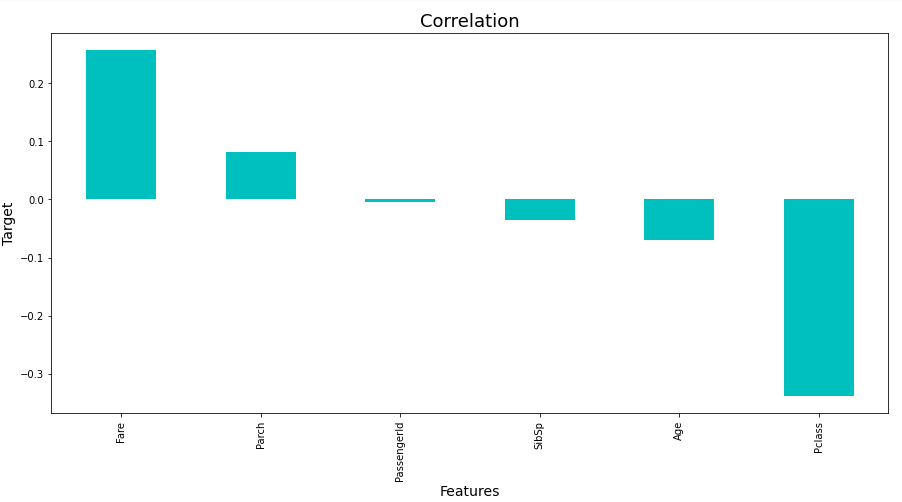
df.corr()['Survived'].sort\_values(ascending=False).drop(['Survived']).plot(kind='bar',color='c')

plt.xlabel('Features',fontsize=14)

plt.ylabel('Target',fontsize=14)

plt.title('Correlation',fontsize=18)

plt.show()

****

It is observed that for negative correlation with target column,

- The maximum -ve correlation belonged to Pclass feature

- The minimum -ve correlation belonged to PassengerId feature

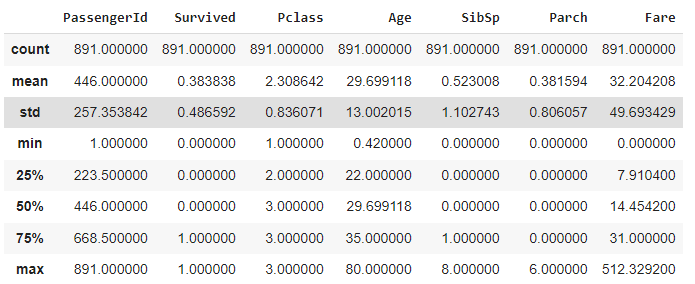
And for positive correlation with target column,

- The maximum +ve correlation belonged to Fare feature

- The minimum +ve correlation belonged to Parch feature

1. **Data Description**

df.describe()



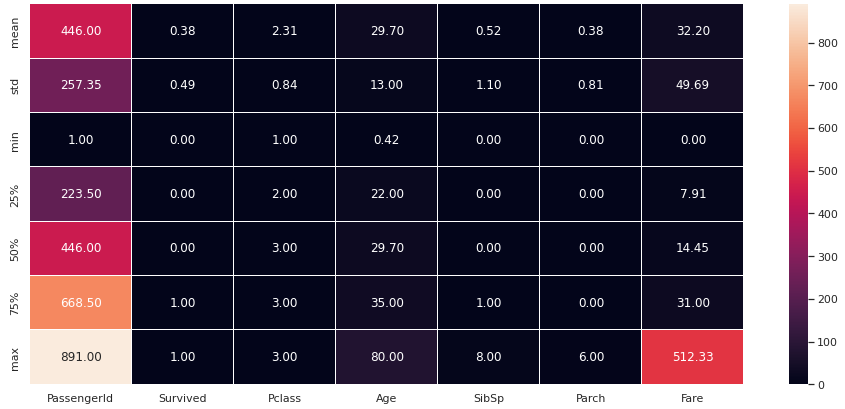
In this step, we use data description method known as describe() to get statistical information from each numeric column such as

* count – This gives us number of non null values.
* mean – The gives us average value of the column.
* std - The standard deviation value.
* min – The smallest value in the column.
* 25% - Indicates 25th percentile
* 50% - Indicates 50th percentile.
* 75% - Indicates 75th percentile.
* max – The Largest value.

import matplotlib.pyplot as plt

plt.figure(figsize=(16,7))

sns.heatmap(df.describe()[1:],annot=True,linewidths=0.1,linecolor="White",fmt=".2f")



It is observed that

- There is huge gap between value at 75th percentile and Max value which indicates presence of outlier in some of the features.

- Data is also widely spread in PassengerId, Fare and Age compared to other features.

1. **Handling Outliers**

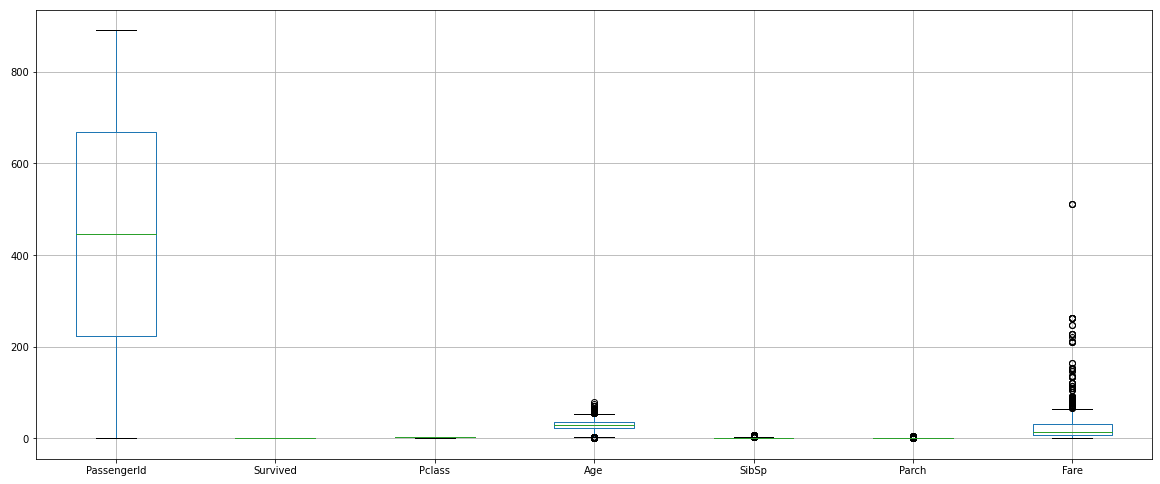
The outliers or the abnormal data points that are responsible for the skewness in the data are removed from the dataset in this phase. To do this, we first use the boxplot graph to plot all the columns as seen below,

import matplotlib.pyplot as plt

df.iloc[:,0:13].boxplot(figsize=(20,8))

plt.subplots\_adjust(bottom=0.1)

plt.show()



( Note: The Black dot that appear on the graph indicate the presence of outlier in that particular column )

It is observed that

* SibSp and Parch are categorical feature and hence their outliers cannot be removed
* But since age and Fare are numerical features their outliers can be handled.

Once the outliers are detected, we can use flooring and capping technique where outlier are either floored or capped depending on whether they are above upper control limit (UCL) or below lower control limit (LCL).

In our case, since we use 75th percentile value as the upper control limit and 25th percentile value as lower control limit for each feature, any value exceeding the UCL will be capped or replaced by 75th percentile value and any value below LCL will be floored or replaced by 25th percentile value.

capped\_df = df.copy()

def remove\_outliers(n):

Q1 = np.percentile(df[n], 25, interpolation = 'midpoint')

Q3 = np.percentile(df[n], 75, interpolation = 'midpoint')

iqr = Q3 - Q1

percentile25 = df[n].quantile(0.25)

percentile75 = df[n].quantile(0.75)

upper\_limit\_val = percentile75 + 1.5 \* iqr

lower\_limit\_val = percentile25 - 1.5 \* iqr

df[df[n] > upper\_limit\_val]

df[df[n] < lower\_limit\_val]

capped\_df [n] = np.where(

capped\_df [n] > upper\_limit\_val,

upper\_limit\_val,

np.where(

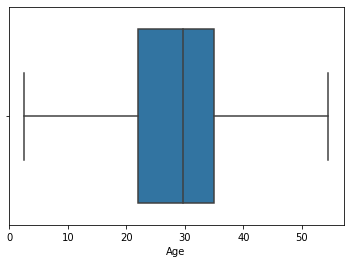
        capped\_df [n] < lower\_limit\_val,

        lower\_limit\_val,

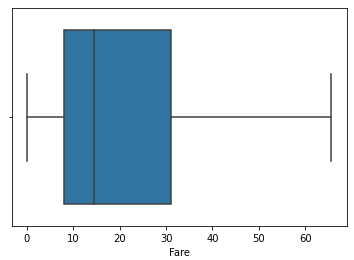
        capped\_df[n] ) )

sns.boxplot(capped\_df[n])

remove\_outliers("Age")



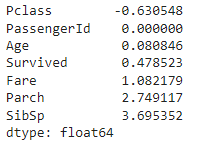
remove\_outliers("Fare")



1. **Handling Skewness**

Whenever the data is not equally spread along the mean, it results in skewness, that can be visualized using the bell shaped curve (a graph depicting feature distribution)

capped\_df.skew().sort\_values()

****

Keeping +/-0.5 as the range of skewness, it is observed that following features have skewness

- Pclass

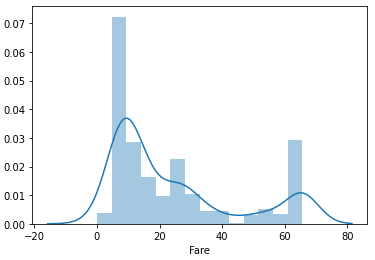
- Fare

- Parch

- SibSp

Since all the features mentioned in above list (except Fare) are categorical in nature and hence their skewness cannot be removed.

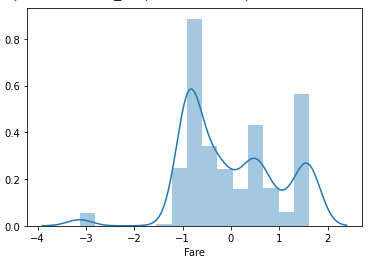
sns.distplot(capped\_df["Fare"])



from sklearn.preprocessing import power\_transform

capped\_df["Fare"]=power\_transform(capped\_df["Fare"].values.reshape(-1,1))

sns.distplot(capped\_df["Fare"])



Since the distribution is not normal in graph, we further apply transformation method to see the results.

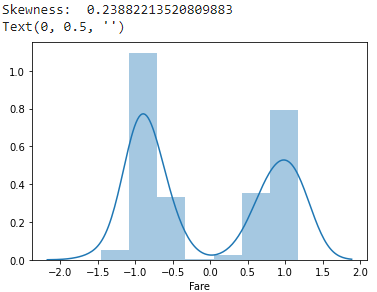
k=np.cbrt(new\_df\_cap["Fare"])

k=np.cbrt(k)

sns.distplot(k)

print("Skewness: ",k.skew())

plt.ylabel("")



capped\_df["Fare"]=k

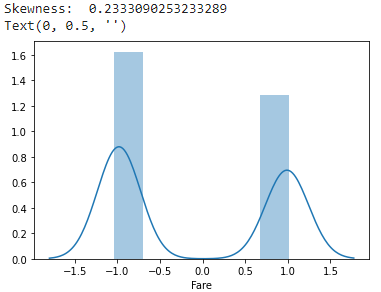
j=np.cbrt(new\_df\_cap["Fare"])

j=np.cbrt(j)

sns.distplot(j)

print("Skewness: ",j.skew())

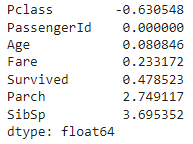
plt.ylabel("")



The distribution does not improve any further but since the skewness is within the range of +/-0.5, hence we decide to keep the feature.

capped\_df["Fare"]=j

capped\_df.skew().sort\_values()



**3) EDA Concluding Remarks**

* Null values were detected in Age, Cabin and Embarked columns
* Some features such as Name, Passenger Id, Ticket are redundant for analysis
* The feature with strongest positive correlation with target feature is Fare
* The feature with strongest negative correlation with target feature is Pclass
* There were total of 891 passengers onboard the ship, of which 577 were male and 314 female.
* Total of 549 passengers died and 342 passengers survived.
* Majority passengers (644) that survived the disaster embarked on the journey from location C.
* 678 survived passengers did not have parents/children onboard.
* Also, the highest number of passenger that survived the disaster were from passenger class 1.
* Among female survivors, the passengers with no siblings/spouse had the highest survival count whereas among male survivors, the passengers with 1 sibling/spouse had highest survival count.

**4) Preprocessing Pipeline**

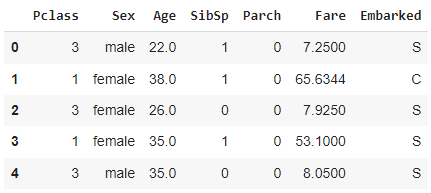
In this phase, using preprocessing steps the raw data is made suitable for a machine learning model.

1. **Separating Input and Output features**

There are certain columns which are not of much importance as they don’t contribute any significant input in the prediction and hence we drop these from dataset.

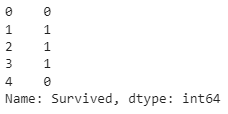
x=capped\_df.drop(["PassengerId"  ,"Survived","Name","Ticket","Cabin"],axis=1)

x.head()

****

y=capped\_df ["Survived"]

y.head()

****

1. **Encoding categorical features**

Next, we convert all the categorical features in the dataset to numeric type so that it can be easily fitted into the model.

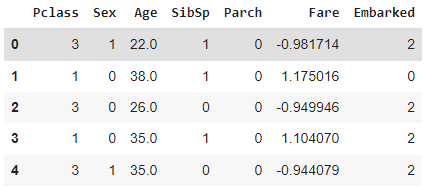
from sklearn.preprocessing import OrdinalEncoder

enc=LabelEncoder()

x["Sex"]=enc.fit\_transform(x["Sex"].values.reshape(-1,1))

x["Embarked"]=enc.fit\_transform(x["Embarked"].values.reshape(-1,1))

x.head()



1. **Checking Multicollinearity**

In order to avoid two or more input features from feeding same information into the model, it is necessary to detect correlation between independent variables using Variance Inflation factor or VIF method.

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

def calc\_vif(X):

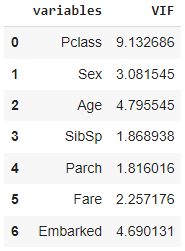
    vif = pd.DataFrame()

    vif["variables"] = X.columns

    vif["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)

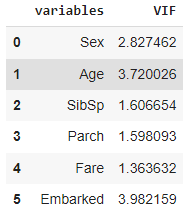
calc\_vif(x)



Pclass have VIF>5, indicating presence of multicollinearity in datastet.

X = x.drop(["Pclass"],axis=1)

calc\_vif(X)



1. **Class Distribution**

Number of samples that belong to each class is referred to as class distribution. Whenever there is a bias or imbalance in this distribution, a sampling technique is used to balance the class; in this case oversampling is used, a technique that creates duplicate rows of minority class to balance the distribution. It is used there is less than sufficient amount of data.

y.value\_counts()



from imblearn.over\_sampling import SMOTE

sm = SMOTE()

x\_new, y\_new = sm.fit\_resample(X, y)

y=y\_new

y.value\_counts()



1. **Feature Scaling**

Feature scaling is a technique frequent used in machine learning preprocessing to generalize the data points so that the distance between them is smaller.The **Min-Max Normalization** uses distribution value between 0 and 1 to re-scales all the feature values.

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import r2\_score

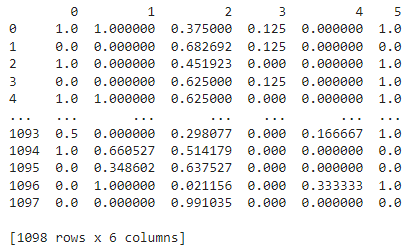
from sklearn.model\_selection import train\_test\_split

mns=MinMaxScaler()

s=mns.fit\_transform(x\_new)

x=pd.DataFrame(data=s)

print(x)



**5) Building Machine Learning Model**

In this phase, different classification models are built and tested out on the dataset to determine the best suitable model.

1. **Logistic Regression**

**Training and Testing**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

lg=LogisticRegression()

for i in range(0,1000):

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=i)

lg.fit(x\_train,y\_train)

pred\_train=lg.predict(x\_train)

pred\_test=lg.predict(x\_test)

if  round(accuracy\_score(y\_train,pred\_train)\*100,1)==round(accuracy\_score(y\_test,pred\_test)\*100,1):

print("At random state",i)

print("training accuracy: ",round(accuracy\_score(y\_train,pred\_train)))

print("testing accuracy:", round(accuracy\_score(y\_test,pred\_test)))

print("\n")

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=877)

lg.fit(x\_train,y\_train)

pred\_train=lg.predict(x\_train)

pred\_test=lg.predict(x\_test)

print("Train Accuracy:",accuracy\_score(y\_train,pred\_train))

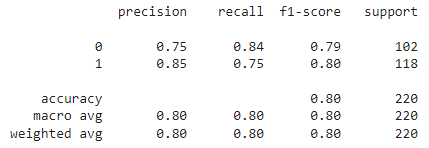
print("Test Accuracy:",accuracy\_score(y\_test,pred\_test))

from sklearn.metrics import classification\_report

print(classification\_report(y\_test,pred\_test))







**Cross Validation**

Cross-validation is a technique that is frequently used to test the model efficiency by training it on subsets of the available input data and then testing on the unseen data.

pred\_lg=lg.predict(x\_test)

lss=accuracy\_score(y\_test,pred\_lg)

from sklearn.model\_selection import cross\_val\_score

for j in range(2,10):

    cv\_score=cross\_val\_score(lg,x,y,cv=j)

    print(cv\_score)

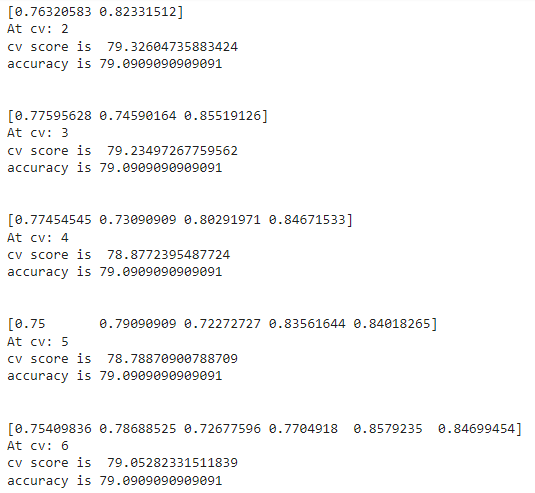
    cv\_mean=cv\_score.mean()

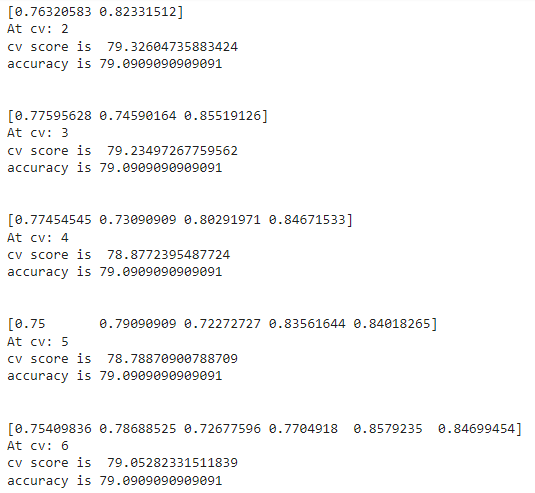
    print("At cv:",j)

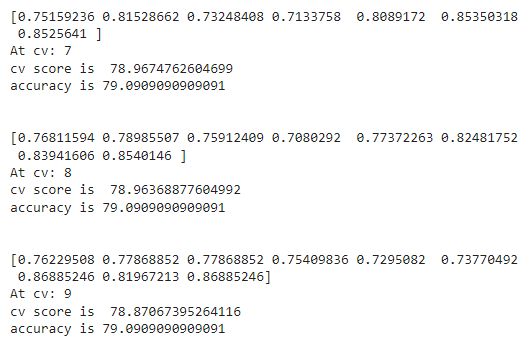
    print("cv score is ",cv\_mean\*100)

    print("accuracy is", lss\*100)

    print("\n")







**AUC ROC Curve**

The AUC - ROC curve is a performance measurement model based graph that is used for classification problems. In this graph, the ROC or Receiver Operating Characteristics represents the probability curve whereas AUC or Area Under Curve represents the distinguishing capability of the model between the classes.

from sklearn.metrics import roc\_curve,auc

fpr, tpr, thresholds = roc\_curve(pred\_test,y\_test)

roc\_auc= auc(fpr,tpr)

plt.figure()

plt.plot(fpr,tpr,color='darkorange',lw=10,label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0,1],[0,1],color='navy',lw=10,linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0,1.05])

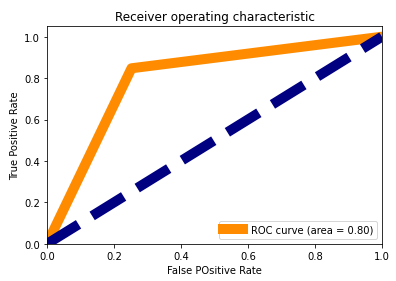
plt.xlabel("False POsitive Rate")

plt.ylabel("True Positive Rate")

plt.title("Receiver operating characteristic")

plt.legend(loc='lower right')

plt.show()

****

We can create a function for model selection and call it each time we need to test a new model.

**User Defined Function For Selecting Model**

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve,auc

def model\_selection(algorithm\_instance,x\_train,y\_train,x\_test,y\_test):

algorithm\_instance.fit(x\_train,y\_train)

model\_pred\_train=algorithm\_instance.predict(x\_train)

model\_pred\_test=algorithm\_instance.predict(x\_test)

print("Accuracy of training model :",round(accuracy\_score(y\_train,model\_pred\_train)\*100,2))

print("Accuracy of test data :",round(accuracy\_score(y\_test,model\_pred\_test)\*100,2))

cv\_score=cross\_val\_score(algorithm\_instance,x,y,cv=5)

cv\_mean=cv\_score.mean()

print("cv score is ",round(cv\_mean\*100,2))

print("\nClassification report for test data\n",classification\_report(y\_test,model\_pred\_test))

print("Classification report for training data\n",classification\_report(y\_train,model\_pred\_train))

print("Confusion Matrix\n",confusion\_matrix(y\_test,model\_pred\_test))

print("\n")

fpr, tpr, thresholds = roc\_curve(model\_pred\_test,y\_test)

roc\_auc= auc(fpr,tpr)

plt.figure()

plt.plot(fpr,tpr,color='darkorange',lw=10,label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0,1],[0,1],color='navy',lw=10,linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0,1.05])

plt.xlabel("False POsitive Rate")

plt.ylabel("True Positive Rate")

plt.title("Receiver operating characteristic")

plt.legend(loc='lower right')

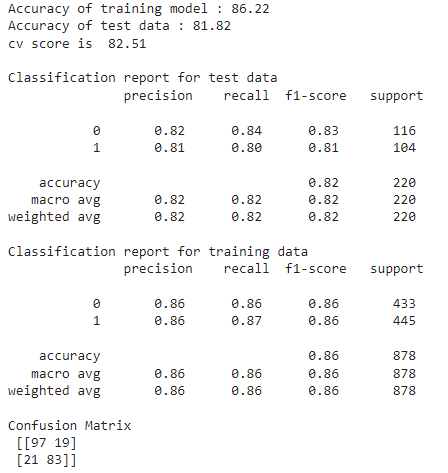
plt.show()

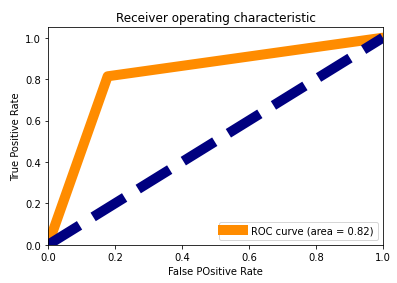
1. **KNeighbors Classifier**

from sklearn.neighbors import KNeighborsClassifier

k=KNeighborsClassifier()

model\_selection(k,x\_train,y\_train,x\_test,y\_test)



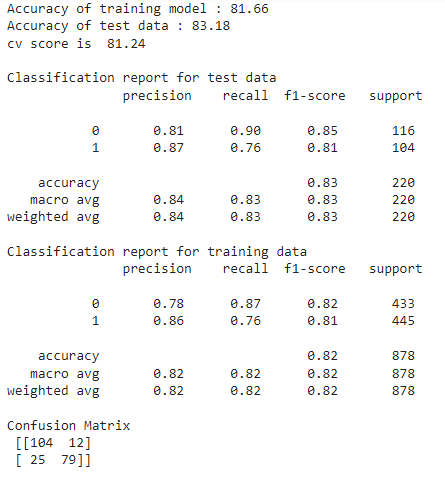
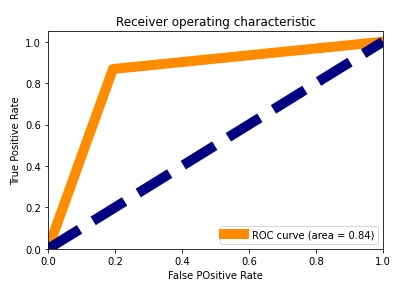
****

1. **SVM**

from sklearn import svm

s=svm.SVC()

model\_selection(s,x\_train,y\_train,x\_test,y\_test)

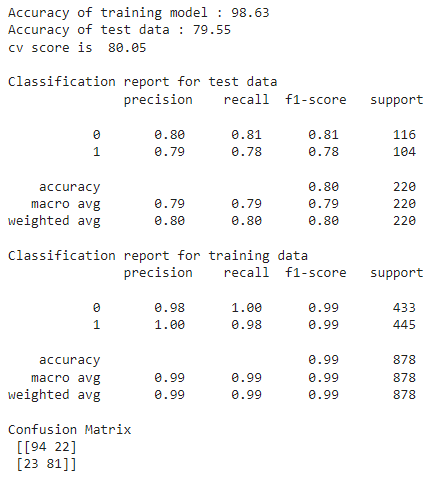
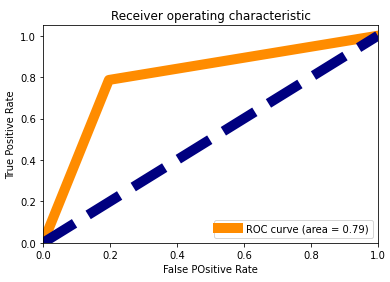
 

1. **Decision Tress Classifier**

from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()

model\_selection(dtc,x\_train,y\_train,x\_test,y\_test)

** **

**Ensemble Method**

Instead of using single model ensemble method allows us to combine multiple methods in order to achieve greater accuracy.

By testing different combination of various specified parameters, the grid search technique returns a list of hyper parameters that can be used to fine tune a model.

1. **Random Forest Classifier**

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

parameter={'criterion':['gini', 'entropy', 'log\_loss'],

          'max\_features' : [None,"sqrt","log2"],

           'class\_weight':['balanced',' balanced\_subsample'],

           'oob\_score':['True','False'],

           'n\_estimators':range(0,100,10)}

rf=RandomForestClassifier()

clf=GridSearchCV(rf,parameter)

clf.fit(x\_train,y\_train)

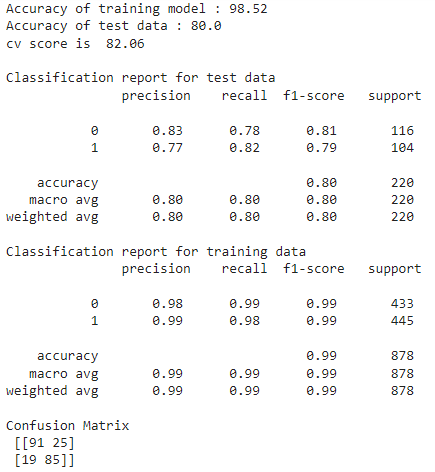
print(clf.best\_params\_)

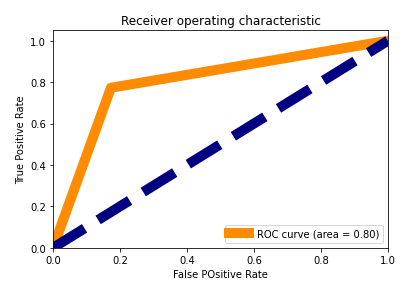


from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n\_estimators=90, class\_weight= 'balanced',criterion='entropy', max\_features=None, oob\_score= 'True')

model\_selection(rf,x\_train,y\_train,x\_test,y\_test)





1. **Ada Boost Classifier**

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import AdaBoostClassifier

parameter={'algorithm':['SAMME', 'SAMME.R'],

          'learning\_rate' : [0.0,1.0],

           'random\_state':range(0,100,20),

           'n\_estimators':range(0,100,20)}

rf2=AdaBoostClassifier()

clf=GridSearchCV(rf2,parameter)

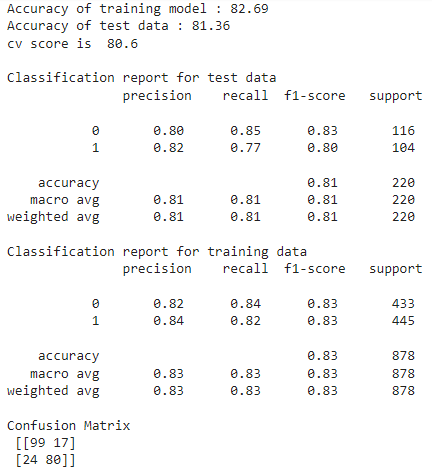
clf.fit(x\_train,y\_train)

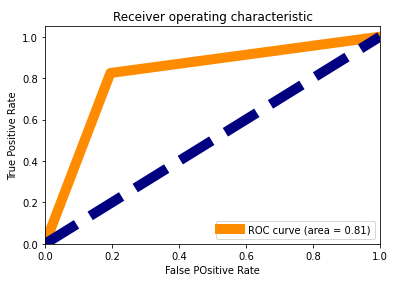
print(clf.best\_params\_)



rf2=AdaBoostClassifier(algorithm='SAMME.R', learning\_rate= 1.0, n\_estimators= 20,random\_state= 0)

model\_selection(rf2,x\_train,y\_train,x\_test,y\_test)



****

1. **Gradient Boosting Classifier**

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import GradientBoostingClassifier

parameter={'loss':['log\_loss', 'deviance', 'exponential'],

          'learning\_rate' : [0.0,1.0],

           'criterion':['friedman\_mse', 'squared\_error', 'mse'],

           'max\_features':['auto', 'sqrt', 'log2'],

           'n\_estimators':range(0,100,20)}

rf3=GradientBoostingClassifier()

clf=GridSearchCV(rf3,parameter)

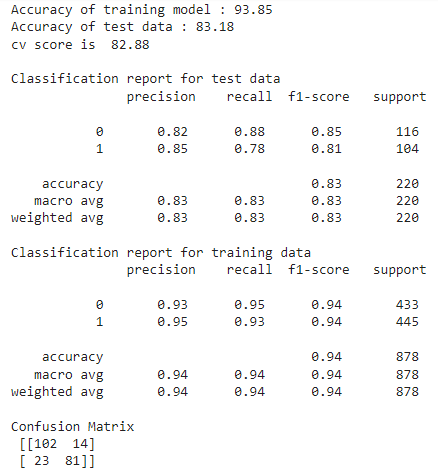
clf.fit(x\_train,y\_train)

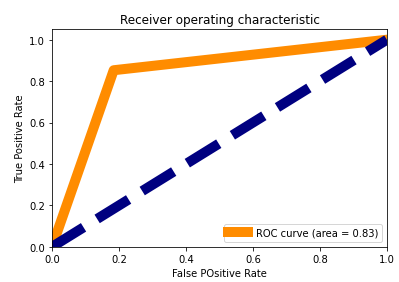
print(clf.best\_params\_)



rf3=GradientBoostingClassifier(criterion='mse',learning\_rate=1.0, loss='exponential', max\_features= 'log2', n\_estimators= 60)

model\_selection(rf3,x\_train,y\_train,x\_test,y\_test)

****

****

1. **Bagging Classifier**

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import BaggingClassifier

parameter={

           'oob\_score': [True,False],

           'n\_jobs':range(0,100,20),

           'random\_state':range(0,100,20),

           'n\_estimators':range(0,100,20)}

rf4=BaggingClassifier()

clf=GridSearchCV(rf4,parameter)

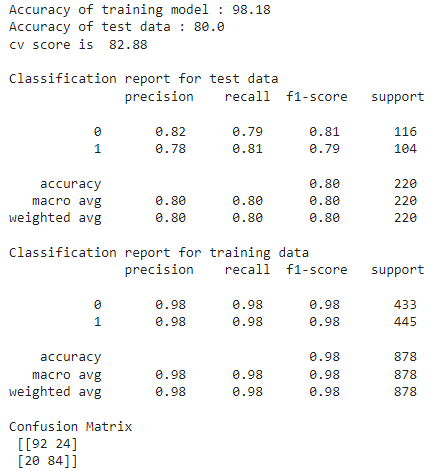
clf.fit(x\_train,y\_train)

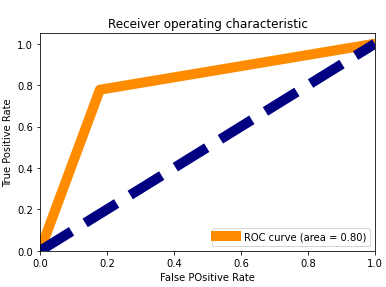
print(clf.best\_params\_)



rf4=BaggingClassifier(n\_jobs=20, oob\_score= True, random\_state=20, n\_estimators= 40)

model\_selection(rf4,x\_train,y\_train,x\_test,y\_test)





1. **Stacking Classifier**

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import StackingClassifier

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

estimators = [ ('knc', KNeighborsClassifier()), ('svr',SVC()) ]

parameter={

            'stack\_method':['auto', 'predict\_proba', 'decision\_function', 'predict'],

           'n\_jobs':range(0,100,20)

         }

rf5=StackingClassifier(estimators)

clf=GridSearchCV(rf5,parameter)

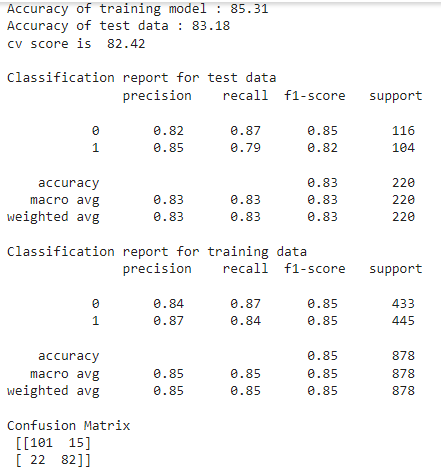
clf.fit(x\_train,y\_train)

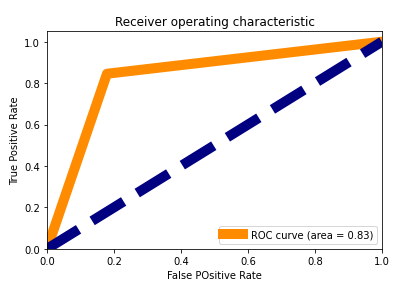
print(clf.best\_params\_)

****

rf5=StackingClassifier(estimators,n\_jobs=20, stack\_method= 'auto')

model\_selection(rf5,x\_train,y\_train,x\_test,y\_test)

****

****

**Model Saving**

Model that has best AUC score is identified as model with best accuracy and hence we save it using pickle library.

import pickle

filename='rfz.pk1'

pickle.dump(s,open(filename,'wb'))

**Model Loading**

We load the saved model for testing

loaded\_model=pickle.load(open('rfz.pk1','rb'))

result=loaded\_model.score(x\_test,y\_test)

print("Model Accuracy: ",round(result\*100,2))

****

**Testing Loaded Model**

We create a function that take user input and then predicts the result.

def test\_inp():

inp = []

i=0

print("Enter either 0 or 1 as single input\n")

while i<6:

inn=int(input(f"Enter input {i+1}: "))

inp.append(inn)

i=i+1

input\_data\_as\_numpy\_array = np.asarray(inp)

input\_data\_reshaped = input\_data\_as\_numpy\_array.reshape(1,-1)

prediction = loaded\_model.predict(input\_data\_reshaped)

if prediction[0]==0:

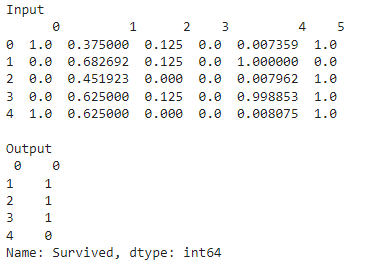
print("\nStatus: Dead")

if prediction[0]==1:

print("\nStatus: Alive")

We can now use the x feature and y feature to cross check if the model works properly on the data.

print("Input\n",x.head(),"\n\nOutput\n",y.head())

****

In above table we have seen that

- Input 0th row gives output 0 i.e dead

- Input 1st row gives output 1 i.e alive

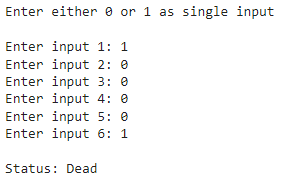
- Input 2nd row gives output 1 i.e alive

- Input 3rd row gives output 1 i.e alive

- Input 4th row gives output 0 i.e dead

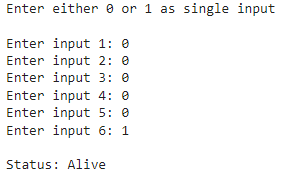
This input data can be used in loaded model to test if model gives  correct output

test\_inp()

****

Try again with another input

test\_inp()

****

**6) Conclusion**

Although the model performed well on the given dataset, it has potential to improve even further if we use bigger dataset to train the model.