Ai Assignment-1 Part-B

Question: -The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. In this assignment, you should build a predictive model that answers the question: “what sorts of people were more likely to survive?” using passenger data.

1. **DATA VISUALIZATION: -** In the first cell we are importing the required Modules for data manipulation like numpy,pandas and plot visualisation we used plotly,seaborn,matplotlib.We loaded the required datasheet for our assignment.

In [1]:
```
import numpy as np
import matplotlib as matlab
import matplotlib.pyplot as plt
import pandas as pd
import urllib.request as url
import seaborn as sea
import plotly.express as px
titanic_pdf = pd.read_csv('titanic.csv')
titanic_pdf
```

Out[1]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In the second cell we used info() method to get all the required basic information in datasheet like features(column), non-null count(which is a count of the specified info about a features it excludes missing information like Nan and return count),and finally datatype of each feature(datatype).

In [8]:
```
titanic_pdf.info()
```

```
<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
```

3) In the third cell we used the head() method to display the required number of rows in the given datasheet.

In [3]:
```
titanic_pdf.head(10)
```

Out[3]:

	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
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

4) In the fourth cell we used the iloc() method to display required features(columns) in the datasheet.It only displays it wont delete it like drop() method.

In [17]:
```
titanic_pdf.iloc[0:5,[0,3,5,4]]
```

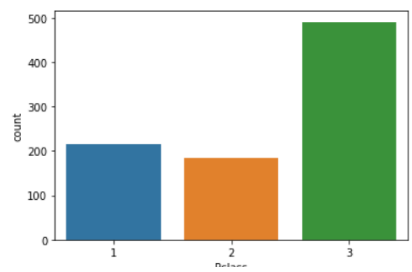
Out[17]:

	PassengerId	Name	Age	Sex
0	1	Braund, Mr. Owen Harris	22.0	male
1	2	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	female
2	3	Heikkinen, Miss. Laina	26.0	female
3	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	female
4	5	Allen, Mr. William Henry	35.0	male

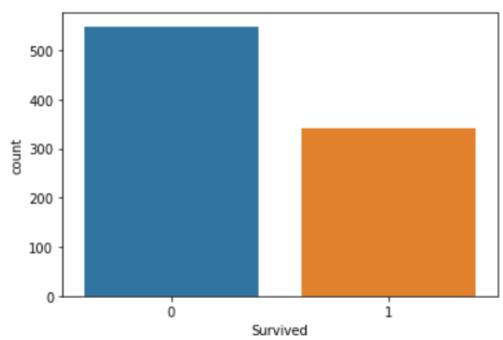
2. <u>**Data Analysis:-**</u> In the fifth,sixth,seventh cells we had created plot environment,plotted pclass vs count and survived vs count.

In [76]:
```
sea.set_style('whitegrid')
matlab.rcParams['font.size'] = 14
matlab.rcParams['figure.figsize'] = (10, 6)
matlab.rcParams['figure.facecolor'] = '#00000000'
```

In [4]:
```
ax = sea.countplot(x="Pclass", data=titanic_pdf)
```



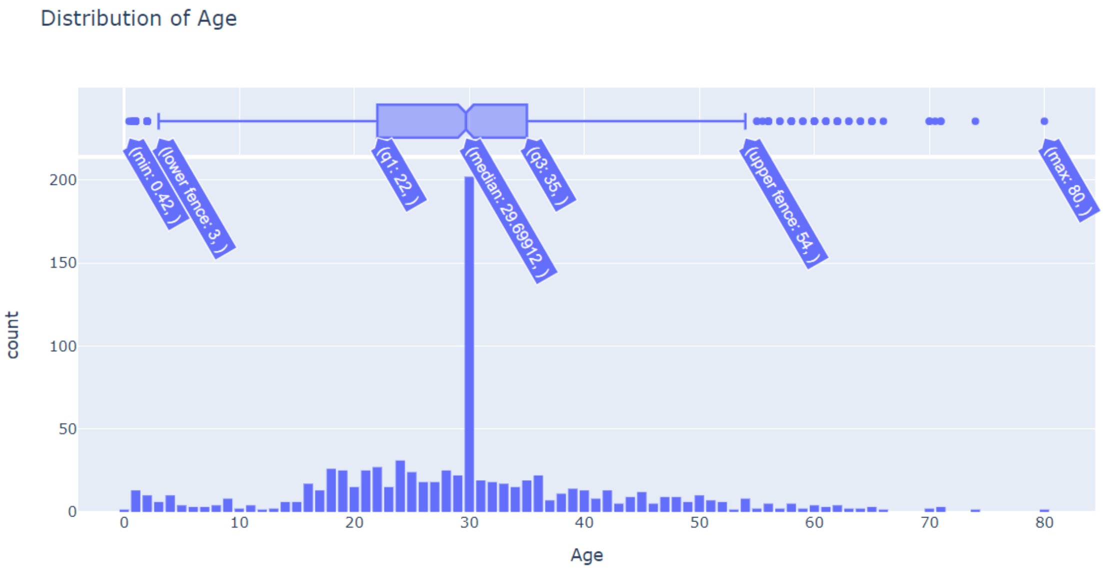
In [5]:
```
ax = sea.countplot(x="Survived", data=titanic_pdf)
```



- [ ] In the eighth cell we plotted but unfortunately we replaced age with Nan with mean so now the mean will has high count.we used ploty its an interactive plot check in the ipynb file for visual experience.

```
In [78]: fig = px.histogram(titanic_pdf,
                            x='Age',
                            marginal='box',
                            nbins=80,
                            title='Distribution of Age')
         fig.update_layout(bargap=0.2)
         fig.show()
```

- [ ]



<u>III. DATA WRANGLING & FEATURE SELECTION: - </u> In the next cell we dropped the unnecessary columns.We used drop() method.

```
In [9]: titanic_pdf.drop(['PassengerId', 'Name', 'Ticket', 'Cabin', 'Embarked'],axis='columns',inplace=True)
        titanic_pdf
```

Out[9]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	male	22.0	1	0	7.2500
1	1	1	female	38.0	1	0	71.2833
2	1	3	female	26.0	0	0	7.9250
3	1	1	female	35.0	1	0	53.1000
4	0	3	male	35.0	0	0	8.0500
...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000
887	1	1	female	19.0	0	0	30.0000
888	0	3	female	NaN	1	2	23.4500
889	1	1	male	26.0	0	0	30.0000
890	0	3	male	32.0	0	0	7.7500

891 rows × 7 columns

In the next cell we calculated the mean of the age and we rounded it and replace all the Nan values in the age with mean. Unfortunately the above age plot will change now because almost 177 will be replaced with mean.

```
In [13]: a = titanic_pdf['Age'].mean()
         a
Out[13]: 29.69911764705882

In [24]: round(a,2)
Out[24]: 29.7

In [25]: titanic_pdf['Age'].fillna(a,inplace=True)

In [26]: titanic_pdf['Age']
Out[26]: 0      22.000000
         1      38.000000
         2      26.000000
         3      35.000000
         4      35.000000
                  ...
         886    27.000000
         887    19.000000
         888    29.699118
         889    26.000000
         890    32.000000
         Name: Age, Length: 891, dtype: float64
```

See now the count in the age is 891 instead 714 because we replaced lost data with mean.We added mean because it doesn't affect the data.

```
In [79]: titanic_pdf['Age'].describe()
Out[79]: count    891.000000
         mean      29.699118
         std       13.002015
         min        0.420000
         25%       22.000000
         50%       29.699118
         75%       35.000000
         max       80.000000
         Name: Age, dtype: float64
```

Now we will classify some features like gender and Pclass in binary.For this we dummies method.

```
In [4]: Gender = pd.get_dummies(titanic_pdf['Sex'])
        Gender
Out[4]:
```

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1
...	...	...
886	0	1
887	1	0
888	1	0
889	0	1
890	0	1

891 rows × 2 columns

Now we remove a column as we need only one column to predict the gender. We used the iloc()method instead of drop(). We stored the iloc() method final value in an object so we can use this object to concat.

```
In [5]: A =Gender.iloc[:,1]
        A
```

```
Out[5]: 0      1
        1      0
        2      0
        3      0
        4      1
              ..
        886    1
        887    0
        888    0
        889    1
        890    1
        Name: male, Length: 891, dtype: uint8
```

Similarly for PClass.

```
In [6]: PasenCls=pd.get_dummies(titanic_pdf['Pclass'])
        PasenCls
```

Out[6]:

	1	2	3
0	0	0	1
1	1	0	0
2	0	0	1
3	1	0	0
4	0	0	1
...	...	...	...
886	0	1	0
887	1	0	0
888	0	0	1
889	1	0	0
890	0	0	1

891 rows × 3 columns

```
In [7]: B = PasenCls.iloc[:,[0,1]]
        B
```

Out[7]:

	1	2
0	0	0
1	1	0
2	0	0
3	1	0
4	0	0
...	...	...
886	0	1
887	1	0
888	0	0
889	1	0
890	0	0

891 rows × 2 columns

Now we concatinate these to the modified datasheet,Using concat() method.

```
In [19]: Titanic_pdf = pd.concat([titanic_pdf,A,B],axis=1)
         Titanic_pdf
```

Out[19]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	male	1	2
0	0	3	male	22.000000	1	0	7.2500	1	0	0
1	1	1	female	38.000000	1	0	71.2833	0	1	0
2	1	3	female	26.000000	0	0	7.9250	0	0	0
3	1	1	female	35.000000	1	0	53.1000	0	1	0
4	0	3	male	35.000000	0	0	8.0500	1	0	0
...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.000000	0	0	13.0000	1	0	1
887	1	1	female	19.000000	0	0	30.0000	0	1	0
888	0	3	female	29.699118	1	2	23.4500	0	0	0
889	1	1	male	26.000000	0	0	30.0000	1	1	0
890	0	3	male	32.000000	0	0	7.7500	1	0	0

891 rows × 10 columns

Now drop the features Pclass,Sex and we change column male,1,2 to Sex , pclass_1,pclass_2.For this we used rename() method.

```
Titanic_pdf.drop(['Pclass','Sex'],axis=1,inplace=True)
```

```
In [33]: Titanic_pdf.rename(columns={'male':'Sex',1:'Pclass_1',2:'Pclass_2'},inplace=True)
         Titanic_pdf
```

Out[33]:

	Survived	Age	SibSp	Parch	Fare	Sex	Pclass_1	Pclass_2
0	0	22.000000	1	0	7.2500	1	0	0
1	1	38.000000	1	0	71.2833	0	1	0
2	1	26.000000	0	0	7.9250	0	0	0
3	1	35.000000	1	0	53.1000	0	1	0
4	0	35.000000	0	0	8.0500	1	0	0
...	...	...	...	...	...	...	...	...
886	0	27.000000	0	0	13.0000	1	0	1
887	1	19.000000	0	0	30.0000	0	1	0
888	0	29.699118	1	2	23.4500	0	0	0
889	1	26.000000	0	0	30.0000	1	1	0
890	0	32.000000	0	0	7.7500	1	0	0

891 rows × 8 columns

Before going to test the data we should look at once all the data is perfect or faulty.

```
In [81]: Titanic_pdf.describe()
```

Out[81]:

	Survived	Age	SibSp	Parch	Fare	Sex	Pclass_1	Pclass_2
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	29.699118	0.523008	0.381594	32.204208	0.647587	0.242424	0.206510
std	0.486592	13.002015	1.102743	0.806057	49.693429	0.477990	0.428790	0.405028
min	0.000000	0.420000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	22.000000	0.000000	0.000000	7.910400	0.000000	0.000000	0.000000
50%	0.000000	29.699118	0.000000	0.000000	14.454200	1.000000	0.000000	0.000000
75%	1.000000	35.000000	1.000000	0.000000	31.000000	1.000000	0.000000	0.000000
max	1.000000	80.000000	8.000000	6.000000	512.329200	1.000000	1.000000	1.000000

It seems there is no problem with the processed datasheet.

IV. TRAINING & TESTING: -

Logistic Regression:- we use logistic regression algorithm to develop a model.

Importing required modules , assigning test and train data(30 to 70) and finding the equation.

```
In [38]: from sklearn.model_selection import train_test_split
         y = Titanic_pdf['Survived']
         x = Titanic_pdf.drop(['Survived'], axis = 1)
         x_train, x_test, y_train, y_test = train_test_split(
                 x, y, test_size = 0.3, random_state = 0)
```

In [40]:
```python
print(x_train,x_test)
```

```
           Age  SibSp  Parch      Fare  Sex  Pclass_1  Pclass_2
857  51.000000      0      0   26.5500    1         1         0
52   49.000000      1      0   76.7292    0         1         0
386   1.000000      5      2   46.9000    1         0         0
124  54.000000      0      1   77.2875    1         1         0
578  29.699118      1      0   14.4583    0         0         0
..         ...    ...    ...       ...  ...       ...       ...
835  39.000000      1      1   83.1583    0         1         0
192  19.000000      1      0    7.8542    0         0         0
629  29.699118      0      0    7.7333    1         0         0
559  36.000000      1      0   17.4000    0         0         0
684  60.000000      1      1   39.0000    1         0         1

[623 rows x 7 columns]            Age  SibSp  Parch      Fare  Sex  Pclass_1  Pclass_2
495  29.699118      0      0   14.4583    1         0         0
648  29.699118      0      0    7.5500    1         0         0
278   7.000000      4      1   29.1250    1         0         0
31   29.699118      1      0  146.5208    0         1         0
255  29.000000      0      2   15.2458    0         0         0
..         ...    ...    ...       ...  ...       ...       ...
263  40.000000      0      0    0.0000    1         1         0
718  29.699118      0      0   15.5000    1         0         0
620  27.000000      1      0   14.4542    1         0         0
786  18.000000      0      0    7.4958    0         0         0
64   29.699118      0      0   27.7208    1         1         0

[268 rows x 7 columns]
```

In [41]:
```python
print(y_train,y_test)
```

```
857    1
52     1
386    0
124    0
578    0
      ..
835    1
192    1
629    0
559    1
684    0
Name: Survived, Length: 623, dtype: int64 495    0
648    0
278    0
31     1
255    1
      ..
263    0
718    0
620    0
786    1
64     0
Name: Survived, Length: 268, dtype: int64
```

In [45]:
```python
from sklearn.preprocessing import StandardScaler as SS
sc = SS()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
print(x_train)
print(x_test)
```

```
[[ 1.62393675 -0.457246   -0.47299765 ...  0.72592065  1.73019934
  -0.51849697]
 [ 1.47020331  0.4033711  -0.47299765 ... -1.37756104  1.73019934
  -0.51849697]
 [-2.21939923  3.8458395   1.93253327 ...  0.72592065 -0.57796809
  -0.51849697]
 ...
 [-0.0133922  -0.457246   -0.47299765 ...  0.72592065 -0.57796809
  -0.51849697]
 [ 0.47093596  0.4033711  -0.47299765 ... -1.37756104 -0.57796809
  -0.51849697]
 [ 2.31573723  0.4033711   0.72976781 ...  0.72592065 -0.57796809
   1.92865159]]
[[-0.0133922  -0.457246   -0.47299765 ...  0.72592065 -0.57796809
  -0.51849697]
 [-0.0133922  -0.457246   -0.47299765 ...  0.72592065 -0.57796809
  -0.51849697]
 [-1.75819891  2.9852224   0.72976781 ...  0.72592065 -0.57796809
  -0.51849697]
 ...
 [-0.22086452  0.4033711  -0.47299765 ...  0.72592065 -0.57796809
  -0.51849697]
 [-0.91266499 -0.457246   -0.47299765 ... -1.37756104 -0.57796809
  -0.51849697]
 [-0.0133922  -0.457246   -0.47299765 ...  0.72592065  1.73019934
  -0.51849697]]
```

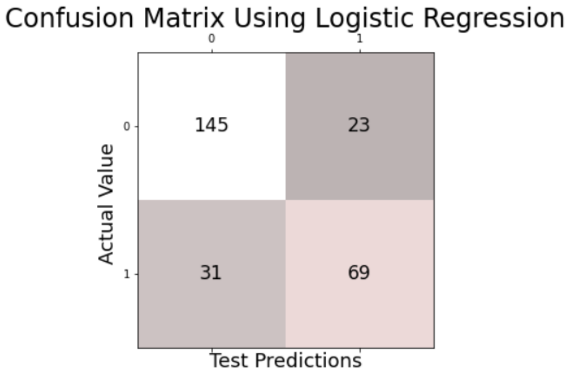
Here we should process the dataframe to list of lists i.e each innerlist represents a list of features for a particular label here label is survived.

We are going to fit() method to calculate the required equation and test with predict() method,and obtain confusion matrix using confusion_matrix() method.

```
  -0.51849697]]

In [46]: from sklearn.linear_model import LogisticRegression as LR
         classifier = LR(random_state = 0)
         classifier.fit(x_train, y_train)

Out[46]: LogisticRegression(random_state=0)

In [48]: y_pred = classifier.predict(x_test)
         y_pred

Out[48]: array([0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1,
                0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
                1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
                1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
                0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0,
                1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
                1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1,
                0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
                0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1,
                0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 0], dtype=int64)

In [54]: from sklearn.metrics import confusion_matrix as CM
         cm = CM(y_test, y_pred)
         print ("Confusion Matrix : \n", cm)

         Confusion Matrix :
          [[145  23]
          [ 31  69]]
```

We plotted this confusion matrix in 2-D plot using matshow()metod.

```
In [63]: fig, ax = plt.subplots(figsize=(5, 5))
         ax.matshow(cm, cmap=plt.cm.pink, alpha=0.3)
         for i in range(cm.shape[0]):
             for j in range(cm.shape[1]):
                 ax.text(x=j, y=i,s=cm[i, j], va='center', ha='center', size='xx-large')

         plt.xlabel('Test Predictions', fontsize=18)
         plt.ylabel('Actual Value ', fontsize=18)
         plt.title('Confusion Matrix Using Logistic Regression', fontsize=24)
         plt.show()
```



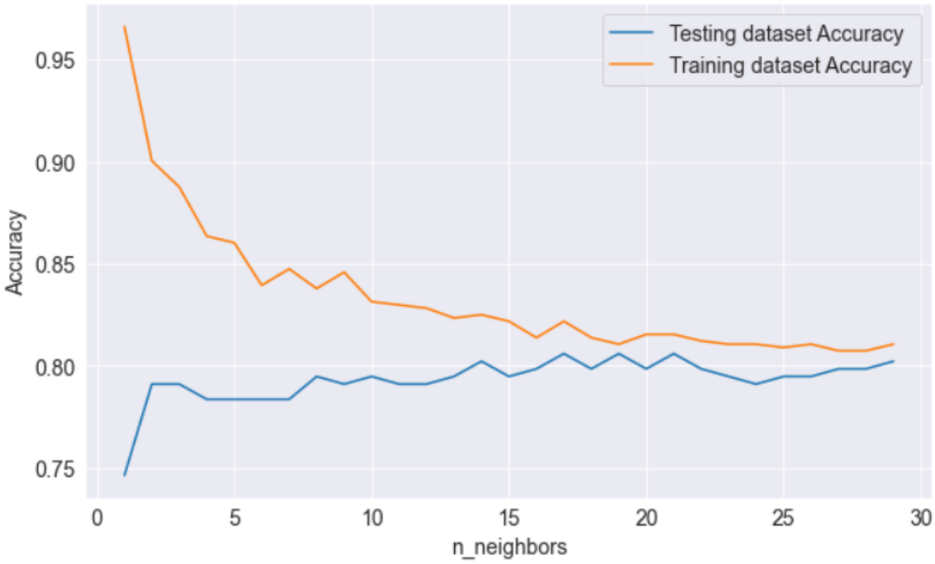
We calculated different parameters obtained from confusion matrix.

```
In [62]: from sklearn.metrics import accuracy_score
         print ("Accuracy : ", accuracy_score(y_test, y_pred))
         from sklearn.metrics import precision_score
         print ("Precession : ", precision_score(y_test, y_pred))
         from sklearn.metrics import f1_score
         print ("F1 score : ", f1_score(y_test, y_pred))
         from sklearn.metrics import recall_score
         print ("Recall score : ", recall_score(y_test, y_pred))

         Accuracy :  0.7985074626865671
         Precession :  0.75
         F1 score :  0.71875
         Recall score :  0.69
```

KNN Algorithm:-

First we need to find the value of K

```
In [96]: neighbors = np.arange(1, 30)
         train_accuracy = np.empty(len(neighbors))
         test_accuracy = np.empty(len(neighbors))

         # Loop over K values
         for i, k in enumerate(neighbors):
             knn = KNeighborsClassifier(n_neighbors=k)
             knn.fit(x_train, y_train)# Compute training and test data accuracy
             train_accuracy[i] = knn.score(x_train, y_train)
             test_accuracy[i] = knn.score(x_test, y_test)

         # Generate plot
         plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
         plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

         plt.legend()
         plt.xlabel('n_neighbors')
         plt.ylabel('Accuracy')
         plt.show()
```



From this plot we can take k as between 17 to 19.

I had taken k = 17;

```
In [101]: from sklearn.neighbors import KNeighborsClassifier
          from sklearn.model_selection import train_test_split
          knn_clf = KNeighborsClassifier(n_neighbors = 17).fit(x_train, y_train.values.ravel())
          y_knn = knn_clf.predict(x_test)
          print(y_knn)

          [0 0 0 1 0 0 1 1 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0
           0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0
           0 1 1 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 1 0
           1 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 1 1 0 1
           1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0
           0 1 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1
           1 0 0 0 0 1 0 0 0 1 1 1 0 0 1 1 1 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 0 0
           0 0 0 0 0 0 0 1 0]
```

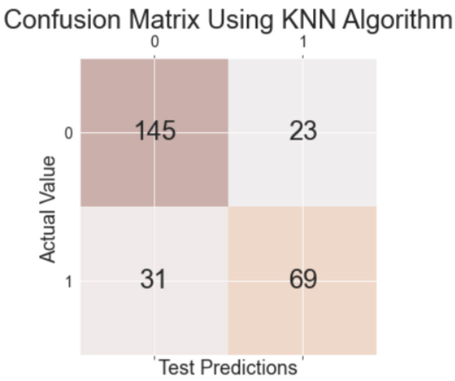
Obtaining confusion matrix,

```
In [102]: from sklearn.metrics import confusion_matrix as CM
          knn_cm = CM(y_test, y_knn)
          print(knn_cm)

          [[149  19]
           [ 33  67]]
```

Plotting confusion matrix,

In [103]:

```
fig, ax = plt.subplots(figsize=(5, 5))
ax.matshow(knn_cm, cmap=plt.cm.Oranges, alpha=0.3)
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(x=j, y=i,s=cm[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Test Predictions', fontsize=18)
plt.ylabel('Actual Value ', fontsize=18)
plt.title('Confusion Matrix Using KNN Algorithm', fontsize=24)
plt.show()
```



In [104]:

```
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_knn))
from sklearn.metrics import precision_score
print ("Precession : ", precision_score(y_test, y_knn))
from sklearn.metrics import f1_score
print ("F1 score : ", f1_score(y_test, y_knn))
from sklearn.metrics import recall_score
print ("Recall score : ", recall_score(y_test, y_knn))
```

```
Accuracy :  0.8059701492537313
Precession :  0.7790697674418605
F1 score :  0.7204301075268817
Recall score :  0.67
```