The RMS Titanic sank in the early morning hours of 15 April 1912 in the North Atlantic Ocean four days into her maiden voyage from Southampton to New York City. The largest ocean liner in service at the time, when she struck an iceberg at around 23:40 (ship's time) on Sunday, 14 April 1912. Her sinking two hours and forty minutes later at 02:20(ship's time; 05:18 GMT) on Monday,15 April,resulted in the deaths of people.making it one of the deadliest peacetime maritime disasters in history.

Gautamkumar Mishra - Titanic ML

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Numpy-Numerical Python that Deals with numbers.

Pandas-Used to analyse data.It has functions for analyzing,cleaning,exploring and manipulating Data.

Matplotlib-comprehensive library for creating static, animated, and interactive visualizations.

Seaborn-Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Warnings are provided to warn the developer of situations that aren't necessarily exceptions and ignore them.

In [2]:
```
df=pd.read_csv('titanic.csv')
```

By read_csv() function we are reading the dataset present in 'Titanic.csv' file.

In [3]:
```
df
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	
...	...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	

891 rows × 12 columns

This Titanic.csv dataset contains the information of people travelling from the ship RMS-Titanic.with 891 rows and 12 columns.

In [4]:
```
df.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
```

This dataset consists 891 rows and 12 columns out of which there are 7 numerical columns and 5 object columns.

In [5]:
```
df.isna().sum()/len(df)*100
```

Out[5]:
```
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
```

With the help of isna().sum() function we get the null count of all columns. And by calculating the null value's percentage we get The column 'Cabin' consist null value upto 77% and 'Age' column consists null values upto 19% and 'Embarked' column consist upto 0.22%.

In [6]:
```
df.drop('Cabin',axis=1,inplace=True)
```

In [7]:
```
df
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	E
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	
...	...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	

891 rows × 11 columns

As the missing data is more than 40% we can drop that column so we drop the column 'Cabin' present in df.

In [8]:
```
df['Age'].unique()
```

Out[8]:
```
array([22.  , 38.  , 26.  , 35.  ,   nan, 54.  ,  2.  , 27.  , 14.  ,
        4.  , 58.  , 20.  , 39.  , 55.  , 31.  , 34.  , 15.  , 28.  ,
        8.  , 19.  , 40.  , 66.  , 42.  , 21.  , 18.  ,  3.  ,  7.  ,
       49.  , 29.  , 65.  , 28.5 ,  5.  , 11.  , 45.  , 17.  , 32.  ,
       16.  , 25.  ,  0.83, 30.  , 33.  , 23.  , 24.  , 46.  , 59.  ,
       71.  , 37.  , 47.  , 14.5 , 70.5 , 32.5 , 12.  ,  9.  , 36.5 ,
       51.  , 55.5 , 40.5 , 44.  ,  1.  , 61.  , 56.  , 50.  , 36.  ,
       45.5 , 20.5 , 62.  , 41.  , 52.  , 63.  , 23.5 ,  0.92, 43.  ,
       60.  , 10.  , 64.  , 13.  , 48.  ,  0.75, 53.  , 57.  , 80.  ,
       70.  , 24.5 ,  6.  ,  0.67, 30.5 ,  0.42, 34.5 , 74.  ])
```

The column 'Age' consist null values so we can fill that null value by mean age of passengers travelling on Titanic Ship.

In [9]:
```
df['Age'].mean()
```

Out[9]:
```
29.69911764705882
```

In [10]:
```
df['Age']=df['Age'].fillna(df['Age'].mean())
```

We,have filled the missing value of Age column by mean value.

In [11]:
```
df.isna().sum()/len(df)*100
```

Out[11]:
```
PassengerId    0.000000
Survived       0.000000
Pclass         0.000000
Name           0.000000
Sex            0.000000
Age            0.000000
SibSp          0.000000
Parch          0.000000
Ticket         0.000000
Fare           0.000000
Embarked       0.224467
dtype: float64
```

The 'Embarked' column consist of null value of 0.2% so we can drop that row containing null value

In [12]:
```
df.dropna(inplace=True)
```

In [13]:
```
df.isna().sum()/len(df)*100
```

Out[13]:
```
PassengerId    0.0
Survived       0.0
Pclass         0.0
Name           0.0
Sex            0.0
Age            0.0
SibSp          0.0
Parch          0.0
Ticket         0.0
Fare           0.0
Embarked       0.0
dtype: float64
```

Now,our data does not contain any null values.

In [14]:
```
df.drop('PassengerId',axis=1,inplace=True)
```

The Column 'PassengerId' consist of Index number so we can drop that column as we our own Index

In [15]:
```
df.reset_index(inplace=True,drop=True)
```

In [16]:
```
df
```

Out[16]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	S
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	C
2	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	S
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.1000	S
4	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.0500	S
...	...	...	...	...	...	...	...	...	...	...
884	0	2	Montvila, Rev. Juozas	male	27.000000	0	0	211536	13.0000	S
885	1	1	Graham, Miss. Margaret Edith	female	19.000000	0	0	112053	30.0000	S
886	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.699118	1	2	W./C. 6607	23.4500	S
887	1	1	Behr, Mr. Karl Howell	male	26.000000	0	0	111369	30.0000	C
888	0	3	Dooley, Mr. Patrick	male	32.000000	0	0	370376	7.7500	Q

889 rows × 10 columns

In [17]:
```
df.shape
```

Out[17]:
```
(889, 10)
```

The dataframe now consist of 889 rows and 10 columns.

In [18]:
```
for i in df:
    print(df[i].unique())
```

```
[0 1]
[3 1 2]
['Braund, Mr. Owen Harris'
 'Cumings, Mrs. John Bradley (Florence Briggs Thayer)'
 'Heikkinen, Miss. Laina' 'Futrelle, Mrs. Jacques Heath (Lily May Peel)'
 'Allen, Mr. William Henry' 'Moran, Mr. James' 'McCarthy, Mr. Timothy J'
 'Palsson, Master. Gosta Leonard'
 'Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)'
 'Nasser, Mrs. Nicholas (Adele Achem)' 'Sandstrom, Miss. Marguerite Rut'
 'Bonnell, Miss. Elizabeth' 'Saundercock, Mr. William Henry'
 'Andersson, Mr. Anders Johan' 'Vestrom, Miss. Hulda Amanda Adolfina'
 'Hewlett, Mrs. (Mary D Kingcome) ' 'Rice, Master. Eugene'
 'Williams, Mr. Charles Eugene'
 'Vander Planke, Mrs. Julius (Emelia Maria Vandemoortele)'
 'Masselmani, Mrs. Fatima' 'Fynney, Mr. Joseph J' 'Beesley, Mr. Lawrence'
 'McGowan, Miss. Anna "Annie"' 'Sloper, Mr. William Thompson'
 'Palsson, Miss. Torborg Danira'
 'Asplund, Mrs. Carl Oscar (Selma Augusta Emilia Johansson)'
 'Emir, Mr. Farred Chehab' 'Fortune, Mr. Charles Alexander'
```

Here, we used loop to iterate over each columns unique values instead of iterating per column.

In [19]:
```
num_col=df.select_dtypes(include=['int64','float']).columns
num_col
```

Out[19]:
```
Index(['Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare'], dtype='object')
```

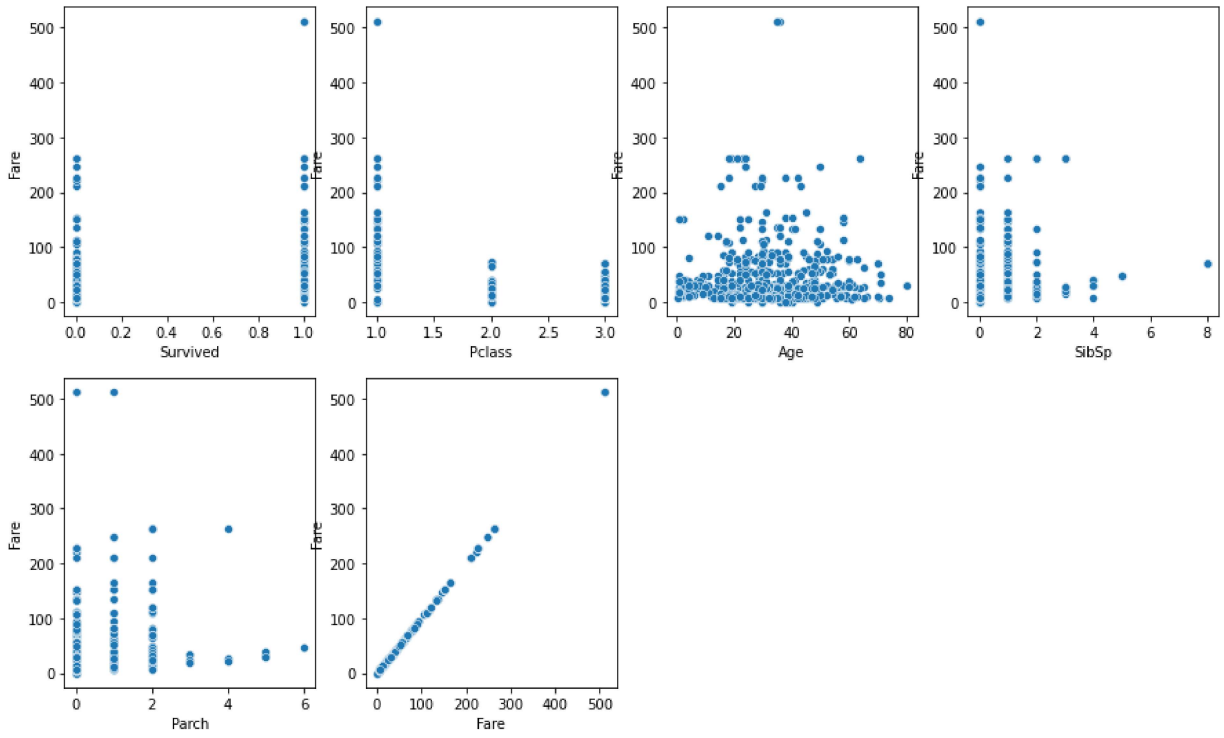
There are 6 Numerical columns in dataset and we store it in a variable named 'num_col'.

In [20]:
```
cat_col=df.select_dtypes(include=['O']).columns
cat_col
```

Out[20]:
```
Index(['Name', 'Sex', 'Ticket', 'Embarked'], dtype='object')
```

There are 4 Numerical columns in dataset and we store it in a variable name 'cat_col'

In [21]:
```
plt.figure(figsize=(15,14))
count=1
for i in num_col:
    plt.subplot(3,4,count)
    sns.scatterplot(x=i,y='Fare',data=df)
    count+=1
plt.show()
```



Iterating over each numerical column to plot scatterplot between all Numerical columns. And, There are no co-relation between any numerical columns.

How many people boarded on Titanic where Male?

In [22]:
```
df[df['Sex']=='male']['Sex'].count()
```

Out[22]: 577

There were 577 Male boarded on Titanic.

What was the average age of people onboard?

In [23]:
```
df['Age'].mean()
```

Out[23]:
```
29.653446370674192
```

The average age of people onboard was 29 years.

How many people where there from each class?

In [24]:
```
df['Pclass'].value_counts()
```

Out[24]:
```
3    491
1    214
2    184
Name: Pclass, dtype: int64
```

From class 1 there were 214 people, From class 2 there were 184 people and from class 3 there were 491 people onboard.

What was the highest paid price for the ticket of titanic?

In [25]:
```
df['Fare'].max()
```

Out[25]:
```
512.3292
```

The highest paid price for ticket was approx $512

What was the lowest paid price for the ticket of titanic?

In [26]:
```
df['Fare'].min()
```

Out[26]:
```
0.0
```

The lowest paid price for ticket was $0.

Find the people with fare $0.

In [27]:
```
df[df['Fare']==0]
```

Out[27]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
178	0	3	Leonard, Mr. Lionel	male	36.000000	0	0	LINE	0.0	S
262	0	1	Harrison, Mr. William	male	40.000000	0	0	112059	0.0	S
270	1	3	Tornquist, Mr. William Henry	male	25.000000	0	0	LINE	0.0	S
276	0	2	Parkes, Mr. Francis "Frank"	male	29.699118	0	0	239853	0.0	S
301	0	3	Johnson, Mr. William Cahoone Jr	male	19.000000	0	0	LINE	0.0	S
412	0	2	Cunningham, Mr. Alfred Fleming	male	29.699118	0	0	239853	0.0	S
465	0	2	Campbell, Mr. William	male	29.699118	0	0	239853	0.0	S
480	0	2	Frost, Mr. Anthony Wood "Archie"	male	29.699118	0	0	239854	0.0	S
596	0	3	Johnson, Mr. Alfred	male	49.000000	0	0	LINE	0.0	S
632	0	1	Parr, Mr. William Henry Marsh	male	29.699118	0	0	112052	0.0	S
673	0	2	Watson, Mr. Ennis Hastings	male	29.699118	0	0	239856	0.0	S
731	0	2	Knight, Mr. Robert J	male	29.699118	0	0	239855	0.0	S
805	0	1	Andrews, Mr. Thomas Jr	male	39.000000	0	0	112050	0.0	S
814	0	1	Fry, Mr. Richard	male	29.699118	0	0	112058	0.0	S
821	0	1	Reuchlin, Jonkheer. John George	male	38.000000	0	0	19972	0.0	S

There were 15 people from all classes with fare 0$.

What was the highest price of class 3 passengers?

In [28]:
```
df.groupby('Pclass').get_group(3)['Fare'].max()
```

Out[28]:
```
69.55
```

The highest price for class 3 passengers was $69.55

What was the lowest price of class 3 passengers?

In [29]:
```
df.groupby('Pclass').get_group(3)['Fare'].min()
```

Out[29]:
```
0.0
```

The lowest price for class 3 passengers were 0$

What was the highest price for class 1 passengers?

In [30]:
```
df.groupby('Pclass').get_group(1)['Fare'].max()
```

Out[30]:
```
512.3292
```

The highest price for class 1 passengers was $512.3 approx.

What was the lowest price for class 1 passengers?

In [31]:
```
df.groupby('Pclass').get_group(1)['Fare'].min()
```

Out[31]:
```
0.0
```

The lowest price for class 1 passengers was $0.

What was the highest price for class 2 passengers?

In [32]:
```
df.groupby('Pclass').get_group(2)['Fare'].max()
```

Out[32]:
```
73.5
```

The highest price for class 2 passengers was $73.5 approx.

How many male survived from sinking of titanic?

In [33]:
```
df[(df['Sex']=='male')&(df['Survived']==1)].count()[0:1]
```

Out[33]:
```
Survived    109
dtype: int64
```

There were 109 male survivals from sinking of titanic.

How many Female survived from sinking of titanic?

In [34]:
```
df[(df['Sex']=='female')&(df['Survived']==1)].count()[0:1]
```

Out[34]:
```
Survived    231
dtype: int64
```

There were 231 female survival from titanic's sinking.

How many people survived?

In [35]:
```
df[df['Survived']==1].count()[0:1]
```

Out[35]:
```
Survived    340
dtype: int64
```

304 people survived from titanic's accident.

How many children were on-board on titanic?

In [36]:
```
df[df['Age']<=18].count()
```

Out[36]:
```
Survived    139
Pclass      139
Name        139
Sex         139
Age         139
SibSp       139
Parch       139
Ticket      139
Fare        139
Embarked    139
dtype: int64
```

There were 139 children board on titanic.

How many of children were less than 10 years and how many from them survived and how many of them didn't survive?

In [37]:
```
df[df['Age']<=10].count()
```

Out[37]:
```
Survived    64
Pclass      64
Name        64
Sex         64
Age         64
SibSp       64
Parch       64
Ticket      64
Fare        64
Embarked    64
dtype: int64
```

In [38]:
```
df[(df['Age']<=10)&(df['Survived']==1)].count()[0:1]
```

Out[38]:
```
Survived    38
dtype: int64
```

In [39]:
```
df[(df['Age']<=10)&(df['Survived']==0)].count()
```

Out[39]:
```
Survived    26
Pclass      26
Name        26
Sex         26
Age         26
SibSp       26
Parch       26
Ticket      26
Fare        26
Embarked    26
dtype: int64
```

64 children were less than 10 years on titanic ship.Out of which only 38 survived and 26 of them didn't survived.

How many from surviving people were female child less than 10 years?

In [40]:
```
df[(df['Age']<=10)&(df['Survived']==1)&(df['Sex']=='female')].count()[0:1]
```

Out[40]:
```
Survived    19
dtype: int64
```

From surviving people there were 19 female child with age less than 10 years.

How many people from not surviving were male child less than 10 years?

In [41]:
```
df[(df['Age']<=10)&(df['Survived']==0)&(df['Sex']=='male')].count()
```

Out[41]:
```
Survived    14
Pclass      14
Name        14
Sex         14
Age         14
SibSp       14
Parch       14
Ticket      14
Fare        14
Embarked    14
dtype: int64
```

There were 14 people not surviving who were male and age was less than 10 years.

What was the surviving rate on titanic?

In [42]:
```
df[df['Survived']==1].count()[0:1]/len(df)*100
```

Out[42]:
```
Survived    38.245219
dtype: float64
```

The surviving rate on titanic was 38.25%

What was the death rate on titanic?

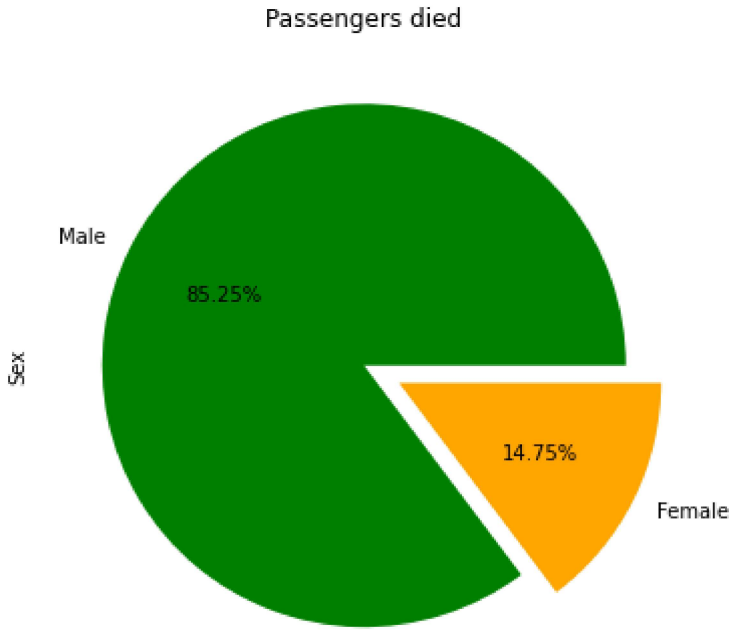
In [43]:
```
df[df['Survived']==0].count()[0:1]/len(df)*100
```

Out[43]:
```
Survived    61.754781
dtype: float64
```

The Death rate on titanic was 61.75%

Out of passengers died on titanic what was the percentage of male and female?

In [44]:
```
plt.figure(figsize=(6,6))
df[df['Survived']==0]['Sex'].value_counts().plot.pie(autopct='%1.2f%%',labels=['M
plt.title('Passengers died')
plt.show()
```



Out of all passengers who died on titanic there were 85.25% Male and 14.75% people.

What was the count of people who coludn't survive from each class?

In [45]:
```
df[df['Survived']==0]['Pclass'].value_counts()
```

Out[45]:
```
3    372
2     97
1     80
Name: Pclass, dtype: int64
```

The count of people who couldn't survive from each class are: from class 1 - 80 people couldn't survive from class 2 - 97 people couldn't survive from class 3 - 372 people couldn't survive

How many passengers who couldn't survive were male from each class?

In [46]:
```
df[(df['Survived']==0)&(df['Sex']=='male')]['Pclass'].value_counts()
```

Out[46]:
```
3    300
2     91
1     77
Name: Pclass, dtype: int64
```

From class 1 there were 77 people who were male and were not able to survive. From class 2 there were 91 people who were male and were not able to survive. From class 3 there were 300 people who were male and were not able to survive.

How many people from each class survived and were male?

In [47]:
```
df[(df['Survived']==1)&(df['Sex']=='male')]['Pclass'].value_counts()
```

Out[47]:
```
3    47
1    45
2    17
Name: Pclass, dtype: int64
```

From class 1 there were 45 people who were male and were able to survive. From class 2 there were 17 people who were male and were able to survive. From class 3 there were 47 people who were male and were able to survive.

What was the average age of people who survived the crash?

In [48]:
```
df[df['Survived']==0]['Age'].mean()
```

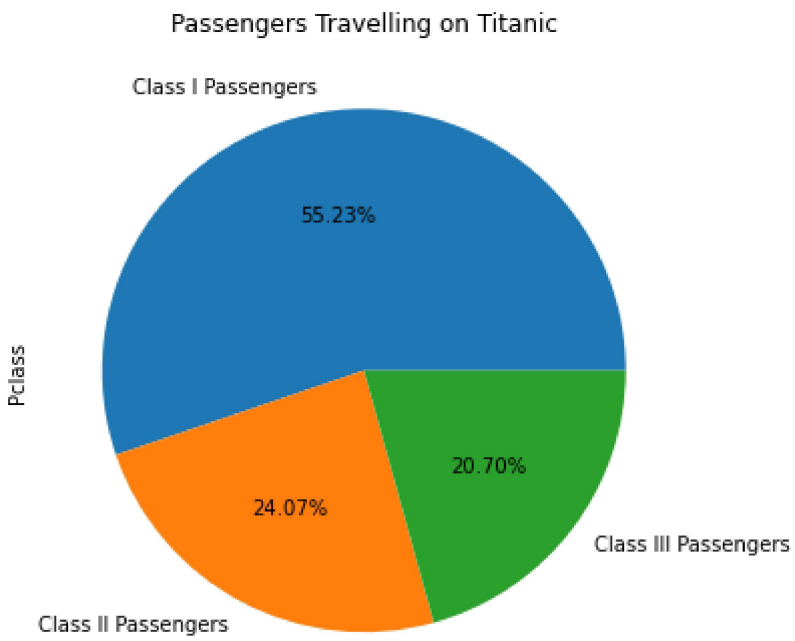
Out[48]:
```
30.415099646415896
```

The average age of people surviving the crash was around 30 years.

What was the rate of passengers 'class' travelling through titanic?

What was the rate of passengers 'class' travelling through titanic?

In [49]:
```
plt.figure(figsize=(6,6))
df['Pclass'].value_counts().plot.pie(autopct='%1.2f%%',labels=['Class I Passenger
plt.title('Passengers Travelling on Titanic')
plt.show()
```



Among the people travelling through the titanic there were 55.23% of people from class 1, 24.07% of people from class 2, 20.70% of people from class 3.

Most number of people embarked were from which place?

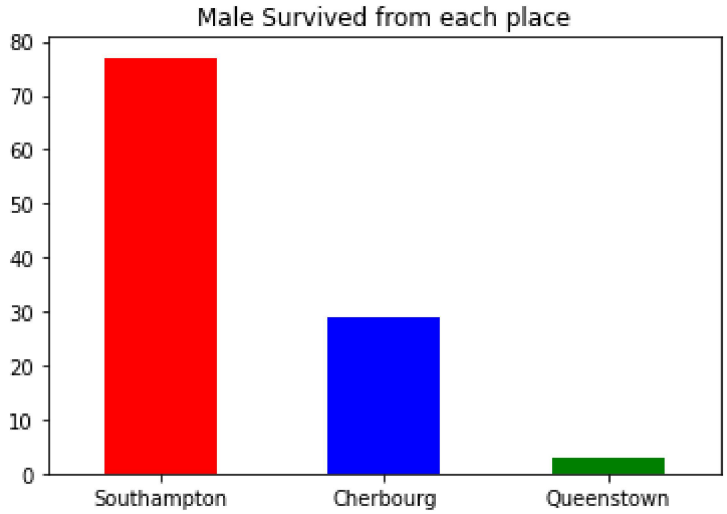
In [50]:
```
df[df['Pclass']==1]['Embarked'].mode()
```

Out[50]:
```
0    S
Name: Embarked, dtype: object
```

The most number of passengers embarked were from 'Southampton'.

What was the count of male survived from each city?

In [51]:
```
df[(df['Sex']=='male')&(df['Survived']==1)]['Embarked'].value_counts().plot.bar(c
plt.title('Male Survived from each place')
plt.xticks(rotation=360,ticks=[0,1,2],labels=['Southampton','Cherbourg','Queensto
plt.show()
```

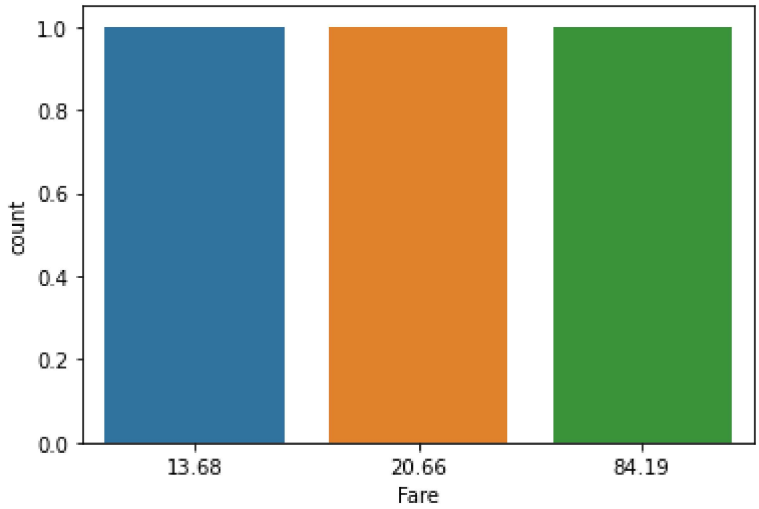


The Male survived from Southampton were 77, from Cherbourg were 29 and male survived from Queenstown were 3.

What was the average fare of each class?

In [52]:
```
df.groupby('Pclass')['Fare'].mean()
sns.countplot(round(df.groupby('Pclass')['Fare'].mean(),2))
```

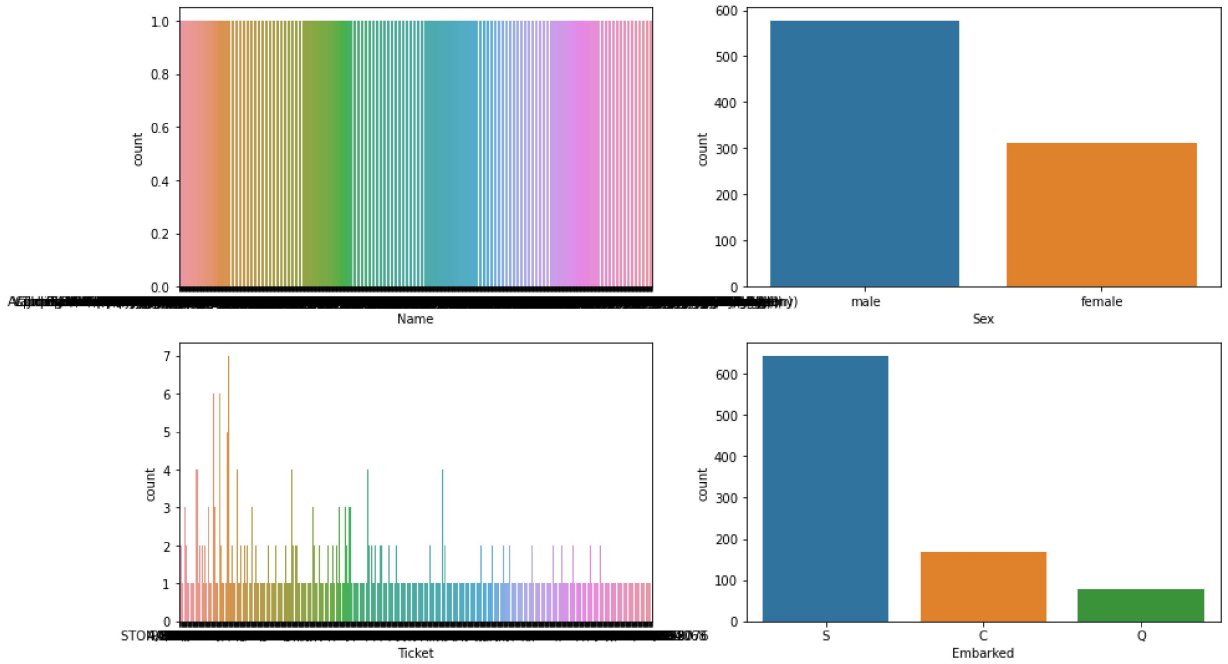
Out[52]:
```
<AxesSubplot:xlabel='Fare', ylabel='count'>
```



The average fare of class 1 was $84.19, The average fare of class 2 was 20.66, and average fare of class 3 was $13.68.

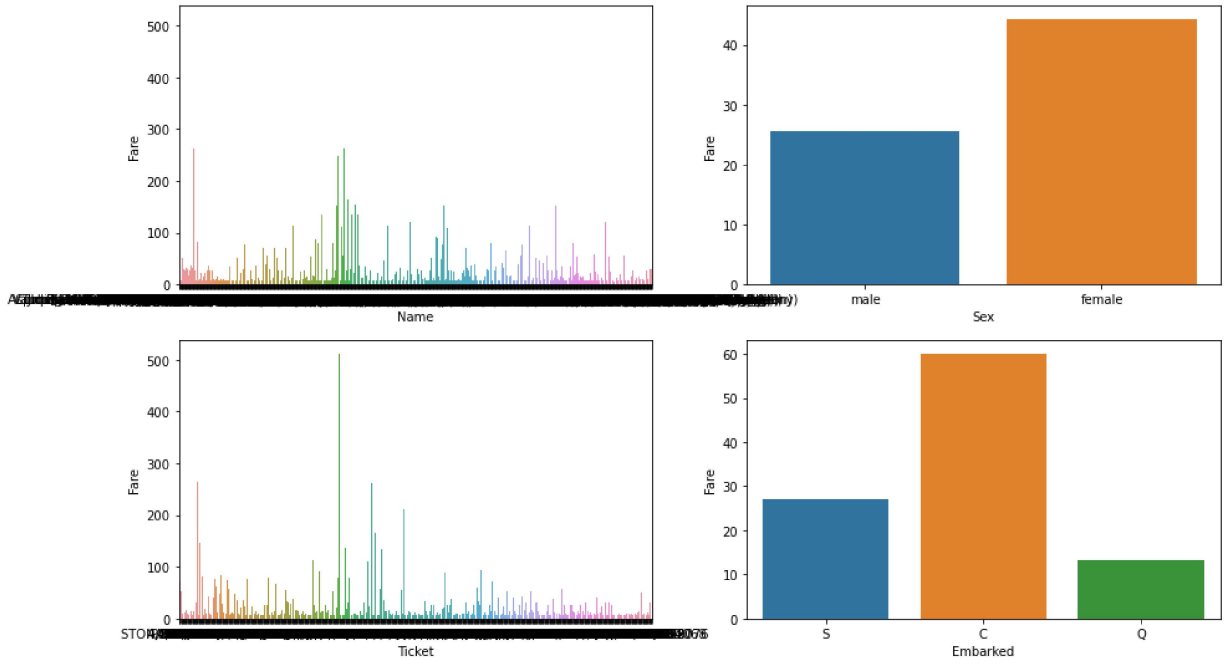
Countplot of data

In [53]:
```
plt.figure(figsize=(15,14))
count=1
for i in cat_col:
    plt.subplot(3,2,count)
    sns.countplot(df[i])
    count+=1
plt.show()
```



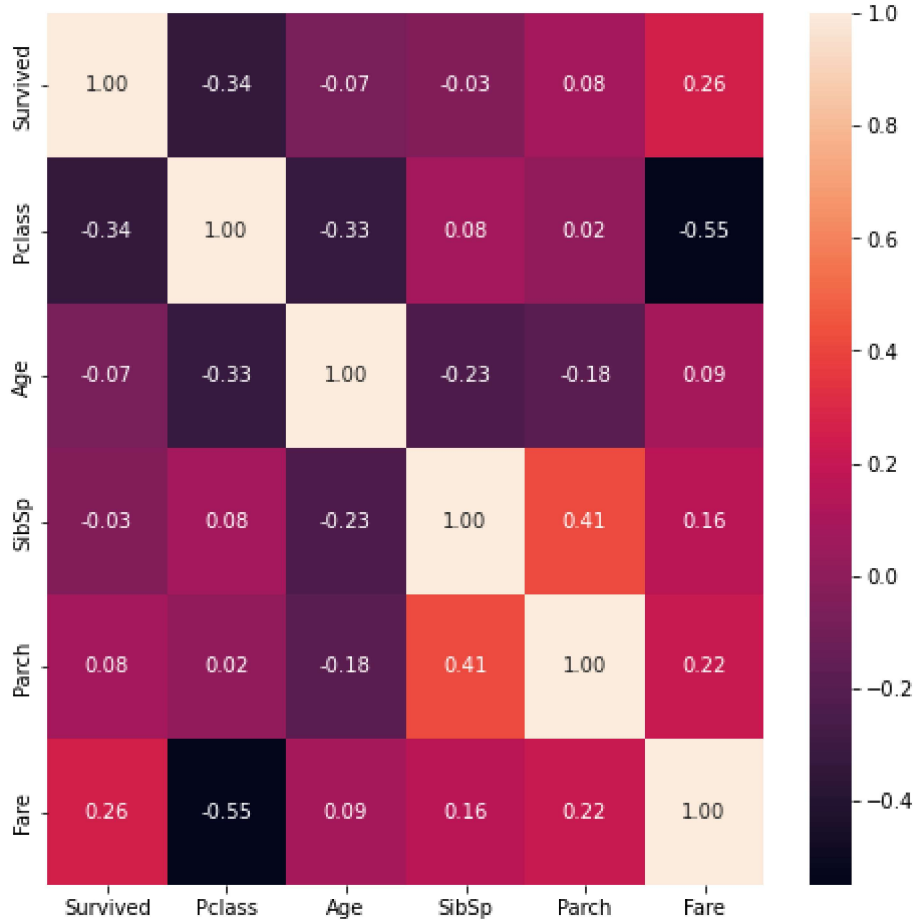
Bar Plot of data with respect to Fare

In [54]:
```
plt.figure(figsize=(15,14))
count=1
for i in cat_col:
    plt.subplot(3,2,count)
    sns.barplot(x=i,y='Fare',data=df,ci=False)
    count+=1
plt.show()
```



Numerical Correlation between data using heatmap

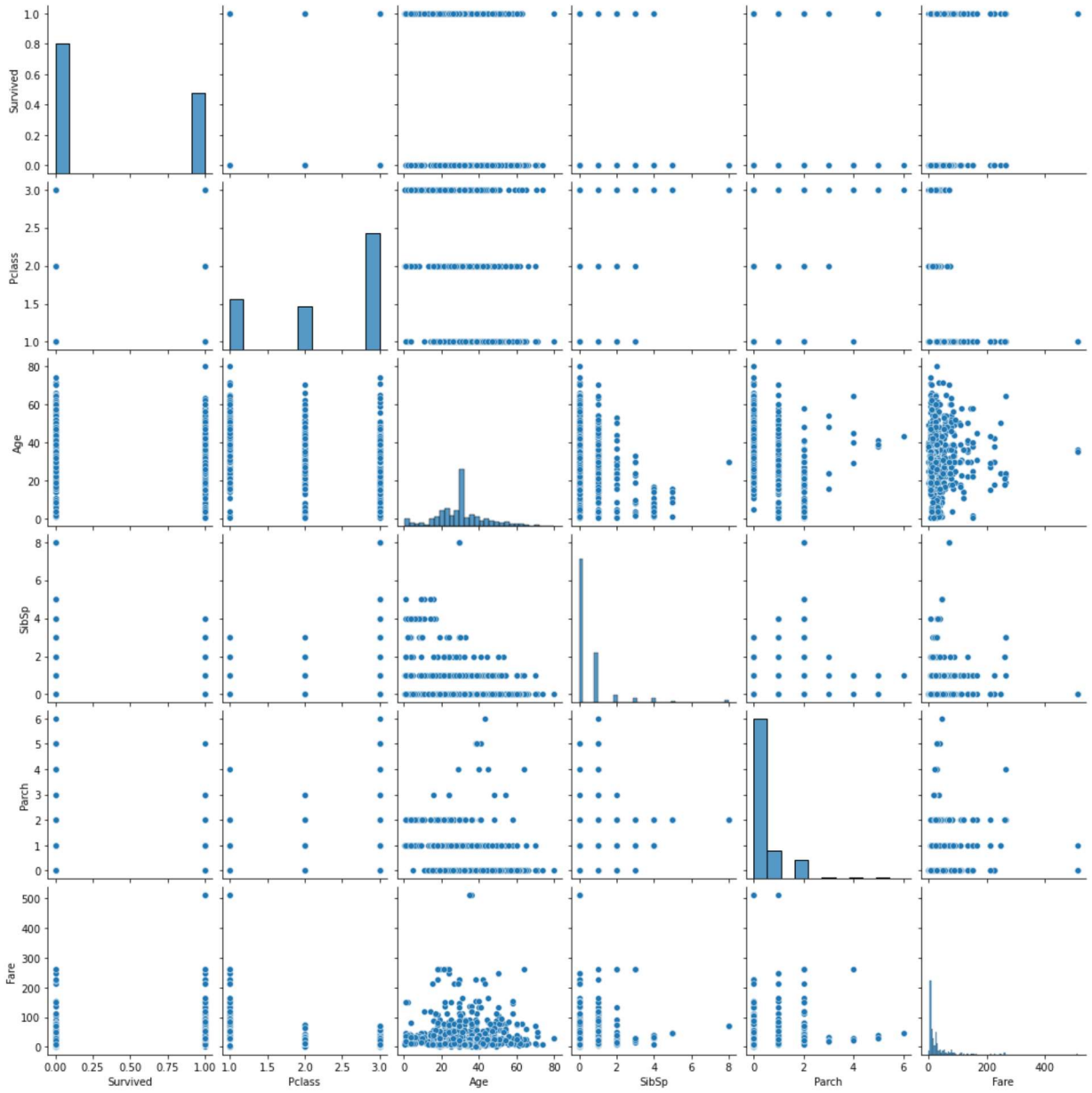
In [55]:
```
plt.figure(figsize=(8,8))
sns.heatmap(df.corr(),annot=True,fmt='.2f')
plt.show()
```



Graphical representation of numerical data using Pairplot

In [56]:
```
plt.figure(figsize=(15,15))
sns.pairplot(df)
plt.show()
```

```
<Figure size 1080x1080 with 0 Axes>
```



In [57]:
```
cat_col
```

Out[57]:
```
Index(['Name', 'Sex', 'Ticket', 'Embarked'], dtype='object')
```

Encoding Categorical data to numerical

In [58]:
```
from sklearn.preprocessing import LabelEncoder
```

In [59]:
```
le=LabelEncoder()
```

In [60]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [61]:
```
ohe=OneHotEncoder(sparse=False)
```

In [62]:
```
df['Embarked']=ohe.fit_transform(df[['Embarked']])
```

In [63]:
```
df['Name']=ohe.fit_transform(df[['Embarked']])
```

In [64]:
```
df['Ticket']=ohe.fit_transform(df[['Ticket']])
```

In [65]:
```
df['Sex']=le.fit_transform(df['Sex'])
```

In [66]:
```
df
```

Out[66]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	1	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	0	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...	...
884	0	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	0	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	0	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 10 columns

Survival Prediction using Logistic Regression

In [67]:
```
df.head()
```

Out[67]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	1.0	1	22.0	1	0	0.0	7.2500	0.0
1	1	1	0.0	0	38.0	1	0	0.0	71.2833	1.0
2	1	3	1.0	0	26.0	0	0	0.0	7.9250	0.0
3	1	1	1.0	0	35.0	1	0	0.0	53.1000	0.0
4	0	3	1.0	1	35.0	0	0	0.0	8.0500	0.0

Splitting data into features and target

In [68]:
```
X=df.iloc[:,1:]
X
```

Out[68]:

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...
884	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 9 columns

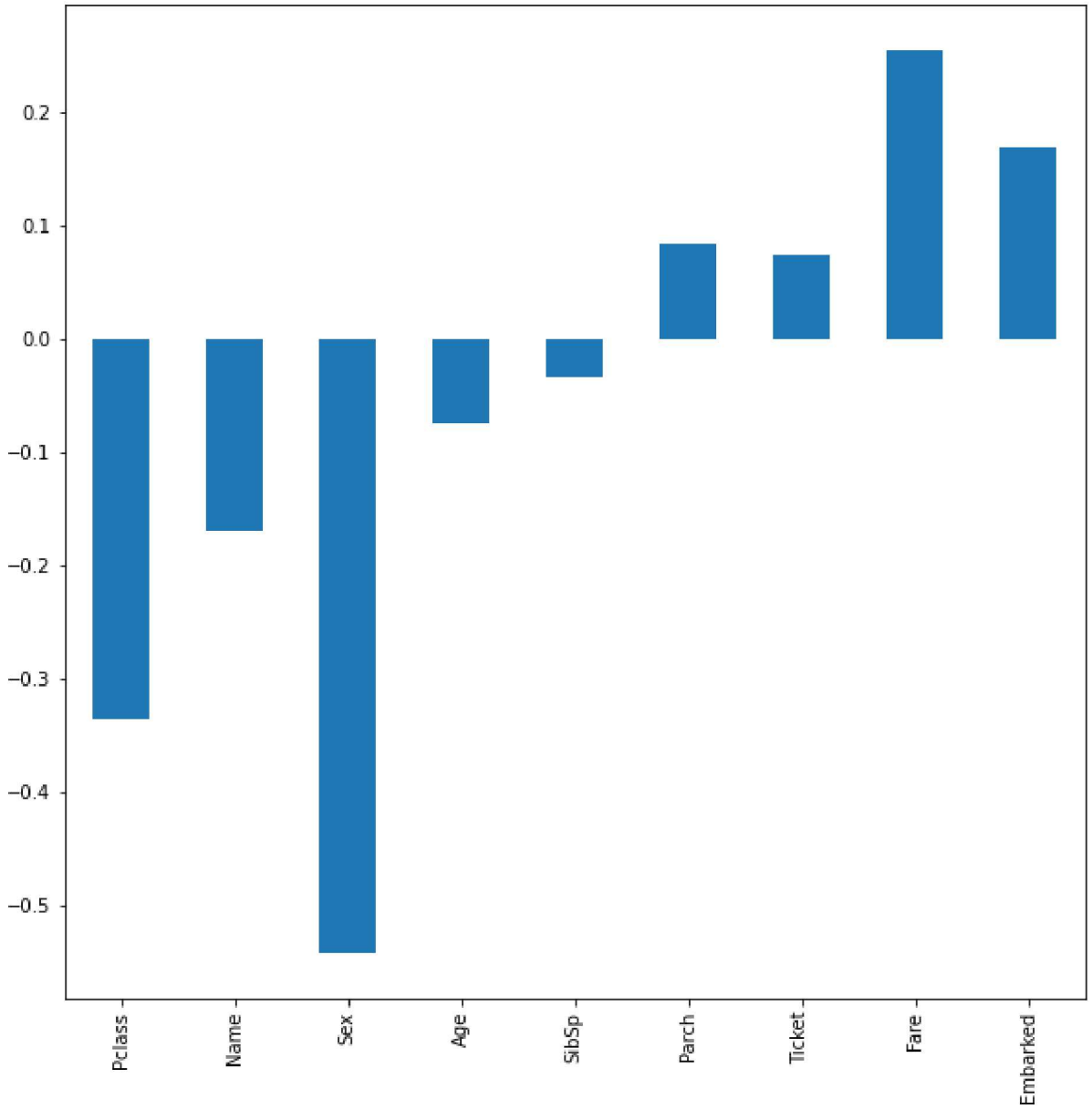
In [69]:
```
y=df['Survived']
y
```

Out[69]:
```
0      0
1      1
2      1
3      1
4      0
      ..
884    0
885    1
886    0
887    1
888    0
Name: Survived, Length: 889, dtype: int64
```

Correlation of features and target variable

In [70]:
```
X.corrwith(y).plot.bar(figsize=(10,10))
```

Out[70]: <AxesSubplot:>



Train Test Split for separating data into training and testing phase

In [71]:
```
from sklearn.model_selection import train_test_split
```

In [72]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=1)
```

In [73]:
```
X_train.shape
```

Out[73]: (711, 9)

In [74]:
```
X_test.shape
```

Out[74]: (178, 9)

In [75]:
```
y_train.shape
```

Out[75]: (711,)

In [76]:
```
y_test.shape
```

Out[76]: (178,)

Applying Standard Scaler to scale the data at one level

In [77]:
```
from sklearn.preprocessing import StandardScaler
```

In [78]:
```
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [79]:
```
X_train
```

Out[79]:
```
array([[ 0.81095497,  0.47747175,  0.72508694, ..., -0.06509446,
        -0.5475599 , -0.47747175],
       [ 0.81095497,  0.47747175,  0.72508694, ..., -0.06509446,
        -0.49483135, -0.47747175],
       [ 0.81095497,  0.47747175,  0.72508694, ..., -0.06509446,
        -0.55501867, -0.47747175],
       ...,
       [ 0.81095497, -2.09436473,  0.72508694, ..., -0.06509446,
        -0.38833499,  2.09436473],
       [-0.39529813,  0.47747175,  0.72508694, ..., -0.06509446,
        -0.10127665, -0.47747175],
       [ 0.81095497,  0.47747175,  0.72508694, ..., -0.06509446,
        -0.5475599 , -0.47747175]])
```

In [80]:
```
X_test
```

Out[80]:
```
array([[-0.39529813,  0.47747175, -1.37914496, ..., -0.06509446,
        -0.42449015, -0.47747175],
       [-0.39529813,  0.47747175, -1.37914496, ..., -0.06509446,
        -0.10127665, -0.47747175],
       [ 0.81095497,  0.47747175,  0.72508694, ..., -0.06509446,
        -0.55242799, -0.47747175],
       ...,
       [-1.60155123, -2.09436473,  0.72508694, ..., -0.06509446,
         2.62448964,  2.09436473],
       [ 0.81095497,  0.47747175,  0.72508694, ..., -0.06509446,
        -0.5475599 , -0.47747175],
       [-0.39529813,  0.47747175,  0.72508694, ..., -0.06509446,
        -0.48664659, -0.47747175]])
```

Training model on Logistic Regression

In [81]:
```
from sklearn.linear_model import LogisticRegression
```

In [82]:
```
reg=LogisticRegression()
reg.fit(X_train,y_train)
y_pred_train=reg.predict(X_train)
y_pred_test=reg.predict(X_test)
```

In [83]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,r2_score,mean_squared
```

In [84]:
```
print('Train Data')
print(confusion_matrix(y_train,y_pred_train))
print('Test Data')
print(confusion_matrix(y_test,y_pred_test))
```

```
Train Data
[[382  62]
 [ 84 183]]
Test Data
[[90 15]
 [15 58]]
```

In [85]:
```
print('Train Data')
print(accuracy_score(y_train,y_pred_train))
print('Test Data')
print(accuracy_score(y_test,y_pred_test))
```

```
Train Data
0.7946554149085795
Test Data
0.8314606741573034
```

In [86]:
```
y_train_proba=reg.predict_proba(X_train)[:,1]
y_train_proba
```

```
       0.95807046, 0.15521056, 0.81272835, 0.09694706, 0.34018136,
       0.10142875, 0.62839206, 0.1014174 , 0.08268587, 0.13626438,
       0.83831603, 0.101439  , 0.14102689, 0.14097205, 0.75419223,
       0.10153543, 0.07838357, 0.87549794, 0.81618801, 0.57783087,
       0.50341506, 0.41039458, 0.10391765, 0.10470507, 0.30642408,
       0.06909336, 0.51382291, 0.67073551, 0.54571586, 0.54417666,
       0.9059282 , 0.59229556, 0.27562739, 0.9593741 , 0.09364846,
       0.15521056, 0.1559031 , 0.71465314, 0.50217165, 0.27108563,
       0.35335991, 0.12324815, 0.5778331 , 0.13626438, 0.89204437,
       0.04632952, 0.11014436, 0.64170076, 0.101439  , 0.86724338,
       0.11129487, 0.10186857, 0.41669068, 0.79161428, 0.93360882,
       0.06567766, 0.18582873, 0.5778331 , 0.09695823, 0.19335259,
       0.36986612, 0.1191262 , 0.85514721, 0.24248048, 0.35591504,
       0.41428077, 0.13848612, 0.95133421, 0.93806883, 0.10142875,
       0.06113072, 0.89210006, 0.10048403, 0.15521015, 0.47725362,
       0.20133216, 0.28353687, 0.23731081, 0.91224663, 0.26144786,
       0.36302746, 0.70800085, 0.94359907, 0.28693139, 0.10749219,
       0.79150851, 0.10137255, 0.48635748, 0.27792016, 0.94799711,
       0.101439  , 0.50972237, 0.90418642, 0.10142875, 0.27599916,
       0.70488439, 0.43146784, 0.05430371, 0.61590797, 0.04737914,
```

In [87]:
```
y_test_proba=reg.predict_proba(X_test)[:,1]
y_test_proba
```

Out[87]:
```
array([0.75460459, 0.63328221, 0.0528634 , 0.69868674, 0.04522877,
       0.87632174, 0.6825713 , 0.45780851, 0.17734607, 0.61215053,
       0.27123438, 0.77009971, 0.24422621, 0.85368639, 0.13179583,
       0.64832467, 0.09092807, 0.724999  , 0.84297968, 0.55991444,
       0.09669069, 0.44286681, 0.13309368, 0.05048416, 0.45687095,
       0.50856735, 0.95976619, 0.12756241, 0.07896119, 0.57783977,
       0.57788647, 0.6378936 , 0.85655855, 0.58406818, 0.48271451,
       0.84536852, 0.34843513, 0.10144316, 0.76652799, 0.39506133,
       0.58048159, 0.08815371, 0.67889027, 0.101439  , 0.21170408,
       0.7819814 , 0.84800636, 0.18571733, 0.94469468, 0.07575311,
       0.10140577, 0.60478963, 0.07564837, 0.92136231, 0.24888513,
       0.93661501, 0.22145625, 0.04556131, 0.29486895, 0.04480449,
       0.19307847, 0.13630802, 0.4836251 , 0.101439  , 0.60271703,
       0.77385499, 0.10186857, 0.08722675, 0.1495789 , 0.95591016,
       0.15561519, 0.94846382, 0.34288337, 0.1362583 , 0.70939176,
       0.87623477, 0.10141906, 0.07274488, 0.11740137, 0.16756159,
       0.10141906, 0.1318087 , 0.9593741 , 0.94050374, 0.23510477,
       0.3804955 , 0.44103301, 0.61775199, 0.101439  , 0.6484376 ,
       0.10038849, 0.5778331 , 0.06300779, 0.15521056, 0.69021662,
       0.15680469, 0.78036567, 0.72818283, 0.69021727, 0.80073157,
       0.14559023, 0.67206091, 0.34512868, 0.14085103, 0.14092238,
       0.07894186, 0.87099761, 0.14557624, 0.19812917, 0.18428785,
       0.3642204 , 0.24248048, 0.15521056, 0.79685971, 0.65915785,
       0.91183488, 0.06994318, 0.61213827, 0.95025321, 0.20825792,
       0.11908858, 0.75959916, 0.57790203, 0.49842492, 0.80971127,
       0.10627939, 0.56275268, 0.07246184, 0.11515775, 0.36258369,
       0.50972237, 0.34493297, 0.11312081, 0.48208627, 0.6478943 ,
       0.93423055, 0.13696761, 0.07314567, 0.10190861, 0.10857354,
       0.14258802, 0.70092874, 0.07579631, 0.62018609, 0.47725362,
       0.26533341, 0.78818693, 0.90409601, 0.93113432, 0.05209849,
       0.09345968, 0.06330777, 0.84248526, 0.14557624, 0.58182838,
       0.64798164, 0.21891617, 0.61201421, 0.10765759, 0.10142072,
       0.50005671, 0.22793421, 0.36258369, 0.07892938, 0.1288839 ,
       0.16054626, 0.85934285, 0.95528384, 0.21199769, 0.77669014,
       0.14486146, 0.08353676, 0.10141906, 0.12322812, 0.27561895,
       0.68821244, 0.1318087 , 0.2865585 ])
```

Classification Report of model trained on Logistic Regression

In [89]:
```
from sklearn.metrics import classification_report
```

In [90]:
```
print('Train Data')
print(classification_report(y_train,y_pred_train))
print('Test Data')
print(classification_report(y_test,y_pred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       444
           1       0.75      0.69      0.71       267

    accuracy                           0.79       711
   macro avg       0.78      0.77      0.78       711
weighted avg       0.79      0.79      0.79       711

Test Data
              precision    recall  f1-score   support

           0       0.86      0.86      0.86       105
           1       0.79      0.79      0.79        73

    accuracy                           0.83       178
   macro avg       0.83      0.83      0.83       178
weighted avg       0.83      0.83      0.83       178
```

PR Curve

In [92]:
```
from sklearn.metrics import precision_recall_curve
p,r,th=precision_recall_curve(y_train,y_train_proba)
```

In [93]:
```
p.shape
```

Out[93]: (602,)

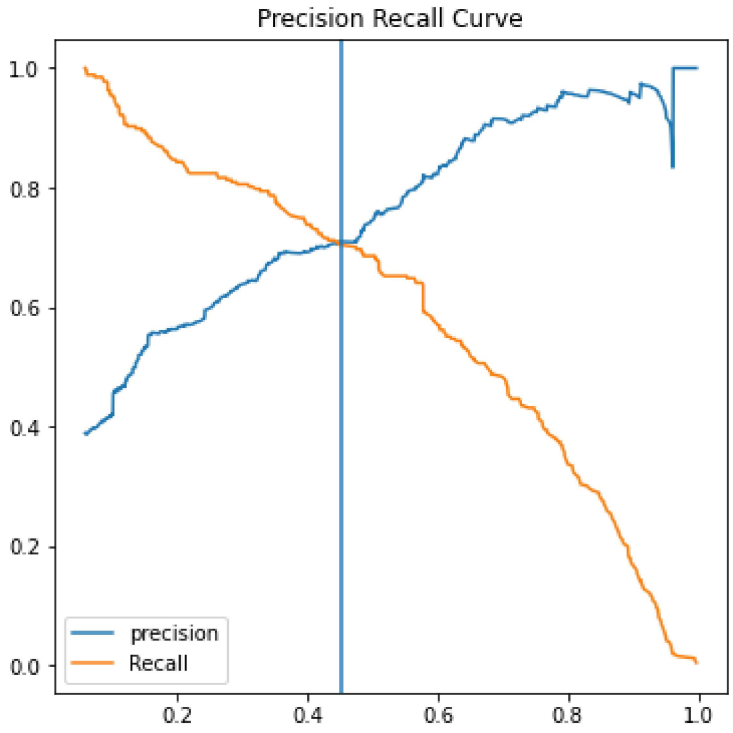
In [94]:
```
r.shape
```

Out[94]: (602,)

In [95]:
```
th.shape
```

Out[95]: (601,)

In [96]:
```
plt.figure(figsize=(6,6))
sns.lineplot(x=th,y=p[:-1],label='precision')
sns.lineplot(x=th,y=r[:-1],label='Recall')
plt.title('Precision Recall Curve')
plt.axvline(0.452)
plt.show()
```



In [97]:
```
from sklearn.metrics import accuracy_score,recall_score,precision_score,roc_auc_s
```

In [98]:
```
def metrics(y_actual,y_proba,th):
    y_pred_temp=[1 if p>th else 0 for p in y_proba]
    accuracy=accuracy_score(y_actual,y_pred_temp)
    recall=recall_score(y_actual,y_pred_temp)
    precision=precision_score(y_actual,y_pred_temp)
    f1=f1_score(y_actual,y_pred_temp)
    roc_auc=roc_auc_score(y_actual,y_pred_temp)
    return {"Accuracy":round(accuracy,2),'Recall':round(recall,2),'Precision':rou
```

In [99]:
```
print("Train Data")
print(metrics(y_train,y_train_proba,0.452))
print("Test Data")
print(metrics(y_test,y_test_proba,0.452))
```

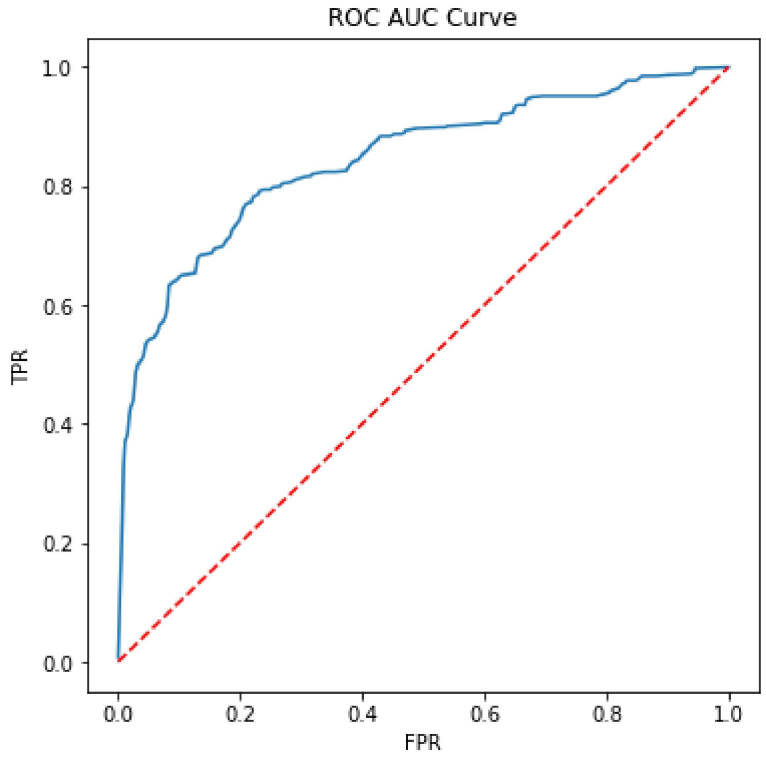
```
Train Data
{'Accuracy': 0.78, 'Recall': 0.7, 'Precision': 0.71, 'F1_Score': 0.7, 'ROC_AU
C': 0.76}
Test Data
{'Accuracy': 0.83, 'Recall': 0.84, 'Precision': 0.76, 'F1_Score': 0.8, 'ROC_AU
C': 0.83}
```

ROC-AUC Curve

In [100]:
```
from sklearn.metrics import roc_curve,auc
```

In [101]:
```
fpr,tpr,th=roc_curve(y_train,y_train_proba)
```

In [102]:
```
plt.figure(figsize=(6,6))
sns.lineplot(x=fpr,y=tpr)
sns.lineplot(x=[0.0,1],y=[0.0,1],color='red',linestyle='--')
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title('ROC AUC Curve')
plt.show()
```



Survival prediction using naive bayes

In [103]:
```
df
```

Out[103]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	1	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	0	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...	...
884	0	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	0	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	0	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 10 columns

Splitting data into features and target

In [105]:
```
X=df.iloc[:,1:]
X
```

Out[105]:

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...
884	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 9 columns

In [106]:
```
y=df['Survived']
y
```

Out[106]:
```
0      0
1      1
2      1
3      1
4      0
      ..
884    0
885    1
886    0
887    1
888    0
Name: Survived, Length: 889, dtype: int64
```

Performing train_test_split to split data into training and testing phase

In [108]:
```
from sklearn.model_selection import train_test_split
```

In [109]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=1)
```

In [110]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Applying Naive Bayes Classifier

In [111]:
```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X_train, y_train)
```

Out[111]:
```
GaussianNB()
```

In [ ]:
```
y_pred_train=clf.predict(X_train)
y_pred_test=clf.predict(X_test)
```

Classification Report of Naive Bayes Classifier

In [113]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred_test)
```

Out[113]:
```
array([[164,   2],
       [ 93,   8]], dtype=int64)
```

In [114]:
```
print('Train Data')
print(accuracy_score(y_train,y_pred_train))
print('Test Data')
print(accuracy_score(y_test,y_pred_test))
```

```
Train Data
0.6318327974276527
Test Data
0.6441947565543071
```

In [116]:
```
print('Train Data')
print(classification_report(y_train,y_pred_train))
print('Test Data')
print(classification_report(y_test,y_pred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.63      0.99      0.77       383
           1       0.78      0.06      0.11       239

    accuracy                           0.63       622
   macro avg       0.70      0.52      0.44       622
weighted avg       0.69      0.63      0.51       622

Test Data
              precision    recall  f1-score   support

           0       0.64      0.99      0.78       166
           1       0.80      0.08      0.14       101

    accuracy                           0.64       267
   macro avg       0.72      0.53      0.46       267
weighted avg       0.70      0.64      0.54       267
```

Survival Prediction using KNN Algorithm

In [117]:
```
df
```

Out[117]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	1	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	0	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...	...
884	0	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	0	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	0	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 10 columns

Splitting featurse and target

In [118]:
```
X=df.iloc[:,1:]
X
```

Out[118]:

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...
884	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 9 columns

In [119]:
```
y=df['Survived']
y
```

Out[119]:
```
0      0
1      1
2      1
3      1
4      0
      ..
884    0
885    1
886    0
887    1
888    0
Name: Survived, Length: 889, dtype: int64
```

Performing train_test_split

In [120]:
```
from sklearn.model_selection import train_test_split
```

In [121]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=1)
```

In [122]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Applying K-Nearest Neighbours

In [123]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [124]:
```
clf=KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train,y_train)
y_pred_train=clf.predict(X_train)
y_pred_test=clf.predict(X_test)
```

In [125]:
```
from sklearn.metrics import classification_report,f1_score
```

Classification Report

In [126]:
```
print('Train Data')
print(classification_report(y_train,y_pred_train))
print('Test Data')
print(classification_report(y_test,y_pred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.86      0.88      0.87       383
           1       0.80      0.76      0.78       239

    accuracy                           0.83       622
   macro avg       0.83      0.82      0.82       622
weighted avg       0.83      0.83      0.83       622

Test Data
              precision    recall  f1-score   support

           0       0.87      0.86      0.86       166
           1       0.77      0.79      0.78       101

    accuracy                           0.83       267
   macro avg       0.82      0.82      0.82       267
weighted avg       0.83      0.83      0.83       267
```

Survival Prediction using Decision Tree

In [127]:
```
df
```

Out[127]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	1	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	0	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...	...
884	0	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	0	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	0	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 10 columns

In [128]:
```
X=df.iloc[:,1:]
X
```

Out[128]:

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...
884	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 9 columns

In [129]:
```
y=df['Survived']
y
```

Out[129]:
```
0      0
1      1
2      1
3      1
4      0
      ..
884    0
885    1
886    0
887    1
888    0
Name: Survived, Length: 889, dtype: int64
```

Splitting Train data and Test data

In [130]:
```
from sklearn.model_selection import train_test_split
```

In [131]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=1)
```

Applying Decision Tree Classifier

In [132]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [133]:
```
dt=DecisionTreeClassifier()
```

In [134]:
```
dt.fit(X_train,y_train)
```

Out[134]:
```
DecisionTreeClassifier()
```

In [135]:
```
from sklearn.metrics import classification_report
```

In [136]:
```
y_train_pred=dt.predict(X_train)
y_test_pred=dt.predict(X_test)
```

Classification Report on Decision Tree classifier

In [137]:
```
print('Train Data')
print(classification_report(y_train,y_train_pred))
print('Test Data')
print(classification_report(y_test,y_test_pred))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       383
           1       0.99      0.97      0.98       239

    accuracy                           0.99       622
   macro avg       0.99      0.98      0.99       622
weighted avg       0.99      0.99      0.99       622

Test Data
              precision    recall  f1-score   support

           0       0.84      0.77      0.80       166
           1       0.66      0.75      0.70       101

    accuracy                           0.76       267
   macro avg       0.75      0.76      0.75       267
weighted avg       0.77      0.76      0.76       267
```

Hyper-Parameter tuning

In [138]:
```
param={
    'criterion':['gini','entropy'],
    'class_weight':[None,'balanced'],
    'max_depth':np.arange(1,11),
    'min_samples_leaf':np.arange(1,6),
    'min_samples_split':np.arange(1,6)
}
```

In [139]:
```
from sklearn.model_selection import GridSearchCV
```

In [140]:
```
gridclf=GridSearchCV(dt,param_grid=param,scoring='f1',cv=5,n_jobs=-1)
gridclf.fit(X_train,y_train)
```

Out[140]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'class_weight': [None, 'balanced'],
                         'criterion': ['gini', 'entropy'],
                         'max_depth': array([ 1,  2,  3,  4,  5,  6,  7,  8,
9, 10]),
                         'min_samples_leaf': array([1, 2, 3, 4, 5]),
                         'min_samples_split': array([1, 2, 3, 4, 5])},
             scoring='f1')
```

In [141]:
```
gridclf.best_params_
```

Out[141]:
```
{'class_weight': 'balanced',
 'criterion': 'entropy',
 'max_depth': 4,
 'min_samples_leaf': 3,
 'min_samples_split': 4}
```

In [142]:
```
grid_train_predictions = gridclf.predict(X_train)
grid_test_predictions = gridclf.predict(X_test)
```

Classification report of Decision tree after tuning the model

```
print(classification_report(y_train, grid_train_predictions))
print(classification_report(y_test, grid_test_predictions))
```

Survival Prediction using Random Forest

In [144]:
```
df
```

Out[144]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	1	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	0	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...	...
884	0	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	0	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	0	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 10 columns

Splitting data into features and target

In [145]:
```
X=df.iloc[:,1:]
X
```

Out[145]:

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...
884	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 9 columns

In [146]:
```
y=df['Survived']
y
```

Out[146]:
```
0      0
1      1
2      1
3      1
4      0
      ..
884    0
885    1
886    0
887    1
888    0
Name: Survived, Length: 889, dtype: int64
```

Train test split for splitting data into training and testing phase

In [147]:
```
from sklearn.model_selection import train_test_split
```

In [148]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=1)
```

In [149]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [150]:
```
rf=RandomForestClassifier()
```

In [151]:
```
rf.fit(X_train,y_train)
```

Out[151]:
```
RandomForestClassifier()
```

In [152]:
```
from sklearn.metrics import classification_report
```

In [153]:
```
y_train_pred=dt.predict(X_train)
y_test_pred=dt.predict(X_test)
```

Classification Report of Random Forest Classifier

In [154]:
```
print('Train Data')
print(classification_report(y_train,y_train_pred))
print('Test Data')
print(classification_report(y_test,y_test_pred))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       383
           1       0.99      0.97      0.98       239

    accuracy                           0.99       622
   macro avg       0.99      0.98      0.99       622
weighted avg       0.99      0.99      0.99       622

Test Data
              precision    recall  f1-score   support

           0       0.84      0.77      0.80       166
           1       0.66      0.75      0.70       101

    accuracy                           0.76       267
   macro avg       0.75      0.76      0.75       267
weighted avg       0.77      0.76      0.76       267
```

Hyper-Parameter tuning of Random Forest Classifier

In [155]:
```
param_grid={'criterion':['gini','entropy'],
            'class_weight':[None,'balanced'],
            'n_estimators':[100,150,300,600,500],
            'max_depth':np.arange(1,50,2),
            'min_samples_leaf':np.arange(1,20),
            'min_samples_split':np.arange(2,15,1)
           }
```

In [156]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [157]:
```
clf=RandomizedSearchCV(rf,param_distributions=param_grid,n_iter=20,scoring='f1',r
```

In [158]:
```
clf.fit(X_train,y_train)
```

Out[158]:
```
RandomizedSearchCV(estimator=RandomForestClassifier(), n_iter=20, n_jobs=-1,
                   param_distributions={'class_weight': [None, 'balanced'],
                                        'criterion': ['gini', 'entropy'],
                                        'max_depth': array([ 1,  3,  5,  7,  9,
11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
       35, 37, 39, 41, 43, 45, 47, 49]),
                                        'min_samples_leaf': array([ 1,  2,  3,
4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19]),
                                        'min_samples_split': array([ 2,  3,  4,
5,  6,  7,  8,  9, 10, 11, 12, 13, 14]),
                                        'n_estimators': [100, 150, 300, 600,
                                                         500]},
                   scoring='f1')
```

In [159]:
```
clf.best_params_
```

Out[159]:
```
{'n_estimators': 100,
 'min_samples_split': 9,
 'min_samples_leaf': 1,
 'max_depth': 43,
 'criterion': 'gini',
 'class_weight': 'balanced'}
```

In [160]:
```
grid_train_predictions = gridclf.predict(X_train)
grid_test_predictions = gridclf.predict(X_test)
```

Classification Report of Random Forest Classifier after tuning the model

In [161]:
```
print(classification_report(y_train, grid_train_predictions))
print(classification_report(y_test, grid_test_predictions))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86       383
           1       0.82      0.71      0.76       239

    accuracy                           0.83       622
   macro avg       0.82      0.80      0.81       622
weighted avg       0.83      0.83      0.82       622

              precision    recall  f1-score   support

           0       0.87      0.91      0.89       166
           1       0.84      0.77      0.80       101

    accuracy                           0.86       267
   macro avg       0.85      0.84      0.85       267
weighted avg       0.86      0.86      0.86       267
```

Survival prediction using boosting algorithm

In [162]:
```
df
```

Out[162]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	1	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	0	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...	...
884	0	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	0	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	0	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 10 columns

Splitting features and target variable

In [163]:
```
X=df.iloc[:,1:]
X
```

Out[163]:

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	3	1.0	1	22.000000	1	0	0.0	7.2500	0.0
1	1	0.0	0	38.000000	1	0	0.0	71.2833	1.0
2	3	1.0	0	26.000000	0	0	0.0	7.9250	0.0
3	1	1.0	0	35.000000	1	0	0.0	53.1000	0.0
4	3	1.0	1	35.000000	0	0	0.0	8.0500	0.0
...	...	...	...	...	...	...	...	...	...
884	2	1.0	1	27.000000	0	0	0.0	13.0000	0.0
885	1	1.0	0	19.000000	0	0	0.0	30.0000	0.0
886	3	1.0	0	29.699118	1	2	0.0	23.4500	0.0
887	1	0.0	1	26.000000	0	0	0.0	30.0000	1.0
888	3	1.0	1	32.000000	0	0	0.0	7.7500	0.0

889 rows × 9 columns

In [164]:
```
y=df['Survived']
y
```

Out[164]:
```
0      0
1      1
2      1
3      1
4      0
      ..
884    0
885    1
886    0
887    1
888    0
Name: Survived, Length: 889, dtype: int64
```

Performing train_test_split

In [166]:
```
from sklearn.model_selection import train_test_split
```

In [167]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=1)
```

AdaBoostClassifier

In [168]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [169]:
```
ada_clf=AdaBoostClassifier(n_estimators=200)
```

In [170]:
```
ada_clf.fit(X_train,y_train)
y_pred_train=ada_clf.predict(X_train)
y_pred_test=ada_clf.predict(X_test)
```

Classification Report

In [171]:
```
print('Train Data')
print(classification_report(y_train,y_pred_train))
print('Test Data')
print(classification_report(y_test,y_pred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.87      0.89      0.88       383
           1       0.81      0.79      0.80       239

    accuracy                           0.85       622
   macro avg       0.84      0.84      0.84       622
weighted avg       0.85      0.85      0.85       622

Test Data
              precision    recall  f1-score   support

           0       0.88      0.83      0.85       166
           1       0.74      0.81      0.77       101

    accuracy                           0.82       267
   macro avg       0.81      0.82      0.81       267
weighted avg       0.83      0.82      0.82       267
```

Hyper-Parameter tuning of Ada-Boost classifier

In [172]:
```
param_grid={'n_estimators':[100,150,300,600,500],
            'learning_rate':[0.1,0.01,1,2,3]
           }
```

In [173]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [174]:
```
clf=RandomizedSearchCV(ada_clf,param_distributions=param_grid,n_iter=20,scoring='
```

In [175]:
```
clf.fit(X_train,y_train)
```

Out[175]:
```
RandomizedSearchCV(estimator=AdaBoostClassifier(n_estimators=200), n_iter=20,
                   n_jobs=-1,
                   param_distributions={'learning_rate': [0.1, 0.01, 1, 2, 3],
                                        'n_estimators': [100, 150, 300, 600,
                                                         500]},
                   scoring='f1')
```

In [176]:
```
clf.best_params_
```

Out[176]:
```
{'n_estimators': 150, 'learning_rate': 0.1}
```

In [177]:
```
grid_train_predictions = gridclf.predict(X_train)
grid_test_predictions = gridclf.predict(X_test)
```

Classification Report of Ada-Boost Classifier after tuning the model

In [178]:
```
print(classification_report(y_train, grid_train_predictions))
print(classification_report(y_test, grid_test_predictions))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86       383
           1       0.82      0.71      0.76       239

    accuracy                           0.83       622
   macro avg       0.82      0.80      0.81       622
weighted avg       0.83      0.83      0.82       622

              precision    recall  f1-score   support

           0       0.87      0.91      0.89       166
           1       0.84      0.77      0.80       101

    accuracy                           0.86       267
   macro avg       0.85      0.84      0.85       267
weighted avg       0.86      0.86      0.86       267
```

Gradient Boosting Algorithm

In [179]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [180]:
```
gdb_clf=GradientBoostingClassifier(n_estimators=200)
```

In [181]:
```
gdb_clf.fit(X_train,y_train)
y_pred_train=gdb_clf.predict(X_train)
y_pred_test=gdb_clf.predict(X_test)
```

Classification Report

In [182]:
```
print('Train Data')
print(classification_report(y_train,y_pred_train))
print('Test Data')
print(classification_report(y_test,y_pred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       383
           1       0.93      0.87      0.90       239

    accuracy                           0.93       622
   macro avg       0.93      0.92      0.92       622
weighted avg       0.93      0.93      0.93       622

Test Data
              precision    recall  f1-score   support

           0       0.85      0.92      0.88       166
           1       0.84      0.74      0.79       101

    accuracy                           0.85       267
   macro avg       0.85      0.83      0.84       267
weighted avg       0.85      0.85      0.85       267
```

Hyper-Parameter Tuning of Gradient Boosting Algorithm

In [183]:
```
param_grid={'n_estimators':[100,150,300,600,500],
            'learning_rate':[0.1,0.01,1,2,3]
           }
```

In [184]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [185]:
```
clf=RandomizedSearchCV(gdb_clf,param_distributions=param_grid,n_iter=20,scoring=
```

In [186]:
```
clf.fit(X_train,y_train)
```

Out[186]:
```
RandomizedSearchCV(estimator=GradientBoostingClassifier(n_estimators=200),
                   n_iter=20, n_jobs=-1,
                   param_distributions={'learning_rate': [0.1, 0.01, 1, 2, 3],
                                        'n_estimators': [100, 150, 300, 600,
                                                         500]},
                   scoring='f1')
```

In [187]:
```
clf.best_params_
```

Out[187]:
```
{'n_estimators': 300, 'learning_rate': 0.1}
```

In [188]:
```
grid_train_predictions = gridclf.predict(X_train)
grid_test_predictions = gridclf.predict(X_test)
```

Classification Report on Gradient Boosting after tuning

In [189]:
```
print(classification_report(y_train, grid_train_predictions))
print(classification_report(y_test, grid_test_predictions))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86       383
           1       0.82      0.71      0.76       239

    accuracy                           0.83       622
   macro avg       0.82      0.80      0.81       622
weighted avg       0.83      0.83      0.82       622

              precision    recall  f1-score   support

           0       0.87      0.91      0.89       166
           1       0.84      0.77      0.80       101

    accuracy                           0.86       267
   macro avg       0.85      0.84      0.85       267
weighted avg       0.86      0.86      0.86       267
```

DataFrame of Algorithms

In [219]:
```
data={'Classification Algorithms':['Logistic Regression','Naive Bayes Classifier'
```

In [227]:
```
pd.DataFrame(data)
```

Out[227]:

	Classification Algorithms	Train Data	Test Data	After Tuning Train	After Tuning Test
0	Logistic Regression	79	83		
1	Naive Bayes Classifier	63	64		
2	K-Nearest Neighbours Classifier	83	83		
3	Decision Tree Classifier	99	76	83	86
4	Random Forest	99	76	83	86
5	AdaBoost	85	82	83	86
6	Gradient Boost	93	85	83	86

The Best Prediction was done by model trained on XGB Gradient Boosting Algorithm with accuracy of 93% on Train Data and 85% on Test Data