Perform data cleaning and exploratory data analysis (EDA) on a dataset of your choice, such as the Titanic dataset from Kaggle. Explore the relationships between variables and identify patterns and trends in the data.

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv("titanic_train.csv")
df
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.28
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.92
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.10
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75

891 rows × 12 columns

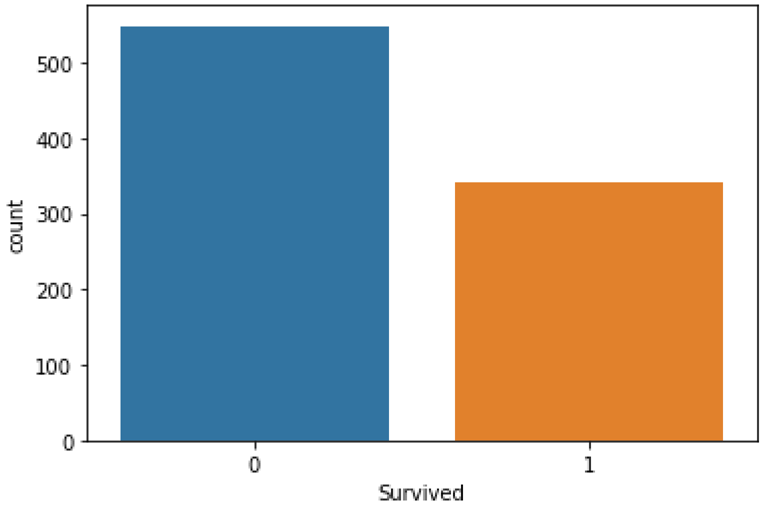
```
information about the dataset:-

- Our Variable Features:

- PassengerId: Unique number of each passenger.
- Survived: 0 = No, 1 = Yes.
- Pclass: Ticket class 1 = 1st, 2 = 2nd, 3 = 3rd.
- Name :- Name of the passengers.
- Sex :- 1:Male, 0:Female.
- Age :- Age of the passengers.
- SibSp: Siblings / Spouses Onboard in the Titanic Ship.
```

```
- Parch: Parents / Children Onboard in the Titanic Ship.
- Ticket: Ticket number.
- Fare: Passenger fare (Ticket Price).
- Cabin: Cabin number.
- Embarked: Port of Embarkation C = Cherbourg, Q = Queenstown, S =
Southampton




- pclass: A proxy for socio-economic status (SES)
      1st = Upper.
      2nd = Middle.
      3rd = Lower.
- age: Age is fractional if less than 1. If the age is estimated, is it
in the form of xx.5.
- sibsp: The dataset defines family relations in this way...

      Sibling = Brother, Sister, Stepbrother, Stepsister.
      Spouse = Husband, Wife (Mistresses and Fiancés were ignored).
- parch: The dataset defines family relations in this way...

      Parent = Mother, Father.
      Child = Daughter, Son, Stepdaughter, Stepson.

      Some children travelled only with a nanny, therefore parch=0 for
them.


The SibSp column represents the number of siblings or spouses that a
passenger had on board the Titanic. A sibling is defined as a brother or
sister of the passenger, while a spouse is defined as a husband or wife.
For example, a value of 1 in the SibSp column means that the passenger
had one sibling or spouse on board, while a value of 0 means that the
passenger was traveling alone.

The Parch column represents the number of parents or children that a
passenger had on board the Titanic. A parent is defined as a mother or
father of the passenger, while a child is defined as a son or daughter.
For example, a value of 1 in the Parch column means that the passenger
had one parent or child on board, while a value of 0 means that the
passenger was not traveling with any parents or children.
```

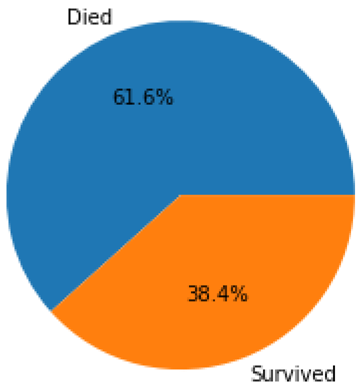
studying the data

In [3]:
```
sns.countplot(data=df,x="Survived")
```

Out[3]: <AxesSubplot:xlabel='Survived', ylabel='count'>



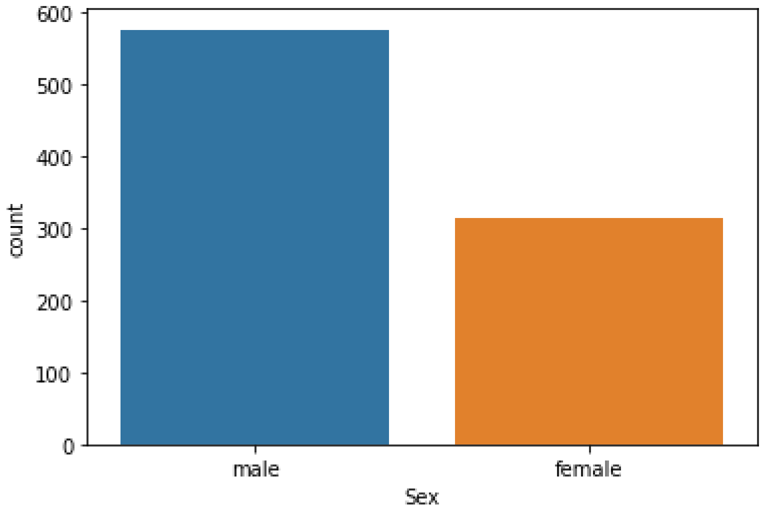
In [4]:
```
plt.pie(df["Survived"].value_counts(),autopct="%1.1f%%",labels=["Died","Sur
plt.show()
```



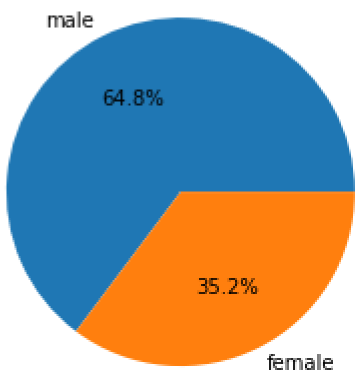

```
From the visualization it is clear that more people are died and less
people are survived
```

In [5]:
```
sns.countplot(data=df,x="Sex")
```

Out[5]: <AxesSubplot:xlabel='Sex', ylabel='count'>

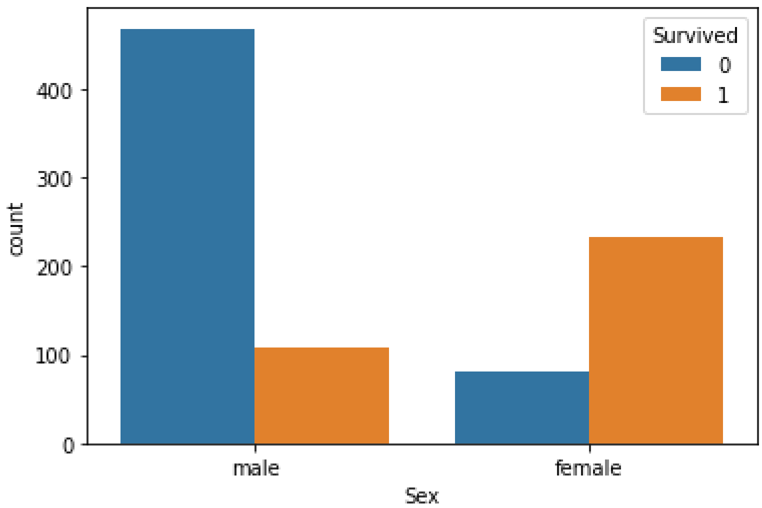


In [6]:
```
plt.pie(df["Sex"].value_counts(),labels=["male","female"],autopct="%1.1f%%"
plt.show()
```



In [7]:
```
sns.countplot(data=df,x="Sex",hue="Survived")
```

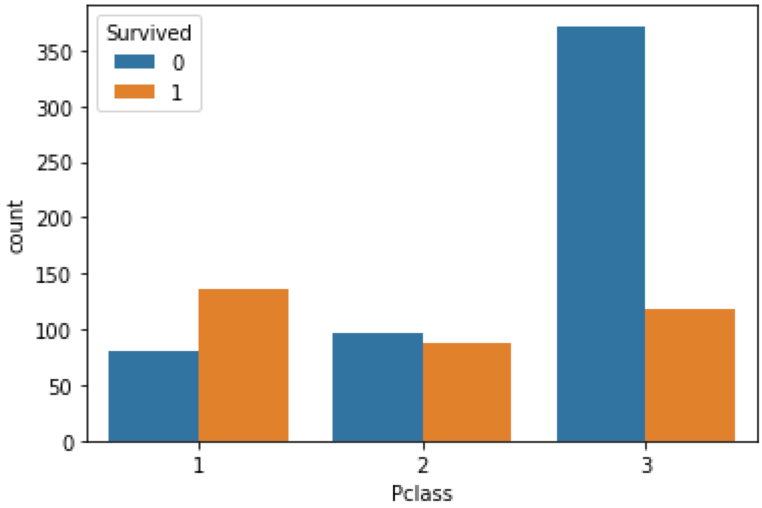
Out[7]: <AxesSubplot:xlabel='Sex', ylabel='count'>



-From the above visualizations it was found that their are more number of male than female and the death to survive ratio is more for male than female

In [8]:
```
sns.countplot(data=df,x="Pclass",hue="Survived")
```

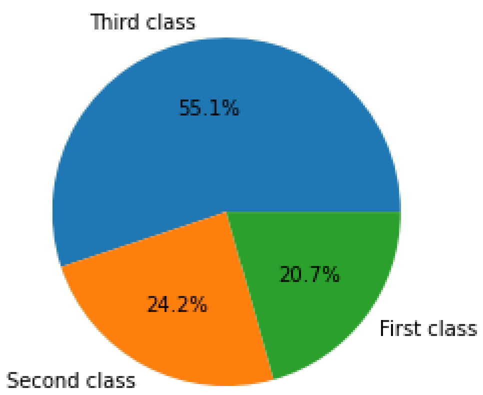
Out[8]: <AxesSubplot:xlabel='Pclass', ylabel='count'>



In [9]:
```
df["Pclass"].value_counts()
```

Out[9]:
```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

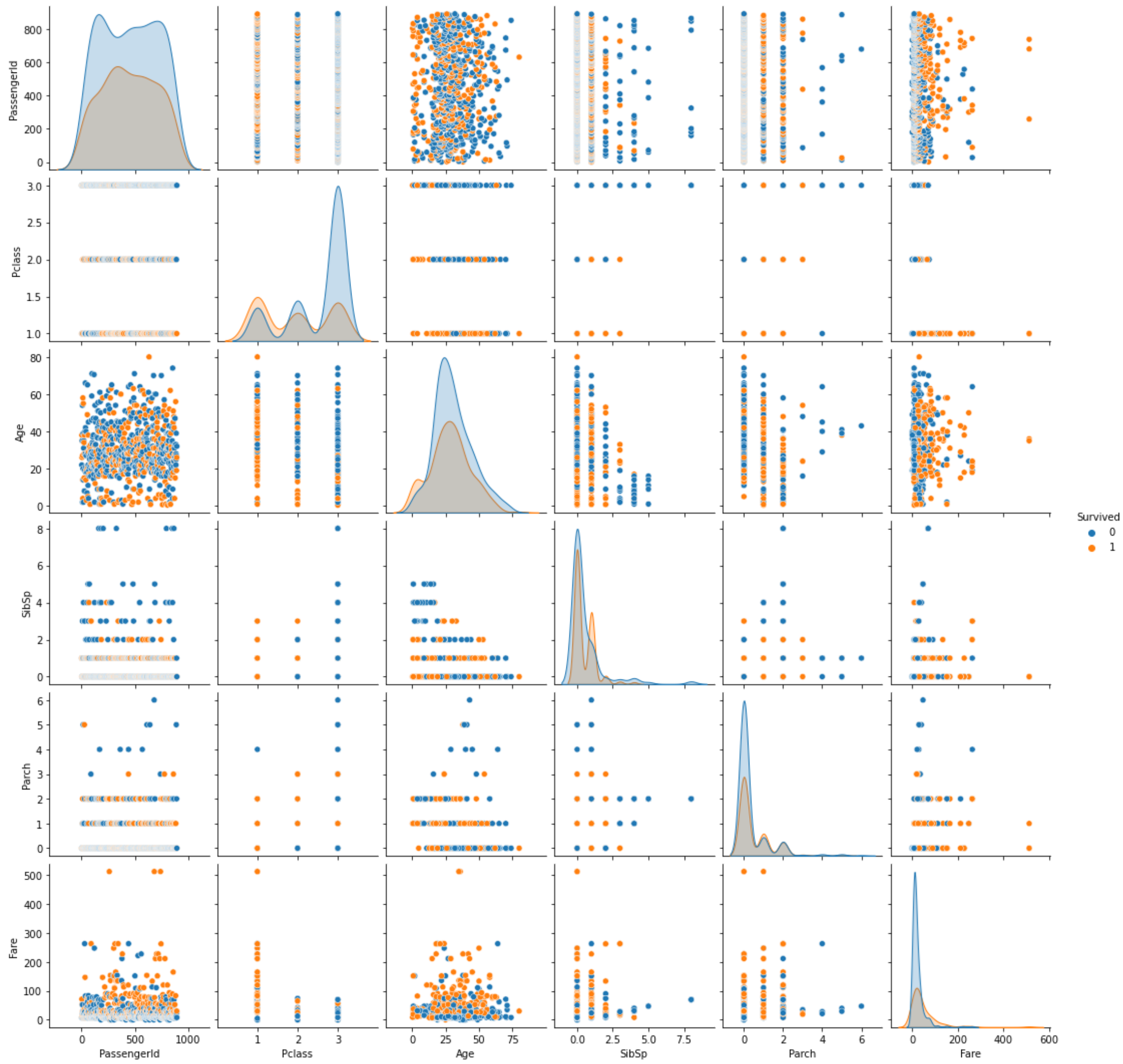
In [10]:
```
plt.pie(df["Pclass"].value_counts(),labels=["Third class","Second class","F
plt.show()
```



-From the above visualizations it is clear that maximum number of people were in Pclass 3 and that death were more in case of Pclass 3 followed by Pclass 2 and Pclass 1 in respetive order

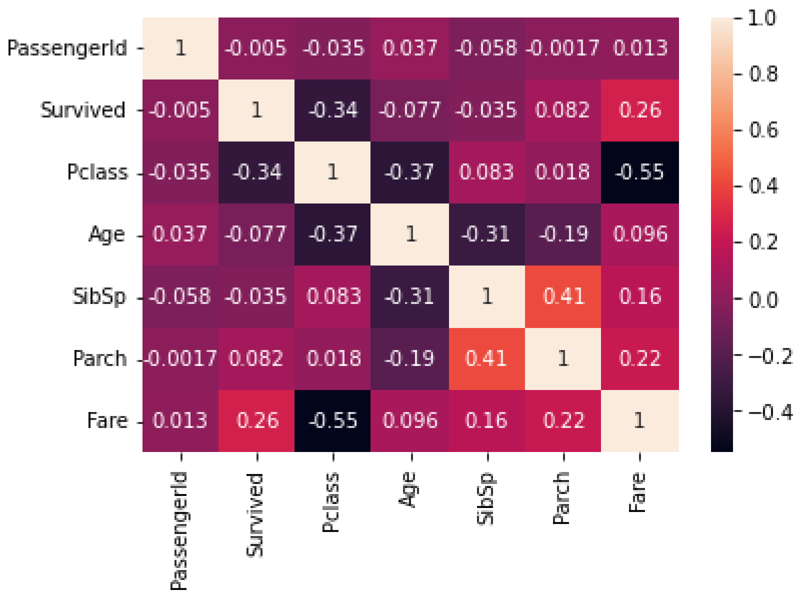
In [11]:
```python
sns.pairplot(df,hue="Survived")
```

Out[11]: <seaborn.axisgrid.PairGrid at 0x29ff30fbca0>



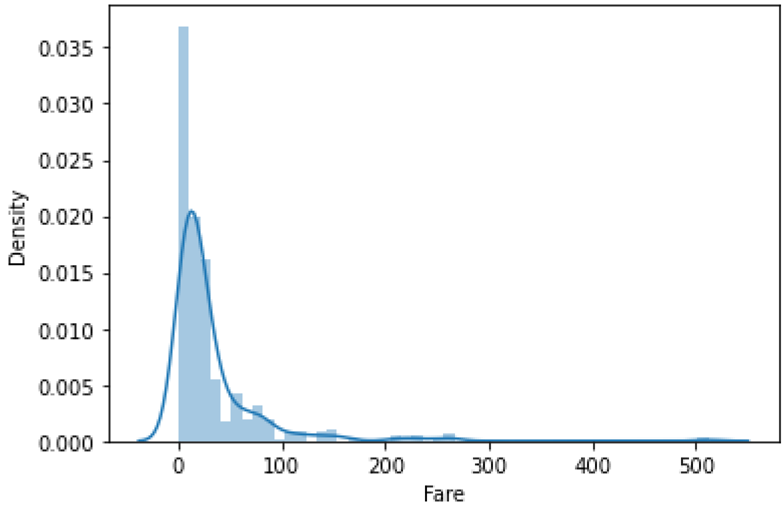
In [12]:
```python
sns.heatmap(df.corr(), annot=True)
```

Out[12]: <AxesSubplot:>



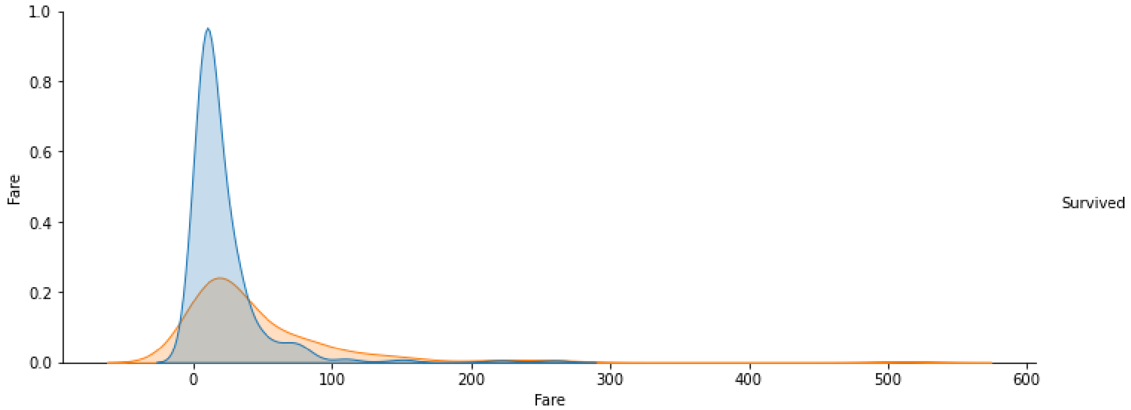
In [13]:
```
sns.distplot(df["Fare"])
```

Out[13]: <AxesSubplot:xlabel='Fare', ylabel='Density'>



In [14]:
```
sns.pairplot(df,x_vars="Fare",y_vars="Fare",height=4,aspect=2.5,hue="Surviv
```

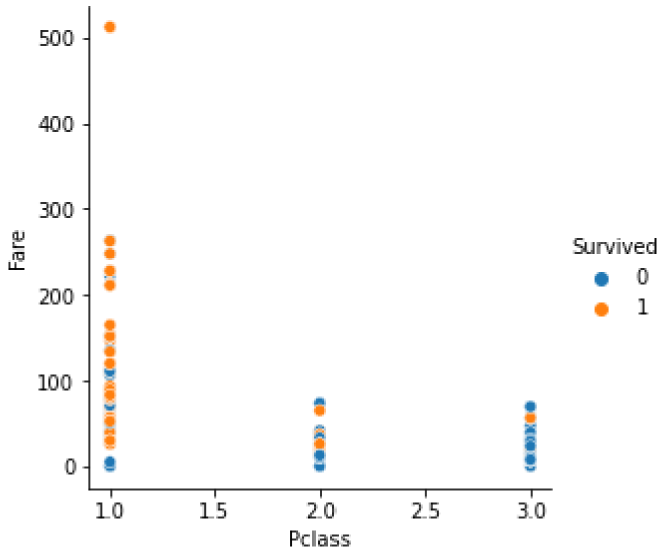
Out[14]: <seaborn.axisgrid.PairGrid at 0x29ff593b850>




```
-From above visualizations it was found that more people were present
with low fare tickets and the probability of survival for low fare ticket
is also very low.
```

In [15]:
```
sns.pairplot(df,x_vars="Pclass",y_vars="Fare",height=4,aspect=1,hue="Surviv
```

Out[15]: <seaborn.axisgrid.PairGrid at 0x29ff5feee50>



-The passengers from Pclass 1 have a good survival probability

Data cleaning, Handling missing values, Preprocessing

In [16]:
```
df.info() #by studying the info we found missing values in age,cabin and em
```

```
<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
```

-As embarked have less than 3% of missing data we will drop the row in which embarked is null (i.e 3 rows in total)
-In case of cabin column more than 30% of data is missing, hence we need to drop the column

-In case of age column less than 30% of data is missing hence we need to fill those null values with average values

In [17]:
```
df.head()
```

Out[17]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
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

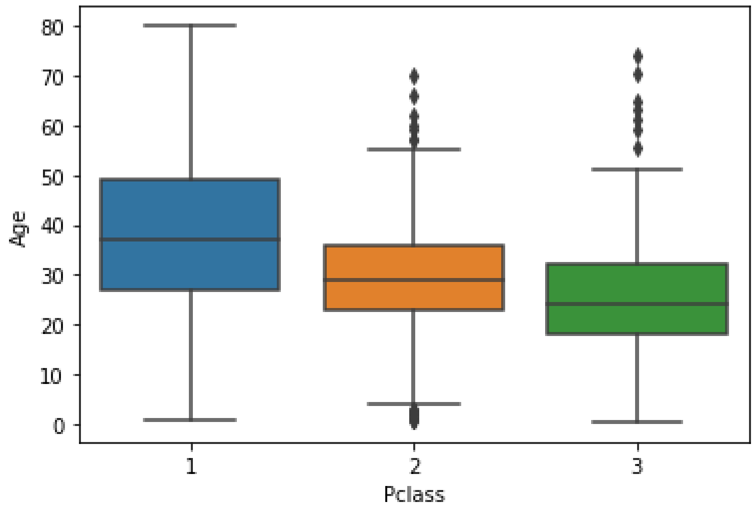
-From the observation we can see that PassengerId,Name,Ticket does not have any use in making the prediction model and needed to be dropped

In [18]:
```
df.drop(['PassengerId','Name','Ticket','Cabin'], axis=1, inplace=True)
```

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
Survived      0
Pclass        0
Sex           0
Age         177
SibSp         0
Parch         0
Fare          0
Embarked      2
dtype: int64
```

In [20]:
```
sns.boxplot(data=df,x="Pclass",y="Age")
plt.show()
```



from above diagram we can see that the mean of age for each Pclass is different and hence for better prediction we need to fill the missing values of Age columns according to their class

In [21]:
```
print("Mean of Pclass 1:- ",df[df["Pclass"]==1]["Age"].mean())
print("Mean of Pclass 2:- ",df[df["Pclass"]==2]["Age"].mean())
print("Mean of Pclass 3:- ",df[df["Pclass"]==3]["Age"].mean())
```

```
Mean of Pclass 1:-  38.233440860215055
Mean of Pclass 2:-  29.87763005780347
Mean of Pclass 3:-  25.14061971830986
```

In [22]:
```
#filling the ages according to their class

def fillage(cols):
    age = cols[0]
    pclass = cols[1]

    if pd.isnull(age):
        if pclass==1:
            return 38
        elif pclass==2:
            return 30
        else:
            return 25
    else:
        return age


df["Age"]=df[["Age","Pclass"]].apply(fillage, axis=1)
```

In [23]:
```
df.isna().sum()
```

Out[23]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    2
dtype: int64
```

In [24]:
```
df.dropna(inplace=True)
```

In [25]:
```
df.isna().sum()
```

Out[25]:
```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

In [26]:
```
df.head()
```

Out[26]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

In [27]:
```
x=df.iloc[:,1:]
y=df.iloc[:,0]
```

In [28]:
```
x
```

Out[28]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	22.0	1	0	7.2500	S
1	1	female	38.0	1	0	71.2833	C
2	3	female	26.0	0	0	7.9250	S
3	1	female	35.0	1	0	53.1000	S
4	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...
886	2	male	27.0	0	0	13.0000	S
887	1	female	19.0	0	0	30.0000	S
888	3	female	25.0	1	2	23.4500	S
889	1	male	26.0	0	0	30.0000	C
890	3	male	32.0	0	0	7.7500	Q

889 rows × 7 columns

In [29]:
```
y
```

Out[29]:
```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 889, dtype: int64
```

In [30]:
```
print(df["Sex"].value_counts())
print()
print(df["Embarked"].value_counts())
```

```
male      577
female    312
Name: Sex, dtype: int64

S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

In [31]:
```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
x[["Sex","Embarked"]]=oe.fit_transform(x[["Sex","Embarked"]])
```

In [32]:
```
x
```

Out[32]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	1.0	22.0	1	0	7.2500	2.0
1	1	0.0	38.0	1	0	71.2833	0.0
2	3	0.0	26.0	0	0	7.9250	2.0
3	1	0.0	35.0	1	0	53.1000	2.0
4	3	1.0	35.0	0	0	8.0500	2.0
...	...	...	...	...	...	...	...
886	2	1.0	27.0	0	0	13.0000	2.0
887	1	0.0	19.0	0	0	30.0000	2.0
888	3	0.0	25.0	1	2	23.4500	2.0
889	1	1.0	26.0	0	0	30.0000	0.0
890	3	1.0	32.0	0	0	7.7500	1.0

889 rows × 7 columns

In [33]:
```
print(x["Sex"].value_counts())
print()
print(x["Embarked"].value_counts())
```

```
1.0    577
0.0    312
Name: Sex, dtype: int64

2.0    644
0.0    168
1.0     77
Name: Embarked, dtype: int64
```

```
after encoding

in sex column:
male = 1.0
female = 0.0

in Embarked column:
S = 2.0
C = 0.0
Q = 1.0
```

Model Building

In [34]:
```
df["Survived"].value_counts()
```

Out[34]:
```
0    549
1    340
Name: Survived, dtype: int64
```

Hence stratify should be used as their is data imbalance

In [35]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state
```

In [36]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(xtrain,ytrain)
ypred = logreg.predict(xtest)
```

In [37]:
```
from sklearn.metrics import accuracy_score, confusion_matrix,classification

ac = accuracy_score(ytest,ypred)
cm = confusion_matrix(ytest,ypred)
cr = classification_report(ytest,ypred)

print(f"Accuracy:- {ac}\n {cm}\n {cr}")
```

```
Accuracy:- 0.8052434456928839
 [[141  24]
 [ 28  74]]
              precision    recall  f1-score   support

           0       0.83      0.85      0.84       165
           1       0.76      0.73      0.74       102

    accuracy                           0.81       267
   macro avg       0.79      0.79      0.79       267
weighted avg       0.80      0.81      0.80       267
```

In [38]:
```
train = logreg.score(xtrain,ytrain)
test = logreg.score(xtest,ytest)

print(f"Training Accuracy:- {train}\n Testing Accuracy:- {test}")
```

```
Training Accuracy:- 0.8038585209003215
 Testing Accuracy:- 0.8052434456928839
```

Forecast New Observation

In [42]:
```
def predictsurvived():
    pclass=int(input("Enter Passenger Class:- "))
    sex = input("Enter Gender Of Passeneger:- ")
    age = int(input("Enter Passeneger Age:- "))
    sibsp = int(input("Enter Sib/Sp Of The Passenger:- "))
    parch = int(input("Enter Parch Of The Passenger:- "))
    fare = int(input('Enter Ticket Price:- '))
    embarked = input("Enter Port Of Embarkation:- ")

    newob = [pclass,sex,age,sibsp,parch,fare,embarked]
    newob[1],newob[-1] = oe.transform([[newob[1], newob[-1]]])[0]
    v = logreg.predict([newob])[0]

    if v==1:
        print("Yes, With The Given Feature the Person Will Survive..!!!")

    else:
        print("No, With The Given Feature the Person Will Not Survive..!!!"
    return v
```

In [43]:
```
predictsurvived()
```

```
Enter Passenger Class:- 1
Enter Gender Of Passeneger:- male
Enter Passeneger Age:- 21
Enter Sib/Sp Of The Passenger:- 0
Enter Parch Of The Passenger:- 0
Enter Ticket Price:- 100
Enter Port Of Embarkation:- S
Yes, With The Given Feature the Person Will Survive..!!!
```

Out[43]: 1

In [ ]: