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

In [2]:
```
titanic_df=pd.read_csv(r"C:\Users\s323\Desktop\Gatherings\Data Science\ML\Amit Misl
```

In [3]:
```
titanic_df.head()
```

Out[3]:

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

Embarked - Southampton(S) [UK] on 10 April 1912, Titanic called at Cherbourg(C) in France and Queenstown (Q) in Ireland, before heading west to New York. These are three places where ship was halted to carry the passenger

Data Wrangling

In [4]:
```
titanic_df.isnull().sum()
```

Out[4]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [5]:
```
titanic_df.shape
```

Out[5]:
```
(891, 12)
```

Cabin is of not use because only first class had cabin so we can easily remove them

In [6]:
```
titanic_df.columns
```

Out[6]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [7]:
```
titanic_df.dtypes
```

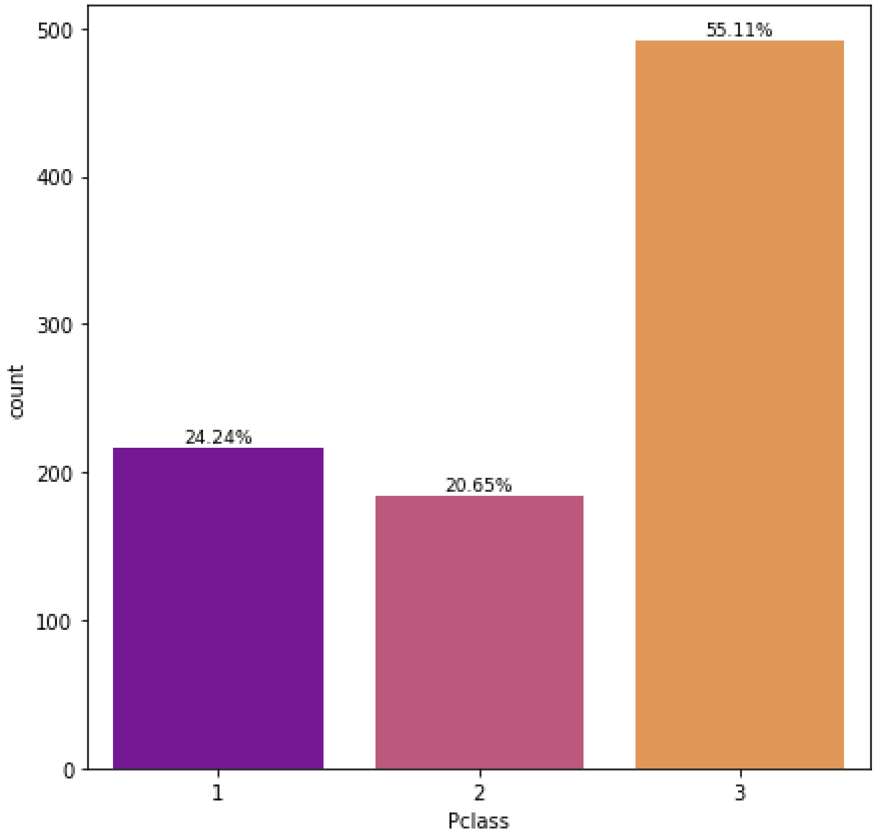
Out[7]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

EDA

- Most of the passengers are in which class?
- To check this best is count plot

In [8]:
```
plt.figure(figsize = (7,7))
fig = sns.countplot(x="Pclass",data=titanic_df,palette="plasma")
sizes=[]
# sometimes people are intrested in showing labels
# patches will return hieght and width
for p in fig.patches:
    height = p.get_height()
    sizes.append(height)
    # text will display or add the text on the plot
    # p.get_x-  return the label like 1,2,3.
    # p.get_widthwidth of bar graph
    # hieght of the plot and 4 is just a value
    # value part last column, ha - test alignment in the center
```

```
    # x axis together had label and width and y value had hieght and it is used to
    # width/2 is used for keeping the value in center
    #.format to print the value to give round off
    # height is returning you value count
    fig.text(p.get_x() + p.get_width()/2, height + 4,
            '{:1.2f}%'.format(height/len(titanic_df)* 100), ha = 'center', fontsiz
```



To find percentage we can either do

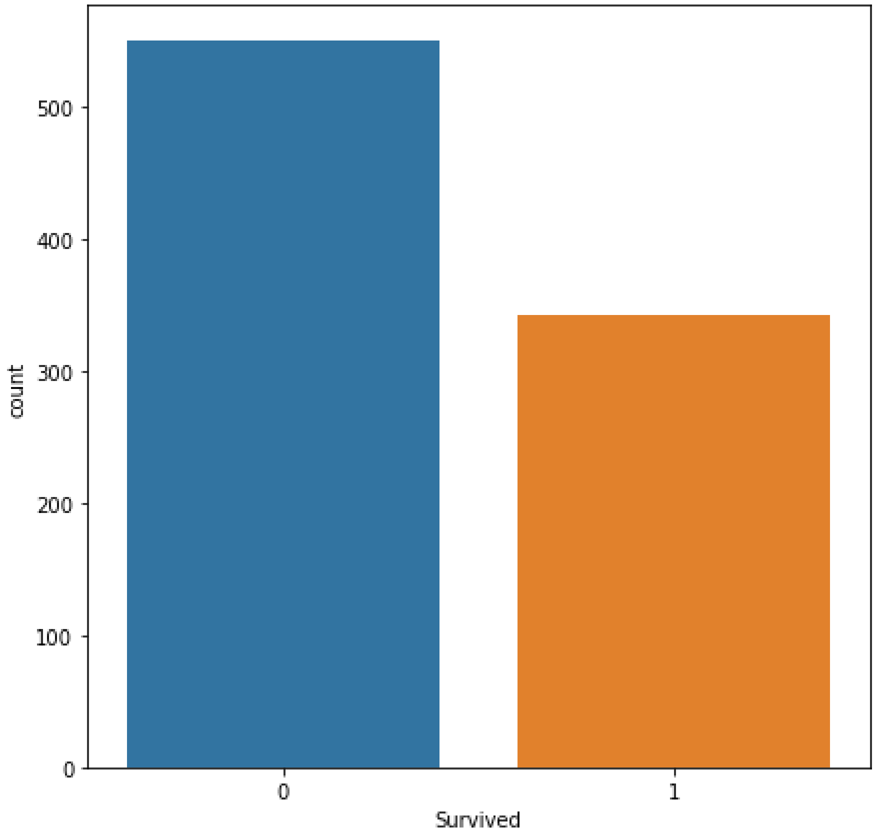
In [9]:
```
# short cut part
round(titanic_df["Pclass"].value_counts()/len(titanic_df),2)
```

Out[9]:
```
3    0.55
1    0.24
2    0.21
Name: Pclass, dtype: float64
```

distribution for survived or not

In [10]:
```
plt.figure(figsize=(7,7))
# sns.countplot(x=["Survived"],data=data=titanic_df,palette="plasma")
sns.countplot(x = 'Survived', data = titanic_df)
```

Out[10]: <AxesSubplot:xlabel='Survived', ylabel='count'>



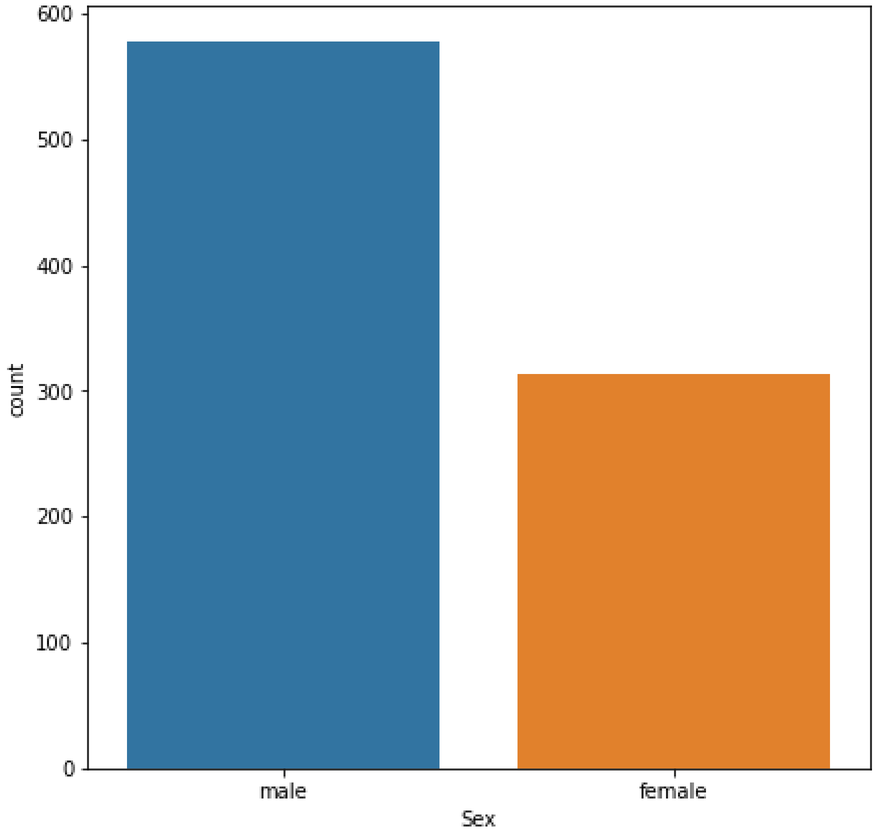
In [11]:
```
# short cut part, using value count
round(titanic_df["Survived"].value_counts()/len(titanic_df),2)
```

Out[11]:
```
0    0.62
1    0.38
Name: Survived, dtype: float64
```

Distribution of Gender

In [12]:
```
plt.figure(figsize=(7,7))
sns.countplot(x="Sex", data=titanic_df)
```

Out[12]:
```
<AxesSubplot:xlabel='Sex', ylabel='count'>
```



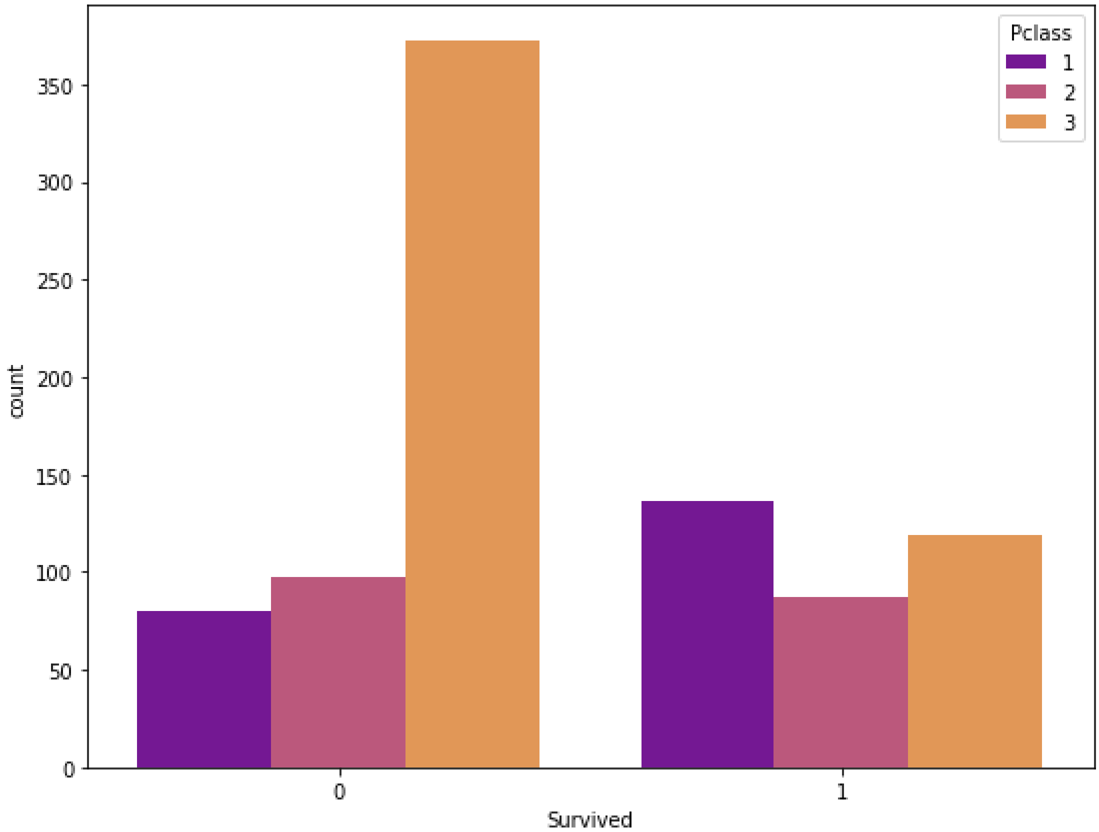
In [13]:
```
round(titanic_df["Sex"].value_counts()/len(titanic_df),2)
```

Out[13]:
```
male      0.65
female    0.35
Name: Sex, dtype: float64
```

Catplot

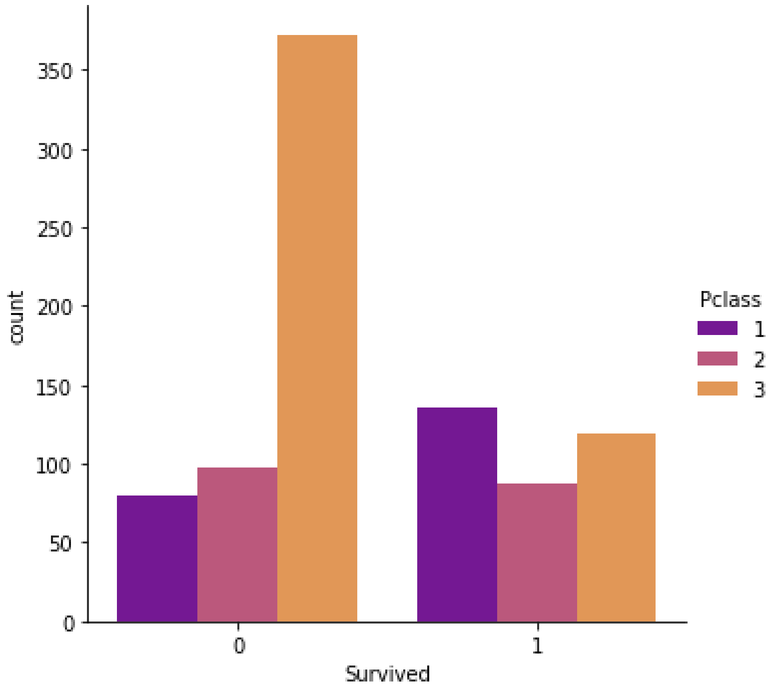
In [14]:
```
plt.figure(figsize=(9,7))
sns.countplot(x="Survived", data= titanic_df, palette="plasma", hue="Pclass")
# hue is used for cateogries
```

Out[14]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



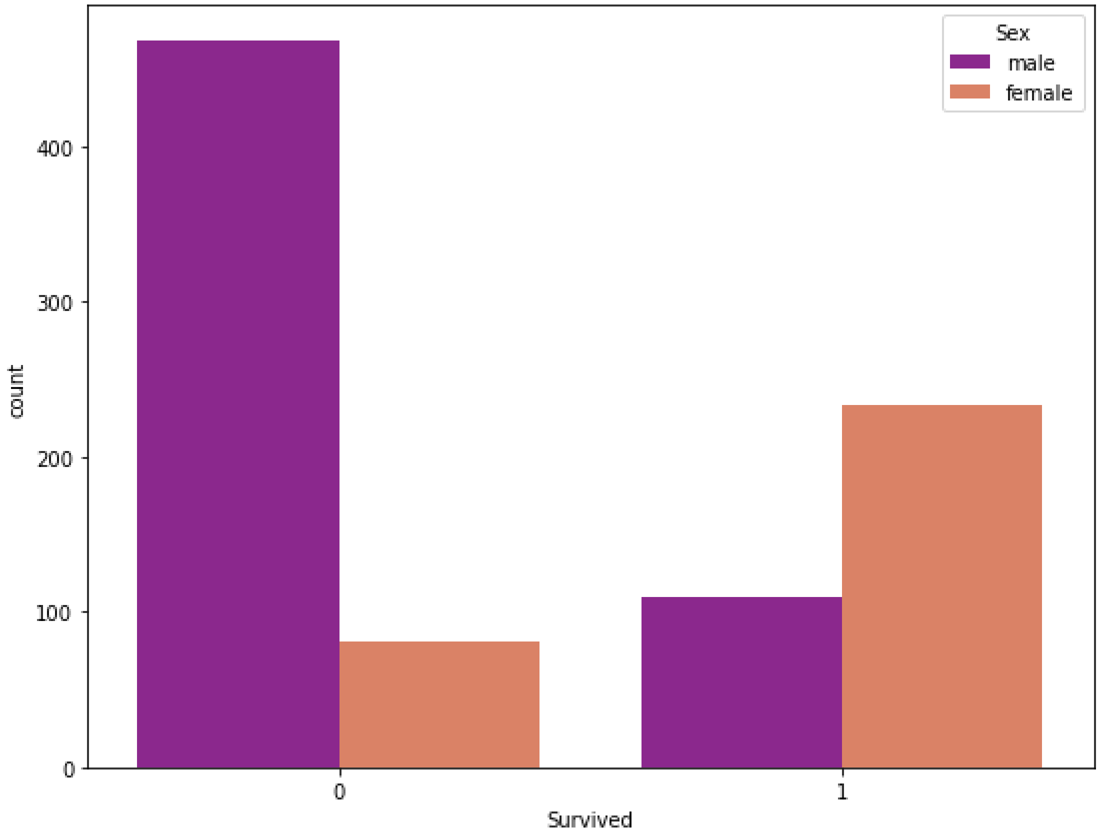
In [15]:
```
### or,
sns.catplot(x = 'Survived', kind = 'count', hue = 'Pclass', data = titanic_df, pal
```

Out[15]: <seaborn.axisgrid.FacetGrid at 0x16a8c153ca0>



In [16]:
```
# Most of the passenger survived are Feamle
plt.figure(figsize=(9,7))
sns.countplot(x="Survived", data=titanic_df, palette= "plasma", hue="Sex")
```

Out[16]: <AxesSubplot:xlabel='Survived', ylabel='count'>



Create a contingency table - It gives us nice frequency table

In [17]:
```
# crosstable

contingency_table=pd.crosstab(titanic_df["Pclass"], titanic_df["Survived"])
contingency_table
```

Out[17]:

Survived	0	1
Pclass		
1	80	136
2	97	87
3	372	119

In [18]:
```
### Crosstable in percentage of survived or not survived

contingency_table = pd.crosstab(titanic_df['Pclass'], titanic_df['Survived']) / le
contingency_table
```

Out[18]:

Survived	0	1
Pclass		
1	8.978676	15.263749
2	10.886644	9.764310
3	41.750842	13.355780

Handling Missing Values

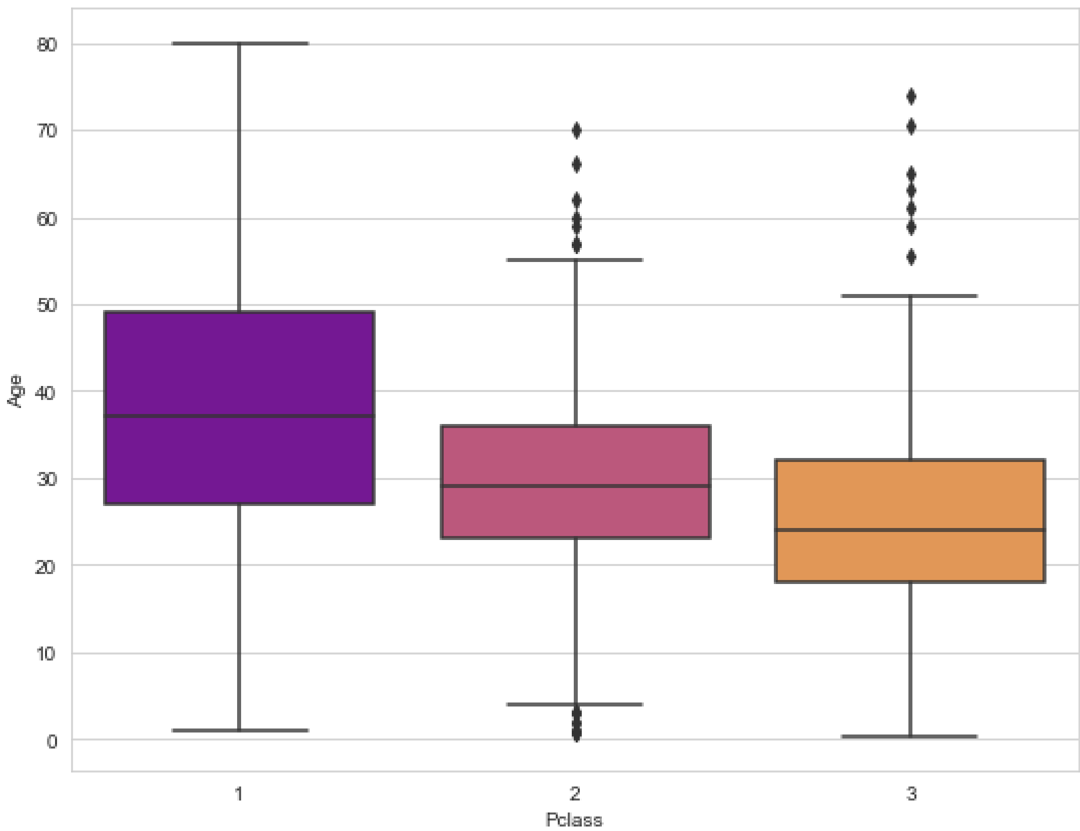
In [19]:
```
titanic_df.isnull().sum()
```

Out[19]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [20]:
```
# there are two ways to adjust missing values in age
# 1. Median age of all classes of passengers and replace it
# but we will do the best way... specially when name is given and age is missing
```

In [21]:
```
sns.set_style("whitegrid")
plt.figure(figsize=(9,7))
sns.boxplot(x="Pclass",y="Age",data=titanic_df,palette="plasma")
```

Out[21]:
```
<AxesSubplot:xlabel='Pclass', ylabel='Age'>
```



we can see that passenger class had outliers, so can't do mean... so we always prefer median or mode over mean, because Median got unaffected from outliers while mean gets highly affected

In [22]:
```
# Alternative approach to Replace Missing Age by Title of Name
titanic_df['Title'] = titanic_df.Name.str.extract(' ([A-Za-z]+)\.', expand=False)
```

In [23]:
```
titanic_df.Name.str.extract('([A-Za-z]+)\.', expand=False)
# to understand
```

Out[23]:
```
0        Mr
1       Mrs
2      Miss
3       Mrs
4        Mr
       ...
886     Rev
887    Miss
888    Miss
889      Mr
890      Mr
Name: Name, Length: 891, dtype: object
```

In [24]:
```
titanic_df['Title'].unique()
```

Out[24]:
```
array(['Mr', 'Mrs', 'Miss', 'Master', 'Don', 'Rev', 'Dr', 'Mme', 'Ms',
       'Major', 'Lady', 'Sir', 'Mlle', 'Col', 'Capt', 'Countess',
       'Jonkheer'], dtype=object)
```

In [25]:
```
# cross table is used to understand the 2 diffrent cateogorical data
pd.crosstab(titanic_df['Title'], titanic_df['Sex'])
```

Out[25]:

Sex Title	female	male
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

In [26]:
```
titanic_df['Title'] = titanic_df['Title'].replace(['Don','Rev','Dr','Major','Lady'
                                                 'Rare')
```

In [27]:
```
titanic_df['Title'] = titanic_df['Title'].replace('Mlle','Miss')
titanic_df['Title'] = titanic_df['Title'].replace('Ms','Miss')
titanic_df['Title'] = titanic_df['Title'].replace('Mme','Mrs')
```

In [28]:
```
titanic_df[['Title','Age']].groupby(['Title']).median()
```

Out[28]:

Title	Age
Master	3.5
Miss	21.0
Mr	30.0
Mrs	35.0
Rare	48.5

In [29]:
```
titanic_df
```

Out[29]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
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	N
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	N
...	...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	N
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	I
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	N
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	N

891 rows × 13 columns

UDF - Simple Imputer

- Replacing missing value with median

In [30]:

```
#def imputer_age(age,title) or as a single argument
# if my age is missing
def imputer_age(cols):
    Age = cols[0]
```

```
    Title = cols[1]
    if pd.isnull(Age):
        if Title == 'Master':
            return 3.5
        elif Title == 'Miss':
            return 21
        elif Title == 'Mr':
            return 30
        elif Title == 'Mrs':
            return 35
        else:
            return 48.5
    else:
        return Age
```

In [31]:
```
titanic_df['Age'] = titanic_df[['Age','Title']].apply(imputer_age, axis = 1)
# applying function against the dataframe
```

In [32]:
```
titanic_df.isnull().sum()
```

Out[32]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
Title            0
dtype: int64
```

In [33]:
```
# Drop Cabin
titanic_df.drop('Cabin', axis = 1, inplace=True)
```

In [34]:
```
# It will drop two rows of Embardked Column Missing
titanic_df.dropna(inplace=True)
```

In [35]:
```
titanic_df.isnull().sum()
```

Out[35]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
Title          0
dtype: int64
```

In [36]:
```
titanic_df.shape
```

Out[36]:
```
(889, 12)
```

Median is used for numerical data and mode is used for cateogrical data

Data Preprocessing

In [37]:
```
# As this data had used lot of categorical  datas- will do one hot encoding

# How to handle cateorical data set
## Replace
## Impute
### Encode - Label encoder (for ordinal datas) , One hot encoding (and it uses- .ge
```

In [38]:
```
sex= pd.get_dummies(titanic_df["Sex"],drop_first=True)
sex
```

Out[38]:

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

889 rows × 1 columns

In [39]:
```
Embarked=pd.get_dummies(titanic_df["Embarked"],drop_first=True)
Embarked
```

Out[39]:

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

889 rows × 2 columns

In [40]:
```
Title=pd.get_dummies(titanic_df["Title"],drop_first=True)
Title
```

Out[40]:

	Miss	Mr	Mrs	Rare
0	0	1	0	0
1	0	0	1	0
2	1	0	0	0
3	0	0	1	0
4	0	1	0	0
...	...	...	...	...
886	0	0	0	1
887	1	0	0	0
888	1	0	0	0
889	0	1	0	0
890	0	1	0	0

889 rows × 4 columns

In [41]:
```
#titanic_df1=pd.concat([titanic_df, sex, embarked, title],axis=1)
```

In [42]:
```
titanic_df1 = pd.concat([titanic_df, sex, Embarked, Title], axis = 1)
titanic_df1
```

Out[42]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Em
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	21.0	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	

889 rows × 19 columns

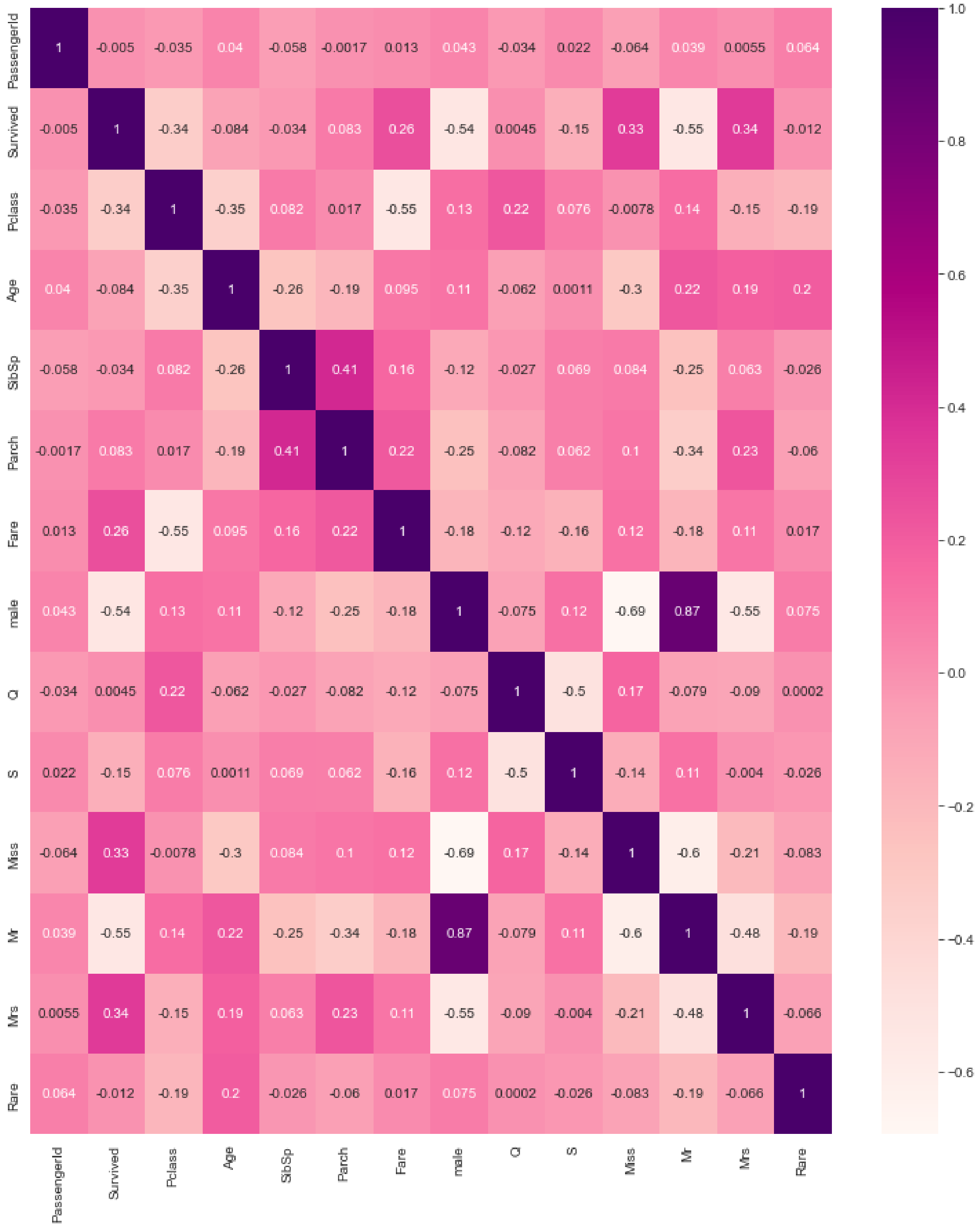
In [43]:

```
# no of columns which will not be helpful for understanding the survived patient, w
#1. Passenger id, 2.Name, 3.Ticket, 4. Title, 5.Sex 6.Parch
#Name is never used- nominal type of data is always dropped
# MLv never uses cateogrical data
# Label encoder- label encoding such as 1,2 is given if we have ordinal data
```

In [44]:

```
#### Corelation is selection for feature, but here data is not contious data
### features are selected on basis of corerelation, here nature of target column is
### Survived isn't continious data, only in regression not in classification where
```

In [45]:
```
plt.figure(figsize=(13,15))
sns.heatmap(titanic_df1.corr(), annot=True, cmap='RdPu')
plt.show()
```



In [46]:
```
###(Mr, male),( Male, mr), (Parch, Q)###
```

Features and Target

In [60]:
```
#X = titanic_df1.drop(['PassengerId', 'Survived', 'Name', 'Sex', 'Ticket', 'Embarke
#Y = titanic_df1['Survived']

X = titanic_df1.drop(["PassengerId", 'Survived', 'Name', 'Sex', 'Ticket', 'Embarke
Y = titanic_df1["Survived"]
```

Cross Validation for weak feature selection

- Splitting data into train and test data samples

In [61]:
```
#from sklearn.model_selection import train_test_split
#x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size =0.2, random_state=1)
```

Logistic Regression - it is used at classification

In [71]:
```
#from sklearn.linear_model import LogisticRegression
#logit_model = LogisticRegression(solver = 'lbfgs', C = 1e5, max_iter=1e7, penalty=
# to reduce the optimize errors, by adding hyperparamter- such as max_iter,solver -
from sklearn.linear_model import LogisticRegression
logit_model=LogisticRegression(max_iter=1e7,solver="liblinear",penalty="l2")
# we can learn all those by shift+tab
```

In [72]:
```
logit_model.fit(x_train,y_train)
```

Out[72]:
```
LogisticRegression(max_iter=10000000.0, solver='liblinear')
```

In [73]:
```
# Accuracy
logit_model.score(x_test, y_test)
```

Out[73]:
```
0.8539325842696629
```

Classification Metrics

- Confusion Matrix - that shows us the result btw actual prediction and correct prediction
- Classification Report - it provides us over all report of the classification,such as decision, recall

In [75]:
```
# How classification report is generated and for other classification regression w
# first we will create our prediction value for confusion matrix
# predicted output for the given test value,
# other name for predicted value is yhat
# for predictions we will use, .predict function
predictions=logit_model.predict(x_test)
```

In [76]:
```
predictions
```

Out[76]:
```
array([1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0], dtype=int64)
```

1 means person had survived and 0 means he is dead

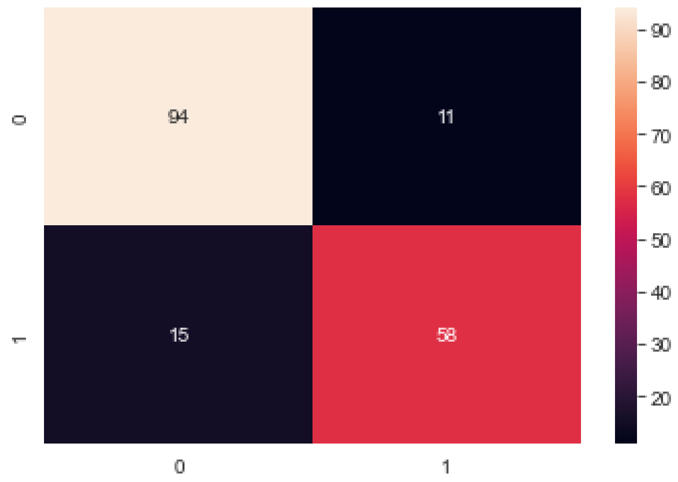
Now prediciction for the unknown data, untrained dataset

In [77]:
```
from sklearn.metrics import confusion_matrix, classification_report
confusion_matrix(y_test, predictions)
```

Out[77]:
```
array([[94, 11],
       [15, 58]], dtype=int64)
```

In [81]:
```
# Now we reprsent with a heat map
#1st result, 2nd result classification report
sns.heatmap(confusion_matrix(y_test, predictions),annot=True,fmt="0.0f")
```

Out[81]:
```
<AxesSubplot:>
```



Out of 105, 94 were correct and 11 false- class 0 , class 1 - 58 correct and 15 incorrect in 73 samples

In [82]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       105
           1       0.84      0.79      0.82        73

    accuracy                           0.85       178
   macro avg       0.85      0.84      0.85       178
weighted avg       0.85      0.85      0.85       178
```

Make New Predictions for train data - for remaining samples apart from 178 samples which is test data

In [83]:
```
X
```

Out[83]:

	Pclass	Age	SibSp	Parch	Fare	male	Q	S	Miss	Mr	Mrs	Rare
0	3	22.0	1	0	7.2500	1	0	1	0	1	0	0
1	1	38.0	1	0	71.2833	0	0	0	0	0	1	0
2	3	26.0	0	0	7.9250	0	0	1	1	0	0	0
3	1	35.0	1	0	53.1000	0	0	1	0	0	1	0
4	3	35.0	0	0	8.0500	1	0	1	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
886	2	27.0	0	0	13.0000	1	0	1	0	0	0	1
887	1	19.0	0	0	30.0000	0	0	1	1	0	0	0
888	3	21.0	1	2	23.4500	0	0	1	1	0	0	0
889	1	26.0	0	0	30.0000	1	0	0	0	1	0	0
890	3	32.0	0	0	7.7500	1	1	0	0	1	0	0

889 rows × 12 columns

In [84]:

```
# copy one row data from the above
x_jack = [[3, 22.0, 0, 0, 7.920, 1, 0, 1, 0, 1, 0, 0]]
x_rose = [[1, 24.0, 1, 2, 72.920, 0, 0, 1, 1, 0, 0, 0]]
```

In [85]:

```
logit_model.predict(x_jack)
```

Out[85]:

```
array([0], dtype=int64)
```

In [86]:

```
logit_model.predict(x_rose)
```

Out[86]:

```
array([1], dtype=int64)
```

In [ ]: