Titanic Data Set

Problem Statement- we will be working with the Titanic Data Set from HYPERLINK "https://www.kaggle.com/c/titanic"Kaggle. This is a very famous data set. We'll be trying to predict a classification- survival or deceased

URL to get Dataset: https://www.kaggle.com/c/titanic

Method: Logistic Regression/source of Data: Udemy/Tool: Python

Library: - **Pandas, Matplotlib, sklearn, seaborn, Numpy**

Dataset: head of dataset is as below and Target Colum**: survival**

```
In [5]: train.head()
```

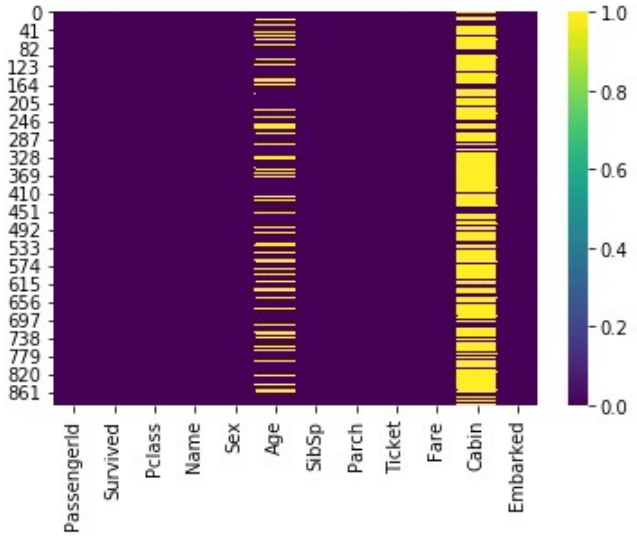
Out[5]:

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

Exploratory Data Analysis: Let's begin some exploratory data analysis! We'll start by checking out missing data! We can use seaborn to create a simple heatmap to see where we are missing data!

```
sns.heatmap(data=train.isnull(),linecolor='y',cbar="False",cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x188219eea20>
```

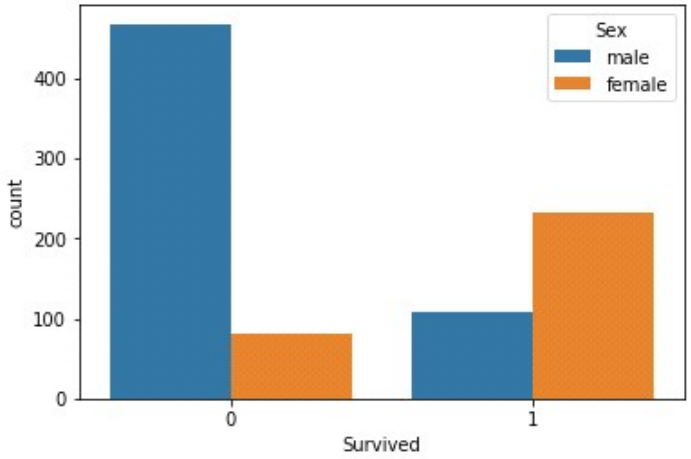


Observation: Roughly 20 percent of the Age data is missing. The proportion of Age missing is likely small enough for reasonable replacement with some form of imputation. Looking at the Cabin column, it looks like we are just missing too much of that data to do something useful with at a basic level. We'll probably drop this later, or change it to another feature like "Cabin Known: 1 or 0"

Examination of Target Variable (to check survived or not)

```
sns.countplot(x='Survived',data=train, hue='Sex')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18821c3c6d8>
```

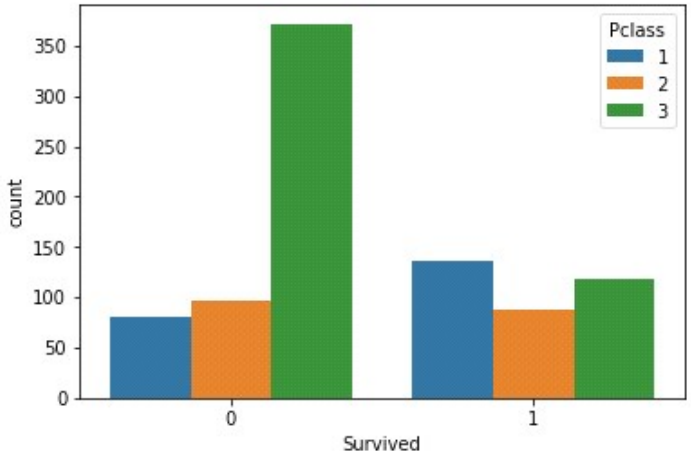


Observation: we can see most of them have been not survived and few of them have survived and with hue, most of the male has not survived in comparison to women.

Examination of Passenger class wise survived

```
sns.countplot(data=train,x='Survived',hue='Pclass')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18821c80080>
```

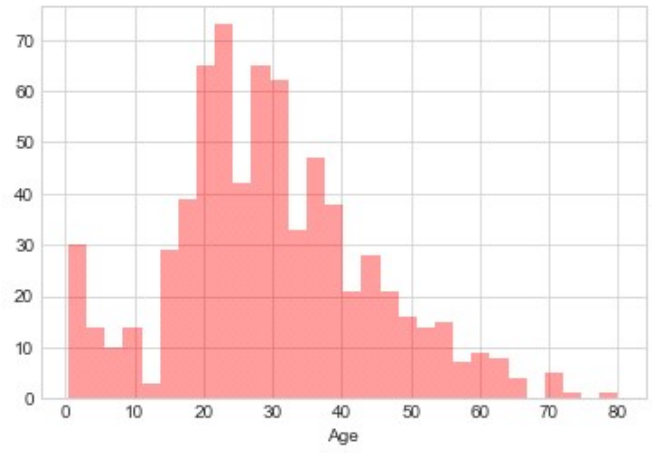


Observation: as we can see into survived vs. passenger class, most of them from class 1st have survived rather than class 3rd passenger.

Examination of Passenger Age on board on titanic

```
sns.distplot(train['Age'].dropna(),bins=30,kde=False,color='red')
sns.set_style('whitegrid')
```

```
C:\Users\INDRAJEET YADAV\Anaconda3\lib\site-packages\matplotlib\a
ed, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



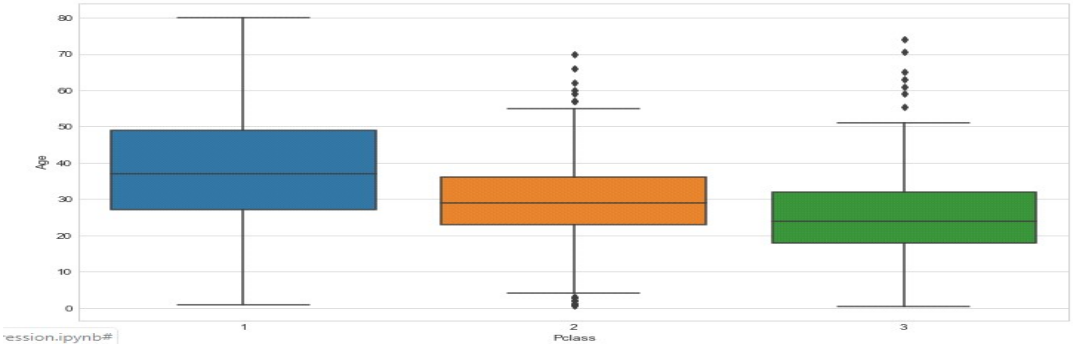
Observation:- easily can tell, most of them were very young who on boarded titanic ship

Data Cleaning:

We want to fill in missing age data instead of just dropping the missing age data rows. One way to do this is by filling in the mean age of all the passengers (imputation). However we can be smarter about this and check the average age by passenger class.

```
plt.figure(figsize=(12,7))
sns.boxplot(x='Pclass',y='Age',data=train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18821e21a58>
```



Observation: We can see the wealthier passengers in the higher classes tend to be older, which makes sense. We'll use these average age values to impute based on Pclass for Age.

Replace Age‘s nil value with average of Passenger class value

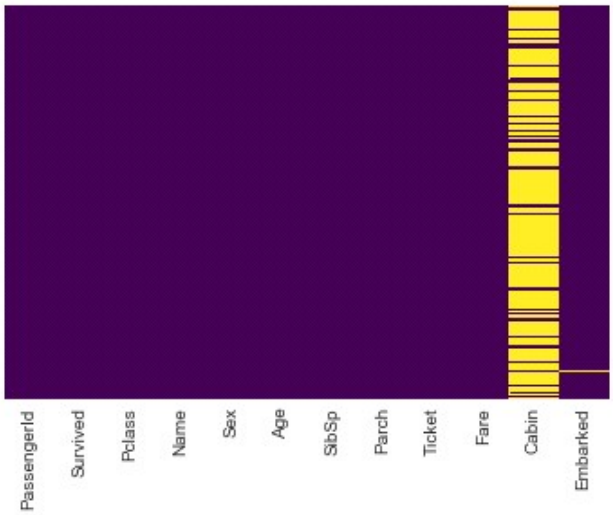
```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

Now apply that function!

```
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x22bcddda630>
```



Observation: Now data looks clean since remain cabin columns have lot of noise so better to leave same columns.

Converting Categorical Features:

We'll need to convert categorical features to dummy variables using pandas! Otherwise our machine learning algorithm won't be able to directly take in those features as inputs

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

```
pd.concat([train,sex,embark],axis=1)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	male	Q	S
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	1	0	1

we have new dummy colums for sex as male, and Q,S for embarked so will drop extra columns .

will also drop out other feature also like Sex,Embarked,Name,Ticket becasue there is no use in logistic regression due to non convertable variables

```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	1	0	3	22.0	1	0	7.2500
1	2	1	1	38.0	1	0	71.2833

after this my data set looks better to use logistic regression. lets build model

Building a Logistic Regression model

Let's start by splitting our data into a training set and test set (there is another test.csv file that you can play around with in case you want to use all this data for training).

Train Test Split: we need to split data in training and test.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived', axis=1),train['Survived'], test_size=0.3, random_state=101)

Training and Predicting

```
from sklearn.linear_model import LogisticRegression
```

```
lm=LogisticRegression()
```

```
lm.fit(X_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
prediction=lm.predict(X_test)
```

Evaluation: will use **confusion matrix** to evaluate model.

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test,prediction)
```

```
array([[111,  17],
       [ 42,  44]], dtype=int64)
```

Final observation:- we can see clearly predicted vs reality result with the help of confusion matrix since type 1 error 17 and type 2 error is 42.