Logistic Regression Case Study On Titanic Dataset

Logistic regression is a supervised machine learning algorithm that is applicable only when we have target variable that contains binary values such as true/false, 0/1 ,yes/no and so on.Unlike,linear regression algorithm it provides the sigmoid curve of values that ranges in between 0 and 1,these values will never exceeds this range of 0-1 and hence provide the discrete sets of values.If I say the basic margin is of 0.5 or my threshold value is 0.5 than the values less than 0.5 will be predicted as 0 whereas the values greater than 0.5 will be predicted as 1 only.

Titanic Dataset Introduction

On 15 April 1912,the british's titanic sank in the atlantic ocean and around 2,224 passengers suffered huge pandemic of sinking and 1,500 approx people died,during its journey from Southampton to New York City.

Importing Required packages/Libraries

```
#importing warnings
import warnings
warnings.filterwarnings('ignore')
```

```
#importing other required libraries/packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Reading the data

we have two dataset: train_dataset =training model & test_dataset = testing model

```
#Reading train data as well as test data
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
```

we are going to use train_data first to do all the predictions and analysis and then will go for test data to check whether our model is capable for purely new data or not via comparing the accuracy score in both cases.

Data Visualisation

```
#using head to have data vision
train_data.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques	female	35.0	1	0	113803	53.100

```
#using shape to see the no. of rows and no.columns in the given data
train_data.shape
```

```
(891, 12)
```

Data Understanding

```
#using describe to analyse the data
train_data.describe()
```

PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare

```
#using info to analyse categorical &continous variables
train_data.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
```

```
#checking continous variables and assigning them to the variable var
var = train_data[['PassengerId','Survived','Pclass','Age','SibSp','Parch','Fare']]
```

```
#checking percentiles for all the continous variables
var.describe(percentiles=[.25,.5,.75,.90,.95,.99])
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
90%	802.000000	1.000000	3.000000	50.000000	1.000000	2.000000	77.958300
95%	846.500000	1.000000	3.000000	56.000000	3.000000	2.000000	112.079150
99%	882.100000	1.000000	3.000000	65.870000	5.000000	4.000000	249.006220
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Treating Missing Values

```
#checking sum of missing values group by columns
train_data.isnull().sum()
```

```
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
```

```
#checking percentage rate for null/missing values
(train_data.isnull().sum()/len(train_data.index))*100
```

```
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
```

```
#checking value count for embarked
train_data['Embarked'].value_counts()
```

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

```
#filling the most common value inplace of null
common_value = 'S'
for i in train_data:
    train_data['Embarked']=train_data['Embarked'].fillna(common_value)
```

```
#again checking embarked
train_data.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         0
dtype: int64
```

```
#adding all the ages
total=train_data['Age'].sum()
```

```
#assinging the avaerage age to the common age variable
common_age=total/891
```

```
#filling common age inplace of null /missing values
for i in train_data:
    train_data['Age'] = train_data['Age'].fillna(common_age)
```

```
#again checking for null values in age
train_data['Age'].isnull().sum()
```

```
0
```

```
#checking  remaining missing values in dataset
train_data.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         0
dtype: int64
```

Feature Engineering

As,we are seeing here the column 'Cabin' contains maximum missing values approx(70%)so,it's better to drop this column but here I can see sum of the info is given in the cabin values that may be useful.Hence, I am gonna add new feature to my model as deck by extracting the charcter deck value from cabin and use deck as my new column inplace of Cabin and then drop cabin which is of no use.

```
#extracting the deck values from cabin values using re library
import re
deck = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7, "U": 8}
data = [train_data, test_data]

for dataset in data:
    dataset['Cabin'] = dataset['Cabin'].fillna("U0")
    dataset['Deck'] = dataset['Cabin'].map(lambda x: re.compile("([a-zA-Z]+)").search(x).grou
    dataset['Deck'] = dataset['Deck'].map(deck)
    dataset['Deck'] = dataset['Deck'].fillna(0)
    dataset['Deck'] = dataset['Deck'].astype(int)
#dropping the cabin feature
train_data = train_data.drop(['Cabin'], axis=1)
```

```
#checking for null/missing values in dataset
train_data.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
Deck           0
dtype: int64
```

Finally! We have no more missing value in our data.

Dividing dataset into train_test split

```
#importing train test split from sklearn model selection to split the datset into two sets fo
from sklearn.model_selection import train_test_split
```

Segregating the independent and dependent variables

```
#dropping dependent variable from train data and assign that dataset to x
x = train_data.drop(['Survived','PassengerId'],axis=1)
x.head(10)
```

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Deck
0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	S	8
1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	C	3
2	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	S	8
3	1	Futrelle, Mrs. Jacques	female	35.000000	1	0	113803	53.1000	S	3

```
#assingning all the dependent values of dependent target variable y to y.
y = train_data['Survived']
y.head(10)
```

```
0    0
1    1
2    1
3    1
4    0
5    0
6    0
7    0
8    1
9    1
Name: Survived, dtype: int64
```

Variable Transformation

In this step,we are going to scale all the variable roughly to the same scale for better accuracy and results.I am using Standard Scaler class for the same.before using it,we have to import the same from sklearn.preprocessing.

```
#to scale the data on roughly same scale,importing standardscaler from sklearn preprocessing.
from sklearn.preprocessing import StandardScaler
```

```
#fitting variable and performing standard scale transformation.
scaler = StandardScaler()
x[['Pclass','Age','SibSp','Parch','Fare','Deck']] = scaler.fit_transform(x[['Pclass','Age','S
x.head(10)
```

	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embar
0	0.827377	Braund, Mr. Owen Harris	male	-0.494245	0.432793	-0.473674	A/5 21171	-0.502445	
1	-1.566107	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	0.717307	0.432793	-0.473674	PC 17599	0.786845	
2	0.827377	Heikkinen, Miss. Laina	female	-0.191357	-0.474545	-0.473674	STON/O2. 3101282	-0.488854	
3	-1.566107	Futrelle, Mrs. Jacques	female	0.490141	0.432793	-0.473674	113803	0.420730	

```
#checking the total surviving rate
survived_rate = (sum(train_data['Survived'])/len(train_data['Survived'].index))*100
survived_rate
```

```
38.38383838383838
```

```
#analysing data correlations via heatmap
plt.figure(figsize = (25,10))
sns.heatmap(train_data.corr(),annot = True)
plt.show()
```

Converting categorical variable to Numerical value

```
train_data.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques	female	35.000000	1	0	113803

```
#converting male to 0 & female to 1
genders = {'male': 0, 'female': 1}
train_data['Sex'] = train_data['Sex'].map(genders)

#converting all the ports to numeral values 0,1,2
ports = {'S':0,'C':1,'Q':2}
train_data['Embarked'] = train_data['Embarked'].map(ports)

train_data.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	0	22.000000	1	0	A/5 21171	7.:
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.000000	1	0	PC 17599	71.:
2	3	1	3	Heikkinen, Miss. Laina	1	26.000000	0	0	STON/O2. 3101282	7.!
				Futrelle						

```
#dropping all the categorical variables to the variable nameticket
nameticket = train_data.drop(labels=['Name','Ticket'],axis=1,inplace=True)
```

```
train_data.head(10)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Deck
0	1	0	3	0	22.000000	1	0	7.2500	0	8
1	2	1	1	1	38.000000	1	0	71.2833	1	3
2	3	1	3	1	26.000000	0	0	7.9250	0	8
3	4	1	1	1	35.000000	1	0	53.1000	0	3
4	5	0	3	0	35.000000	0	0	8.0500	0	8
5	6	0	3	0	23.799293	0	0	8.4583	2	8
6	7	0	1	0	54.000000	0	0	51.8625	0	5
7	8	0	3	0	2.000000	3	1	21.0750	0	8
8	9	1	3	1	27.000000	0	2	11.1333	0	8
9	10	1	2	1	14.000000	1	0	30.0708	1	8

```
#importing accuracy rate from sklearn .metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
x = train_data.drop('Survived',axis=1)
y = train_data['Survived']
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.33,random_state = 42)
```

```
x_train.shape
```

```
(596, 9)
```

```
x_test.shape
```

```
(295, 9)
```

After training and testing split.we are going to apply logistic regression algorithm but before using this algo we have to import it from sklearn.linear_model library.

```
#importing logistic Regression from sklearn
from sklearn.linear_model import LogisticRegression
```

Training Model

```
#training model
model = LogisticRegression(solver = 'lbfgs',max_iter = 200)
model.fit(x_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=200,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Testing Model

```
# model predicting values
y_predict = model.predict(x_test)
```

Accuracy Score

```
#accuracy over train dataset
Accuracy_over_train_data =model.score(x_test,y_test)
Accuracy_over_train_data
```

```
0.8169491525423729
```

Accuracy Percentage

```
#accuracy %
accuracy =Accuracy_over_train_data*100
```

```
accuracy
```

```
81.69491525423729
```

Test dataset

```
#importing required libraries/packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#reading the required test dataset
test_data = pd.read_csv('test.csv')
```

```
#using head to have vision on test data
test_data.head(10)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Er
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	
			Hirvonen, Mrs.								

```
#checking total rows &columns
test_data.shape
```

```
(418, 11)
```

```
#analysing testdata
test_data.describe()
```

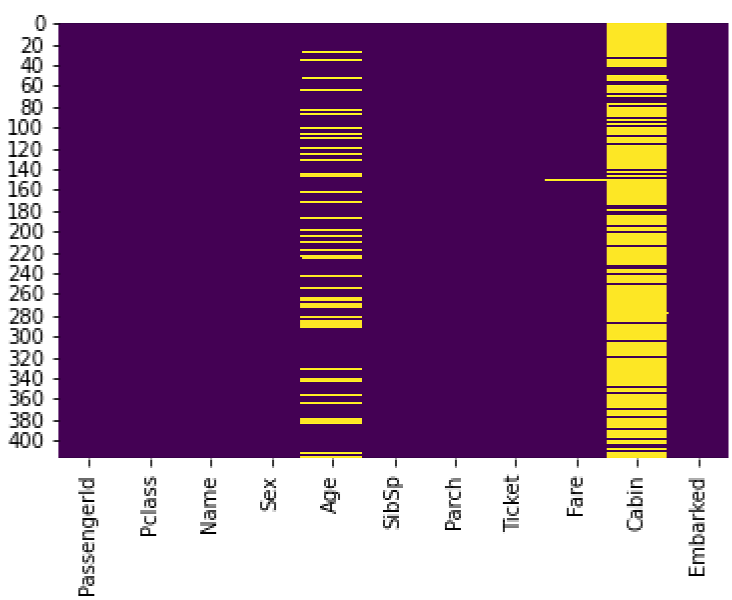
	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200

```
#checking for missing values
test_data.isnull().sum()
```

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
#using heatmap for visualising dataset for missing values
sns.heatmap(test_data.isnull(),cbar = False,cmap = 'viridis')
```

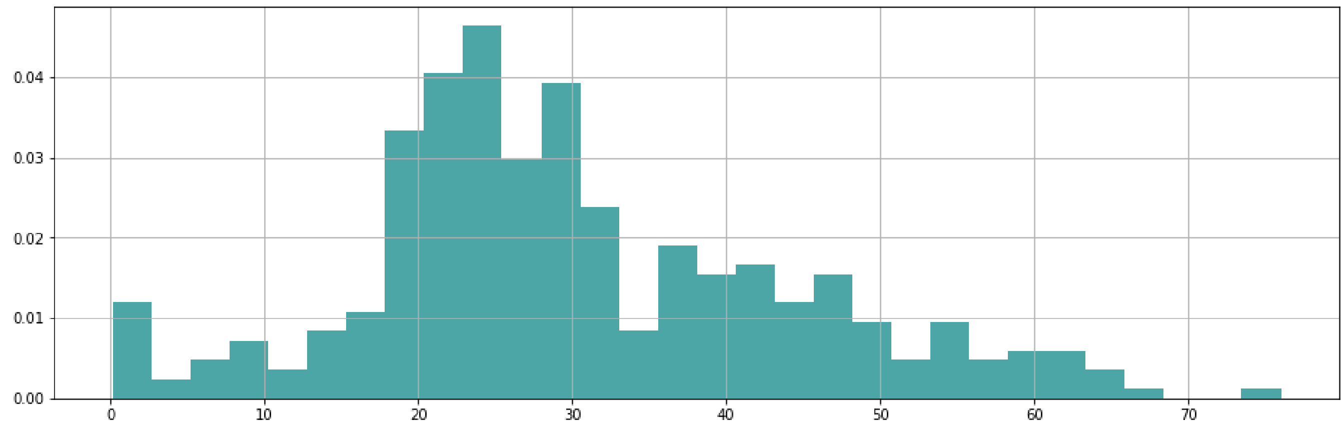
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb573b41400>
```



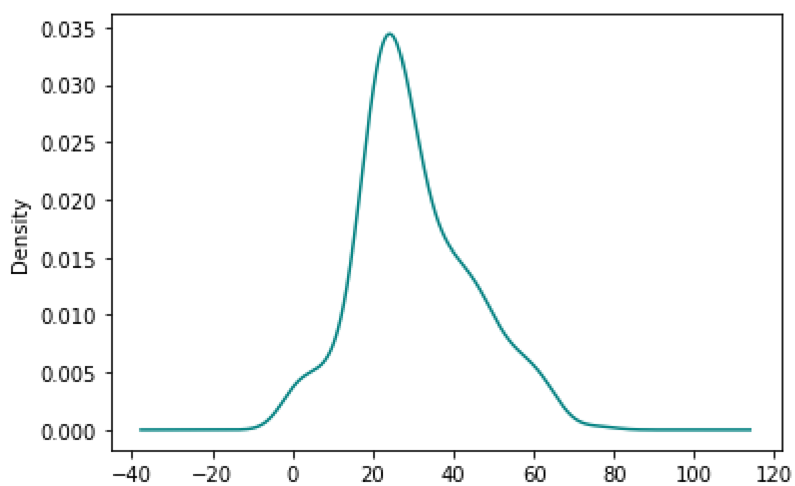

```
#checking  missing percentage for age
test_data['Age'].isnull().sum()/test_data.shape[0]*100
```

```
20.574162679425836
```

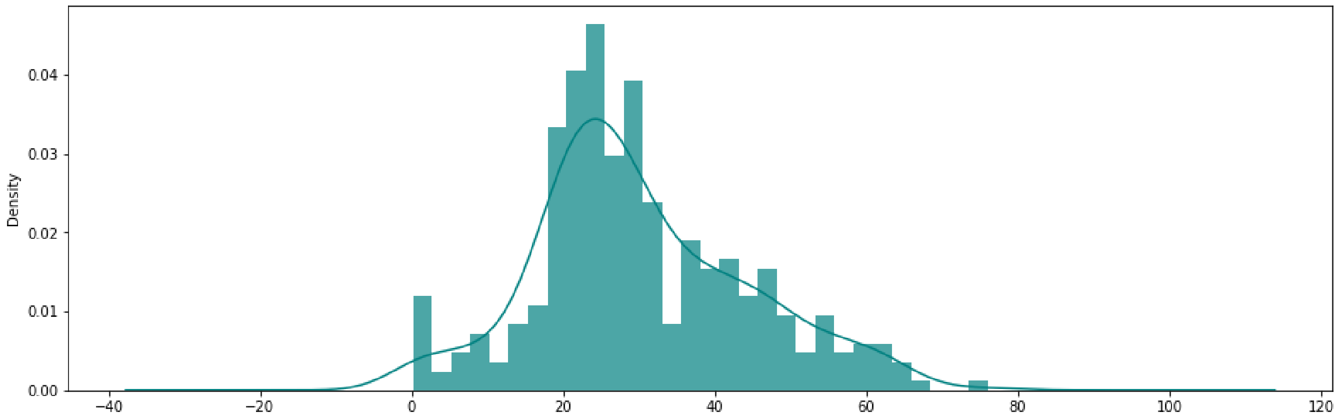
```
#analysing age data via plot graph
plot = test_data['Age'].hist(bins=30,density = True,stacked= True,color='teal',alpha=0.7,figs
```




```
test_data['Age'].plot(kind = 'density',color = 'teal')
plot.set_label('Age')
plt.show()
```




```
plot = test_data['Age'].hist(bins=30,density = True,stacked= True,color='teal',alpha=0.7,figs
test_data['Age'].plot(kind = 'density',color = 'teal')
plot.set_label('Age')
plt.show()
```

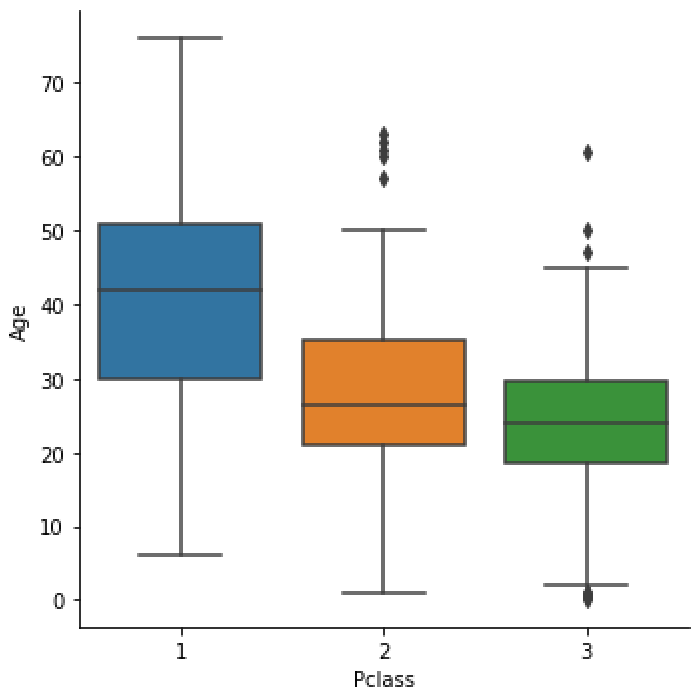



```
#checking value counts for column sex
test_data['Sex'].value_counts()
```

```
male      266
female    152
Name: Sex, dtype: int64
```

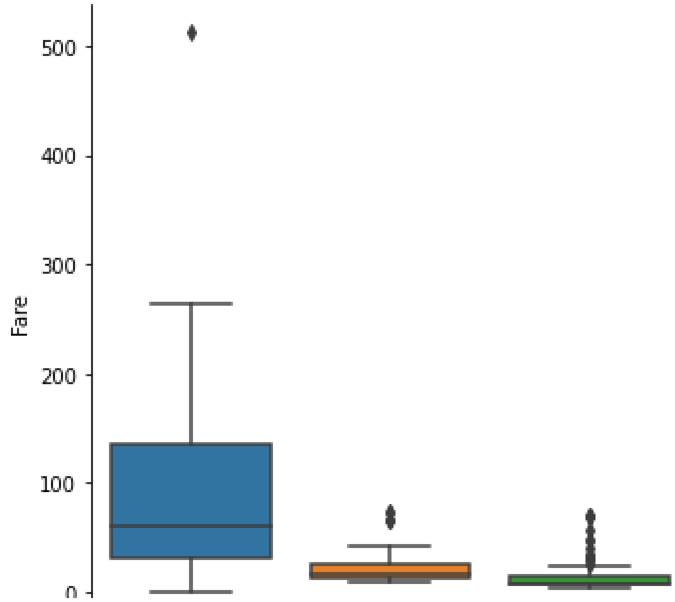
```
#checking boxplot for quartiles for pclass and age relation
sns.catplot(x = 'Pclass',y = 'Age',data = test_data ,kind = 'box')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb5739a0f60>
```




```
#checking box plot for pclass & fare
sns.catplot(x = 'Pclass',y = 'Fare',data = test_data ,kind = 'box')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb573a19198>
```




```
#total missing % value group by columns
(test_data.isnull().sum()/len(test_data.index))*100
```

```
PassengerId     0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            20.574163
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.239234
Cabin          78.229665
Embarked        0.000000
dtype: float64
```

```
#calculating avg age for pclass==1
test_data[test_data['Pclass']==1]['Age'].mean()
```

```
40.91836734693877
```

```
#calculating avg age for pclass==2
test_data[test_data['Pclass']==2]['Age'].mean()
```

```
28.7775
```

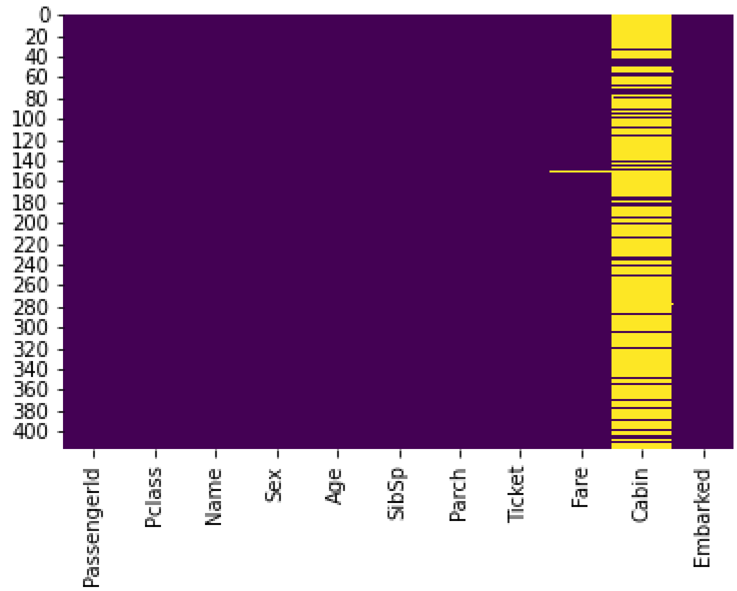
```
#calculating avg age for pclass==3
test_data[test_data['Pclass']==3]['Age'].mean()
```

```
24.02794520547945
```

```
#filling avg age values inplace of null values
total=train_data['Age'].sum()
```

```
common_age = total/418

for i in test_data:
    test_data['Age'] = test_data['Age'].fillna(common_age)


#again checking dataset for missing values via heatmap
sns.heatmap(test_data.isnull(),cbar = False ,cmap = 'viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb57392af60>
```



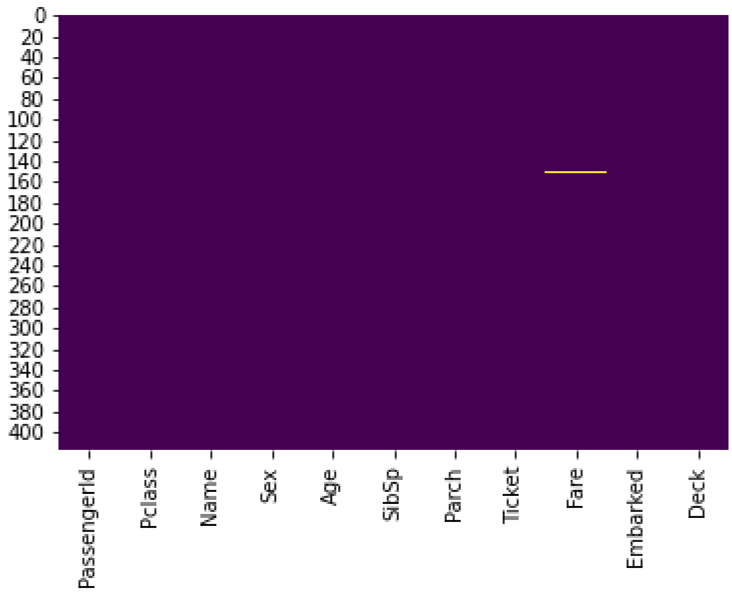

```
#extracting the deck values from cabin values using re library
import re
deck = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7, "U": 8}
data = [test_data]

for dataset in data:
    dataset['Cabin'] = dataset['Cabin'].fillna("U0")
    dataset['Deck'] = dataset['Cabin'].map(lambda x: re.compile("([a-zA-Z]+)").search(x).grou
    dataset['Deck'] = dataset['Deck'].map(deck)
    dataset['Deck'] = dataset['Deck'].fillna(0)
    dataset['Deck'] = dataset['Deck'].astype(int)
#dropping the cabin feature
test_data = test_data.drop(['Cabin'], axis=1)


#check for the remaining missing values
test_data.isnull().sum()
```

```
PassengerId    0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
```

```
Ticket          0
Fare            1
Embarked        0
Deck            0
dtype: int64
```

```
#cehcking for missing values via heatmap
sns.heatmap(test_data.isnull(),cbar = False ,cmap = 'viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb57ab2d390>
```




```
#checking for total sum of missing values in fare
test_data['Fare'].isnull().sum()
```

```
1
```

```
test_data.head(20)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	892	3	Kelly, Mr. James	male	34.500000	0	0	330911	7.8292
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.000000	1	0	363272	7.0000
2	894	2	Myles, Mr. Thomas Francis	male	62.000000	0	0	240276	9.6875
3	895	3	Wirz, Mr. Albert	male	27.000000	0	0	315154	8.6625
4	896	3	Hirvonen, Mrs. Alexander	female	22.000000	1	1	3101298	12.2875

```
#calculating the mean value for fare
np.mean(test_data['Fare'])
```

35.6271884892086

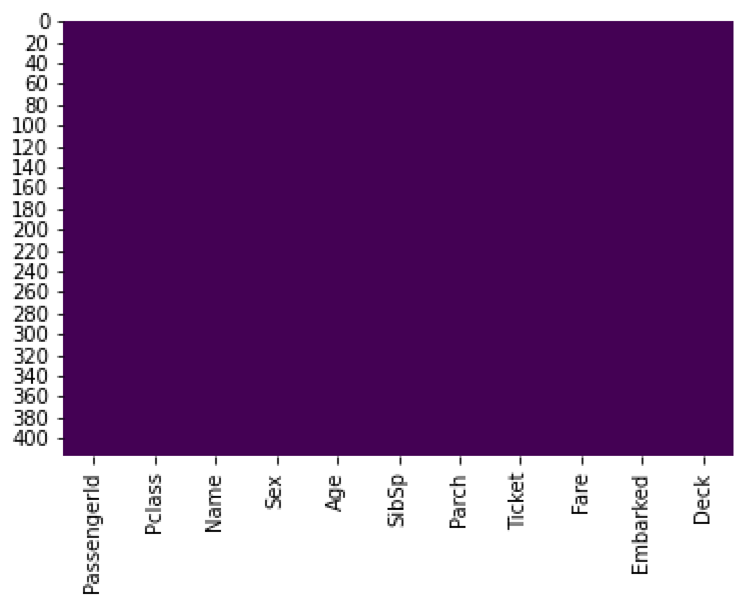
Cervin

```
#filling mean value of fare to that column that contain one single null value
for i in test_data:
    test_data['Fare'] = test_data['Fare'].fillna(np.mean(test_data['Fare']))
```

7	899	2	Mr. Albert	male	26.000000	1	1	248738	29.0000

```
#using heatmap for checking data for missing values
sns.heatmap(test_data.isnull(),cbar = False ,cmap = 'viridis')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fb573785048>




```
#using isnull to check the data for null values
test_data.isnull().sum()
```

```
PassengerId    0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
Deck           0
dtype: int64
```

```
#converting categorical values to numeral and making our test datset ready for testing model
genders = {'male': 0, 'female': 1}
test_data['Sex'] = test_data['Sex'].map(genders)

ports = {'S':0,'C':1,'Q':2}
test_data['Embarked'] = test_data['Embarked'].map(ports)

test_data.head(10)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	D
0	892	3	Kelly, Mr. James	0	34.5	0	0	330911	7.8292	2	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	1	47.0	1	0	363272	7.0000	0	
2	894	2	Myles, Mr. Thomas Francis	0	62.0	0	0	240276	9.6875	2	
3	895	3	Wirz, Mr. Albert	0	27.0	0	0	315154	8.6625	0	
			Hirvonen, Mrs.								

```
#dropping all the remainning categorical that can't be numeral to the nameticket2
nameticket2 = test_data.drop(labels=['Name','Ticket'],axis=1,inplace=True)
```

```
#visulaizing data
test_data.head(10)
```

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Deck
0	892	3	0	34.5	0	0	7.8292	2	8
1	893	3	1	47.0	1	0	7.0000	0	8
2	894	2	0	62.0	0	0	9.6875	2	8
3	895	3	0	27.0	0	0	8.6625	0	8
4	896	3	1	22.0	1	1	12.2875	0	8
5	897	3	0	14.0	0	0	9.2250	0	8
6	898	3	1	30.0	0	0	7.6292	2	8
7	899	2	0	26.0	1	1	29.0000	0	8

Congrats!!Test Data Is Ready now for testing model

```
#importing accuracy rate from sklearn metrics
from sklearn.metrics import accuracy_score
```

```
x_test1 = test_data
x = train_data.drop('Survived',axis=1)
y = train_data['Survived']
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.33,random_state=42)
```

```
x_train.shape
```

```
(596, 9)
```

```
model = LogisticRegression(solver = 'lbfgs',max_iter=400)
```

```
model.fit(x_train,y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=400,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
#dividing our test data into two data sets for testing model test1 &test2
x_test1,x_test2,y_test1,y_test2 = train_test_split(x,y,test_size = 0.33,random_state=42)
```

```
x_test1.shape
```

```
(596, 9)
```

```
#testing model over test1
y_predict = model.predict(x_test1)
```

```
#accuracy score for test 1
model.score(x_test1,y_test1)
```

```
0.7969798657718121
```

```
x_test2.shape
```

```
(295, 9)
```

```
#testing model over test2
y_predict = model.predict(x_test2)
```

```
#accuracy score for test2
model.score(x_test2,y_test2)
```

```
0.8135593220338984
```

```
#average accuracy rate over test1&test2
avg_accuracy = ((model.score(x_test1,y_test1))+(model.score(x_test2,y_test2)))*0.5
avg_accuracy
```

```
0.8052695939028552
```

```
#checking accuracy difference rate for training dataset and testing dataset
difference_accuracy_train_test = model.score(x_test,y_test)- avg_accuracy
difference_accuracy_train_test
```

```
0.008289728131043117
```

Accuracy Score over train.csv dataset = 0.8169491525423729 Accuracy Score over test.csv dataset = 0.8052695939028552 very neglible difference between accuracies that is 0.008289728131043117 Hence,I can say my model is pretty good which is able to predict values for purely unseen data (test.csv) after getting training on separate dataset(train.csv).

Feature Selection Using RFE

```
 from sklearn.linear_model import LogisticRegression
 logreg = LogisticRegression()
```

```
from sklearn.feature_selection import RFE
```

```
rfe = RFE(logreg, 15)
rfe = rfe.fit(x_train,y_train)
```

```
rfe.support_
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True])
```

```
list(zip(x_train.columns, rfe.support_, rfe.ranking_))
```

```
[('PassengerId', True, 1),
 ('Pclass', True, 1),
 ('Sex', True, 1),
 ('Age', True, 1),
 ('SibSp', True, 1),
 ('Parch', True, 1),
 ('Fare', True, 1),
 ('Embarked', True, 1),
 ('Deck', True, 1)]
```

```
c = x_train.columns[rfe.support_]
```

```
x_train.columns[~rfe.support_]
```

```
Index([], dtype='object')
```

Assessing the model with StatsModels

```
import statsmodels.api as sm
x_train_sm = sm.add_constant(x_train[c])
logm2 = sm.GLM(y_train,x_train_sm, family = sm.families.Binomial())
res = logm2.fit()
res.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	Survived	**No. Observations:**	596
Model:	GLM	**Df Residuals:**	586
Model Family:	Binomial	**Df Model:**	9
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-273.61
Date:	Tue, 25 Aug 2020	**Deviance:**	547.22

```
 # Getting the predicted values on the train set
y_train_pred = res.predict(x_train_sm)
y_train_pred[:10]
```

```
6      0.268788
718    0.199808
685    0.222094
73     0.103991
882    0.665343
328    0.495545
453    0.363051
145    0.195079
234    0.224522
220    0.131954
dtype: float64
```

Deck 0.0414 0.076 0.545 0.585 0.190 0.107

```
y_train_pred = y_train_pred.values.reshape(-1)
y_train_pred[:10]
```

```
array([0.26878776, 0.19980754, 0.22209417, 0.10399124, 0.66534285,
       0.4955455 , 0.36305091, 0.19507875, 0.22452182, 0.13195433])
```

Creating a dataframe with the actual Survival rate and the predicted probabilities

```
y_train_pred_final = pd.DataFrame({'Survived':y_train.values, 'Survived_Prob':y_train_pred})
y_train_pred_final['PassengerId'] = y_train.index
y_train_pred_final .head()
```

	Survived	Survived_Prob	PassengerId
0	0	0.268788	6
1	0	0.199808	718
2	0	0.222094	685
3	0	0.103991	73
4	0	0.665343	882

Creating new column ‘Survival_Rate’ with 1 if Survived_Prob > 0.5 else 0

```
y_train_pred_final['Survival_Rate'] = np.where(y_train_pred_final['Survived_Prob'] >= 0.50, 1
y_train_pred_final.head()
```

	Survived	Survived_Prob	PassengerId	Survival_Rate
0	0	0.268788	6	0
1	0	0.199808	718	0
2	0	0.222094	685	0
3	0	0.103991	73	0
4	0	0.665343	882	1

Confusion Matrix

```
from sklearn import metrics
```

```
confusion_matrix = metrics.confusion_matrix(y_train_pred_final.Survived, y_train_pred_final.S
print(confusion_matrix)
```

```
[[324  50]
 [ 72 150]]
```

```
print(metrics.accuracy_score(y_train_pred_final.Survived, y_train_pred_final.Survival_Rate))
```

```
0.7953020134228188
```

Checking VIFs

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
vif_data = pd.DataFrame()
```

```
vif_data['Features'] = x_train[c].columns
vif_data['VIF'] = [variance_inflation_factor(x_train[c].values, i) for i in range(x_train[c].
vif_data['VIF'] = round(vif_data['VIF'], 2)
vif_data = vif_data.sort_values(by = "VIF", ascending = False)
vif_data
```

	Features	VIF
8	Deck	25.16
1	Pclass	22.61
3	Age	4.40
0	PassengerId	3.63
6	Fare	1.75

```
c = c.drop('Deck', 1)
c
```

```
Index(['PassengerId', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare',
       'Embarked'],
      dtype='object')
```

```
c = c.drop('Pclass', 1)
c
```

```
Index(['PassengerId', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked'], dtype='object
```

```
# Let's re-run the model using the selected variables
x_train_sm = sm.add_constant(x_train[c])
logm3 = sm.GLM(y_train,x_train_sm, family = sm.families.Binomial())
res = logm3.fit()
res.summary()
```

Generalized Linear Model Regression Results

Dep. Variable: Survived **No. Observations:** 596

```
y_train_pred = res.predict(x_train_sm).values.reshape(-1)
y_train_pred[:10]
```

```
array([0.19129475, 0.27040719, 0.16364582, 0.14419276, 0.76551232,
       0.6127844 , 0.30212147, 0.14525998, 0.16668015, 0.17418434])
```

Time: 12:52:14 **Pearson chi2:** 590.

```
y_train_pred_final['Survived_Prob'] = y_train_pred
```

```
y_train_pred_final['Survival_Rate'] = np.where(y_train_pred_final['Survived_Prob'] >= 0.50, 1
y_train_pred_final.head()
```

	Survived	Survived_Prob	PassengerId	Survival_Rate
0	0	0.191295	6	0
1	0	0.270407	718	0
2	0	0.163646	685	0
3	0	0.144193	73	0
4	0	0.765512	882	1

```
print(metrics.accuracy_score(y_train_pred_final.Survived, y_train_pred_final.Survival_Rate))
```

```
0.7869127516778524
```

checking VIFs again

```
vif = pd.DataFrame()
vif['Features'] = x_train[c].columns
vif['VIF'] = [variance_inflation_factor(x_train[c].values, i) for i in range(x_train[c].shape
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

	Features	VIF
2	Age	3.00

```
x_train_sm = sm.add_constant(x_train[c])
logm4 = sm.GLM(y_train,x_train_sm, family = sm.families.Binomial())
res = logm4.fit()
res.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	Survived	**No. Observations:**	596
Model:	GLM	**Df Residuals:**	588
Model Family:	Binomial	**Df Model:**	7
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-288.66
Date:	Tue, 25 Aug 2020	**Deviance:**	577.33
Time:	12:59:46	**Pearson chi2:**	590.
No. Iterations:	5		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-1.5633	0.355	-4.409	0.000	-2.258	-0.868
PassengerId	0.0004	0.000	0.866	0.387	-0.000	0.001
Sex	2.5418	0.229	11.105	0.000	2.093	2.990
Age	-0.0117	0.009	-1.363	0.173	-0.028	0.005
SibSp	-0.3352	0.113	-2.973	0.003	-0.556	-0.114
Parch	-0.2343	0.140	-1.672	0.094	-0.509	0.040
Fare	0.0145	0.003	4.221	0.000	0.008	0.021
Embarked	0.1863	0.163	1.144	0.253	-0.133	0.505

```
y_train_pred = res.predict(x_train_sm).values.reshape(-1)
```

```
y_train_pred[:10]
```

```
array([0.19129475, 0.27040719, 0.16364582, 0.14419276, 0.76551232,
       0.6127844 , 0.30212147, 0.14525998, 0.16668015, 0.17418434])
```

```
confusion = metrics.confusion_matrix(y_train_pred_final.Survived, y_train_pred_final.Survival
confusion
```

```
array([[323,  51],
       [ 76, 146]])
```

```
metrics.accuracy_score(y_train_pred_final.Survived, y_train_pred_final.Survival_Rate)
```

```
0.7869127516778524
```

We have seen the accuracy via confusion metrics decreases though it's a slight change but it shows only accuracy is not enough to test model on the basis of its predictions.there are many more methods to test model and its predictions via precision ,recall, f1 score ,sensitivity,specificity n so on...

Overall accuracy is 80%approx, that is acceptable.

THANK YOU!!