# Titanic Survival

September 27, 2023

## 1 DATA SCIENCE PROJECT (TITANIC SURVIVAL)

[377]:
```
import pandas as pd
# data manipulation

import numpy as np
# Mathematical Operation
```

[378]:
```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
import seaborn as sns
```

[379]:
```
# Loading the data
data=pd.read_csv('Titanic-Dataset.csv')
data.head()
```

[379]:

| | 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 | | | | | |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | | | | | |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | | | | | |

```
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[380]:
```
# checking the shape of the Dataset
data.shape
```

[380]:
```
(891, 12)
```

[381]:
```
data.describe()
```

[381]:
```
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

[382]:
```
data.info()
# To get the overall information of dataset
```

```
<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
```

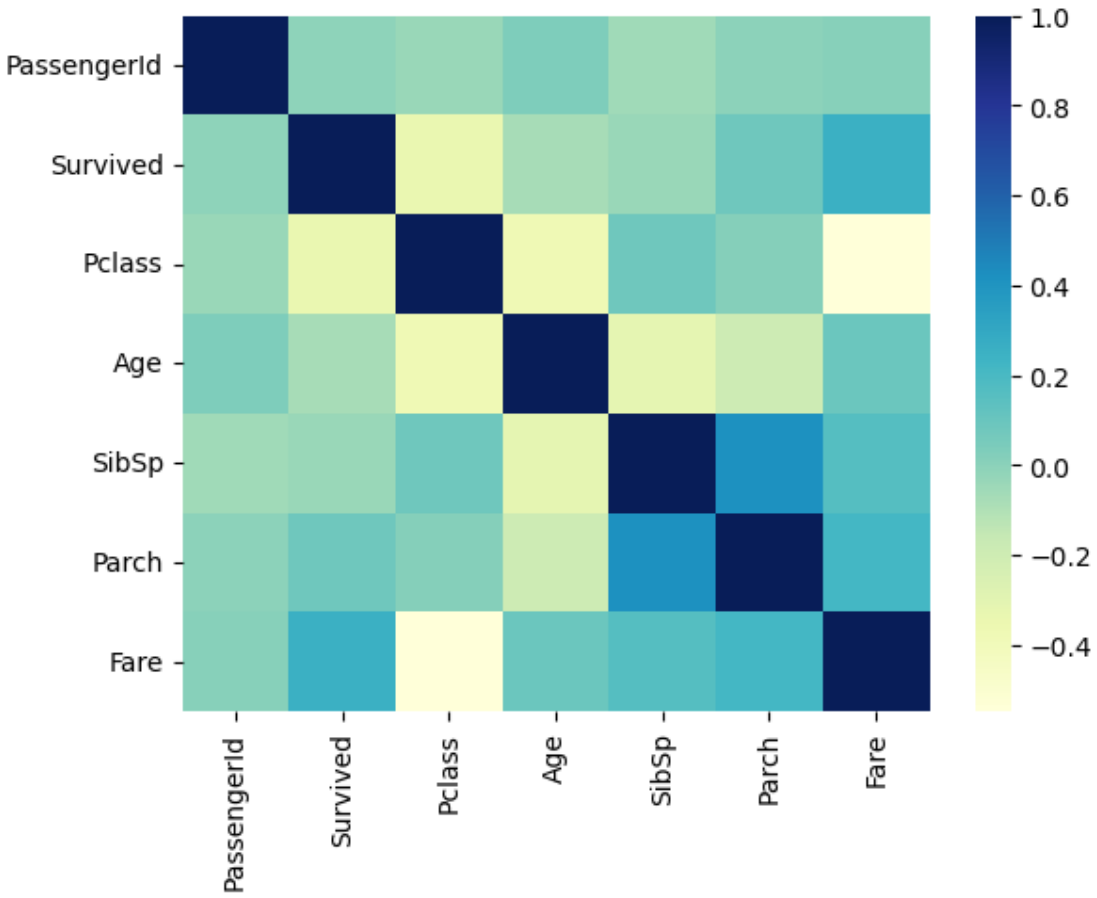
[383]:
```
data.corr()
```

[383]:
```
             PassengerId  Survived    Pclass       Age     SibSp     Parch  \
PassengerId     1.000000 -0.005007 -0.035144  0.036847 -0.057527 -0.001652
Survived       -0.005007  1.000000 -0.338481 -0.077221 -0.035322  0.081629
Pclass         -0.035144 -0.338481  1.000000 -0.369226  0.083081  0.018443
Age             0.036847 -0.077221 -0.369226  1.000000 -0.308247 -0.189119
SibSp          -0.057527 -0.035322  0.083081 -0.308247  1.000000  0.414838
Parch          -0.001652  0.081629  0.018443 -0.189119  0.414838  1.000000
Fare            0.012658  0.257307 -0.549500  0.096067  0.159651  0.216225

                 Fare
PassengerId  0.012658
Survived     0.257307
Pclass      -0.549500
Age          0.096067
SibSp        0.159651
Parch        0.216225
Fare         1.000000
```

The great aspect of the pandas module is corr() method. The corr() method calculated the realtionship between each column in your data set. df.corr()

[384]:
```
sns.heatmap(data.corr(), cmap='YlGnBu')
plt.show()
```




[385]:
```
data.drop(columns=['Cabin'],inplace=True)
```

[386]:
```
data.head()
```

[386]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th… | female | 38.0 | 1 |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 |

```
   Parch            Ticket     Fare Embarked
0      0         A/5 21171   7.2500        S
1      0          PC 17599  71.2833        C
2      0  STON/O2. 3101282   7.9250        S
3      0            113803  53.1000        S
4      0            373450   8.0500        S
```

[387]:
```
data.isnull().sum()
```

[387]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Embarked         2
dtype: int64
```

This describes the count of null values in that column of the dataset

[388]:
```
data['Embarked'].value_counts()
```

[388]:
```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

It shows the number of people travelling to that cities

[389]:
```
data['Embarked'].fillna('S',inplace=True)
```

There were 2 missing values in Embarked which we replaced that 2 values by S

[390]:
```
data.isnull().sum()
```

[390]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
```

```
Embarked         0
dtype: int64
```

Embarked has no null values

[391]:
```
data['Fare'].fillna(data['Fare'].mean(),inplace=True)
```

Now Fare column has no Null Values . Similarly we will predict the values of empty age values with the average.

[392]:
```
data['Age'].fillna(data['Age'].mean(),inplace=True)
```

[393]:
```
data.isnull().sum()
```

[393]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

[394]:
```
# EDA
```

Now we will filter out the survived according to pclass

[395]:
```
data[data['Pclass']==1]['Survived'].value_counts()
```

[395]:
```
1    136
0     80
Name: Survived, dtype: int64
```

Here 136 have survived and 80 are dead in class 1

[396]:
```
data[data['Pclass']==2]['Survived'].value_counts()
```

[396]:
```
0    97
1    87
Name: Survived, dtype: int64
```

87 survived and 97 are dead

[397]:
```
data.groupby(['Pclass'])['Survived'].mean()
```

[397]:
```
Pclass
1    0.629630
2    0.472826
3    0.242363
Name: Survived, dtype: float64
```

SO here we got the percentage of the people who survived in particular Pclasses according to data for
class 1 == 62% ,
class 2== 47% ,
class 3== 24%

Now we will find the Percentage of people those who are dead on the basis of sex

[398]:
```
data.groupby(['Sex'])['Survived'].mean()
```

[398]:
```
Sex
female    0.742038
male      0.188908
Name: Survived, dtype: float64
```

So Female who survived are 74% and male are 18%

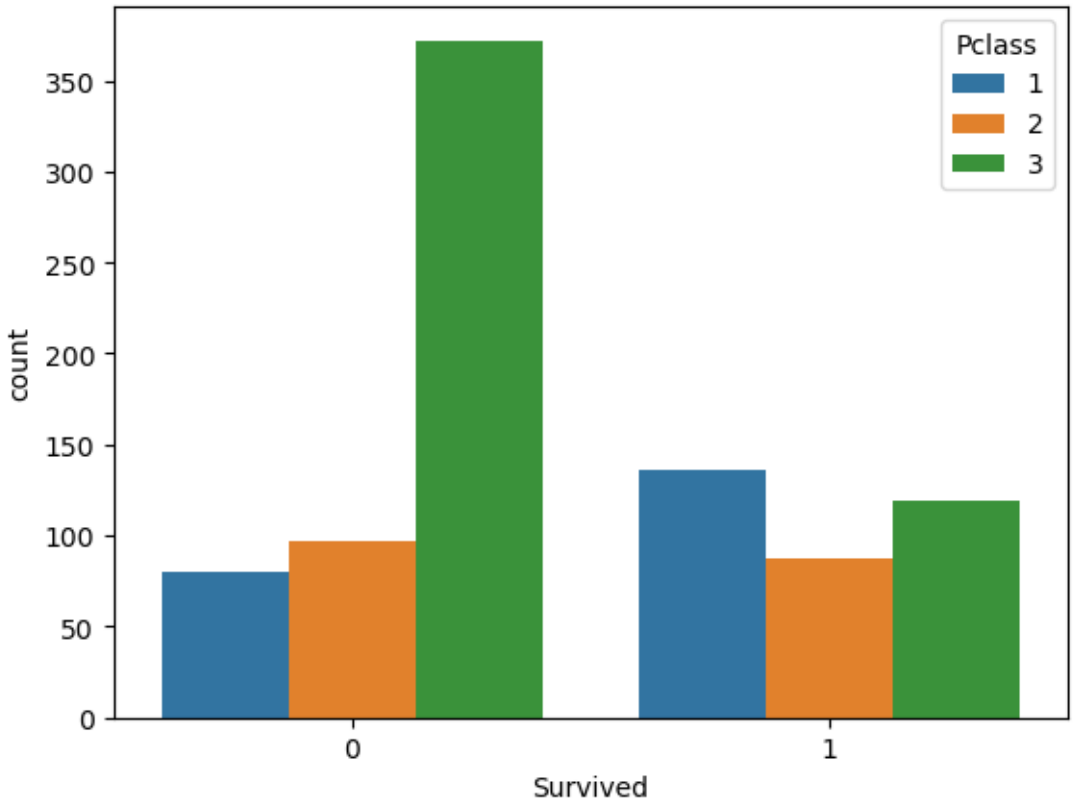
[399]:
```
data.groupby(['Embarked'])['Survived'].mean()
```

[399]:
```
Embarked
C    0.553571
Q    0.389610
S    0.339009
Name: Survived, dtype: float64
```

The chance of people surviving for particular city is given as above

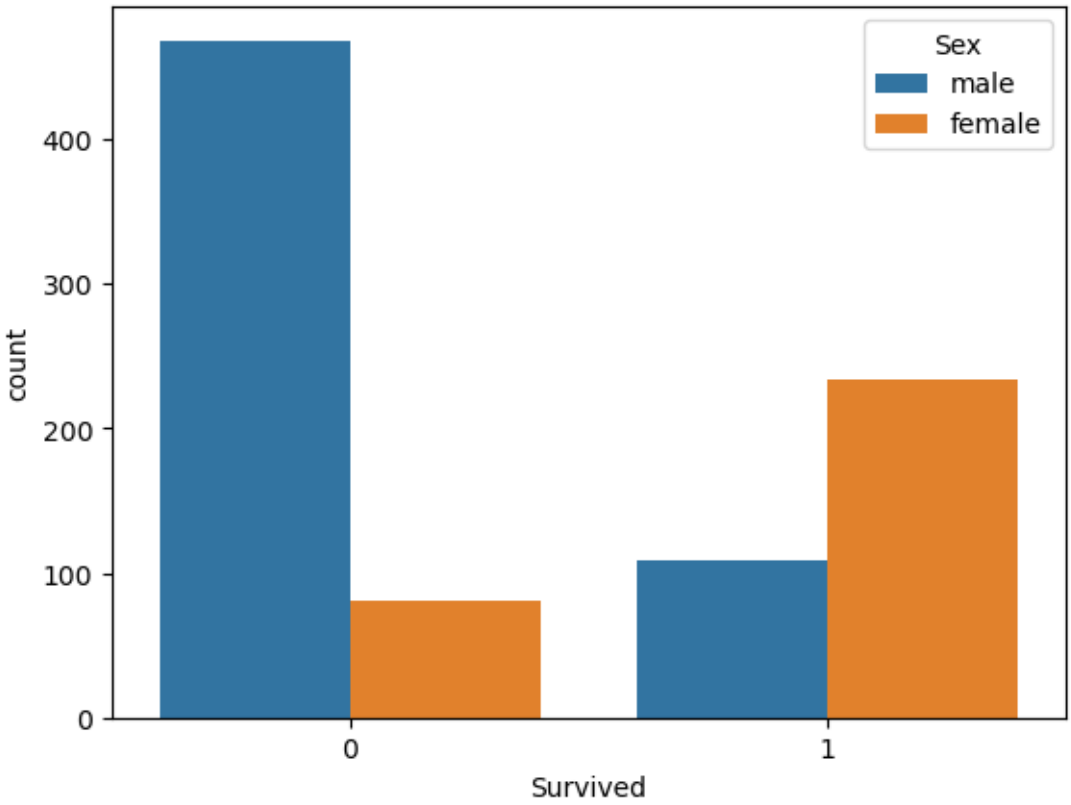
[400]:
```
sns.countplot(x=data['Survived'],hue=data['Pclass'])
```

[400]:
```
<Axes: xlabel='Survived', ylabel='count'>
```




[401]:
```
sns.countplot(x=data['Survived'],hue=data['Sex'])
```

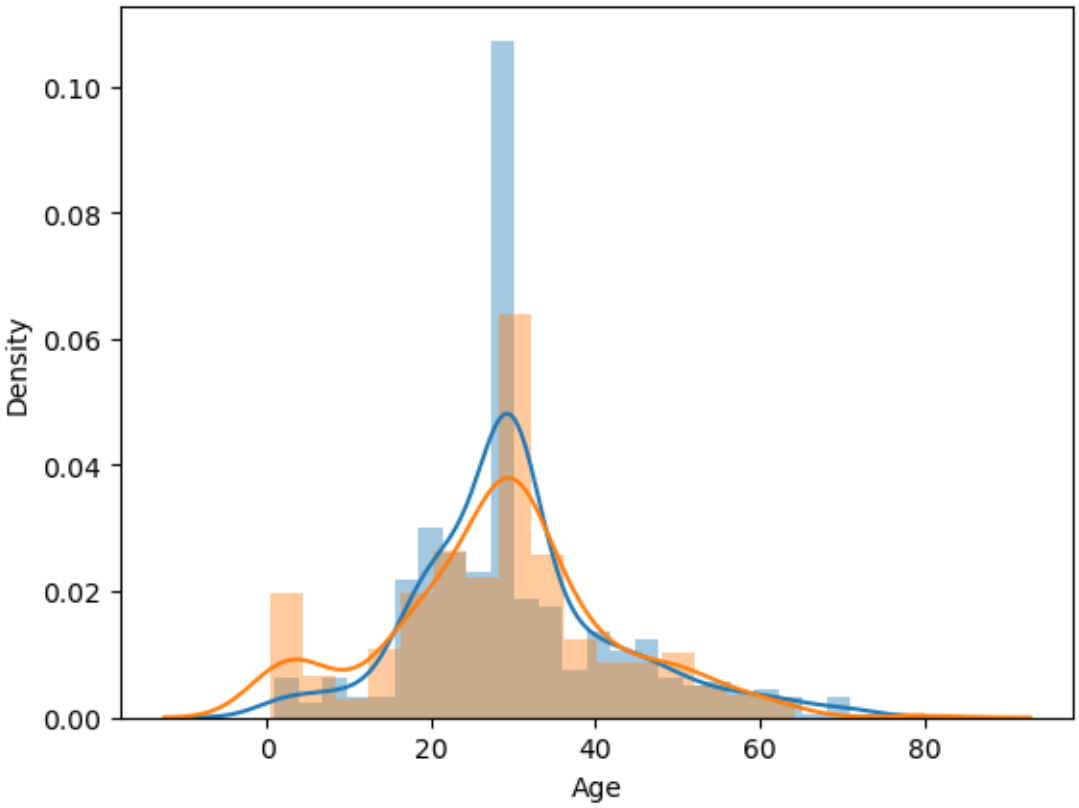
[401]: <Axes: xlabel='Survived', ylabel='count'>




We can plot the number of people who are dead according to age and also the peoplw who survived people with low age have survived more , age medium have many dead , and people with old age are having more chances of being dead

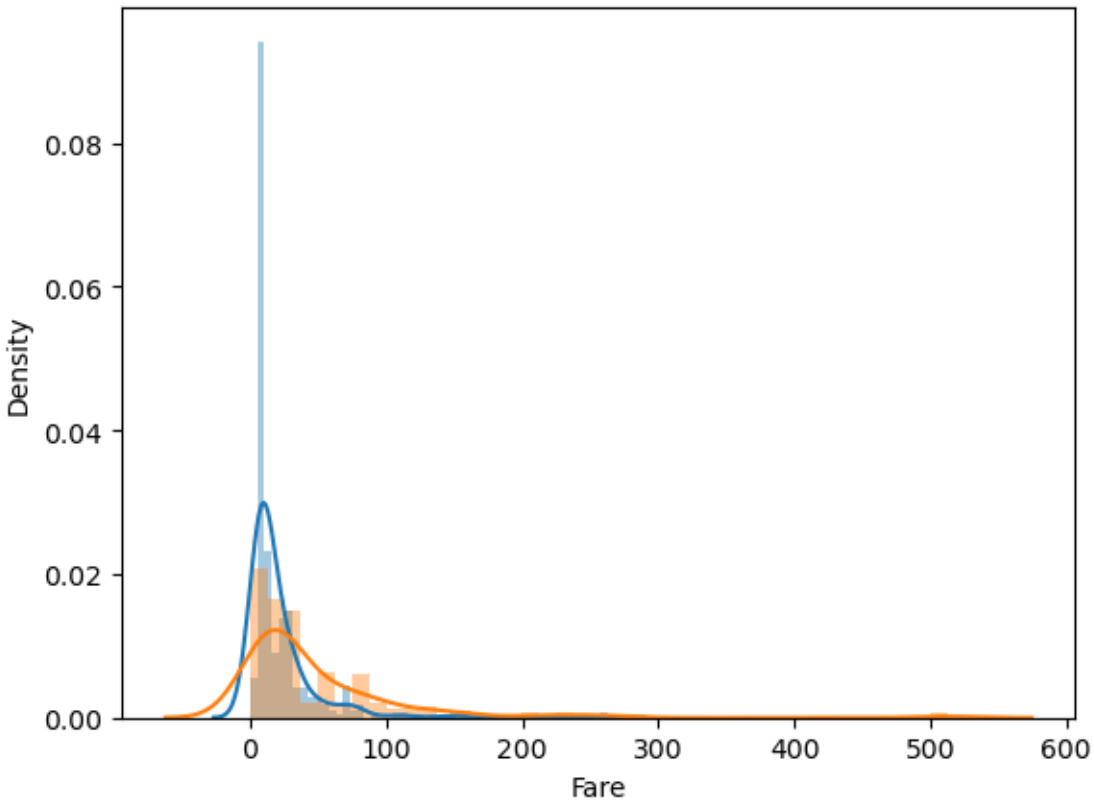
[402]:
```
sns.distplot(data['Age'][data['Survived']==0])
sns.distplot(data['Age'][data['Survived']==1])
```

[402]:
```
<Axes: xlabel='Age', ylabel='Density'>
```




[403]:
```
sns.distplot(data['Fare'][data['Survived']==0])
sns.distplot(data['Fare'][data['Survived']==1])
```

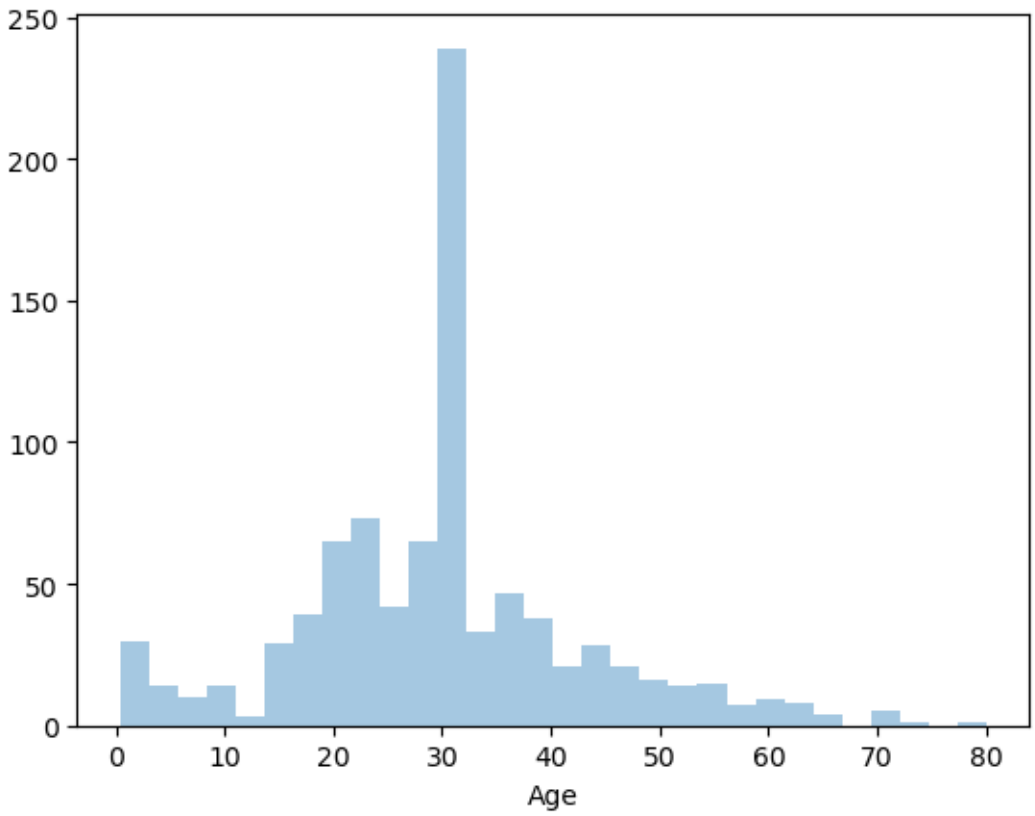
[403]: <Axes: xlabel='Fare', ylabel='Density'>




[ ]:

[404]:
```
age1=data['Age'].dropna()
#dropna will drop the null values
sns.distplot(age1,bins=30,kde=False)
plt.show()

#kde is used to plot the data whisch is not normal inshort kde will
# plot a line on the graph of the data
#bins (i.e is the number of bars in the graph)
```




[405]:
```
data.drop(columns=['Ticket'],inplace=True)
```

[406]:
```
data.head()
```

[406]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Fare | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | 7.2500 | S |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th… | female | 38.0 | 1 | 0 | 71.2833 | C |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | | | |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | | | |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | | | |

```
2      0   7.9250        S
3      0  53.1000        S
4      0   8.0500        S
```

[407]:
```
data['Family']=data['SibSp']+data['Parch']+1
data.head()
```

[407]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch     Fare Embarked  Family
0      0   7.2500        S       2
1      0  71.2833        C       2
2      0   7.9250        S       1
3      0  53.1000        S       2
4      0   8.0500        S       1
```

So here we are creating the column of family where we are describing the no of people including himself/herself (Sibsp and Parch)

[408]:
```
data['Family'].value_counts()
```

[408]:
```
1     537
2     161
3     102
4      29
6      22
5      15
7      12
11      7
8       6
Name: Family, dtype: int64
```

[409]:
```
data.groupby(['Family'])['Survived'].mean()
```

[409]:
```
Family
1     0.303538
```

```
2     0.552795
3     0.578431
4     0.724138
5     0.200000
6     0.136364
7     0.333333
8     0.000000
11    0.000000
Name: Survived, dtype: float64
```

This shows that travelling alone i.e 1 has chances of surving around 30% and so on the data, and people travelling with 11 members have 0 chances

[410]:
```
def cal(number):
    if number==1:
        return "Alone"
    elif number>1 and number <5:
        return "Medium"
    else:
        return "Large"
```

Here we create a function for defining a column based on number of people depending upon large medium alone

[411]:
```
data['Family_size']=data['Family'].apply(cal)
```

[412]:
```
data.head()
```

[412]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Fare | Embarked | Family | Family_size |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | 7.2500 | S | 2 | Medium |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th… | female | 38.0 | 1 | 0 | 71.2833 | C | 2 | Medium |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | 7.9250 | S | 1 | Alone |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 53.1000 | S | 2 | Medium |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 8.0500 | S | 1 | Alone |

so now sibsp and parch is not needed

[413]:
```
data.drop(columns=['SibSp','Parch','Family'],inplace=True)
```

[414]:
```
data.head()
```

[414]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age     Fare  \
0                            Braund, Mr. Owen Harris    male  22.0   7.2500
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0  71.2833
2                             Heikkinen, Miss. Laina  female  26.0   7.9250
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0  53.1000
4                           Allen, Mr. William Henry    male  35.0   8.0500

  Embarked Family_size
0        S      Medium
1        C      Medium
2        S       Alone
3        S      Medium
4        S       Alone
```

[415]:
```
data.shape
```

[415]:
```
(891, 9)
```

[416]:
```
data.isnull().sum()
```

[416]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
Fare           0
Embarked       0
Family_size    0
dtype: int64
```

[417]:
```
data.drop(columns=['PassengerId','Name'],inplace=True)
data.head()
```

[417]:

| | Survived | Pclass | Sex | Age | Fare | Embarked | Family_size |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | male | 22.0 | 7.2500 | S | Medium |
| 1 | 1 | 1 | female | 38.0 | 71.2833 | C | Medium |
| 2 | 1 | 3 | female | 26.0 | 7.9250 | S | Alone |
| 3 | 1 | 1 | female | 35.0 | 53.1000 | S | Medium |
| 4 | 0 | 3 | male | 35.0 | 8.0500 | S | Alone |

This values are based on categorical data or values so this values need to be converted to numerical

[418]:
```
data=pd.
↪get_dummies(data,columns=['Pclass','Sex','Embarked','Family_size'],drop_first=True)
```

This will change the categorical data into numerical data by creating different columns for male female and for embarked and so on

[419]:
```
data
```

[419]:

| | Survived | Age | Fare | Pclass_2 | Pclass_3 | Sex_male | Embarked_Q | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 22.000000 | 7.2500 | 0 | 1 | 1 | 0 | |
| 1 | 1 | 38.000000 | 71.2833 | 0 | 0 | 0 | 0 | |
| 2 | 1 | 26.000000 | 7.9250 | 0 | 1 | 0 | 0 | |
| 3 | 1 | 35.000000 | 53.1000 | 0 | 0 | 0 | 0 | |
| 4 | 0 | 35.000000 | 8.0500 | 0 | 1 | 1 | 0 | |
| .. | … | … | … | … | … | … | … | |
| 886 | 0 | 27.000000 | 13.0000 | 1 | 0 | 1 | 0 | |
| 887 | 1 | 19.000000 | 30.0000 | 0 | 0 | 0 | 0 | |
| 888 | 0 | 29.699118 | 23.4500 | 0 | 1 | 0 | 0 | |
| 889 | 1 | 26.000000 | 30.0000 | 0 | 0 | 1 | 0 | |
| 890 | 0 | 32.000000 | 7.7500 | 0 | 1 | 1 | 1 | |

| | Embarked_S | Family_size_Large | Family_size_Medium |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 |
| 3 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 |
| .. | … | … | … |
| 886 | 1 | 0 | 0 |
| 887 | 1 | 0 | 0 |
| 888 | 1 | 0 | 1 |
| 889 | 0 | 0 | 0 |
| 890 | 0 | 0 | 0 |

[891 rows x 10 columns]

[420]:
```
data.shape
```

[420]: (891, 10)

[421]:
```
x=data.iloc[:,1:].values
y=data.iloc[:,0].values
```

[422]:
```
x
```

[422]:
```
array([[22.        ,  7.25      ,  0.        , …,  1.        ,
         0.        ,  1.        ],
       [38.        , 71.2833    ,  0.        , …,  0.        ,
         0.        ,  1.        ],
       [26.        ,  7.925     ,  0.        , …,  1.        ,
         0.        ,  0.        ],
       …,
       [29.69911765, 23.45      ,  0.        , …,  1.        ,
         0.        ,  1.        ],
       [26.        , 30.        ,  0.        , …,  0.        ,
         0.        ,  0.        ],
       [32.        ,  7.75      ,  0.        , …,  0.        ,
         0.        ,  0.        ]])
```

[423]:
```
y
```

[423]:
```
array([0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
```

```
       1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

[424]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

[425]:
```
clf = DecisionTreeClassifier()
```

[426]:
```
clf.fit(x_train,y_train)
```

[426]:
```
DecisionTreeClassifier()
```

[427]:
```
y_pred= clf.predict(x_test)
y_pred
```

[427]:
```
array([0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1], dtype=int64)
```

[428]:
```
accuracy_score(y_pred,y_test)
```

[428]:
```
0.7821229050279329
```

[430]:
```
reg= SVC()
```

[431]:
```
reg.fit(x_train,y_train)
```

[431]:
```
SVC()
```

[435]:
```
y_pred=reg.predict(x_test)
```

[436]:
```
accuracy_score(y_test,y_pred)
```

[436]:
```
0.6871508379888268
```

The SVM (Support vector Machine classifier )gives about 68% of accuracy

Now we will test the model with the help of Logistic Regression

[438]:
```
clf= LogisticRegression()
```

[ ]:
```

```

[439]:
```
clf.fit(x_train,y_train)
```

[439]:
```
LogisticRegression()
```

[440]:
```
y_pred=clf.predict(x_test)
```

[441]:
```
clf.score(x_test,y_test)
```

[441]:
```
0.8044692737430168
```

[442]:
```
accuracy_score(y_pred,y_test)
```

[442]:
```
0.8044692737430168
```

## 2 Predicting Model

[451]:
```
import warnings
warnings.filterwarnings('ignore')
res=clf.predict([[29,0,0,1,0,1,0,0,1]]) # Enter the input
if(res==0):
    print("Not survived")
else:
    print("Survived")

    # change the 5th index from 0 to 1 for testing
```

```
Survived
```

This predicts whether the person Suvived or not !

## 3 CONCLUSION

Based on the accuracy scores provided, the Logistic Regression classifier and Decision Tree gives the best fit model for the Titanic dataset. The accuracy score for the Logistic Regression classifier is 81.0% , indicating that it approximately predicted all instances in the test set.

[ ]:
```

```

# Titanic Survival - Analysis

40.4%

Survival Rate

Gender

All

PassengerClass

All

Embarked

All