TITANIC PROJECT

SUB

MITTED BY: ANKUSH CHAUDHARI

Developing Machine Learning Model to predict ‘Survived the Sinking'

Introduction

For this assignment, I'd like to look into the tragic sinking of the Titanic. The film "Titanic," which I saw as a child, has made a lasting impression on me. The ship crashed with an iceberg early in the morning of April 15, 1912, killing more than 1500 passengers out of a total of 2,224.

In this blog, I will go through the whole process of creating a machine learning model on the famous Titanic dataset, which is used by many people all over the world. It provides information on the fate of passengers on the Titanic, summarized according to economic status (class), sex, age, and survival.

The Problem Statement

The dataset I'm working with includes demographic data as well as other details such as ticket class, cabin number, and flight amount for 891 travelers. The fundamental subject that interests me is: What are the factors that are associated with passenger survival?

About Dataset

Dataset Link: https://github.com/dsrscientist/dataset1/blob/master/titanic_train.csv

Load Dataset

Before delving deeper, I'd like to gain a general perspective of the data and see if there is any additional data cleaning or wrangling that needs to be done. To begin, I import the CSV file into a Pandas Dataframe.

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

In [2]:
```
df=pd.read_csv('titanic')
df
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

After loading the dataset and taking close look, we can see that there are 891 rows (examples) and 12 columns. Out of 12, 11 columns are independent features and 1 is a target feature or variable i.e. Survived

We need to think practically for each and every feature with respect to the relationship with our target feature and hence, deal accordingly.

For example – As you can see above, at 0 index there is one feature called 'PassengerID'. We can remove that column because it will not help in defining our target.

Data Analysis

In this part, we analyze the data by checking its data type, data info, missing values or null values, statistics, and std. deviation, value count, and uniqueness in every column etc.

In [4]:
```
df.head()
```
Out[4]:

	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

In [5]:
```
df.tail()
```
Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

In [6]:
```
df.shape
```
Out[6]: (891, 12)

In [7]:
```
df.columns
```
Out[7]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

- We can see that we have, 891 rows and 12 columns in our dataset

In [8]:
```
df.dtypes
```
Out[8]:
```
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
```

In [9]:
```
df.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
```

- The columns at index 0, 1, 2, 5, 6, and 7, 9 are with numeric (float or integer) datatypes, and at index 3, 4, 8, 9, and 10 columns consisting object data type. Hence, we need to convert them into numeric for our model.

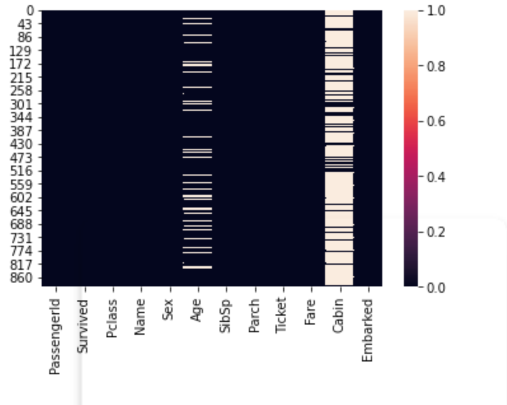
Checking Missing values

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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 [11]:
```
sns.heatmap(df.isnull())
```

Out[11]: <AxesSubplot:>



- There are null values present in the Age, Cabin, and the Embarked column, we will fill missing values in Age and Embarked column only because we are going to drop the Cabin column as it is not contributing to our dataset.

Dropping the column

- We are dropping the columns which are not contributing anything to our dataset and not necessary to keep those for model training.
- The columns we are going to drop are- Cabin, PassengerID, Name, Ticket, and Fare.

In [14]:
```
#we will drop few columns which are not contributing anything to our dataset
df=df.drop(columns=['Cabin','PassengerId','Name','Ticket','Fare'],axis=1)
```

In [15]:
```
df
```

Out[15]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
0	0	3	male	22.0	1	0	S
1	1	1	female	38.0	1	0	C
2	1	3	female	26.0	0	0	S
3	1	1	female	35.0	1	0	S
4	0	3	male	35.0	0	0	S
...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	S
887	1	1	female	19.0	0	0	S
888	0	3	female	NaN	1	2	S
889	1	1	male	26.0	0	0	C
890	0	3	male	32.0	0	0	Q

891 rows × 7 columns

- Now we can see above that we have dropped the column and we have only 6 features left along with 1 target column. (Total 7 columns we have)

About the columns

1. PassengerID- It is the unique ID of the passenger.
2. Survived- Total no. of passengers survived during the incident.
3. Pclass- It is the Passenger's class (1st, 2nd, or 3rd class).
4. Name- Name of individual Passenger.
5. Sex- Passenger's sex, whether (Male or Female).
6. Age- Age of passenger.
7. Sibsp- Number of Siblings or number of Spouses Aboard.
8. Parch- Number of Parents or number of children Aboard.
9. Ticket- Ticket number of passenger.
10. Fare- Ticket fare.
11. Cabin- Cabin number allotted to the respective passenger of 1st class.
12. Embarked- Port from where the passenger boarded the ship.

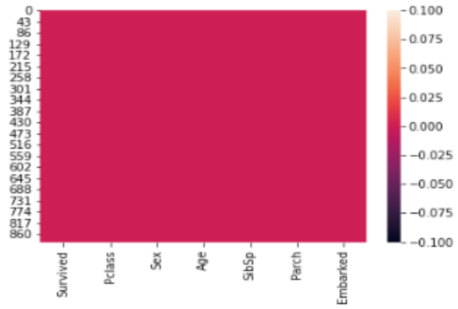
Handling Missing Values

Handling missing values

```
In [16]: df['Age'].mean()
Out[16]: 29.69911764705882

In [17]: df['Age']=df['Age'].fillna(df['Age'].mean())  #mean because its numerical column

In [18]: df['Embarked'].mode()[0]
Out[18]: 'S'

In [19]: df['Embarked']=df['Embarked'].fillna(df['Embarked'].mode()[0]) #mode beacuse its categoricalcolumn

In [20]: df.isnull().sum()
Out[20]: Survived    0
         Pclass      0
         Sex         0
         Age         0
         SibSp       0
         Parch       0
         Embarked    0
         dtype: int64
```

We can see that all the missing values are filled in Age and Embarked column

```
In [21]: sns.heatmap(df.isnull())
Out[21]: <AxesSubplot:>
```



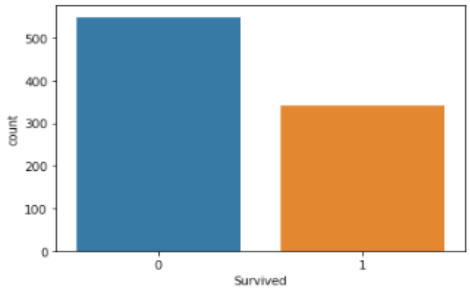
- We calculated the mean of the Age column as it includes numerical datatype and replaced all the missing values with the mean using the fillna method.
- Same way, we found the mode of the Embarked column as it includes object datatype and replaced all the missing values with the mode using fillna method.
- We can see that there are no null values in our dataset now.

Data Visualization

In [23]:

```
sns.countplot(df['Survived'])

# Graphical representation of people who died and survived the incident
```

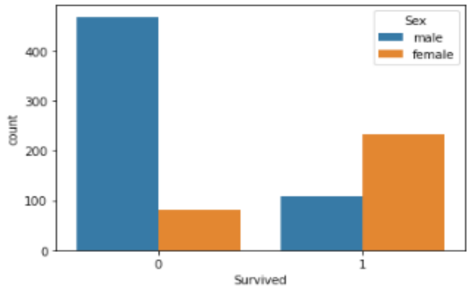
Out[23]: <AxesSubplot:xlabel='Survived', ylabel='count'>



In [24]:

```
sns.countplot(x ="Survived",hue = "Sex" ,data =df)

# Graphical representation of people who died and survived the incident with their genders
# We can see that most males died during the incident while most females were survived
```

Out[24]: <AxesSubplot:xlabel='Survived', ylabel='count'>

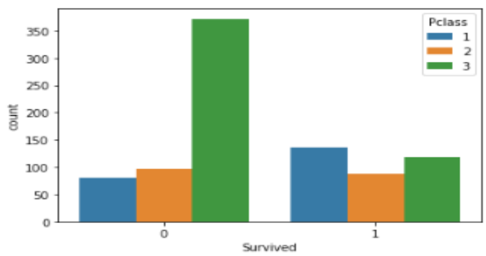


- In the first plot we can see that more people have died during this event and fewer people survived.
- In the second plot we can see that more males died as compared to females and more females survived as compared to males.

In [25]:

```
sns.countplot(x ="Survived",hue = "Pclass" ,data =df)

# Graphical representation of people who were travelling in Titanic different classes
# Most people who died were travelling in 3rd class or the Lower class
```

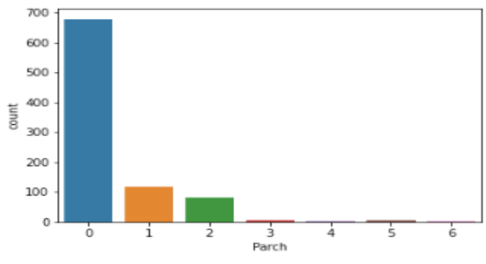
Out[25]: <AxesSubplot:xlabel='Survived', ylabel='count'>



In [26]:

```
sns.countplot(x ="Parch",data =df)

# Graphical representation of the passengers with number of Parents and Children
# We can see that most people are travelling with 0 Parents and Chidren
# Very few people are travelling with 3 to 6 Parents and Chidren combined
```

Out[26]: <AxesSubplot:xlabel='Parch', ylabel='count'>

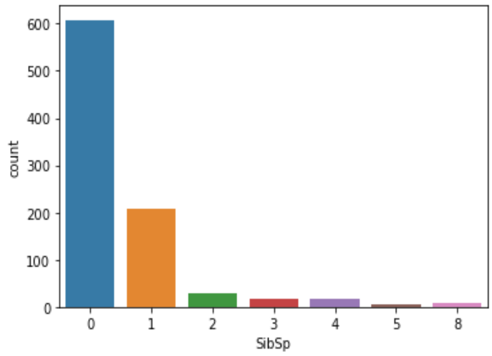


- From the first graph we can see that people from the third class died more as compared to other classes.

- From the second graph we can see that the number of travelers with no partner is more.

In [27]:
```
sns.countplot(x ="SibSp",data =df)

# Graphical representation of the passengers with number of Siblings and Spouse
# We can see that most people are travelling with 0 Siblings and Spouse
# Very few people are travelling with 2 to 6 Siblings and Spouse combined
```

Out[27]: <AxesSubplot:xlabel='SibSp', ylabel='count'>



- From the above plot we can see that most of the travelers were traveling with 0 siblings and spouse and very few people were traveling with 2 to 6 siblings and spouse combined.

Label Encoder

The performance of a machine learning model is determined not only by the model and hyperparameters but also by how different types of variables are processed and fed into the model. Because most machine learning models only accept numerical variables, categorical variables must be pre-processed. We must transform these categorical variables to integers in order for the model to comprehend and extract useful information.

Changing String to int values using LabelEncoder:

In [28]:
```
from sklearn.preprocessing import LabelEncoder
```

In [29]:
```
le=LabelEncoder()
df['Sex']=le.fit_transform(df['Sex'])
df['Embarked']=le.fit_transform(df['Embarked'])
df
```

Out[29]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
0	0	3	1	22.000000	1	0	2
1	1	1	0	38.000000	1	0	0
2	1	3	0	26.000000	0	0	2
3	1	1	0	35.000000	1	0	2
4	0	3	1	35.000000	0	0	2
...	...	...	...	...	...	...	...
886	0	2	1	27.000000	0	0	2
887	1	1	0	19.000000	0	0	2
888	0	3	0	29.699118	1	2	2
889	1	1	1	26.000000	0	0	0
890	0	3	1	32.000000	0	0	1

891 rows × 7 columns

- Here we can see that using the Label Encoder library from sklearn, we have changed the objective datatypes into numerical values.

Exploratory Data Analysis (EDA) and Visualisation Concluding Remarks

Exploratory Data Analysis (EDA) is a way of analyzing data sets in order to summarise their primary characteristics, which frequently involves the use of statistical graphics and other data visualization techniques. Univariate, Bivariate, and Multivariate Analysis are the three types of analysis.

Describing Dataset

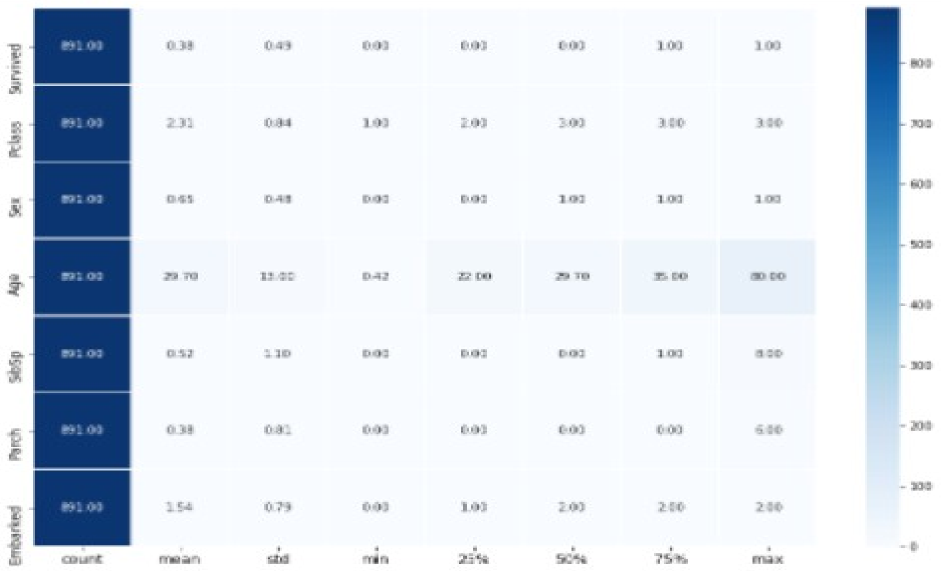
Describing dataset:

In [30]:
```
df.describe()
```

Out[30]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	0.647587	29.699118	0.523008	0.381594	1.536476
std	0.486592	0.836071	0.477990	13.002015	1.102743	0.806057	0.791503
min	0.000000	1.000000	0.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	0.000000	22.000000	0.000000	0.000000	1.000000
50%	0.000000	3.000000	1.000000	29.699118	0.000000	0.000000	2.000000
75%	1.000000	3.000000	1.000000	35.000000	1.000000	0.000000	2.000000
max	1.000000	3.000000	1.000000	80.000000	8.000000	6.000000	2.000000

In [31]:
```
plt.figure(figsize=(14,10))
sns.heatmap(df.describe().transpose(),annot=True,fmt='.2f',linewidth=0.4,cmap='Blues')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



- The description of the dataset gives important information such as mean, std deviation, 25%, 50% and 75%, as well as minimum and maximum value of each and every column.

Correlation of Features

checking correlation:

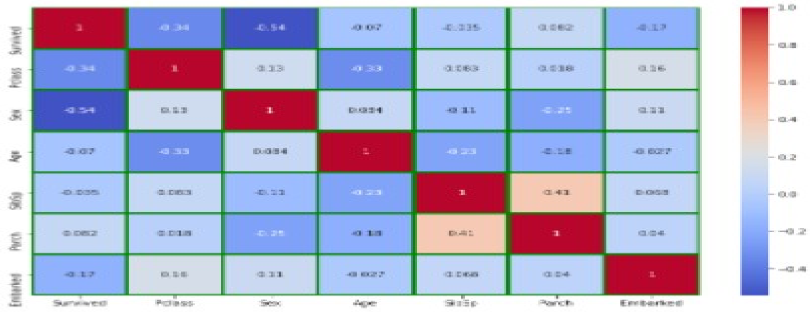
In [32]:
```
df.corr()
```

Out[32]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
Survived	1.000000	-0.338481	-0.543351	-0.069809	-0.035322	0.081629	-0.167675
Pclass	-0.338481	1.000000	0.131900	-0.331339	0.083081	0.018443	0.162098
Sex	-0.543351	0.131900	1.000000	0.084153	-0.114631	-0.245489	0.108262
Age	-0.069809	-0.331339	0.084153	1.000000	-0.232625	-0.179191	-0.026749
SibSp	-0.035322	0.083081	-0.114631	-0.232625	1.000000	0.414838	0.068230
Parch	0.081629	0.018443	-0.245489	-0.179191	0.414838	1.000000	0.039798
Embarked	-0.167675	0.162098	0.108262	-0.026749	0.068230	0.039798	1.000000

In [33]:
```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(),annot=True,linewidth=2,cmap='coolwarm',linecolor='green')
```

Out[33]: <AxesSubplot:>



Outcome of Correlation:

Max correlation: Parch

Min correlation: SibSp

- Checking the correlation of the dataset is very important as it gives us information about how each column is correlated to another column in our dataset. High correlation means it is contributing more to the target column and less correlation means it is not contributing to the target column.
- Here we can see that Parch has maximum correlation and SibSp has less correlation to our target Survived column.

Outlier(s): Detection & Removal

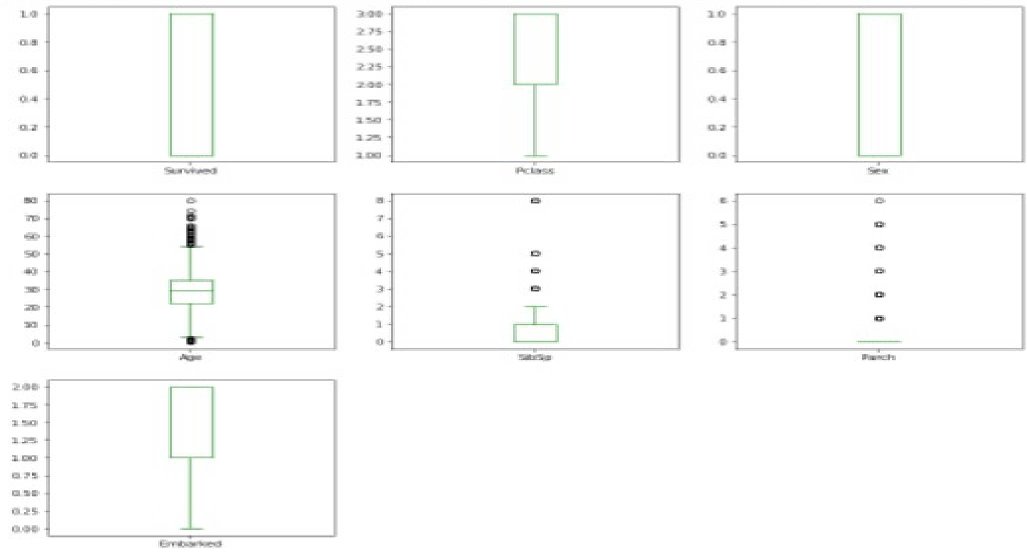
- Outliers are extreme values that are far from other observations. It can be detected and removed using either the ZScore or Interquartile Range (IQR) methods. We will use zscore for this purpose here.

Checking outliers:

In [34]:
```
df.columns
```

Out[34]: Index(['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked'], dtype='object')

In [35]:
```
df.plot(kind='box',subplots=True,layout=(4,3),figsize=(14,18),color='green')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()

# The Subplot shows boxplots of all the variable at once without passing boxplot function individually for each variable
```



- We have checked the outliers present in the dataset using boxplot method. There are many outliers present in the Age column and also in SibSp and Parch.

- To perform the model well, we have to remove these outliers in further steps.

Outlier Removal

Outlier removal:

```
In [36]: from scipy.stats import zscore

In [37]: z=np.abs(zscore(df))
         z.shape

Out[37]: (891, 7)

In [38]: threshold=3
         print(np.where(z>3))

(array([ 13,  16,  25,  50,  59,  68,  71,  86,  96, 116, 119, 159, 164,
       167, 171, 180, 182, 201, 233, 261, 266, 278, 324, 360, 386, 437,
       438, 480, 493, 541, 542, 567, 610, 630, 638, 672, 678, 683, 686,
       736, 745, 774, 787, 792, 813, 824, 846, 850, 851, 858, 863, 885],
      dtype=int64), array([5, 4, 5, 4, 4, 4, 4, 5, 3, 3, 4, 4, 4, 5, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 5, 4, 5, 5, 4, 3, 4, 4, 5, 5, 3, 5, 3, 5, 4, 4, 5, 3, 5, 4, 4,
       4, 4, 4, 4, 3, 5, 4, 5], dtype=int64))

In [39]: df_new=df[(z<3).all(axis=1)]
         print(df.shape)
         print(df_new.shape)

(891, 7)
(839, 7)
```

Percentage Data loss:

```
In [40]: loss_percent=(891-839)/891*100
         loss_percent

Out[40]: 5.836139169472503
```

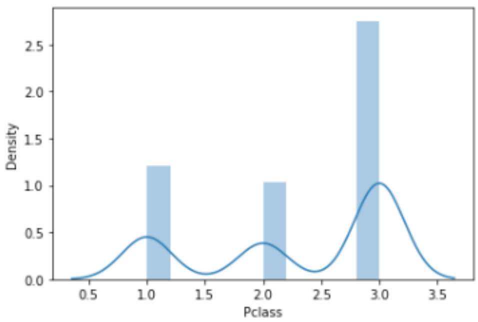
- We have imported zscore from scipy.stats for outlier removal.
- After removal of outliers, we have 839 rows left out of 891 rows but the columns are the same.
- We have calculated the percentage of data loss and found that we have lost 5.8% of data during outlier removal.

Data Distribution using Distplot

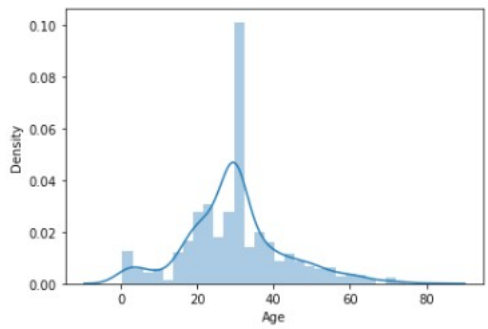
The distribution of data basically tells us about the mean, median, mode, maximum, minimum, standard deviation, and skewness of data. We can get and visualize them using the distribution plot of the seaborn library as shown below:

- We can see that the distribution plot shows, that the data is not distributed normally in all of the features and this is understandable because all of the features are of categorical type.

```
In [42]: sns.distplot(df["Pclass"])

         # Shows the high skewness of "Pclass"

Out[42]: <AxesSubplot:xlabel='Pclass', ylabel='Density'>
```

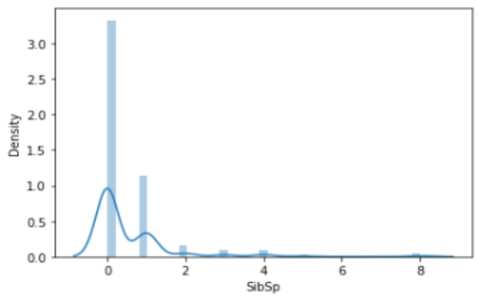



```
In [43]: sns.distplot(df["Age"])

         # Shows the high skewness of "Age"

Out[43]: <AxesSubplot:xlabel='Age', ylabel='Density'>
```



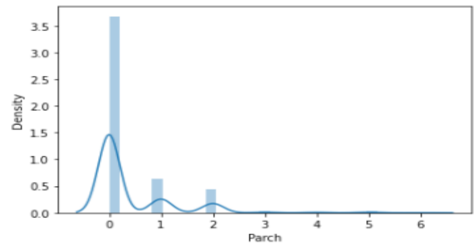
In [44]:
```
sns.distplot(df["SibSp"])

# Shows the high skewness of "SibSp"
```

Out[44]: <AxesSubplot:xlabel='SibSp', ylabel='Density'>

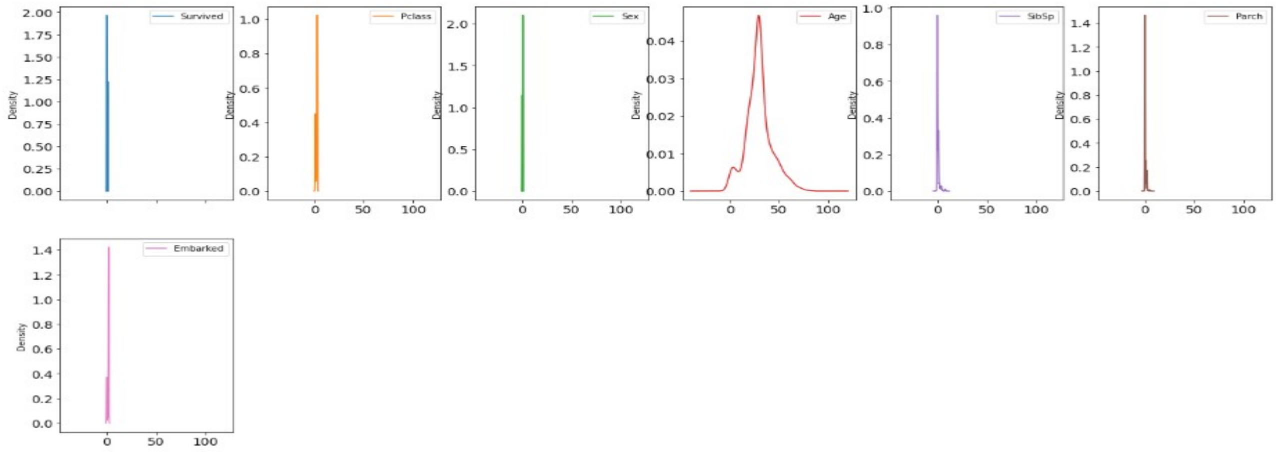


In [45]:
```
sns.distplot(df["Parch"])

# Shows the high skewness of "Parch"
```

Out[45]: <AxesSubplot:xlabel='Parch', ylabel='Density'>



In [46]:
```
df.plot(kind='kde',subplots=True,layout=(4,6),figsize=(22,22),fontsize=14)
plt.show()

# Plotting the Skewness of all variables in dataset with subplots
```



Skewness: Detection & Treatment

Checking skewness:

In [41]:
```
df.skew()

# As we can see most of the Variables have high skewness which needs to be treated
```

Out[41]:
```
Survived    0.478523
Pclass     -0.630548
Sex        -0.618921
Age         0.434488
SibSp       3.695352
Parch       2.749117
Embarked   -1.264823
dtype: float64
```

We are taking Skewness threshold as +/-0.5. From the above observation, the columns falls under high skewness are: sibSp and Parch and Embarked

- We can see that the columns with high skewness are SibSp, Parch, and Embarked.
- We are taking a threshold of +/-0.5 here.

Separate Input and Output/Target Variable and Scale Feature Data for Model Training

- Now, we can separate the features into the input as X and output/target as Y to continue further with data preparation.

Removing skewness:

Dividing data in x and y

In [48]:
```
x=df_new.drop(columns=['Survived'])
y=df_new['Survived']
```

In [49]:
```
x.dtypes
```

Out[49]:
```
Pclass        int64
Sex           int32
Age         float64
SibSp         int64
Parch         int64
Embarked      int32
dtype: object
```

In [50]:
```
from sklearn.preprocessing import power_transform

# Importing the power_transform function to treat the over skewness in dataset
```

In [51]:
```
x=power_transform(x,method='yeo-johnson')
x
```

Out[51]:
```
array([[ 0.89050277,  0.73255036, -0.62045292,  1.54523598, -0.49848718,
         0.61962311],
       [-1.43471107, -1.36509385,  0.68979747,  1.54523598, -0.49848718,
        -1.75376156],
       [ 0.89050277, -1.36509385, -0.28420981, -0.63447179, -0.49848718,
         0.61962311],
       ...,
       [ 0.89050277, -1.36509385,  0.02105288,  1.54523598,  2.03470008,
         0.61962311],
       [-1.43471107,  0.73255036, -0.28420981, -0.63447179, -0.49848718,
        -1.75376156],
       [ 0.89050277,  0.73255036,  0.20851729, -0.63447179, -0.49848718,
        -1.18678987]])
```

- Above we can see that we have removed skewness using power transform method.

Standard Scaler

- Scaling data for model training refers to normalizing the range of independent variables or features of data. Here we are using StandardScaler for this purpose

Applying Standard Scaler.

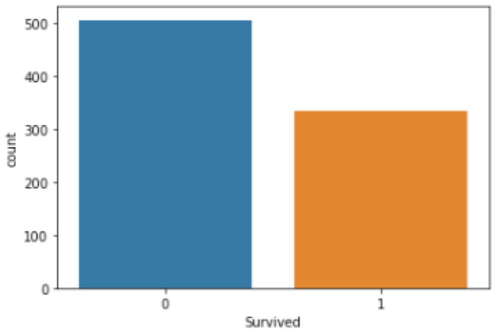
In [52]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_t=sc.fit_transform(x)
x_t
```

Out[52]:
```
array([[ 0.89050277,  0.73255036, -0.62045292,  1.54523598, -0.49848718,
         0.61962311],
       [-1.43471107, -1.36509385,  0.68979747,  1.54523598, -0.49848718,
        -1.75376156],
       [ 0.89050277, -1.36509385, -0.28420981, -0.63447179, -0.49848718,
         0.61962311],
       ...,
       [ 0.89050277, -1.36509385,  0.02105288,  1.54523598,  2.03470008,
         0.61962311],
       [-1.43471107,  0.73255036, -0.28420981, -0.63447179, -0.49848718,
        -1.75376156],
       [ 0.89050277,  0.73255036,  0.20851729, -0.63447179, -0.49848718,
        -1.18678987]])
```

Verifying Class Imbalance in the target column

In [53]:
```
y.value_counts()
```
Out[53]:
```
0    505
1    334
Name: Survived, dtype: int64
```

In [54]:
```
sns.countplot(y)
```
Out[54]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



- Using the value counts function we have checked the numbers of people who survived and numbers of people who died.
- We can see that 505 people died and 334 people survived.
- Using the count plot also we can easily see that there is a huge class imbalance issue here.
- In this case we have to balance the class before building the model.
- We will use SMOTE method to balance the target variable

SMOTE

- We have imported SMOTE from the imblearn oversampling to balance the dataset
- We can see below that the target column is balanced now.
- The number of survived and died people are equal now.
- Treating unbalanced data is necessary before the model training as unbalanced data can predict the biased result. So the accuracy of the model will be not true.

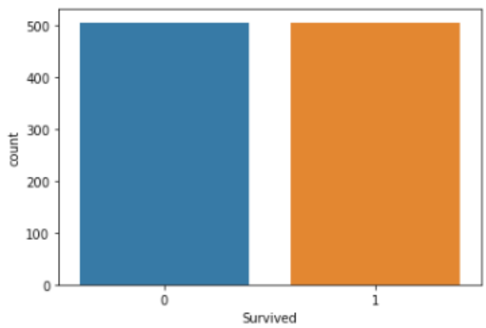
Applying SMOTE to balance the target column data:

In [56]:
```
from imblearn.over_sampling import SMOTE
```

In [57]:
```
sm=SMOTE()
x,y=sm.fit_resample(x_t,y)

#Resampling the x_t and y to make the class balanced and saving the new x and y in x and y again
```

In [58]:
```
y.value_counts()              # Now we can see that both the classes are balanced without losing any data
```
Out[58]:
```
0    505
1    505
Name: Survived, dtype: int64
```

In [61]:
```
sns.countplot(y)
```
Out[61]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



Prepare Dataset for Model Training

Preparing dataset Model training is the one of core parts of machine learning model building and includes different types of data modification and transformation to achieve better model performance.

Model Training: Finding the best model

The models that I have decided to train for this dataset are LogisticRegression, SVC (Support Vector Classifier), KNeighboursClassifier, DecisionTreeClassifier, RandomForestClassifier, GaussianNB models. The goal here is to find the best hyper-tuned models for further processing.

Importing Libraries

In [64]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix,classification_report
```

Applying loop for getting best random_state

Finding best random State

In [65]:
```
maxAccu=0
maxRS=0

for i in range(1,500):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=i)
    lr=LogisticRegression()
    lr.fit(x_train,y_train)
    predlr=lr.predict(x_test)
    acc=accuracy_score(y_test,predlr)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i

print('Best accuracy score is: ', maxAccu, 'At random state: ', maxRS)

```

```
Best accuracy score is:  0.8663366336633663 At random state:  287
```

we found our best random state 287, we will create our train test split using this random state

- We found our best random state 287 which is giving us high accuracy of 86%
- Now, we are going to use this random state for creating train_test_split

Prepare Model List and Test to get Best Model

Creating train test split

In [66]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=287)
```

In [67]:
```
x_train.shape
```
Out[67]: (808, 6)

In [68]:
```
y_train.shape
```
Out[68]: (808,)

In [69]:
```
x_test.shape
```
Out[69]: (202, 6)

In [70]:
```
y_test.shape
```
Out[70]: (202,)

- We have created train test_test_split using the random state 287
- We will train all our above-listed model with this train test split
- After creating train_test_split we have checked the shape of the x_train, x_test, y_train and y_test.

Importing all the models

Checking different model building techniques

In [79]:

```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report,r2_score
from sklearn.model_selection import cross_val_score
```

Model Building

A. Logistic Regression

Logistic regression

72]:

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

lr.fit(x_train,y_train)
predlr=lr.predict(x_test)
print("Accuracy score:", accuracy_score(y_test,predlr)*100)
print(confusion_matrix(y_test,predlr))
print(classification_report(y_test,predlr))
```

```
Accuracy score: 86.63366336633663
[[87 12]
 [15 88]]
              precision    recall  f1-score   support

           0       0.85      0.88      0.87        99
           1       0.88      0.85      0.87       103

    accuracy                           0.87       202
   macro avg       0.87      0.87      0.87       202
weighted avg       0.87      0.87      0.87       202
```

- Logistic regression model is giving us 86.6% accuracy score

B. Support Vector Classifier

Support vector classifier:

n [74]:

```
from sklearn.svm import SVC

svc=SVC()
svc.fit(x_train,y_train)
predsvc=svc.predict(x_test)
print("Accuracy score: ", accuracy_score(y_test,predsvc)*100)
print(confusion_matrix(y_test,predsvc))
print(classification_report(y_test,predsvc))
```

```
Accuracy score:  85.14851485148515
[[91  8]
 [22 81]]
              precision    recall  f1-score   support

           0       0.81      0.92      0.86        99
           1       0.91      0.79      0.84       103

    accuracy                           0.85       202
   macro avg       0.86      0.85      0.85       202
weighted avg       0.86      0.85      0.85       202
```

- Support vector classifier model is giving us an accuracy score of 85.1%

C. KNeighbors Classifier

KNeighborsClassifier

[75]:
```python
from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
predknn=knn.predict(x_test)
print("Accuracy Score: ", accuracy_score(y_test,predknn)*100)
print(confusion_matrix(y_test,predknn))
print(classification_report(y_test,predknn))
```

```
Accuracy Score:  85.14851485148515
[[84 15]
 [15 88]]
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        99
           1       0.85      0.85      0.85       103

    accuracy                           0.85       202
   macro avg       0.85      0.85      0.85       202
weighted avg       0.85      0.85      0.85       202
```

- KNeighbors classifier model is giving us an accuracy score of 85.1%

D. Decision Tree Classifier

DecisionTreeClassifier

[76]:
```python
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(x_train,y_train)
preddtc=dtc.predict(x_test)
print("Accuracy Score: ", accuracy_score(y_test,preddtc)*100)
print(confusion_matrix(y_test,preddtc))
print(classification_report(y_test,preddtc))
```

```
Accuracy Score:  83.66336633663366
[[81 18]
 [15 88]]
              precision    recall  f1-score   support

           0       0.84      0.82      0.83        99
           1       0.83      0.85      0.84       103

    accuracy                           0.84       202
   macro avg       0.84      0.84      0.84       202
weighted avg       0.84      0.84      0.84       202
```

- Decision Tree classifier model is giving us an accuracy score of 83.6%

E. Random Forest Classifier

RandomForestClassifier

78]:
```python
from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier()
rf.fit(x_train,y_train)
predrf=rf.predict(x_test)
print("Accuracy score:", accuracy_score(y_test,predrf)*100)
print(confusion_matrix(y_test,predrf))
print(classification_report(y_test,predrf))
```

```
Accuracy score: 85.14851485148515
[[83 16]
 [14 89]]
              precision    recall  f1-score   support

           0       0.86      0.84      0.85        99
           1       0.85      0.86      0.86       103

    accuracy                           0.85       202
   macro avg       0.85      0.85      0.85       202
weighted avg       0.85      0.85      0.85       202
```

- Random Forest classifier model is giving us an accuracy score of 85.1%

F. GaussianNB

GaussianNB

In [80]:
```python
from sklearn.naive_bayes import GaussianNB

gnb=GaussianNB()
gnb.fit(x_train,y_train)
predgnb=gnb.predict(x_test)
print("Accuracy Score is: ",accuracy_score(y_test,predgnb)*100)
print(confusion_matrix(y_test,predgnb))
print(classification_report(y_test,predgnb))
```

```
Accuracy Score is:  84.65346534653465
[[84 15]
 [16 87]]
              precision    recall  f1-score   support

           0       0.84      0.85      0.84        99
           1       0.85      0.84      0.85       103

    accuracy                           0.85       202
   macro avg       0.85      0.85      0.85       202
weighted avg       0.85      0.85      0.85       202
```

- GaussianNB model is giving us an accuracy score of 84.6%

CROSS VALIDATION SCORE

CROSS VALIDATION:

In [81]:
```python
from sklearn.model_selection import cross_val_score
```

In [105]:
```python
scr1=cross_val_score(lr,x,y,cv=5)
print("Cross validation score of Logistic Regression model is: ", scr1.mean())

scr2=cross_val_score(svc,x,y,cv=5)
print("Cross validation score of support vector classifier model is: ", scr2.mean())

scr3=cross_val_score(knn,x,y,cv=5)
print("Cross validation score of KNeighbors classifier model is: ", scr3.mean())

scr4=cross_val_score(dtc,x,y,cv=5)
print("Cross validation score of Decision tree classifier model is: ", scr4.mean())

scr5=cross_val_score(rf,x,y,cv=5)
print("Cross validation score of random Forest classifier model is: ", scr5.mean())

scr6=cross_val_score(gnb,x,y,cv=5)
print("Cross validation score of GuassianNB model is: ", scr6.mean())

```

```
Cross validation score of Logistic Regression model is:  0.789108910891089
Cross validation score of support vector classifier model is:  0.799009900990099
Cross validation score of KNeighbors classifier model is:  0.805940594059406
Cross validation score of Decision tree classifier model is:  0.7960396039603961
Cross validation score of random Forest classifier model is:  0.803960396039604
Cross validation score of GuassianNB model is:  0.7821782178217823
```

KNeighbors classifier model is performing well as it has high cross val score of 80.6% and high accuracy score of 85.1%. (or least diff between cross val score and accuracy score. i.e, 4.6%)

we could chose Logistic regression model also because it has highest accuracy score of 86.6% but the cross val score is less and because of this the difference between cross val score and accoracy score is high (i.e, 7.7%)

- After getting accuracy from all the models we have also checked the cross-validation score of all the models.
- We have given CV=5 so that every model will predict accuracy 5 times and give us an average of the 5 accuracies.
- In this case, KNeighbors Classifier model is performing well as it has a high cross-validation score of 80.6% and high accuracy score of 85.1% (or the least difference between cross-validation score and accuracy score. i.e. 4.6%).
- We could choose the Logistic regression model also because it has the highest accuracy score of 86.6% but the cross-validation score is less and because of this the difference between the cross-validation score and accuracy score is high (i.e., 7.7%)

HYPERPARAMETER TUNNING USING GRID SEARCH CV

Hyper parameter tunning

In [109]:
```
from sklearn.model_selection import GridSearchCV
# Importing the GridSearchCV to get the best parameters of the KNeighbors Classifier model

#creating parameter list to pass in GridSearchCV
#parameters are different for different models
#for KNeighbors classifier i am using these

parameters={"n_neighbors":np.arange(2,10),
           "weights":["uniform","distance"],
           "algorithm":['auto', 'ball_tree', 'kd_tree', 'brute'],
           "leaf_size":np.arange(2,10)}

# Setting the Parameters to apply to GridSearchCV to get the best parameter score
```

In [110]:
```
GCV=GridSearchCV(KNeighborsClassifier(),parameters,cv=5,scoring="accuracy")

GCV.fit(x_train,y_train)         #fitting data in model

GCV.best_params_                 #Printing the best parameters found by GridSearchCV
```

Out[110]:
```
{'algorithm': 'brute', 'leaf_size': 2, 'n_neighbors': 8, 'weights': 'uniform'}
```

In [111]:
```
GCV_pred=GCV.best_estimator_.predict(x_test)   #predicting with best parameters

accuracy_score(y_test,GCV_pred)                  #checking final accuracy
```

Out[111]:
```
0.8465346534653465
```

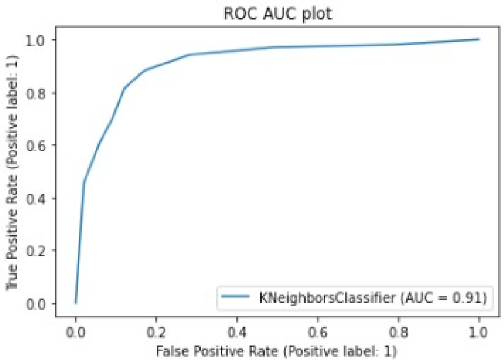
- In machine learning, optimizing or adjusting hyperparameters is a problem in choosing the best set of hyperparameters for the learning algorithm. Hyperparameters are the parameters whose values are used to control the learning process. In contrast, the values of other parameters (usually node weights) are learned.
- The same type of machine learning model may require different constraints, weights, or learning rates to generalize to different data patterns. These measurements are called hyperparameters and need to be adjusted so that the model can optimally solve machine learning problems. Hyperparameter optimization finds a tuple of hyperparameters that produces an optimal model that minimizes the predefined loss function of a given independent data. The objective function takes a tuple of hyperparameters and returns the associated loss. Cross-validation is often used to estimate this generalization performance.
- Hyperparameters are important as they manipulate the general behavior of a gadget studying model. The remaining purpose is to discover the most advantageous aggregate of hyperparameters that minimizes a predefined loss feature to present higher results.
- For KNeighbors classifier model we have done Hyperparameter tunning and found that the accuracy score is 84.6%

PLOT ROC-AUC CURVE AND DETERMINE SCORES

Measuring performance is an important task in machine learning. Therefore, for classification issues, you can rely on the AUC-ROC curve. If you need to see or visualize the performance of a multiclass classification problem, use the AUC curve (under-curve area) ROC curve (receiver operating characteristic). This is one of the most important assessments to check the performance of your classification model. It is also referred to as AUROC (area under receiver operating characteristics).

ROC AUC curve

In [112]:
```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(GCV.best_estimator_,x_test,y_test)
plt.title("ROC AUC plot")
plt.show()
```



Auc score is 91.0% and final accuracy is 84.6%

As per the ROC AUC curve score, KNeighbors Classifier is the best-fit model.

Model Selection: The Final Model

In this step, we will save or serialize the final model which gives the highest performance into an object or pickle file.

Saving the model in pickle format

In [113]:
```
import joblib
joblib.dump(GCV.best_estimator_,"Titanic_final.pkl")
```

Out[113]: ['Titanic_final.pkl']

In [ ]:

Conclusion

The final model performance is good with **Accuracy 84.6%** and the **ROC AUC score is 91%**

Concluding Remarks

In order to extend this project to get a better final model, I would like to see if similar data can be used for further analysis. Doing so will allow you to further test your model and gain more insight into what works best in real-world situations. My model worked perfectly with my dataset, so it's worth asking if the dataset could be at risk. Therefore, testing the model on a different dataset may provide further validation.