Titanic - Machine Learning from Disaster

Introduction

Data Science:- Data Science is the field in which Mathematics and Statistics, Specialized Programming, advance analytics artificial intelligence and Machine Learning with specific subject matter expertise to uncover actionable insights hidden in an organizations data. These insights can be used to guide decision making and strategic planning.

Data science lifecycle involves various roles, tools, and processes which enables to analysts to glean actionable insights . Typically, a Data Science project undergoes the following stages:-

1. **Data Ingestion:-** The lifecycle begins with the data collection – both raw structured and unstructured from all relevant sources using a variety of method. These methods can include manual entry, Web Scraping, a real time streaming data from systems and devices.
2. **Data Storage and Data Processing:-** Since data can have different formats and structures, companies need to consider different storage systems based on the type of data that needs to be captured. Data management teams help to set standards around data storage and structure, which facilitate workflows around analytics, machine learning and deep learning models.
3. **Data analysis:-** Data Scientists conduct an exploratory data analysis to examine biases, patterns, ranges, and distributions of values within the data. This data analytics exploration drives hypothesis generation for a/b testing.
4. **Communicate:-** Finally, insights are presented as reports and other data visualizations that make the insights—and their impact on business—easier for business analysts and other decision-makers to understand.

Data science tools:- Data scientists rely on popular programming languages to conduct exploratory data analysis and statistical regression. These open source tools support pre-built statistical modeling, machine learning, and graphics capabilities.

R Studio:- An open source programming language and environment for developing statistical computing and graphics.

Python:- It is a dynamic and flexible programming language. The Python includes numerous libraries, such as NumPy, Pandas, Matplotlib, for analyzing data quickly.

To facilitate sharing code and other information, data scientists may use GitHub and Jupyter notebooks. Some data scientists may prefer a user interface, and two common enterprise tools for statistical analysis include:

SAS:- A comprehensive tool suite, including visualizations and interactive dashboards, for analyzing, reporting, data mining, and predictive modeling.

IBM SPSS:- Offers advanced statistical analysis, a large library of machine learning algorithms, text analysis, open source extensibility, integration with big data, and seamless deployment into applications.

Machine Learning

Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior.

Artificial intelligence systems are used to perform complex tasks in a way that is similar to how humans solve problems.

Machine Learning techniques are divided mainly into the following 3 categories:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

Supervised Learning:- Supervised Learning is also known as Supervised Machine Learning, is defined by its use of labeled datasets to train algorithms to classify data or predict outcomes accurately. As input data is fed into the model, the model adjusts its weights until it has been fitted appropriately.

Unsupervised Learning:- Unsupervised Learning also known as Unsupervised Machine Learning, uses machine learning algorithms to analyze and cluster

unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention.

Reinforcement Learning:- Reinforcement machine learning is a machine learning model that is similar to supervised learning, but the algorithm isn't trained using sample data. This model learns as it goes by using trial and error. A sequence of successful outcomes will be reinforced to develop the best recommendation or policy for a given problem.

Problem Statement:- Problem Statement: The Titanic Problem is based on the sinking of the 'Unsinkable' ship Titanic in early 1912. It gives you information about multiple people like their ages, sexes, sibling counts, embankment points, and whether or not they survived the disaster. Based on these features, you have to predict if an arbitrary passenger on Titanic would survive the sinking or not.

Dataset:- We will be using One dataset which contains 10692 records in which 891 rows and 12 Columns by which we have to predict if an arbitrary passenger on Titanic would survive the sinking or not. Here we will use Regression techniques as the output predicted will be of a continuous type.

The Dataset contains the following features:-

PassengerId:- It contains the data or Sr. no. of the person

Survived:- It have 2type of data 0 and 1 which is Survived or Not Survived

Pclass:- It tells the journey Class of the Passenger

Name:-It contains the name of the Traveler

Sex:- It tell us weather the traveler is male or female

Age:- Its about the Age of the Traveler

SibSp:- It tell the Number of Siblings

Parch:- It tell us Of parents and children aboard the titanic

Ticket:- It Contains the ticket Number of the Passengers

Fare:- It contains the Total fare of the passengers

Cabin:- Its shows the Cabin Number of the Passengers

Embarked:- It tell Port of Embarkation

C = Cherbourg, Q = Queenstown, S = Southampton

Article Contents This article explains the whole process to build a machine learning model. Contents mentioned below have various steps that we will go through, throughout the project –

➢ Data Collection and Pre-processing

➢ Exploratory data analysis

➢ Encoding the data (Label Encoder)

➢ Outlier detection and skewness treatment

➢ Scaling the data (Standard scaler)

➢ Model Building and Evaluation

➢ Cross-validation of the selected model

➢ Model hyper-tuning

➢ Saving the final model and prediction using saved model

So, Let's get check out with our dataset. We will explore each and every feature of the dataset.

Data Collection and Pre-processing:-

We load the dataset using Pandas library in Jupyter Notebook.

```
In [2]: Data=pd.read_csv("C:/Users/sunee/Desktop/Data trained Projects/titanic_train.csv")

In [3]: Data
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	N
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N
...	...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	N
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	N

Here we are analyzing the dataset, and also we will check the first look of rows and columns of data.

```
In [4]: Data.head()
```

Out[4]:

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

We will check the first five rows and columns of data.

After observing the above five rows of train and test data, we can mention the below points.

PassengerId:- It contains the data or Sr. no. of the person

Survived:- It have 2type of data 0 and 1 which is Survived or Not Survived

Pclass:- It tells the journey Class of the Passenger

Name:-It contains the name of the Traveler

Sex:- It tell us weather the traveler is male or female

Age:- Its about the Age of the Traveler

SibSp:- It tell the Number of Siblings

Parch:- It tell us Of parents and children aboard the titanic

Ticket:- It Contains the ticket Number of the Passengers

Fare:- It contains the Total fare of the passengers

Cabin:- Its shows the Cabin Number of the Passengers

Embarked:- It tell Port of Embarkation

C = Cherbourg, Q = Queenstown, S = Southampton

We will proceed further to explore the dataset.

Let's check the whole summary of the dataset.
We ran a simple command data.info () and got the whole information about the dataset.

```
In [9]: Data.info()
```

```
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
```

Here from the data Summary, we observe that, there are total 12 columns, 5 with object data types and 5 with integer data type and also having 2 float data type i.e., target column.

Now we will check Min, Max, Mean.

In [5]:
```
Data.min()
```

Out[5]:
```
PassengerId                    1
Survived                       0
Pclass                         1
Name           Abbing, Mr. Anthony
Sex                         female
Age                           0.42
SibSp                            0
Parch                            0
```

In [6]:
```
Data.max()
```

Out[6]:
```
PassengerId                          891
Survived                               1
Pclass                                 3
Name           van Melkebeke, Mr. Philemon
Sex                                 male
Age                                 80.0
SibSp                                  8
Parch                                  6
```

In [7]:
```
Data.mean()
```

Out[7]:
```
PassengerId    446.000000
Survived         0.383838
Pclass           2.308642
Age             29.699118
SibSp            0.523008
Parch            0.381594
```

Now we will check the null or missing values in the dataset with the command Data.isnull and it will give us the following results.

In [15]:
```
#Checking All NULl Values in dataset
Data.isnull()
```

Out[15]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	True	False
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	False	False	True	False	False	False	False	True	False

Now we will check the null or missing values in the dataset with the command train.isnull(). sum() and it will give us the following results.

In [16]:
```
Data.isnull().sum()
```

Out[16]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
```

Here, we have missing values in Age column, Cabin column and Embarked column. We will handle these missing values using mode Imputation.

In [19]:
```
Data=Data.drop(columns="Cabin", axis=1)
```

Droping "cabin" column from dataset because most of the values (687) are missing and we cannot take mean, mode etc of it. So, we will drop this column.

In [21]:
```
# Filling Null Values of "Age" column in dataset by mean value
Data["Age"].fillna(Data["Age"].mean(), inplace=True)
```

Filling Null Values of "Age" column in dataset by mean value

Filling Null Values of "Embarked" column in dataset by mode value Checking all values of "Embarked" column

In [25]:
```
#Filling Null Values of "Embarked" column in dataset by mode value
Data["Embarked"].fillna(Data["Embarked"].mode()[0], inplace=True)
```

Now we will again check the null values.

In [27]:
```
#Checking Again for Null Values after Handling it
Data.isnull().sum()
```

Out[27]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
```

Describing Data : works only on continuous column and do not work on categorical column

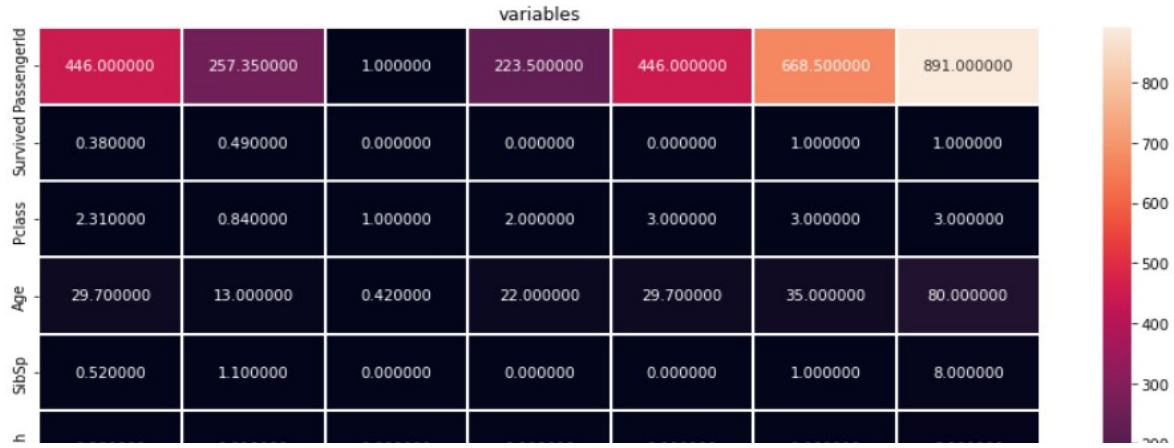
In [29]:
```
#Descriptive Statistics
# Describing Data : works only on continuous column and do not work on categorical column
Data.describe()
```

Out[29]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	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	13.002015	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	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200

Checking Description through heatmap also.

In [30]:
```
#Checking Description through heatmap also.
plt.figure(figsize = (15,7))
sns.heatmap(round(Data.describe()[1:].transpose(),2),linewidth = 2,annot = True,fmt = 'f')
plt.xticks(fontsize = 18)
plt.xticks(fontsize = 12)
plt.title('variables')
plt.show()
```



Outcome of Describe of Datasets:-

We are Determining Mean, Standard Deviation, Minimum and Maximum Values of each column.

The summary of this dataset looks good as there are no negative/ invalid value present.

Total No of Rows: 891 Total No. of Columns: 12

Passenger Id:-

Mean= 446.000000, std= 257.350000, Min= 1.000000, Max=891.000000

Survived: -

Mean= 0.380000, std= 0.490000, Min= 0.000000, Max= 1.000000

PClass:-

Mean = 2.310000, std= 0.840000, Min= 1.000000, Max= 3.000000

Age:-

Mean= 29.700000, std= 13.000000, Min= 0.420000, Max= 80.000000

Minimum Age is 0.42 which means approx 4months and Maximum Age is 80. Mean of age is 29

SibSp:-

Mean= 0.520000, std= 1.100000, Min= 0.000000, Max= 8.000000

Parch:-

Mean= 0.380000, std= 0.810000, Min= 0.000000, Max= 6.000000

Fare:-

Mean= 32.200000, std= 49.690000, Min= 0.000000, Max= 512.330000 Minimum Fare is 0 which means travelling in free and Maximum Fare is 512.

Some columns : Name, Sex, Ticket, Cabin and Embarked are not included in the Describe Method

As these are categorical column and Describe Method works only on continuous column

We observe that the dataset seems to be having more outliers as well as skewness in the data.

The column Age and Fare has huge outliers from the max of 80 and 512 respectively which is quite far from

Their mean with their second quantile median(50%).

Data Visualization:-

Using Countplot for categorical columns we will see here for some columns

Count Plot for "Survived" column

Not Survived People are more than Survived People.

Here 0= not survived and 1= survived

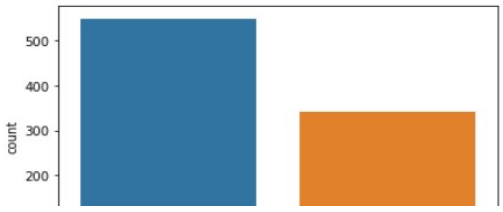
Total no of Survived= 342 and not survived= 549

In [32]:
```
#    Data Visualization
#   Using Countplot for categorical columns

#Count Plot for "Survived" column
print(Data["Survived"].value_counts())
sns.countplot("Survived",data = Data)
```

```
0    549
1    342
Name: Survived, dtype: int64
```

Out[32]:



Count Plot of "Pclass" Column

In [38]:
```
#Count Plot of "Pclass" Column
print((Data["Embarked"].unique()),"\n")
print("Total no:","\n")
print(Data["Embarked"].value_counts())

sns.countplot("Embarked",  data = Data)
```

```
['S' 'C' 'Q']

Total no:

S    646
```

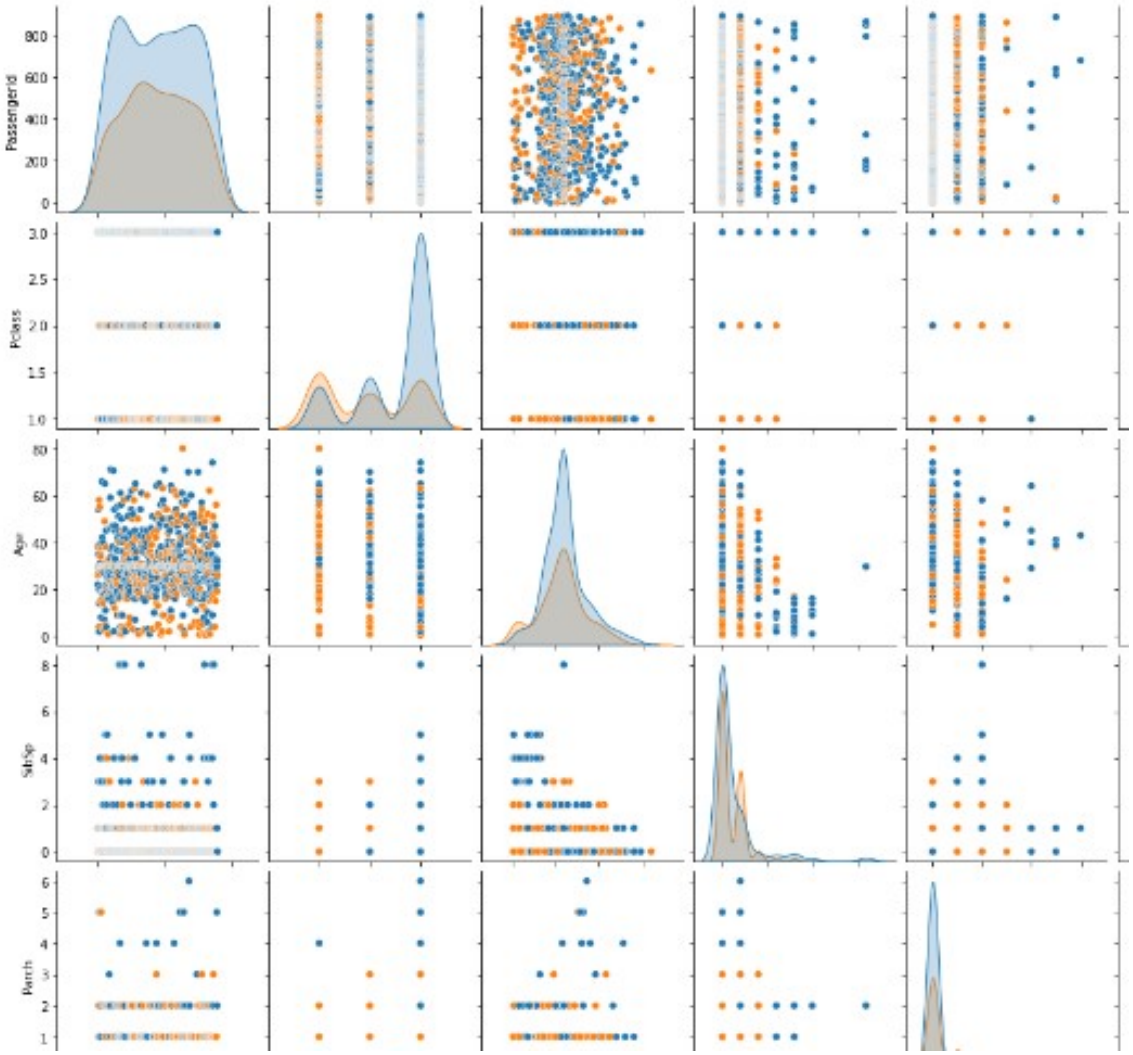
Count Plot of "Embarked" Column

In [41]:
```
#Count Plot for Survived people in "Embarked" column
sns.countplot("Parch", hue="Survived", data = Data)
```

Out[41]:

In [42]:
```
plt.figure(figsize = (15,10))
sns.pairplot(data = Data , hue = "Survived")
```

Out[42]:



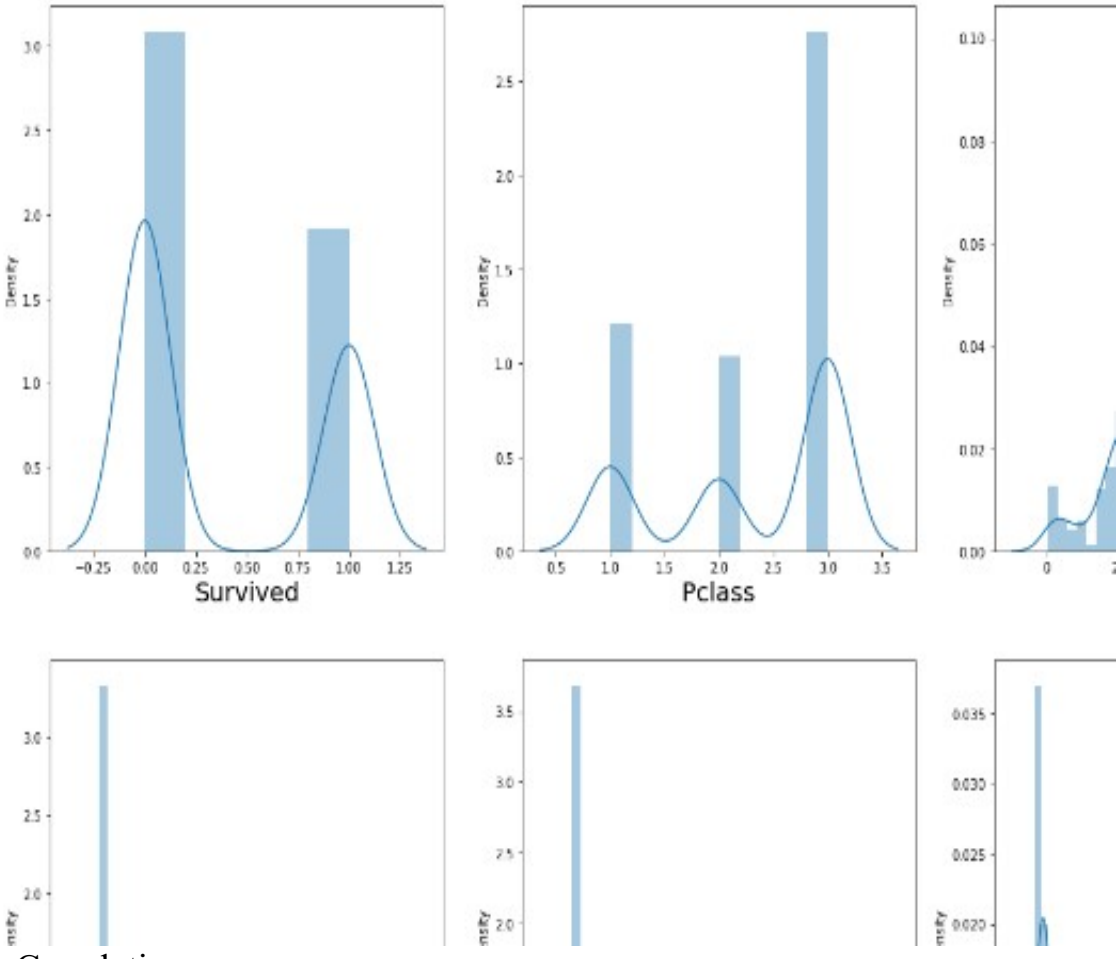
This pair plot gives the relation between the columns which are plotted on the basis of target variable 'Survived'.

Normal Distribution Curve:-

```
In [43]:
#     This pair plot gives the relation between the columns which are plotted on the basis of target variable 'Survi
#      Normal Distribution Curve:

collist=[ 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch',  'Fare',]
plt.figure(figsize=(20,25))
plotnumber = 1

for column in Data[collist]:
    if plotnumber<=9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(Data[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



Checking Correlation

In [44]:
```
#Checking Correlation
Data.corr()
```

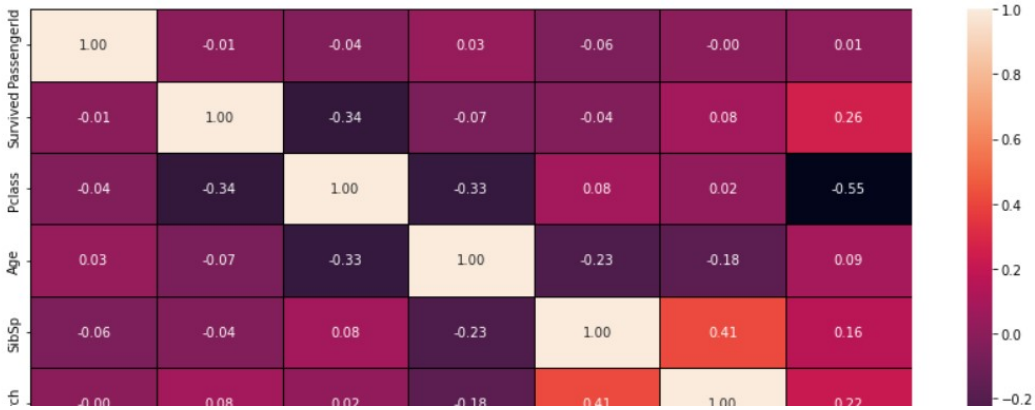
Out[44]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.033207	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.069809	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.331339	0.083081	0.018443	-0.549500
Age	0.033207	-0.069809	-0.331339	1.000000	-0.232625	-0.179191	0.091566
SibSp	-0.057527	-0.035322	0.083081	-0.232625	1.000000	0.414838	0.159651

Checking Correlation with heatmap

In [46]:
```
#Checking Correlation with heatmap
plt.figure(figsize = (15,7))
sns.heatmap(Data.corr(),annot = True, linewidth = 0.5, linecolor = 'black', fmt = '.2f')
```

Out[46]:



Outcome of Correlation:-

PassengerId has 1% correlation with the target column which can be considered as positively correlated

PClass has -55% correlation with the target column which can be considered as highly negatively correlated

Age has 9% correlation with the target column which can be considered as positively correlated

SibSp has 16% correlation with the target column which can be considered as positively correlated

Parch has 22% correlation with the target column which can be considered as positively correlated

Fare has 100% correlation with the target column which can be considered as positively correlated

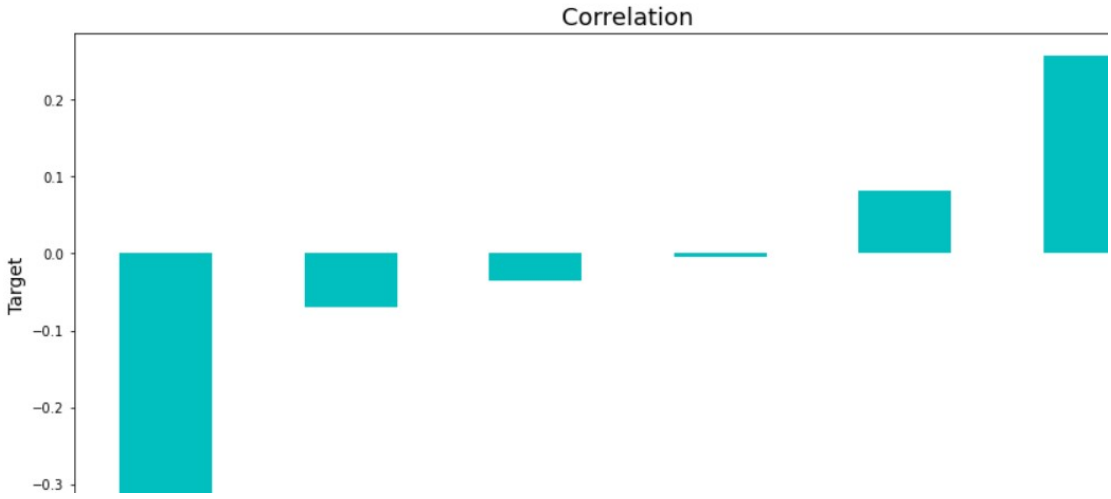
SibSp and Parch are positively correlated with each other

Max Correlation: Fare

Min Correlation: PClass

In [48]:
```
#Checking correlation with barplot

plt.figure(figsize=(15,7))
Data.corr()['Survived'].sort_values(ascending=True).drop(['Survived']).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=18)
plt.ylabel('Target',fontsize=14)
plt.title('Correlation',fontsize=18)
plt.show()
```



Observation of the correlation:-

Positively correlated with : Parch and Fare

Negatively correlated with : Pclass, Age, SibSp and PassengerId

In heatmap we also observed that the PassengerId column has no relation with the target variable, so we can drop that column.

In [50]:
```
#Dropping PassengerId column
Data = Data.drop(columns = "PassengerId", axis = 1)
```

Encoding the data (Label Encoder):- Machine learning models require all input and output variables to be numeric. T his means that if our data contains

categorical data, we must encode it to number s before we fit and evaluate a model. So, we will use label Encoder for encoding

In [53]:
```
#Label Encoding
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
```

In [54]:
```
#checking Original Data
Data["Sex"].unique()
```

Out[54]:
```
array(['male', 'female'], dtype=object)
```

In [55]:
```
#Coverting "Sex" column data which contains categorical data into continuous data.
Data['Sex']=encoder.fit_transform(Data['Sex'])
Data["Sex"].unique()
```

Out[55]:
```
array([1, 0])
```

In [56]:
```
#Male denoted as 1 and Female denoted as 0
```

In [57]:
```
#checking Original Data
Data["Embarked"].unique()
```

Out[57]:
```
array(['S', 'C', 'Q'], dtype=object)
```

In [58]:
```
#Coverting "Embarked" column data which contains categorical data into continuous data.
Data['Embarked']=encoder.fit_transform(Data['Embarked'])
Data["Embarked"].unique()
```

In [61]:
```
Data.head()
```

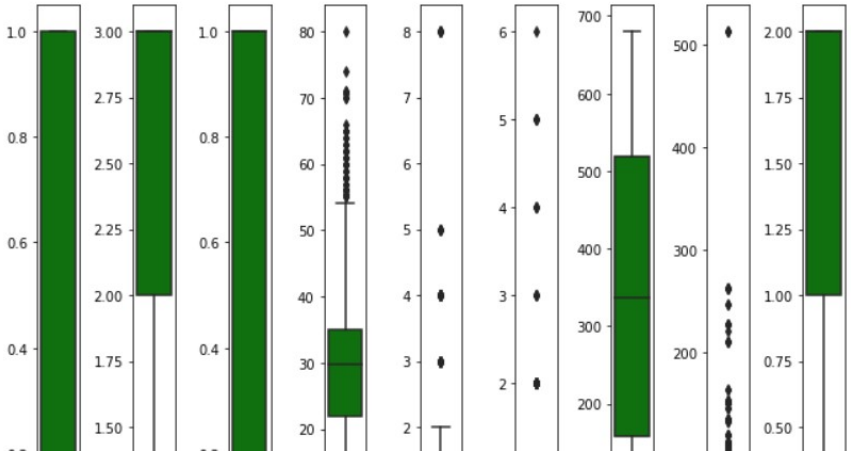
Out[61]:

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	523	7.2500	2
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	596	71.2833	0
2	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	669	7.9250	2
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	49	53.1000	2

Here we can see the dataset is totally converted into numerical form and is perfe ctly ready for further procedure.

Before process further we need to check outliner

In [66]:
```
#Checking Outliers
collist = Data.columns.values
ncol = 30
nrows = 14
plt.figure(figsize = (ncol,3*ncol))
for i in range(0,len(collist)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(data = Data[collist[i]],color='green',orient='v')
    plt.tight_layout()
```



Observation:-

Outliers present in columns: Age, SibSp, Parch and Fare

Outliers not present in columns: Survived, Pclass, Sex, Ticket and Embarked

Removing Outliers:-

1. Zscore method using Scipy:-

In [69]:
```
from scipy.stats import zscore
```

In [70]:
```
#  Outliers will be removed only from Continuous column variable i.e; Age and Fare.
#We will not remove outliers from Categorical column i.e; SibSp, Parch

variables = Data[['Age','Fare']]

z=np.abs(zscore(variables))

# Creating new dataframe
Data_2 = Data[(z<3).all(axis=1)]
Data_2.head()
```

Out[70]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	0	3	1	22.0	1	0	523	7.2500	2
1	1	1	0	38.0	1	0	596	71.2833	0
2	1	3	0	26.0	0	0	669	7.9250	2
3	1	1	0	35.0	1	0	49	53.1000	2
4	0	3	1	35.0	0	0	472	8.0500	2

In [71]:
```
z.head()
```

Out[71]:

	Age	Fare
0	0.592481	0.502445

In [73]:
```
print("Old DataFrame data in Rows and Column:",Data.shape)
print("New DataFrame data in Rows and Column:",Data_2.shape)
print("Total Dropped rows:",Data.shape[0]-Data_2.shape[0])
```

```
Old DataFrame data in Rows and Column: (891, 9)
New DataFrame data in Rows and Column: (864, 9)
Total Dropped rows: 27
```

In [74]:
```
#   Percentage Data Loss using Z score
loss_percent=(891-864)/891*100
print(loss_percent,"%")
```

2. IQR (Inter Quantile Range) Method

In [75]:
```
#     2. IQR (Inter Quantile Range) method
#1st quantile
Q1=variables.quantile(0.25)

# 3rd quantile
Q3=variables.quantile(0.75)

#IQR
IQR=Q3 - Q1
Data_new=Data[~((Data < (Q1 - 1.5 * IQR)) |(Data > (Q3 + 1.5 * IQR))).any(axis=1)]
```

In [76]:
```
print("Old DataFrame data in Rows and Column:",Data.shape)
print("\nNew DataFrame data in Rows and Column:",Data_new.shape)
print("\nTotal Dropped rows:",Data.shape[0]-Data_new.shape[0])
```

```
Old DataFrame data in Rows and Column: (891, 9)

New DataFrame data in Rows and Column: (721, 9)

Total Dropped rows: 170
```

In [77]:

There are 2 method which are used to remove outliner which we used in 1st method and 2nd method the difference is loss of data in 1st Zscore method its 3.030 and in IQR (Inter Quantile Range) Method its 19.079 So we will consider Zscore

We also will Check for the skewness

In [79]:
```
#Checking for the skewness
Data_2.skew()
```

Out[79]:
```
Survived    0.502011
Pclass     -0.708375
Sex        -0.632203
Age         0.239002
SibSp       3.751753
Parch       2.840412
Ticket      0.012485
Fare        2.257178
```

Observation:-

Skewness threshold taken is +/-0.65. Columns which are having skewness: Sex, SibSp, Parch, Fare, Embarked

The Fare column data is highly skewed

All the columns are not normallly distributed

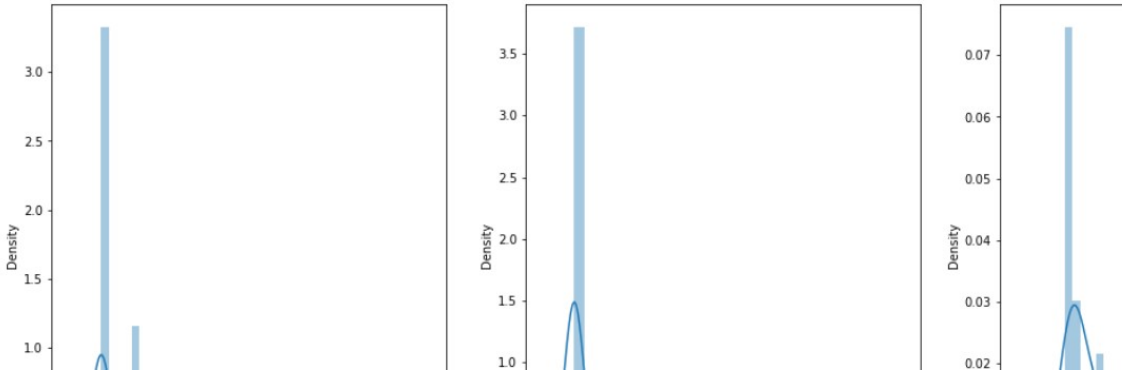
Since Embarked and Sex is categorical column so we will not remove skewness from them.

Only we will remove skewness from Sex, Parch and Fare as these column contains continuous data.

In [81]:
```
#Data visualization of skewed continuous column using distplot

collist=["SibSp", "Parch", "Fare"]
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in Data[collist]:
    if plotnumber<=9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(Data_2[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



Removing skewness using yeo-johnson method

In [82]:
```
#Removing skewness using yeo-johnson method

from sklearn.preprocessing  import power_transform
collist=["SibSp", "Parch", "Fare"]
Data_2[collist]=power_transform(Data_2[collist],method='yeo-johnson')
Data_2[collist]
```

Out[82]:

	SibSp	Parch	Fare
0	1.374574	-0.548637	-0.882769
1	1.374574	-0.548637	1.529038
2	-0.680931	-0.548637	-0.792444
3	1.374574	-0.548637	1.216581
4	-0.680931	-0.548637	-0.776493
...	...	...	...
886	-0.680931	-0.548637	-0.279691
887	-0.680931	-0.548637	0.607981
888	1.374574	1.879004	0.345275

We will again check skewness after removal

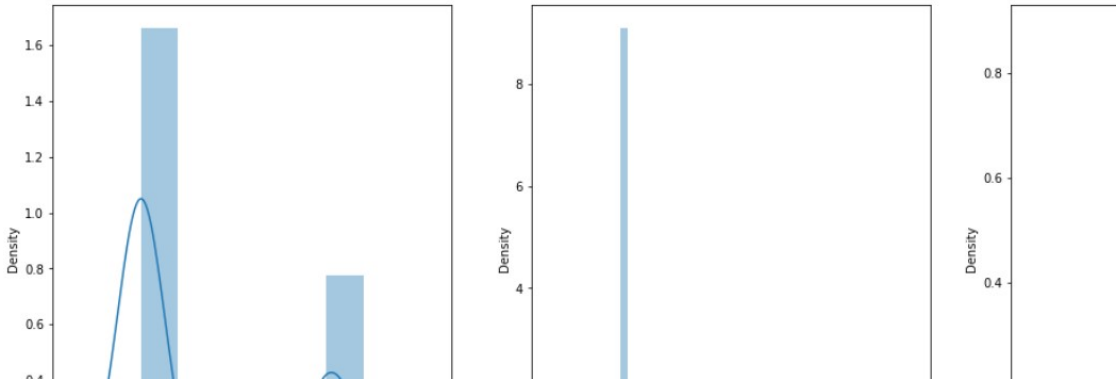
In [83]:
```
#checking skewness after removal
Data_2.skew()
```

Out[83]:
```
Survived    0.502011
Pclass     -0.708375
Sex        -0.632203
Age         0.239002
SibSp       0.805038
Parch       1.277888
Ticket      0.012485
```

In [84]:
```
#checking skewness after removal through data visualization using distplot

plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in Data[collist]:
    if plotnumber<=9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(Data_2[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



As we can see the data is not normal but the skewness has got removed compared to the old data.

Now we will Splitting data into Target and Features:

In [86]:
```
#Spliting data into Target and Features:

x=Data_2.drop("Survived",axis=1)
y=Data_2["Survived"]
x.head()
```

Out[86]:

	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	3	1	22.0	1.374574	-0.548637	523	-0.882769	2
1	1	0	38.0	1.374574	-0.548637	596	1.529038	0
2	3	0	26.0	-0.680931	-0.548637	669	-0.792444	2

```
In [87]: y.head()

Out[87]: 0    0
         1    1
         2    1
         3    1
         4    0
         Name: Survived, dtype: int64

In [88]: x.shape

Out[88]: (864, 8)

In [89]: y.shape

Out[89]: (864,)
```

We can see our data is not balanced. So, we will use oversampling method to balance it.

```
In [93]: #Oversampling using the SMOTE

         from imblearn import under_sampling, over_sampling
         from imblearn.over_sampling import SMOTE

In [94]: SM = SMOTE()
         x, y = SM.fit_resample(x,y)

In [95]: y.value_counts()
```

After using oversampling method, we check that now our data is balanced.

Here x represents all the independent variable that are features and y represents target variable i.e., a dependent variable.

Scaling the data (MinMaxScaler):-

Scaling of the data makes it easy for a model to learn and understand the problem. And our dataset needs standardization so we will use MinMaxScaler for data scaling.

In [100…

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x = pd.DataFrame(mms.fit_transform(x), columns=x.columns)
```

In [101…

```
x.head()
```

Out[101…

	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1.0	1.0	0.329064	0.83479	0.0	0.769118	0.432617	1.0
1	0.0	0.0	0.573041	0.83479	0.0	0.876471	0.848336	0.0
2	1.0	0.0	0.390058	0.00000	0.0	0.983824	0.448186	1.0

By using min max scaler we can see that the values are between 0 and 1 and thus it is our scaled data.

Model Building and Evaluation:-

We fit the dataset into multiple regression models to compare the performance of all models and then will select the best model. Firstly, we will import different ML models from sklearn.

In [103…

```
#Creating Model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc, roc_auc_score, accuracy_score, classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier as KNN
```

Checking the best random state for getting best accuracy score.

In [104…

```
#Finding the best random state among all the models

maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.30, random_state =i)
    DTC = DecisionTreeClassifier()
    DTC.fit(x_train, y_train)
    pred = DTC.predict(x_test)
    acc=accuracy_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
```

Here we got best accuracy score of 86% at random state 136 and. So we will use this random state for all remaining models.

As it is a continuous data , so we have to understand That it is a "Classification problem"

In [105…
```
#   Creating train-test-split
#   creating new train test split using the random state.
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=maxRS)
```

In [106…
```
x_train.shape,y_train.shape, x_test.shape,y_test.shape
```

Out[106… ((751, 8), (751,), (323, 8), (323,))

In [107…
```
x.shape, y.shape
```

Out[107… ((1074, 8), (1074,))

In [108…
```
#We can see the x.shape value is divided into x_train.shape and x_test.shape and like this y.shape is also divided.
#As it is a continuous data , so we have to understand That it is a  "Classification problem"
```

In [109…
```
#Logistic Regression
lr=LogisticRegression()
lr.fit(x_train,y_train)
pred_lr=lr.predict(x_test)

print(accuracy_score(y_test, pred_lr))
print(confusion_matrix(y_test, pred_lr))
print(classification_report(y_test,pred_lr))
```

```
0.8111455108359134
[[122  30]
 [ 31 140]]
```

We can see for the Logistic Regression we get the Accuracy of 81%

And we will also check with others to know with one gives batter accuracy.

1. Decision Tree Classifier

In [112…
```
#Classification Algorithms
# 1. Decision Tree Classifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
pred_dtc = dtc.predict(x_test)

print(accuracy_score(y_test, pred_dtc))
print(confusion_matrix(y_test, pred_dtc))
print(classification_report(y_test,pred_dtc))
```

```
0.8637770897832817
[[133  19]
 [ 25 146]]
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       152
           1       0.88      0.85      0.87       171
```

We can see for Decision Tree Classifier we get the Accuracy of 86%

2. Random Forest Classifier

In [115…

```
#2. Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=200)
rfc.fit(x_train,y_train)
pred_rfc = rfc.predict(x_test)

print(accuracy_score(y_test, pred_rfc))
print(confusion_matrix(y_test, pred_rfc))
print(classification_report(y_test,pred_rfc))
```

```
0.8575851393188855
[[134  18]
 [ 28 143]]
              precision    recall  f1-score   support

           0       0.83      0.88      0.85       152
           1       0.89      0.84      0.86       171
```

Here we are getting 85% accuracy using Random Forest Classifier.

3. Support Vector Machine Classifier

In [118…

```
#3. Support Vector Machine Classifier
svc = SVC(kernel='linear', gamma=3)
svc.fit(x_train,y_train)
pred_svc = svc.predict(x_test)

print(accuracy_score(y_test, pred_svc))
print(confusion_matrix(y_test, pred_svc))
print(classification_report(y_test,pred_svc))
```

```
0.8173374613003096
[[131  21]
 [ 38 133]]
              precision    recall  f1-score   support

           0       0.78      0.86      0.82       152
           1       0.86      0.78      0.82       171
```

Here we are getting 81% accuracy using Random Forest Classifier.

4. KNN Classifier

In [121…

```
#4. KNN Classifier
knn = KNN()
knn.fit(x_train,y_train)
pred_knn = knn.predict(x_test)

print(accuracy_score(y_test, pred_knn))
print(confusion_matrix(y_test, pred_knn))
print(classification_report(y_test,pred_knn))
```

```
0.8080495356037152
[[129  23]
 [ 39 132]]
              precision    recall  f1-score   support

           0       0.77      0.85      0.81       152
           1       0.85      0.77      0.81       171
```

Here we are getting 80% accuracy using KNN Classifier.

5. Gradient Boosting Classifier

In [124…

```
#5. Gradient Boosting Classifier
gb = GradientBoostingClassifier(n_estimators =300,learning_rate=0.1, max_depth=4)
gb.fit(x_train,y_train)
pred_gb = gb.predict(x_test)

print(accuracy_score(y_test, pred_gb))
print(confusion_matrix(y_test, pred_gb))
print(classification_report(y_test,pred_gb))
```

```
0.8730650154798761
[[138  14]
 [ 27 144]]
              precision    recall  f1-score   support

           0       0.84      0.91      0.87       152
           1       0.91      0.84      0.88       171
```

Here we are getting 87% accuracy using KNN Classifier.

Now we will Check Cross Validation Score for all the model

In [128…

```
from sklearn.model_selection import cross_val_score
```

In [129…

```
#  CV Score for Logistic Regression
print('CV score for Logistic Regression: ',cross_val_score(lr,x,y,cv=9).mean())
```

```
CV score for Logistic Regression:  0.8082788671023964
```

In [130…

```
#CV Score for Decision Tree Classifier
print('CV score for Decision Tree Classifier: ',cross_val_score(dtc,x,y,cv=9).mean())
```

```
CV score for Decision Tree Classifier:  0.8148537192654841
```

In [131…

```
#CV Score for Random Forest Classifier
print('CV score for Random forest Classifier: ',cross_val_score(rfc,x,y,cv=9).mean())
```

```
CV score for Random forest Classifier:  0.8707438530967944
```

In [132…

```
#CV Score for Support Vector Classifier
print('CV score for Support Vector  Classifier: ',cross_val_score(svc,x,y,cv=9).mean())
```

```
CV score for Support Vector  Classifier:  0.7989495798319327
```

In [133…

```
#  CV Score for KNN Classifier
print('CV score for KNN Classifier: ',cross_val_score(knn,x,y,cv=9).mean())
```

In [135…

```
score= pd.DataFrame({'CV_GB':0.891640866873065, 'accuracy_score_GB':0.8623949579831933}, index=['0'])
score
```

Out[135…

	CV_GB	accuracy_score_GB

We choose the model on basis of lowest difference between model accuracy sco re and cross validation score of that model, we observe that we got less differenc e/almost equal score for Gradient Boosting Classifier, so we will perform hyper par ameter tunning for Gradient Boosting Classifier.

Hyper Parameter Tunning:-

Hyper parameter tuning consists of finding a set of optimal hyper parameter values for a learning algorithm while applying this optimized algorithm to any data set. That combination of hyper parameters maximizes the model's performance, minimizing a predefined loss function to produce better results with fewer errors

We will do hyper parameter tunning using Gradient Boosting Classifier, it is the process of performing hyper parameter tuning in order to determine the optimal values for a given model. As mentioned above, the performance of a model significantly depends on the value of hyper parameters.

In [138…
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import precision_score, recall_score, make_scorer
```

In [139…
```
gb=GradientBoostingClassifier(random_state=42)
```

In [140…
```
parameters = {
    "loss":["deviance"],
    "learning_rate": [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],
    "min_samples_split": np.linspace(0.1, 0.5, 12),
    "min_samples_leaf": np.linspace(0.1, 0.5, 12),
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],
    "criterion": ["friedman_mse",  "mae"],
    "subsample":[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0],
    "n_estimators":[10]
}
```

In [143…
```
CV_GB = GridSearchCV(GradientBoostingClassifier(), parameters, cv= 5, n_jobs=-1)
CV_GB.fit(x_train, y_train)
```

Out[143…
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'criterion': ['friedman_mse', 'mae'],
                         'learning_rate': [0.01, 0.025, 0.05, 0.075, 0.1, 0.15,
                                           0.2],
                         'loss': ['deviance'], 'max_depth': [3, 5, 8],
                         'max_features': ['log2', 'sqrt'],
                         'min samples leaf': array([0.1       , 0.13636364, 0.17272727, 0.20909091, 0.24545455
```

In [146…
```
CV_GB.best_params_
```

Out[146…
```
{'criterion': 'friedman_mse',
 'learning_rate': 0.15,
 'loss': 'deviance',
 'max_depth': 5,
 'max_features': 'sqrt',
 'min_samples_leaf': 0.1,
 'min_samples_split': 0.42727272727272736,
 'n_estimators': 10,
 'subsample': 1.0}
```

In [148…
```
survived = GradientBoostingClassifier(criterion='friedman_mse',max_depth=30, max_features='auto', max_leaf_nodes=40,
survived.fit(x_train, y_train)
pred = survived.predict(x_test)
```

After hyper parameter tunning, we got the r2 score of almost 86% which is reall y good. Hence, we are selecting Gradient Boosting Classifier as our final model , saving the model using best parameters, and creating model object using pickle.

In [ ]:

```
Saving the Model
import pickle
filename='Titanic_Train_Project.pickle'
pickle.dump(CV_GB,open(filename,'wb'))
loaded_model = pickle.load(open(filename, 'rb'))
loaded_model.predict(x_test)
```

Conclusion:-

Here after fitting the best parameters, we got the r2 score of almost 86% and we saved it using pickle object.