SURVIVAL PREDICTION ON TITANIC SHIP BY USING MACHINE LEARNING ALGORITHMS

CONTENT :

Predictive Modeling for Titanic Passenger Survival

by vishalyadav

INTRODUCTION :

Greetings,

I am excited to present my work on developing a predictive model for determining the likelihood of survival for passengers aboard the Titanic. Leveraging data science techniques in Python, I have embarked on this project with the goal of providing insights into the factors that influenced survival during this historic event.

Objective:

The primary objective of this predictive model is to analyze the Titanic dataset and build a robust machine learning model that can accurately predict whether a passenger survived or not based on various features such as class, gender, age, and family relationships. The model's performance will be assessed using metrics like accuracy, and we'll explore additional aspects such as feature importance and cross-validation to ensure reliability.

Methodology:

The methodology involves a comprehensive process, including data exploration, preprocessing, feature selection, model training, and evaluation. I have utilized popular Python libraries such as pandas, numpy, seaborn, and scikit-learn to streamline these tasks efficiently. Additionally, hyperparameter tuning and cross-validation techniques have been applied to enhance the model's predictive capabilities

Survival Prediction on Titanic Ship

Libraries we required for data analysis

Current working directory will helps to open the csv files without giving a proper location if our csv files present in that folder

Here my current working directory is Downloads and my csv file also present at that folder so i dont need to mentioned a proper location eg. C:\Users\admin\Downloads\titanic.csv

```
#for checking our current working directory in which we working
import os
os.getcwd()
```

```
'C:\\Users\\admin\\Downloads'
```

```
# Importing required Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings("ignore")
```

READING A FAST FOOD DATASET INFORMATION

1. READING THE DATA

```
df=pd.read_csv(r"titanic.csv")
df
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         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
..                                                 ...     ...   ...
...
886                              Montvila, Rev. Juozas    male  27.0
0
887                       Graham, Miss. Margaret Edith  female  19.0
0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN
1
889                              Behr, Mr. Karl Howell    male  26.0
0
890                                Dooley, Mr. Patrick    male  32.0
```

```
0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     ...               ...      ...   ...      ...
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

2. ANALYSING THE DATAFRAME COLUMNS

```
# for a look of datasets columns names
df.columns

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

Dataset has contain a detail of columns as follows :

(a) PassengerId : It is a unique id for each passenger.

(b) Survived : The value '0' represent not survied and '1' represent the passenger will survived. This is our targeted columns .

(c) Pclass : This is a class of passengers which based on their tickets fairs.

(d) Name : Name of passenger.

(e) Sex : Gender of passenger.

(f) Age : Age of passenger.

(g) SibSp : Number of Siblings/Spouses Aboard.

(h) Parch : Number of Parents/Children Aboard..

(i) Ticket : Ticket number have unique codes.

(j) Fare : Total Fair paid by the passenger.

(k) Cabin : This contain a number of cabin which alot to a passenger.

(l) Embarked : From where the traveler mounted from. There are three Embark Location — Southampton, Cherbourg, and Queenstown.

BASIC INFORMATION OF TITANIC DATASET

```
# for information of columns names dtype and for counts of null
objects
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
```

```
# for above 5 rows
df.head()
```

```
   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            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
# fr belows 5 rows
df.tail()
```

```
     PassengerId  Survived  Pclass
Name  \
886          887         0       2                    Montvila, Rev.
Juozas
887          888         1       1             Graham, Miss. Margaret
Edith
888          889         0       3  Johnston, Miss. Catherine Helen
"Carrie"
889          890         1       1                    Behr, Mr. Karl
Howell
890          891         0       3                      Dooley, Mr.
Patrick

        Sex   Age  SibSp  Parch      Ticket   Fare Cabin Embarked
886    male  27.0      0      0      211536  13.00   NaN        S
887  female  19.0      0      0      112053  30.00   B42        S
888  female   NaN      1      2  W./C. 6607  23.45   NaN        S
889    male  26.0      0      0      111369  30.00  C148        C
890    male  32.0      0      0      370376   7.75   NaN        Q
```

STATISTICAL INFORMATION OF DATASET

```
# for correlatin of columns
df.corr()
```

```
             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
```

```python
# for stastical viwes of dataset
df.describe()
```

```
       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
```

Here a minimum values for all columns is showing ZERO, It's mean we need to remove some of outlyier from our data.

3. DATA CLEANING :- EDA process

Handling the missing values

```python
#Finding null/missing values in the data if any.
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
# for dtypes of each columns and null values in dataset
df.info()

print(("*")*125)
print("\t\t\t\t\t**************ANALYSIS***************\n\n As we see in
above info the dtypes of few columns is objects but the data contains
in it was floats so we first replace this ? with means of that columns
and also the null values we will replace it with the mean values ")
print("\n\n THE MISSED DATA TYPES COLUMNS ARE : \n\n 5   Age
714 non-null    float64 \n 10  Cabin        204 non-null    object \n
11  Embarked     889 non-null    object")
```

```
<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
**********************************************************************
******************************************************
                      **************ANALYSIS***************

 As we see in above info the dtypes of few columns is objects but the
```

```
data contains in it was floats so we first replace this ? with means
of that columns and also the null values we will replace it with the
mean values


 THE MISSED DATA TYPES COLUMNS ARE :

 5   Age          714 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
```

```
# for null values
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

AS we see their is some null values in 3 columns let us short out it

```
#Dropping  the “Cabin” column from the data frame as it won’t be of
much importance
df = df.drop(columns='Cabin', axis=1)

#Replacing the missing values in the “Age” column with the mean value
df['Age'].fillna(df['Age'].mean(), inplace=True)

#Finding the mode value of the “Embarked” column as it will have
occurred the maximum number of times
print(df['Embarked'].mode())
```

```
0    S
Name: Embarked, dtype: object
```

```
#Replacing the missing values in the “Embarked” column with mode value
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

# for checking again the not null values after replacing
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 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          891 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  Embarked     891 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

```
# for null values
df.isnull().sum()
```

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

Droping a unnecessary columns which is not related to our target columns

```
# drop unnecessary columns
df.drop(['PassengerId','Ticket','Name','Embarked'],axis=1,inplace=True
)

df
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	male	22.000000	1	0	7.2500
1	1	1	female	38.000000	1	0	71.2833
2	1	3	female	26.000000	0	0	7.9250
3	1	1	female	35.000000	1	0	53.1000
4	0	3	male	35.000000	0	0	8.0500

```
..        ...     ...     ...        ...     ...     ...       ...
886         0       2    male  27.000000       0       0  13.0000
887         1       1  female  19.000000       0       0  30.0000
888         0       3  female  29.699118       1       2  23.4500
889         1       1    male  26.000000       0       0  30.0000
890         0       3    male  32.000000       0       0   7.7500

[891 rows x 7 columns]
```

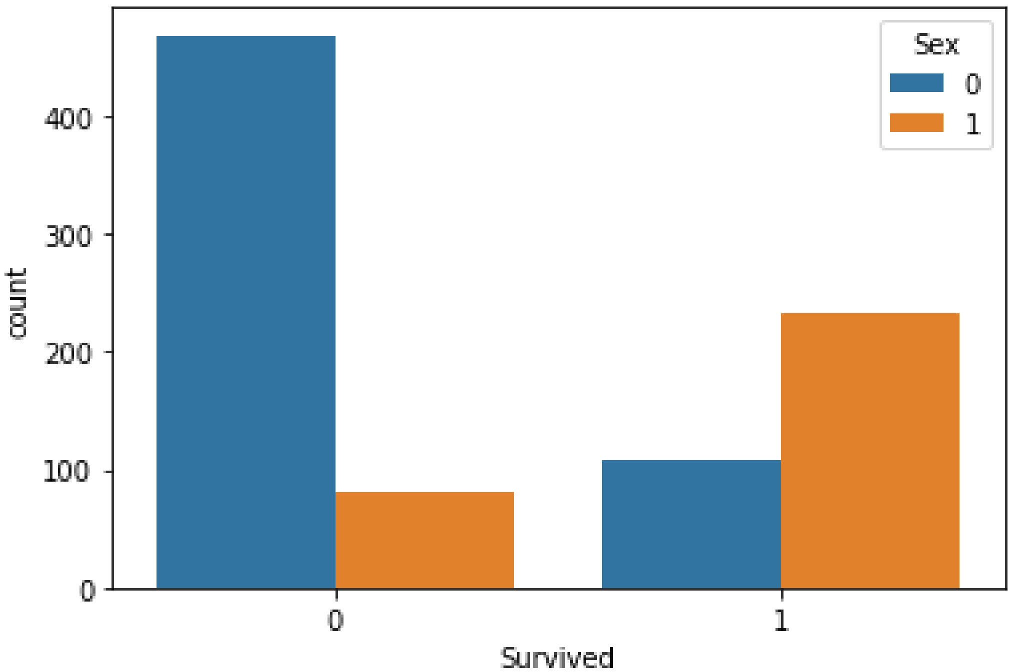
Transformation into a categorical column.

```
#Let’s convert that into integer type values, and transform it into a
categorical column
df.replace({'Sex':{'male':0,'female':1}, 'Embarked':
{'S':0,'C':1,'Q':2}}, inplace=True)

df

     Survived  Pclass  Sex        Age  SibSp  Parch     Fare
0           0       3    0  22.000000      1      0   7.2500
1           1       1    1  38.000000      1      0  71.2833
2           1       3    1  26.000000      0      0   7.9250
3           1       1    1  35.000000      1      0  53.1000
4           0       3    0  35.000000      0      0   8.0500
..        ...     ...  ...        ...    ...    ...      ...
886         0       2    0  27.000000      0      0  13.0000
887         1       1    1  19.000000      0      0  30.0000
888         0       3    1  29.699118      1      2  23.4500
889         1       1    0  26.000000      0      0  30.0000
890         0       3    0  32.000000      0      0   7.7500

[891 rows x 7 columns]
```

4. DATA VISUALIZATION

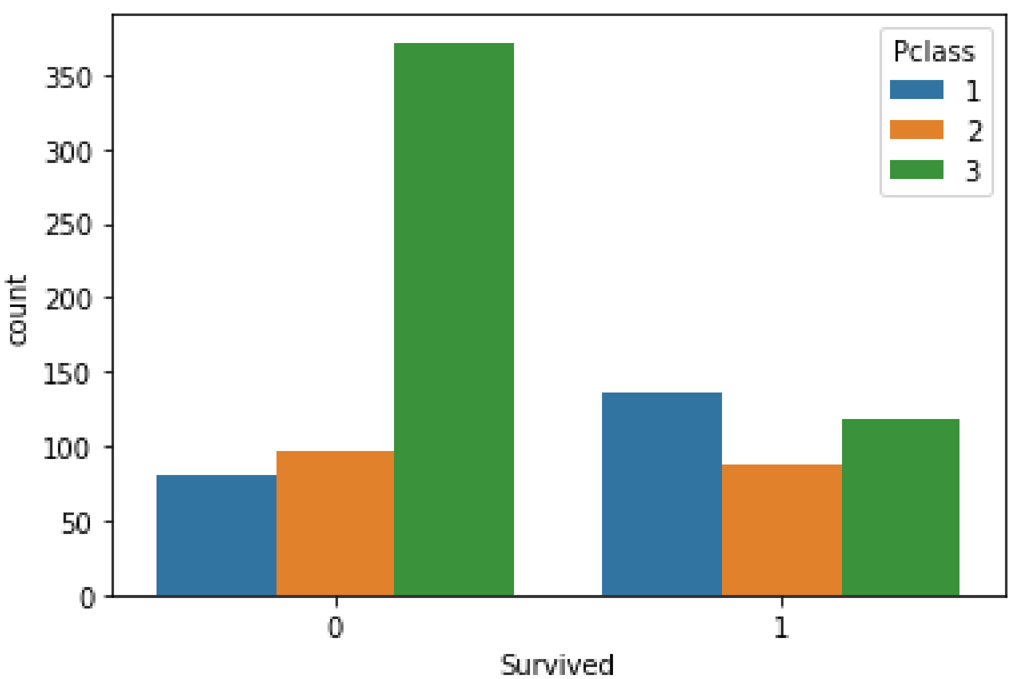
UNDERSTANDING THE DATA BY GRAPHS

```
# Survival of Male and Female
sns.countplot(x='Survived',data=df,hue = 'Sex')

<AxesSubplot:xlabel='Survived', ylabel='count'>
```




```
# Survival Based on Pclass
sns.countplot(x='Survived',data=df,hue = 'Pclass')
```

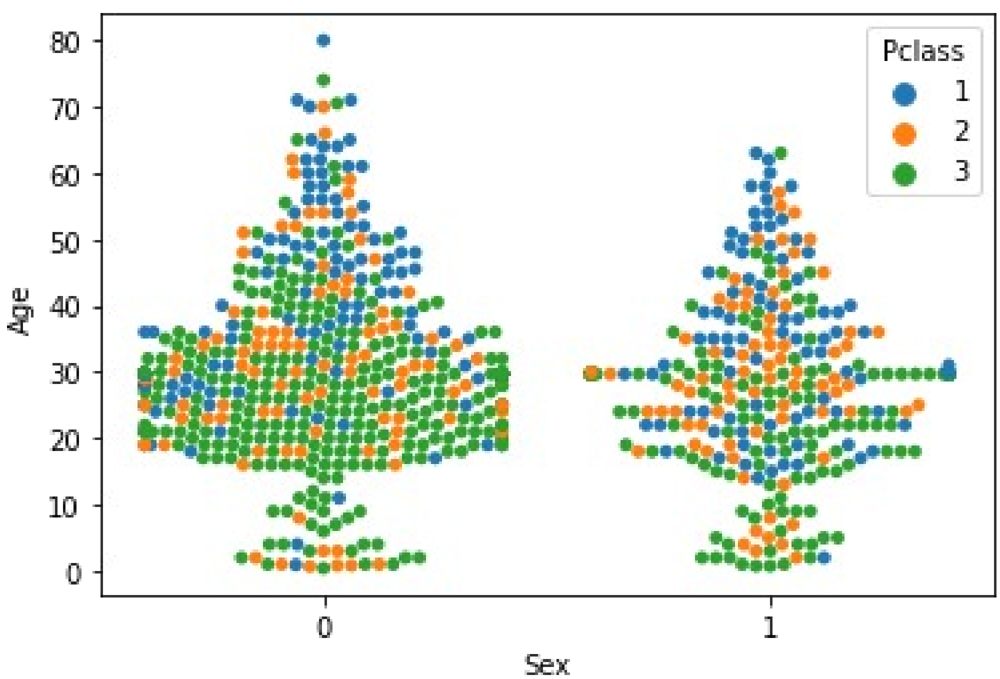
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



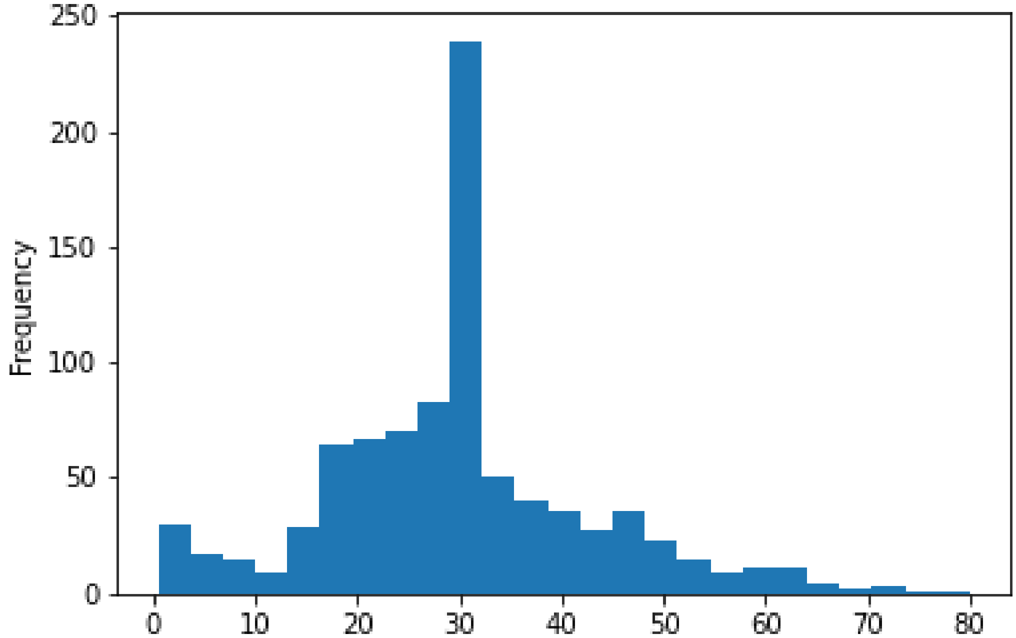

```
df
```

```
     Survived  Pclass  Sex        Age  SibSp  Parch     Fare
0           0       3    0  22.000000      1      0   7.2500
1           1       1    1  38.000000      1      0  71.2833
2           1       3    1  26.000000      0      0   7.9250
3           1       1    1  35.000000      1      0  53.1000
4           0       3    0  35.000000      0      0   8.0500
..        ...     ...  ...        ...    ...    ...      ...
886         0       2    0  27.000000      0      0  13.0000
887         1       1    1  19.000000      0      0  30.0000
888         0       3    1  29.699118      1      2  23.4500
889         1       1    0  26.000000      0      0  30.0000
890         0       3    0  32.000000      0      0   7.7500

[891 rows x 7 columns]
```

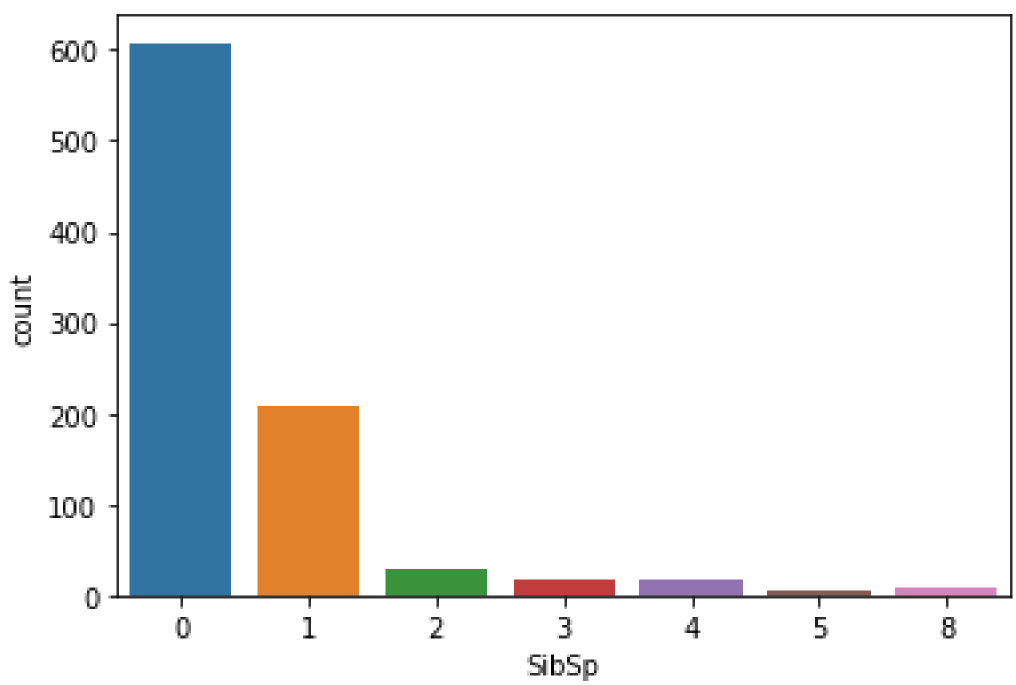
```
# Distribution of data by pclass
sns.swarmplot(x = "Sex", y = "Age", hue = "Pclass", data = df )
plt.show()
```




```
# Survival based on age
df['Age'].plot.hist(bins=25)

<AxesSubplot:ylabel='Frequency'>
```




```
# survival by siblings
sns.countplot(x='SibSp',data=df)

<AxesSubplot:xlabel='SibSp', ylabel='count'>
```



5. DATA SEPERATION AS TEST AND TRAIN DATA.

Separating into Features and Target Coulmns

Our targeted column is Survived

```
# assigning a variable x as a features columns
# Here we drop our target columns and assign all remaining columns as
a feature columns to variable x
print(("*")*125)
print("\t\t\t\t*************ANALYSIS**************\n\nExcet Survived
Columns all other remaing columns is our feature data.")
print(("*")*125)

x=df.drop(columns=['Survived'])
x
```

```
**********************************************************************
*******************************************************
                      *************ANALYSIS**************

Excet Survived Columns all other remaing columns is our feature data.
**********************************************************************
*******************************************************

     Pclass  Sex        Age  SibSp  Parch     Fare
0         3    0  22.000000      1      0   7.2500
1         1    1  38.000000      1      0  71.2833
2         3    1  26.000000      0      0   7.9250
3         1    1  35.000000      1      0  53.1000
4         3    0  35.000000      0      0   8.0500
..      ...  ...        ...    ...    ...      ...
886       2    0  27.000000      0      0  13.0000
887       1    1  19.000000      0      0  30.0000
888       3    1  29.699118      1      2  23.4500
889       1    0  26.000000      0      0  30.0000
890       3    0  32.000000      0      0   7.7500

[891 rows x 6 columns]
```

```
# our target columns is energy so by using iloc i will assign that
columns to y.
y=df.iloc[:,0]
y
```

```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

Split the data into training and testing

```
# Importing sklearn library to split our data in test and train data
from sklearn.model_selection import train_test_split, GridSearchCV
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_s
tate=42)

# Values of our xtrain
xtrain

     Pclass  Sex        Age  SibSp  Parch      Fare
298       1    0  29.699118      0      0   30.5000
884       3    0  25.000000      0      0    7.0500
247       2    1  24.000000      0      2   14.5000
478       3    0  22.000000      0      0    7.5208
305       1    0   0.920000      1      2  151.5500
..      ...  ...        ...    ...    ...       ...
106       3    1  21.000000      0      0    7.6500
270       1    0  29.699118      0      0   31.0000
860       3    0  41.000000      2      0   14.1083
435       1    1  14.000000      1      2  120.0000
102       1    0  21.000000      0      1   77.2875

[668 rows x 6 columns]

# Values of our xtest
xtest

     Pclass  Sex        Age  SibSp  Parch     Fare
709       3    0  29.699118      1      1  15.2458
439       2    0  31.000000      0      0  10.5000
840       3    0  20.000000      0      0   7.9250
720       2    1   6.000000      0      1  33.0000
39        3    1  14.000000      1      0  11.2417
..      ...  ...        ...    ...    ...      ...
880       2    1  25.000000      0      1  26.0000
```

```
425        3    0  29.699118      0       0   7.2500
101        3    0  29.699118      0       0   7.8958
199        2    1  24.000000      0       0  13.0000
424        3    0  18.000000      1       1  20.2125

[223 rows x 6 columns]
```

```python
# Values of our ytrain
ytrain
```

```
298    1
884    0
247    1
478    0
305    1
      ..
106    1
270    0
860    0
435    1
102    0
Name: Survived, Length: 668, dtype: int64
```

```python
# Values of our ytest
ytest
```

```
709    1
439    0
840    0
720    1
39     1
      ..
880    1
425    0
101    0
199    0
424    0
Name: Survived, Length: 223, dtype: int64
```

6. MODEL TRAINING BY USING MACHINE LEARNING ALGORITHM

1 - Finding the Accuracy of data by using Logistic Regression

```python
# Logistic Regression
from sklearn.linear_model import LogisticRegression

logistic_regression_model = LogisticRegression(random_state=42)
logistic_regression_model.fit(xtrain, ytrain)
```

```
LogisticRegression(random_state=42)
```

```
y_pred = logistic_regression_model.predict(xtest)
acc_logreg = round(accuracy_score(y_pred, ytest) * 100, 2)
print(f'Logistic Regression accuracy {acc_logreg}')
print(classification_report(ytest,y_pred))
print(f'The accuracy achieved from Logistic Regression is :
{acc_logreg}\n\n')
```

```
Logistic Regression accuracy 80.72
              precision    recall  f1-score   support

           0       0.82      0.87      0.84       134
           1       0.78      0.72      0.75        89

    accuracy                           0.81       223
   macro avg       0.80      0.79      0.80       223
weighted avg       0.81      0.81      0.81       223

The accuracy achieved from Logistic Regression is : 80.72
```

2 - Finding the Accuracy of data by using Decission Tree Classifier

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier

decisiontree_model = DecisionTreeClassifier(random_state=42)
decisiontree_model.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier(random_state=42)
```

```
y_pred = decisiontree_model.predict(xtest)
acc_decisiontree = round(accuracy_score(ytest,y_pred) * 100, 2)
print(f'Decision Tree Classifier accuracy {acc_decisiontree: .2f}')
print(classification_report(ytest, y_pred))
print(f'The accuracy achieved from Decision Tree Classifier is :
{acc_decisiontree}\n\n')
```

```
Decision Tree Classifier accuracy  73.99
              precision    recall  f1-score   support

           0       0.78      0.80      0.79       134
           1       0.68      0.65      0.67        89

    accuracy                           0.74       223
   macro avg       0.73      0.73      0.73       223
weighted avg       0.74      0.74      0.74       223

The accuracy achieved from Decision Tree Classifier is : 73.99
```

3 - Finding the Accuracy of data by using Random Forest Classifier

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier

random_forest_model = RandomForestClassifier(random_state=42)
random_forest_model.fit(xtrain, ytrain)

RandomForestClassifier(random_state=42)

predictions = random_forest_model.predict(xtest)
ranfor_accuracy = accuracy_score(ytest, predictions)
print(f"Random Forest Classifier Accuracy: {ranfor_accuracy:.2f}")
print(classification_report(ytest, predictions))
print(f'The accuracy achieved from Random Forest Classifier is :
{ranfor_accuracy*100}\n\n')

Random Forest Classifier Accuracy: 0.79
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       134
           1       0.75      0.72      0.74        89

    accuracy                           0.79       223
   macro avg       0.79      0.78      0.78       223
weighted avg       0.79      0.79      0.79       223

The accuracy achieved from Random Forest Classifier is :
79.37219730941703
```

4 - Finding the Accuracy of data by using XGBoost Classifier

```
# XGBOOST Classifier
from xgboost import XGBClassifier

xgb_model = XGBClassifier(random_state=42)
xgb_model.fit(xtrain, ytrain)

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None,
importance_type=None,
              interaction_constraints=None, learning_rate=None,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
```

```
                 min_child_weight=None, missing=nan,
monotone_constraints=None,
                 n_estimators=100, n_jobs=None, num_parallel_tree=None,
                 predictor=None, random_state=42, ...)

xgb_predictions = xgb_model.predict(xtest)
xgb_accuracy = accuracy_score(ytest, xgb_predictions)
print(f'XGBoost Accuracy: {xgb_accuracy:.2f}')
print(classification_report(ytest, xgb_predictions))
print(f'The accuracy achieved from XGBoost Accuracy Classifier is :
{xgb_accuracy*100}\n\n')

XGBoost Accuracy: 0.79
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       134
           1       0.75      0.72      0.74        89

    accuracy                           0.79       223
   macro avg       0.79      0.78      0.78       223
weighted avg       0.79      0.79      0.79       223

The accuracy achieved from XGBoost Accuracy Classifier is :
79.37219730941703
```

7. MODEL EVOLUTION

we got the accuracy of different modules of classifier

```
print(f'The accuracy achieved from Logistic Regression is :
{acc_logreg}\n\n')
print(f'The accuracy achieved from Decision Tree Classifier is :
{acc_decisiontree}\n\n')
print(f'The accuracy achieved from Random Forest Classifier is :
{ranfor_accuracy*100}\n\n')
print(f'The accuracy achieved from XGBoost Accuracy Classifier is :
{xgb_accuracy*100}\n\n')

The accuracy achieved from Logistic Regression is : 80.72


The accuracy achieved from Decision Tree Classifier is : 73.99


The accuracy achieved from Random Forest Classifier is :
79.37219730941703
```

```
The accuracy achieved from XGBoost Accuracy Classifier is :
79.37219730941703
```

PREDICTING A NEW VALUES BASED ON OUR MODULES

By logistic_regression_model

```
new_data = pd.DataFrame({'Pclass': [1], 'Sex': [1], 'Age': [25],
'SibSp': [1], 'Parch': [2], 'Fare':[45.45]})
prediction_new_data = logistic_regression_model.predict(new_data)
print(f'Survival Prediction From logistic_regression_model :
{prediction_new_data[0]}')
```

```
Survival Prediction From logistic_regression_model : 1
```

By decisiontree_model

```
new_data = pd.DataFrame({'Pclass': [1], 'Sex': [1], 'Age': [25],
'SibSp': [1], 'Parch': [2], 'Fare':[45.45]})
prediction_new_data = decisiontree_model.predict(new_data)
print(f'Survival Prediction From decisiontree_model:
{prediction_new_data[0]}')
print(("*")*125)
print(f' As we see durinhg the prediction the accuracy achieved from
Decision Tree Classifier is : {acc_decisiontree} which is not as much
good, just Because of that their Survival Prediction was wrong')
print(("*")*125)
```

```
Survival Prediction From decisiontree_model: 0
**********************************************************************
*******************************************************
 As we see durinhg the prediction the accuracy achieved from Decision
Tree Classifier is : 73.99 which is not as much good, just Because of
that their Survival Prediction was wrong
**********************************************************************
*******************************************************
```

By random_forest_model

```
new_data = pd.DataFrame({'Pclass': [1], 'Sex': [1], 'Age': [25],
'SibSp': [1], 'Parch': [2], 'Fare':[45.45]})
prediction_new_data = random_forest_model.predict(new_data)
print(f'Survival Prediction From random_forest_model:
{prediction_new_data[0]}')
```

```
Survival Prediction From random_forest_model: 1
```

By xgboost_model

```
new_data = pd.DataFrame({'Pclass': [1], 'Sex': [1], 'Age': [25],
'SibSp': [1], 'Parch': [2], 'Fare':[45.45]})
prediction_new_data = xgb_model.predict(new_data)
print(f'Survival Prediction From xgboost_model :
{prediction_new_data[0]}')
```

```
Survival Prediction From xgboost_model : 1
```

8. HYPER PARAMETER TUNING BY USING GRID_SEARCH_CV

```
# Hyperparameter Tuning using GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

model = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(xtrain, ytrain)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [None, 10, 20],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [50, 100, 200]})
```

```
# Best hyperparameters
best_params = grid_search.best_params_
print(f'Best Hyperparameters: {best_params}')
```

```
Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 100}
```

```
# Evaluate model with best hyperparameters
best_model = grid_search.best_estimator_
predictions = best_model.predict(xtest)
accuracy = accuracy_score(ytest, predictions)
print(f'The accuracy achieved from Random Forest Classifier Before
Hyper Parameter Tunning is : {ranfor_accuracy*100}\n\n')

print(f'Accuracy with Best Hyperparameters : {accuracy*100}')
```

```
The accuracy achieved from Random Forest Classifier Before Hyper
Parameter Tunning is : 79.37219730941703


Accuracy with Best Hyperparameters : 81.16591928251121
```

CONCLUSION

```
print(("*")*125)
print(f' As we see durinhg the prediction the accuracy achieved from Decision Tree Classifier is : {acc_decisiontree} which is not as much good, just Because of that their Survival Prediction was wrong')
print(("*")*125)
print(f"\tBased on the above accuracy scores, we should go ahead with \n\nLogistic Regression :{acc_logreg},\n\nDecision Tree Classifier : {acc_decisiontree},\n\nRandom Forest Classifier : {ranfor_accuracy*100},\n\nXGBoost Accuracy Classifier is : {xgb_accuracy*100},\n\nAccuracy with Best Hyperparameters : {accuracy*100}\n\n The best predictive model for the above dataset is Logistic Regression and its Accuracy is {acc_logreg}. So Logistic Regression Classifier is best for Surviival Prediction on Titanic.csv ")
print(("*")*125)
print('This project not only serves as an exercise in predictive modeling but also provides valuable insights into the factors influencing survival rates on the Titanic. The skills developed and lessons learned during this project contribute to a broader understanding of data science methodologies and their applications I look forward to discussing the details of this project and how it aligns with the objectives and expectations of Bharat Intern')
print( '\n\nBest regards')
print('[Yadav Vishal]')
print('Bharat Intern Working Intern')
print(("*")*125)
```

```
*****************************************************************************************************************************
 As we see durinhg the prediction the accuracy achieved from Decision Tree Classifier is : 73.99 which is not as much good, just Because of that their Survival Prediction was wrong
*****************************************************************************************************************************
	Based on the above accuracy scores, we should go ahead with 

Logistic Regression :80.72,

Decision Tree Classifier : 73.99,

Random Forest Classifier : 79.37219730941703,

XGBoost Accuracy Classifier is : 79.37219730941703,

Accuracy with Best Hyperparameters : 81.16591928251121

 The best predictive model for the above dataset is Logistic Regression and its Accuracy is 80.72. So Logistic Regression
```

```
Classifier is best for Surviival Prediction on Titanic.csv
**********************************************************************
*******************************************************
This project not only serves as an exercise in predictive modeling but
also provides valuable insights into the factors influencing survival
rates on the Titanic. The skills developed and lessons learned during
this project contribute to a broader understanding of data science
methodologies and their applications I look forward to discussing the
details of this project and how it aligns with the objectives and
expectations of Bharat Intern


Best regards
[Yadav Vishal]
Bharat Intern Working Intern
**********************************************************************
*******************************************************
```