TITANIC SURVIVAL PREDICTION

NAME : - SIDDHI PRASAD GAMBHIR

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

In [2]:
```
df=pd.read_csv(r'C:\Users\Siddhi\Desktop\Titanic-Dataset.csv')
```

In [3]:
```
df
```

Out[3]:

	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

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.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
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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 [8]:
```
df.drop('Cabin',axis=1,inplace=True)
```

In [9]:
```
df['Age'].fillna(df['Age'].mean(),inplace=True)
```

In [10]:
```
df.dropna(inplace=True)
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

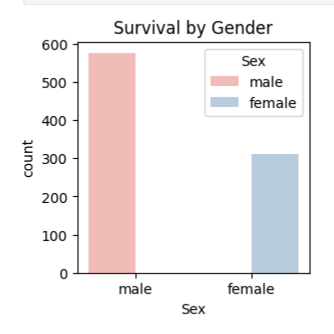
In [12]:
```
df.drop(['PassengerId','Name','Ticket','Fare'],axis=1,inplace=True)
```

In [13]:
```
df.head()
```

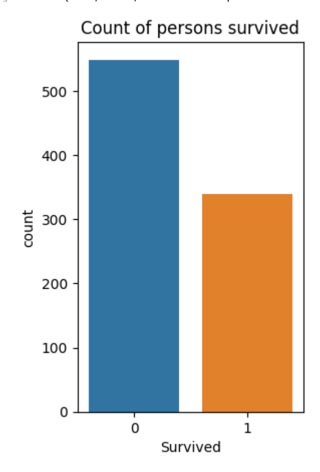
Out[13]:

	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

In [14]:
```
plt.figure(figsize = (3,3))
sns.countplot(x ="Sex", data = df, hue ="Sex", palette ="Pastel1")
plt.title("Survival by Gender")
plt.show()
plt.subplot(1,2,1)
sns.countplot(x=df['Survived'])
plt.title('Count of persons survived')
```

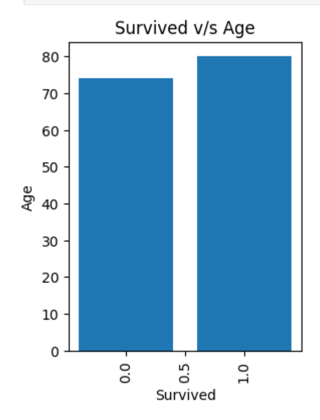


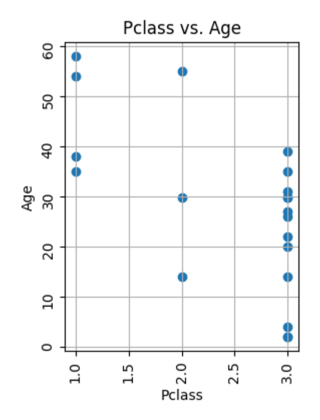
Out[14]:
```
Text(0.5, 1.0, 'Count of persons survived')
```



This both univariate graphs clearly shows Male passangers survived more than the Female passangers

In [15]:
```
x=df['Survived']
y=df['Age']
plt.figure(figsize=(3,4))
plt.bar(x,y)
plt.xticks(rotation=90)
plt.title("Survived v/s Age")
plt.xlabel('Survived')
plt.ylabel('Age')
plt.show()
plt.figure(figsize=(3, 4))
plt.scatter(x=df['Pclass'][0:20], y=df['Age'][0:20])
plt.title('Pclass vs. Age')
plt.xlabel('Pclass')
plt.ylabel('Age')
plt.xticks(rotation=90)
plt.yticks(rotation=90)
plt.grid()
plt.show()
```





In this both bivariate graph represents the survival by age and the passangers ages with their different classes.

STATISTICAL INFORMATION OF DATASET

In [16]:
```
df.describe()
```

Out[16]:

	Survived	Pclass	Age	SibSp	Parch
count	889.000000	889.000000	889.000000	889.000000	889.000000
mean	0.382452	2.311586	29.653446	0.524184	0.382452
std	0.486260	0.834700	12.968366	1.103705	0.806761
min	0.000000	1.000000	0.420000	0.000000	0.000000
25%	0.000000	2.000000	22.000000	0.000000	0.000000
50%	0.000000	3.000000	29.699118	0.000000	0.000000
75%	1.000000	3.000000	35.000000	1.000000	0.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000

In [17]:
```
x=df.iloc[:,1:]
x.head()
```

Out[17]:

	Pclass	Sex	Age	SibSp	Parch	Embarked
0	3	male	22.0	1	0	S
1	1	female	38.0	1	0	C
2	3	female	26.0	0	0	S
3	1	female	35.0	1	0	S
4	3	male	35.0	0	0	S

In [18]:
```
y=df['Survived']
y.head()
```

Out[18]:
```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

In [19]:
```
col=x.select_dtypes(['int','float']).columns
col
```

Out[19]:
```
Index(['Pclass', 'Age', 'SibSp', 'Parch'], dtype='object')
```

In [20]:
```
from scipy.stats import skew
skew(y)
```

Out[20]:
```
0.4837496405947267
```

In [21]:
```
skew(x[['Pclass','Age','SibSp','Parch']])
```

Out[21]:
```
array([-0.63592246,  0.43099149,  3.68482683,  2.74052607])
```

In [22]:
```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
```

In [23]:
```
oe.fit_transform(x[['Embarked','Sex']])
```

Out[23]:
```
array([[2., 1.],
       [0., 0.],
       [2., 0.],
       ...,
       [2., 0.],
       [0., 1.],
       [1., 1.]])
```

In [24]:
```
catcol = x.select_dtypes(object).columns
x[catcol] = oe.fit_transform(x[catcol])
x.head()
```

Out[24]:

	Pclass	Sex	Age	SibSp	Parch	Embarked
0	3	1.0	22.0	1	0	2.0
1	1	0.0	38.0	1	0	0.0
2	3	0.0	26.0	0	0	2.0
3	1	0.0	35.0	1	0	2.0
4	3	1.0	35.0	0	0	2.0

In [25]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

In [26]:
```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()
logreg.fit(xtrain,ytrain)
ypred=logreg.predict(xtest)
```

In [27]:
```
from sklearn.metrics import classification_report, accuracy_score
```

In [28]:
```
cr=classification_report(ytest,ypred)
print(cr)
ac = accuracy_score(ytest,ypred)
print("Accuracy score : ",ac)
```

```
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       166
           1       0.78      0.78      0.78       101

    accuracy                           0.84       267
   macro avg       0.82      0.82      0.82       267
weighted avg       0.84      0.84      0.84       267

Accuracy score :  0.8352059925093633
```

We have achieved an Average Accuracy of 84 % which is almost good. Lets see, if we can increase this accuracy by hyper tuning.

In [29]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [30]:
```
dt = DecisionTreeClassifier()
```

In [31]:
```
def mymodel(model):
 model.fit(xtrain,ytrain)
 ypred = model.predict(xtest)
 print(accuracy_score(ytest,ypred))
 print(classification_report(ytest,ypred))

 return model
```

In [32]:
```
mymodel(dt)
```

```
0.8014981273408239
              precision    recall  f1-score   support

           0       0.85      0.83      0.84       166
           1       0.73      0.75      0.74       101

    accuracy                           0.80       267
   macro avg       0.79      0.79      0.79       267
weighted avg       0.80      0.80      0.80       267
```

Out[32]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

By using Decision Tree, we get accuracy of 79% which is not good but lets check whether we get more accurcy by hyper tunning.

In this phase of EDA, after finding the skewness of all Numerical features understand that no need to remove it. and it does not impact the model's predictive performance.

In [33]:
```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
ada.fit(xtrain,ytrain)
ypred = ada.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.86       166
           1       0.77      0.78      0.78       101

    accuracy                           0.83       267
   macro avg       0.82      0.82      0.82       267
weighted avg       0.83      0.83      0.83       267
```

By using Gradient Boosting algorithm we get 84% of accuracy.

XG Boosting

In [34]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\siddhi\appdata\local\programs\python\python311\lib\site-packages (2.0.3)
Requirement already satisfied: numpy in c:\users\siddhi\appdata\local\programs\python\python311\lib\site-packages (from xgboost) (1.25.2)
Requirement already satisfied: scipy in c:\users\siddhi\appdata\local\programs\python\python311\lib\site-packages (from xgboost) (1.11.2)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 23.2.1 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [35]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(xtrain,ytrain)
ypred = xgb.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.83      0.84       166
           1       0.73      0.75      0.74       101

    accuracy                           0.80       267
   macro avg       0.79      0.79      0.79       267
weighted avg       0.80      0.80      0.80       267
```

By using XG Boost algorithm, we get 80% of accuracy.

BAGGING

In [36]:
```
from sklearn.ensemble import BaggingClassifier
bg = BaggingClassifier(LogisticRegression())
bg.fit(xtrain,ytrain)
ypred = bg.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.86       166
           1       0.77      0.78      0.77       101

    accuracy                           0.83       267
   macro avg       0.82      0.82      0.82       267
weighted avg       0.83      0.83      0.83       267
```

By using Bagging Classifier on Logostic Regression, we get 82% of accuracy which is not good for prediction.

In [37]:
```
bg = BaggingClassifier(DecisionTreeClassifier())
bg.fit(xtrain,ytrain)
ypred = bg.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.86      0.86       166
           1       0.77      0.75      0.76       101

    accuracy                           0.82       267
   macro avg       0.81      0.81      0.81       267
weighted avg       0.82      0.82      0.82       267
```

By using Bagging Classifier on Decision Tree we get 83% of accuracy.

In [38]:
```
models =[]
models.append(("lr",LogisticRegression()))
models.append(("dt",DecisionTreeClassifier()))
```

In [39]:
```
from sklearn.ensemble import VotingClassifier
vc = VotingClassifier(estimators = models) # estimators --> model name
vc.fit(xtrain,ytrain)
ypred = vc.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.93      0.88       166
           1       0.85      0.69      0.77       101

    accuracy                           0.84       267
   macro avg       0.84      0.81      0.82       267
weighted avg       0.84      0.84      0.83       267
```

In [40]:
```
from sklearn.ensemble import VotingClassifier
vc = VotingClassifier(estimators = models,voting='soft') # estimators --> model na
vc.fit(xtrain,ytrain)
ypred = vc.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.84      0.85       166
           1       0.75      0.75      0.75       101

    accuracy                           0.81       267
   macro avg       0.80      0.80      0.80       267
weighted avg       0.81      0.81      0.81       267
```

By using Voting Classifier, we get 84% (hard voting) and 82% (soft voting) of of accuracy.

CONCLUSION

Based on the above all Algorithm of accuracy scores, we should go ahead with ,Gradient Boosting or Logistics Regression.

Thank You

In [ ]: