Task 1 : Titanic Classification

About the Dataset:
On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there were not enough lifeboats for everyone on board, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

In this challenge, we ask you to build a predictive model that answers the question: “what sorts of people were more likely to survive?” using passenger data (ie name, age, gender, socio-economic class, etc).

Objective:

Understand the Dataset & cleanup (if required).
Build a strong classification model to predict whether the passenger survives or not.
Also fine-tune the hyperparameters & compare the evaluation metrics of various classification algorithms.

link to the Dataset : https://www.kaggle.com/datasets/yasserh/titanic-dataset

DATA PREPARATION

```
#Importing Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

```
#Loading the Dataset
titanic = pd.read_csv('Titanic-Dataset.csv')
titanic
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53

```
#Reading first 5 rows
titanic.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence	female	38.0	1	0	PC 17599	71.2

```
#Reading last 5 rows
titanic.tail()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00
				Graham,						

```
#Showing the number of rows and columns of dataset
titanic.shape
```

```
(891, 12)
```

```
#Checking for columns
titanic.columns
```

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

```
DATA PREPROCESSING AND DATA CLEANING
```

```
#Checking for data types
titanic.dtypes
```

```
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
```

```
#Checking for duplicated values
titanic.duplicated().sum()
```

```
0
```

```
#Checking for null values
nv = titanic.isna().sum().sort_values(ascending=False)
nv = nv[nv>0]
nv
```

```
Cabin       687
Age         177
Embarked      2
dtype: int64
```

```
#Cheecking what percentage column contain missing values
titanic.isnull().sum().sort_values(ascending=False)*100/len(titanic)
```

```
Cabin          77.104377
Age            19.865320
Embarked        0.224467
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
dtype: float64
```

```
#Since Cabin Column has more than 75 % null values .So , we will drop this column
titanic.drop(columns = 'Cabin', axis = 1, inplace = True)
titanic.columns
```

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

```
#Filling Null Values in Age column with mean values of age column
titanic['Age'].fillna(titanic['Age'].mean(),inplace=True)
```

```
#Filling null values in Embarked Column with mode values of embarked column
titanic['Embarked'].fillna(titanic['Embarked'].mode()[0],inplace=True)
```

```
#Checking for null values
titanic.isna().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
```

```
#Finding no. of unique values in each column of dataset
titanic[['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Embarked']].nunique().sort_values()
```

```
Survived         2
Sex              2
Pclass           3
Embarked         3
SibSp            7
Parch            7
Age             89
Fare           248
Ticket         681
PassengerId    891
Name           891
dtype: int64
```

```
titanic['Survived'].unique()
```

```
array([0, 1])
```

```
titanic['Sex'].unique()
```

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

```
titanic['Pclass'].unique()
```

```
array([3, 1, 2])
```

```
titanic['SibSp'].unique()
```

```
array([1, 0, 3, 4, 2, 5, 8])
```

```
titanic['Parch'].unique()
```

```
array([0, 1, 2, 5, 3, 4, 6])
```

```
titanic['Embarked'].unique()
```

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

```
#Dropping Some Unnecessary Columns
titanic.drop(columns=['PassengerId','Name','Ticket'],axis=1,inplace=True)
titanic.columns
```

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

```
#Showing inforamation about the dataset
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
```

```
 2   Sex       891 non-null    object
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
 7   Embarked  891 non-null    object
dtypes: float64(2), int64(4), object(2)
memory usage: 55.8+ KB
```

```
#Showing info. about numerical columns
titanic.describe()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	13.002015	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	22.000000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
#Showing info. about categorical columns
titanic.describe(include='O')
```

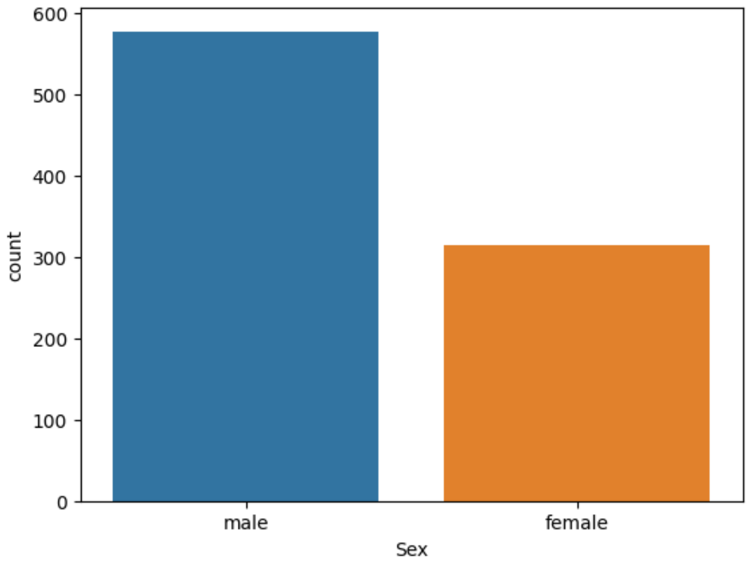
	Sex	Embarked
count	891	891
unique	2	3
top	male	S
freq	577	646

```
DATA VISUALIZATION
```

```
d1 = titanic['Sex'].value_counts()
d1
```

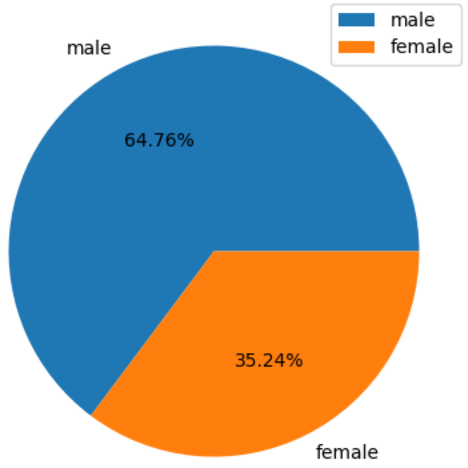
```
male      577
female    314
Name: Sex, dtype: int64
```

```
#Plotting Count plot for sex column
sns.countplot(x=titanic['Sex'])
plt.show()
```

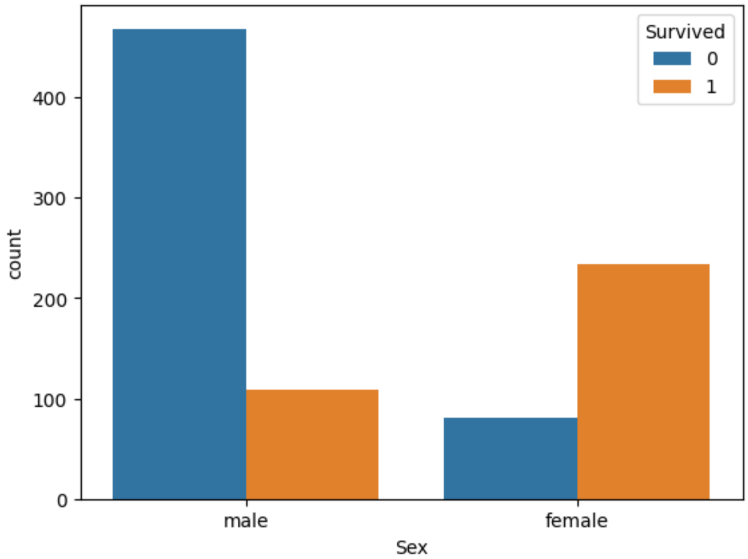



```
#Plotting Percentage Distribution of Sex Column
plt.figure(figsize=(5,5))
plt.pie(d1.values,labels=d1.index,autopct='%.2f%%')
```

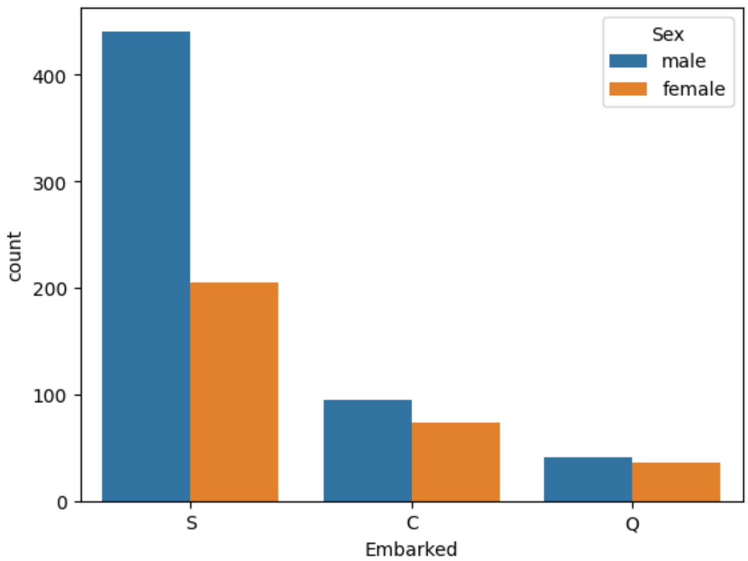
```
plt.pie(df.values,labels=df.index,autopct='%.2f%%')
plt.legend()
plt.show()
```



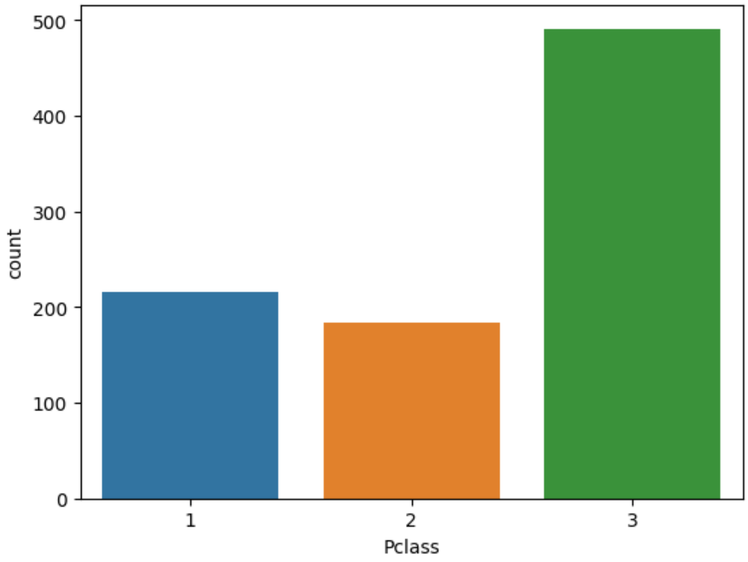

```
#Showing Distribution of Sex Column Survived Wise
sns.countplot(x=titanic['Sex'],hue=titanic['Survived']) # In Sex (0 represents female and 1 represents male)
plt.show()
```



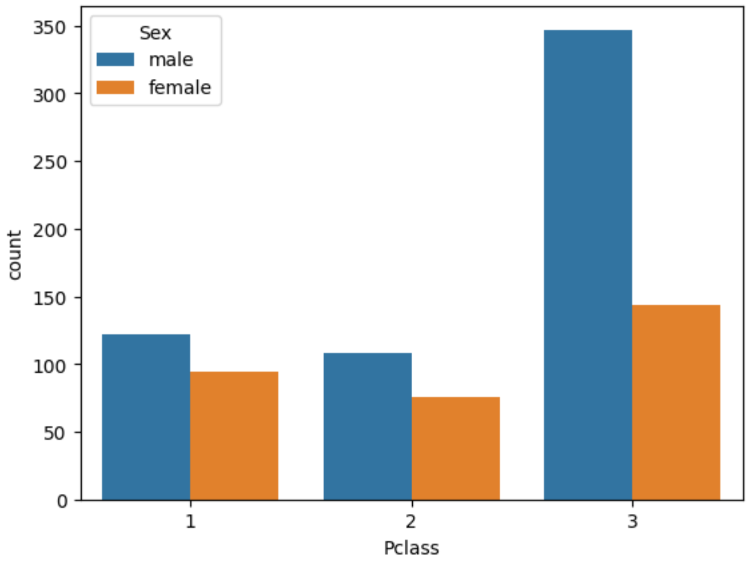

```
#Showing Distribution of Embarked Sex wise
sns.countplot(x=titanic['Embarked'],hue=titanic['Sex'])
plt.show()
```



```
#Plotting CountPlot for Pclass Column
sns.countplot(x=titanic['Pclass'])
plt.show()
```



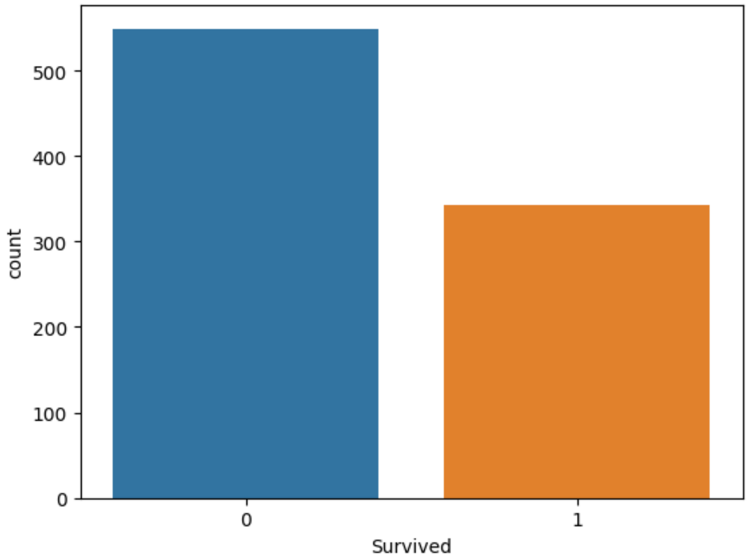

```
#Showing Distribution of Pclass Sex wise
sns.countplot(x=titanic['Pclass'],hue=titanic['Sex'])
plt.show()
```



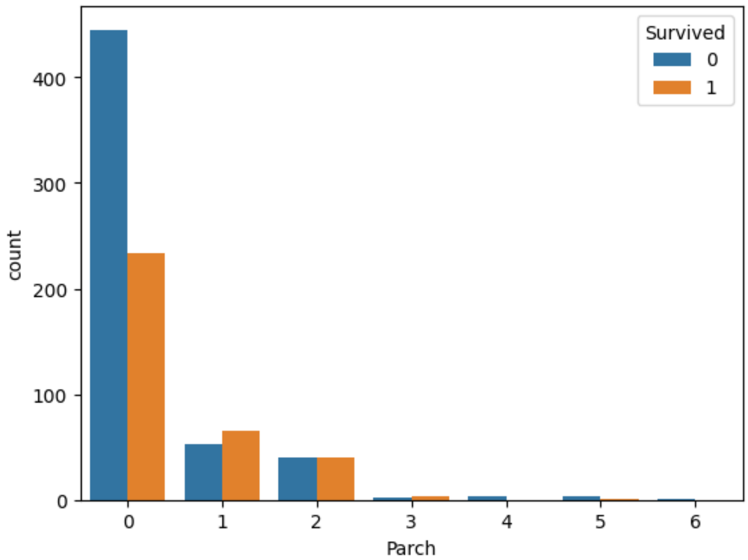

```
#Age Distribution
sns.kdeplot(x=titanic['Age'])
plt.show()
```

```
#Plotting CountPlot for Survived Column
print(titanic['Survived'].value_counts())
sns.countplot(x=titanic['Survived'])
plt.show()
```

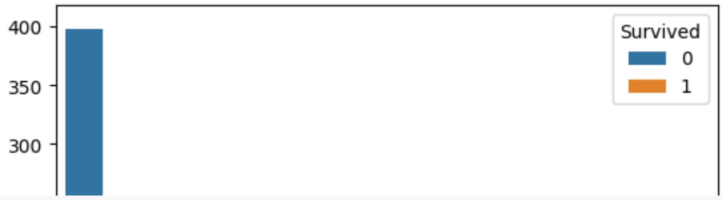
```
0    549
1    342
Name: Survived, dtype: int64
```



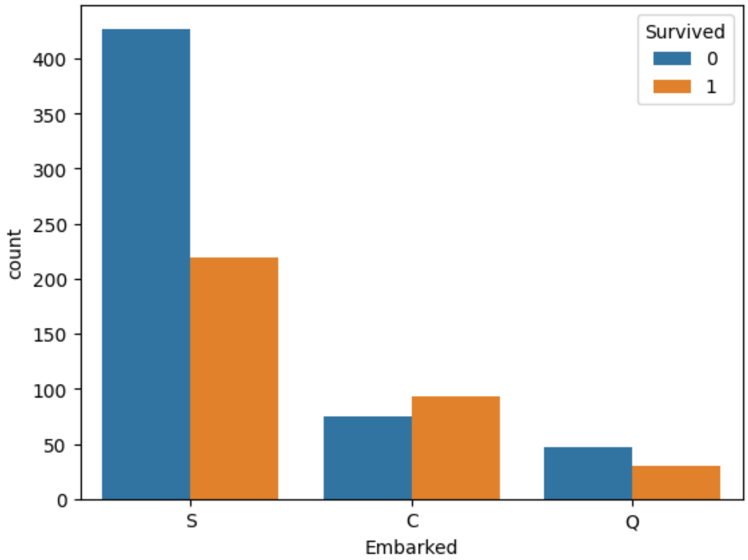

```
#Showing Distribution of Parch Survived Wise
sns.countplot(x=titanic['Parch'],hue=titanic['Survived'])
plt.show()
```



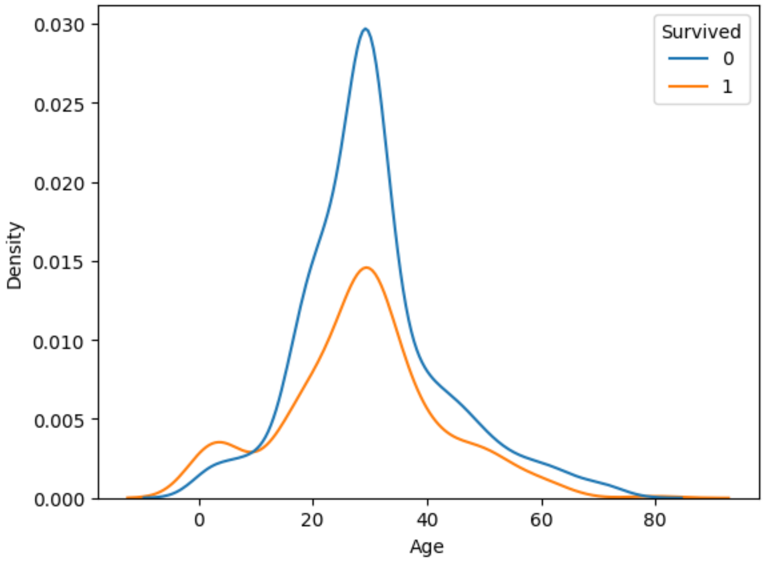

```
#Showing Distribution of SibSp Survived Wise
sns.countplot(x=titanic['SibSp'],hue=titanic['Survived'])
plt.show()
```



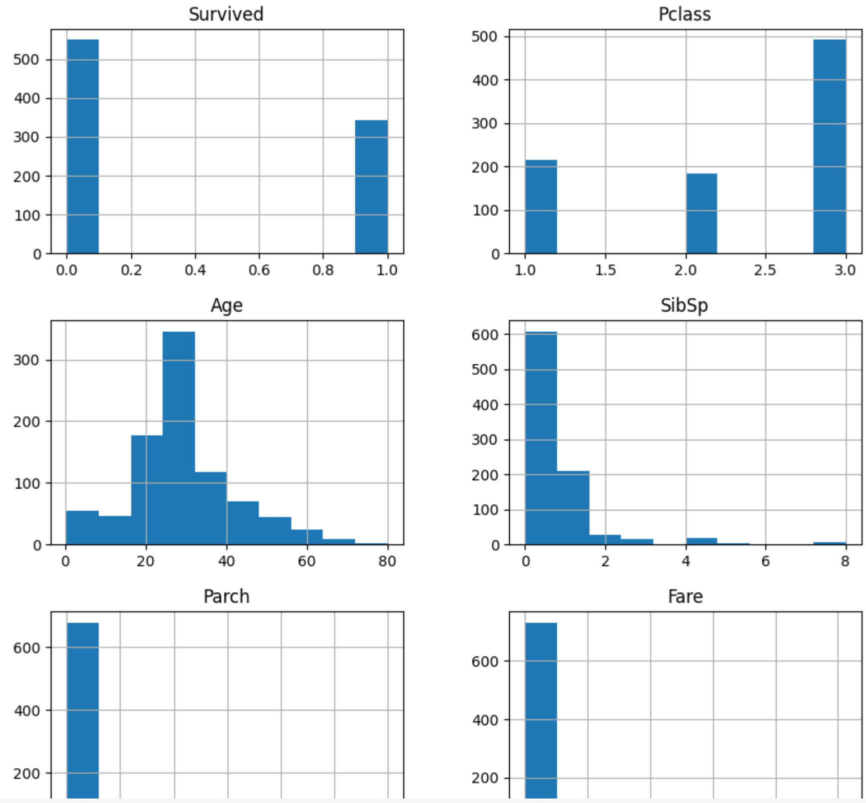

```
#Showing Distribution of Embarked Survived wise
sns.countplot(x=titanic['Embarked'],hue=titanic['Survived'])
plt.show()
```



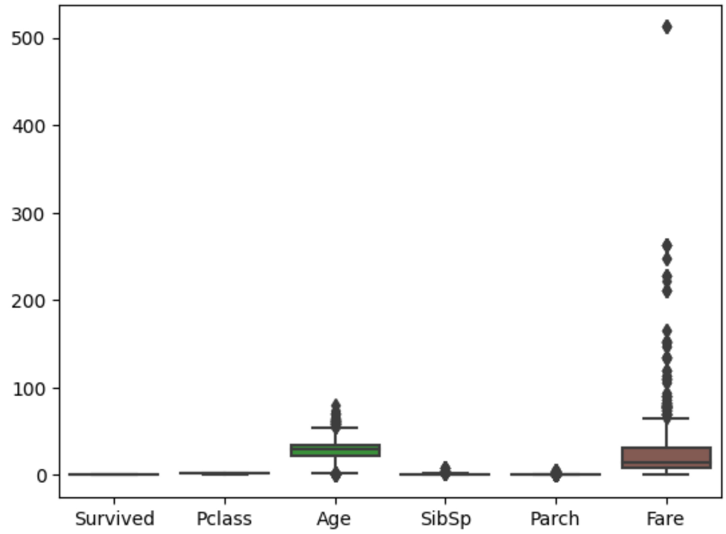

```
#Showinf Distribution of Age Survived Wise
sns.kdeplot(x=titanic['Age'],hue=titanic['Survived'])
plt.show()
```




```
#Plotting Histplot for Dataset
titanic.hist(figsize=(10,10))
plt.show()
```



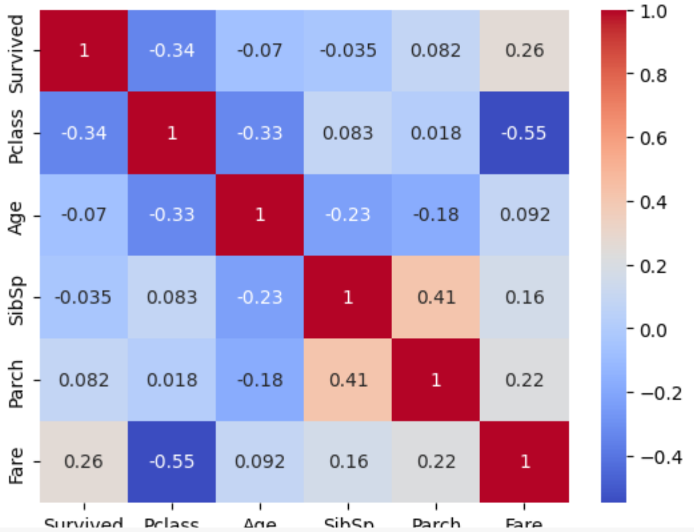

```
# Plotting Boxplot for dataset
# Checking for outliers
sns.boxplot(titanic)
plt.show()
```



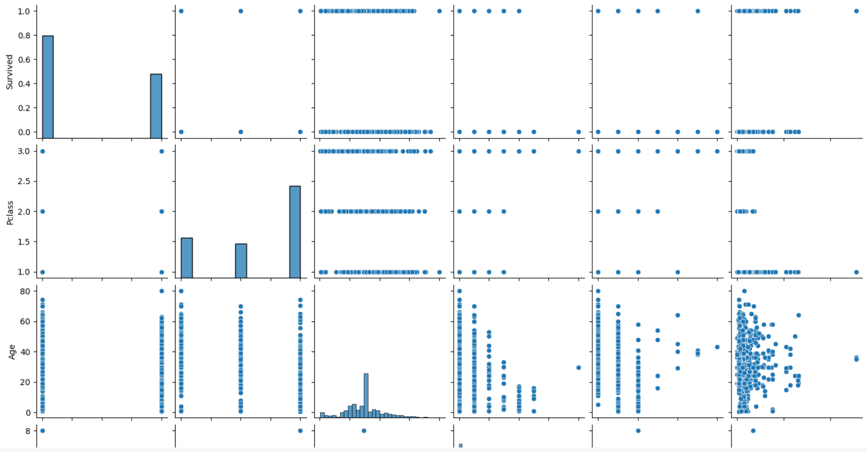

```
# showing Correlation
titanic.corr()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
Survived	1.000000	-0.338481	-0.069809	-0.035322	0.081629	0.257307
Pclass	-0.338481	1.000000	-0.331339	0.083081	0.018443	-0.549500
Age	-0.069809	-0.331339	1.000000	-0.232625	-0.179191	0.091566
SibSp	-0.035322	0.083081	-0.232625	1.000000	0.414838	0.159651
Parch	0.081629	0.018443	-0.179191	0.414838	1.000000	0.216225
Fare	0.257307	-0.549500	0.091566	0.159651	0.216225	1.000000

```
# Showing Correlation Plot
sns.heatmap(titanic.corr(),annot=True,cmap='coolwarm')
plt.show()
```



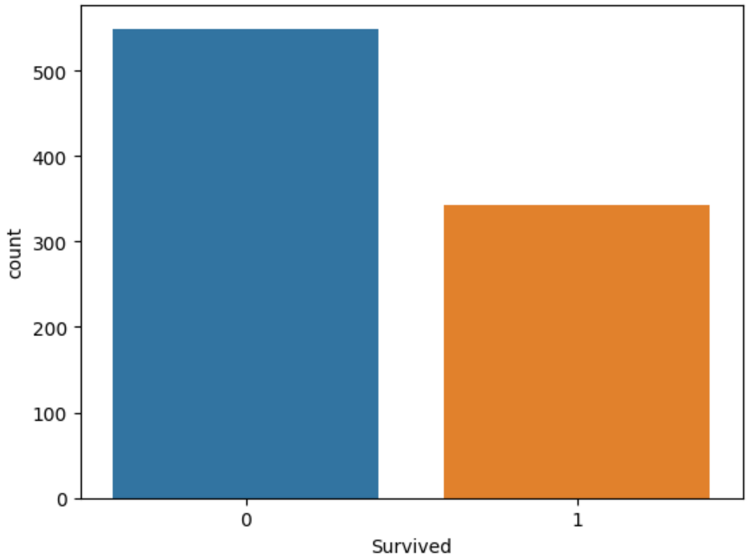

```
# Plotting pairplot
sns.pairplot(titanic)
plt.show()
```




```
#Checking the target variable
titanic['Survived'].value_counts()
```

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

```
sns.countplot(x=titanic['Survived'])
plt.show()
```




```
LABEL ENCODING
```

```
from sklearn.preprocessing import LabelEncoder
#Create an instance of LabelEncoder
le = LabelEncoder()

#Apply label encoding to each categorical column
for column in ['Sex','Embarked']:
    titanic[column] = le.fit_transform(titanic[column])

titanic.head()

#Sex Column

#0 represents female
#1 represents Male

#Embarked Column

#0 represents C
#1 represents Q
#2 represents S
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.0	1	0	7.2500	2
1	1	1	0	38.0	1	0	71.2833	0

```
DATA MODELLING
```

3	1	1	0	35.0	1	0	53.1000	2

```
#Importing libraries

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
```

```
#Selecting the independent and dependent Features
cols = ['Pclass','Sex','Age','SibSp','Parch','Fare','Embarked']
x = titanic[cols]
y = titanic['Survived']
print(x.shape)
print(y.shape)
print(type(x))  # DataFrame
print(type(y))  # Series
```

```
(891, 7)
(891,)
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

```
x.head()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	1	22.0	1	0	7.2500	2
1	1	0	38.0	1	0	71.2833	0
2	3	0	26.0	0	0	7.9250	2
3	1	0	35.0	1	0	53.1000	2
4	3	1	35.0	0	0	8.0500	2

```
y.head()
```

```
0    0
1    1
2    1
3    1
4    0
Name: Survived, dtype: int64
```

```
#Train_Test_Split
print(891*0.10)
```

```
89.10000000000001
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.10,random_state=1)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(801, 7)
(90, 7)
(801,)
(90,)
```

```
#Creating Functions to compute Confusion Matrix, Classification Report and to generate Training and the Testing Score(Accuracy)
def cls_eval(ytest,ypred):
    cm = confusion_matrix(ytest,ypred)
    print('Confusion Matrix\n',cm)
    print('Classification Report\n',classification_report(ytest,ypred))

def mscore(model):
    print('Training Score',model.score(x_train,y_train))  # Training Accuracy
    print('Testing Score',model.score(x_test,y_test))     # Testing Accuracy
```

```
    print('Testing Score',model.score(x_test,y_test))     # Testing Accuracy
```

```
# Building the logistic Regression Model
lr = LogisticRegression(max_iter=1000,solver='liblinear')
lr.fit(x_train,y_train)
```

```
▾                 LogisticRegression
LogisticRegression(max_iter=1000, solver='liblinear')
```

```
#Computing Training and Testing score
mscore(lr)
```

```
Training Score 0.8052434456928839
Testing Score 0.7666666666666667
```

```
#Generating Prediction
ypred_lr = lr.predict(x_test)
print(ypred_lr)
```

```
[1 0 1 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 1 0 1 1 0 0 1 0 0 0
 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0
 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1]
```

```
#Evaluate the model - confusion matrix, classification Report, Accuracy score
cls_eval(y_test,ypred_lr)
acc_lr = accuracy_score(y_test,ypred_lr)
print('Accuracy Score',acc_lr)
```

```
Confusion Matrix
 [[46  7]
 [14 23]]
Classification Report
               precision    recall  f1-score   support

           0       0.77      0.87      0.81        53
           1       0.77      0.62      0.69        37

    accuracy                           0.77        90
   macro avg       0.77      0.74      0.75        90
weighted avg       0.77      0.77      0.76        90

Accuracy Score 0.7666666666666667
```

```
#Building the knnClassifier Model
knn=KNeighborsClassifier(n_neighbors=8)
knn.fit(x_train,y_train)
```

```
▾       KNeighborsClassifier
KNeighborsClassifier(n_neighbors=8)
```

```
#Computing Training and Testing score
mscore(knn)
```

```
Training Score 0.7752808988764045
Testing Score 0.6777777777777778
```

```
#Generating Prediction
ypred_knn = knn.predict(x_test)
print(ypred_knn)
```

```
[1 0 0 1 1 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0
 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0]
```

```
#Evaluate the model - confusion matrix, classification Report, Accuracy score
cls_eval(y_test,ypred_knn)
acc_knn = accuracy_score(y_test,ypred_knn)
print('Accuracy Score',acc_knn)
```

```
Confusion Matrix
 [[47  6]
 [23 14]]
Classification Report
               precision    recall  f1-score   support

           0       0.67      0.89      0.76        53
           1       0.70      0.38      0.49        37

    accuracy                           0.68        90
```

```
   macro avg       0.69      0.63      0.63        90
weighted avg       0.68      0.68      0.65        90

Accuracy Score 0.6777777777777778
```

```
#Building Support Vector Classifier Model
svc = SVC(C=1.0)
svc.fit(x_train, y_train)
```

```
▾ SVC
SVC()
```

```
#Computing Training and Testing score
mscore(svc)
```

```
Training Score 0.6891385767790262
Testing Score 0.6333333333333333
```

```
#Generating Prediction
ypred_svc = svc.predict(x_test)
print(ypred_svc)
```

```
[0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0]
```

```
#Evaluate the model - confusion matrix, classification Report, Accuracy score
cls_eval(y_test,ypred_svc)
acc_svc = accuracy_score(y_test,ypred_svc)
print('Accuracy Score',acc_svc)
```

```
Confusion Matrix
 [[48  5]
 [28  9]]
Classification Report
               precision    recall  f1-score   support

           0       0.63      0.91      0.74        53
           1       0.64      0.24      0.35        37

    accuracy                           0.63        90
   macro avg       0.64      0.57      0.55        90
weighted avg       0.64      0.63      0.58        90

Accuracy Score 0.6333333333333333
```

```
#Building the RandomForest Classifier Model
rfc=RandomForestClassifier(n_estimators=80,criterion='entropy',min_samples_split=5,max_depth=10)
rfc.fit(x_train,y_train)
```

```
▾                          RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_split=5,
                       n_estimators=80)
```

```
#Computing Training and Testing score
mscore(rfc)
```

```
Training Score 0.9188514357053683
Testing Score 0.7777777777777778
```

```
#Generating Prediction
ypred_rfc = rfc.predict(x_test)
print(ypred_rfc)
```

```
[1 0 1 1 1 0 0 1 1 1 0 0 0 0 1 1 0 0 0 1 0 0 1 0 1 0 1 1 0 1 1 0 0 0 0 0 1
 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1]
```

```
#Evaluate the model - confusion matrix, classification Report, Accuracy score
cls_eval(y_test,ypred_rfc)
acc_rfc = accuracy_score(y_test,ypred_rfc)
print('Accuracy Score',acc_rfc)
```

```
Confusion Matrix
 [[47  6]
 [14 23]]
Classification Report
               precision    recall  f1-score   support
```

```
              0       0.77      0.89      0.82        53
              1       0.79      0.62      0.70        37

       accuracy                           0.78        90
      macro avg       0.78      0.75      0.76        90
   weighted avg       0.78      0.78      0.77        90

Accuracy Score 0.7777777777777778
```

```
#Building the DecisionTree Classifier Model
dt = DecisionTreeClassifier(max_depth=5,criterion='entropy',min_samples_split=10)
dt.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_split=10)
```

```
#Computing Training and Testing score
mscore(dt)
```

```
Training Score 0.8526841448189763
Testing Score 0.7777777777777778
```

```
#Generating Prediction
ypred_dt = dt.predict(x_test)
print(ypred_dt)
```

```
[1 0 1 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 1 0 1 1 0 0 1 0 0 1
 0 0 0 1 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0
 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1]
```

```
#Evaluate the model - confusion matrix, classification Report, Accuracy score
cls_eval(y_test,ypred_dt)
acc_dt = accuracy_score(y_test,ypred_dt)
print('Accuracy Score',acc_dt)
```

```
Confusion Matrix
 [[46  7]
 [13 24]]
Classification Report
               precision    recall  f1-score   support

              0       0.78      0.87      0.82        53
              1       0.77      0.65      0.71        37

       accuracy                           0.78        90
      macro avg       0.78      0.76      0.76        90
   weighted avg       0.78      0.78      0.77        90

Accuracy Score 0.7777777777777778
```

```
#Builing the Adaboost model
ada_boost  = AdaBoostClassifier(n_estimators=80)
ada_boost.fit(x_train,y_train)
```

```
AdaBoostClassifier
AdaBoostClassifier(n_estimators=80)
```

```
#Computing the Training and Testing Score
mscore(ada_boost)
```

```
Training Score 0.8564294631710362
Testing Score 0.7666666666666667
```

```
#Generating the predictions
ypred_ada_boost = ada_boost.predict(x_test)
```

```
#Evaluate the model - confusion matrix, classification Report, Accuracy Score
cls_eval(y_test,ypred_ada_boost)
acc_adab = accuracy_score(y_test,ypred_ada_boost)
print('Accuracy Score',acc_adab)
```

```
Confusion Matrix
 [[45  8]
 [13 24]]
Classification Report
               precision    recall  f1-score   support
```

```
               0       0.78      0.85      0.81        53
               1       0.75      0.65      0.70        37

        accuracy                           0.77        90
       macro avg       0.76      0.75      0.75        90
    weighted avg       0.77      0.77      0.76        90

    Accuracy Score 0.7666666666666667
```

```
models = pd.DataFrame({
    'Model': ['Logistic Regression','knn','SVC','Random Forest Classifier','Decision Tree Classifier','Ada Boost Classifier'],
    'Score': [acc_lr,acc_knn,acc_svc,acc_rfc,acc_dt,acc_adab]})

models.sort_values(by = 'Score', ascending = False)
```

	Model	Score
3	Random Forest Classifier	0.777778
4	Decision Tree Classifier	0.777778
0	Logistic Regression	0.766667
5	Ada Boost Classifier	0.766667
1	knn	0.677778
2	SVC	0.633333

```
colors = ["blue", "green", "red", "yellow","orange","purple"]

sns.set_style("whitegrid")
plt.figure(figsize=(15,5))
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
sns.barplot(x=models['Model'],y=models['Score'], palette=colors )
plt.show()
```