Machine Learning with the Titanic Dataset

An end-to-end guide to predict the Survival of Titanic passenger

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, embarkment 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.

Problem Overview:

The sinking of the Titanic is one of the most deadliest tragedy in history.

RMS *Titanic* , in full **Royal Mail Ship (RMS) Titanic**, British luxury passenger liner, operated by the White Star Line, which sank in the North Atlantic Ocean on 15 April 1912 after striking an iceberg during her maiden voyage from Southampton, UK, to New York City.

Of the estimated 2,224 passengers and crew aboard, more than 1,500 died, which made the sinking possibly one of the deadliest for a single ship up to that time.

While there were some factors which made impact on the survival of passenger , in this problem we need to predict the survival of passenger depending on the factors provided in dataset .

Model Building Steps :

1. Analise and visualize the Dataset
2. Clean and prepare the dataset for our ML model
3. Build & Train Our Model
4. Calculate the Accuracy for the model
5. Prepare the submission file

Import the required Libraries

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Load & Analyze Our Dataset

Link to download Dataset :

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

```python
titanic_df = pd.read_csv(r'titanic_train.csv')
#df=pd.read_csv(r'https://raw.githubusercontent.com/dsrscientist/dataset1/master/titanic_train.csv')
titanic_df
```

	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

Below are the features provided in the dataset:

- Passenger Id: and id given to each traveler on the boat
- Pclass: the passenger class. It has three possible values: 1,2,3 (first, second and third class)
- The Name of the passenger
- Sex
- Age
- SibSp: number of siblings and spouses traveling with the passenger
- Parch: number of parents and children traveling with the passenger
- The ticket number
- The ticket Fare

- The cabin number
- The embarkation. This describe three possible areas of the Titanic from which the people embark. Three possible values S,C,Q
- Survived : 1 for survived and 0 for died

```
1 titanic_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
```

we can see dataset is combination of both categorical and continuous features:

Categorical Features :

1) Survived
2) Pclass
3) Name
4) Sex
5) Ticket
6) Cabin
7) Embarked

Continuous Features:

1) PassengerId
2) Age
3) Fare

Lets see the count of Survived and died passenger count :

```
1 titanic_df['Survived'].value_counts()
```

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

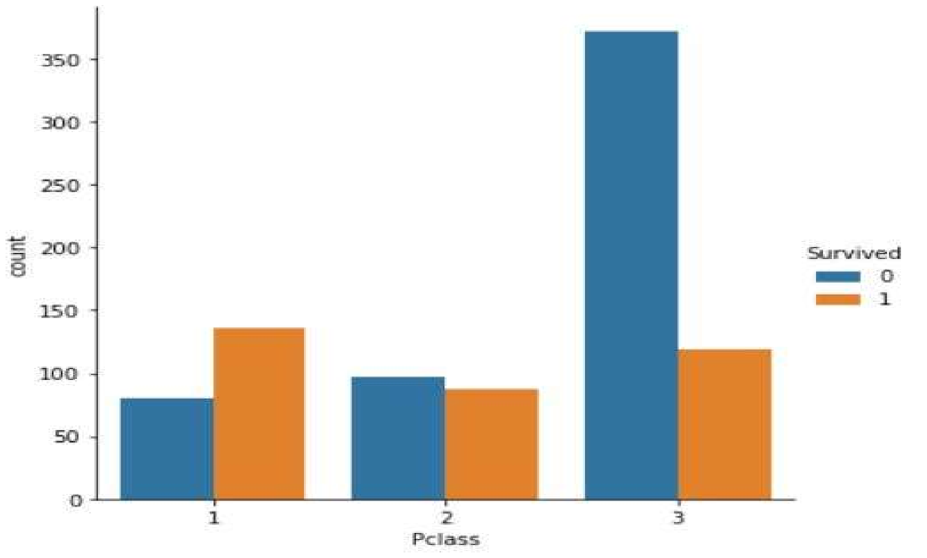
We can see around 62 :38 target distribution is there so we can consider it as imbalanced dataset .

survived(1) =342

sunk(0) = 549

Impact of Pclass on Survival:

<seaborn.axisgrid.FacetGrid at 0x1a6e1b4c220>




```
1 titanic_df['Pclass'].value_counts()
```

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

from the information we can have following observations:

1)highest no. of survival is in 1st class out of 216 passengers around 145 saved saving rate is 67%

2)Then in 3rd class out of 491 passengers around 140 saved but saving rate is 28%

3)lowest no. of people survived in 3rd class out of 184 passengers around 90 saved but saving rate is 49%

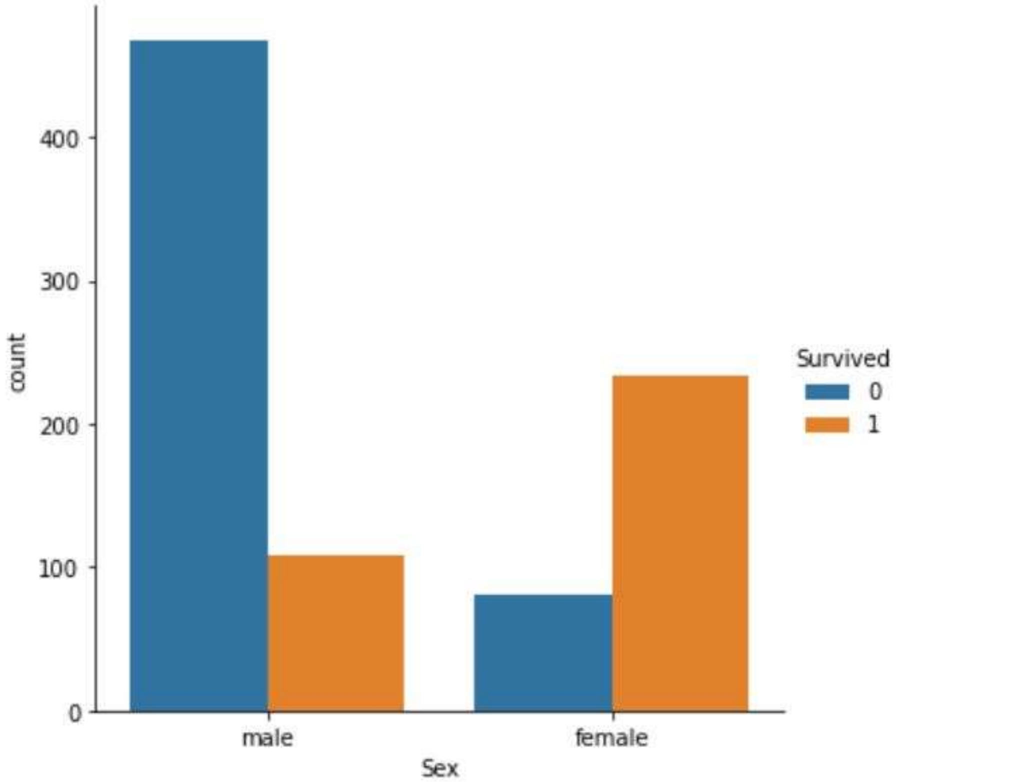
survival rate is highest in 1st class than 2nd class and lowest in 3rd class

Impact of Sex on Survival:

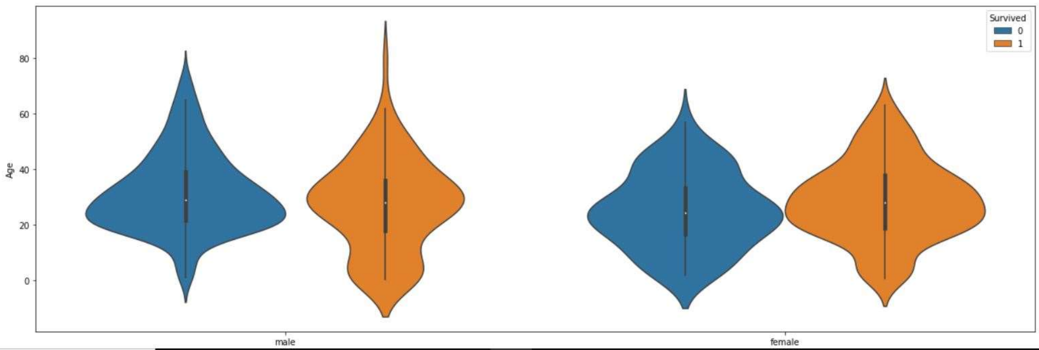
```
1 titanic_df.groupby('Survived')['Sex'].value_counts()
```

```
Survived  Sex
0         male      468
          female     81
1         female    233
          male      109
Name: Sex, dtype: int64
```

```
<seaborn.axisgrid.FacetGrid at 0x1a6e1c08ca0>
```




```
# histplot of to see distribution of servivied age
sns.violinplot(x='Sex',y='Age',hue='Survived' ,data=titanic_df)
plt.rcParams['figure.figsize']=(21,7)
```



1) Male survival rate is less than Female.

2) Survival rate in the range of 15-40 is high in both male and female.

3) it is moderate in below 15 and lowest as age increases above 40.

So, Age and sex are also some deciding survival features.

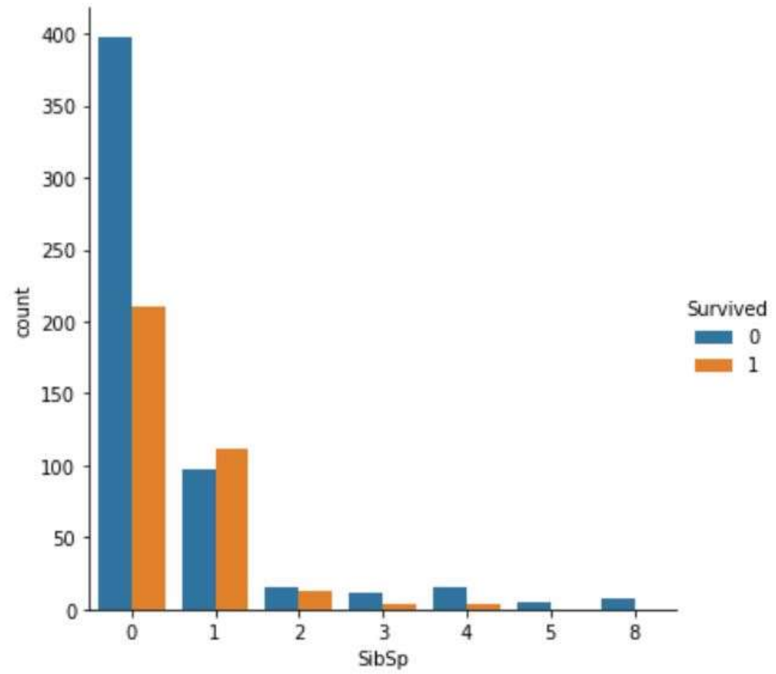
Impact of Sibsp on Survival:

```
titanic_df['SibSp'].value_counts()
```

```
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: SibSp, dtype: int64
```

```
1 # survival count of SibSp
2 sns.catplot(x='SibSp' , hue='Survived' ,data=titanic_df ,kind='count')
```

<seaborn.axisgrid.FacetGrid at 0x1a6e1ca8220>



passenger with SibSp=5 has less chances of survival , whereas passenger with SibSp=1 has highest chances of survival.

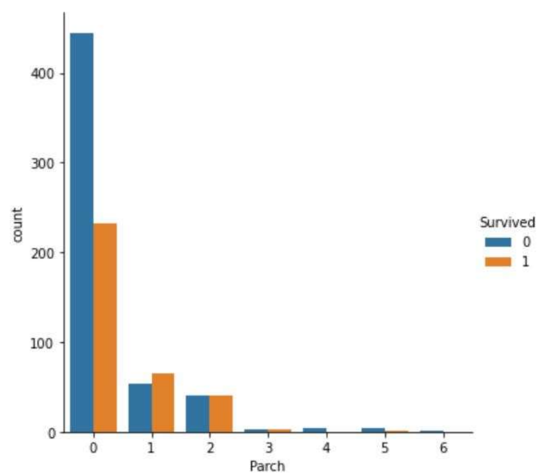
Impact of Parch on Survival:

```
1 titanic_df['Parch'].value_counts()
```

```
0    678
1    118
2     80
3      5
5      5
4      4
6      1
Name: Parch, dtype: int64
```

```
1 sns.catplot(x='Parch' , hue='Survived' ,data=titanic_df ,kind='count')
```

<seaborn.axisgrid.FacetGrid at 0x1a6e1c08dc0>

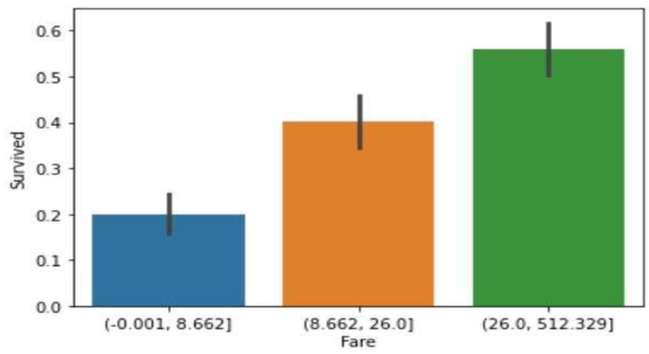


passenger with parch>3 has less chances of survival , whereas passenger with parch=1 has highest chances of survival and parch=2 has 50% chances of survive

Impact of fair on Survival:

```
df_Fair_range = pd.qcut(titanic_df['Fare'],3)
sns.barplot(x=df_Fair_range ,y='Survived'  ,data=titanic_df )
```

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



We have divided fair in 3 groups for analysis.

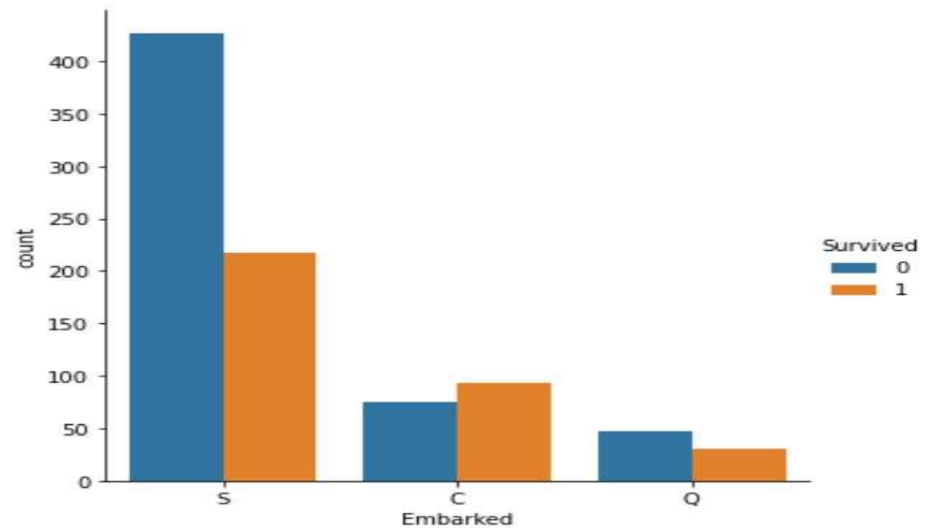
Fair and survival rate has positive correlation with each other , as fair increases survival rate increases .

That means passengers paying higher fairs having higher chnaces of survival as compared to those who are paying less.

Impact of Embarked on Survival:

```
sns.catplot(x='Embarked' ,hue='Survived' ,kind='count' ,data=titanic_df)
```

```
<seaborn.axisgrid.FacetGrid at 0x1a6e1db6ac0>
```



'C' category has highest rate of survive as compared to 'S 'and 'Q'.

Feature Selection:

we can see ['PassengerId','Name','Cabin'] are not Survival deciding features. So, we will drop them.

```
titanic_df.drop(['PassengerId','Name','Cabin'] ,axis=1 ,inplace=True)
```

Check for null entries:

```
titanic_df.isnull().sum()
```

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

Age has 177 null entries and Embarked has 2 null entries.

Filling null values in data in 'Embarked':

Since Embarked is categorical feature will fill null values with mode .

```
# calculating most frequently occured value in column
titanic_df['Embarked'].mode()
```

```
0    S
dtype: object
```

we can see in 'Embarked' most frequently occured value in column is 'S' ,so will fill null values with it

```
titanic_df['Embarked'].fillna('S',inplace=True)
titanic_df['Embarked'].isnull().sum()
```

```
0
```

Filling the null values in age column:

Since we have 177 NULL values in Age column, if we will fill it with mean /median it will make feature biased so it won't be a great idea. So, we will fill them randomly with values in range of +/- 1 std deviation from mean.

```
age_mean=titanic_df['Age'].mean()
age_mean
age_std_dev=titanic_df['Age'].std()
print("Mean of Age :",age_mean,'\nStandard deviation of Age :',age_std_dev)
```

```
Mean of Age : 29.69911764705882
Standard deviation of Age : 14.526497332334044
```

```
age_upper=round(age_mean+age_std_dev)
age_lower=round(age_mean-age_std_dev)
print("Upper and Lower limits are :",age_upper,age_lower)
```

```
Upper and Lower limits are : 44 15
```

```
index=titanic_df[titanic_df['Age'].isnull()].index.to_list()
index
```

Made a series containing 177 numbers in range 15-44 using
np.random.randint(start,end,size=length of array)

```
age_mean=titanic_df['Age'].mean()
age_mean
age_std_dev=titanic_df['Age'].std()
print("Mean of Age :",age_mean,'\nStandard deviation of Age :',age_std_dev)
```

```
Mean of Age : 29.69911764705882
Standard deviation of Age : 14.526497332334044
```

We have made list of indices containing null values by df[df[col].isnull()].index.to_list() Then we have assigned values to null index

```
age_upper=round(age_mean+age_std_dev)
age_lower=round(age_mean-age_std_dev)
print("Upper and Lower limits are :",age_upper,age_lower)
```

```
Upper and Lower limits are : 44 15
```

```
index=titanic_df[titanic_df['Age'].isnull()].index.to_list()
index
```

```
# to iterate over multiple lists at a time
import itertools
```

```

for i,j in  zip(fill ,index):
    print(i,j)
    titanic_df['Age'][j]=i
    print(titanic_df['Age'][i])
```

Checking Collinearity:

Collinearity is correlation between independent features.

```
## correlation metrics
titanic_df.corr()
```

	Survived	Pclass	Age	SibSp	Parch	Fare
Survived	1.000000	-0.338481	-0.066407	-0.035322	0.081629	0.257307
Pclass	-0.338481	1.000000	-0.324011	0.083081	0.018443	-0.549500
Age	-0.066407	-0.324011	1.000000	-0.226342	-0.172337	0.094921
SibSp	-0.035322	0.083081	-0.226342	1.000000	0.414838	0.159651
Parch	0.081629	0.018443	-0.172337	0.414838	1.000000	0.216225
Fare	0.257307	-0.549500	0.094921	0.159651	0.216225	1.000000

```
# using pearson correlation
plt.figure(figsize=(12,10))
cor=titanic_df.corr()
sns.heatmap(cor ,annot=True )
```

Checking Collinearity of target variable with other independent variables

```
titanic_df.corr()['Survived'].sort_values(ascending=False)
```

```
Survived    1.000000
Fare        0.257307
Parch       0.081629
SibSp      -0.035322
Age        -0.066407
Pclass     -0.338481
Name: Survived, dtype: float64
```

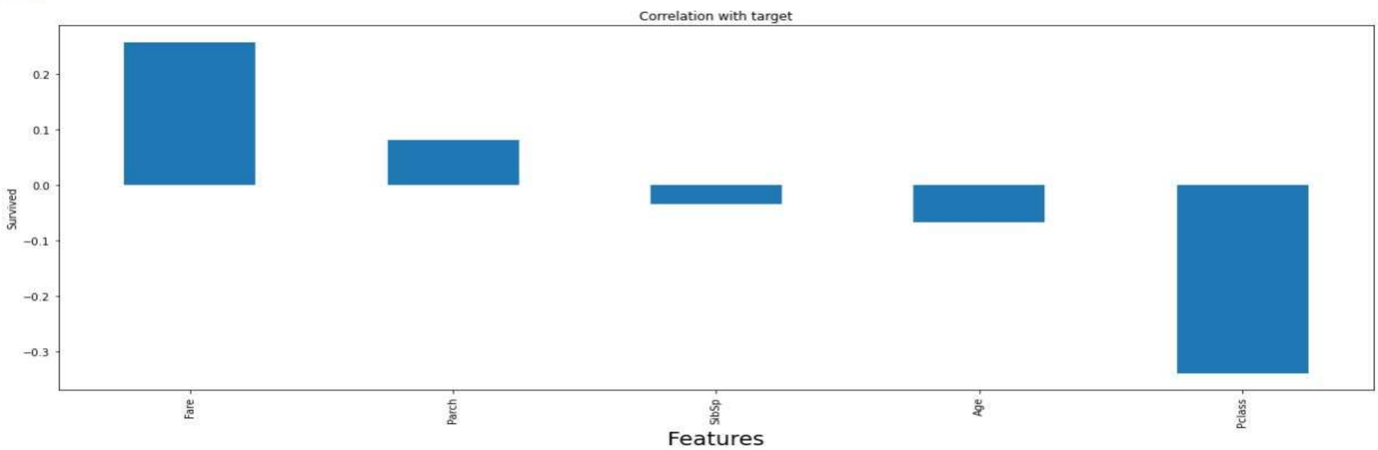
1)We can observe 'Fair'(0.25) has heighest +ve correlation with 'Survived' .Other +ve correlated feature is 'Parch' .

2)We can observe 'Pclass' (-0.33) has heighest -ve correlation with 'Survived' .Other -ve correlated features are 'Age' and 'SibSp'

Plot of correlation with target :

```
titanic_df.corr()['Survived'].sort_values(ascending=False).drop(['Survived']).plot(kind='bar')
plt.xlabel('Features',fontsize=20)
plt.ylabel('Survived')
plt.title('Correlation with target')
```

```
Text(0.5, 1.0, 'Correlation with target')
```



Data Pre-processing for 'sex' feature into a numeric value: male(1) and female(0) and 'Embarked' feature transformation values:

```
from sklearn.preprocessing import LabelEncoder
```

```
LE = LabelEncoder()
titanic_df['Sex'] = LE.fit_transform(titanic_df['Sex'])
titanic_df['Embarked'] = LE.fit_transform(titanic_df['Embarked'])
```

```
print(titanic_df['Embarked'])
```

```
print(titanic_df['Embarked'])
```

```
0      2
1      0
2      2
3      2
4      2
      ..
886    2
```

here we can find S,C and Q got transformed in 1,2 ,3 as S=2,C=0,Q =1

Sex feature transformation values:

```
print(titanic_df['Sex'])
```

```
0      1
1      0
2      0
3      0
4      1
      ..
886    1
887    0
```

Here we can find Male and Female got transformed in 0,1 as Male=1,Female=0

Splitting Data in to Train Test :

Categorical data column encoding using ordinal encoder

```
# Catagorical data column encoding using ordinal encoder
from sklearn.preprocessing import OrdinalEncoder
onc= OrdinalEncoder()
titanic_df['Ticket']=onc.fit_transform(titanic_df['Ticket'].values.reshape(-1,1))
```

Independent Features:

```
x = titanic_df.drop('Survived',axis=1)
x
```

	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	3	1	22.0	1	0	523.0	7.2500	2
1	1	0	38.0	1	0	596.0	71.2833	0
2	3	0	26.0	0	0	669.0	7.9250	2
3	1	0	35.0	1	0	49.0	53.1000	2
4	3	1	35.0	0	0	472.0	8.0500	2

Target Feature:

```
y=titanic_df['Survived']
y
```

```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
```

AS we can see target variable is having only 2 values 0,1 (yes/No) .It is a classification problem so , will try different classification models with dataset then comparing the scores will finalize one of the model for further processing.

Here we have trained our model with Logistic Regression , Random Forest Classifier , SVC , Decision Tree Classifier

Lets import required libraries :

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score ,classification_report,confusion_matrix
lr=LogisticRegression()
```

Before training the model will try to find out best value of Random State at which we will get maximum accuracy score .

```
max_accu = 0
max_randst = 0
for i in range (0,1000):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, random_state=i)
    lr.fit(x_train,y_train)

    pred_test=lr.predict(x_test)
    accu_score= accuracy_score(y_test,pred_test)
    if max_accu<accu_score:
        max_accu= accu_score
        max_randst= i

print(confusion_matrix(y_test,pred_test))
```

```
[[103  12]
 [ 29  35]]
```

```
print("max accuracy score is :", round(max_accu*100 ,1),"at random state :",max_randst)
```

```
max accuracy score is : 87.2 at random state : 455
```

Here we have got Random state value =455 at which we are getting maximum accuracy =87.2%

Now lets train the models with different classification algorithms.

```
from sklearn.metrics import accuracy_score ,classification_report,confusion_matrix
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, random_state=417)
lr.fit(x_train,y_train)

pred_test=lr.predict(x_test)
accu_score= accuracy_score(y_test,pred_test)
```

```
from sklearn.metrics import accuracy_score ,classification_report,confusion_matrix
print("Accuracy score is :",round(accu_score*100 ,2),'\n')
print("Confusion_matrix is :\n")
print(confusion_matrix(y_test,pred_test),'\n')
print("Classification report :\n")
print(classification_report(y_test,pred_test))
```

Accuracy score , confusion matrix and classification report for logistic regression is:

```
Accuracy score is : 86.59

Confusion_matrix is :

[[108   7]
 [ 17  47]]

Classification report :

              precision    recall  f1-score   support

           0       0.86      0.94      0.90       115
           1       0.87      0.73      0.80        64

    accuracy                           0.87       179
   macro avg       0.87      0.84      0.85       179
weighted avg       0.87      0.87      0.86       179
```

Random Forest Classifier Model-2

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
predrf=rf.predict(x_test)
print("Accuracy score is :",round(accuracy_score(y_test,predrf)*100 ,2),'\n')
print("Confusion_matrix is :\n")
print(confusion_matrix(y_test,predrf),'\n')
print("Classification Report score is :\n")
print(classification_report(y_test,predrf))
```

```
Accuracy score is : 88.83

Confusion_matrix is :

[[109   6]
 [ 14  50]]

Classification Report score is :

              precision    recall  f1-score   support

           0       0.89      0.95      0.92       115
           1       0.89      0.78      0.83        64

    accuracy                           0.89       179
   macro avg       0.89      0.86      0.87       179
weighted avg       0.89      0.89      0.89       179
```

We have got accuracy score for Random forest 88.83%
Will now check all these parameters for SVC .

SVC Model-3

```python
from sklearn.svm import SVC
```

```python
sv = SVC()
sv.fit(x_train,y_train)
predrf=sv.predict(x_test)
print("Accuracy score is :",round(accuracy_score(y_test,predrf)*100 ,2),'\n')
print("Confusion_matrix is :\n")
print(confusion_matrix(y_test,predrf),'\n')
print("Classification Report score is :\n")
print(classification_report(y_test,predrf))
```

```
Accuracy score is : 66.48

Confusion_matrix is :

[[102  13]
 [ 47  17]]

Classification Report score is :

              precision    recall  f1-score   support

           0       0.68      0.89      0.77       115
           1       0.57      0.27      0.36        64

    accuracy                           0.66       179
   macro avg       0.63      0.58      0.57       179
weighted avg       0.64      0.66      0.63       179
```

Decision Tree Model-4

```python
def model_implement(model_ref ,x_train,x_test,y_train,y_test):
    model_ref.fit(x_train,y_train)
    predrf=sv.predict(x_test)
    print("Accuracy score is :",round(accuracy_score(y_test,predrf)*100 ,2),'\n')
    print(confusion_matrix(y_test,predrf),'\n')
    print("Classification Report score is :\n")
    print(classification_report(y_test,predrf))
```

```python
from sklearn.tree import DecisionTreeClassifier
```

```python
dc= DecisionTreeClassifier()
model_implement(dc ,x_train,x_test,y_train,y_test)

```

```
Accuracy score is : 66.48

[[102  13]
 [ 47  17]]

Classification Report score is :

              precision    recall  f1-score   support

           0       0.68      0.89      0.77       115
           1       0.57      0.27      0.36        64

    accuracy                           0.66       179
   macro avg       0.63      0.58      0.57       179
weighted avg       0.64      0.66      0.63       179
```

Will make now data Frame to display model report

```
Model_report=pd.DataFrame()
Model_report['Model_name']=['Logistic Regression','Random Forest Classifier','SVC','Decision Tree Classifier']
Model_report['Accuracy_Score']=[87.71,85.47,65.92,70.39]
Model_report['Confusion_matrix']=['[[106   9][ 13  51]]',
                                  '[[104  11][ 15  49]]' ,
                                   '[[101  14][ 47  17]]',
                                   '[[104  11][ 42  22]]']
```

Model_report

```

Model_report
```

	Model_name	Accuracy_Score	Confusion_matrix
0	Logistic Regression	87.71	[[106 9][13 51]]
1	Random Forest Classifier	85.47	[[104 11][15 49]]
2	SVC	65.92	[[101 14][47 17]]
3	Decision Tree Classifier	70.39	[[104 11][42 22]]

Since Logistic regression is performing better in terms of Accuracy score and confusion matrix among rest of the all , we will continue with it for hyper parameter tuning

Will use GridSearchCV for hyper parameter tunning. We have made parameters gride of Logistic regression. Parameters we are considering are **solver_options** = ['newton-cg', 'lbfgs', 'liblinear', 'sag'] ,

multi_class_options = ['ovr', 'multinomial'] , **class_weight_options** = ['None', 'balanced'] .

Hyper parameter tuning

```
from sklearn.model_selection import GridSearchCV
```

```
solver_options = ['newton-cg', 'lbfgs', 'liblinear', 'sag']
multi_class_options = ['ovr', 'multinomial']
class_weight_options = ['None', 'balanced']

param_grid = dict(solver = solver_options, multi_class =
multi_class_options, class_weight = class_weight_options)
grid = GridSearchCV(lr, param_grid, cv=12, scoring =
'accuracy')
grid.fit(x_train,y_train)
grid.best_params_
```

```
{'class_weight': 'balanced', 'multi_class': 'ovr', 'solver': 'newton-cg'}
```

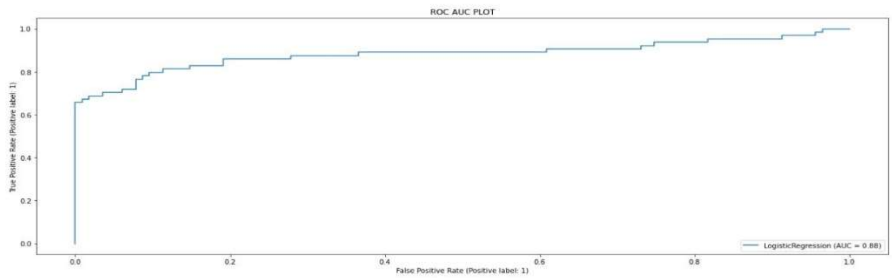
```
#prediction with best parameter
grid_pred=grid.best_estimator_.predict(x_test)
# best score
accuracy_score(y_test,grid_pred)
```

```
0.8491620111731844
```

ROC AUC plot

```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(grid.best_estimator_,x_test , y_test)
plt.title("ROC AUC PLOT")
```

```
Text(0.5, 1.0, 'ROC AUC PLOT')
```



.

We have got AUC is 81% and Accuracy is 84.9% which is very good . Will make a dump file of our model using Joblib library . we have finally got our model Pickle file . as 'Titanic_Prediction.pkl'

```
import joblib
joblib.dump(grid.best_estimator_,"Titanic_Prediction.pkl")
```

```
['Titanic_Prediction.pkl']
```

Author: Vaishali N Sonawane