Titanic Survival Prediction Project

1. Problem Definition:

The Titanic Problem is based on the sinking of the 'Unsinkable' ship Titanic in the early morning of 15th April 1912. In the ship there are total 2208 passengers, but here we get the small dataset of sample size 891 for analysis and do prediction on the basis. It gives us information about multiple features like their ages, sexes, sibling counts, embarkment points, and whether they survived or not after the disaster. Based on these features, we have to predict if an arbitrary passenger on Titanic would survive or not after sinking. In Survived column 0 represents not survived and 1 represent the survived.

In this, article we do data analysis, Exploratory Data Analysis, pre-processing pipeline, building the models, hyper tuning the best model to increase the accuracy score and then predict the passengers survival or not.

2. Data Analysis:

- In the starting, first we have to import the necessary libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import OrdinalEncoder
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import zscore
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix, roc_curve, roc_auc_score

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

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

These libraries have its own importance such as: pandas is used to read the data, numpy used for mathematical functions, seaborn used for different graphs,

matplotlib used for plotting, Ordinal encoder used to encode the object data, zscore use to remove outliers. Like this everyone has their own function.

- Then we read the data from github in csv format

```
data=pd.read_csv('https://raw.githubusercontent.com/dsrscientist/dataset1/master/titanic_train.csv')
data.head()
```

	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 Pclass 1, 2 and 3 means 1st class, 2nd class and 3rd class.

The representation of C, S and Q are in the Embarked column is Cherbourg, Southampton and Queenstown respectively.

- Check the shape and columns of the data

```
data.shape
```

```
(891, 12)
```

```
data.columns
```

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

We see that, the data is not large so it can be easy to analyse. There are 891 rows and 12 columns. In columns there are 11 independent variables and one is the target variable.

The columns of the data set are classified as:

- PassengerId = All the passenger has it's unique id while booking their tickets
- Survived = The passenger is survived or not
- Pclass = There are different- different classes in the ship (eg. 1st class, 2nd class…)
- Name = The name of the passengers
- Sex = The gender of the passengers
- Age = The age of the passengers

- SibSp = The number of spouses/siblings with the passengers
- Parch = The number of children/parents with the passengers
- Ticket = Ticket number
- Fare = The total amount of the ticket
- Cabin = There are type of cabins
- Embarked = From where the passengers get boarded

- Check the missing values

```
data.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
```

In 3 columns we get missing values, they all are less than 80%, so impute the values in replace of NaN.

- Check the data type of all the columns

```
data.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
```

PassengerId, Survived, Pclass, SibSp and Parch are integer data type. Age and Fare are of float data type.

Name, Sex, Ticket, Cabin, Embarked are the object data type.

- Imputing Technique

The age column is continuous so we applied mean() function and the Cabin and Embarked columns are of object data type so we applied mode() function.

```
# impute continuous data
data['Age']=data['Age'].fillna(data['Age'].mean())

# impute categorical data
data['Cabin']=data['Cabin'].fillna(data['Cabin'].mode()[0])
data['Embarked']=data['Embarked'].fillna(data['Embarked'].mode()[0])
```

Now the missing values get filled by fillna() method.

- Encoding Technique

The analysis is not work on object data type, so we have to convert it into float by encoding technique.

```
enc=OrdinalEncoder()

for i in data.columns:
    if data[i].dtypes=="object":
        data[i]=enc.fit_transform(data[i].values.reshape(-1,1))
```

```
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	108.0	1.0	22.0	1	0	523.0	7.2500	47.0	2.0
1	2	1	1	190.0	0.0	38.0	1	0	596.0	71.2833	81.0	0.0
2	3	1	3	353.0	0.0	26.0	0	0	669.0	7.9250	47.0	2.0
3	4	1	1	272.0	0.0	35.0	1	0	49.0	53.1000	55.0	2.0
4	5	0	3	15.0	1.0	35.0	0	0	472.0	8.0500	47.0	2.0

- The information of the dataset

From this we get the information about the columns

```
data.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    float64
 4   Sex          891 non-null    float64
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    float64
 9   Fare         891 non-null    float64
 10  Cabin        891 non-null    float64
 11  Embarked     891 non-null    float64
dtypes: float64(7), int64(5)
```

There are twelve columns and in each column there is no null value present. All the data is of integer and float type. Seven columns are of float type and five are of integer type.

3. EDA Concluding Remarks:

The data is proper for further analysis such as visualization, outliers, skewness, multicolinearity.

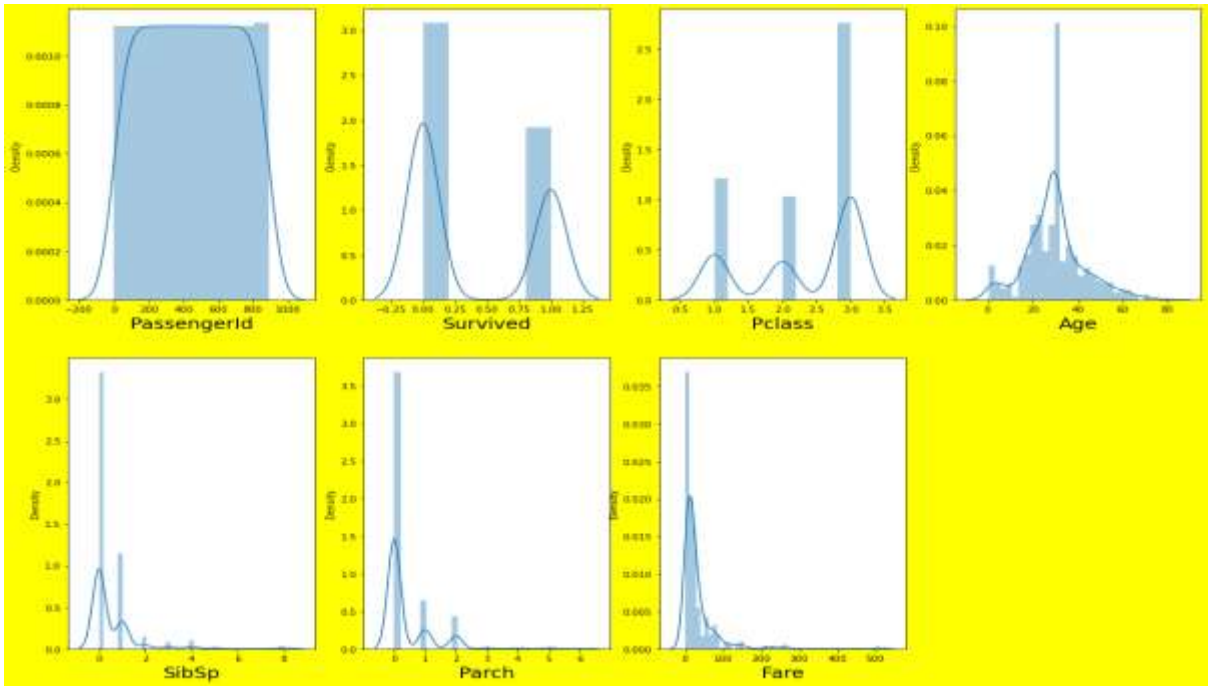
- Visualization of continuous data

```
df=data.drop(columns=['Name','Sex','Ticket','Cabin','Embarked'])

plt.figure(figsize=(20,15),facecolor = 'yellow')
plotnumber =1

for column in df:
    if plotnumber <=7:
        ax = plt.subplot(2,4,plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize=20)

    plotnumber+=1
plt.show()
```



PassengerId is unique for every for every person, that's why it is uniform.

Survived passengers are 342 and not survived passengers are 549, which means that not survived passengers are more than the survived.

In Pclass mostly passengers are belong to the 3rd class.

The passengers are present in the ship of age group 0 to 80 and the maximum passengers are of age group 32 to 34.

In SibSp column mostly 0 and 1 sibling/spouses are there in the ship.

In Parch there are 0, 1 and 2 parents/children in the ship.

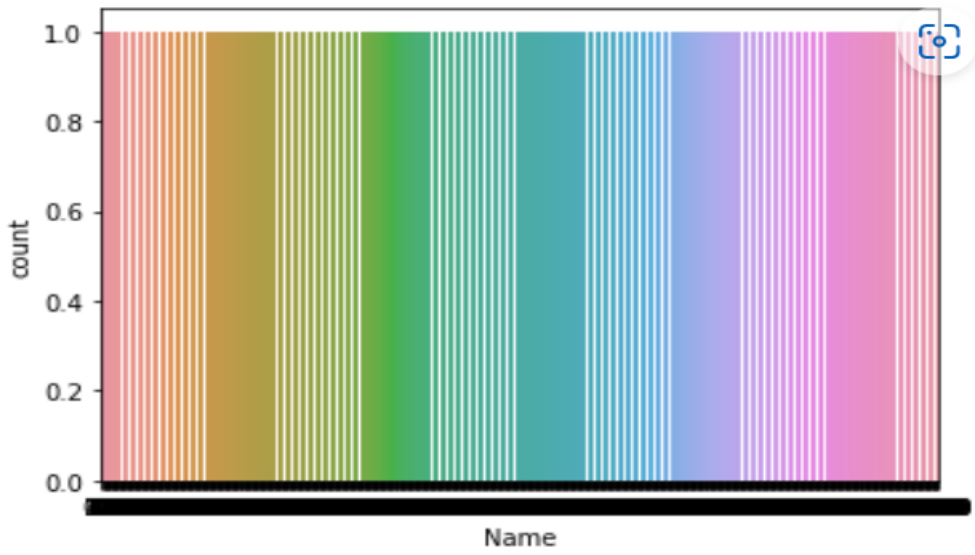
- Visualization of nominal data

```
data_visualization_nominal=data[['Name','Sex','Ticket','Cabin','Embarked']].copy()

data_visualization_nominal.columns
```

```
Index(['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked'], dtype='object')
```

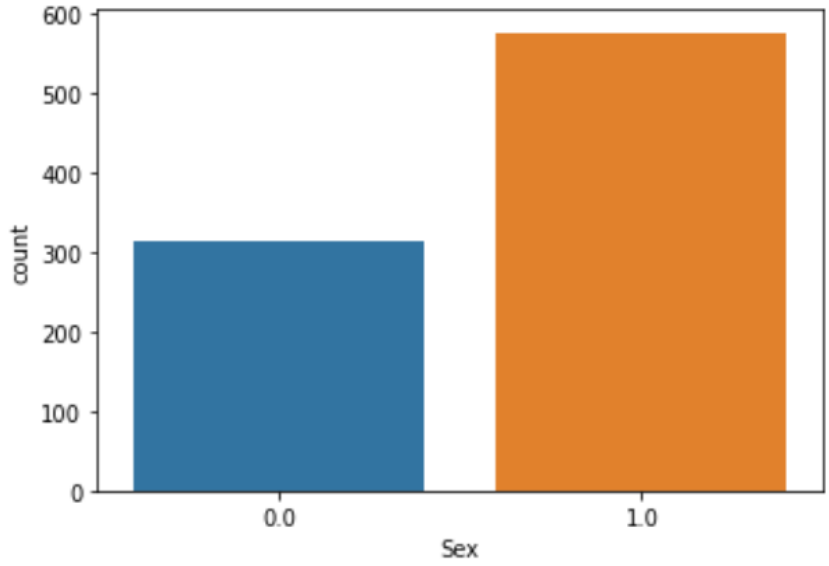
```
ax= sns.countplot(x='Name',data=data_visualization_nominal)
print(data_visualization_nominal['Name'].value_counts())
```



All the values of Name column is 1, because each passengers name are unique.

```
ax= sns.countplot(x='Sex',data=data_visualization_nominal)
print(data_visualization_nominal['Sex'].value_counts())
```

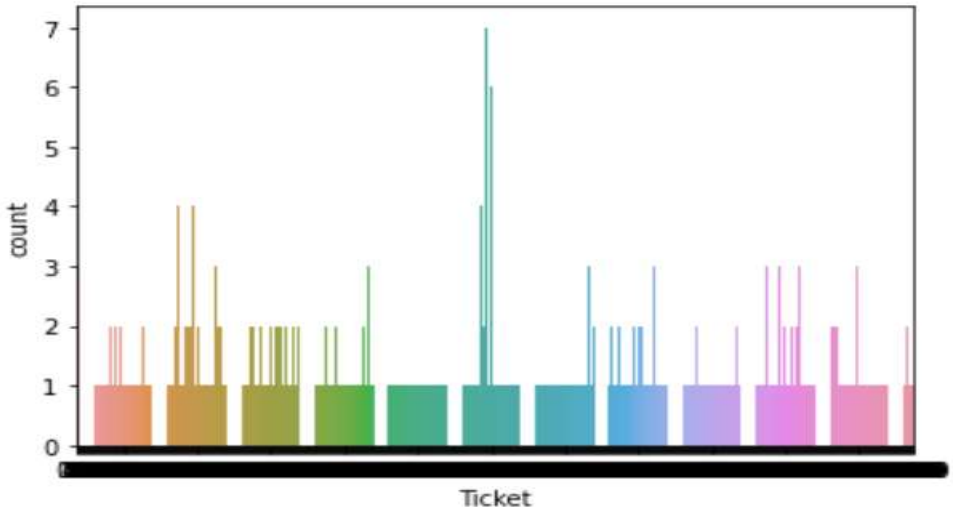
```
1.0    577
0.0    314
Name: Sex, dtype: int64
```



There are 577 males and 314 females, means that maximum male passengers are in the ship.

```
ax= sns.countplot(x='Ticket',data=data_visualization_nominal)
print(data_visualization_nominal['Ticket'].value_counts())
```

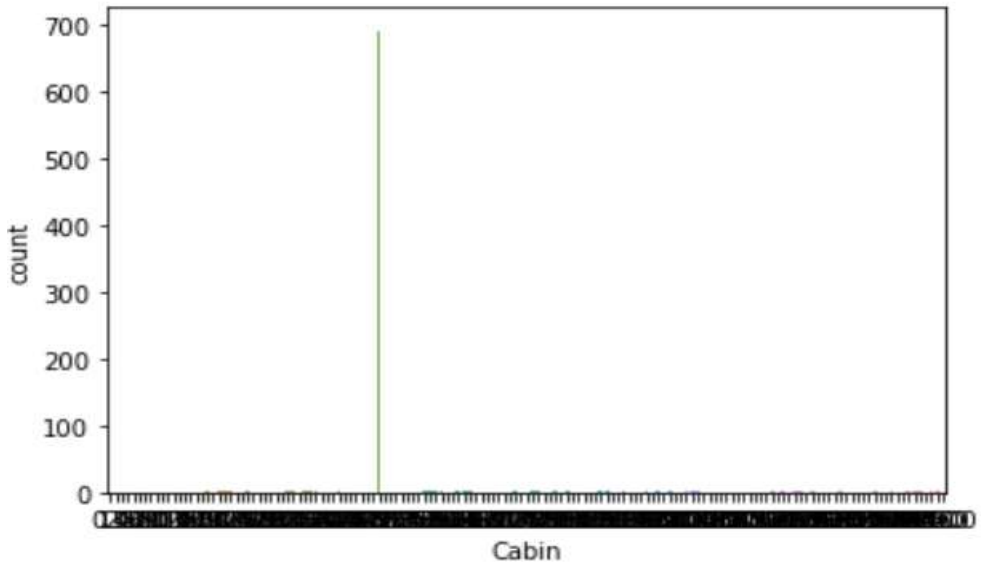
```
333.0    7
568.0    7
80.0     7
249.0    6
566.0    6
        ..
513.0    1
98.0     1
212.0    1
606.0    1
466.0    1
Name: Ticket, Length: 681, dtype: int64
```



Mostly passengers have one and some passengers have more than one ticket.

```
ax= sns.countplot(x='Cabin',data=data_visualization_nominal)
print(data_visualization_nominal['Cabin'].value_counts())
```

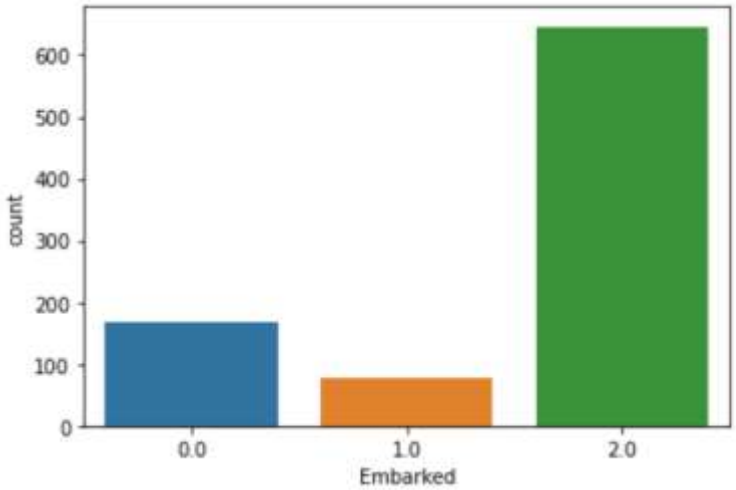
```
47.0     691
145.0      4
63.0       4
62.0       3
142.0      3
         ...
124.0      1
76.0       1
72.0       1
125.0      1
60.0       1
Name: Cabin, Length: 147, dtype: int64
```



Mostly passengers booked Cabin 47 and very rare passengers booked other cabins.

```
ax= sns.countplot(x='Embarked',data=data_visualization_nominal)
print(data_visualization_nominal['Embarked'].value_counts())
```

```
2.0    646
0.0    168
1.0     77
Name: Embarked, dtype: int64
```



Maximum passengers are boarded from Southampton which is approximate 70% of the whole data. 20% passengers boarded from Cherbourg and remaining 10% passengers from Queenstown.

- Describe the data

We get the statistical description such as mean, standard deviation, quantile deviation, range values.

```
data.describe()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	445.000000	0.647587	29.699118	0.523008	0.381594	338.528620	32.204208	53.639731	1.536476
std	257.353842	0.486592	0.836071	257.353842	0.477990	13.002015	1.102743	0.806057	200.860657	49.693429	23.588293	0.791503
min	1.000000	0.000000	1.000000	0.000000	0.000000	0.420000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	222.500000	0.000000	22.000000	0.000000	0.000000	158.500000	7.910400	47.000000	1.000000
50%	446.000000	0.000000	3.000000	445.000000	1.000000	29.699118	0.000000	0.000000	337.000000	14.454200	47.000000	2.000000
75%	668.500000	1.000000	3.000000	667.500000	1.000000	35.000000	1.000000	0.000000	519.500000	31.000000	47.000000	2.000000
max	891.000000	1.000000	3.000000	890.000000	1.000000	80.000000	8.000000	6.000000	680.000000	512.329200	146.000000	2.000000

In the data there are 891 passengers, on an average the survival rate is 38%. Most of the passengers are from 3rd class. The passengers Age range is 0.42 to 80 and the average age is approx. 30%. The SibSp column shows that atleast 50% passengers have no siblings/spouses and in Parch there are atleast 70% passengers have no parents/children.

- Correlation

```
plt.figure(figsize=(16,8))
sns.heatmap(data.corr(),annot=True,linewidth=0.5,linecolor='black')
```

The correlation of feature variable with target variable which is Survived in the dataset. We can see in the heatmap that there is no higher relation between any columns.

In correlation there are positive and negative both the relation. Sex has strong correlation. Pclass, Fare, Cabin, Embarked and Ticket have good correlation. And the remaining columns have week correlation with the target variable.

- Outliers

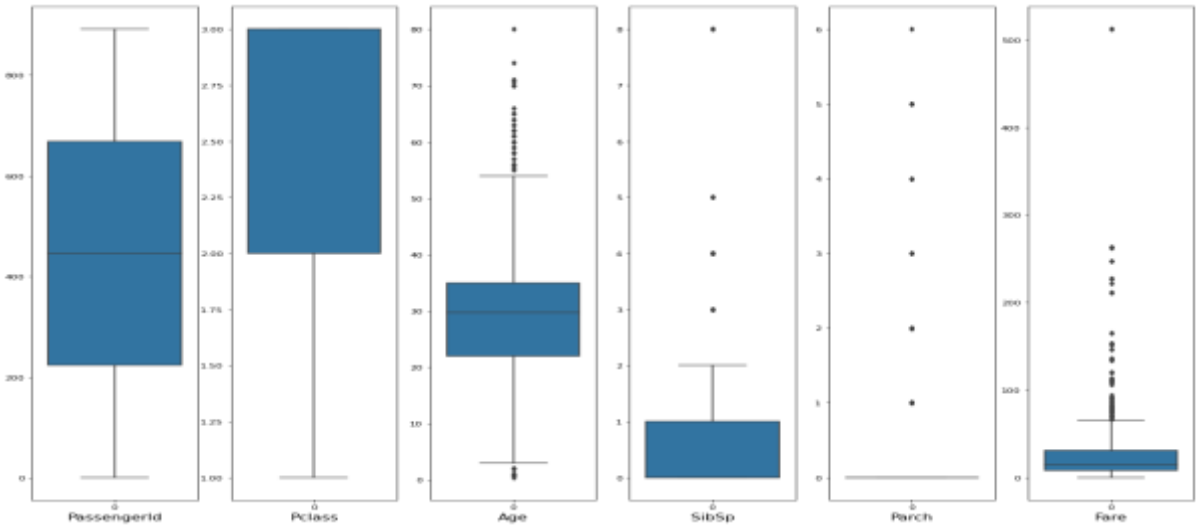
We check outliers only on continuous data, neither on categorical data nor on target variable. So, we make new data with the continuous variable.

```
new_df=data.drop(columns=['Name','Sex','Ticket','Cabin','Embarked','Survived'])
```

```
plt.figure(figsize=(20,15))
number = 1

for column in new_df:
    if number <=6:
        plt.subplot(1,6,number)
        ax = sns.boxplot(data=new_df[column])
        plt.xlabel(column,fontsize=15)

    number+=1
plt.show()
```



In Pclass the mostly passengers are from 3rd class. The most of the passengers are in the ship of age group 3 to 54 and one passenger's age is 80 year old. The maximum Sibling/Spouse is 8 of one passenger but most of them have 0 and 1. The Fare of on passenger is above 500, some of the passengers fare is in between 70 to 280 otherwise the normal range is 7 to 70.

By the boxplot we say that there are four columns having number of outliers in the dataset and if we try to remove all of them by zscore, it will not remove all and not effect that much how much we required for the modeling.

- Skewness

Skewness is also check on continuous data, neither on categorical nor on target variable.

```
data.skew()
```

```
PassengerId    0.000000
Survived       0.478523
Pclass        -0.630548
Name           0.000000
Sex           -0.618921
Age            0.434488
SibSp          3.695352
Parch          2.749117
Ticket         0.000246
Fare           4.787317
Cabin          2.268926
Embarked      -1.264823
dtype: float64
```

We have to consider a threshold range for checking the skewness, so we take (-0.5, 0.5) as range.

In the data there are three columns SibSp, Parch and Fare where skewness is present, which is not good for modeling.

- Variance Inflation Factor

  We have to check the multicolinearity between one feature variable to another feature variable. This can be checked by Variance inflation factor.

  Here we use SatndardScaler() function which we explain below in the Pre-processing pipeline.

```
x=data.drop(['Survived'],axis=1)
y=data['Survived']
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

vif=pd.DataFrame()
vif['VIF']=[variance_inflation_factor(x_scaled,i)for i in  range(x_scaled.shape[1])]
vif['Input']= x.columns
vif
```

	VIF	Input
0	1.011708	PassengerId
1	2.043038	Pclass
2	1.016000	Name
3	1.119118	Sex
4	1.217659	Age
5	1.288122	SibSp
6	1.327062	Parch
7	1.177055	Ticket
8	1.724930	Fare
9	1.086250	Cabin
10	1.085172	Embarked

After applying variance inflation factor we get VIF value of each columns lesser than five, which means that there is multicolinearity exist in the dataset.

- Transform technique to remove skewness

Pclass is left skewed, for this we use square method.

SibSp, Parch and Fare are right skewed, for this we use square root

```
data['Pclass']=pow(data['Pclass'],2)
```

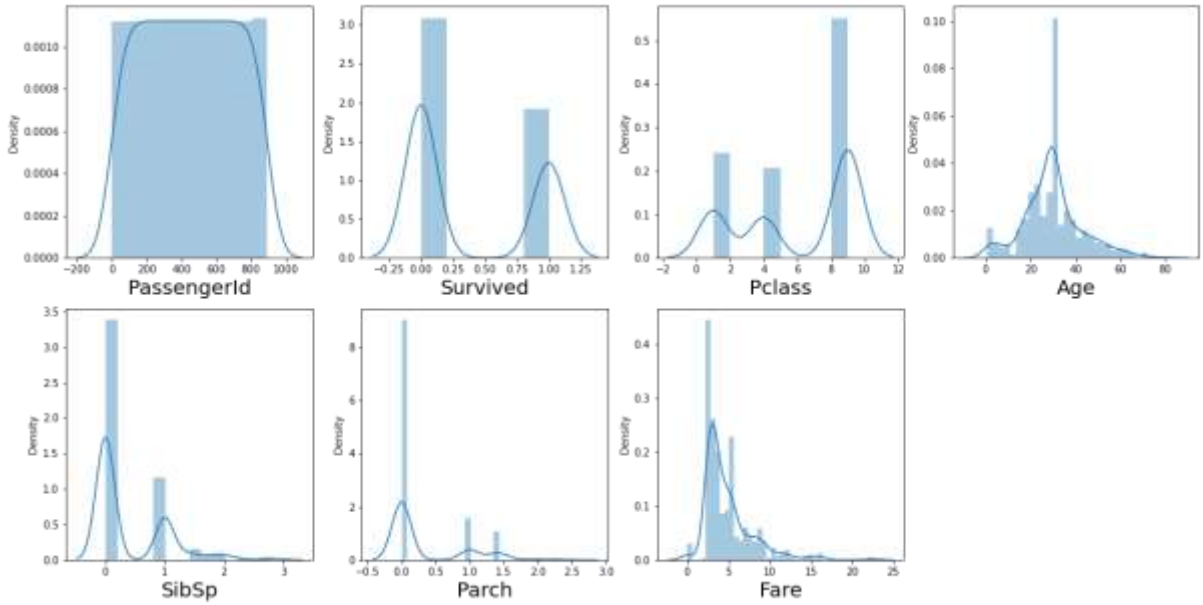
```
data['SibSp']=np.sqrt(data['SibSp'])
data['Parch']=np.sqrt(data['Parch'])
data['Fare']=np.sqrt(data['Fare'])
```

```
data.skew()
```

```
PassengerId    0.000000
Survived       0.478523
Pclass        -0.444064
Name           0.000000
Sex           -0.618921
Age            0.434488
SibSp          1.436526
Parch          1.529799
Ticket         0.000246
Fare           2.085004
Cabin          2.268926
Embarked      -1.264823
dtype: float64
```

The skewness is only removed in the Age column.

We also plot distribution to check the skewness



On our analysis we get that Parch, Fare and SibSp columns are not suitable for modeling because they are not satisfying the condition of normality. So, we drop those columns for increasing our score.

```
data=data.drop(['Parch','Fare','Sibsp'],axis=1)
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	Ticket	Cabin	Embarked
0	1	0	9	108.0	1.0	22.0	523.0	47.0	2.0
1	2	1	1	190.0	0.0	38.0	596.0	81.0	0.0
2	3	1	9	353.0	0.0	26.0	669.0	47.0	2.0
3	4	1	1	272.0	0.0	35.0	49.0	55.0	2.0
4	5	0	9	15.0	1.0	35.0	472.0	47.0	2.0

Now the data is proper for fitting the models and gives better accuracy score. So, by this we can conclude that how much there will be a variation in between the actual data and the predicted data.

4. Pre-processing Pipeline:

Pre-processing is a technique which is done before fitting the model. Here we separate feature variable in ‘x’ and target variable in ‘y’ and fit the

transformation. They scaler transformation are of two type one is StandardScaler() and another one is MinMaxScaler(). We prefer StandarScaler() because there are large number of numerical data.

```
# input data
x=data.drop(['Survived'],axis=1)

# output variable
y=data['Survived']
```

```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

Now the pre-processing is done, and it is ready for modeling the data.

5. Building Machine Learning Models:

The Titanic Survived Prediction is the Supervised Machine Learning Model. In ML there are two type of problem one is regression and another one is classification. This project is a classification problem which is used to calculate or predict the probability of binary (yes/no or 0/1) event. There are number of models for classification but here we use five models from them.

The shape of data set is (891, 12) and after using train_test_slpit() we get

```
x_train,x_test,y_train,y_test = train_test_split(x_scaled,y, test_size=0.20,random_state=68)
```

We take 20% of the random data for the testing purpose and the remaining 80% of the data is used for training. So the shape of trained data is (712, 12) and the shape of test data is (179,12).

Logistic Regression

```
lr=LogisticRegression()
x_train,x_test,y_train,y_test = train_test_split(x_scaled,y, test_size=0.20,random_state=68)
```

```
lr.fit(x_train,y_train)
pred_train=lr.predict(x_train)
y_pred = lr.predict(x_test)

accuracy = accuracy_score(y_test,y_pred)*100
print("accuracy score:",accuracy)
```

```
accuracy score: 81.56424581005587
```

```
cm= confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[94 11]
 [22 52]]
```

```
clr=classification_report(y_test,y_pred)
print(clr)
```

```
              precision    recall  f1-score   support

           0       0.81      0.90      0.85       105
           1       0.83      0.70      0.76        74

    accuracy                           0.82       179
   macro avg       0.82      0.80      0.80       179
weighted avg       0.82      0.82      0.81       179
```

```
lrscore=cross_val_score(lr,x_scaled,y,cv=9)
lrc=lrscore.mean()
print('cross val score:',lrc*100)
```

```
cross val score: 80.13468013468014
```

First we split the train and test data, then we calculate the train score and test score, and take care of not taking test score greater than train data this shows biasedness, so we use smaller value than the test data. The accuracy score we get for Logistic Regression is 81.56 and cv score is 80.13. The total metrics of FN and FP is 33. The f1 score is 85 of not survived and 76 for survived.

K Neighbors Classifier

```
knn= KNeighborsClassifier()
```

```
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)

accuracy = accuracy_score(y_test,y_pred)*100
print("accuracy score:",accuracy)
```

```
accuracy score: 79.3296089385475
```

```
cm= confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[93 12]
 [25 49]]
```

```
clr=classification_report(y_test,y_pred)
print(clr)
```

```
              precision    recall  f1-score   support

           0       0.79      0.89      0.83       105
           1       0.80      0.66      0.73        74

    accuracy                           0.79       179
   macro avg       0.80      0.77      0.78       179
weighted avg       0.79      0.79      0.79       179
```

```
knnscore=cross_val_score(knn,x_scaled,y,cv=9)
knnc=knnscore.mean()
print('cross val score:',knnc*100)
```

```
cross val score: 79.57351290684625
```

Here again we do the same process. Calculate or predict the accuracy score for K Neighbors Classifier which is 79.32 and the cv score is 79.57. The total metrics of FN and FP is 37. The f1 score is 83 for not survived and 73 for survived.

Decision Tree Classifier

```
clf=DecisionTreeClassifier()
```

```
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)

accuracy = accuracy_score(y_test,y_pred)*100
print("accuracy score:",accuracy)
```

```
accuracy score: 74.86033519553072
```

```
cm= confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[87 18]
 [27 47]]
```

```
clr=classification_report(y_test,y_pred)
print(clr)
```

```
              precision    recall  f1-score   support

           0       0.76      0.83      0.79       105
           1       0.72      0.64      0.68        74

    accuracy                           0.75       179
   macro avg       0.74      0.73      0.74       179
weighted avg       0.75      0.75      0.75       179
```

```
clfscore=cross_val_score(clf,x_scaled,y,cv=9)
clfc=clfscore.mean()
print('cross val score:',clfc*100)
```

```
cross val score: 75.64534231200898
```

Here we do the same process and tried to calculate the accuracy score for the train data and test data. The test data score is 74.86 and the cv score is 75.64. The total metrics of TN and FN is 45. The f1 score for survived data is 68 and 79 for not survived.

Random Forest Regression

```
rfc=RandomForestClassifier()
```

```
rfc.fit(x_train,y_train)
y_pred = rfc.predict(x_test)

accuracy = accuracy_score(y_test,y_pred)*100
print("accuracy score:",accuracy)
```

```
accuracy score: 79.88826815642457
```

```
cm= confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[98  7]
 [29 45]]
```

```
clr=classification_report(y_test,y_pred)
print(clr)
```

```
              precision    recall  f1-score   support

           0       0.77      0.93      0.84       105
           1       0.87      0.61      0.71        74

    accuracy                           0.80       179
   macro avg       0.82      0.77      0.78       179
weighted avg       0.81      0.80      0.79       179
```

```
rfcscore=cross_val_score(rfc,x_scaled,y,cv=9)
rfcc=rfcscore.mean()
print('cross val score:',rfcc*100)
```

```
cross val score: 81.59371492704825
```

We apply the process on the Random forest Classifier and calculate the accuracy score which is 79.88 and cv_score is 81.59. The total metrics of TN and FN is 36. The f1 score is 84 of not survived passengers and 71 is if survived passengers.

SVC

```
svc=SVC()
```

```
svc.fit(x_train,y_train)
y_pred = svc.predict(x_test)

accuracy = accuracy_score(y_test,y_pred)*100
print("accuracy score:",accuracy)
```

```
accuracy score: 79.88826815642457
```

```
cm= confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[100   5]
 [ 31  43]]
```

```
clr=classification_report(y_test,y_pred)
print(clr)
```

```
              precision    recall  f1-score   support

           0       0.76      0.95      0.85       105
           1       0.90      0.58      0.70        74

    accuracy                           0.80       179
   macro avg       0.83      0.77      0.78       179
weighted avg       0.82      0.80      0.79       179
```

```
svcscore=cross_val_score(svc,x_scaled,y,cv=9)
svcc=svcscore.mean()
print('cross val score:',svcc*100)
```

```
cross val score: 80.80808080808082
```

In SVC model we get accuracy score 78.99 and cv score 80.80. The sum of TN and FN is 36. The f1 score of survived passengers is 85 and for not survived it is 70.

We applied number of models and almost all of them give better accuracy score. But we have to select one of the best model whose score is maximum

Logistic Regression: 81.56

K Neighbors Classifier: 79.33

Decision Tree Classifier: 74.86

Random Forest Classifier: 79.89

SVC: 79.88

So, we select Logistic Regression model to optimize the better score for the prediction we apply hyper parameter tuning on that model.

Hyper Parameter Tuning on the Logistic Regression

```
lr=LogisticRegression()
```

```
grid_param = {
    'penalty' : ['l1','l2','elasticent','none'],
    'solver'  : ['newton-cg','lbfgs','liblinear','sag'],
    'C'       : [100,10,1.0,0.1,0.01]
}
```

```
grid_search = GridSearchCV(estimator=lr,param_grid=grid_param,cv=9,scoring="accuracy",n_jobs=-1)

grid_search.fit(x_train,y_train)

best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
```

```
clf= LogisticRegression(C= 10, penalty= 'l1', solver= 'liblinear')

clf.fit(x_train,y_train)
```

```
LogisticRegression(C=10, penalty='l1', solver='liblinear')
```

```
pred = clf.predict(x_test)
accuracy_score(y_test,pred)*100
```

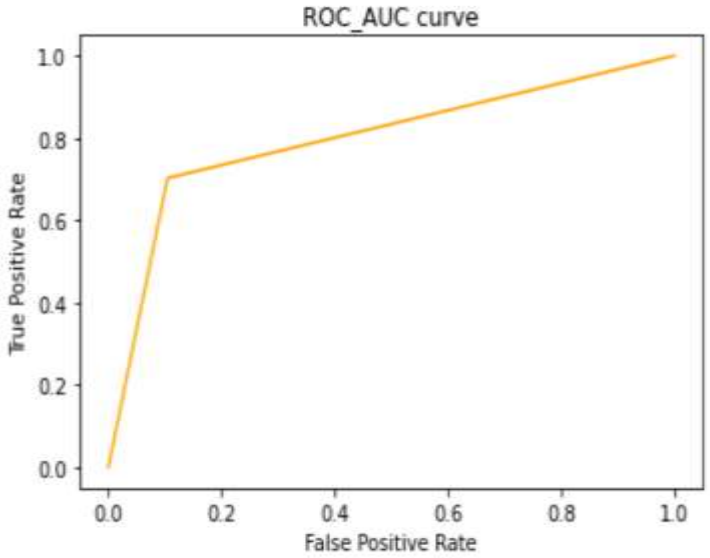
```
81.56424581005587
```

In hyper parameter tuning we use GridSearchCV to optimize the better accuracy for the model. We pass four parameters to increase the score. The accuracy score is 81.56 and after tuning the score is also 81.56. Now this is the accuracy score we use to predict the titanic survival.

We plot the ROC_AUC curve with the optimize accuracy score

```
fpr,tpr,thresholds = roc_curve(y_test,pred)
```

```
plt.plot(fpr,tpr,color='orange')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC_AUC curve')
plt.show()
```



On seeing the graph we can easily say that the roc_auc curve is more than 50 but how much that we have to calculate by roc_auc score.

```
auc_score = roc_auc_score(y_test,pred)*100
print("AUC_score",auc_score)
```

```
AUC_score 79.8970398970399
```

The accuracy score is 81.56 and roc_auc score is 79.89 by this we say that roc_auc score is not good as compare to the accuracy score.

6. Concluding Remarks:

We analyse the whole data and drop all the unnecessary columns which are present in the dataset. On analyzing the data we can conclude those factors on what basis they saves the life of Titanic passengers.

There are large number of males but they rescue 65% female passengers from the ship and 45% passengers are male.

The 70% passengers are from 3rd class and there are 77% passengers whom are from cabin 47.

The passengers who paid high amount they are most preferable than the other passengers and we also see that age is also a factor because they are trying to rescue those whose age is normal, the range of the age is 18 to 42.

By fitting the model we conclude that the 82% of the data is accurate as we given and we predict, but in the 18% of data there may be variation in survival or not.