Predicting the Survival of Titanic Passengers

In this blog, I will explain the process of creating a machine learning model on the Titanic dataset, which is used by many people all over the world. It provides information on the survival of passengers on the Titanic, categorized according to economic status (class), sex, age and survival.

Problem Definition

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.

Importing The Dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix,classification_report,roc_curve,roc_auc_score
import warnings
warnings.filterwarnings('ignore')
```

Getting the Data

```
df=pd.read_csv('titanic.csv')
df.head()
```

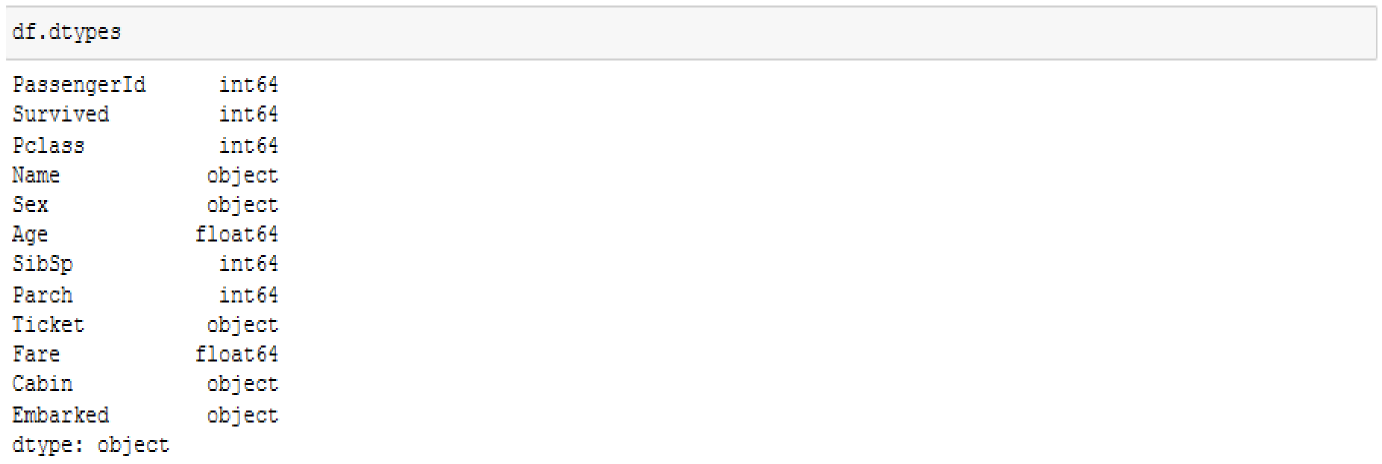
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 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

Exploratory Data Analysis

```
df.shape
```

```
(891, 12)
```

We have 891 entries in our dataset and 11 features and 1 target variable ‘Survived’.



```
df.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
```

2 of the features are floats, 5 are integers and 5 are objects. Below is the description of each of the feature.

PassengerId: Unique Id of a passenger

Survived: Whether passenger survived or not.

Pclass: Ticket class

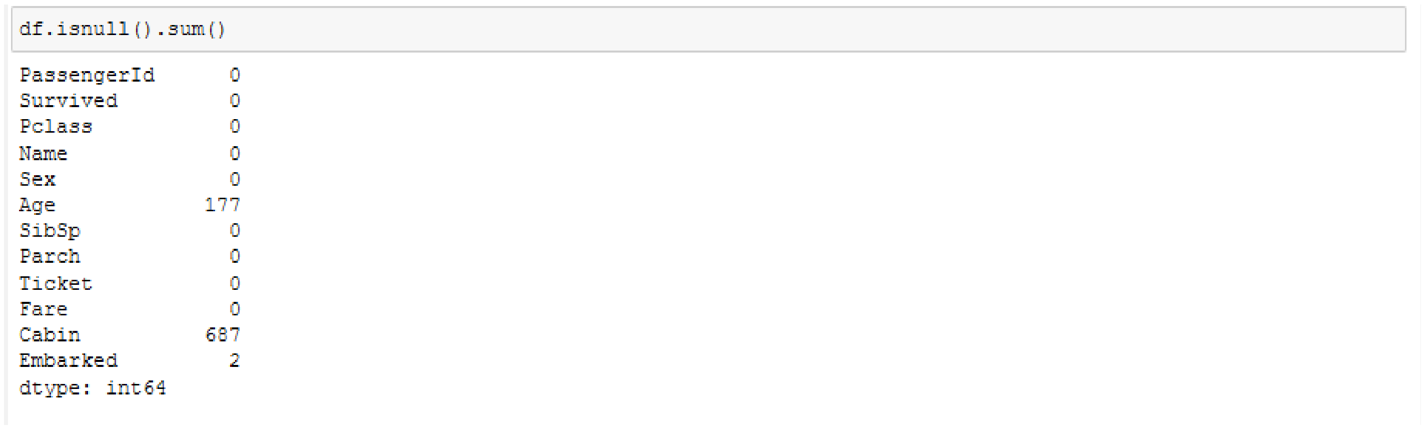
SibSp: Number of siblings and Spouse

Parch: Number of Parents and Children

Embarked: Port of Embarkation

Name, Sex, Age, Ticket, Fare and Cabin are self- explainatory.

Checking For Null Values:

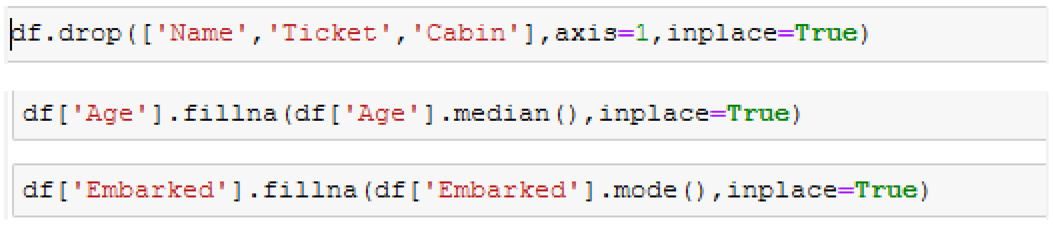


```
df.isnull().sum()
```

```
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
```

Since there is high percentage of missing values in Cabin we will drop the entire Cabin column. We replace the missing values in Age with median value. We replace the missing values in Embarked with the most frequent value.

As name and ticked would be difficult to convert to numbers, so we will drop hem for further calculations. Also, Cabin also has many missing values so we will drop it



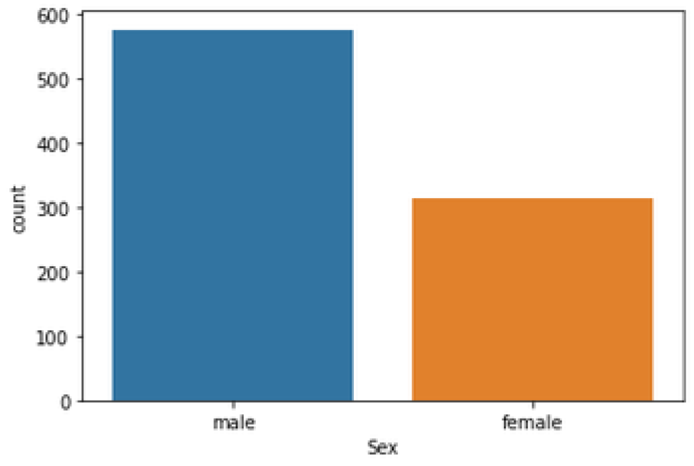
```
df.drop(['Name','Ticket','Cabin'],axis=1,inplace=True)
```

```
df['Age'].fillna(df['Age'].median(),inplace=True)
```

```
df['Embarked'].fillna(df['Embarked'].mode(),inplace=True)
```

```
sns.countplot(df['Sex'])
```

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



Female passengers are less than male passengers

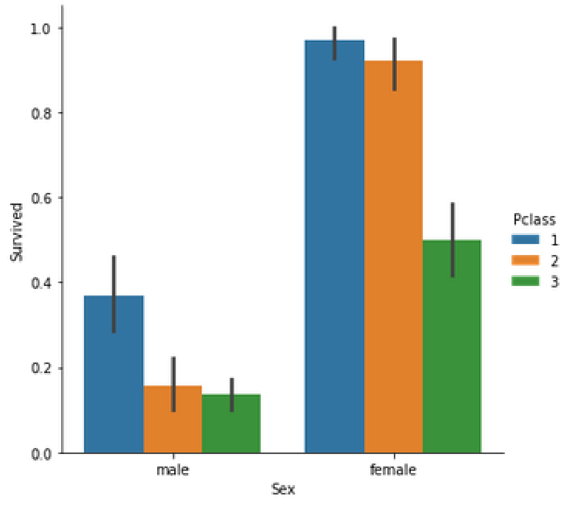
```
df['Survived'].value_counts()
```

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

Our target variable 'Survived' is somewhat balanced, 549 people didn't survuved the sinking while 342 people survived.

```
sns.catplot(x="Sex", y="Survived", hue="Pclass", kind="bar", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x9b6e918280>
```



It is not a surprise, but we can see from the plots that survivors were mostly from the first class and there were more women than men.

Age:

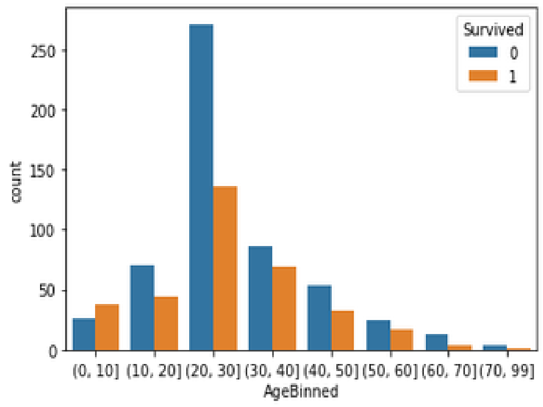
```
bins = [0, 10, 20, 30, 40, 50, 60,70,99]
df['AgeBinned'] = pd.cut(df['Age'], bins)
df[["AgeBinned", "Survived"]].groupby(['AgeBinned']).mean()
```

	Survived
AgeBinned	
(0, 10]	0.593750
(10, 20]	0.382609
(20, 30]	0.334152
(30, 40]	0.445161
(40, 50]	0.383721
(50, 60]	0.404762
(60, 70]	0.235294
(70, 99]	0.200000

We can see most of the survived passengers includes Children, followed by age between 30-40 and 50-60

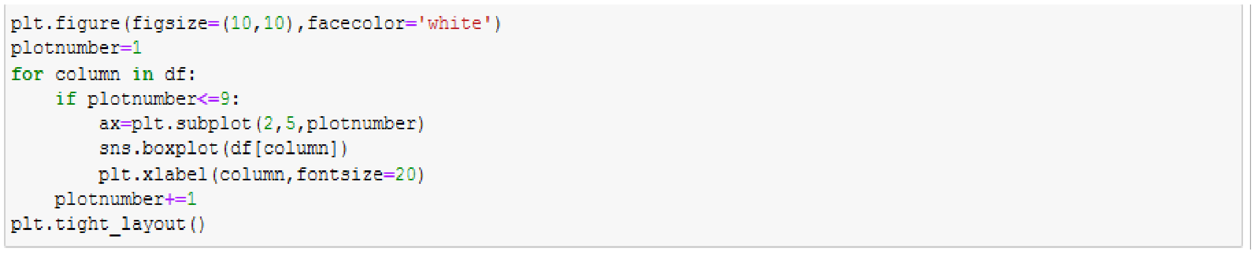
```
sns.countplot(x ='AgeBinned', hue = "Survived", data = df)
```

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

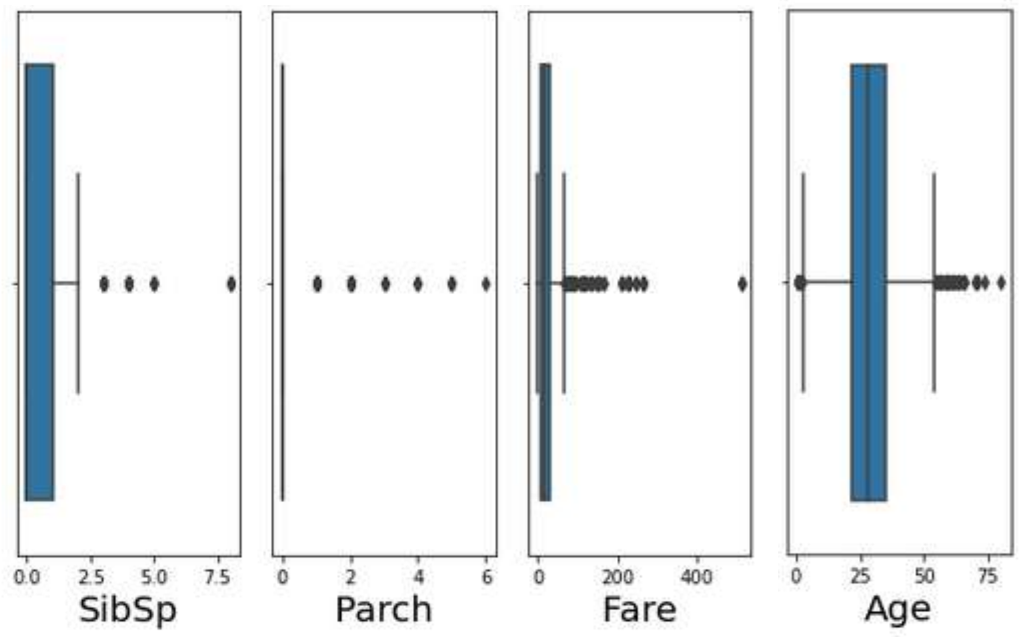


Very less people between the age of 20-30 survived the sinking.

Outlier Detection



```
plt.figure(figsize=(10,10),facecolor='white')
plotnumber=1
for column in df:
    if plotnumber<=9:
        ax=plt.subplot(2,5,plotnumber)
        sns.boxplot(df[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



We can see outliers are present in SibSp, Parch, Fare, Age columns, we will detect the outliers and remove them using z-score method.

Z-Score Method

```
#Removing the outliers:

from scipy.stats import zscore
z=np.abs(zscore(df))
```

```
df=df[(z<3).all(axis=1)]
```

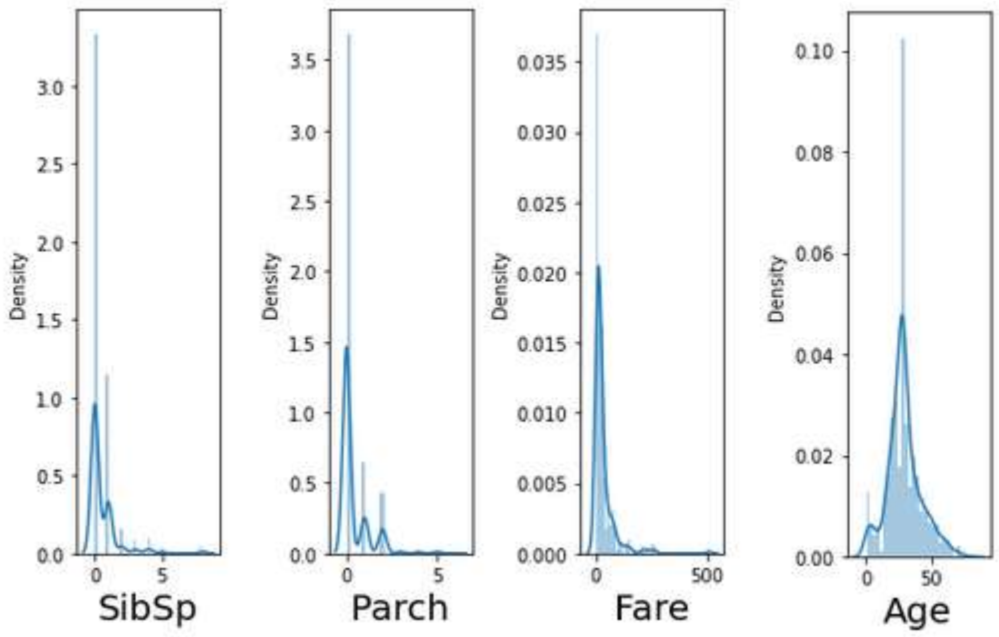
```
df.shape
```

```
(820, 9)
```

After removing the outliers, We have 820 rows and 9 columns in our dataset.

Distribution Plot:

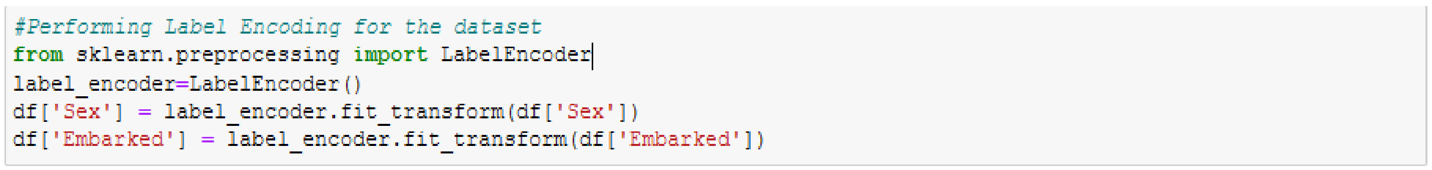
```
#Bivariate Analysis
plt.figure(figsize=(10,10),facecolor='white')
plotnumber=1
for column in df:
    if plotnumber<=9:
        ax=plt.subplot(2,5,plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



We can see data is skewed in SibSp, Parch, Fare and Age. We will remove the skewness using Square root and cube root transformation.

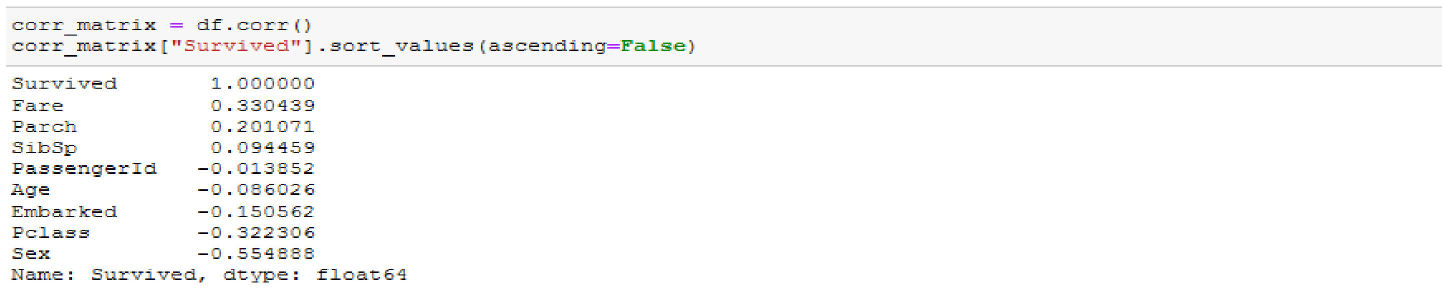
Label Encoding For the Categorical Variables:

We will perform label encoding for Sex and Embarked column using label encoder. We will import LabelEncoder from sklearn.preprocessing



```python
#Performing Label Encoding for the dataset
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df['Embarked'] = label_encoder.fit_transform(df['Embarked'])
```

Checking the correlation with the target variable:



```python
corr_matrix = df.corr()
corr_matrix["Survived"].sort_values(ascending=False)
```

```
Survived       1.000000
Fare           0.330439
Parch          0.201071
SibSp          0.094459
PassengerId   -0.013852
Age           -0.086026
Embarked      -0.150562
Pclass        -0.322306
Sex           -0.554888
Name: Survived, dtype: float64
```

Sex,Pclass and Fare have highest correlation with the survival

```
#Separating x and y for model evaluation
x=df[['PassengerId','Pclass','Sex','Age','SibSp','Parch','Fare','Embarked']]
y=df[['Survived']]
```

We will check the skewness in x.

```
x.skew()
```

```
PassengerId    0.003454
Pclass        -0.632242
Sex           -0.664152
Age            0.401952
SibSp          1.979577
Parch          2.122629
Fare           2.318761
Embarked      -1.257511
dtype: float64
```

We will have to treat skewness in SibSp, Parch and Fare.

```
x['SibSp']=np.cbrt(x['SibSp'])
```

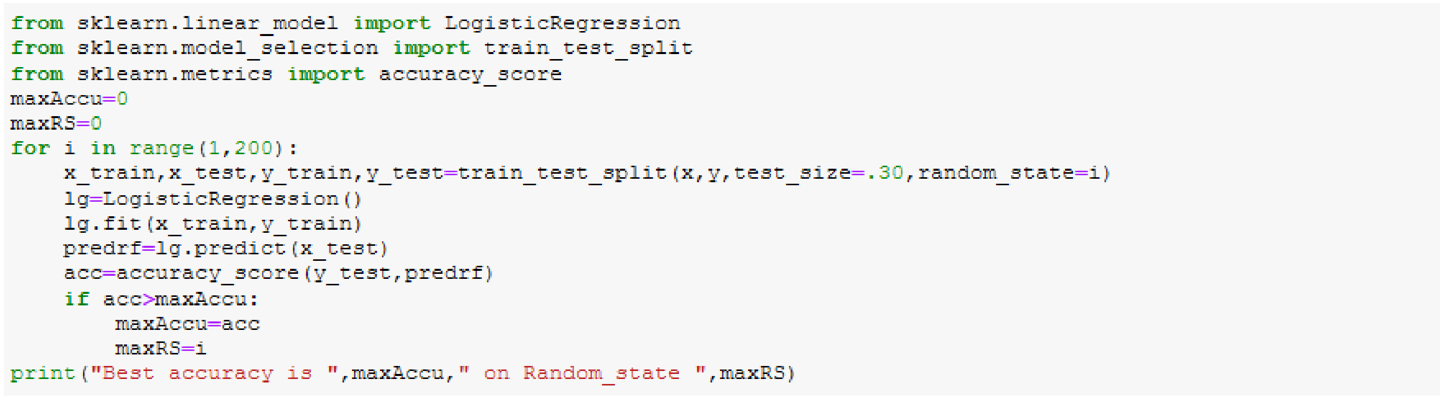
```
x['Parch']=np.cbrt(x['Parch'])
x['Fare']=np.cbrt(x['Fare'])
```

Building Machine Learning Models:

Now we will train several Machine Learning models and compare their results. Note that because the dataset does not provide labels for their testing-set, we need to use the predictions on the training set to compare the algorithms with each other. Later on, we will use cross validation.

Finding the Best Random State:

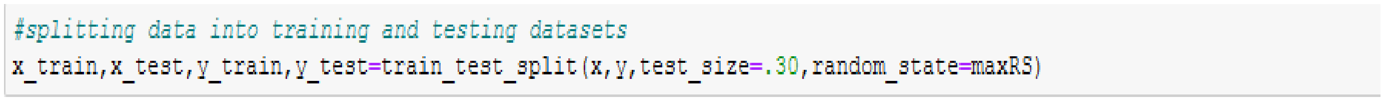
We will find the best random state for Logistic Regression then use these random state to test other models as well



```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=i)
    lg=LogisticRegression()
    lg.fit(x_train,y_train)
    predrf=lg.predict(x_test)
    acc=accuracy_score(y_test,predrf)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

Best accuracy is 0.8414634146341463 on Random_state 198

Splitting the data into train and test data set using random state 198

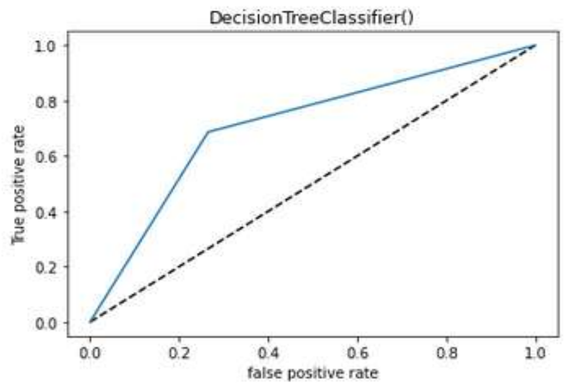


```python
#splitting data into training and testing datasets
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=maxRS)
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix,classification_report,roc_curve,roc_auc_score
dtc=DecisionTreeClassifier()
dtc.fit(x_train,y_train)
dtc.score(x_train,y_train)
preddtc=dtc.predict(x_test)
print(accuracy_score(y_test,preddtc))
print(confusion_matrix(y_test,preddtc))
print(classification_report(y_test,preddtc))
y_pred_prob=dtc.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label=dtc)
plt.xlabel('false positive rate')
plt.ylabel('True positive rate')
plt.title(dtc)
plt.show()
auc_score=roc_auc_score(y_test,dtc.predict(x_test))
print("auc_score:" ,auc_score)
print("\n")
```

```
0.7164179104477612
[[122  44]
 [ 32  70]]
              precision    recall  f1-score   support

           0       0.79      0.73      0.76       166
           1       0.61      0.69      0.65       102

    accuracy                           0.72       268
   macro avg       0.70      0.71      0.71       268
weighted avg       0.72      0.72      0.72       268
```

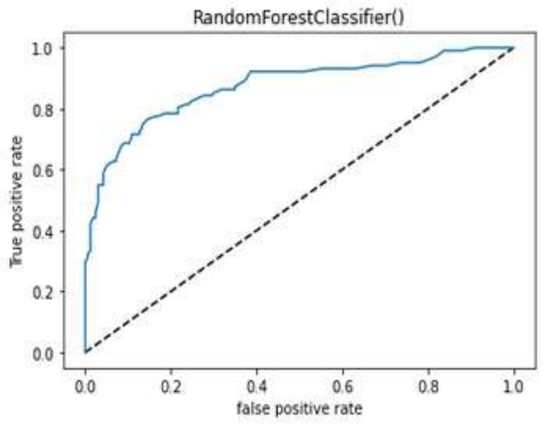


```
auc_score: 0.7106071344200331
```

Random Forest:

```
from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier()
#100 --> default
rf.fit(x_train,y_train)
predrf=rf.predict(x_test)
print(accuracy_score(y_test,predrf))
print(confusion_matrix(y_test,predrf))
print(classification_report(y_test,predrf))
y_pred_prob=rf.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label=rf)
plt.xlabel('false positive rate')
plt.ylabel('True positive rate')
plt.title(rf)
plt.show()
auc_score=roc_auc_score(y_test,rf.predict(x_test))
print("auc_score:" ,auc_score)
print("\n")
```

```
0.8171641791044776
[[145  21]
 [ 28  74]]
              precision    recall  f1-score   support

           0       0.84      0.87      0.86       166
           1       0.78      0.73      0.75       102

    accuracy                           0.82       268
   macro avg       0.81      0.80      0.80       268
weighted avg       0.82      0.82      0.82       268
```

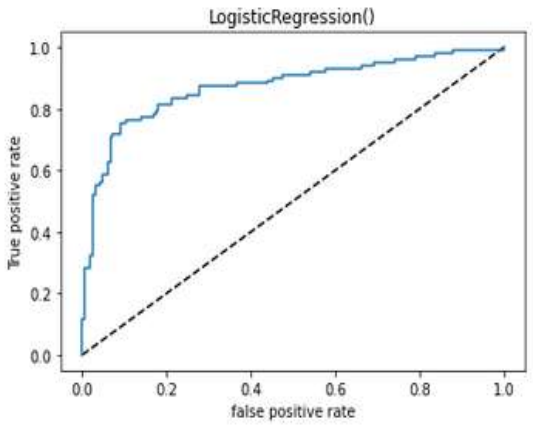


```
auc_score: 0.799492085991023
```

Logistic Regression

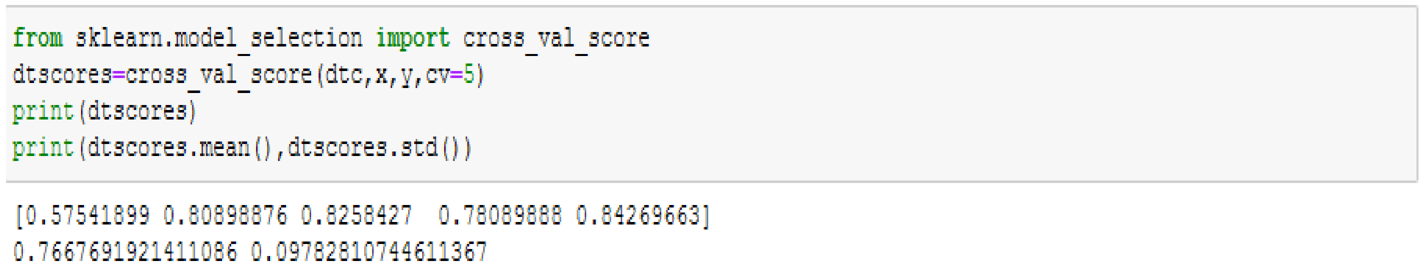
```
lg=LogisticRegression()
lg.fit(x_train,y_train)
predlg=lg.predict(x_test)
print(accuracy_score(y_test,predlg))
print(confusion_matrix(y_test,predlg))
print(classification_report(y_test,predlg))
y_pred_prob=lg.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label=lg)
plt.xlabel('false positive rate')
plt.ylabel('True positive rate')
plt.title(lg)
plt.show()
auc_score=roc_auc_score(y_test,lg.predict(x_test))
print("auc_score:" ,auc_score)
print("\n")
```

```
0.8432835820895522
[[148  18]
 [ 24  78]]
              precision    recall  f1-score   support

           0       0.86      0.89      0.88       166
           1       0.81      0.76      0.79       102

    accuracy                           0.84       268
   macro avg       0.84      0.83      0.83       268
weighted avg       0.84      0.84      0.84       268
```



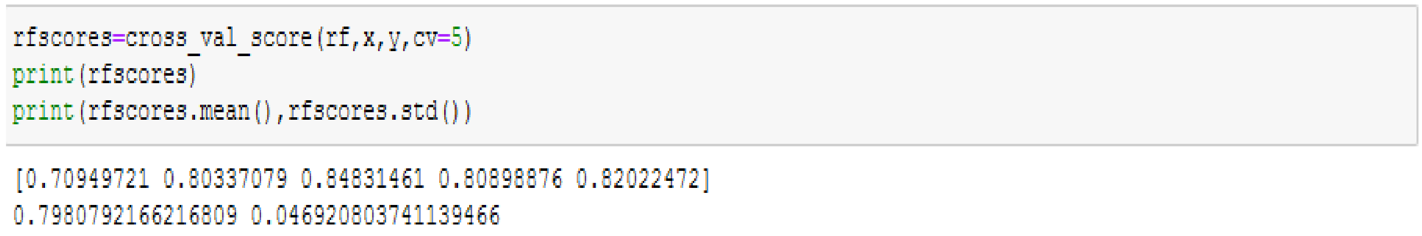
```
auc_score: 0.828136073706591
```

Performing Cross-Validation:



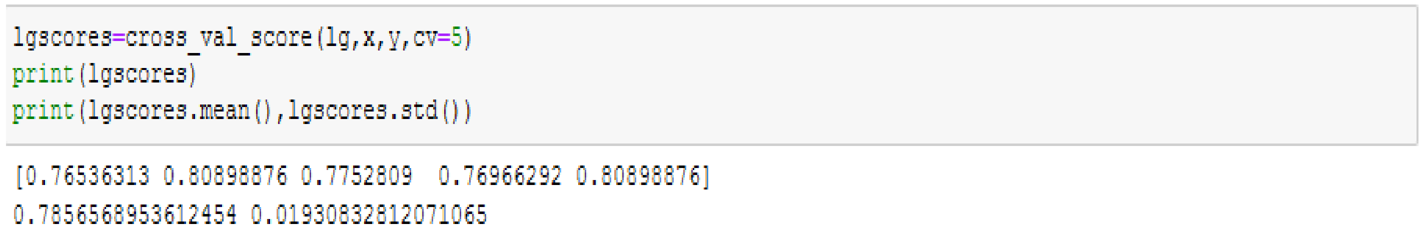
```
from sklearn.model_selection import cross_val_score
dtscores=cross_val_score(dtc,x,y,cv=5)
print(dtscores)
print(dtscores.mean(),dtscores.std())
```

```
[0.57541899 0.80898876 0.8258427  0.78089888 0.84269663]
0.7667691921411086 0.09782810744611367
```



```
rfscores=cross_val_score(rf,x,y,cv=5)
print(rfscores)
print(rfscores.mean(),rfscores.std())
```

```
[0.70949721 0.80337079 0.84831461 0.80898876 0.82022472]
0.7980792166216809 0.046920803741139466
```



```
lgscores=cross_val_score(lg,x,y,cv=5)
print(lgscores)
print(lgscores.mean(),lgscores.std())
```

```
[0.76536313 0.80898876 0.7752809  0.76966292 0.80898876]
0.7856568953612454 0.01930832812071065
```

Which is the best Model ?

As we can see, the Random Forest classifier goes on the first place. But first, let us check, how random-forest and Logistic Regression performs, when we use hyper-Parameter Tuning.

Hyper-Parameter Tuning

Random Forest Hyper-Parameter Tuning

Below you can see the code of the hyperparamter tuning for the parameters criterion, min_samples_leaf, min_samples_split and n_estimators.

```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
```

```
grid_param={
    'criterion':['gini','entropy'],
    'max_depth': [4,5,6,7,8],
    'n_estimators': [100,500],
    'max_features':['auto','sqrt','log2']

}
```

```
grid_search=RandomizedSearchCV(estimator=rf,param_distributions=grid_param,cv=5)
```

```
grid_search.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(),
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [4, 5, 6, 7, 8],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'n_estimators': [100, 500]})
```

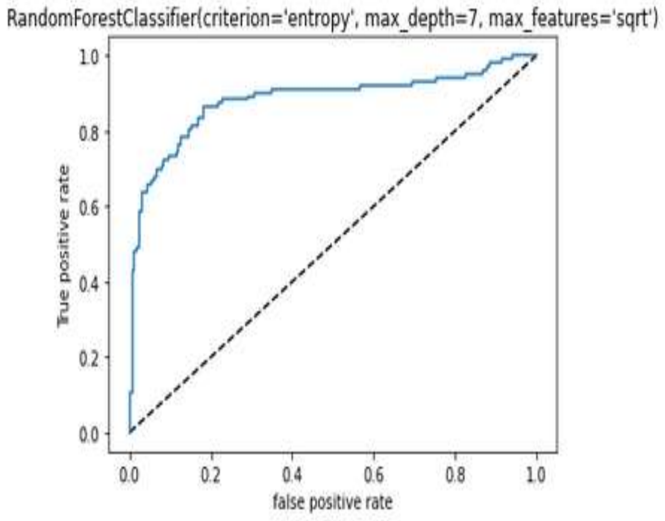
```
best_parameters=grid_search.best_params_
print(best_parameters)
```

```
{'n_estimators': 100, 'max_features': 'sqrt', 'max_depth': 7, 'criterion': 'entropy'}
```

Test New Parameters

```
rfc=RandomForestClassifier(criterion='entropy',max_depth=7,max_features='sqrt',n_estimators=100)
rfc.fit(x_train,y_train)
rfc.score(x_train,y_train)
predrfc=rfc.predict(x_test)
print(accuracy_score(y_test,predrfc))
print(confusion_matrix(y_test,predrfc))
print(classification_report(y_test,predrfc))
y_pred_prob=rfc.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label=rfc)
plt.xlabel('false positive rate')
plt.ylabel('True positive rate')
plt.title(rfc)
plt.show()
auc_score=roc_auc_score(y_test,rfc.predict(x_test))
print("auc_score:" ,auc_score)
print("\n")
```

```
0.835820895522388
[[149  17]
 [ 27  75]]
              precision    recall  f1-score   support

           0       0.85      0.90      0.87       166
           1       0.82      0.74      0.77       102

    accuracy                           0.84       268
   macro avg       0.83      0.82      0.82       268
weighted avg       0.83      0.84      0.83       268
```



```
auc_score: 0.816442239546421
```

Logistic Regression Hyper-Parameter Tuning

```
param_grid = [
    {'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter' : [100, 1000,2500, 5000]
    }
]
```

```
logistic=LogisticRegression()
grid_search=RandomizedSearchCV(logistic,param_grid,cv=5)
```

```
grid_search.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(),
                   param_distributions=[{'C': array([1.00000000e-04, 2.63665090e-04, 6.95192796e-04, 1.8329807
1e-03,
       4.83293024e-03, 1.27427499e-02, 3.35981829e-02, 8.85866790e-02,
       2.33572147e-01, 6.15848211e-01, 1.62377674e+00, 4.28133240e+00,
       1.12883789e+01, 2.97635144e+01, 7.84759970e+01, 2.06913808e+02,
       5.45559478e+02, 1.43844989e+03, 3.79269019e+03, 1.00000000e+04]),
                                          'max_iter': [100, 1000, 2500, 5000],
                                          'penalty': ['l1', 'l2', 'elasticnet',
                                                      'none'],
                                          'solver': ['lbfgs', 'newton-cg',
                                                     'liblinear', 'sag',
                                                     'saga']}])
```

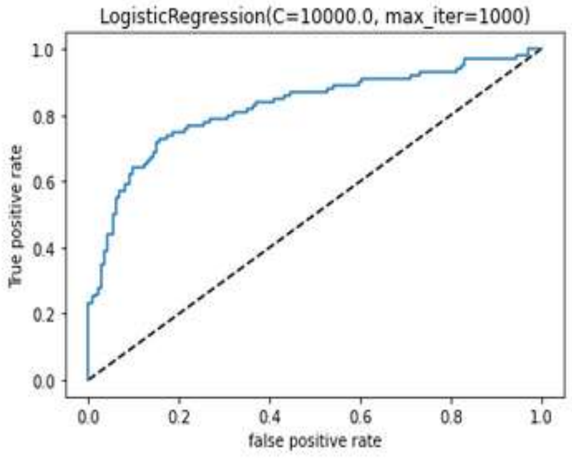
```
best_parameters=grid_search.best_params_
print(best_parameters)
```

```
{'solver': 'lbfgs', 'penalty': 'l2', 'max_iter': 1000, 'C': 10000.0}
```

Test New Parameters

```
lr=LogisticRegression(C=10000.0,penalty='l2',solver='lbfgs',max_iter=1000)
lr.fit(x_train,y_train)
predlg=lr.predict(x_test)
print(accuracy_score(y_test,predlg))
print(confusion_matrix(y_test,predlg))
print(classification_report(y_test,predlg))
y_pred_prob=lr.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label=lr)
plt.xlabel('false positive rate')
plt.ylabel('True positive rate')
plt.title(lr)
plt.show()
auc_score=roc_auc_score(y_test,lr.predict(x_test))
print("auc_score:" ,auc_score)
lrscores=cross_val_score(lr,x,y,cv=5)
print(lrscores)
print(lrscores.mean(),lrscores.std())
print("\n")
```

```
0.7804878048780488
[[125  21]
 [ 33  67]]
              precision    recall  f1-score   support

           0       0.79      0.86      0.82       146
           1       0.76      0.67      0.71       100

    accuracy                           0.78       246
   macro avg       0.78      0.76      0.77       246
weighted avg       0.78      0.78      0.78       246
```



```
auc_score: 0.7630821917808219
[0.78658537 0.7804878  0.76829268 0.79268293 0.79268293]
0.7841463414634147 0.009125993626277914
```

Selecting our Best Model:

As we have min difference between accuracy score and cross val score in Random Forest Classifier. Also, auc score is higher in random forest, So we select ***Random Forest*** as our best model.

What is Random Forest ?

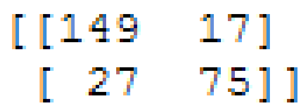
Random Forest is a supervised learning algorithm. Like you can already see from it's name, it creates a forest and makes it somehow random. The forest it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. With a few exceptions a random-forest classifier has all the hyperparameters of a decision-tree classifier and also all the hyperparameters of a bagging classifier, to control the ensemble itself.

The random-forest algorithm brings extra randomness into the model, when it is growing the trees. Instead of searching for the best feature while splitting a node, it searches for the best feature among a random subset of features. This process creates a wide diversity, which generally results in a better model. Therefore when you are growing a tree in random forest, only a random subset of the features is considered for splitting a node. You can even make trees more random, by using random thresholds on top of it, for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Confusion Matrix:



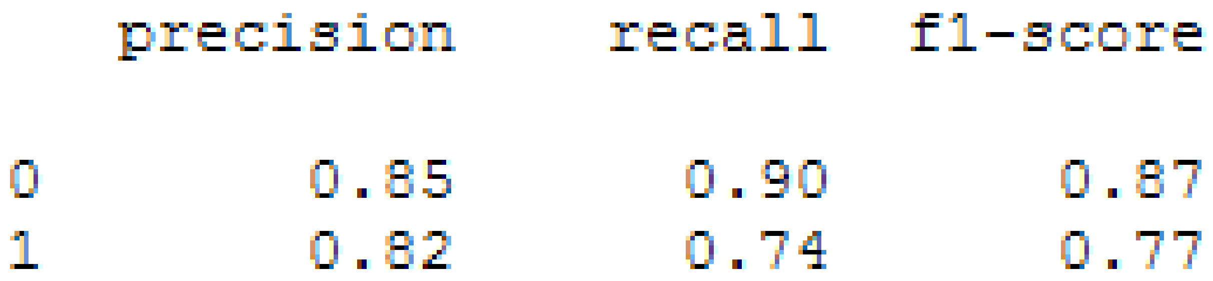
```
[[149  17]
 [ 27  75]]
```

The first row is about the not-survived-passengers: **149 passengers were correctly classified as not survived** (called true negatives) and **17 where wrongly classified as not survived** (false positives).

The second row is about the survived-predictions: **27 passengers where wrongly classified as survived** (false negatives) and **75 where correctly classified as survived** (true positives).

A confusion matrix gives you a lot of information about how well your model does, but there is a way to get even more, like computing the classifiers precision.

Precision Recall and F1-Score

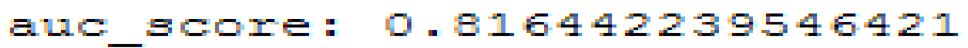


```
     precision    recall  f1-score

0         0.85      0.90      0.87
1         0.82      0.74      0.77
```

Random Forest predicts 85% of the time, a passengers survival correctly (precision). The recall tells us that it predicted the survival of 90 % of the people who actually survived.

We can combine precision and recall into one score, which is called the F-score. The F-score is computed with the harmonic mean of precision and recall. Here, we have a 87 % F-score.

ROC AUC Score



```
auc_score: 0.816442239546421
```

The ROC AUC Score is the corresponding score to the ROC AUC Curve. It is simply computed by measuring the area under the curve, which is called AUC.

Summary

We started with the data exploration where we got a idea for the dataset, inspected about missing fields and learned which features are important using correlation matrix. During this process we used Seaborn and Matplotlib library to do the visualizations. During the data preprocessing, we computed missing values, converted categorical features into numeric ones. After that, we trained 3 different machine learning models, picked two of them (Random Forest & Logistic Regression) and applied hyper parameter tuning on it. Then we discussed how random forest works, took a look at the importance it assigns to the different features and tuned its performance by optimizing its hyper parameter values. Lastly, we looked at its confusion matrix and computed the model’s precision, recall, f-score . We also computed ROC-AUC curve and AUC Score.