Titanic Survivors:

Fifteen-hundred people went into the sea, when Titanic sank from under us. There were twenty boats floating nearby… and only one came back. One. Six were saved from the water, myself included. Six… out of fifteen-hundred. Afterward, the seven-hundred people in the boats had nothing to do but wait… wait to die… wait to live… wait for an absolution… that would never come- Rose.

I start to derive insights from the titanic dataset by quoting the above dialogue from the famous film The Titanic.

**Dataset reference:**

The titanic dataset was extracted from Kaggle Dataset repository.

<https://www.kaggle.com/c/titanic/data>

**Data Dictionary:**

|  |  |  |
| --- | --- | --- |
| Variable | Definition | Key |
| survival | Survival | 0 = No, 1 = Yes |
| pclass | Ticket class | 1 = 1st, 2 = 2nd, 3 = 3rd |
| sex | Sex |  |
| Age | Age in years |  |
| sibsp | # of siblings / spouses aboard the Titanic |  |
| parch | # of parents / children aboard the Titanic |  |
| ticket | Ticket number |  |
| fare | Passenger fare |  |
| cabin | Cabin number |  |
| embarked | Port of Embarkation | C = Cherbourg,  Q = Queenstown, S =  Southampton |

Following are the steps involved to

**Step-1:**

Reading the data:

First step is to read the data using Python Pandas Frames from csv file.

import matplotlib.pyplot as plt

import numpy as np

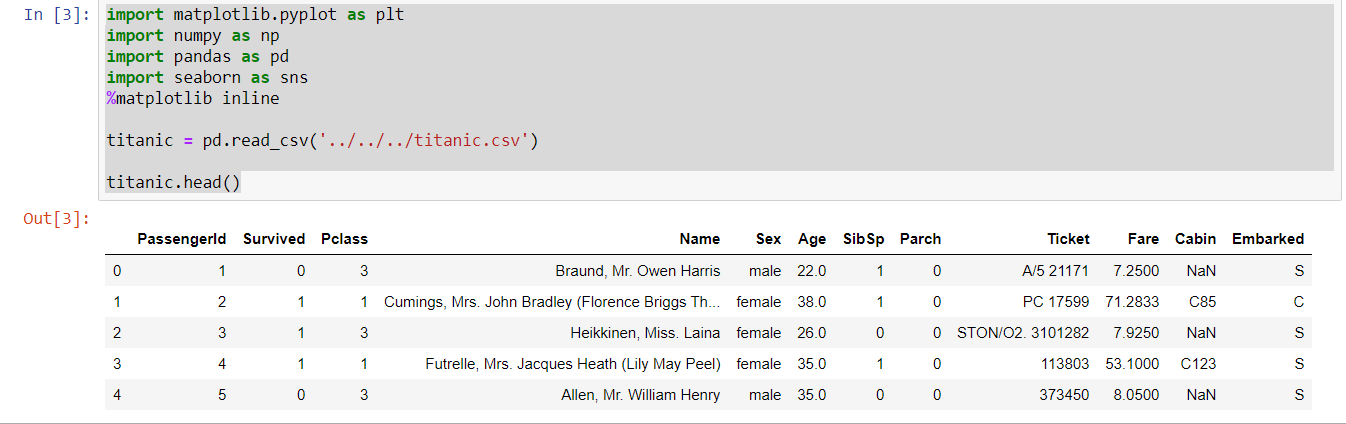
import pandas as pd

import seaborn as sns

%matplotlib inline

titanic = pd.read\_csv('../../../titanic.csv')

titanic.head()



*Inference:*

We got a dataset imported in titanic dataframe. We are displaying its head to get an overview of the imported data.

**Step-2:**

**Dropping Categorical Features:**

Drop all categorical Features 'PassengerId', 'Name', 'Ticket', 'Sex', 'Cabin', 'Embarked' to learn more about the actual column that had a great influence on the decisions.

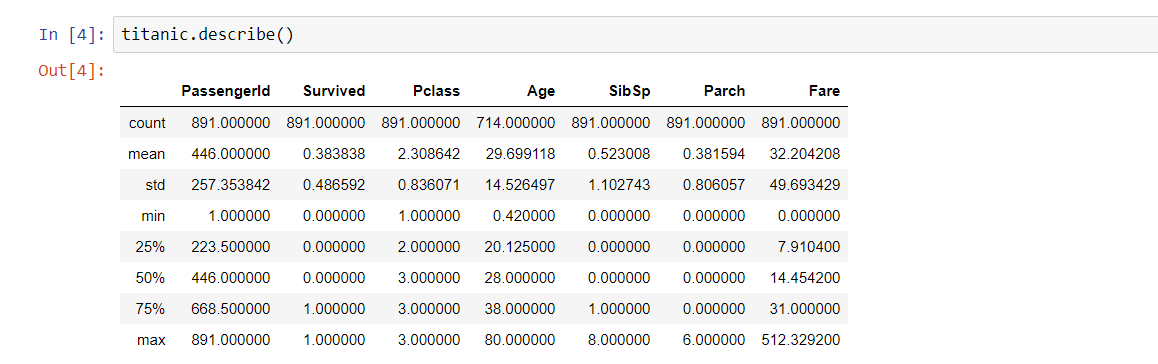
titanic.drop(['PassengerId', 'Name', 'Ticket', 'Sex', 'Cabin', 'Embarked'],axis=1)

This command drops the categorical features in the dataset.

**Step-3:**

**Exploring Continuous Features:**

Titanic.describe()- command explores the continuous features in the dataset.

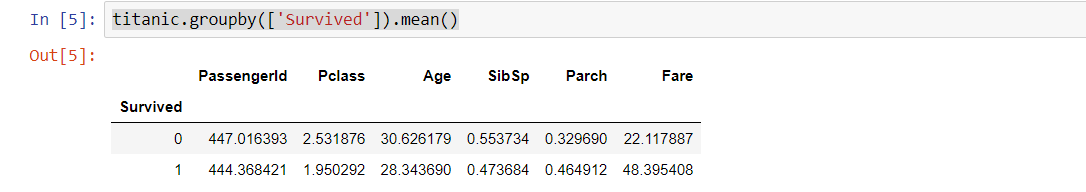


The total count, mean (average values), standard deviation values , maximum and minimum values are displayed by the describe command for the continuous features in the dataset.

**Step-4:**

**Survived Features:**

titanic.groupby(['Survived']).mean()

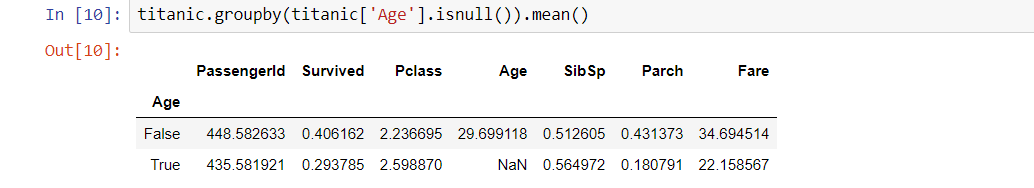


*Insights:*

Those who survived comes under the cadet of 1 and those who didn’t survive comes under the 0. So we can come to a small insight that small aged people and who have less siblings and spouses aboard , high number of parents and those who paid huge had high probability of survival whereas others couldn’t.

**Step-5:**

Taking the titanic dataframe mean values and grouping by age null attribute will provide the following inference.



**Step-6:**

**Plotting the continuous features:**

With the help of the following code we can print the histogram plots for continuous features using for loop.

for i in ['Age', 'Fare']:

died = list(titanic[titanic['Survived'] == 0][i].dropna())

survived = list(titanic[titanic['Survived'] == 1][i].dropna())

xmin = min(min(died), min(survived))

xmax = max(max(died), max(survived))

width = (xmax - xmin) / 40

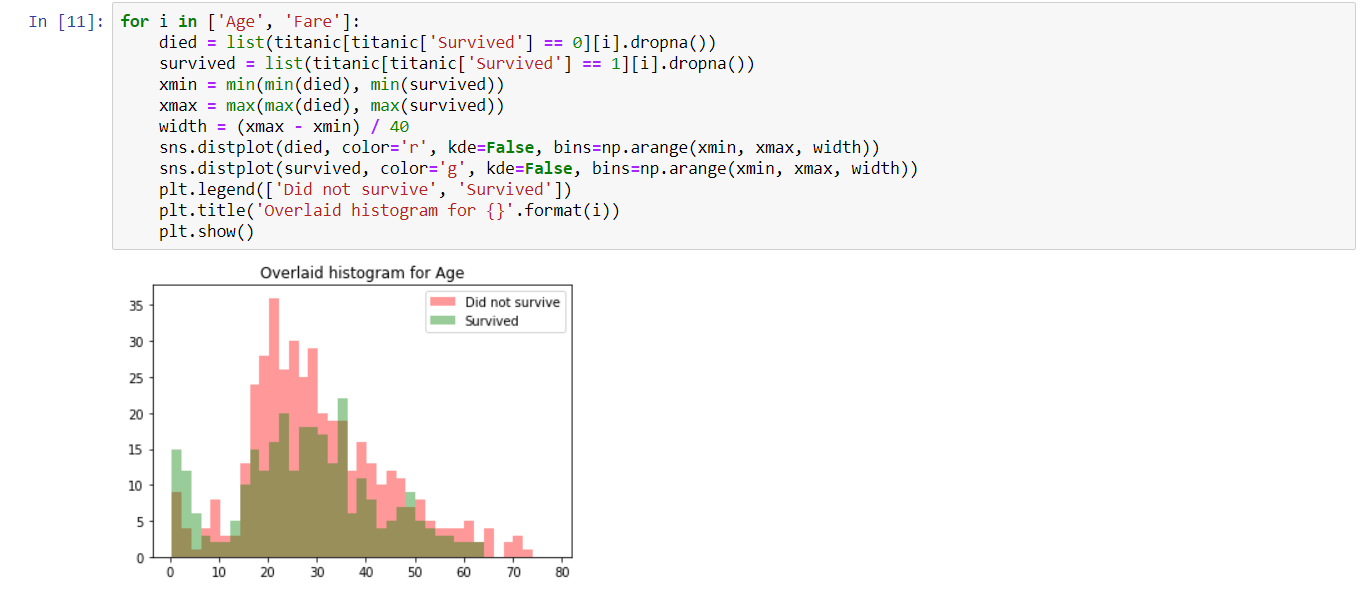
sns.distplot(died, color='r', kde=False, bins=np.arange(xmin, xmax, width))

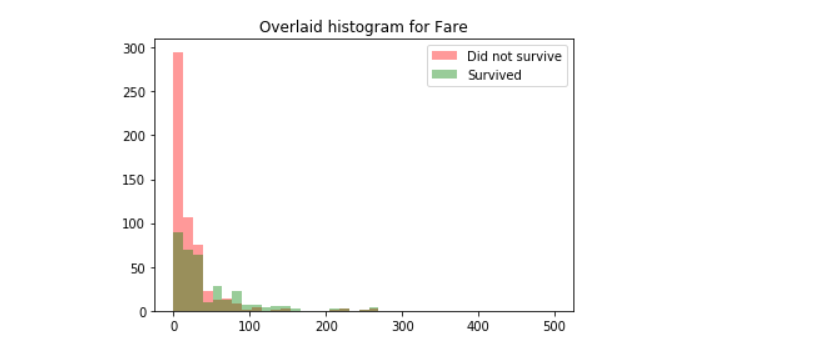
sns.distplot(survived, color='g', kde=False, bins=np.arange(xmin, xmax, width))

plt.legend(['Did not survive', 'Survived'])

plt.title('Overlaid histogram for {}'.format(i))

plt.show()





The plots of histogram between age and survival and fare and survival are displayed. We can conclude that person with less age survived than the old people. First class passengers who paid high fare survived well than the lower class passengers.

**Step-7:**

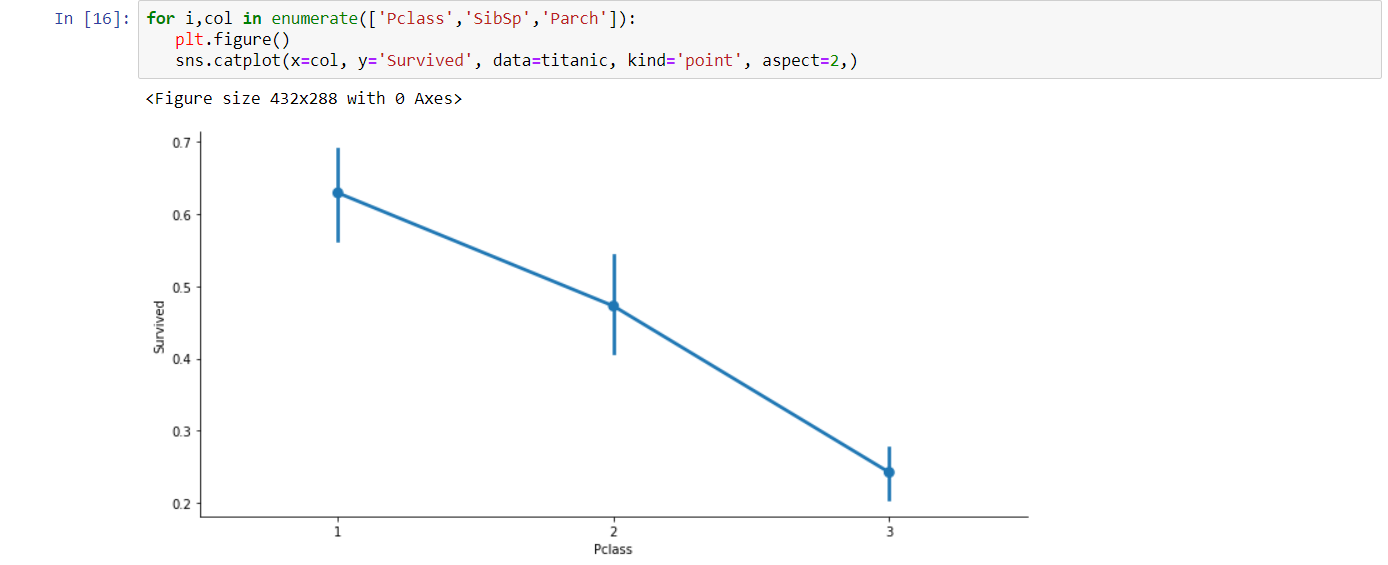
**Discretional Analysis using CaterPlots:**

**catplot** shows frequencies (or optionally fractions or percents) of the categories of one, two or three categorical variables.  For the categorical variables such as Pclass, SIbSp, Parch (details are given above) we have plotted catplot using the below code:

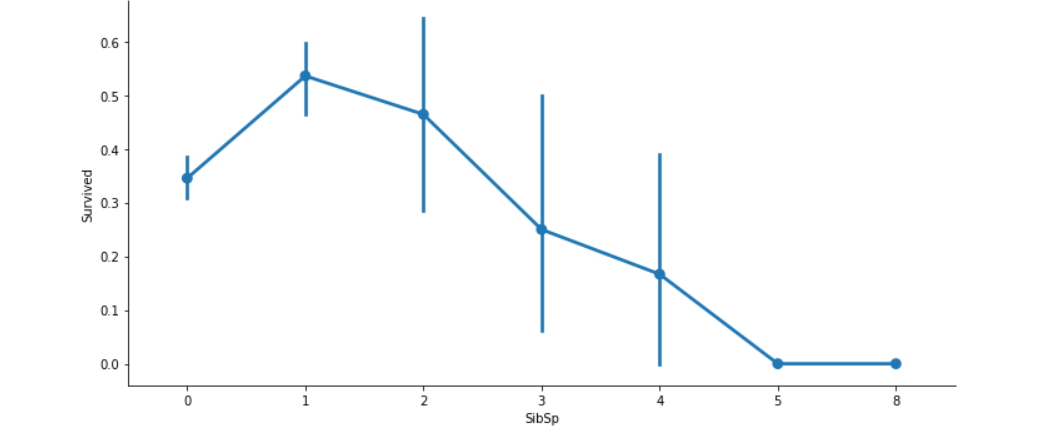
for i,col in enumerate(['Pclass','SibSp','Parch']):

plt.figure()

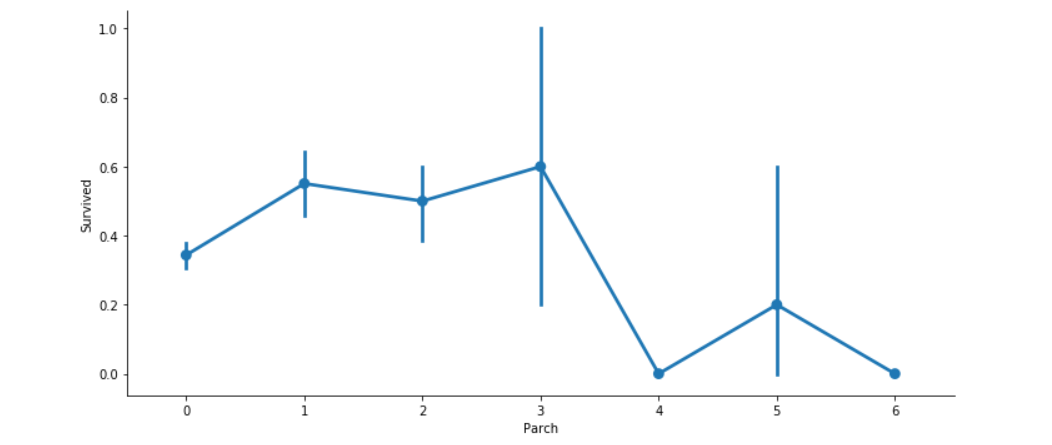
sns.catplot(x=col, y='Survived', data=titanic, kind='point', aspect=2,)



Inference: Those who were in high class(1st class) their survival rate is larger.



Inference: Those who had more number of sibblings and spouses had less survival rate.



The Parent children and survived plot is given above. Its actually the average rate of survival.

In order for convenience we club the attribute counts of SibSp and Parch to family\_cnt with the help of below code:

titanic['family\_cnt']=titanic['SibSp']+titanic['Parch']

Step-8:

Some of data pre-processing step:

Converting categorical attribute to numeric for the convenience of computing with the help of below code:

gender\_num = {'male': 0 , 'female': 1 }

titanic['Sex'] = titanic['Sex'].map(gender\_num)

After cleaning the dataset store the new dataset in a different csv file.

titanic.to\_csv('../../../titanic\_cleaned.csv', index=False)

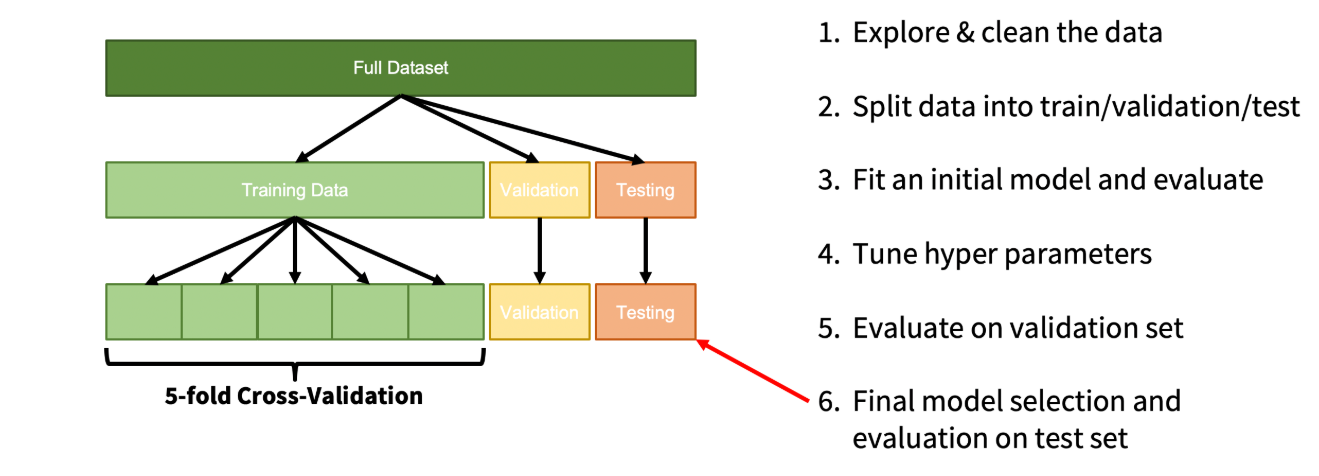
titanic.head()

The following code helps you to write the cleaned date.

**Step-9:**

**Machine Learning:**

From Here onwards Machine Learning comes to the picture. Split the dataset from the dataframe to training ,test and validation dataset. The picture below shows the series of steps that is going to be employed by us



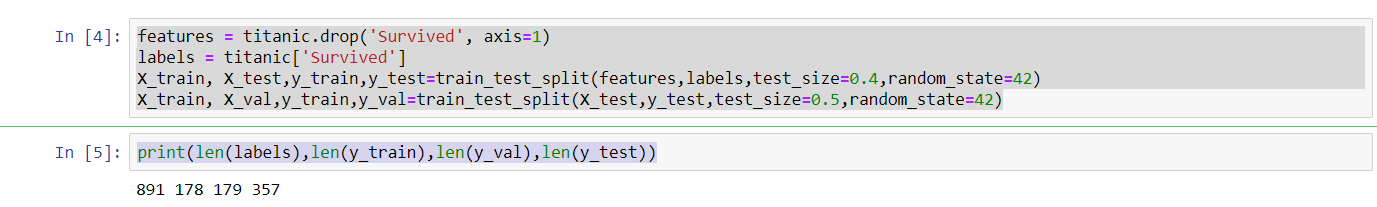
features = titanic.drop('Survived', axis=1)

labels = titanic['Survived']

X\_train, X\_test,y\_train,y\_test=train\_test\_split(features,labels,test\_size=0.4,random\_state=42)

X\_train, X\_val,y\_train,y\_val=train\_test\_split(X\_test,y\_test,test\_size=0.5,random\_state=42)

print(len(labels),len(y\_train),len(y\_val),len(y\_test))

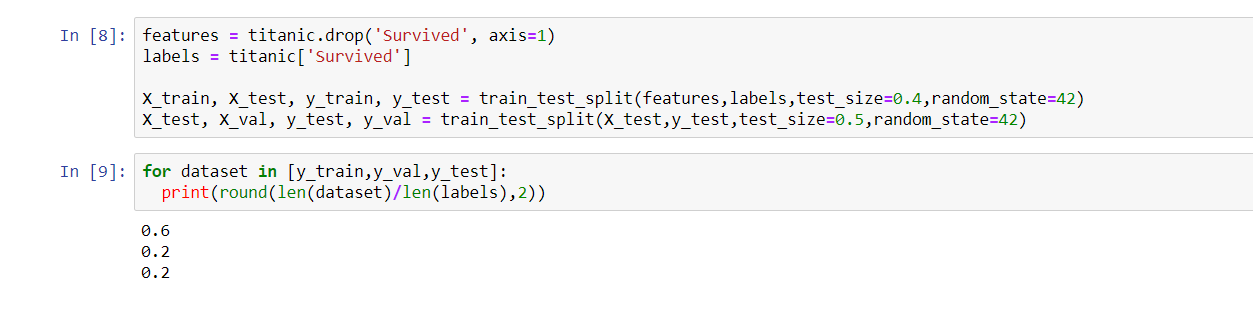


The length of labels, Y\_train, test and validation dataset are displayed for reference.

In order to get a clear clarification of how dataset was separated we can use the following code:

for dataset in [y\_train,y\_val,y\_test]:

print(round(len(dataset)/len(labels),2))



Thus from the output we can see that the dataset for training is allotted with 60% of dataset of the whole. Testing and validation sets are allotted with other 20% each of the data.

With the help of the following codes write the data to each excel file.

X\_train.to\_csv('../../../train\_features.csv', index=False)

X\_val.to\_csv('../../../val\_features.csv', index=False)

X\_test.to\_csv('../../../test\_features.csv', index=False)

y\_train.to\_csv('../../../train\_labels.csv', index=False)

y\_val.to\_csv('../../../val\_labels.csv', index=False)

y\_test.to\_csv('../../../test\_labels.csv', index=False)

**Cross Validation:**

With the help of the following code do the 5-fold cross validation using random classifier algorithm:

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_score

import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

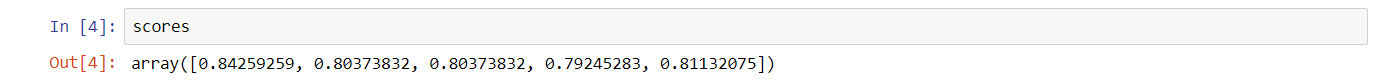
tr\_features = pd.read\_csv('../../../train\_features.csv')

tr\_labels = pd.read\_csv('../../../train\_labels.csv', header=None)

rf=RandomForestClassifier()

scores=cross\_val\_score(rf,tr\_features,tr\_labels.values.ravel(),cv=5)

Print the scores to check the presence of cross validated datasets.



**Tuning the hyper Parameters:**

**GridSearchCV** inherits the methods from the classifier. Estimator with a grid search preamble to tune hyper-parameters. The method picks the optimal parameter from the grid search and uses it with the estimator selected by the user.

def print\_results(results):

print('BEST PARAMS: {}\n'.format(results.best\_params\_))

means = results.cv\_results\_['mean\_test\_score']

stds = results.cv\_results\_['std\_test\_score']

for mean, std, params in zip(means, stds, results.cv\_results\_['params']):

print('{} (+/-{}) for {}'.format(round(mean, 3), round(std \* 2, 3), params))

rf = RandomForestClassifier()

parameters = {

'n\_estimators': [5,50,100],

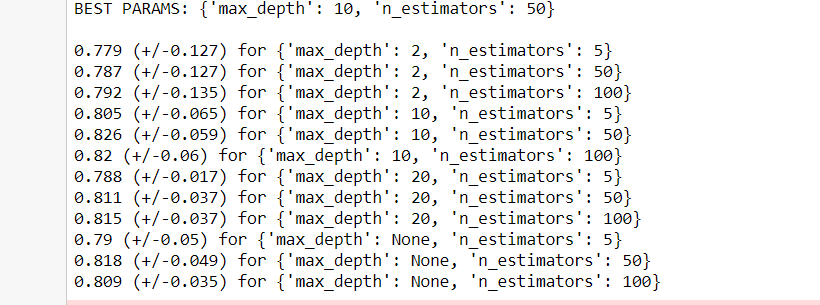
'max\_depth': [2,10,20,None]

}

cv= GridSearchCV(rf,parameters,cv=5)

cv.fit(tr\_features,tr\_labels.values.ravel())

print\_results(cv)



The output that we get is as follows. The Best combinational parameters is 10 and the estimator is 50.

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score

tr\_features = pd.read\_csv('../../../train\_features.csv')

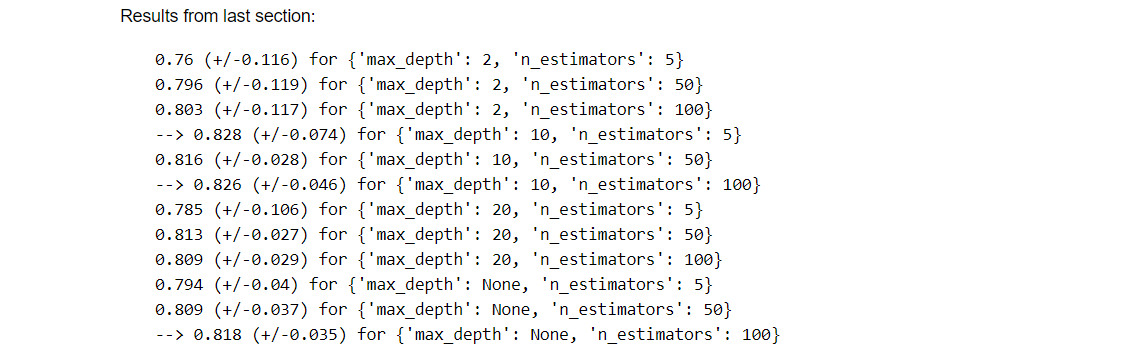
tr\_labels = pd.read\_csv('../../../train\_labels.csv', header=None)

val\_features = pd.read\_csv('../../../val\_features.csv')

val\_labels = pd.read\_csv('../../../val\_labels.csv', header=None)

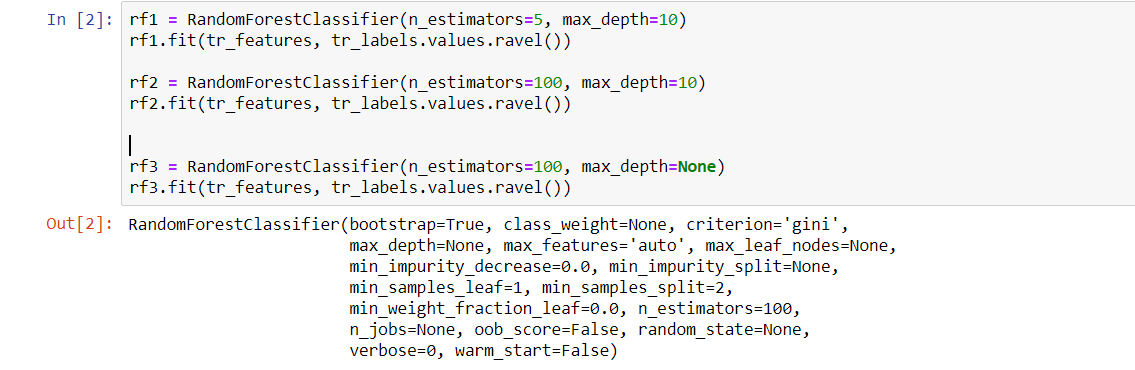
te\_features = pd.read\_csv('../../../test\_features.csv')

te\_labels = pd.read\_csv('../../../test\_labels.csv', header=None)



**Random Forest Classifier :**

Random classification model is employed and made to fit the features whose code to implement the same is given below.



rf1 = RandomForestClassifier(n\_estimators=5, max\_depth=10)

rf1.fit(tr\_features, tr\_labels.values.ravel())

rf2 = RandomForestClassifier(n\_estimators=100, max\_depth=10)

rf2.fit(tr\_features, tr\_labels.values.ravel())

rf3 = RandomForestClassifier(n\_estimators=100, max\_depth=None)

rf3.fit(tr\_features, tr\_labels.values.ravel())

Evaluate Models on Validation Set:

Accuracy= predicted correctly/ total number of examples

Precision=predicted as surviving that actually survived/total number predicted to survive

Recall=predicted as that actually survived/total number that actually survived.

Accuracy, Precision and Recall :

In order to get accuracy,precision,recall use the following code for each random forest. We are almost in the final process of selecting a random forest with high accuracy and precision.

for mdl in [rf1, rf2, rf3]:

y\_pred = mdl.predict(val\_features)

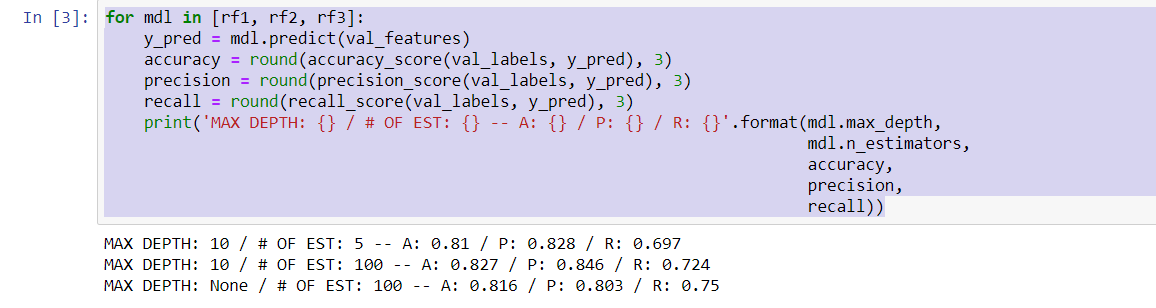
accuracy = round(accuracy\_score(val\_labels, y\_pred), 3)

precision = round(precision\_score(val\_labels, y\_pred), 3)

recall = round(recall\_score(val\_labels, y\_pred), 3)

print('MAX DEPTH: {} / # OF EST: {} -- A: {} / P: {} / R: {}'.format(mdl.max\_depth,

mdl.n\_estimators, accuracy, precision, recall))



Out of the output values its obvious that random forest 2 has highest precision and accuracy.

Now for y\_pred we are feeding the output of rf2’s prediction with attribute of te\_features.

With the help of following code we can achieve it:

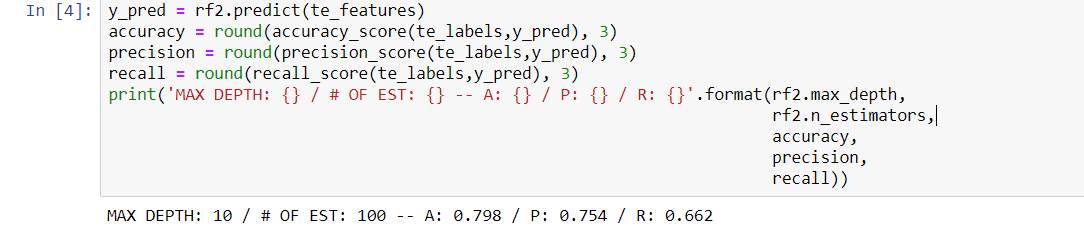
y\_pred = rf2.predict(te\_features)

accuracy = round(accuracy\_score(te\_labels,y\_pred), 3)

precision = round(precision\_score(te\_labels,y\_pred), 3)

recall = round(recall\_score(te\_labels,y\_pred), 3)

print('MAX DEPTH: {} / # OF EST: {} -- A: {} / P: {} / R: {}'.format(rf2.max\_depth, rf2.n\_estimators, accuracy,precision, recall))



At the end we got prediction accuracy for our test features with the help of random forest method as 79.8%

References:

1. <https://www.kaggle.com/>
2. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
3. <https://www.therandomvibez.com/top-titanic-quotes/>
4. <https://seaborn.pydata.org/generated/seaborn.catplot.html>
5. <https://www.datacamp.com/community/tutorials/random-forests-classifier-python>