This data was extracted from the census bureau database found at

http://www.census.gov/ftp/pub/DES/www/welcome.html

Donor: Ronny Kohavi and Barry Becker,

Data Mining and Visualization

Silicon Graphics.

e-mail: ronnyk@sgi.com for questions.

Split into train-test using MLC++ GenCVFiles (2/3, 1/3 random).

48842 instances, mix of continuous and discrete (train=32561, test=16281)

45222 if instances with unknown values are removed (train=30162, test=15060)

Duplicate or conflicting instances : 6

Class probabilities for adult.all file

Probability for the label '>50K' : 23.93% / 24.78% (without unknowns)

Probability for the label '<=50K' : 76.07% / 75.22% (without unknowns)

Extraction was done by Barry Becker from the 1994 Census database. A set of

reasonably clean records was extracted using the following conditions:

((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0)) Prediction task is to

determine whether a person makes over 50K a year. Conversion of original data as

follows:

1. Discretized a gross income into two ranges with threshold 50,000.

2. Convert U.S. to US to avoid periods.

3. Convert Unknown to "?"

4. Run MLC++ GenCVFiles to generate data,test.

Description of fnlwgt (final weight)

The weights on the CPS files are controlled to independent estimates of the civilian

noninstitutional population of the US. These are prepared monthly for us by Population

Division here at the Census Bureau. We use 3 sets of controls.

These are:

1. A single cell estimate of the population 16+ for each state.

2. Controls for Hispanic Origin by age and sex.

3. Controls by Race, age and sex.

We use all three sets of controls in our weighting program and "rake" through them 6

times so that by the end we come back to all the controls we used.

The term estimate refers to population totals derived from CPS by creating "weighted

tallies" of any specified socio-economic characteristics of the population. People with

similar demographic characteristics should have similar weights. There is one important

caveat to remember about this statement. That is that since the CPS sample is actually a

collection of 51 state samples, each with its own probability of selection, the statement

only applies within state.

**Dataset Link**

<https://archive.ics.uci.edu/ml/machine-learning-databases/adult/>

**Problem 1:**

Prediction task is to determine whether a person makes over 50K a year.

**Problem 2:**

Which factors are important

**Problem 3:**

Which algorithms are best for this dataset

Code:

#importing the important libraries

import numpy as np

import pandas as pd

import xgboost as xgb

from sklearn.metrics import accuracy\_score, confusion\_matrix

from xgboost.sklearn import XGBClassifier

import matplotlib.pyplot as plt

%matplotlib inline

dataset = ['train.csv',

'test.csv']

dataset[0]

# loading training set of data

train\_set = pd.read\_csv(dataset[0],header=None)

# loading test set of data

test\_set = pd.read\_csv(dataset[1], header = None)

# name of the columns

col\_labels = ['age', 'workclass', 'fnlwgt', 'education', 'education\_num', 'marital\_status','occupation','relationship', 'race', 'sex',

'capital\_gain', 'capital\_loss', 'hours\_per\_week',

'native\_country', 'wage\_class']

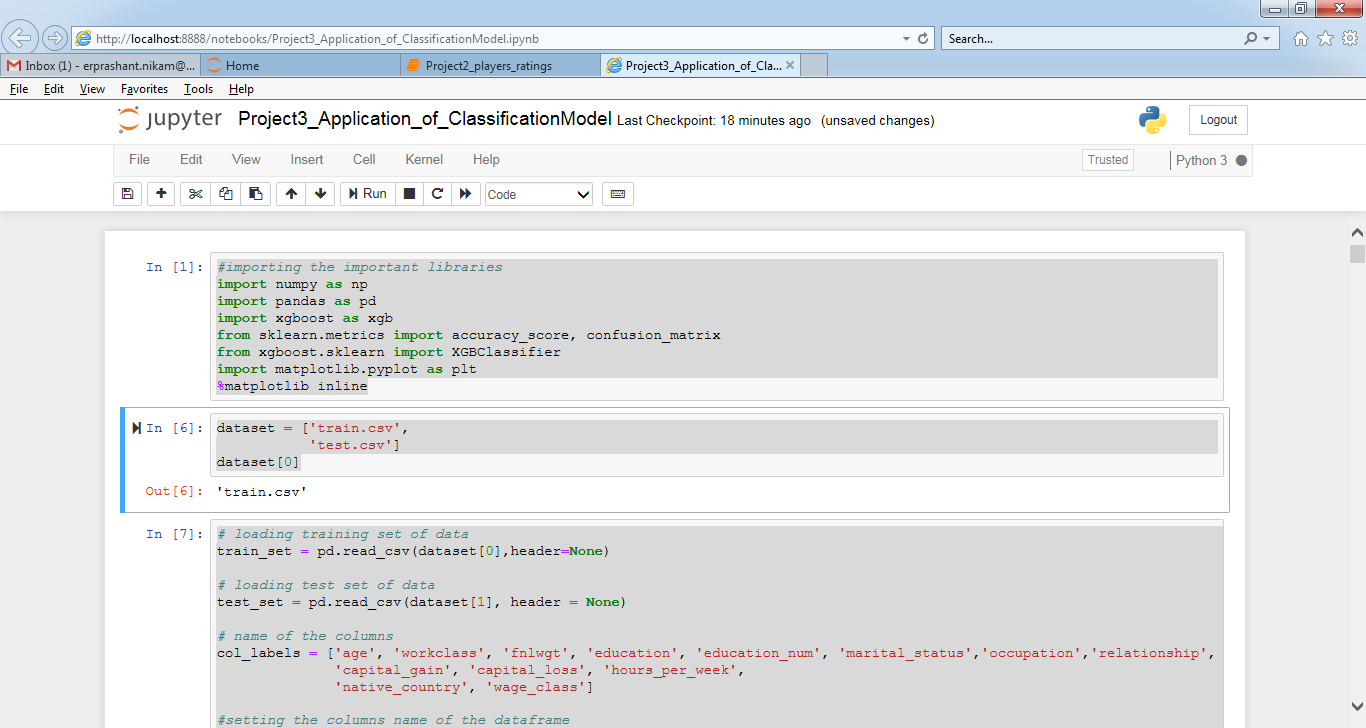
#setting the columns name of the dataframe

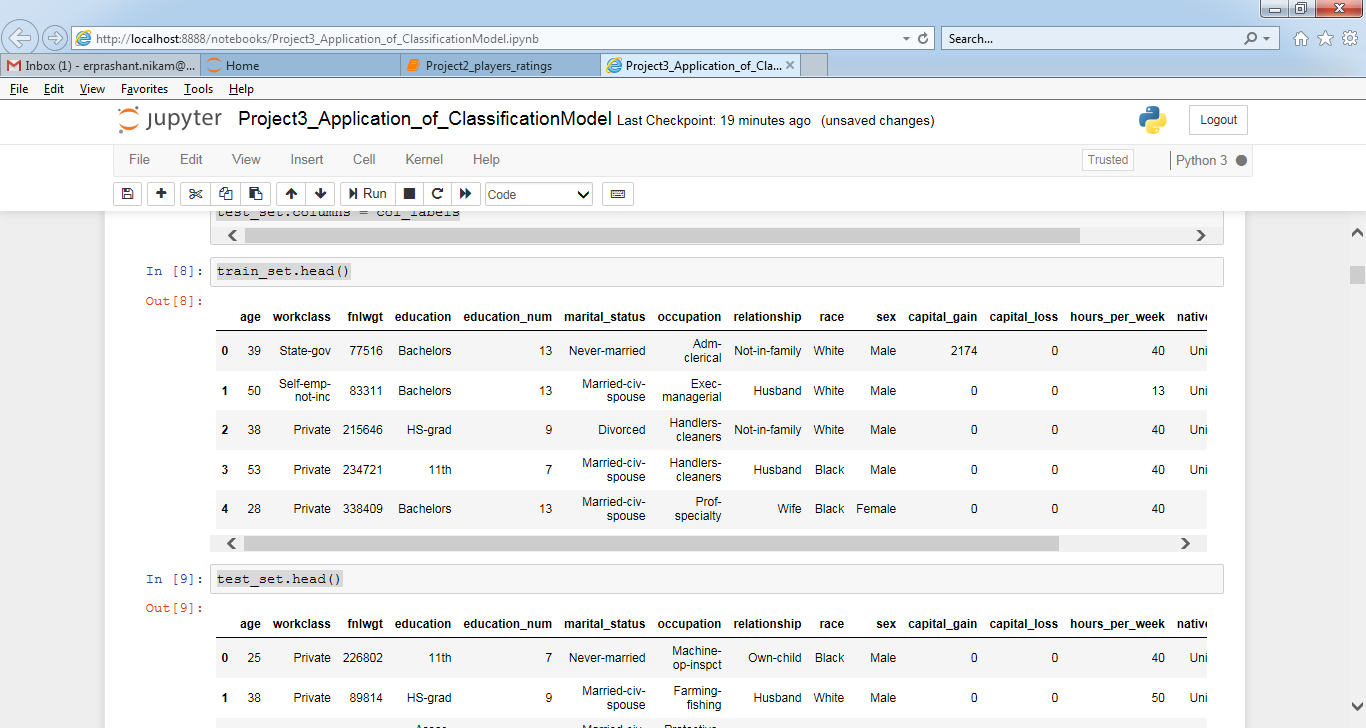
train\_set.columns = col\_labels

test\_set.columns = col\_labels

train\_set.head()

test\_set.head()



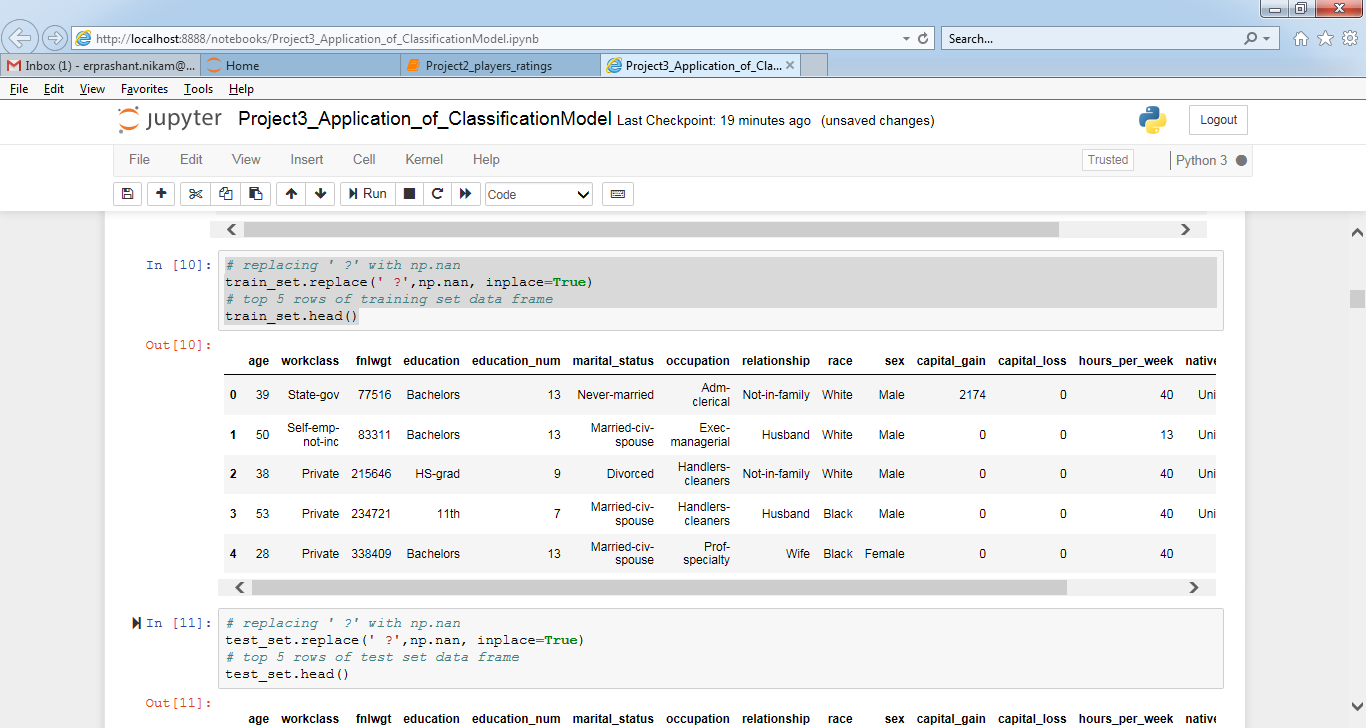


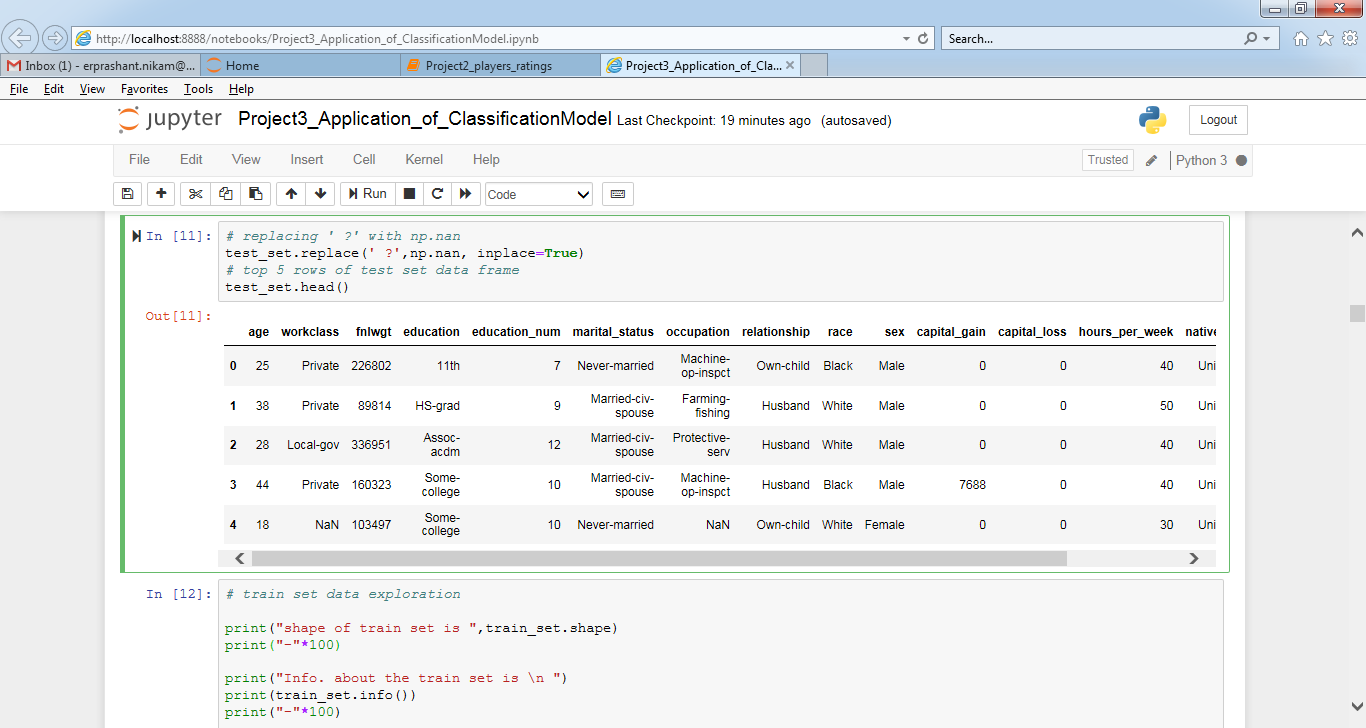
# replacing ' ?' with np.nan

train\_set.replace(' ?',np.nan, inplace=True)

# top 5 rows of training set data frame

train\_set.head()





# train set data exploration

print("shape of train set is ",train\_set.shape)

print("-"\*100)

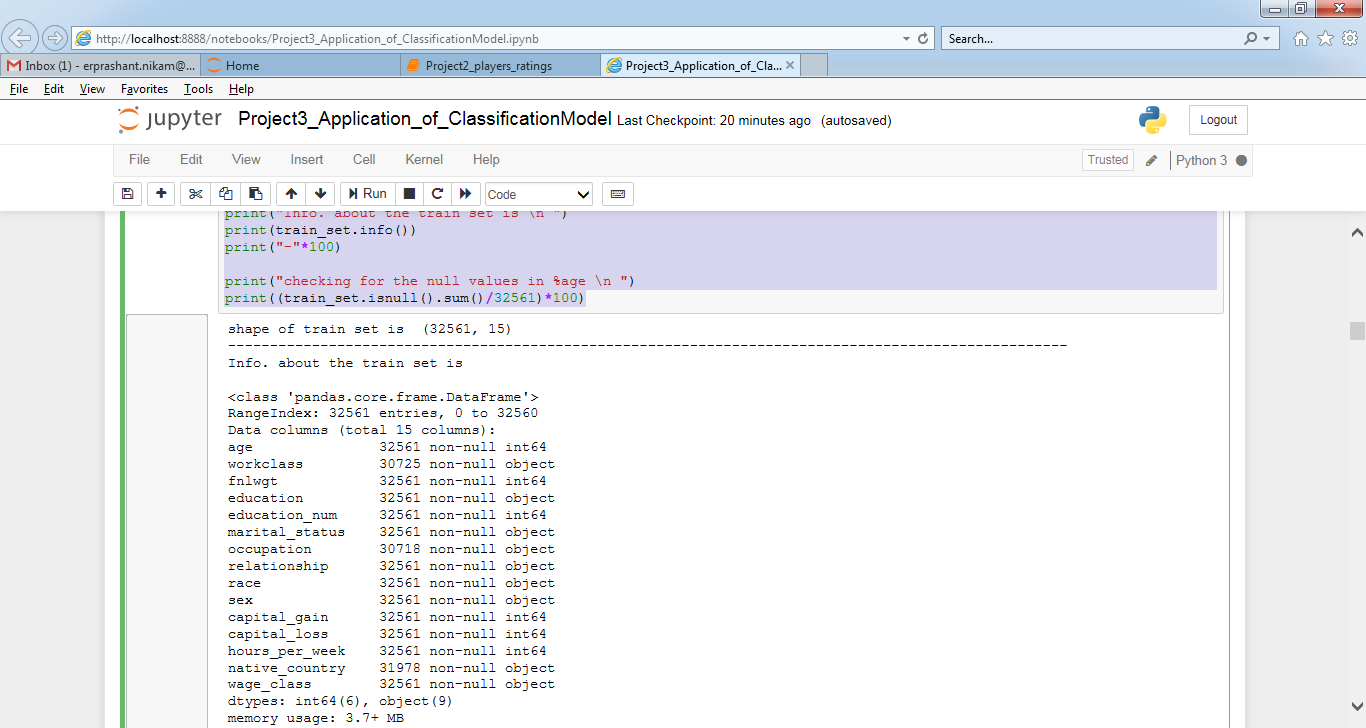
print("Info. about the train set is \n ")

print(train\_set.info())

print("-"\*100)

print("checking for the null values in %age \n ")

print((train\_set.isnull().sum()/32561)\*100)



# test set data exploration

print("shape of test set is ",test\_set.shape)

print("-"\*100)

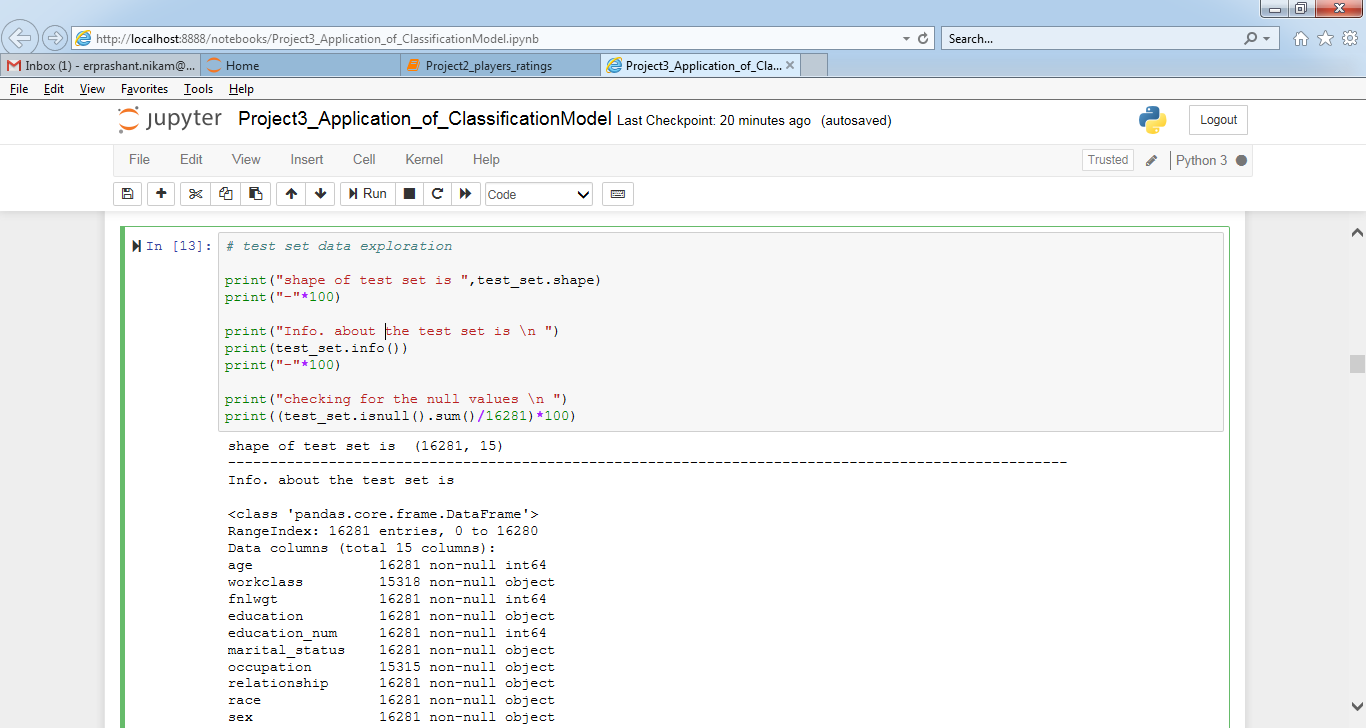
print("Info. about the test set is \n ")

print(test\_set.info())

print("-"\*100)

print("checking for the null values \n ")

print((test\_set.isnull().sum()/16281)\*100)

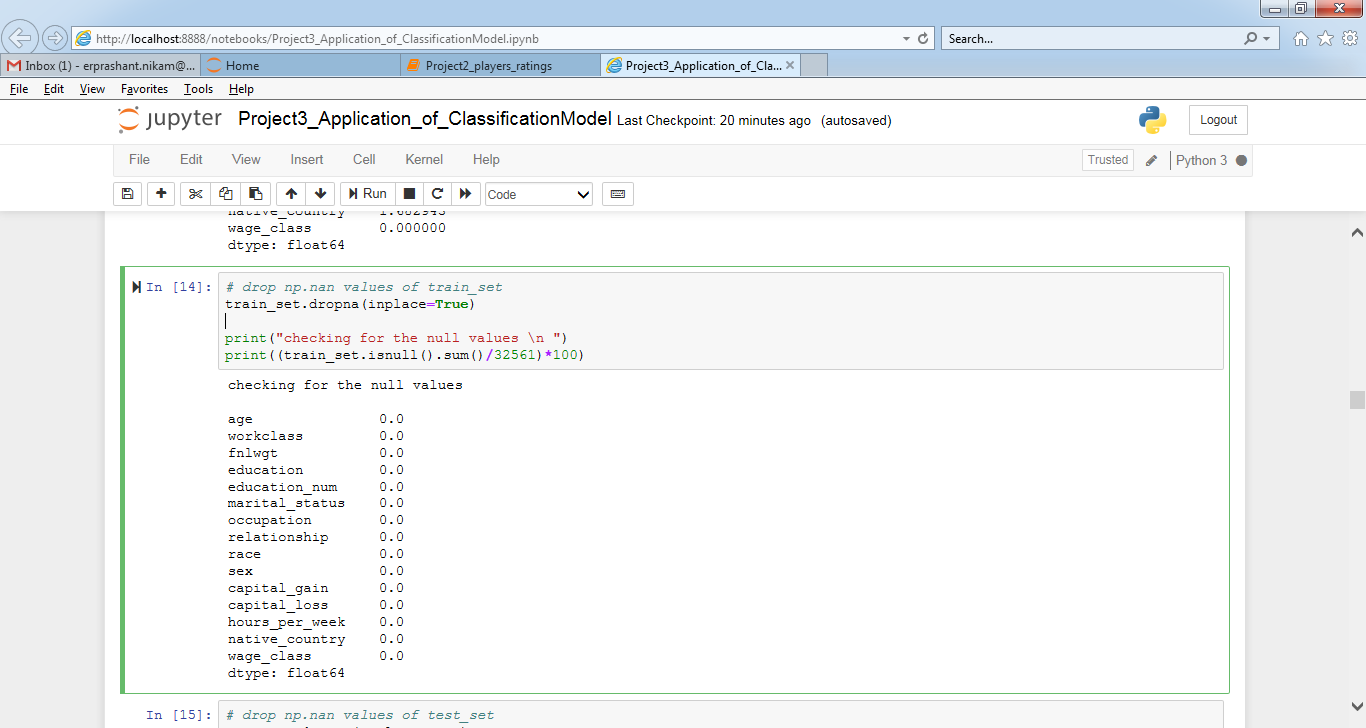


# drop np.nan values of train\_set

train\_set.dropna(inplace=True)

print("checking for the null values \n ")

print((train\_set.isnull().sum()/32561)\*100)

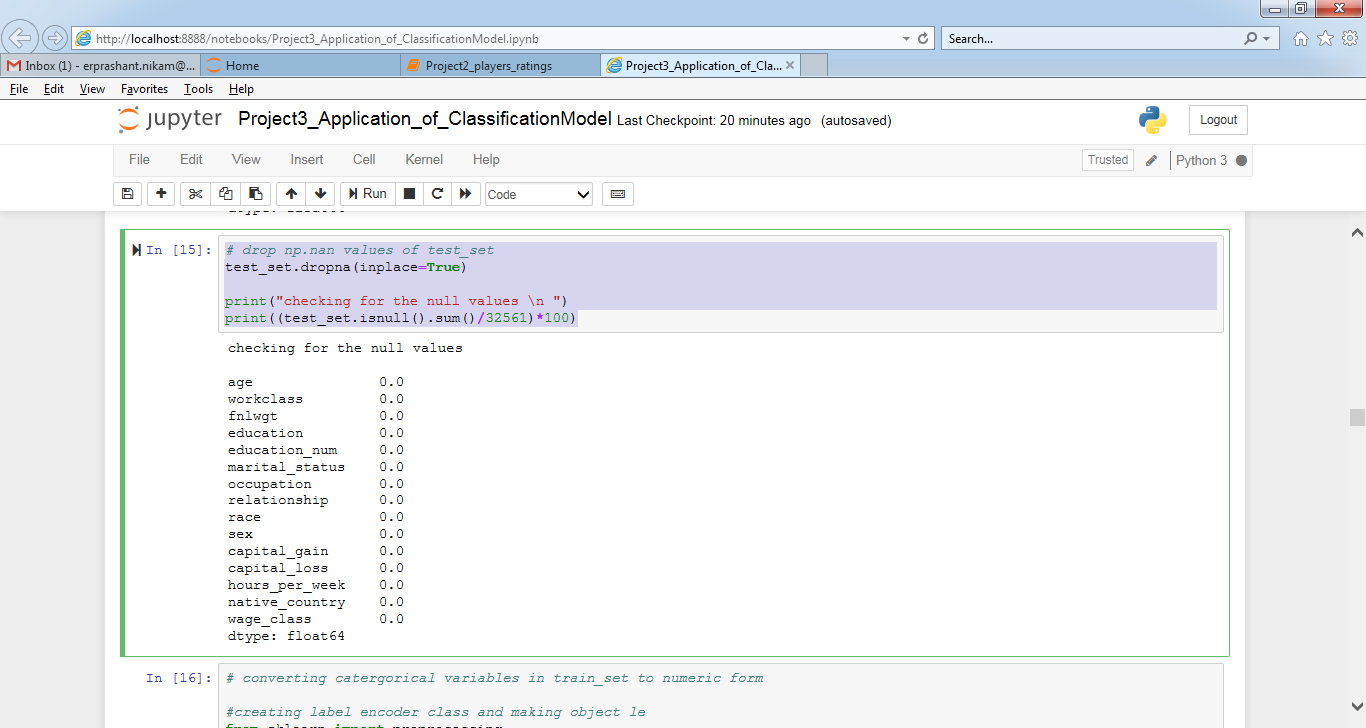


# drop np.nan values of test\_set

test\_set.dropna(inplace=True)

print("checking for the null values \n ")

print((test\_set.isnull().sum()/32561)\*100)



# converting catergorical variables in train\_set to numeric form

#creating label encoder class and making object le

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

# converting catergorical to numeric

train\_set['workclass']=le.fit\_transform(train\_set.workclass.values)

train\_set['education']=le.fit\_transform(train\_set.education.values)

train\_set['marital\_status']=le.fit\_transform(train\_set.marital\_status.values)

train\_set['occupation']=le.fit\_transform(train\_set.occupation.values)

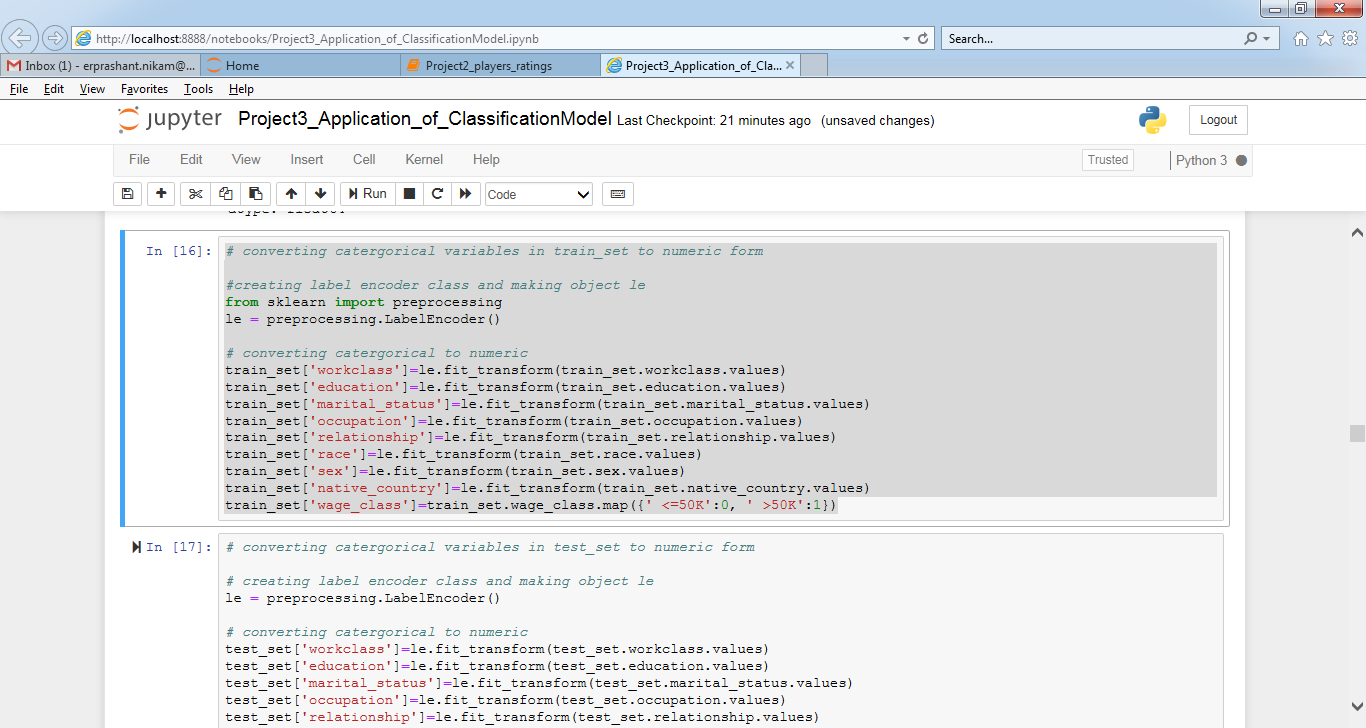
train\_set['relationship']=le.fit\_transform(train\_set.relationship.values)

train\_set['race']=le.fit\_transform(train\_set.race.values)

train\_set['sex']=le.fit\_transform(train\_set.sex.values)

train\_set['native\_country']=le.fit\_transform(train\_set.native\_country.values)

train\_set['wage\_class']=train\_set.wage\_class.map({' <=50K':0, ' >50K':1})



# converting catergorical variables in test\_set to numeric form

# creating label encoder class and making object le

le = preprocessing.LabelEncoder()

# converting catergorical to numeric

test\_set['workclass']=le.fit\_transform(test\_set.workclass.values)

test\_set['education']=le.fit\_transform(test\_set.education.values)

test\_set['marital\_status']=le.fit\_transform(test\_set.marital\_status.values)

test\_set['occupation']=le.fit\_transform(test\_set.occupation.values)

test\_set['relationship']=le.fit\_transform(test\_set.relationship.values)

test\_set['race']=le.fit\_transform(test\_set.race.values)

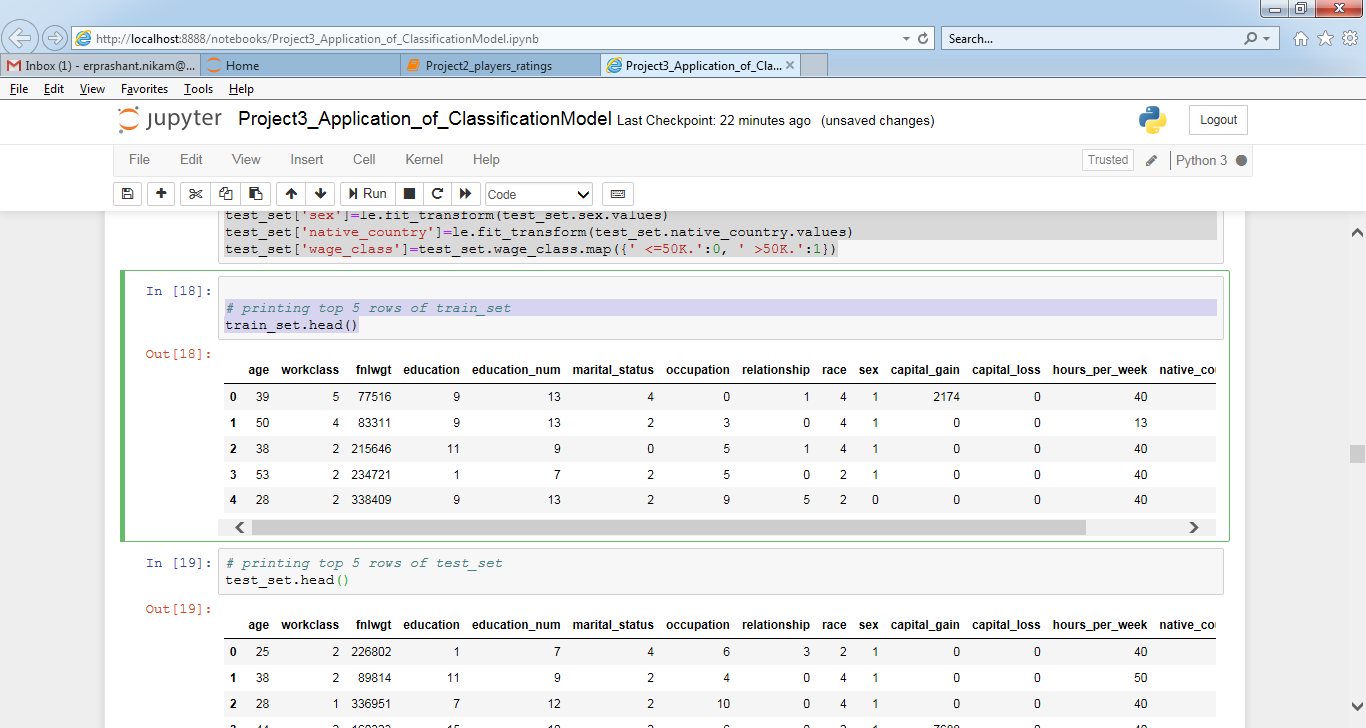
test\_set['sex']=le.fit\_transform(test\_set.sex.values)

test\_set['native\_country']=le.fit\_transform(test\_set.native\_country.values)

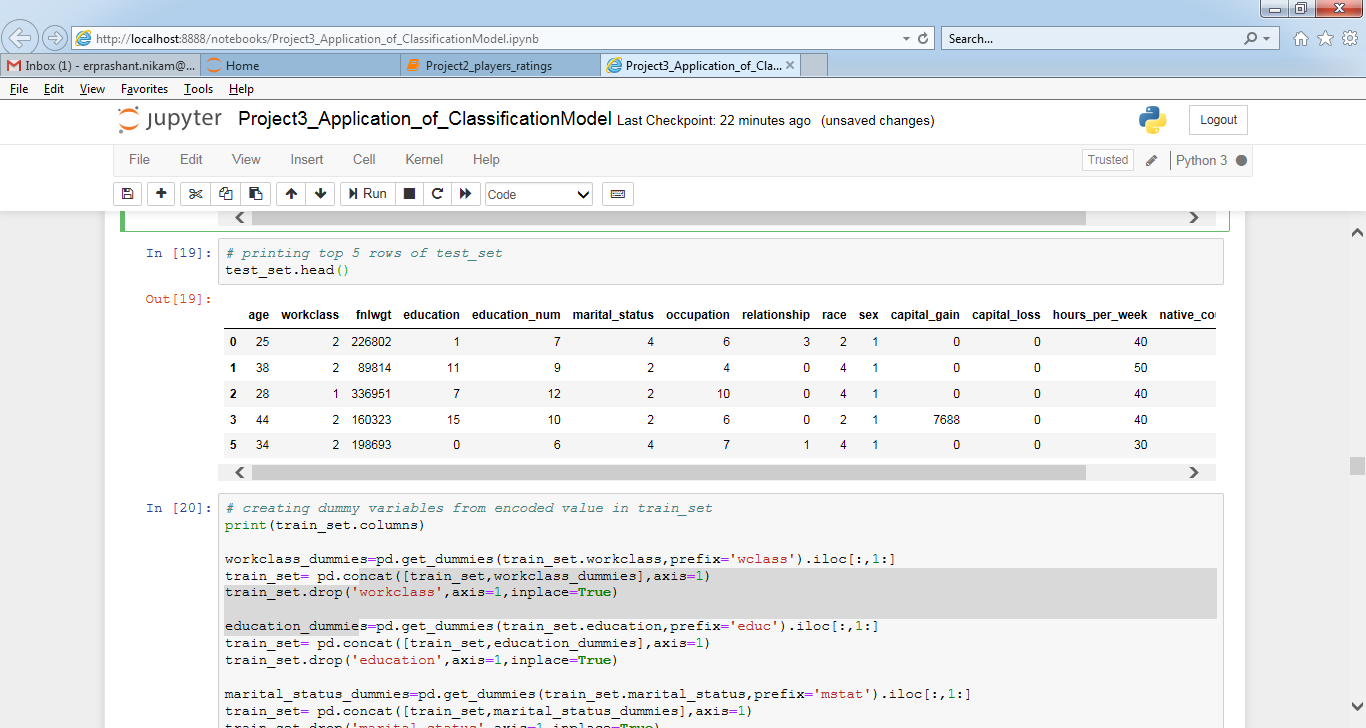
test\_set['wage\_class']=test\_set.wage\_class.map({' <=50K.':0, ' >50K.':1})

# printing top 5 rows of train\_set

train\_set.head()



test\_set.head()



# creating dummy variables from encoded value in train\_set

print(train\_set.columns)

workclass\_dummies=pd.get\_dummies(train\_set.workclass,prefix='wclass').iloc[:,1:]

train\_set= pd.concat([train\_set,workclass\_dummies],axis=1)

train\_set.drop('workclass',axis=1,inplace=True)

education\_dummies=pd.get\_dummies(train\_set.education,prefix='educ').iloc[:,1:]

train\_set= pd.concat([train\_set,education\_dummies],axis=1)

train\_set.drop('education',axis=1,inplace=True)

marital\_status\_dummies=pd.get\_dummies(train\_set.marital\_status,prefix='mstat').iloc[:,1:]

train\_set= pd.concat([train\_set,marital\_status\_dummies],axis=1)

train\_set.drop('marital\_status',axis=1,inplace=True)

occupation\_dummies=pd.get\_dummies(train\_set.occupation,prefix='occup').iloc[:,1:]

train\_set= pd.concat([train\_set,occupation\_dummies],axis=1)

train\_set.drop('occupation',axis=1,inplace=True)

relationship\_dummies=pd.get\_dummies(train\_set.relationship,prefix='rltnshp').iloc[:,1:]

train\_set= pd.concat([train\_set,relationship\_dummies],axis=1)

train\_set.drop('relationship',axis=1,inplace=True)

race\_dummies=pd.get\_dummies(train\_set.race,prefix='race').iloc[:,1:]

train\_set= pd.concat([train\_set,race\_dummies],axis=1)

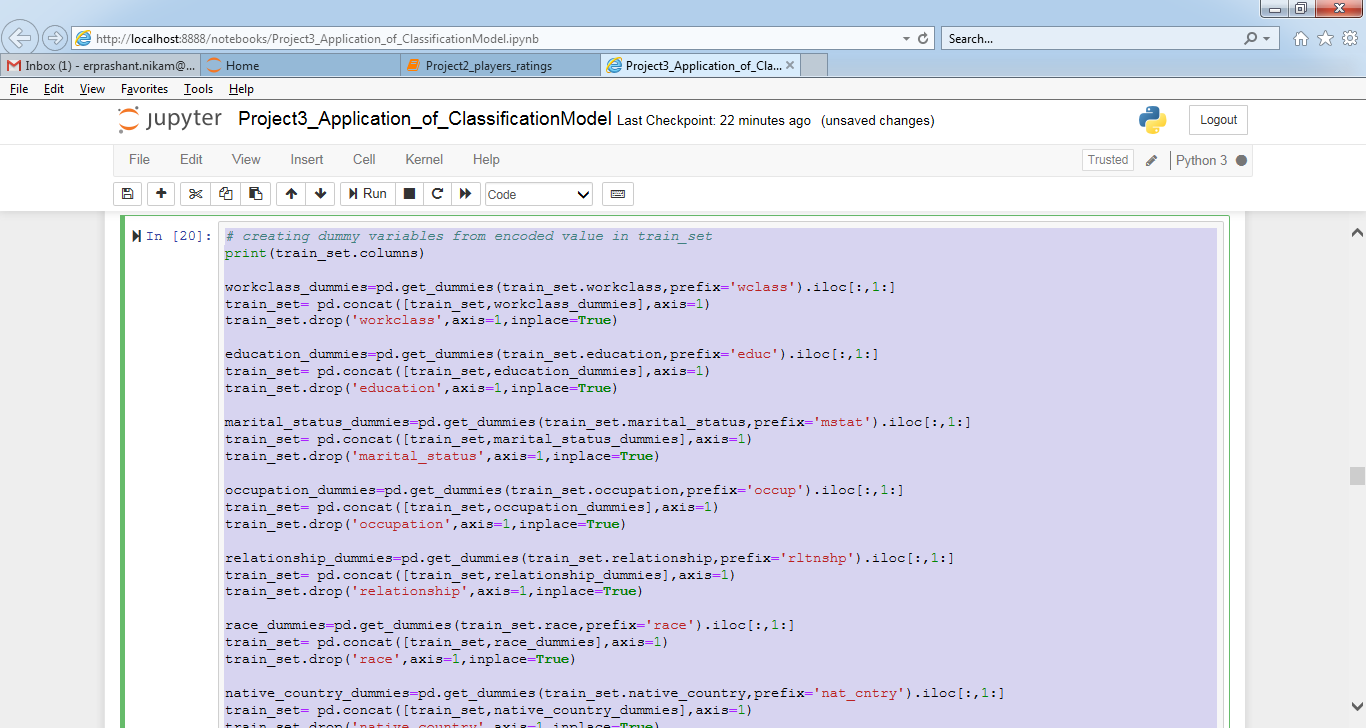
train\_set.drop('race',axis=1,inplace=True)

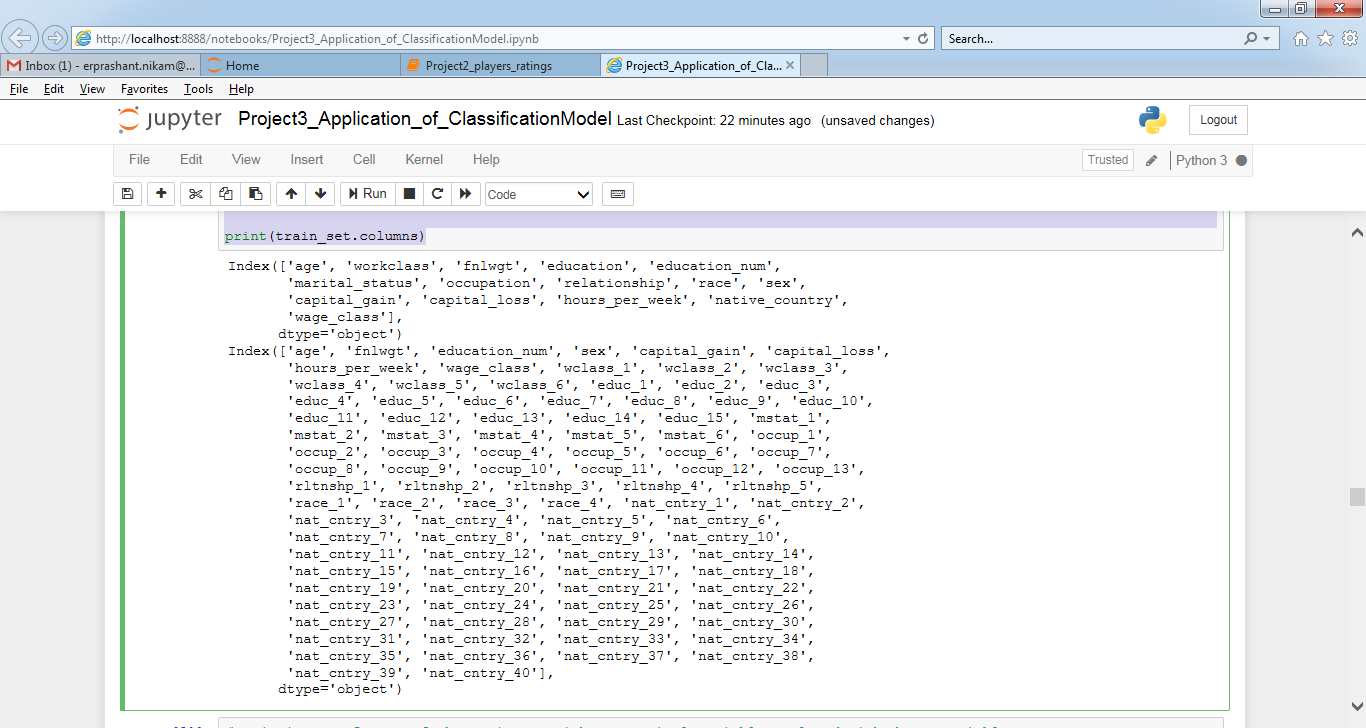
native\_country\_dummies=pd.get\_dummies(train\_set.native\_country,prefix='nat\_cntry').iloc[:,1:]

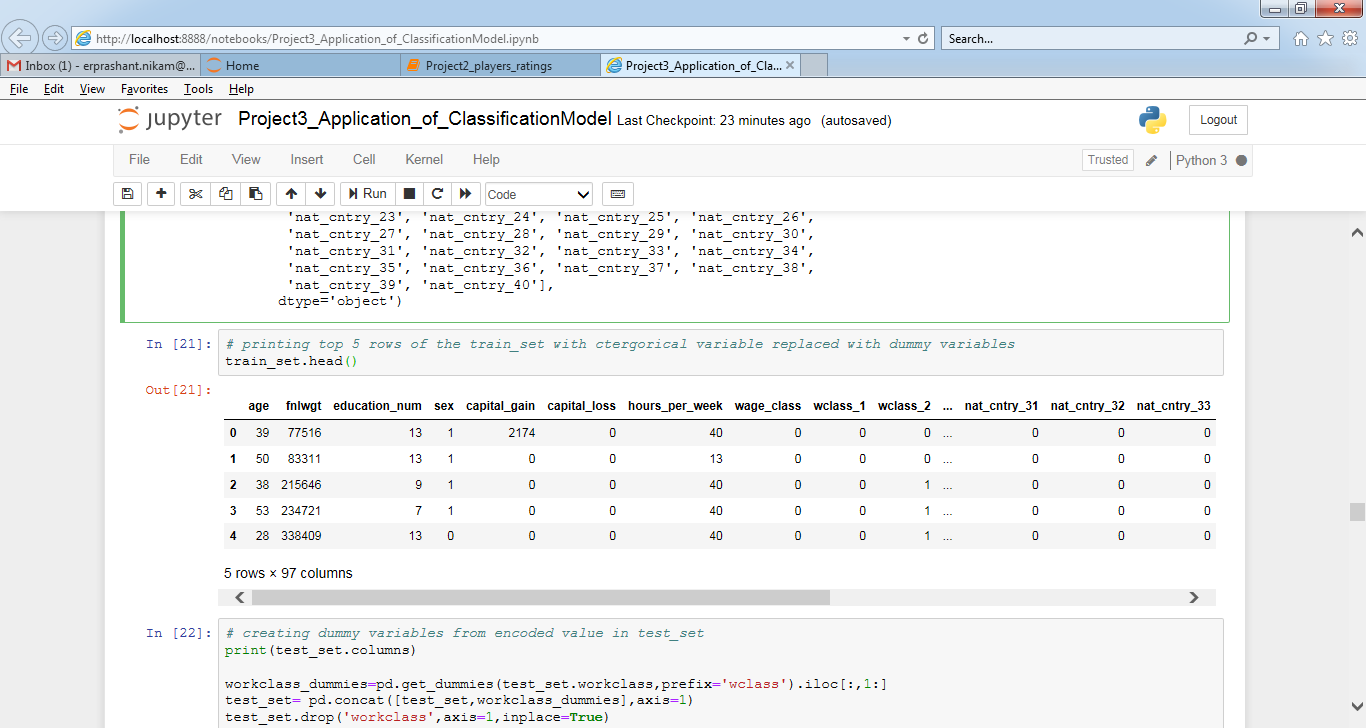
train\_set= pd.concat([train\_set,native\_country\_dummies],axis=1)

train\_set.drop('native\_country',axis=1,inplace=True)

print(train\_set.columns)







# creating dummy variables from encoded value in test\_set

print(test\_set.columns)

workclass\_dummies=pd.get\_dummies(test\_set.workclass,prefix='wclass').iloc[:,1:]

test\_set= pd.concat([test\_set,workclass\_dummies],axis=1)

test\_set.drop('workclass',axis=1,inplace=True)

education\_dummies=pd.get\_dummies(test\_set.education,prefix='educ').iloc[:,1:]

test\_set= pd.concat([test\_set,education\_dummies],axis=1)

test\_set.drop('education',axis=1,inplace=True)

marital\_status\_dummies=pd.get\_dummies(test\_set.marital\_status,prefix='mstat').iloc[:,1:]

test\_set= pd.concat([test\_set,marital\_status\_dummies],axis=1)

test\_set.drop('marital\_status',axis=1,inplace=True)

occupation\_dummies=pd.get\_dummies(test\_set.occupation,prefix='occup').iloc[:,1:]

test\_set= pd.concat([test\_set,occupation\_dummies],axis=1)

test\_set.drop('occupation',axis=1,inplace=True)

relationship\_dummies=pd.get\_dummies(test\_set.relationship,prefix='rltnshp').iloc[:,1:]

test\_set= pd.concat([test\_set,relationship\_dummies],axis=1)

test\_set.drop('relationship',axis=1,inplace=True)

race\_dummies=pd.get\_dummies(test\_set.race,prefix='race').iloc[:,1:]

test\_set= pd.concat([test\_set,race\_dummies],axis=1)

test\_set.drop('race',axis=1,inplace=True)

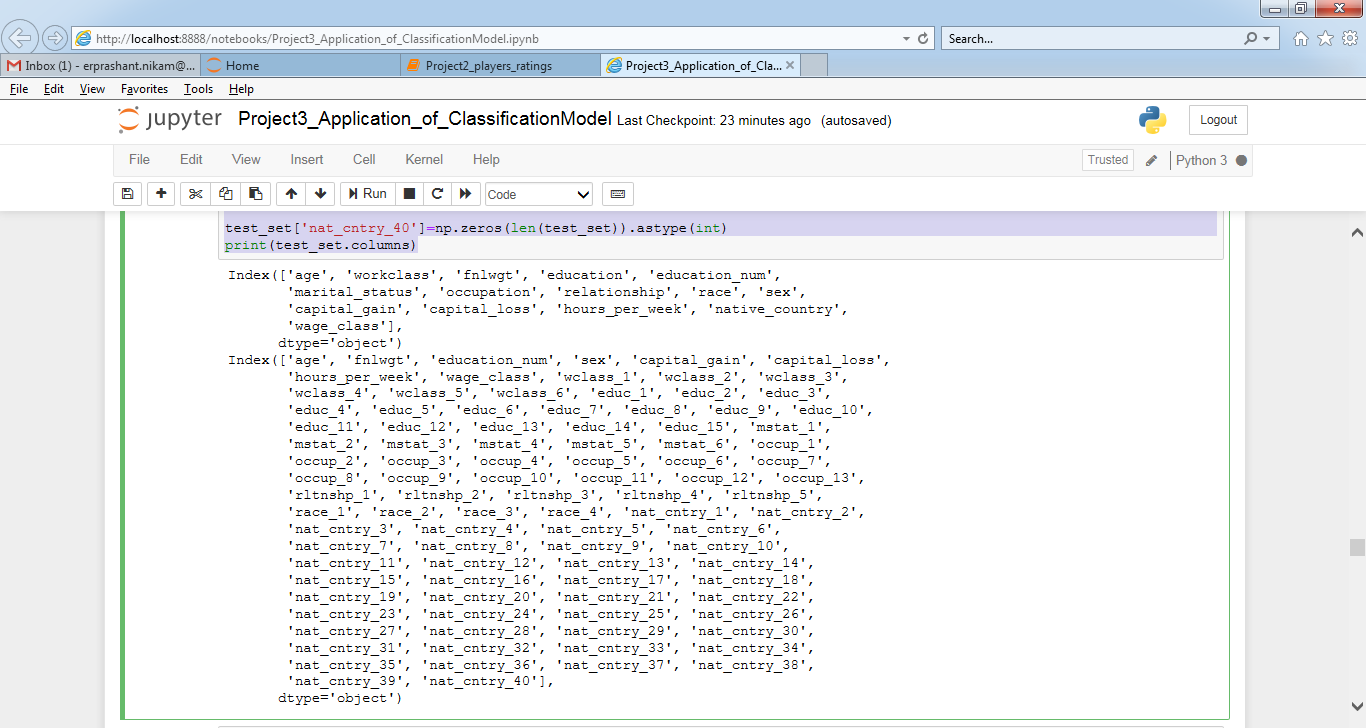
native\_country\_dummies=pd.get\_dummies(test\_set.native\_country,prefix='nat\_cntry').iloc[:,1:]

test\_set= pd.concat([test\_set,native\_country\_dummies],axis=1)

test\_set.drop('native\_country',axis=1,inplace=True)

test\_set['nat\_cntry\_40']=np.zeros(len(test\_set)).astype(int)

print(test\_set.columns)

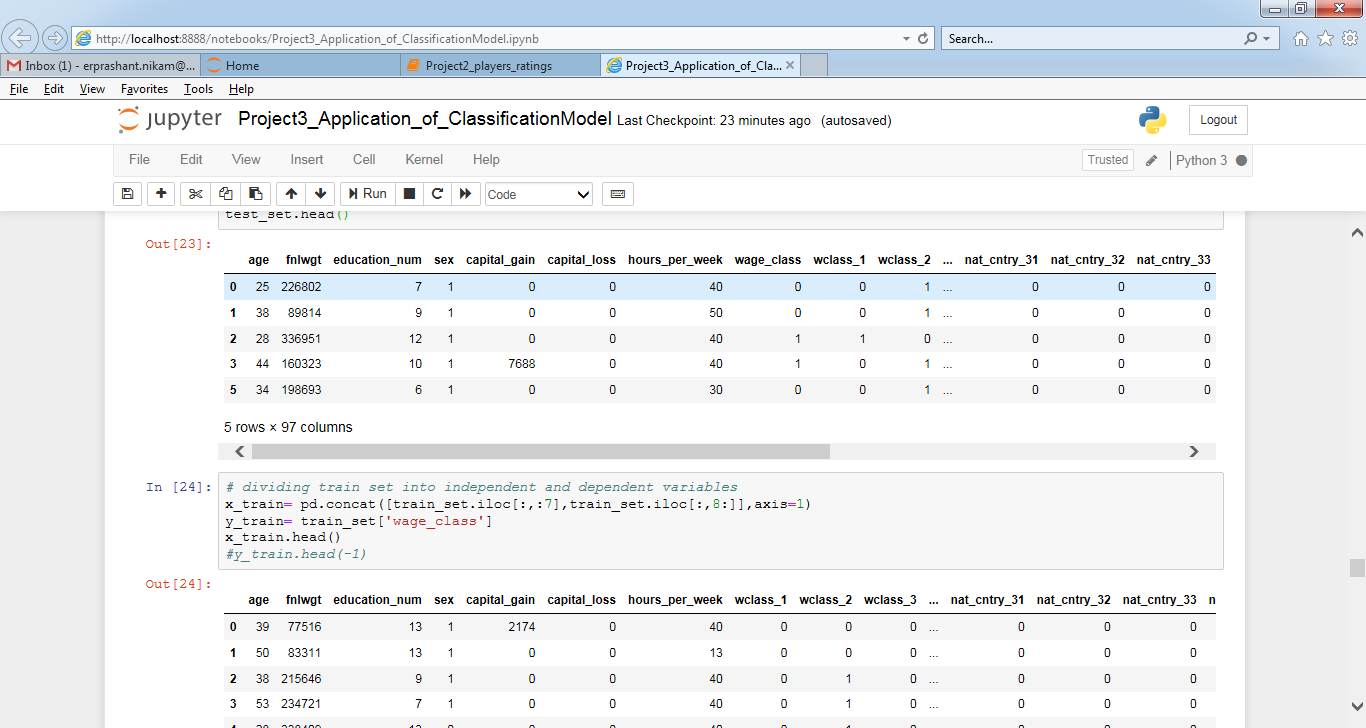


# dividing train set into independent and dependent variables

x\_train= pd.concat([train\_set.iloc[:,:7],train\_set.iloc[:,8:]],axis=1)

y\_train= train\_set['wage\_class']

x\_train.head()

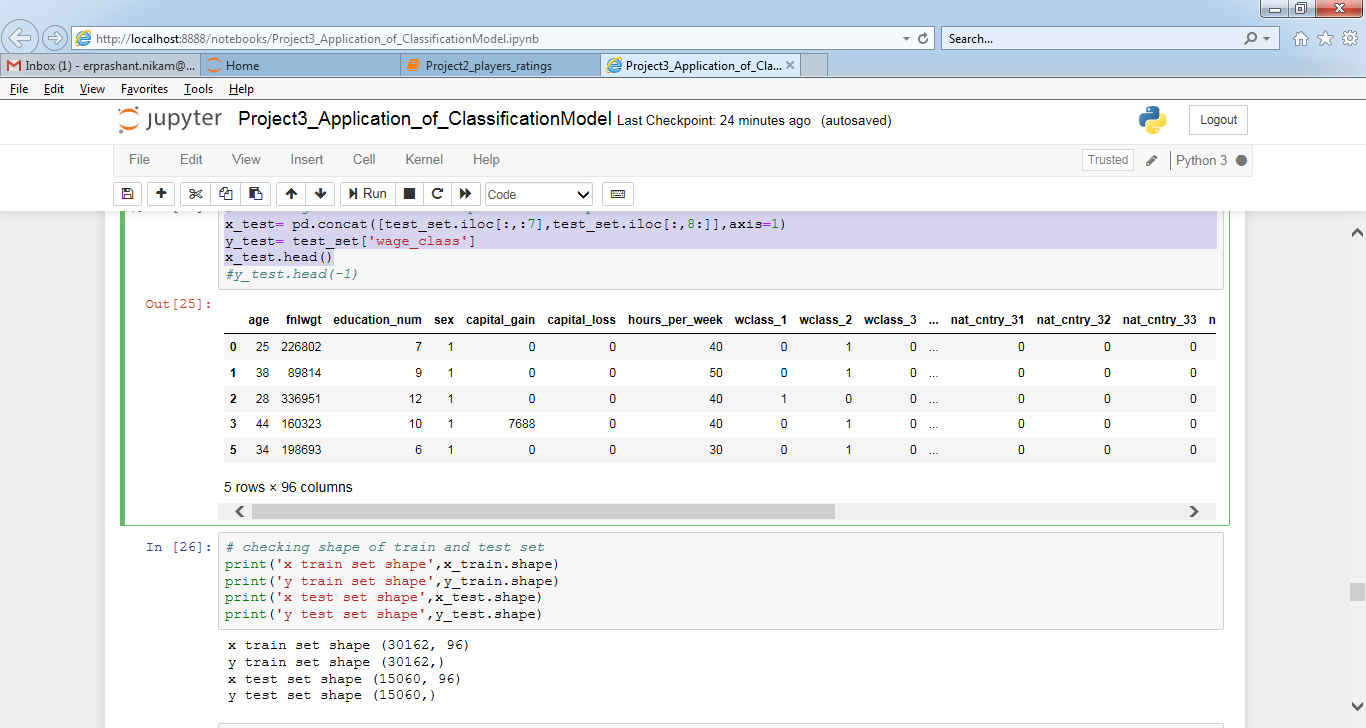


# dividing test set into independent and dependent variables

x\_test= pd.concat([test\_set.iloc[:,:7],test\_set.iloc[:,8:]],axis=1)

y\_test= test\_set['wage\_class']

x\_test.head()



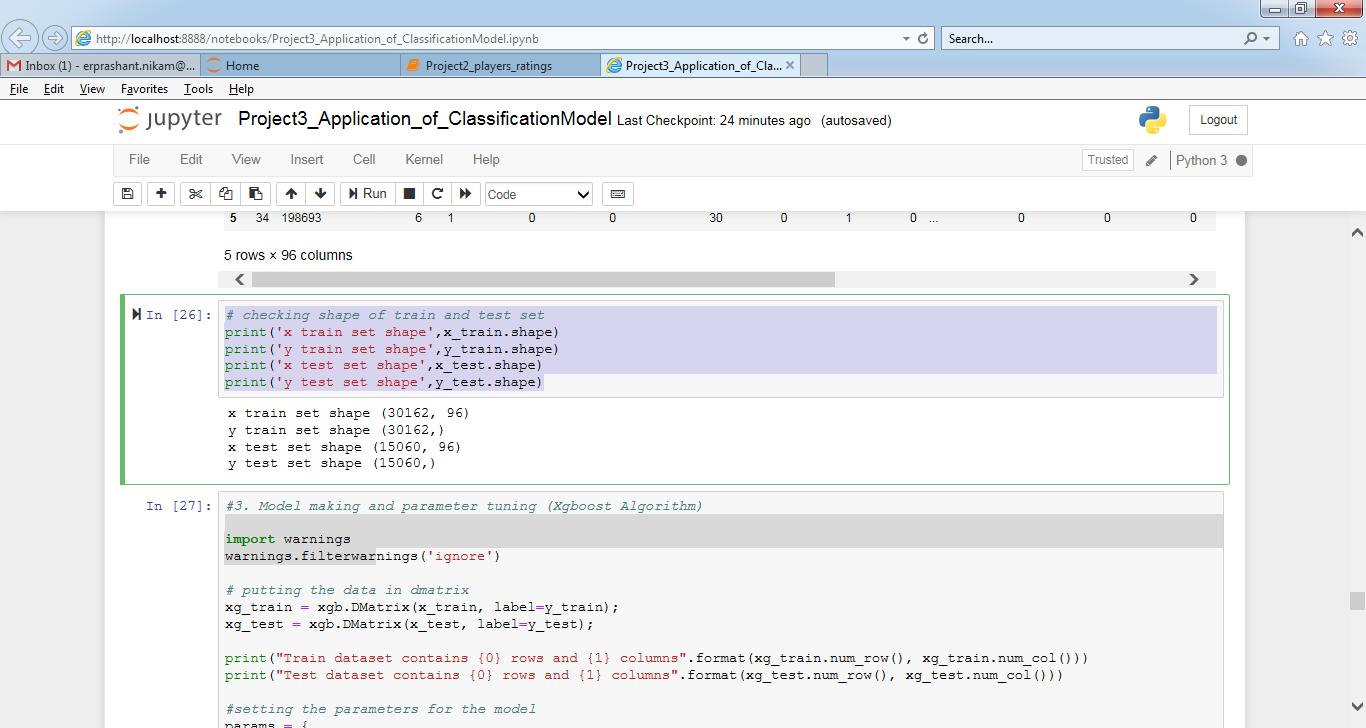
# checking shape of train and test set

print('x train set shape',x\_train.shape)

print('y train set shape',y\_train.shape)

print('x test set shape',x\_test.shape)

print('y test set shape',y\_test.shape)



#3. Model making and parameter tuning (Xgboost Algorithm)

import warnings

warnings.filterwarnings('ignore')

# putting the data in dmatrix

xg\_train = xgb.DMatrix(x\_train, label=y\_train);

xg\_test = xgb.DMatrix(x\_test, label=y\_test);

print("Train dataset contains {0} rows and {1} columns".format(xg\_train.num\_row(), xg\_train.num\_col()))

print("Test dataset contains {0} rows and {1} columns".format(xg\_test.num\_row(), xg\_test.num\_col()))

#setting the parameters for the model

params = {

'objective': 'binary:logistic',

'max\_depth': 5,

'learning\_rate': 0.1,

'silent': 1.0,

'n\_estimators':50

}

# training the model

bst = XGBClassifier(\*\*params).fit(x\_train, y\_train)

# predicting the model on test set

y\_predict = bst.predict(x\_test)

# calculating accuracy and confusion matrix

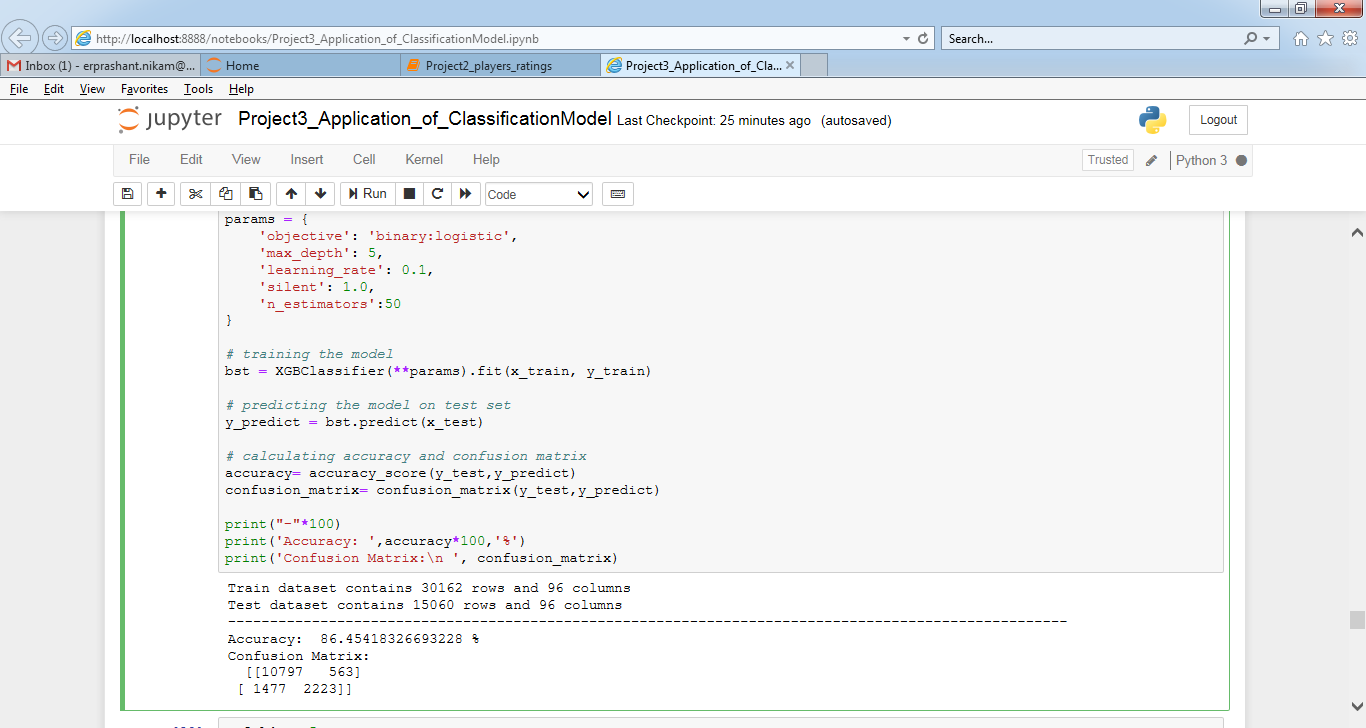
accuracy= accuracy\_score(y\_test,y\_predict)

confusion\_matrix= confusion\_matrix(y\_test,y\_predict)

print("-"\*100)

print('Accuracy: ',accuracy\*100,'%')

print('Confusion Matrix:\n ', confusion\_matrix)



n\_folds = 5

early\_stopping = 50

#setting the params to calculate cross validation

params = {

'eta': 0.1,

'max\_depth': 5,

'objective': 'binary:logistic',

'seed': 99,

'silent': 1,

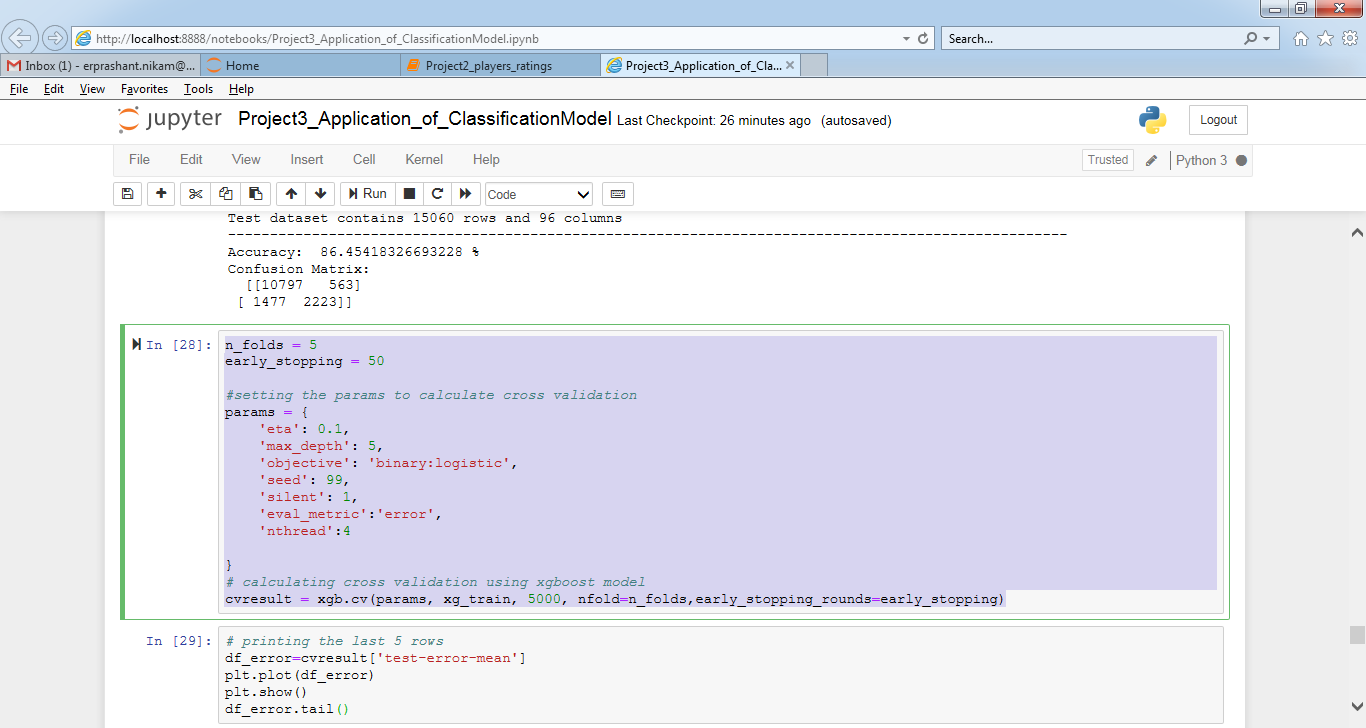
'eval\_metric':'error',

'nthread':4

}

# calculating cross validation using xgboost model

cvresult = xgb.cv(params, xg\_train, 5000, nfold=n\_folds,early\_stopping\_rounds=early\_stopping)



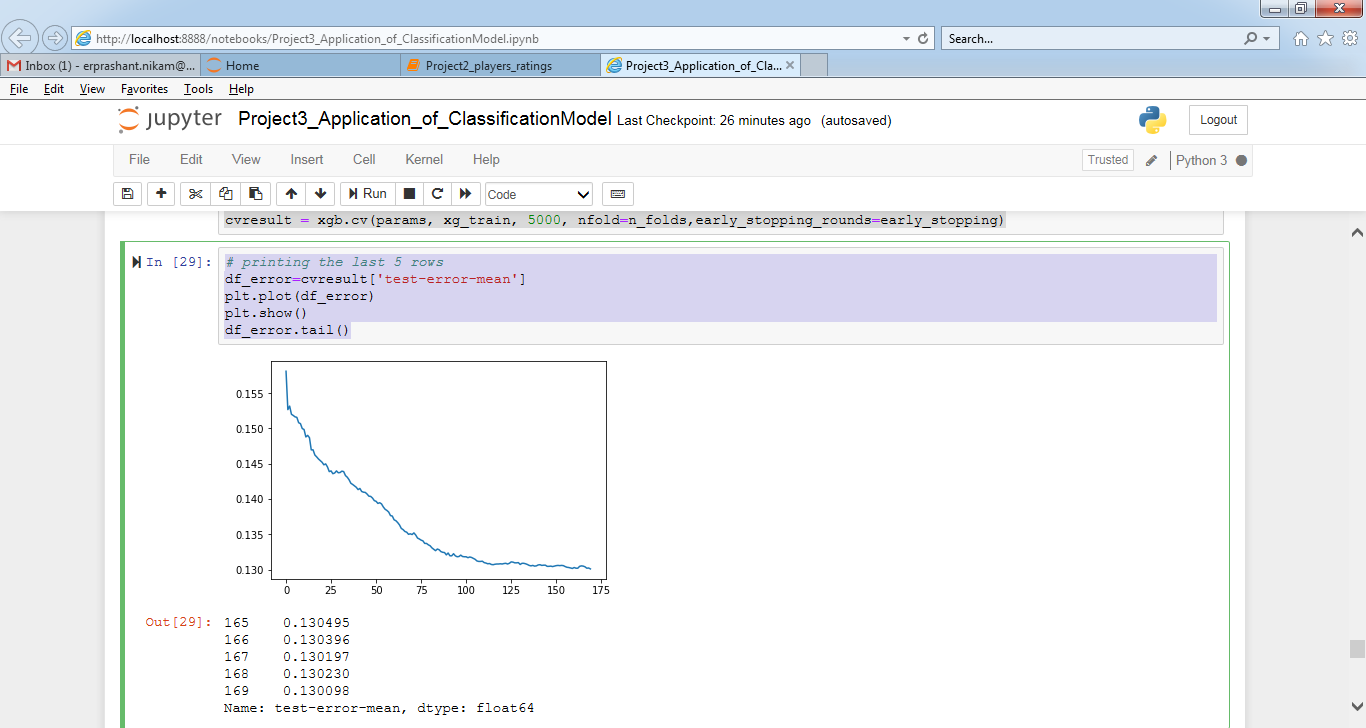
# printing the last 5 rows

df\_error=cvresult['test-error-mean']

plt.plot(df\_error)

plt.show()

df\_error.tail()



# training the model and predicting the value with the updated value of no of estimators =170

from sklearn.metrics import accuracy\_score, confusion\_matrix

from xgboost.sklearn import XGBClassifier

# setting the params to train the model

params1 = {

'objective': 'binary:logistic',

'max\_depth': 5,

'learning\_rate': 0.1,

'silent': 1.0,

'n\_estimators':cvresult.shape[0]

}

#training the model

bst = XGBClassifier(\*\*params1).fit(x\_train, y\_train)

# predicting the value on the test set

y\_predict = bst.predict(x\_test)

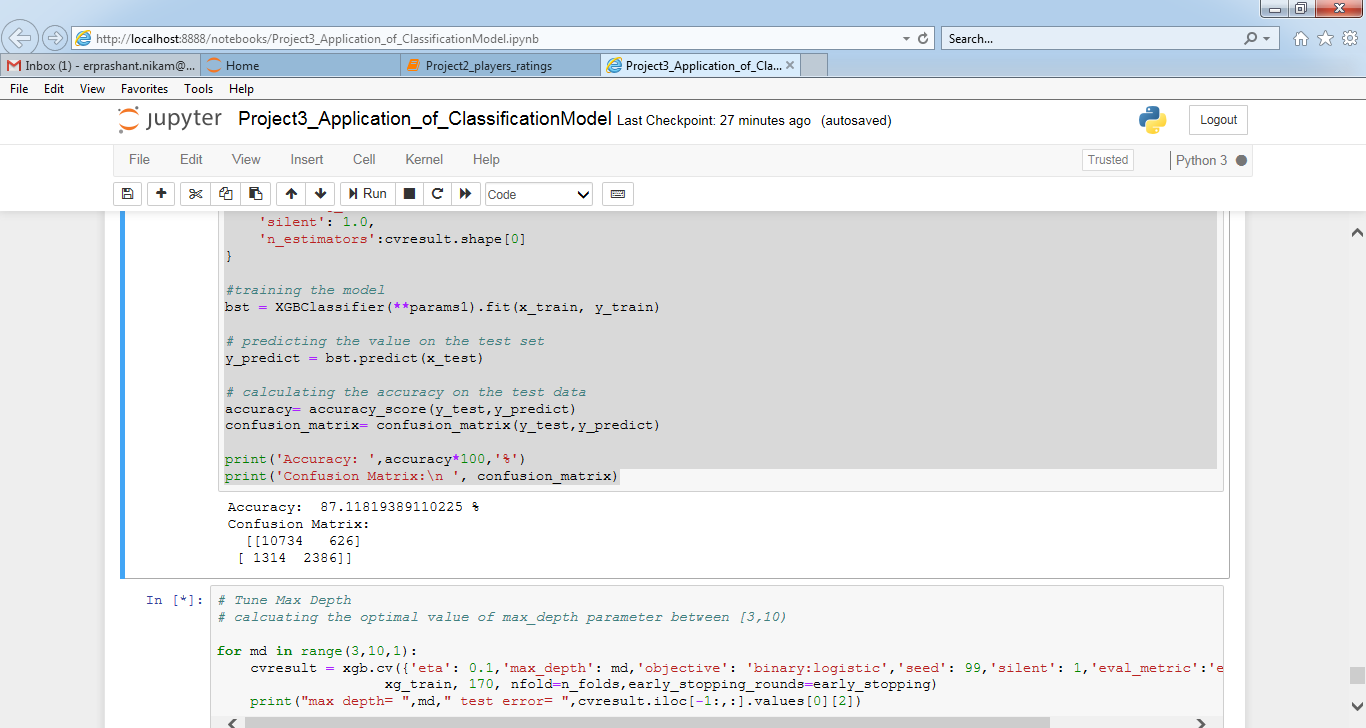
# calculating the accuracy on the test data

accuracy= accuracy\_score(y\_test,y\_predict)

confusion\_matrix= confusion\_matrix(y\_test,y\_predict)

print('Accuracy: ',accuracy\*100,'%')

print('Confusion Matrix:\n ', confusion\_matrix)



# Tune Max Depth

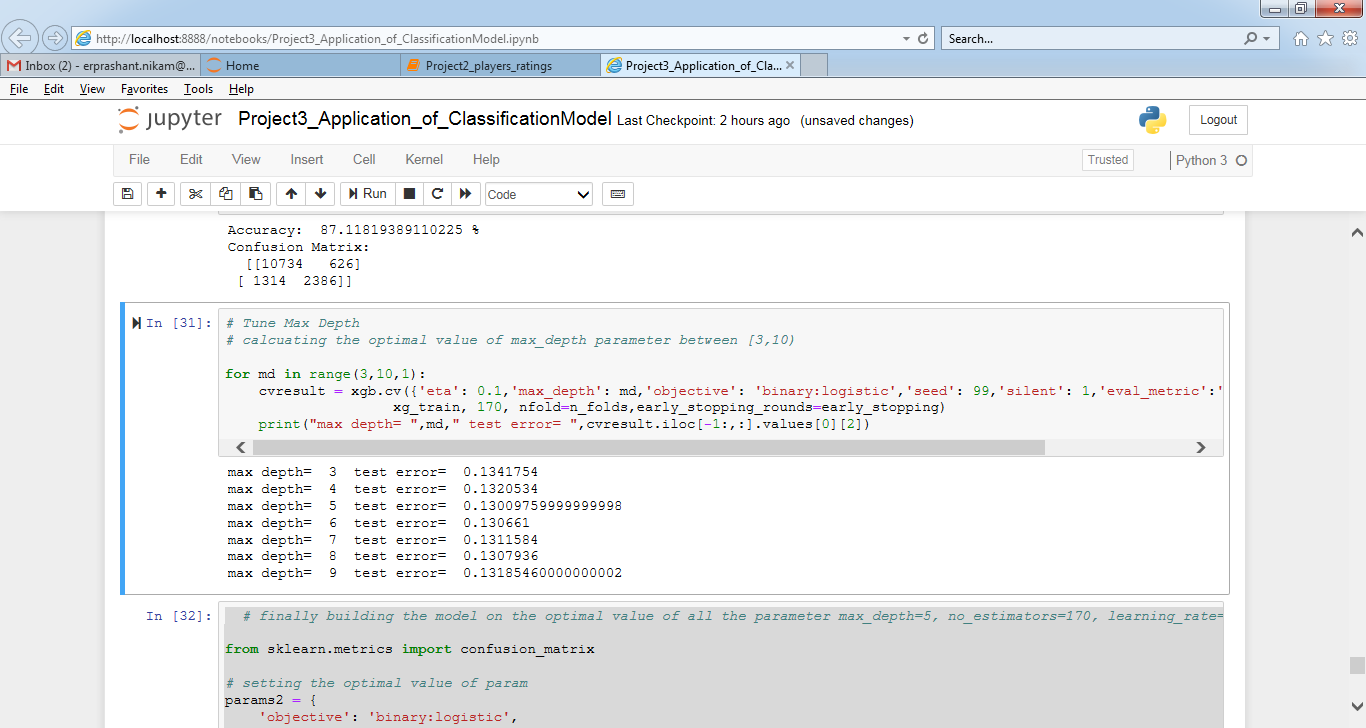
# calcuating the optimal value of max\_depth parameter between [3,10)

for md in range(3,10,1):

cvresult = xgb.cv({'eta': 0.1,'max\_depth': md,'objective': 'binary:logistic','seed': 99,'silent': 1,'eval\_metric':'error', 'nthread':4},

xg\_train, 170, nfold=n\_folds,early\_stopping\_rounds=early\_stopping)

print("max depth= ",md," test error= ",cvresult.iloc[-1:,:].values[0][2])



# finally building the model on the optimal value of all the parameter max\_depth=5, no\_estimators=170, learning\_rate= 0.1

from sklearn.metrics import confusion\_matrix

# setting the optimal value of param

params2 = {

'objective': 'binary:logistic',

'max\_depth': 5,

'learning\_rate': 0.1,

'silent': 1.0,

'n\_estimators':170

}

# training the model on the training data

classifier2 = XGBClassifier(\*\*params2).fit(x\_train, y\_train)

# predicting the value on the test set

y\_predict = classifier2.predict(x\_test)

# calculating the accuarcy score and confusion matrix

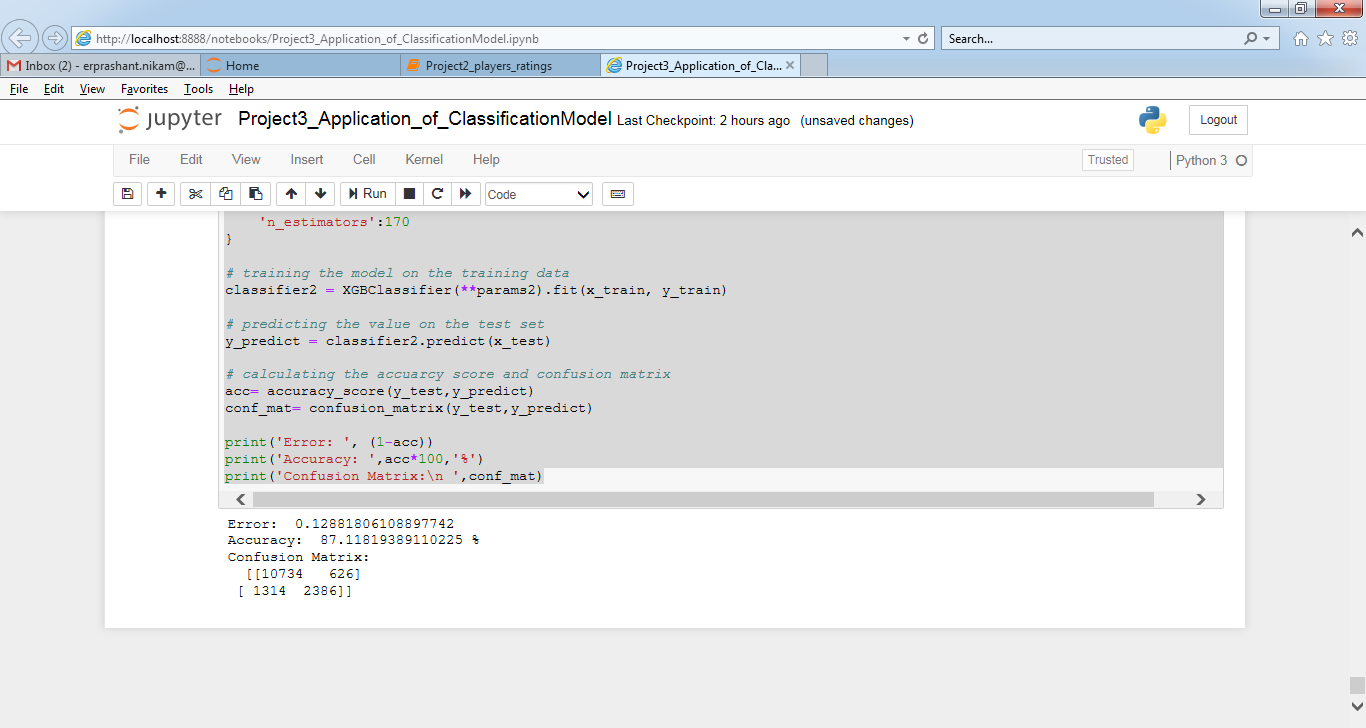
acc= accuracy\_score(y\_test,y\_predict)

conf\_mat= confusion\_matrix(y\_test,y\_predict)

print('Error: ', (1-acc))

print('Accuracy: ',acc\*100,'%')

print('Confusion Matrix:\n ',conf\_mat)



# Using Random forest algorithms to find the factors importance

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_estimators=1000,random\_state=1, max\_depth=10)

model.fit(x\_train,y\_train)

features = x\_train.columns

importances = model.feature\_importances\_

indices = np.argsort(importances)[-9:] # top 10 features

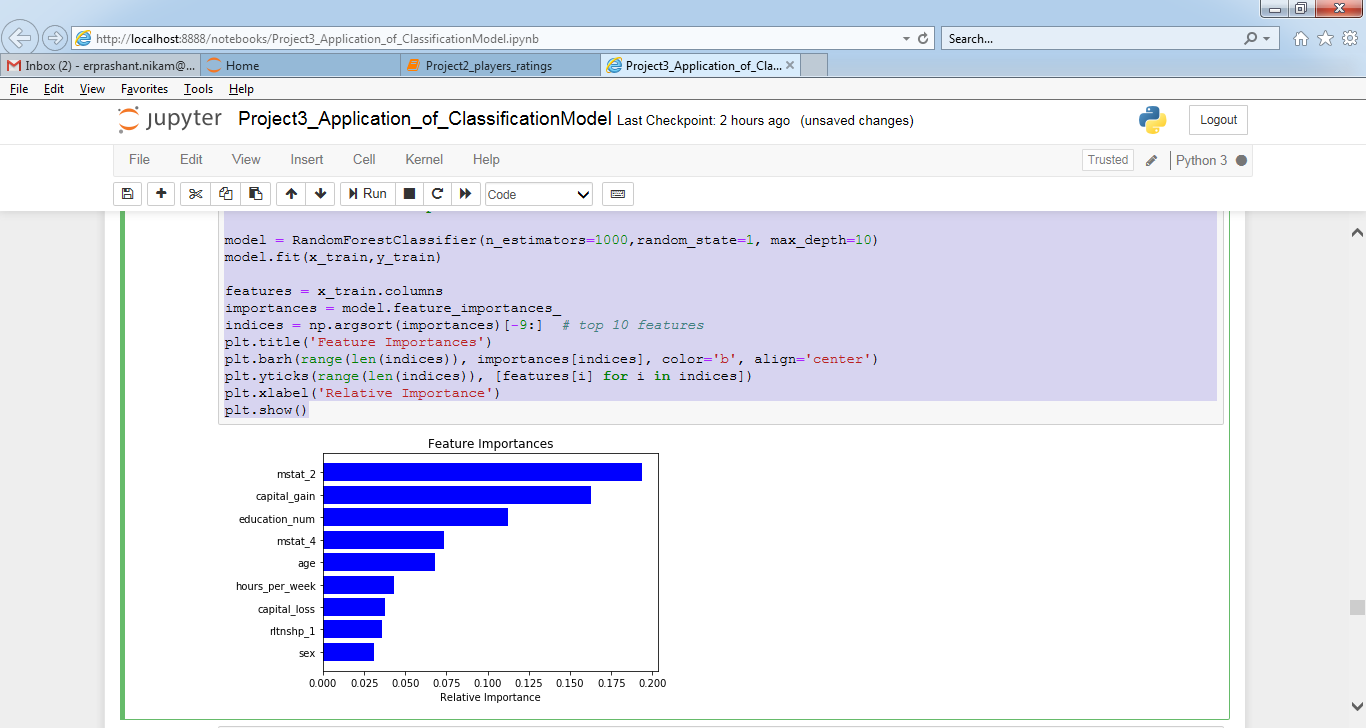
plt.title('Feature Importances')

plt.barh(range(len(indices)), importances[indices], color='b', align='center')

plt.yticks(range(len(indices)), [features[i] for i in indices])

plt.xlabel('Relative Importance')

plt.show()



#Important factors based on accuracy

#['mstat\_2', 'capital\_gain', 'education\_num', 'mstat\_4', 'age', 'hours\_per\_week', 'capital\_loss', 'rltnshp\_1', 'sex']

# saving 9 important columns in a list

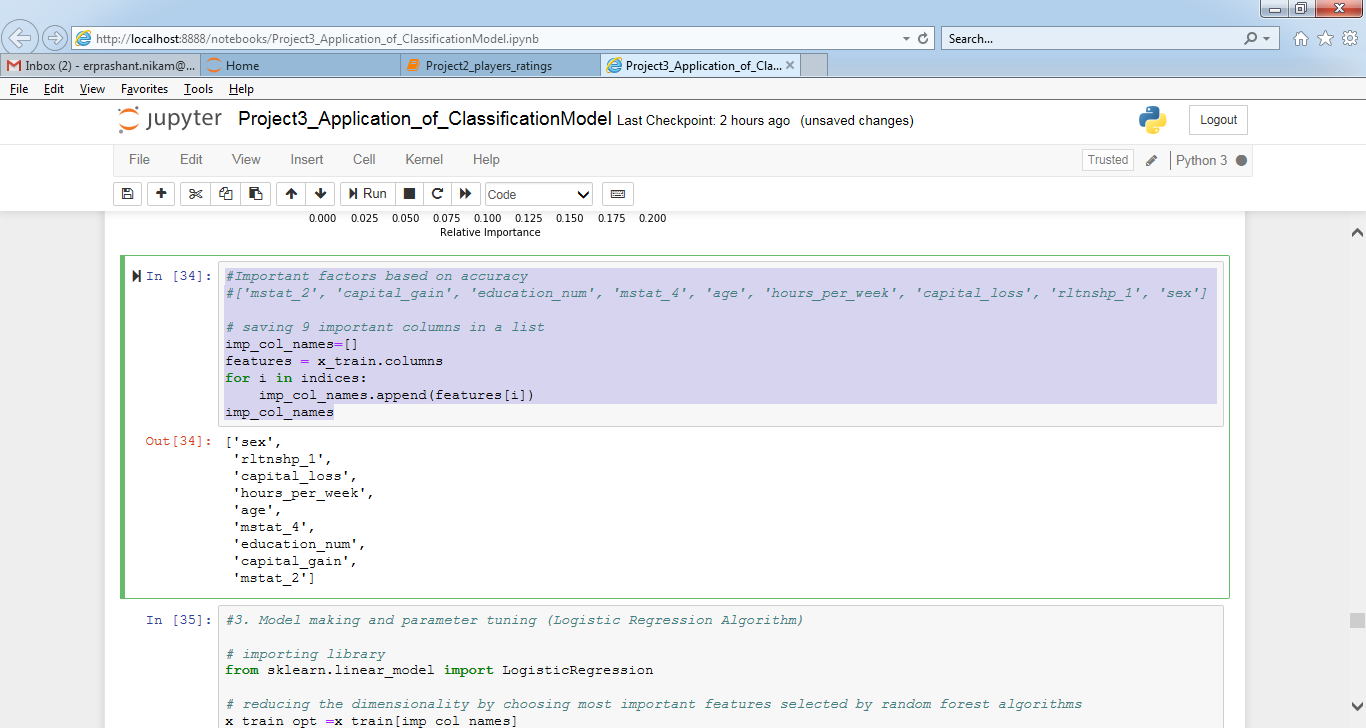
imp\_col\_names=[]

features = x\_train.columns

for i in indices:

imp\_col\_names.append(features[i])

imp\_col\_names



#3. Model making and parameter tuning (Logistic Regression Algorithm)

# importing library

from sklearn.linear\_model import LogisticRegression

# reducing the dimensionality by choosing most important features selected by random forest algorithms

x\_train\_opt =x\_train[imp\_col\_names]

x\_test\_opt =x\_test[imp\_col\_names]

# training model and prediction

regressor= LogisticRegression(random\_state=40)

regressor.fit(x\_train\_opt,y\_train)

y\_pred=regressor.predict(x\_test\_opt)

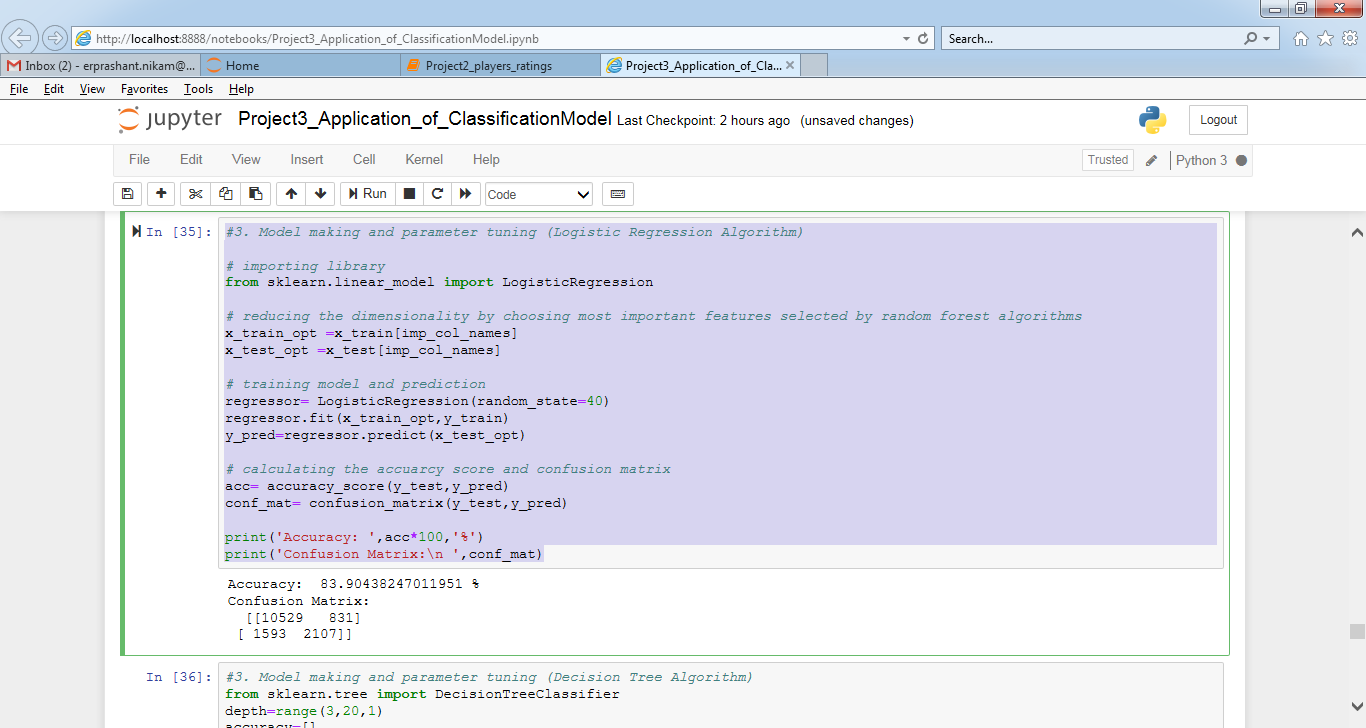
# calculating the accuarcy score and confusion matrix

acc= accuracy\_score(y\_test,y\_pred)

conf\_mat= confusion\_matrix(y\_test,y\_pred)

print('Accuracy: ',acc\*100,'%')

print('Confusion Matrix:\n ',conf\_mat)



#3. Model making and parameter tuning (Decision Tree Algorithm)

from sklearn.tree import DecisionTreeClassifier

depth=range(3,20,1)

accuracy=[]

# training model and prediction

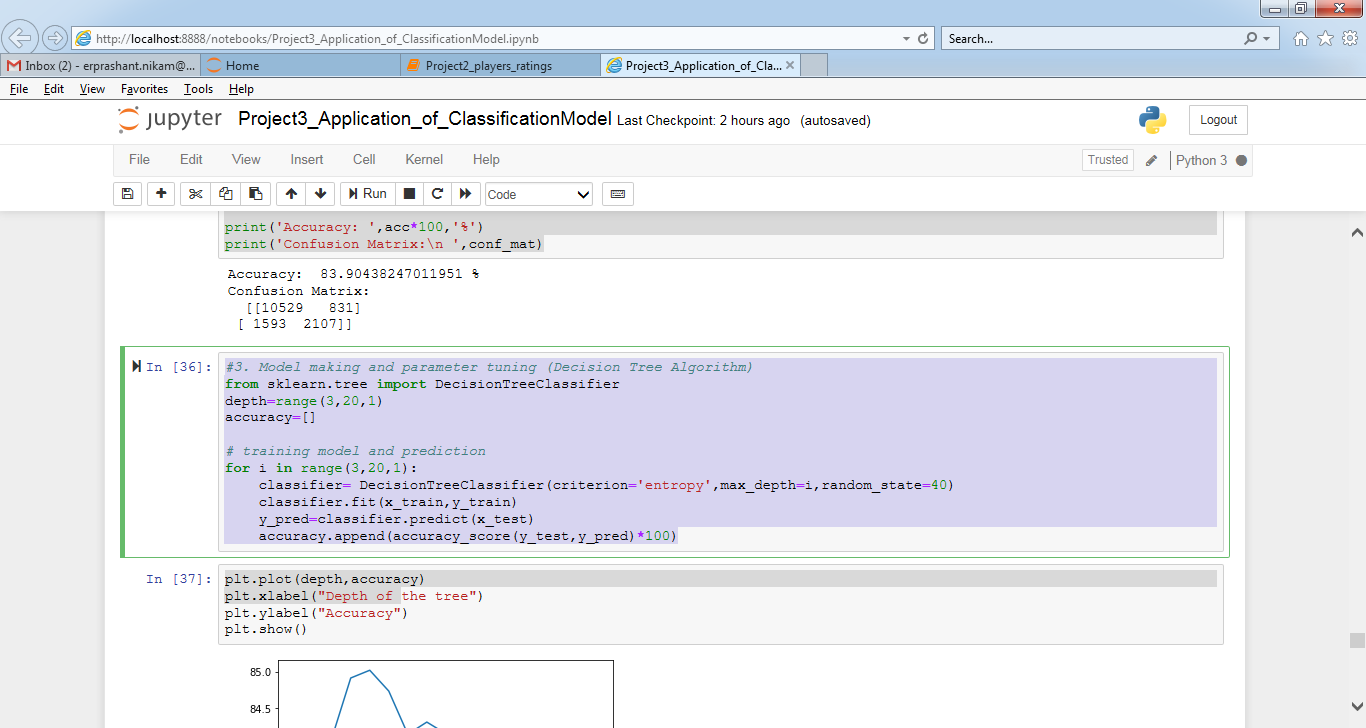
for i in range(3,20,1):

classifier= DecisionTreeClassifier(criterion='entropy',max\_depth=i,random\_state=40)

classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test)

accuracy.append(accuracy\_score(y\_test,y\_pred)\*100)

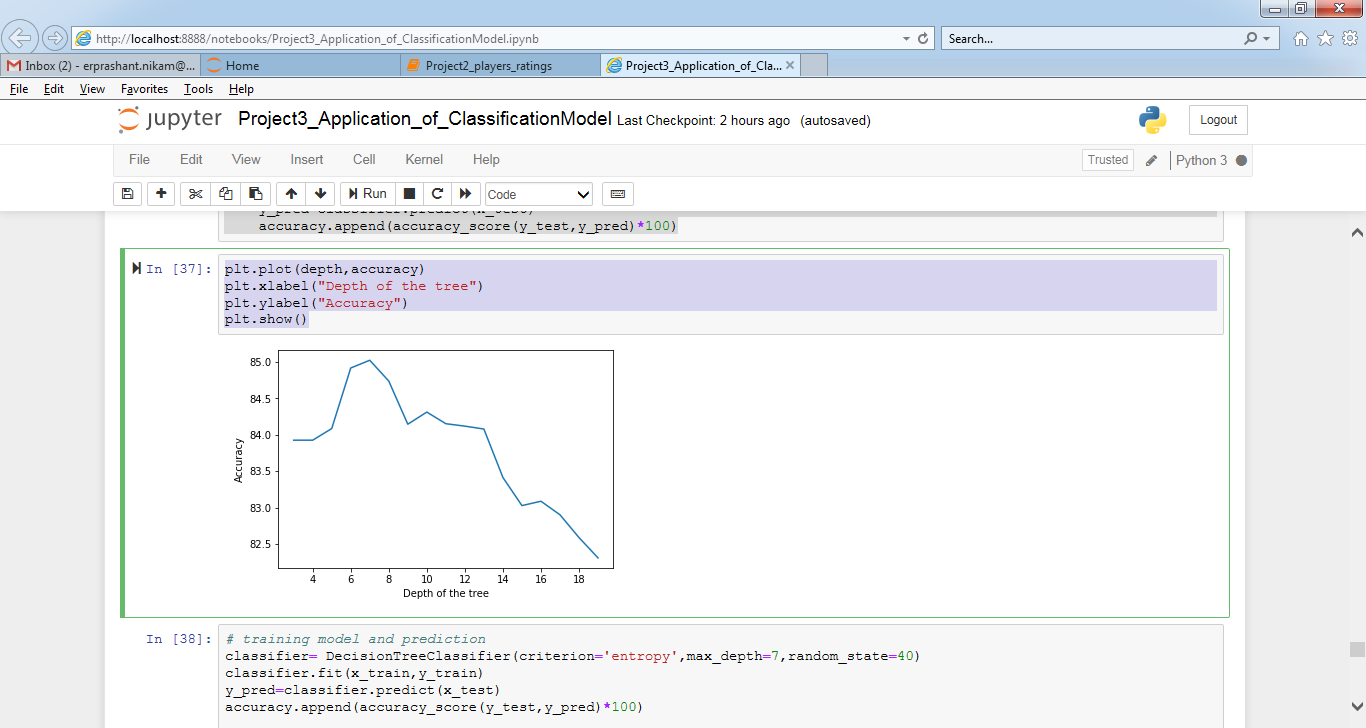


plt.plot(depth,accuracy)

plt.xlabel("Depth of the tree")

plt.ylabel("Accuracy")

plt.show()



# training model and prediction

classifier= DecisionTreeClassifier(criterion='entropy',max\_depth=7,random\_state=40)

classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test)

accuracy.append(accuracy\_score(y\_test,y\_pred)\*100)

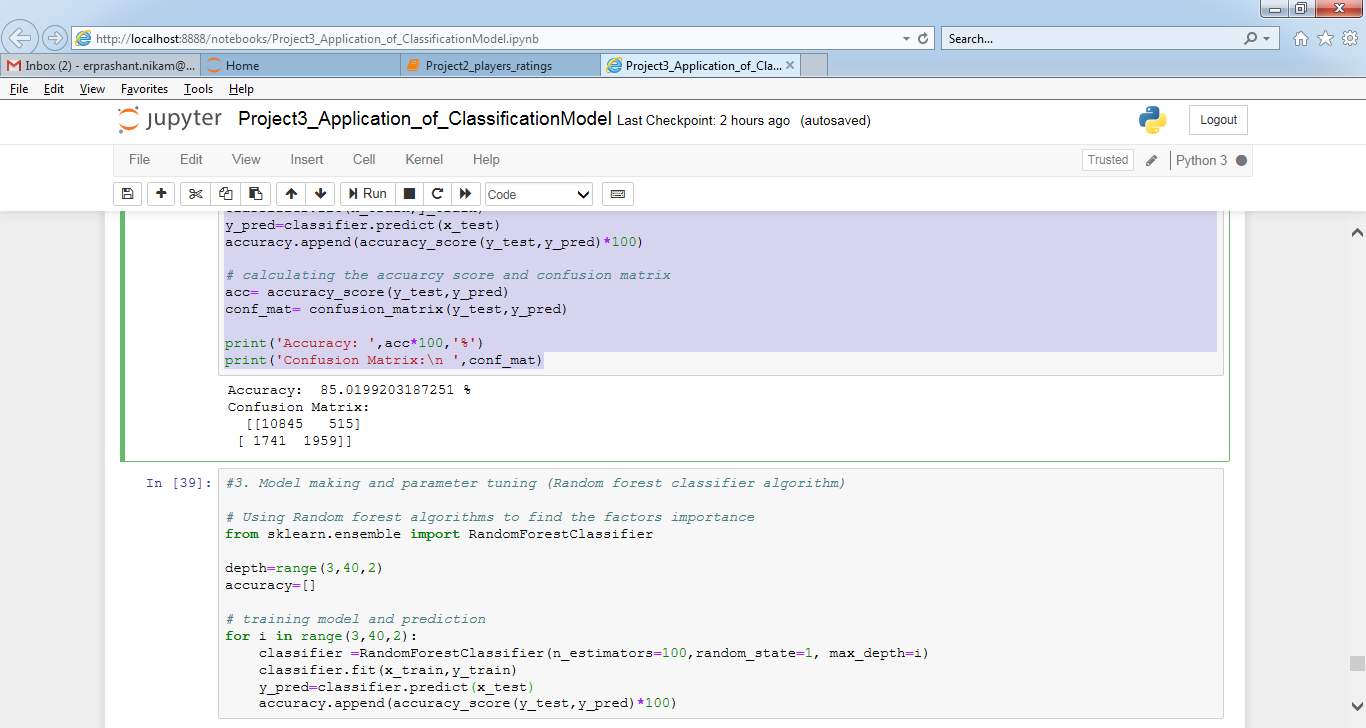
# calculating the accuarcy score and confusion matrix

acc= accuracy\_score(y\_test,y\_pred)

conf\_mat= confusion\_matrix(y\_test,y\_pred)

print('Accuracy: ',acc\*100,'%')

print('Confusion Matrix:\n ',conf\_mat)



#3. Model making and parameter tuning (Random forest classifier algorithm)

# Using Random forest algorithms to find the factors importance

from sklearn.ensemble import RandomForestClassifier

depth=range(3,40,2)

accuracy=[]

# training model and prediction

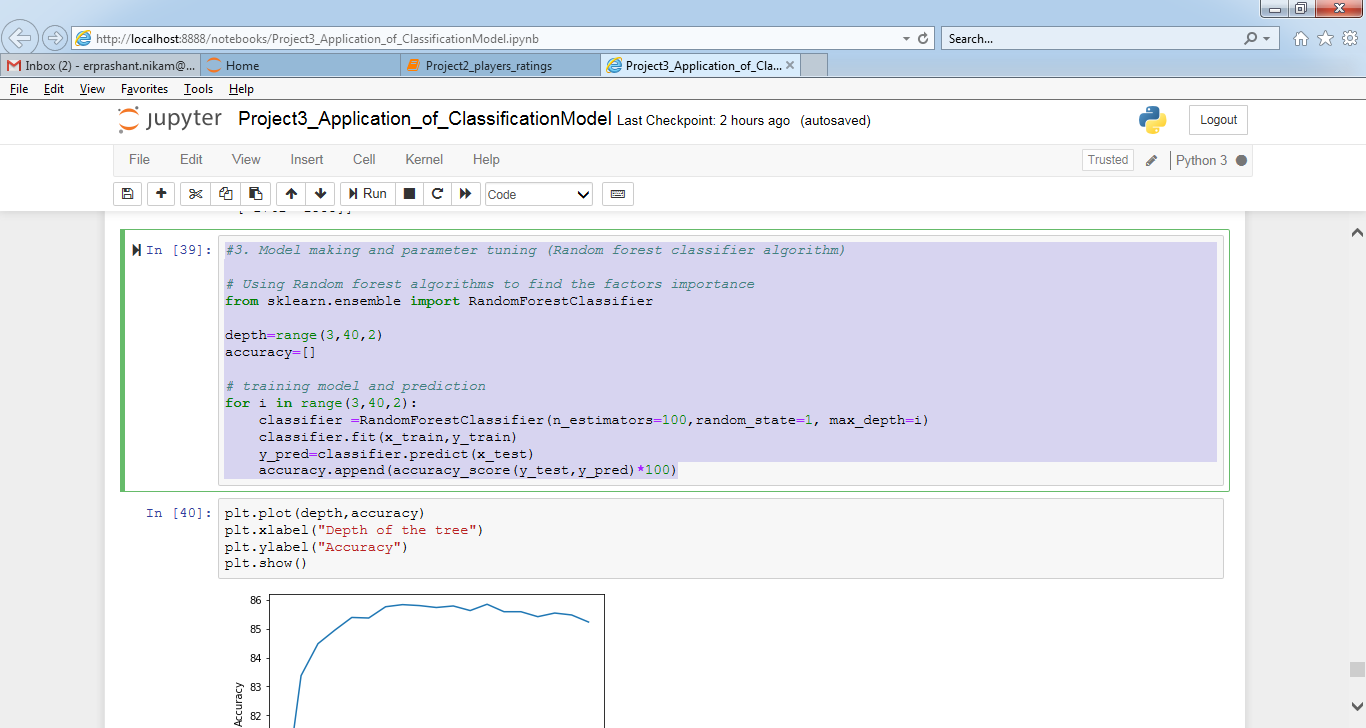
for i in range(3,40,2):

classifier =RandomForestClassifier(n\_estimators=100,random\_state=1, max\_depth=i)

classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test)

accuracy.append(accuracy\_score(y\_test,y\_pred)\*100)

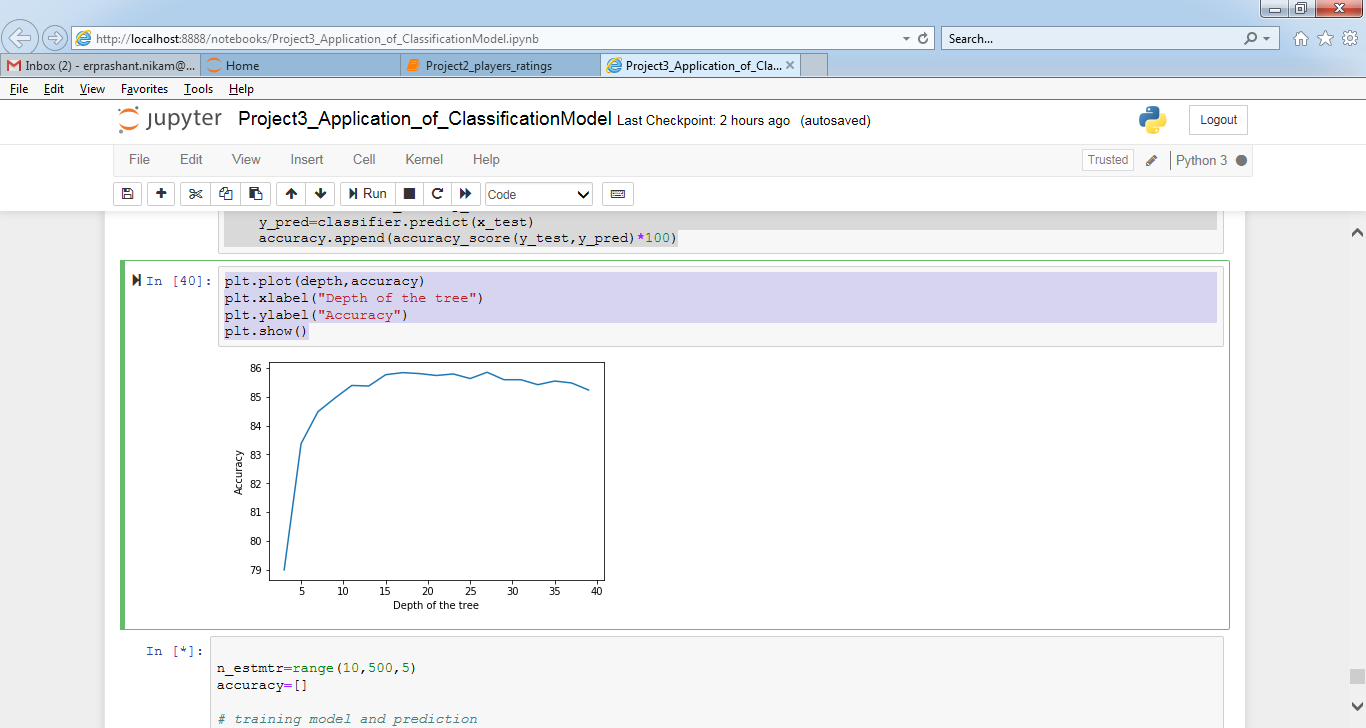


plt.plot(depth,accuracy)

plt.xlabel("Depth of the tree")

plt.ylabel("Accuracy")

plt.show()



n\_estmtr=range(10,500,5)

accuracy=[]

# training model and prediction

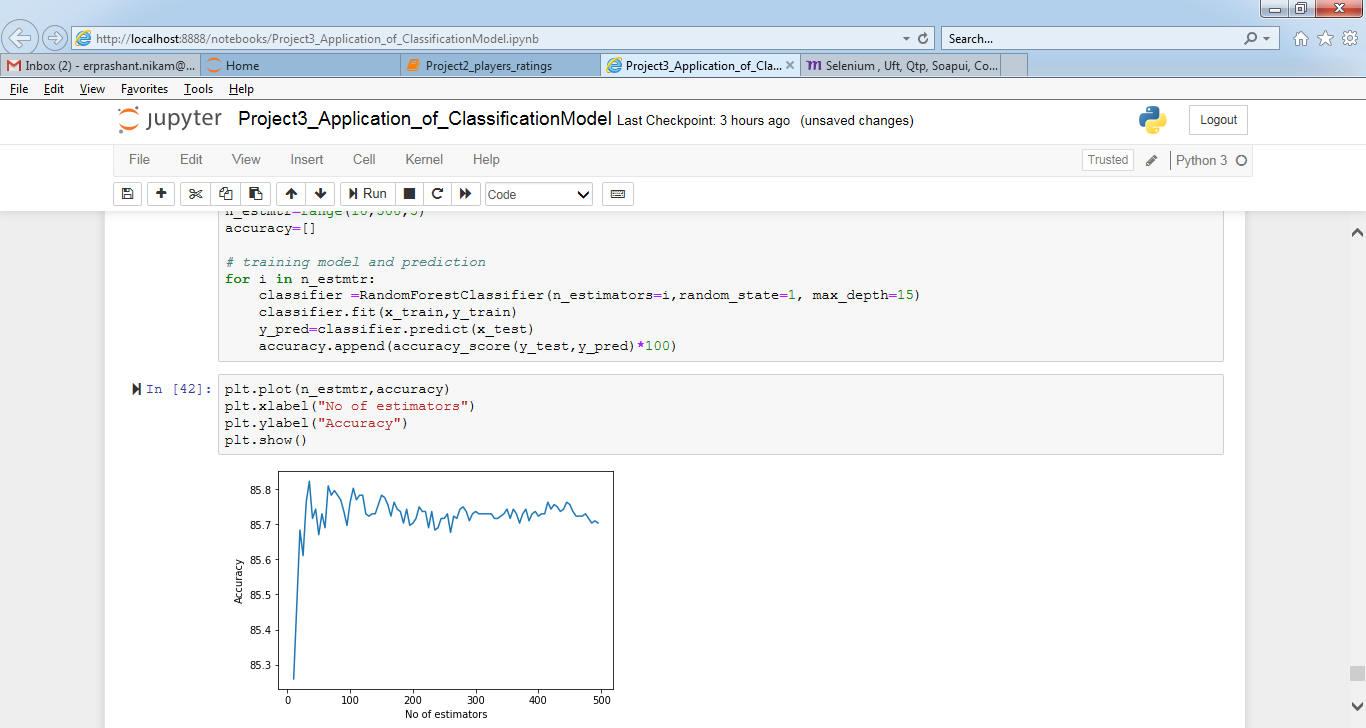
for i in n\_estmtr:

classifier =RandomForestClassifier(n\_estimators=i,random\_state=1, max\_depth=15)

classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test)

accuracy.append(accuracy\_score(y\_test,y\_pred)\*100)



plt.plot(n\_estmtr,accuracy)

plt.xlabel("No of estimators")

plt.ylabel("Accuracy")

plt.show()

# model is best for no of estimators=35

# training model and prediction

classifier =RandomForestClassifier(n\_estimators=35,random\_state=1, max\_depth=15)

classifier.fit(x\_train,y\_train)

y\_pred=classifier.predict(x\_test)

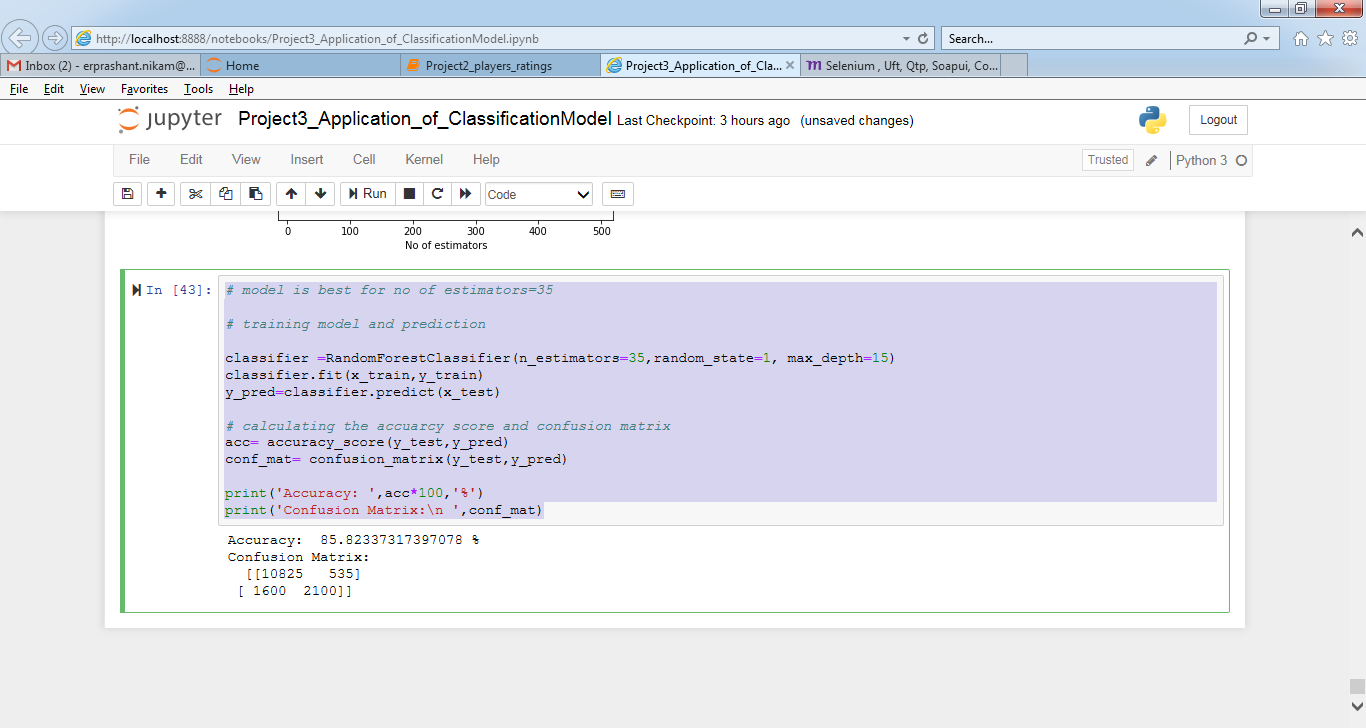
# calculating the accuarcy score and confusion matrix

acc= accuracy\_score(y\_test,y\_pred)

conf\_mat= confusion\_matrix(y\_test,y\_pred)

print('Accuracy: ',acc\*100,'%')

print('Confusion Matrix:\n ',conf\_mat)



Conclusion

Xgboost algorithm is the best algorithm which gives the best prediction for the above data.

We have used the following algorithms for classifications and their accuracy are:-

1. Logistic regression - 83.90438247011951 %

2. Decision tree classifier - 85.0199203187251 %

3. Random Forest classifier - 85.82337317397078 %

4. Xgboost - 87.11819389110225 %