Name - Mohit Mohit

BU id - U33745431

CS 542

Programming Assignment Report

1. **Linear Regression: -**

Please refer ‘LinearRegression.py’ file for this task.

**Keep the dataset ‘detroit.mat’ and python script ‘LinearRegression.py’ in the same directory before running the code.**

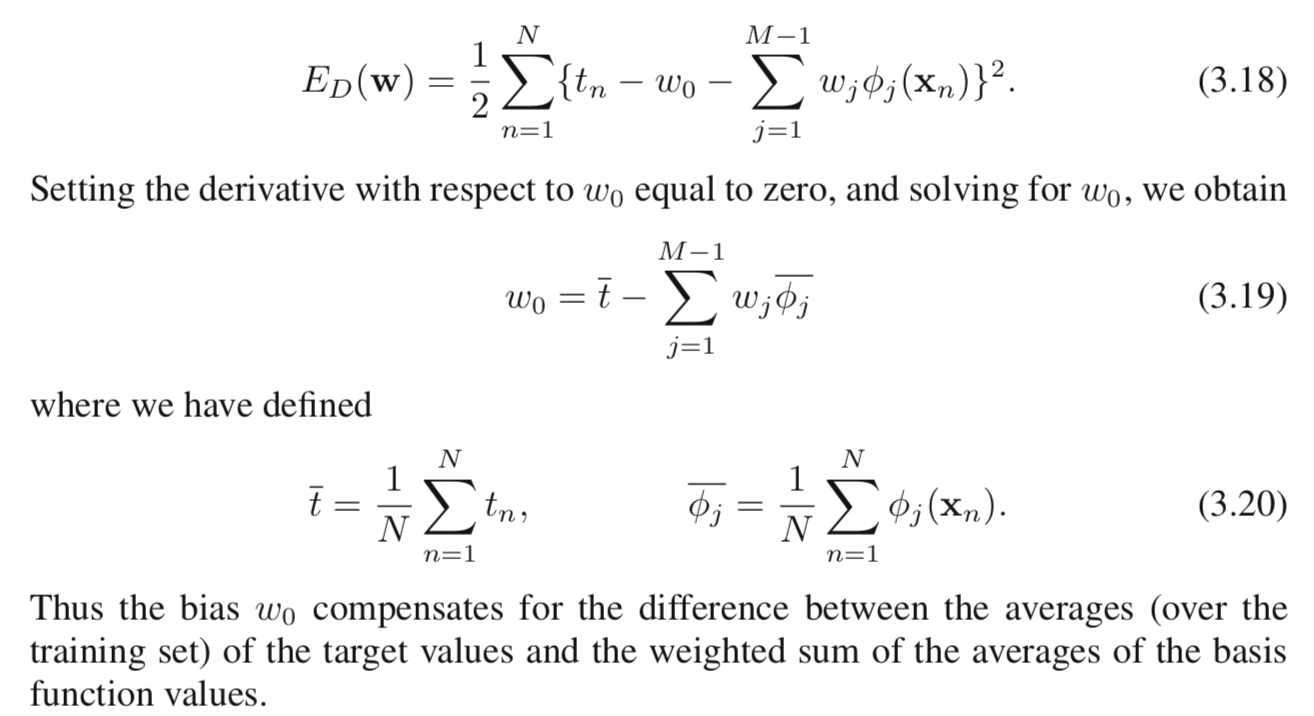
**Task**: - To find the 3rd best Predictor.

**Solution**: -

I have decided to use the error function which is the sum of squared difference between each observation. I will take 3 predictors at a time such as ‘FTP’, ‘WE’ and anyone from the rest attributes one by one.

I choose this approach because we need to find the 3rd best predictors so we need a set of predictors which will give us the least error as the minimum error means the strong predictor. Hence, I Will choose the attribute which gives the least error as the 3rd predictor**.**

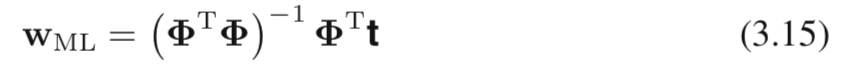
The error function according the book: -



For the given problem: - tn(target) is – Attribute ‘HOM’

Phi(Xn) – Predictors

Weight is given by : -



Will use this formula to find weight.

For Normalization I have used the below z-score normalization formula.



I am normalizing the data as the dataset is small and range of its attributes is varying a lot. So, to keep the data in a common scale, I am using Z-score normalization method.

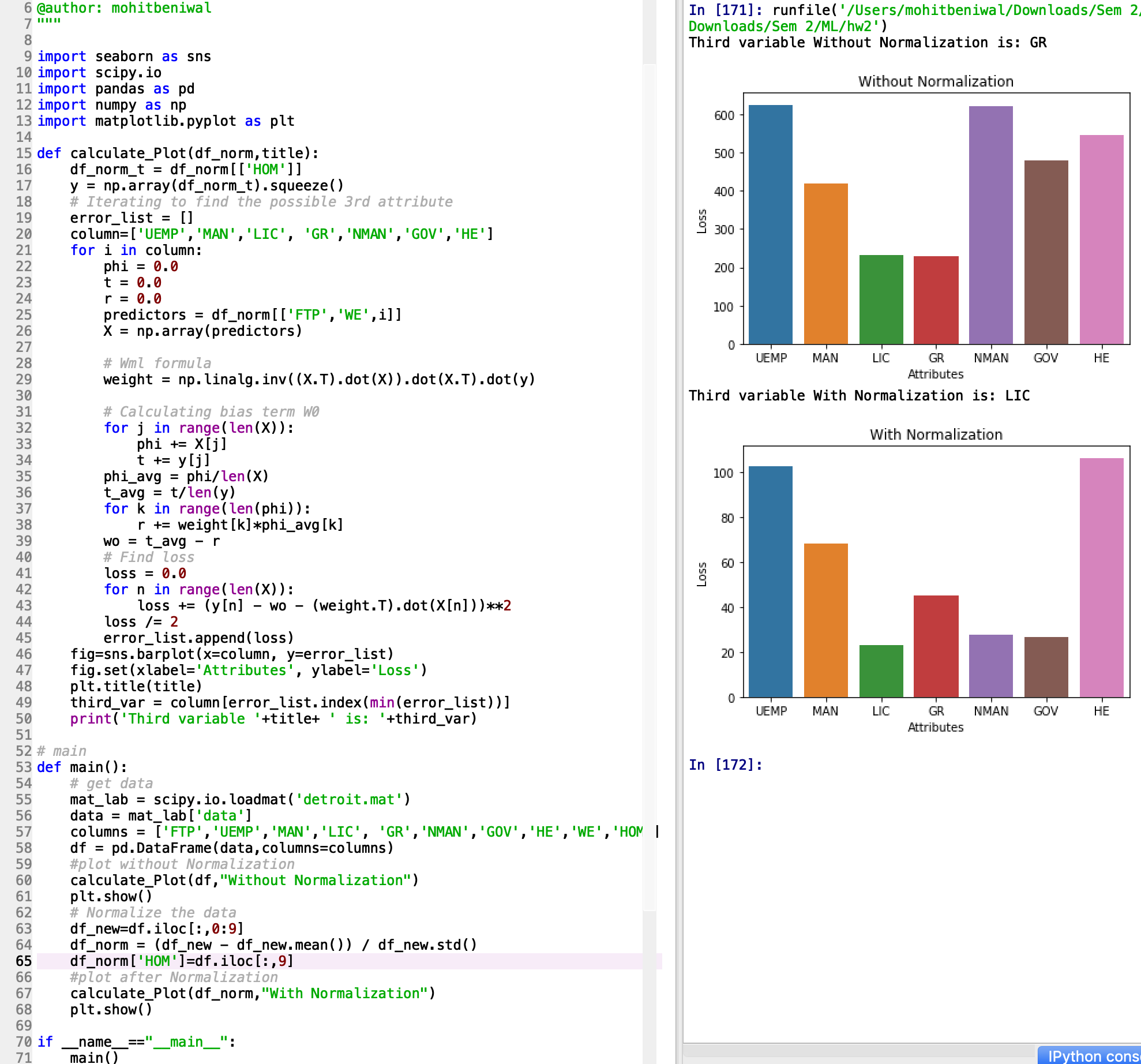
**Conclusion**: -

I have tried the above approach for both Normalized data and without Normalization.

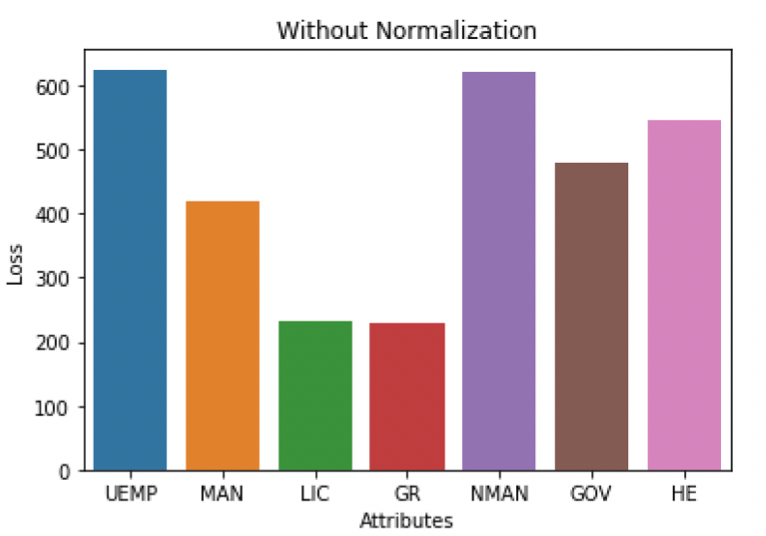
Got ‘GR’ as the 3rd and ‘LIC’ as the 4th predictor without Normalization.

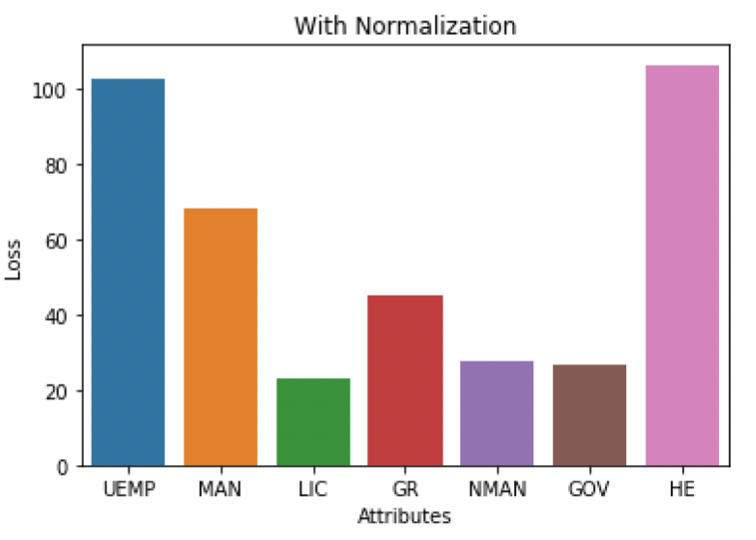
And ‘LIC’ as the 3rd and ‘GR’ as the 4th predictor after normalization as shown in the screenshot.

**Code screenshot: -**



**Error plots: -**





**Code: -**

﻿#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Sat Mar 2 21:41:36 2019

@author: mohitbeniwal

"""

import seaborn as sns

import scipy.io

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

def calculate\_Plot(df\_norm,title):

df\_norm\_t = df\_norm[['HOM']]

y = np.array(df\_norm\_t).squeeze()

# Iterating to find the possible 3rd attribute

error\_list = []

column=['UEMP','MAN','LIC', 'GR','NMAN','GOV','HE']

for i in column:

phi = 0.0

t = 0.0

r = 0.0

predictors = df\_norm[['FTP','WE',i]]

X = np.array(predictors)

# Wml formula

weight = np.linalg.inv((X.T).dot(X)).dot(X.T).dot(y)

# Calculating bias term W0

for j in range(len(X)):

phi += X[j]

t += y[j]

phi\_avg = phi/len(X)

t\_avg = t/len(y)

for k in range(len(phi)):

r += weight[k]\*phi\_avg[k]

wo = t\_avg - r

# Find loss

loss = 0.0

for n in range(len(X)):

loss += (y[n] - wo - (weight.T).dot(X[n]))\*\*2

loss /= 2

error\_list.append(loss)

fig=sns.barplot(x=column, y=error\_list)

fig.set(xlabel='Attributes', ylabel='Loss')

plt.title(title)

third\_var = column[error\_list.index(min(error\_list))]

print('Third variable '+title+ ' is: '+third\_var)

# main

def main():

# get data

mat\_lab = scipy.io.loadmat('detroit.mat')

data = mat\_lab['data']

columns = ['FTP','UEMP','MAN','LIC', 'GR','NMAN','GOV','HE','WE','HOM']

df = pd.DataFrame(data,columns=columns)

#plot without Normalization

calculate\_Plot(df,"Without Normalization")

plt.show()

# Normalize the data

df\_new=df.iloc[:,0:9]

df\_norm = (df\_new - df\_new.mean()) / df\_new.std()

df\_norm['HOM']=df.iloc[:,9]

#plot after Normalization

calculate\_Plot(df\_norm,"With Normalization")

plt.show()

if \_\_name\_\_=="\_\_main\_\_":

main()

1. **Nearest Neighbor: -**

**Keep all datasets and python scripts in same directory before running the code.**

**Task** 1: -

Process the data and generate processed data files.

**Solution**:- For this task please refer ‘process.py’ file.

**Steps**:-

1. I am creating a Pandas dataframe using ‘﻿crx.data.training’ and ‘﻿crx.data.testing’ files.
2. Replacing all “?” in the dataframe with NaN, so that later on we can easily fill NaN with mode/median. Also converting the datatype to numeric for the numeric attributes as the source data is string.
3. Then I am looking for the nominal attributes in the dataframe and filling NaN with the mode of that column.
4. To fill numeric attributes, I am separating the dataframe into two dataframes according to the class attribute. And then I am filling the NaN values of numeric attributes with the mean of the attribute as I have already separated the dataframe according to the class, so I don’t need to take care of the class again.
5. Then I am normalizing the numeric attributes as the range of the attributes vary a lot and to keep the values in a common scale we are using the z-score normalization formula. As below



1. Then I am writing the processed dataframes to the ‘﻿crx.training.processed.csv’ and ‘﻿crx.testing.processed.csv’ file which will get generated in the same directory.

**Code: -**

"""

Created on Sun Mar 3 12:47:52 2019

@author: mohitbeniwal

"""

import pandas as pd

import numpy as np

import traceback

# method to get dataFrame

def get\_df(data):

df=[]

for i in data:

i=i[:-1]

j=i.split(',')

df.append(j)

df=pd.DataFrame(df,columns=['A1','A2','A3','A4','A5','A6','A7','A8','A9','A10','A11','A12','A13','A14','A15','A16'])

#replace "?" with NaN

df.replace('?', np.NaN,inplace=True)

df = df.apply(pd.to\_numeric, errors='ignore')

return df

# this method replaces non-numeric columns with mode value

def replace\_with\_mode(df):

try:

# get categorical columns

cat\_c = df.columns[df.dtypes.apply(lambda c: np.issubdtype(c, np.object\_))]

for col in cat\_c:

df[col].fillna(df[col].mode()[0], inplace=True)

except Exception as e:

print(e)

traceback.print\_exc()

return df

# Function to normalize data

def normalize\_colmns(df):

try:

# columns with numerical data

num\_c = df.\_get\_numeric\_data().columns

for col in num\_c:

#col\_zscore = col + '\_zscore'

df[col] = (df[col] - df[col].mean())/df[col].std()

except Exception as e:

print(e)

traceback.print\_exc()

return df

def main():

# Read files

crx\_train= open('crx.data.training', 'r')

crx\_data\_train=crx\_train.readlines()

crx\_test= open('crx.data.testing', 'r')

crx\_data\_test=crx\_test.readlines()

# train data

df\_train=get\_df(crx\_data\_train)

# test data

df\_test=get\_df(crx\_data\_test)

df\_train=replace\_with\_mode(df\_train)

df\_test=replace\_with\_mode(df\_test)

# separate +ve and \_ve data

df\_pos\_train=df\_train.loc[df\_train.iloc[:,15]=='+'].reset\_index(drop=True)

df\_neg\_train=df\_train.loc[df\_train.iloc[:,15]=='-'].reset\_index(drop=True)

df\_pos\_test=df\_test.loc[df\_test.iloc[:,15]=='+'].reset\_index(drop=True)

df\_neg\_test=df\_test.loc[df\_test.iloc[:,15]=='-'].reset\_index(drop=True)

# replace numerical columns with column mean

df\_pos\_train.fillna(df\_pos\_train.mean(),inplace=True)

df\_neg\_train.fillna(df\_neg\_train.mean(),inplace=True)

df\_pos\_test.fillna(df\_pos\_train.mean(),inplace=True)

df\_neg\_test.fillna(df\_neg\_train.mean(),inplace=True)

# Normalize the data

df\_train\_norm = normalize\_colmns(df\_pos\_train.append(df\_neg\_train))

df\_train\_norm.index = range(len(df\_train\_norm.index)) # resetting index

df\_test\_norm = normalize\_colmns(df\_pos\_test.append(df\_neg\_test))

df\_test\_norm.index = range(len(df\_test\_norm.index))

df\_train\_norm.to\_csv('crx.training.processed.csv',encoding='utf-8-sig')

df\_test\_norm.to\_csv('crx.testing.processed.csv',encoding='utf-8-sig')

if \_\_name\_\_=="\_\_main\_\_":

main()

**Task 2 & 3: -** To perform KNN and create a accuracy table with atleast 2 values of k.

Please refer ‘run.py’ file of this task.

**Steps: -**

1. The script will take the data from ‘﻿crx.training.processed.csv’ and ‘﻿crx.testing.processed.csv’ files, which is being generated by ‘process.py’.
2. Then I am taking ‘a’ as a list of k ranging from (1,18) with a step of 4.
3. And then for each value of k I am calling ‘knn’ method which will give the prediction for the given value of k.



This method will implement the above Euclidean distance formula to calculate the distance for numerical attributes and for nominal attributes if they are not equal to each other then I am adding 1 to the distance.

After calculating the distance with all data points, I am taking K least distances and assigning the class to the new data point according to the majority in the k selected data points.

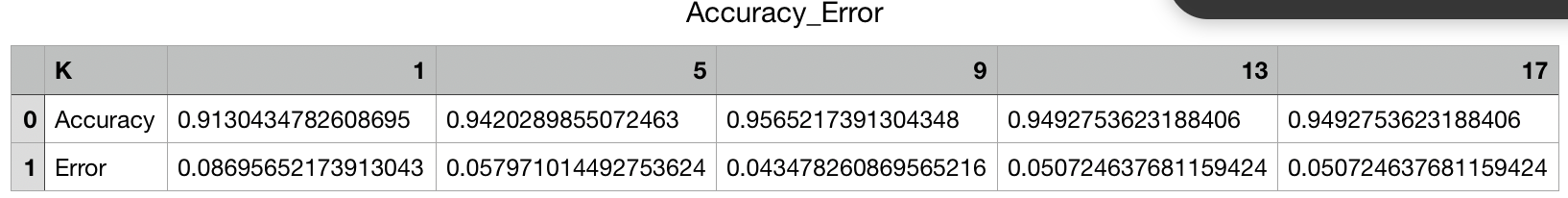
1. As now we have predicted values and actual values. So, I am calculating the accuracy and error as shown in the below screenshot.

**Conclusion**: -

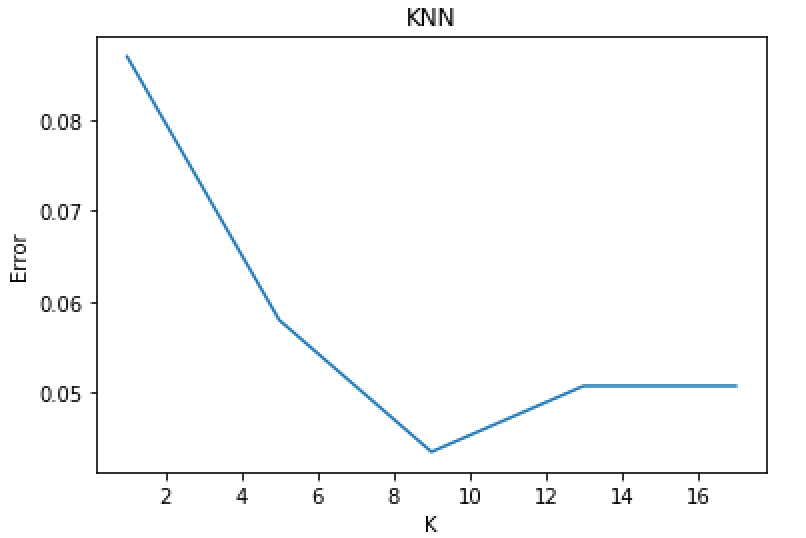
I have observed that if we keep on increasing the value of k, then after a point accuracy of the system come down.

As in our case I have taken (1,5,9,13,17) as the value of k. Till 9 we can observe the growth in accuracy but after 9 for 13 and 17 the accuracy came down.

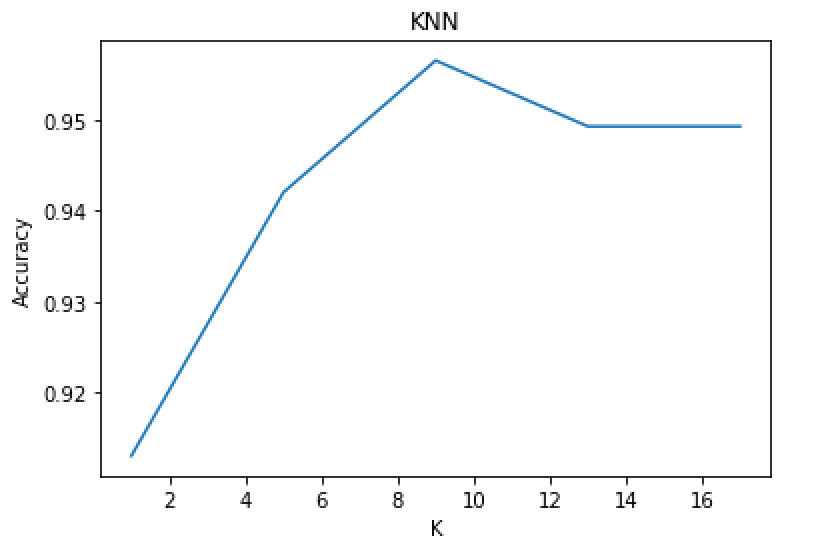
**Accuracy & Error table: -**

****

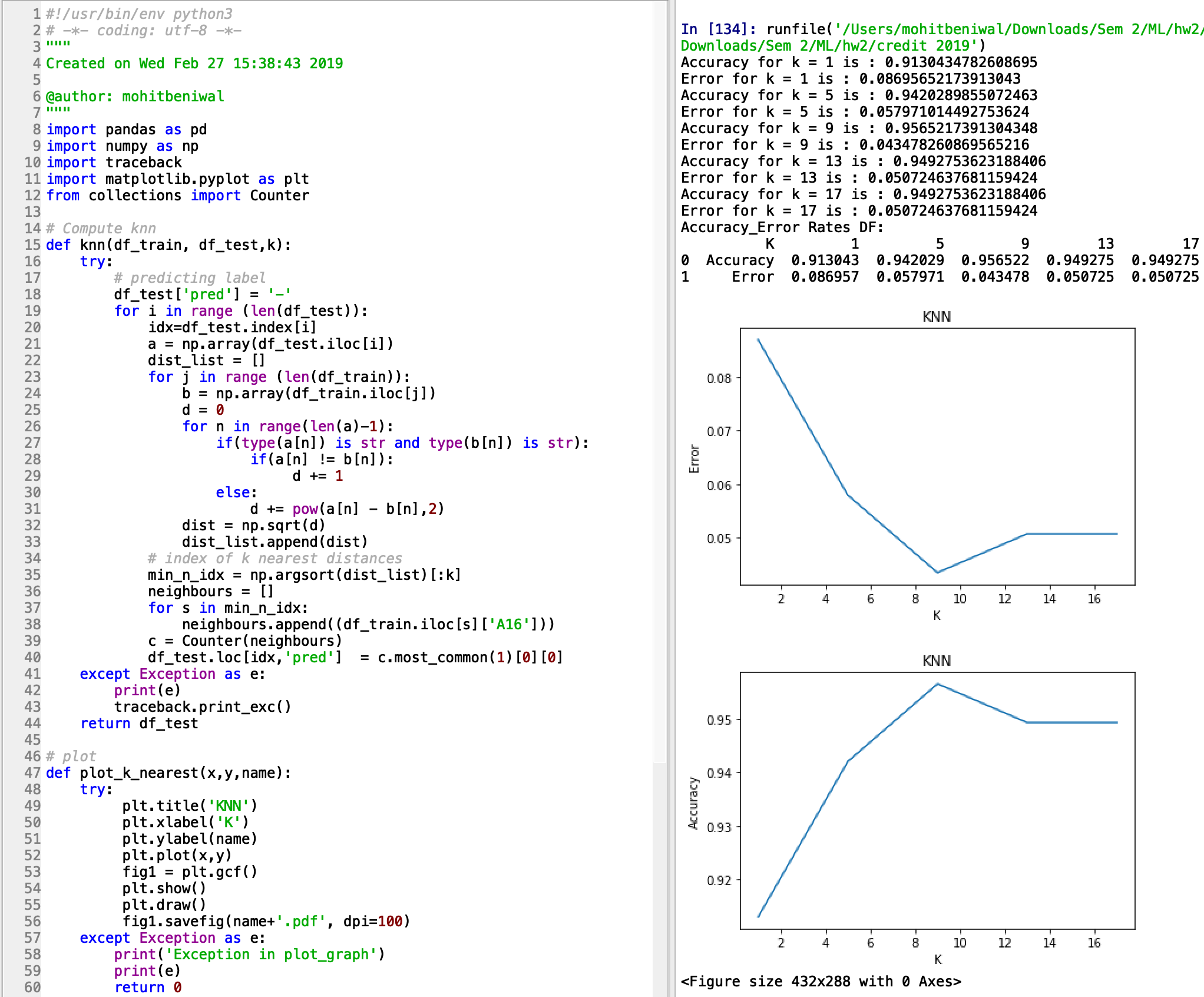
**Error plot: -**

****

**Accuracy Plot: -**

****

**Code screenshot: -**





**Code:-**

﻿#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Feb 27 15:38:43 2019

@author: mohitbeniwal

"""

import pandas as pd

import numpy as np

import traceback

import matplotlib.pyplot as plt

from collections import Counter

# Compute knn

def knn(df\_train, df\_test,k):

try:

# predicting label

df\_test['pred'] = '-'

for i in range (len(df\_test)):

idx=df\_test.index[i]

a = np.array(df\_test.iloc[i])

dist\_list = []

for j in range (len(df\_train)):

b = np.array(df\_train.iloc[j])

d = 0

for n in range(len(a)-1):

if(type(a[n]) is str and type(b[n]) is str):

if(a[n] != b[n]):

d += 1

else:

d += pow(a[n] - b[n],2)

dist = np.sqrt(d)

dist\_list.append(dist)

# index of k nearest distances

min\_n\_idx = np.argsort(dist\_list)[:k]

neighbours = []

for s in min\_n\_idx:

neighbours.append((df\_train.iloc[s]['A16']))

c = Counter(neighbours)

df\_test.loc[idx,'pred'] = c.most\_common(1)[0][0]

except Exception as e:

print(e)

traceback.print\_exc()

return df\_test

# plot

def plot\_k\_nearest(x,y,name):

try:

plt.title('KNN')

plt.xlabel('K')

plt.ylabel(name)

plt.plot(x,y)

fig1 = plt.gcf()

plt.show()

plt.draw()

fig1.savefig(name+'.pdf', dpi=100)

except Exception as e:

print('Exception in plot\_graph')

print(e)

return 0

def main():

# Read processed file

df\_train\_norm = pd.read\_csv("crx.training.processed.csv",index\_col=0)

df\_test\_norm = pd.read\_csv('crx.testing.processed.csv',index\_col=0)

x\_list = []

y\_list = []

z\_list = []

# compute knn

a=np.arange(1,18,4)

columnList=['K']

for k in a:

columnList.append(str(k))

x\_list.append(k)

df\_final\_test = knn(df\_train\_norm, df\_test\_norm,k)

error\_rate = np.sum(1\*(df\_final\_test['A16'] != df\_final\_test["pred"])) / len(df\_final\_test)

accuracy = np.sum(1\*(df\_final\_test['A16'] == df\_final\_test["pred"])) / len(df\_final\_test)

y\_list.append(error\_rate)

z\_list.append(accuracy)

print('Accuracy for k = '+str(k)+' is : '+str(accuracy))

print('Error for k = '+str(k)+' is : '+str(error\_rate))

z\_list.insert(0,'Accuracy')

y\_list.insert(0,'Error')

acc\_err\_list = [z\_list]+[y\_list]

acc\_err\_df = pd.DataFrame(acc\_err\_list,columns=columnList)

print('Accuracy\_Error Rates DF: \n'+str(acc\_err\_df))

acc\_err\_df.to\_csv('Accuracy\_Error.csv',encoding='utf-8-sig')

plot\_k\_nearest(x\_list,y\_list[1:],'Error')

plot\_k\_nearest(x\_list,z\_list[1:],'Accuracy')

if \_\_name\_\_=="\_\_main\_\_":

main()