Web Mining Lab Assignment 4

Name: Om Ashish Mishra

Registration Number: 16BCE0789

Slot: F2

The Question:

Write a python program to perform the following encoding for the ODD numbers between 1 – 30 i) Elias Gamma ii) Elias Delta iii) Golomb (b = 10)

The Answer:

**1. Urany Encoding:**

The Code:

for i in range(1,30,2):

print("1"\*i+"0")

The Output:

**2. Elias Gamma Encoding:**

The Code:

c=0

s=""

for i in range(1,30,2):

c=0

s=format(i,"b")

for j in range(len(s)):

c=c+1

s1 = "0"\*(c-1)+s

print(s1)

s1=0

The Output:

**3. Elias Delta Encoding:**

The Code:

import math

count=0

k1=0;

s2=""

s1=""

s=""

c=0

for i in range(1,30,2):

a = math.log2(i)

b = math.floor(a)

c = b+1

d = format(c,"b")

count=0

for j in range(len(d)):

count=count+1

s1 = "0"\*(count-1)+d

f = format(i,"b")

g = f[1:]

s3 = s1+g

print(s3+"\n")

The Output:

**4. Golumb Encoding:**

The Code:

from math import \*

n=[i for i in range(1,31,2)]

def unary(n):

if n==0:

return '0'

return ('0'\*n + '1')

def golumb(n,b):

q= n//b

r = n-q\*b

x1 = unary(q)

ubits = ceil(log(b,2))

bitlength= 2\*\*ubits - b

if(r>=bitlength):

return x1+bin(r + bitlength)[2:].zfill(ubits)

else:

return x1+bin(r )[2:].zfill(ubits-1)

for i in n:

print(golumb(i,10))

The Output:

**5. Variable Byte Coding:**

The Code:

for i in range(1,30,2):

a = format(i,"b")

c = len(a)

if c < 7:

c = "0"\*(7-c)+a

print(c+"0"+"\n")

else:

b = c[:-7]

print(b+"0"+"\n")

e = c[0:8]

print(e+"1\n")

The Output:

**Indexing**

from os import system, name

import re

def process\_files(filenames):

file\_to\_terms = {}

for file in filenames:

pattern = re.compile('[\W\_]+')

file\_to\_terms[file] = open(file, 'r').read().lower();

file\_to\_terms[file] = pattern.sub(' ',file\_to\_terms[file])

re.sub(r'[\W\_]+','', file\_to\_terms[file])

file\_to\_terms[file] = file\_to\_terms[file].split()

return file\_to\_terms

def index\_one\_file(termlist):

fileIndex = {}

for index, word in enumerate(termlist):

if word in fileIndex.keys():

fileIndex[word].append(index)

else:

fileIndex[word] = [index]

return fileIndex

def make\_indices(termlists):

total = {}

for filename in termlists.keys():

total[filename] = index\_one\_file(termlists[filename])

return total

def fullIndex(regdex):

total\_index = {}

for filename in regdex.keys():

for word in regdex[filename].keys():

if word in total\_index.keys():

if filename in total\_index[word].keys():

total\_index[word][filename].extend(regdex[filename][word][:])

else:

total\_index[word][filename] = regdex[filename][word]

else:

total\_index[word] = {filename: regdex[filename][word]}

return total\_index

def one\_word\_query(word, invertedIndex):

pattern = re.compile('[\W\_]+')

word = pattern.sub(' ',word)

if word in invertedIndex.keys():

return [filename for filename in invertedIndex[word].values()]

else:

return []

def free\_text\_query(string,index):

pattern = re.compile('[\W\_]+')

string = pattern.sub(' ',string)

result = []

for word in string.split():

result += one\_word\_query(word,index)

return list(set(result))

def phrase\_query(string, invertedIndex):

pattern = re.compile('[\W\_]+')

string = pattern.sub(' ',string)

listOfLists, result = [],[]

for word in string.split():

listOfLists.append(free\_text\_query(word,invertedIndex))

setted = set(listOfLists[0]).intersection(\*listOfLists)

for filename in setted:

temp = []

for word in string.split():

temp.append(invertedIndex[word][filename][:])

for i in range(len(temp)):

for ind in range(len(temp[i])):

temp[i][ind] -= i

if set(temp[0]).intersection(\*temp):

result.append(filename)

print('\n temp : \n')

print(temp)

return result

filenames=['document1.txt','document2.txt']

termslist=process\_files(filenames)

print('\nterm list \n')

print(termslist)

print('\n\n')

print('\n\n')

totaldict=make\_indices(termslist)

print('total dictionary \n')

print(totaldict)

print('\n\n')

print('\n\n')

index=fullIndex(totaldict)

print('full index \n')

print(index)

print('\n\n')

#one\_word\_query('exceptions', index)

#query\_word=free\_text\_query('exceptions',index)

#print(query\_word)

system('cls')

print('\n\n')

print('\n\n')

#r=phrase\_query('python has exceptions handling',index)

#print (r)

**Page Rank Algorithm:**

The Code:

import numpy as np

from scipy.sparse import csc\_matrix

def pageRank(G, s = .85, maxerr = .001):

n = G.shape[0]

# transform G into markov matrix M

M = csc\_matrix(G,dtype=np.float)

rsums = np.array(M.sum(1))[:,0]

ri, ci = M.nonzero()

M.data /= rsums[ri]

# bool array of sink states

sink = rsums==0

# Compute pagerank r until we converge

ro, r = np.zeros(n), np.ones(n)

while np.sum(np.abs(r-ro)) > maxerr:

ro = r.copy()

# calculate each pagerank at a time

for i in range(0,n):

# inlinks of state i

Ii = np.array(M[:,i].todense())[:,0]

# account for sink states

Si = sink / float(n)

# account for teleportation to state i

Ti = np.ones(n) / float(n)

r[i] = ro.dot( Ii\*s + Si\*s + Ti\*(1-s) )

# return normalized pagerank

return r/sum(r)

if \_\_name\_\_=='\_\_main\_\_':

# Example extracted from 'Introduction to Information Retrieval'

G = np.array([[0,0,1,0,0,0,0],

[0,1,1,0,0,0,0],

[1,0,1,1,0,0,0],

[0,0,0,1,1,0,0],

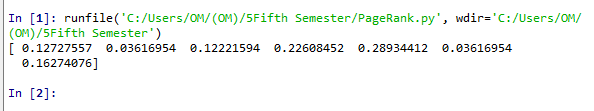
[0,0,0,0,0,0,1],

[0,0,0,0,0,1,1],

[0,0,0,1,1,0,1]])

print (pageRank(G,s=.86))

**The Output:**



Therefore Fifth one is the highest ranked; Second and Third are the smallest.

# The Method taught in lab:

**The Code:**

import numpy

x=int(input("Enter the number of nodes"))

it=int(input("Enter the number of iterations"))

d=input("Enter the damping factor")

matrix=numpy.array([input().split() for \_ in range(x)],float)

tox=(numpy.transpose(matrix))

dm=numpy.array([d for \_ in range(x)],float)

dm=(numpy.transpose(dm))

for i in range(it):

dm=numpy.dot(dm,tox)

dm.sort()

dm=dm[::-1]

print("Page ranking:")

print(\*dm,sep=' ')

**The output:**

runfile('C:/Users/OM/(OM)/5Fifth Semester/Kundan2.py', wdir='C:/Users/OM/(OM)/5Fifth Semester')

Enter the number of nodes6

Enter the number of iterations7

Enter the damping factor0.85

0 0 1 0 0 0

0.5 0 0.5 0 0 0

1 0 0 0 0 0

0 0 0 0 0.5 0.5

0 0 0.5 0.5 0 0

0 0 0 0.5 0.5 0

Page ranking:

0.85 0.85 0.85 0.85 0.85 0.85

StopWords

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

example\_sent = "This is a sample sentence, showing off the stop words filtration."

stop\_words = set(stopwords.words('english'))

word\_tokens = word\_tokenize(example\_sent)

filtered\_sentence = []

for w in word\_tokens:

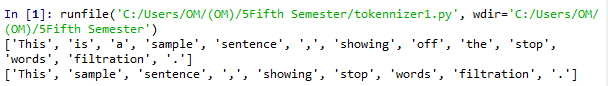
if w not in stop\_words:

filtered\_sentence.append(w)

print(word\_tokens)

print(filtered\_sentence)

The Output:



2. Write a program to tokenize

a) A sentence

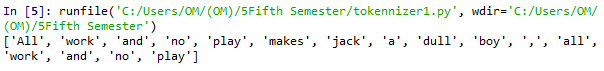
The Code:

from nltk.tokenize import sent\_tokenize, word\_tokenize

data = "All work and no play makes jack a dull boy, all work and no play"

print(word\_tokenize(data))

The Output:



b) Multiple sentences

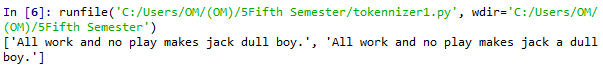
The Code:

from nltk.tokenize import sent\_tokenize, word\_tokenize

data = "All work and no play makes jack dull boy. All work and no play makes jack a dull boy."

print(sent\_tokenize(data))

The Output:



3. Write a program (using nltk toolkit in python environment) to tokenize

a) Sentence

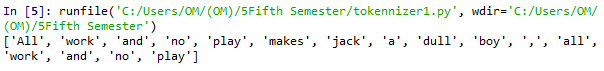
The Code:

from nltk.tokenize import sent\_tokenize, word\_tokenize

data = "All work and no play makes jack a dull boy, all work and no play"

print(word\_tokenize(data))

The Output:



b) Multiple sentences

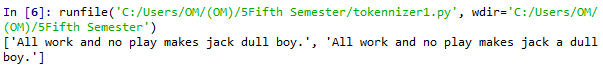
The Code:

from nltk.tokenize import sent\_tokenize, word\_tokenize

data = "All work and no play makes jack dull boy. All work and no play makes jack a dull boy."

print(sent\_tokenize(data))

The Output:



c) A paragraphe

The Code

import re

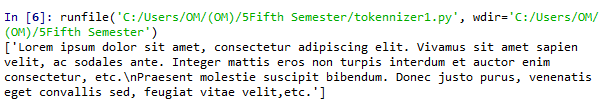
from nltk.tokenize import sent\_tokenize, word\_tokenize

x = """Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus sit amet sapien velit, ac sodales ante. Integer mattis eros non turpis interdum et auctor enim consectetur, etc.

Praesent molestie suscipit bibendum. Donec justo purus, venenatis eget convallis sed, feugiat vitae velit,etc."""

print(re.split('\s{4,}',x))

The Output:



d) Information of a complete web page

The Code:

from urllib import request

from nltk.tokenize import sent\_tokenize, word\_tokenize

url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"

html = request.urlopen(url).read().decode('utf8')

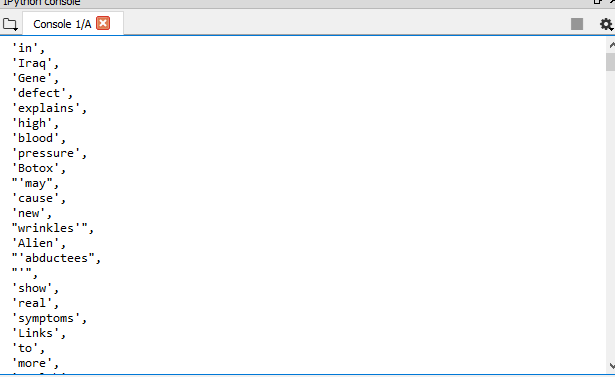
html[:60]

from bs4 import BeautifulSoup

raw = BeautifulSoup(html).get\_text()

tokens = word\_tokenize(raw)

tokens

The Output: 

Stemming

The Code:

from nltk.stem import PorterStemmer

from nltk.tokenize import sent\_tokenize, word\_tokenize

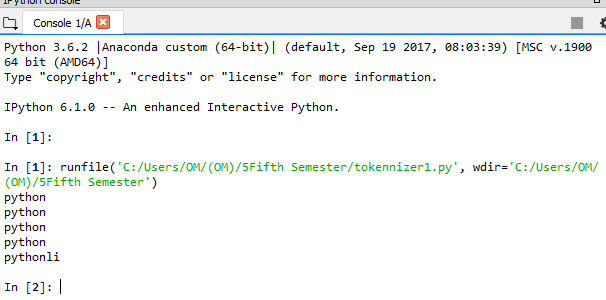
ps = PorterStemmer()

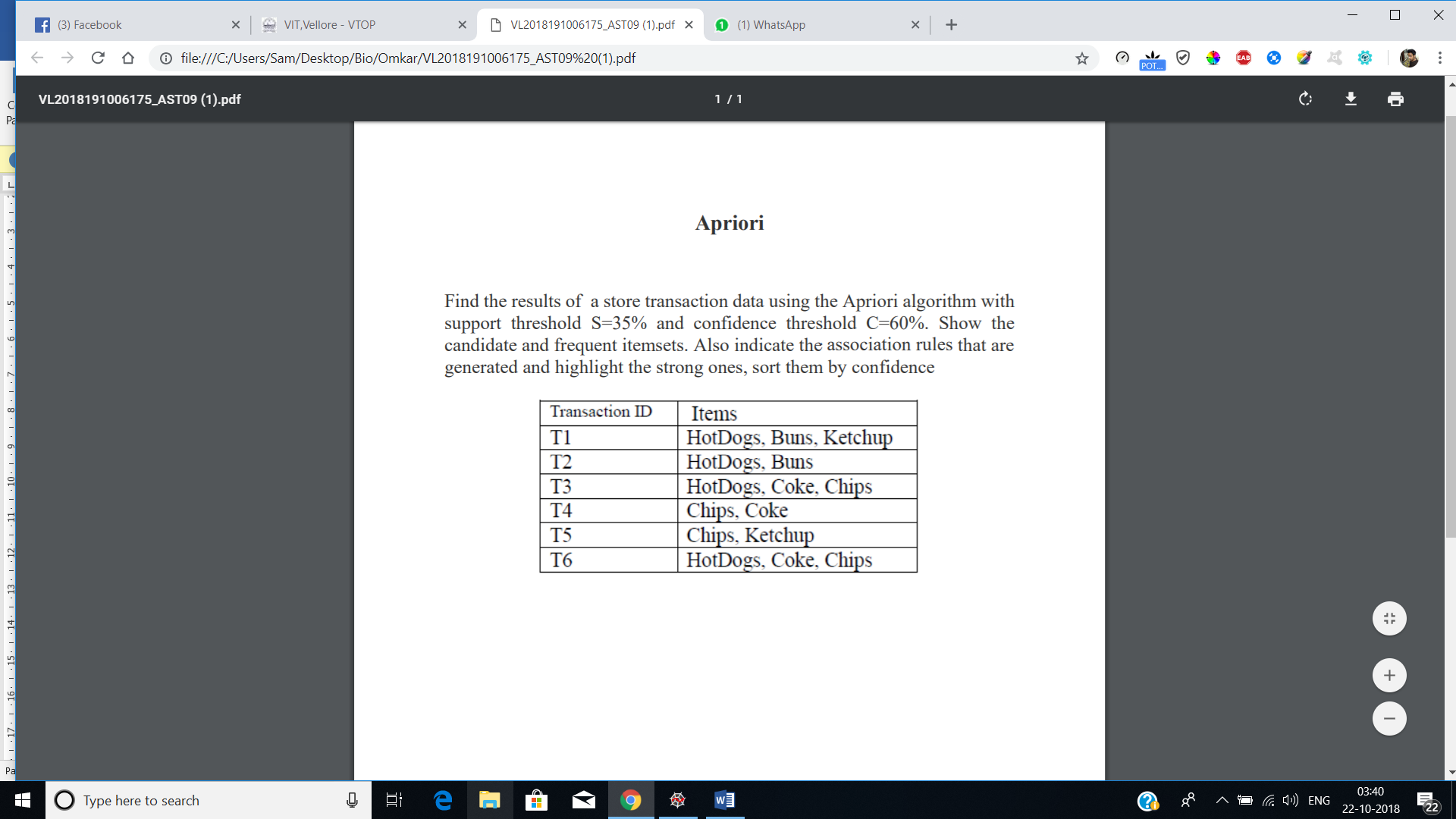
example\_words = ["python","pythoner","pythoning","pythoned","pythonly"]

for w in example\_words:

print(ps.stem(w))

The Output:





**Apriori:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from apyori import apriori

dataset = [['HotDogs','Buns','Ketchup'],['HotDogs','Buns'],['HotDogs','Coke','Chips'],['Chips','Coke'],['Chips','Ketchup'],['HotDogs','Coke','Chips']]

print("Given Dataset:")

print('--------------------------')

for i in dataset:

for j in i:

print(j + ' | ', end="")

print('\n--------------------------')

ar = apriori(dataset, min\_support=0.35, min\_confidence=0.6, min\_lift=1, max\_length=3)

result = list(ar)

lofr = len(result)

finalres=[]

for i in result:

for j in i[2]:

finalres.append([j,i[1]])

finalres.reverse()

print("\n\n")

for i in finalres:

p = i[0]

item=[ x for x in p]

if(len(item[0])>0):

x1, \*\_ = item[0]

else:

x1="Null"

if(len(item[1])>0):

x2, \*\_ = item[1]

else:

x2="Null"

print('\n\n--------------------------------')

print("Association rule: " + x1 + " -> " + x2 )

print("Confidence: " + str(i[0][2]))

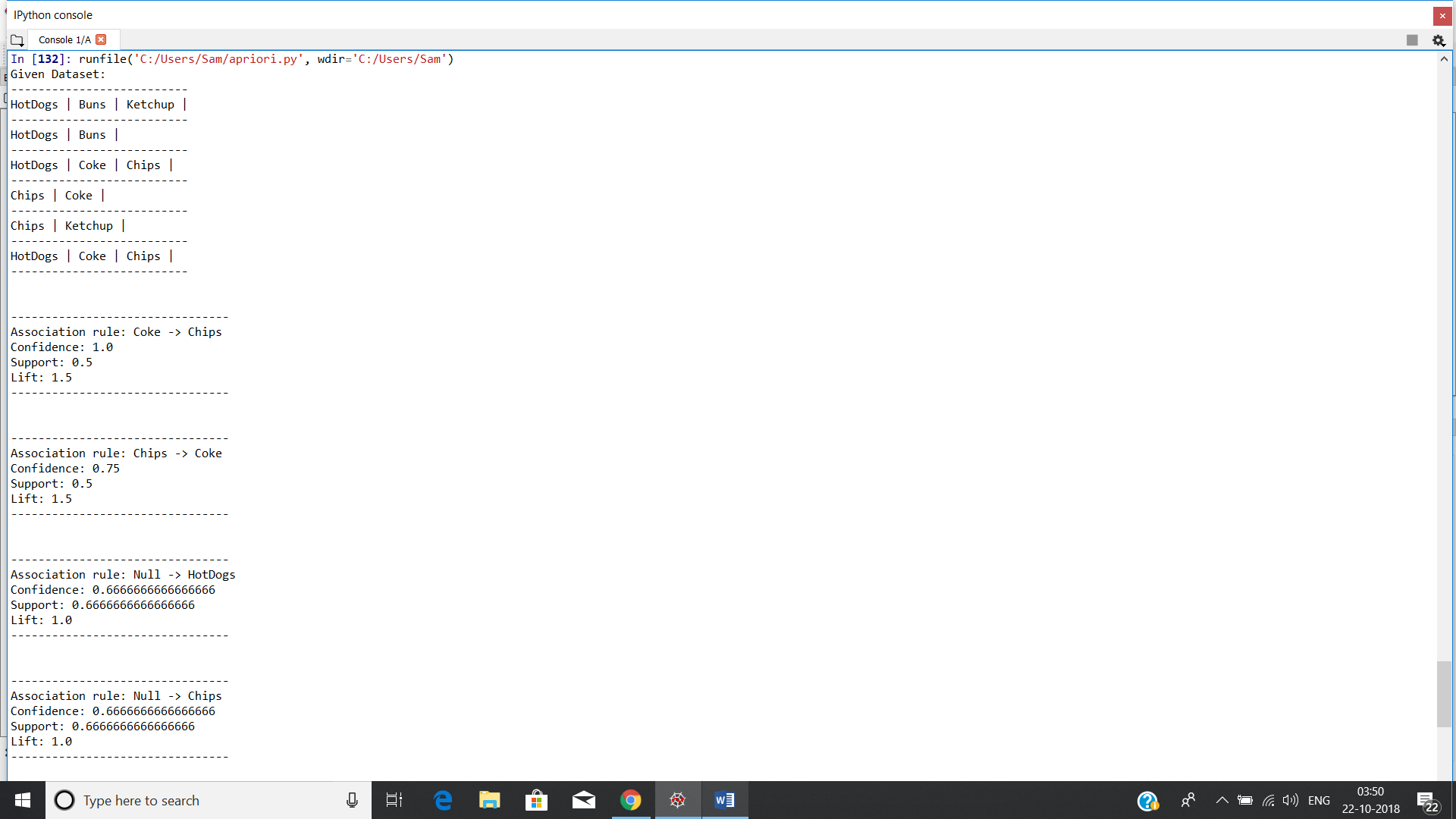
print("Support: " + str(i[1]))

print("Lift: " + str(i[0][3]))

print('--------------------------------')

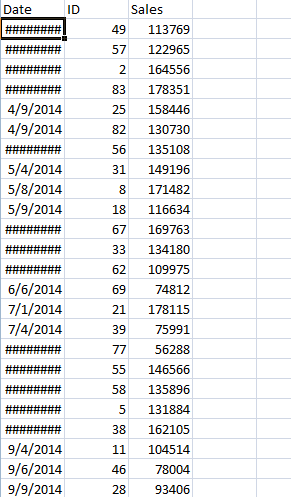
print()

**Output:**



# The K-means

# The dataset:



Note: the date is coming encrypted but it still exits.

# The Code:

import pandas

import pylab as pl

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

variables = pandas.read\_csv('sales.csv')

Y = variables[['Sales']]

X = variables[['ID']]

X\_norm = (X - X.mean()) / (X.max() - X.min())

Y\_norm = (Y - Y.mean()) / (Y.max() - Y.min())

Nc = range(1, 20)

kmeans = [KMeans(n\_clusters=i) for i in Nc]

kmeans

score = [kmeans[i].fit(Y).score(Y) for i in range(len(kmeans))]

score

pl.plot(Nc,score)

pl.xlabel('Number of Clusters')

pl.ylabel('Score')

pl.title('Elbow Curve')

pl.show()

pca = PCA(n\_components=1).fit(Y)

pca\_d = pca.transform(Y)

pca\_c = pca.transform(X)

kmeans=KMeans(n\_clusters=3)

kmeansoutput=kmeans.fit(Y)

kmeansoutput

pl.figure('3 Cluster K-Means')

pl.scatter(pca\_c[:, 0], pca\_d[:, 0], c=kmeansoutput.labels\_)

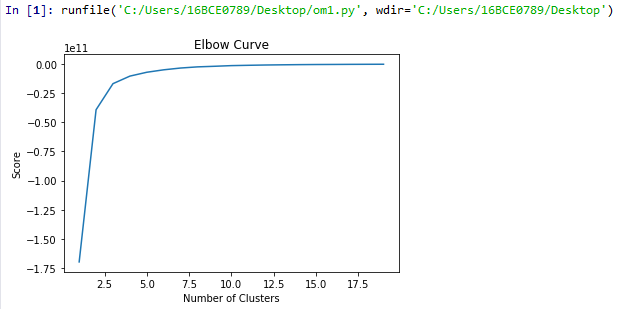
pl.xlabel('Sales')

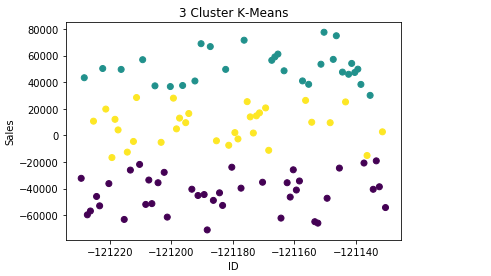
pl.ylabel('ID')

pl.title('3 Cluster K-Means')

pl.show()

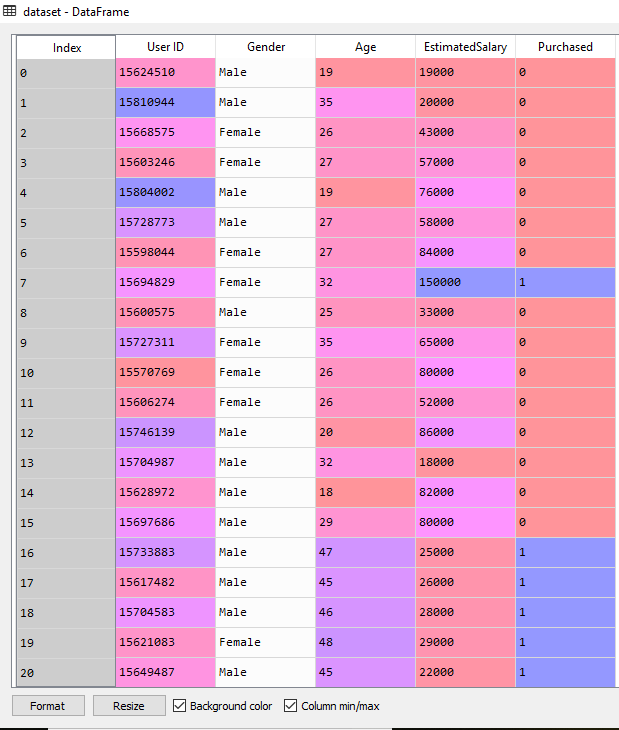
# The Output:





# Classification:

**The Dataset:**



**The Code:**

# Logistic Regression

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

from sklearn.cross\_validation import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fitting Logistic Regression to the Training set

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

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

# Predicting the Test set results

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

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Test set)')

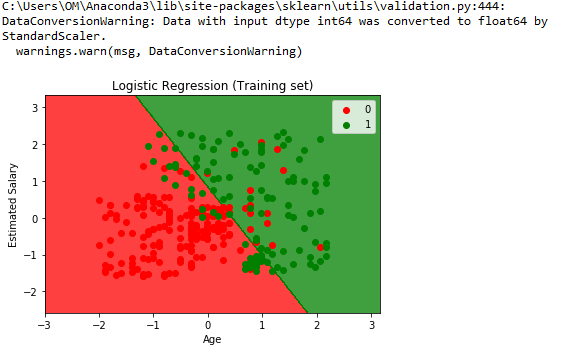
plt.xlabel('Age')

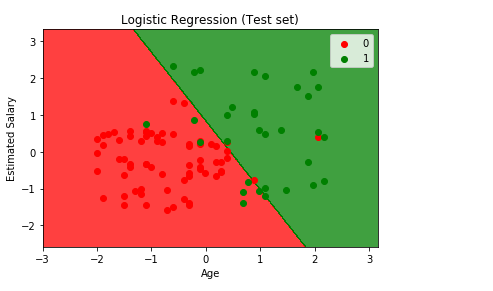
plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

**The Output:**





Tf-idf

import math

f1=open("Doc - 1.txt",'r')

f2=open("Doc - 2.txt",'r')

f3=open("Doc - 3.txt",'r')

l1=f1.read().split()

l2=f2.read().split()

l3=f3.read().split()

tf1=[]

tf2=[]

tf3=[]

s1=[i for i in l1]

s2=[i for i in l2]

s3=[i for i in l3]

s1=set(s1)

s1=list(s1)

s2=set(s2)

s2=list(s2)

s3=set(s3)

s3=list(s3)

for i in s1:

c=l1.count(i)

c=c/float(len(s1))

tf1.append(c)

for i in s2:

c=l2.count(i)

c=c/float(len(s2))

tf2.append(c)

for i in s1:

c=l2.count(i)

c=c/float(len(s3))

tf3.append(c)

i=0

k=0

for m in s1:

g1=1

if(m in s2):

g1+=1

if(m in s3):

g1+=1

g=math.log(3/g1)

tf1[i]=tf1[i]\*g

i+=1

j=0

for m in s2:

g2=1

if(m in s1):

g2+=1

if(m in s3):

g2+=1

g=math.log(3/g2)

tf2[j]=tf2[j]\*g

j+=1

for m in s3:

g3=1

if(m in s1):

g3+=1

if(m in s2):

g3+=1

g=math.log(3/g3)

tf1[k]=tf1[k]\*g

k+=1

myMax1 = tf1[0]

for num in tf1:

if myMax1 < num:

myMax1 = num

val1=tf1.index(myMax1)

myMax3 = tf3[0]

for num in tf3:

if myMax3 < num:

myMax3 = num

val3=tf3.index(myMax3)

myMax2 = tf2[0]

for num in tf2:

if myMax2 < num:

myMax2 = num

val2=tf2.index(myMax2)

print("Top word in Document - 1 is ---- ",s1[val1]," with tf-idf = ",myMax1)

print("Top word in Document - 2 is ---- ",s2[val2]," with tf-idf = ",myMax2)

print("Top word in Document - 3 is ---- ",s3[val3]," with tf-idf = ",myMax3)