# CSE-3024 WEB MINING LAB ASSIGNMENT 1

### Question 1:

**Aim:** Tokenize an input text file without using NLTK and remove the stop words from the tokens.

#### Procedure:

* Firstly, we import the text file in our work space. To do this we can use open method of python which reads the file into our workspace.
* Next, we read each word into a variable as string. This can be done using a nested for loop wherein we split each word whenever we encounter a space.
* Next, using regex in python we remove the punctuations from our string input. This will make sure that tokens are free from sentence structure.
* Using split() function of NumPy lists we tokenize each token in our text and save it in a list.
* Then we use NumPy’s unique method to only include unique

tokens from our identified set of tokens.

* To remove stop words, we create a tentative list of stop words and using a nested for loop we check if the given token belongs to that list or not. If it doesn’t then we save it else we discard it.
* Finally, we print our list that contains the resultant tokens post removal of stop words.

### Code:

#Reading input from a text file and saving it as a string text = ""

with open('test\_file.txt') as file: for line in file:

for word in line.split(): text= text + " " + word

#Removing punctuations from our input file import re

text = re.sub(r'[^\w\s]', '', text) text

#Printing each token print(text.split())

#Printing unique tokens import numpy as np print(np.unique(text.split()))

#Removing StopWords

stopwords = ["i", "a", "am", "and", "at", "for", "in", "is", "my", "of", "this"] res = []

for x in tokens:

if x not in stopwords: res.append(x)

#Printing cleaned tokens in our input text file print(res)

#### Question 2

**2a) Aim:** Tokenize a single sentence text without using NLTK.

**2b) Aim:** Tokenize a multi-line text without using NLTK.

#### Procedure:

* + Firstly, we import the text file into our workspace. To do this we can use open method of python which reads our text file to the workspace.
  + Next, we read each word in the input file as a string input. This can be implemented using nested for loop wherein we split each word whenever we encounter a space.
  + Next, using regex we remove the punctuations from our input string. This will render token a better syntactic structure and break the sentence bonds.
  + Next, we split and store each token into a list using split method of python.
  + Then finally we remove stop words as we did in previous assignment.

# CSE-3024 WEB MINING LAB ASSIGNMENT 2

**Aim:** Tokenize an input text file using NLTK and remove the stop words from the tokens.

#### Procedure:

* Firstly, we import the text file in our work space. To do this we can use open method of python which reads the file into our workspace.
* Next, we import the necessary NLTK libraries including stopwords, sent\_tokenize and word\_tokenize.
* Next, using regex in python we remove the punctuations from our string input. This will make sure that tokens are free from sentence structure.
* Using word\_tokenize we tokenize each word and store it in a variable list named tokens
* Then using Numpy’s unique method we include only unique

tokens from our text.

* To remove stopwords, we use NLTK’s stopwords library and a nested loop that ensures all the words not present in that library are added onto the result list.
* Finally, we print our list that contains the resultant tokens post removal of stop words as well.

**Code:**

#Reading input from a text file and saving it as a string text = ""

with open('test\_file.txt') as file: for line in file:

for word in line.split(): text= text + " " + word

#Importing libraries import re

import nltk

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

from nltk.stem.porter import PorterStemmer

#Removing punctuations from our input file import re

text = re.sub(r'[^\w\s]', '', text) text

#Printing each token

tokens = word\_tokenize(text) print(tokens)

#Printing unique tokens import numpy as np tokens = np.unique(tokens) print(tokens)

#Removing Stopwords res = []

for x in tokens:

if x not in set(stopwords.words('english')): res.append(x)

print(res)

#Printing cleaned tokens in our input text file print(res)

#### Question

* 1. **Aim:** Tokenize a single sentence text using NLTK.
  2. **Aim:** Tokenize a multi-line text using NLTK.

#### Procedure:

* + - Firstly, we import the text file into our workspace. To do this we can use open method of python which reads our text file to the workspace.
    - Next, we read each word in the input file as a string input. This can be implemented using nested for loop wherein we split each word whenever we encounter a space.
    - Next, using regex we remove the punctuations from our input string. This will render token a better syntactic structure and break the sentence bonds.
    - Next, we split and store each token into a list using nltk’s sentence\_tokenize method
    - Then finally we remove stop words as we did in previous assignment.

#### Code here is same as previous one… just that input file is different…

# CSE-3024 WEB MINING LAB ASSIGNMENT 3

**Aim:** To write a python program to find important words from the text using TF-IDF. Use a minimum of 5 documents with the real text source from a web page of some relevance.

#### Procedure:

* Firstly, we import our libraries that would help us in doing this term frequency count.
* Next, we declare and write tf, idf, n\_containing and tf\_idf functions that will help assist the return values and make code more readable.
* Then we create 5 documents and read them in our workspace.
* We then make the bloblist that contains all the documents in list format. And then we print the counts of top 3 words in each document.
* Then we calculate the cosine similarity using inbuilt cosine\_similarity matrix.
* For the above we need to create a pandas data frame of count vectors.

#### Code:

#Importing Libraries import math

from textblob import TextBlob as tb

#Creating the Term Frequency return function def tf(word, blob):

return blob.words.count(word)

#Creaeting containing function def n\_containing(word, bloblist):

return sum(1 for blob in bloblist if word in blob.words)

#Function to return Inverse Document Frequency def idf(word, bloblist):

return math.log(len(bloblist))/(1+n\_containing(word, bloblist))

#Function to return Term Frequency-Inverse Document Frequency def tfidf(word, blob, bloblist):

return tf(word, blob) \* idf(word, bloblist)

#Reading First Input File with open('Test1.txt') as a:

test1 = (a.read()) document1 = tb(test1)

#Reading Second Input File with open('Test2.txt') as a:

test2 = (a.read()) document2 = tb(test2)

#Reading Third Input File with open('Test3.txt') as a:

test3 = (a.read()) document3 = tb(test3)

#Reading Fourth Input File with open('Test4.txt') as a:

test4 = (a.read()) document4 = tb(test4)

#Reading Fifth Input File with open('Test5.txt') as a:

test5 = (a.read()) document5 = tb(test5)

#Printing the top three words in each document

bloblist = [document1, document2, document3, document4, document5] for i, blob in enumerate(bloblist):

print("Top words in document {}". format(i+1))

scores = {word: tfidf(word, blob,bloblist) for word in blob.words} sorted\_words = sorted(scores.items(), key=lambda x:x[1], reverse=True) for word, score in sorted\_words[:3]:

print("\tWord: {}, TF-IDF: {}".format(word, round(score, 5)))

#Calculating Cosine Similarity

from sklearn.feature\_extraction.text import CountVectorizer import pandas as pd

documents = [test1, test2, test3, test4, test5]

#Creating the Document Term Matrix count\_vectorizer = CountVectorizer()

sparse\_matrix = count\_vectorizer.fit\_transform(documents)

#Creating a dataframe to store each count\_vectorizer doc\_term\_matrix = sparse\_matrix.todense()

df = pd.DataFrame(doc\_term\_matrix, columns=count\_vectorizer.get\_feature\_names(), index=['test1', 'test2','test3', 'test4', 'test5'])

df

#Printing the Cosine Similarity

from sklearn.metrics.pairwise import cosine\_similarity print(cosine\_similarity(df, df))

# CSE-3024 WEB MINING LAB ASSIGNMENT 4

**Aim:** Use BeautifulSoup or Scrapy to crawl any one of the E- commerce websites of your choice and perform the same.

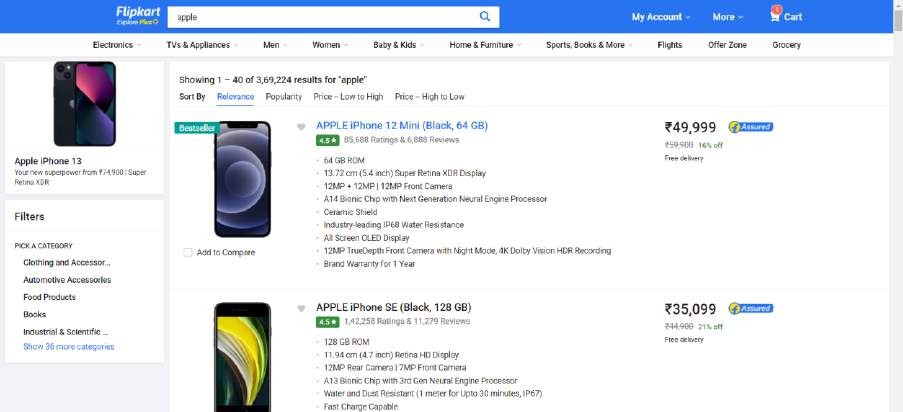
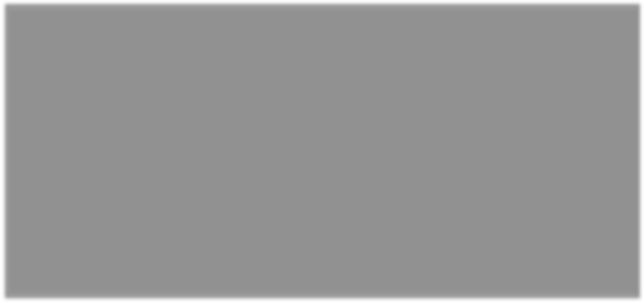
The following information needs to be extracted from the page:

* + 1. Product Name
    2. Product Price
    3. Product Discount
    4. Product Image

#### Procedure:

* Firstly, we import our scraping libraries of BeautifulSoup and requests.
* We then assign a url variable that stores the URL of web page to be scrapped. In our case it is going to be the URL of a flipkart page featuring apple products.
* Using requests library’s get method we take in the page as a text input and store it in variable named text.
* We next create an HTML parser using BeautifulSoup library and store it in soup variable.
* Then we create in a tags variable that store our div block of information that we are seeking. The class name depends on the site we are looking and its DOM structure.
* Then we import our csv related libraries to store our results in csv format.
* Then we open a writeonly file and write in the headers named as Name, Price, Discount and Image.
* Next, we use the getattr and tag methods to successfully parse in the DOM structure of our tags variable.
* This will include details of Name of the product, Price of the product, Discount on the product and Image tag of the product.
* We are getting the image tag to be stored in csv file.

#### Site we are scrapping:



**URL of the above website:**

[https://www.flipkart.com/search?q=apple&otracker=search&otrack](https://www.flipkart.com/search?q=apple&otracker=search&otracker1=search&marketplace=FLIPKART&as-show=on&as=off) [er1=search&marketplace=FLIPKART&as-show=on&as=off](https://www.flipkart.com/search?q=apple&otracker=search&otracker1=search&marketplace=FLIPKART&as-show=on&as=off)

#### Code:

#Importing scrapper librariess from bs4 import BeautifulSoup import requests

#Scrapping the website url =

"https://[www.flipkart.com/search?q=apple&otracker=search&otracker1=search&marketplace=FLIP](http://www.flipkart.com/search?q=apple&otracker=search&otracker1=search&marketplace=FLIP) KART&as-show=on&as=off"

page = requests.get(url).text

soup = BeautifulSoup(page, 'html.parser')

tags = soup.find\_all('div', class\_="\_1AtVbE col-12-12")

#Creating csv

from csv import writer import colorama

from colorama import Fore

print(Fore.WHITE+"Scraping data "+Fore.GREEN+"done...") with open('result.csv','w', encoding='utf8',newline='') as f:

thewriter = writer(f)

header = ['Name','Price','Discount','Image'] thewriter.writerow(header)

for tag in tags:

name = getattr(tag.find('div',class\_="\_4rR01T"),'text', None)

price = getattr(tag.find('div', class\_="\_30jeq3 \_1\_WHN1"),'text', None) discount = getattr(tag.find('div',class\_="\_3Ay6Sb"),'text', None)

image = tag.find('img', class\_="\_396cs4 \_3exPp9") info = [name, price, discount, image] thewriter.writerow(info)

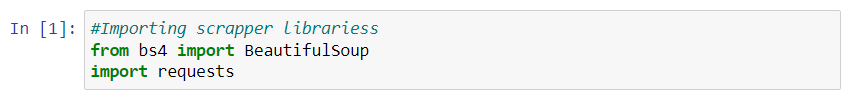
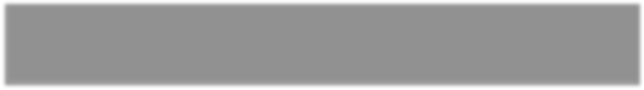
print(Fore.WHITE+"Successfully scrapped data!! Results in " + Fore.GREEN+"result.csv")

#Reading the CSV File import pandas as pd

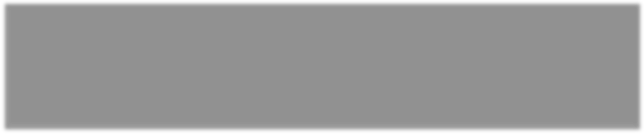
data = pd.read\_csv("result.csv")

#Printing the data data

#### Code Snippet and Outputs:



Here we are importing our BeautifulSoup and requests libraries to be used for scrapping the web page.

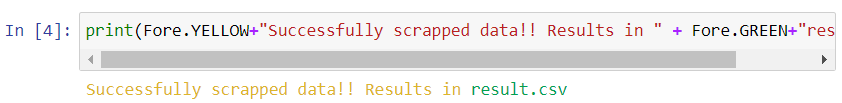
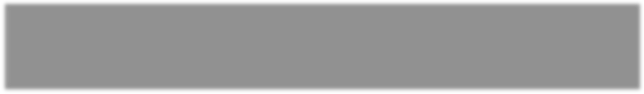
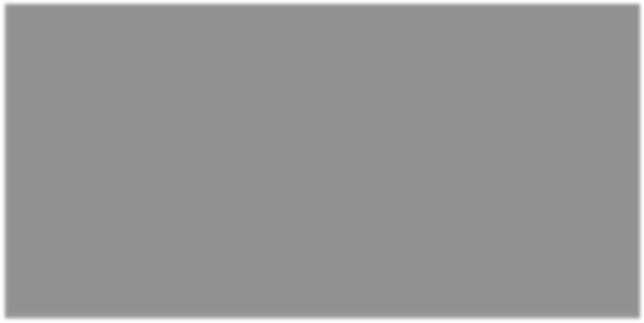


Here we are passing in the URL of our webpage to the variable and then fetching its resources in a text format to store it in a variable named page.

Then we pass it as an html source for Document Object Manipulation to another variable named soup.

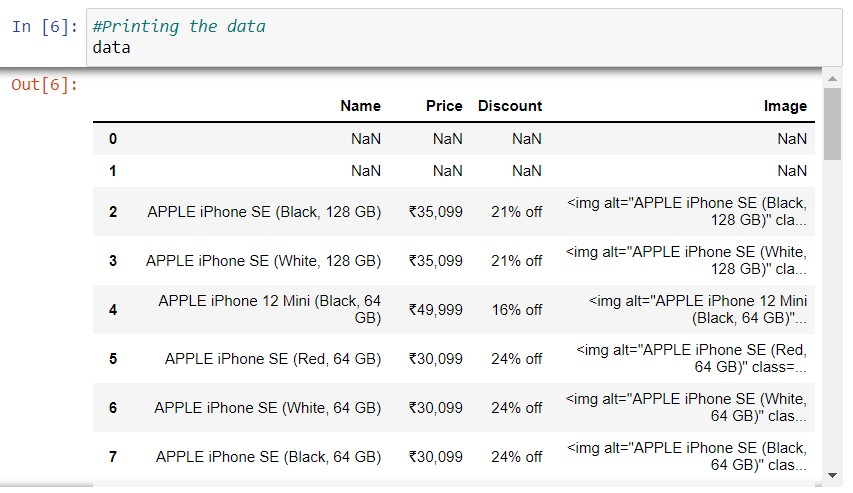
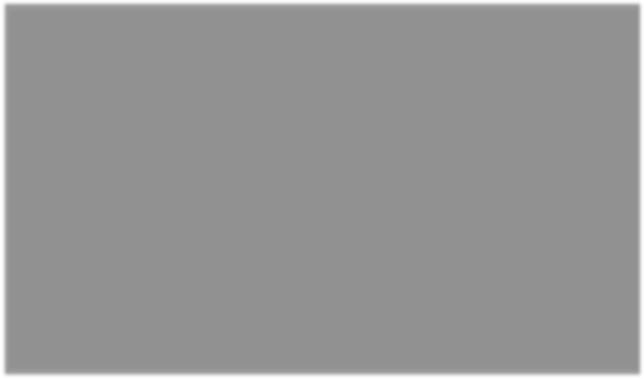
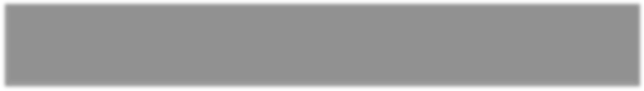
Finally, to get its div tags we use find\_all method of our bs4 library.

Here we are storing our scrapped data in a csv file named result. This file is headed with titles of Name, Price, Discount and Image. We are using getattr method to find the respective name, price and etc of a product and assign them to the variables. Finally, we make a list of these values and write this info in results.csv file. This will update the contents of our result file. We repeat the above process until we run out of tags which we obtained using soup.find\_all method in cell 2.



Finally, we print that the scrapped data is successfully stored in our working directory.

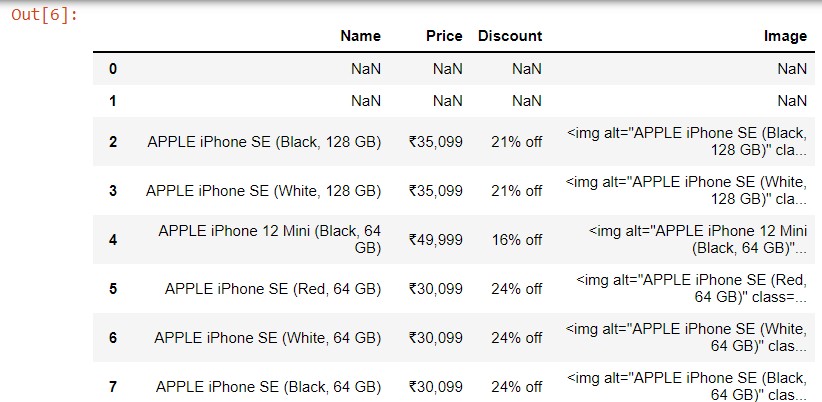
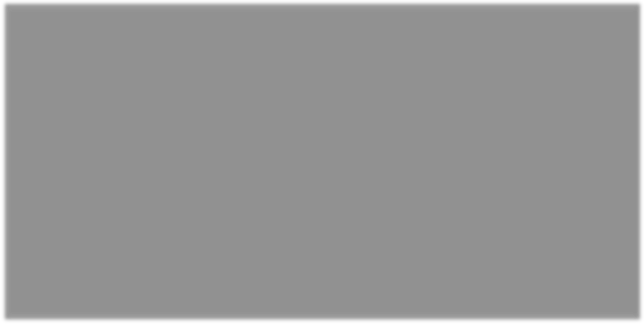
Here we are importing the result.csv file into our workspace.



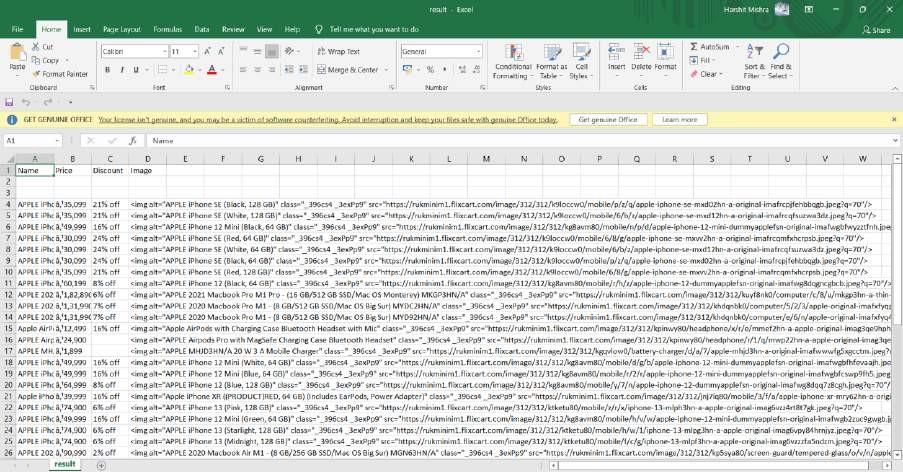
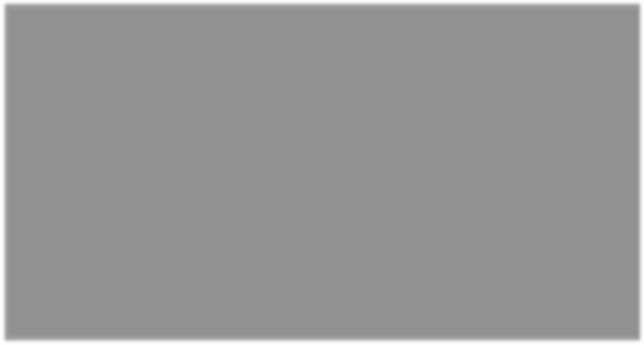
Here we are printing the data. We can see that first column contains Name of the product, second attribute contains price of the product in Indian Rupee, third attribute contains the discount offered in that product and the last attribute contains the HTML element of image formatted in the csv file.

#### Results:

**CSV file as printed in python script:**



**result.csv file:**



# CSE-3024 WEB MINING LAB ASSIGNMENT 4B

**Aim:** Use BeautifulSoup or Scrapy to crawl any one of the E- commerce websites of your choice and perform the same.

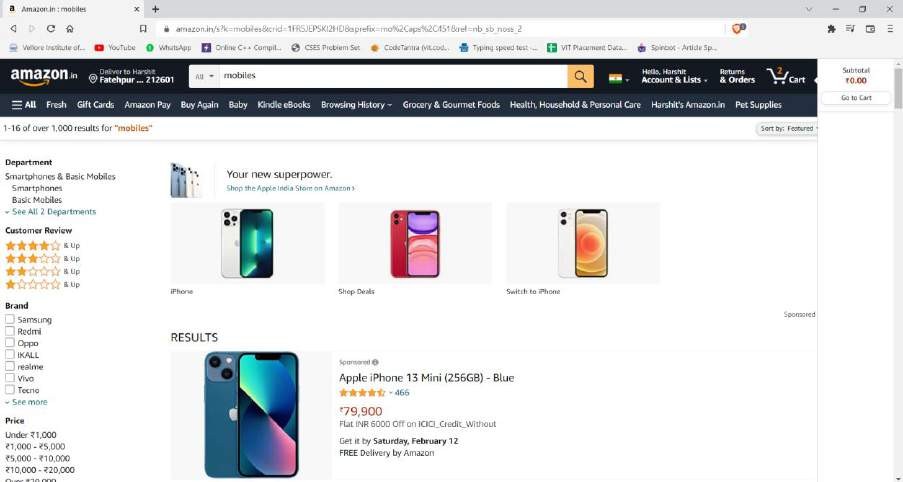
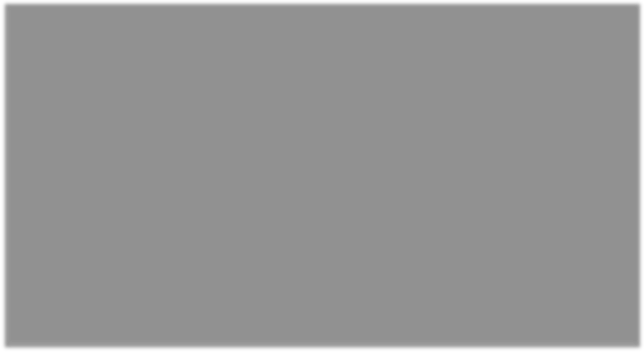
The following information needs to be extracted from the page:

1. Product Name
2. Product Price
3. Product Discount
4. Product Image

#### Procedure:

* Firstly, we install scrapy package using ‘pip install scrapy’ in anaconda prompt shell.
* Then we can start shell by using command ‘scrapy shell’.
* To initiate the crawler, we run default commands as in scrapy genspider example example.com
* Create a folder named ‘mobile’ and change the working directory to current folder.
* Create the python (.py) file inside the mobile directory and initiate the scrapy files using command “scrapy genspider example example.com”
* Here I have scrapped Amazon’s website featuring mobile phones.
* They Python code is written in the scrapy file.
* Finally, it is exported as csv file using command “scrapy crawl -o mobile data.csv”.

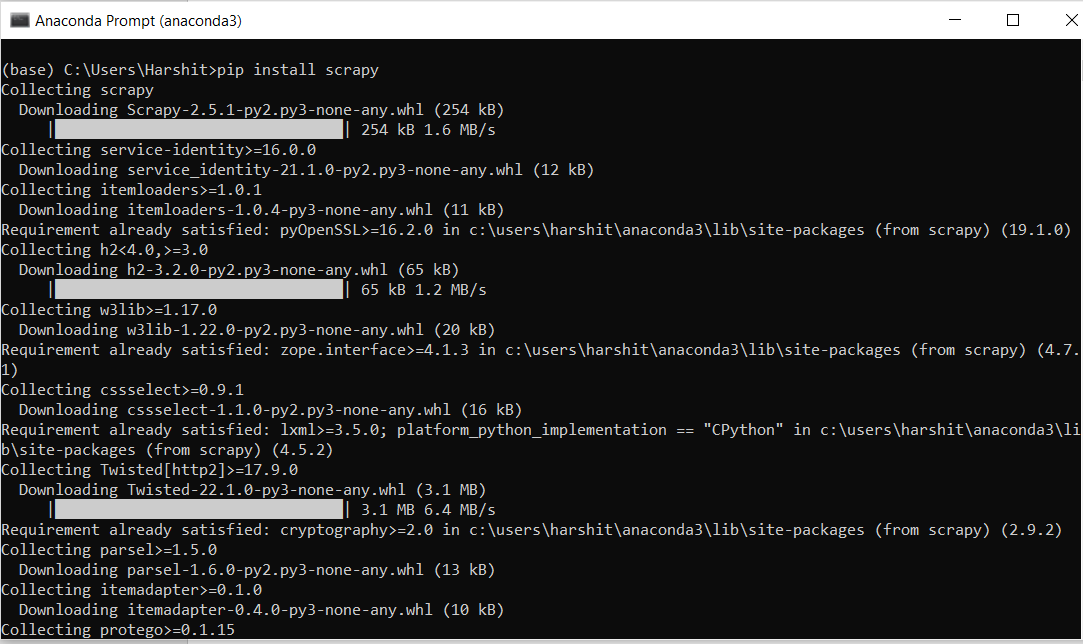
#### Site we are scrapping:



**URL of the above website:**

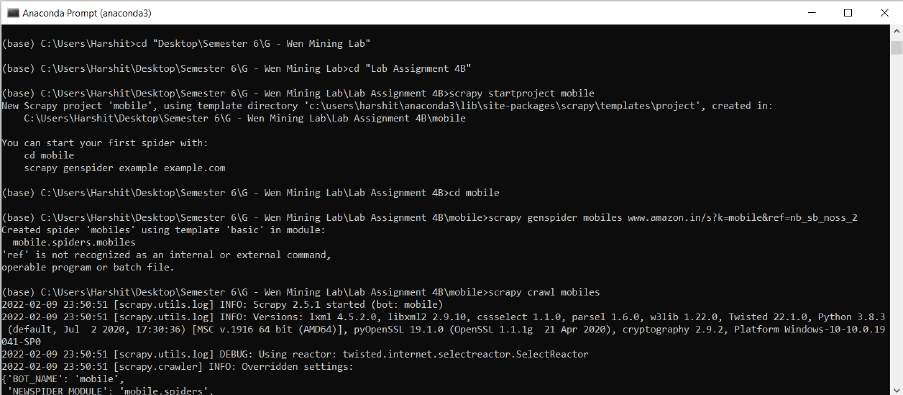
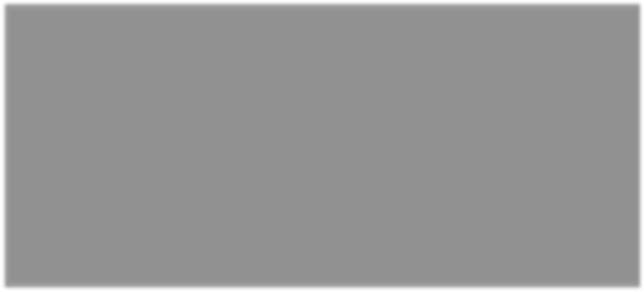
[https://www.amazon.in/s?k=mobiles&crid=1FR5JEPSKI2HD&sprefix=](https://www.amazon.in/s?k=mobiles&crid=1FR5JEPSKI2HD&sprefix=mo%2Caps%2C451&ref=nb_sb_noss_2) [mo%2Caps%2C451&ref=nb\_sb\_noss\_2](https://www.amazon.in/s?k=mobiles&crid=1FR5JEPSKI2HD&sprefix=mo%2Caps%2C451&ref=nb_sb_noss_2)

#### Installing Scrapy in Anaconda:



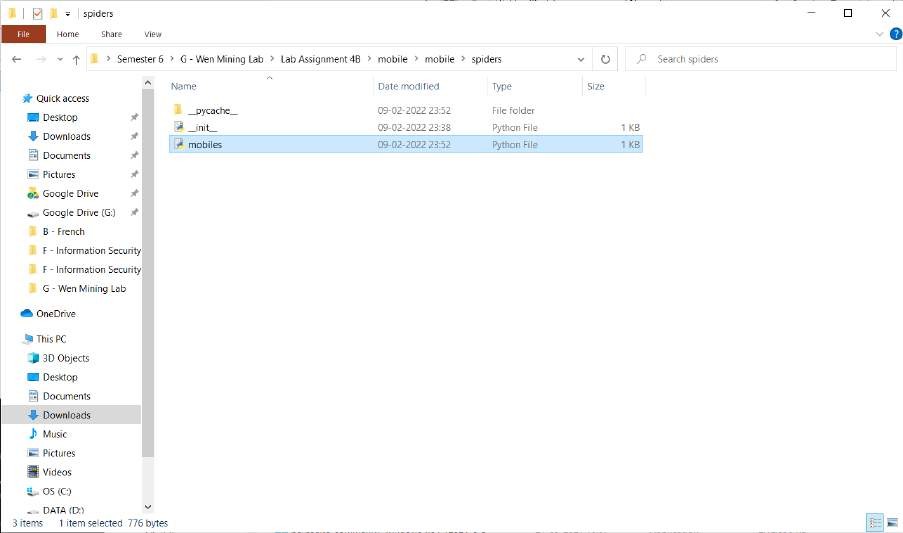
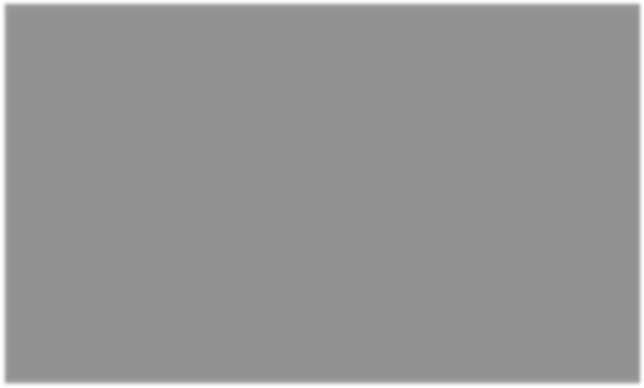
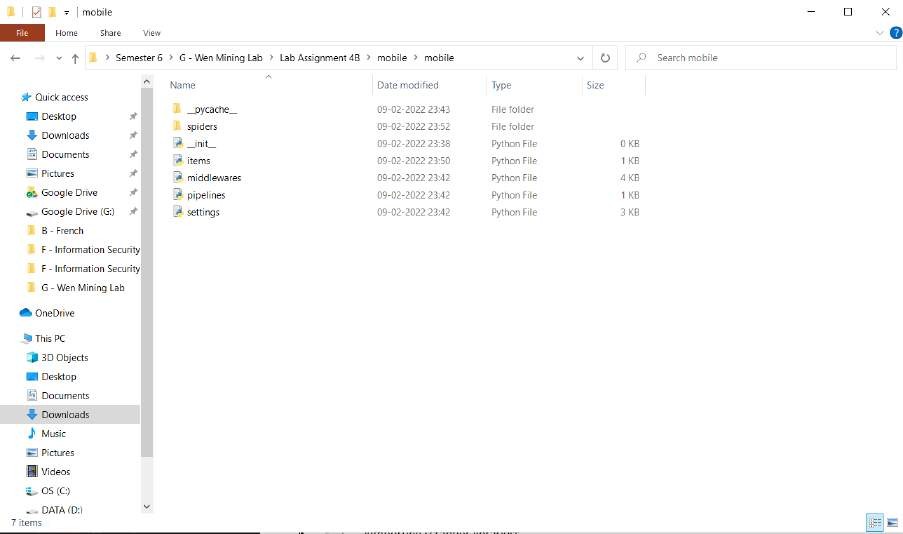
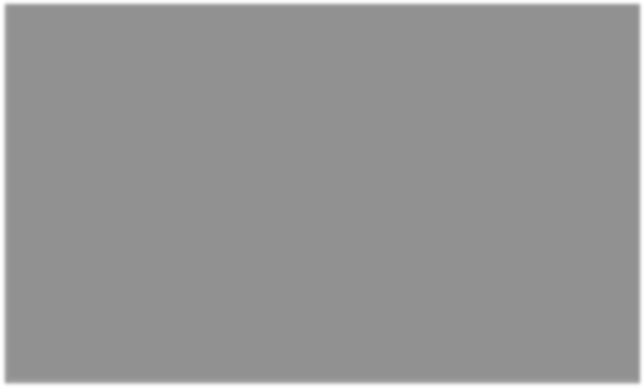
Here we use the command pip install scrapy

#### Creating Scrapy Project in mobile directory:



Here we firstly move the directory mobile that we have created. Then we run the command scrapy genspyder mobiles amazon.com

#### Directory Structure:



**Code in mobiles.py:**

import scrapy import urllib

class MobilesSpider(scrapy.Spider): name = 'mobiles'

allowed\_domains = ['[www.amazon.in/s?k=mobile']](http://www.amazon.in/s?k=mobile%27) start\_urls = ['http://www.amazon.in/s?k=mobile/']

def parse(self, response):

i=0

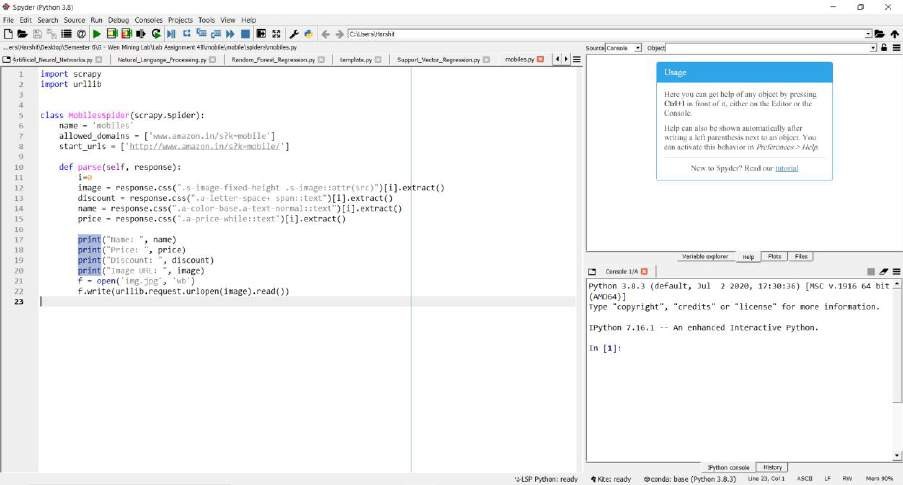
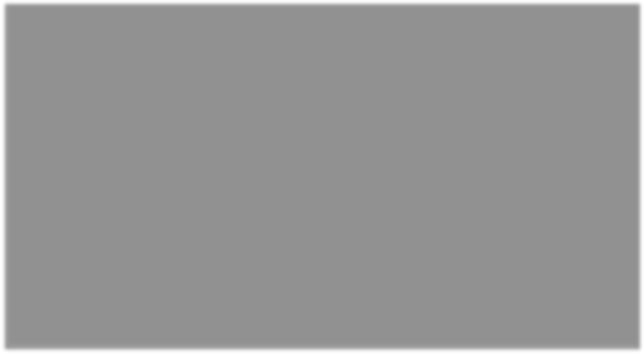
image = response.css(".s-image-fixed-height .s-image::attr(src)")[i].extract() discount = response.css(".a-letter-space+ span::text")[i].extract()

name = response.css(".a-color-base.a-text-normal::text")[i].extract() price = response.css(".a-price-while::text")[i].extract()

print("Name: ", name) print("Price: ", price) print("Discount: ", discount) print("Image URL: ", image) f = open('img.jpg', 'wb')

f.write(urllib.request.urlopen(image).read())data!! Results in " + Fore.GREEN+"result.csv")

#### Code Snippet:



**Code in Items.py:**

# Define here the models for your scraped items #

# See documentation in:

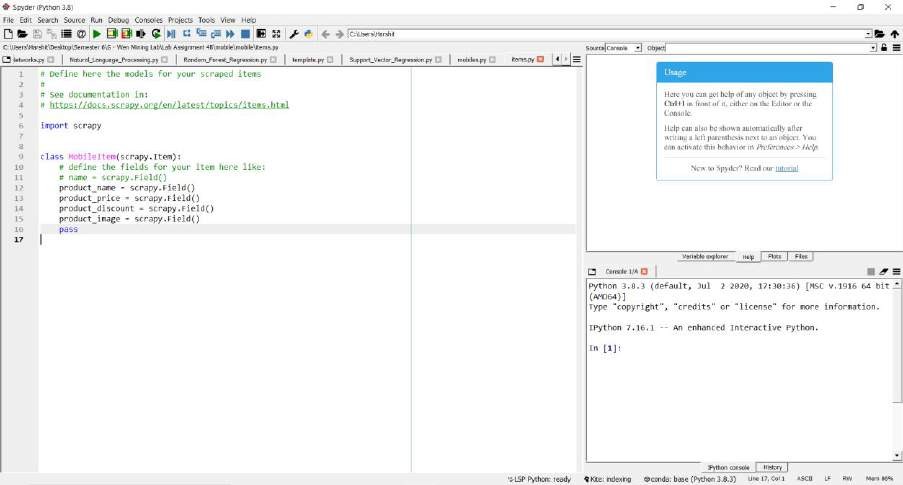
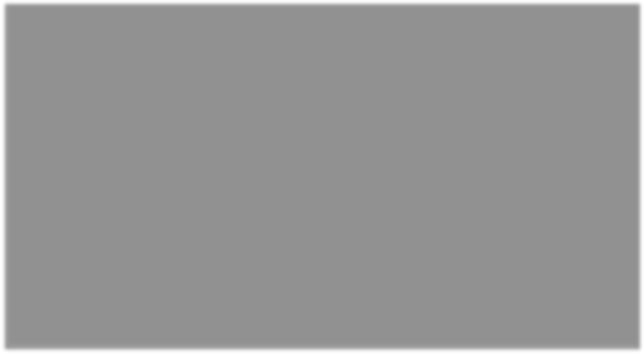
# https://docs.scrapy.org/en/latest/topics/items.html import scrapy

class MobileItem(scrapy.Item):

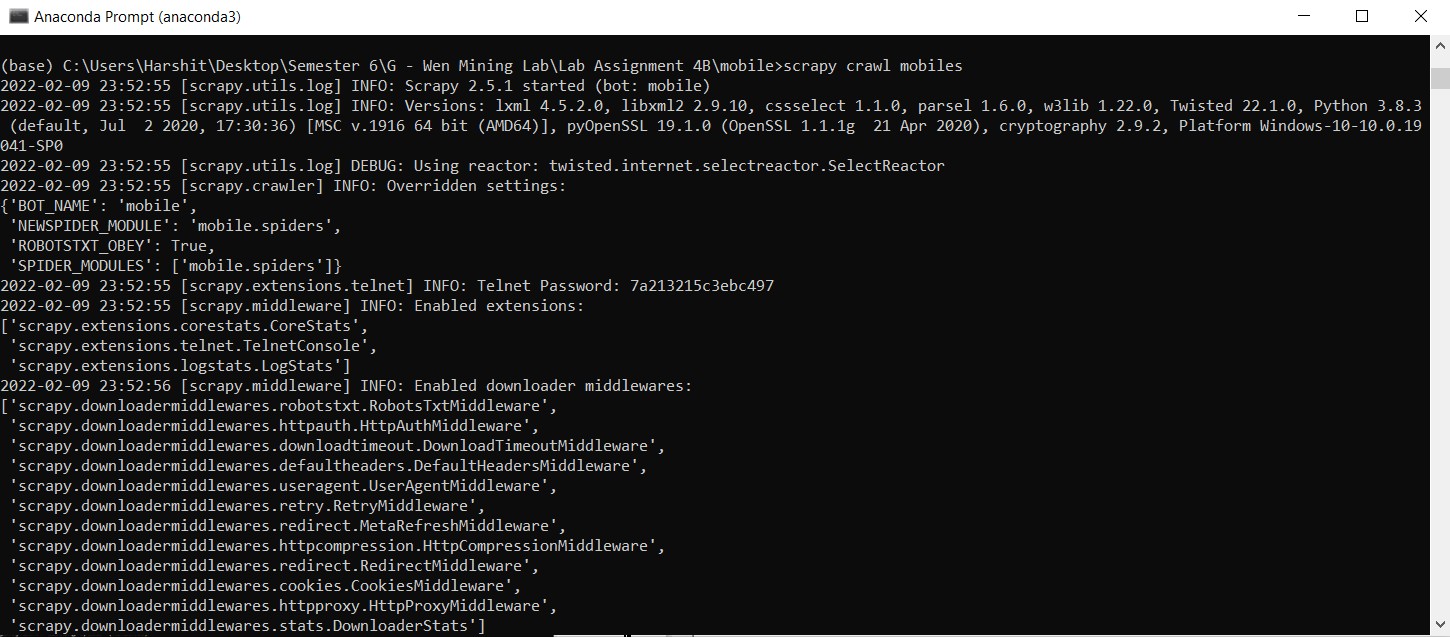
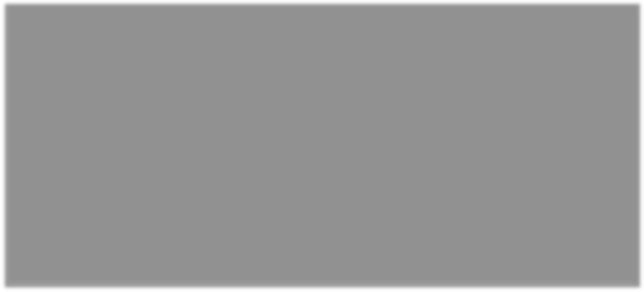
# define the fields for your item here like: # name = scrapy.Field()

product\_name = scrapy.Field() product\_price = scrapy.Field() product\_discount = scrapy.Field() product\_image = scrapy.Field()

#### Code Snippet:

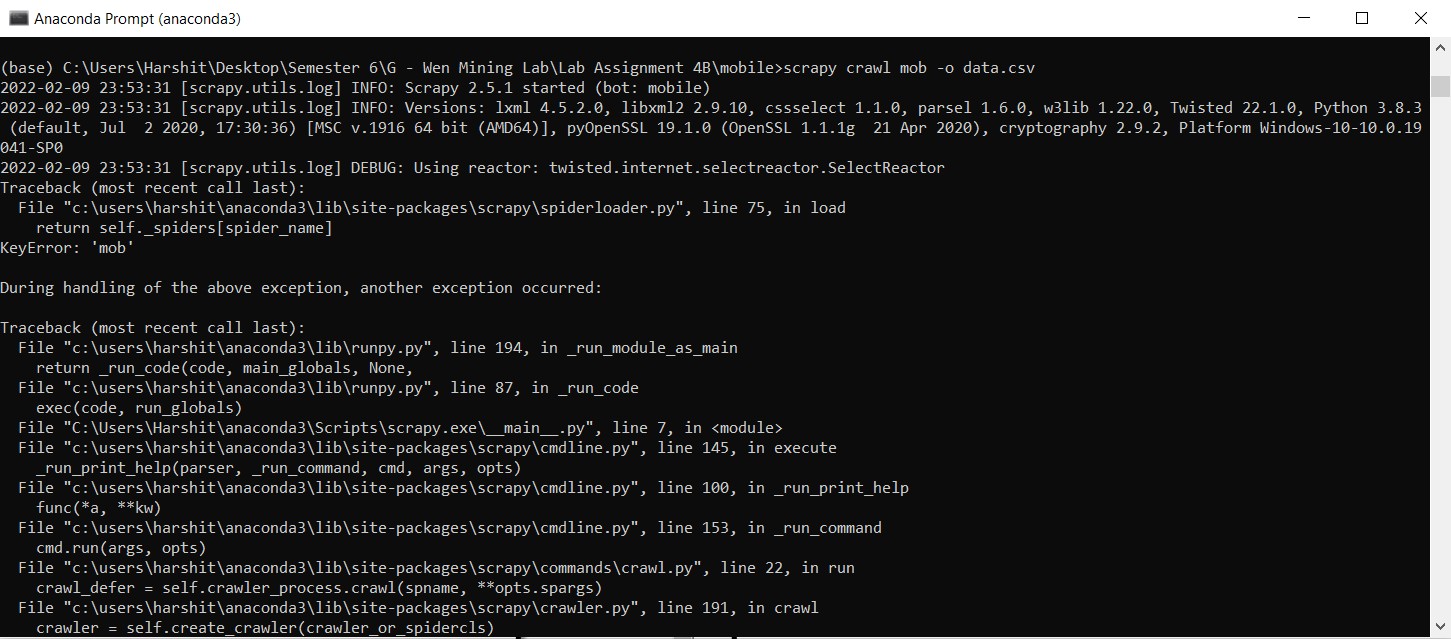
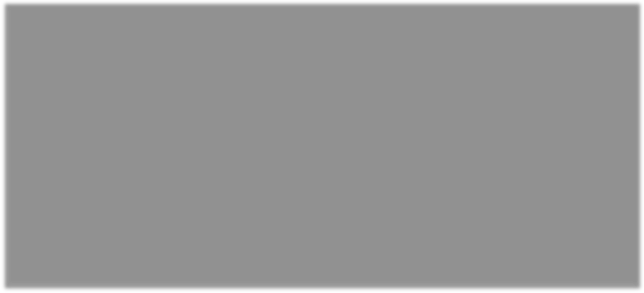


**Running the code in prompt shell:**



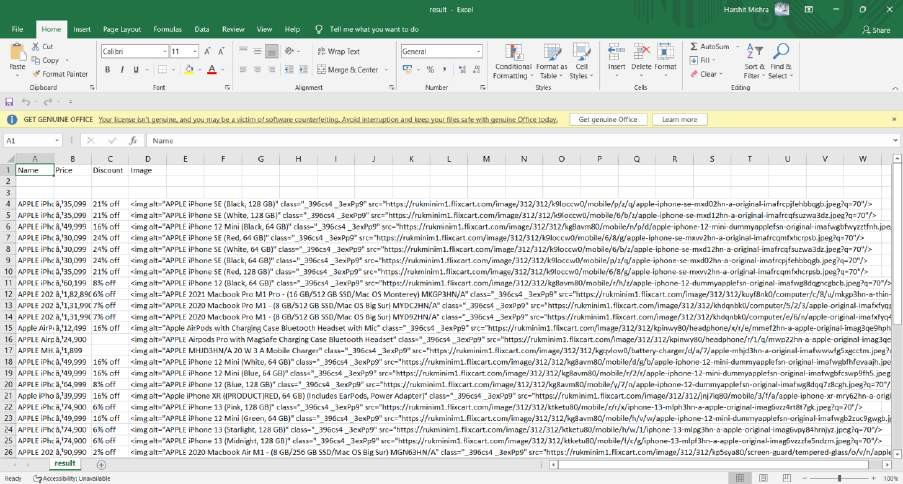
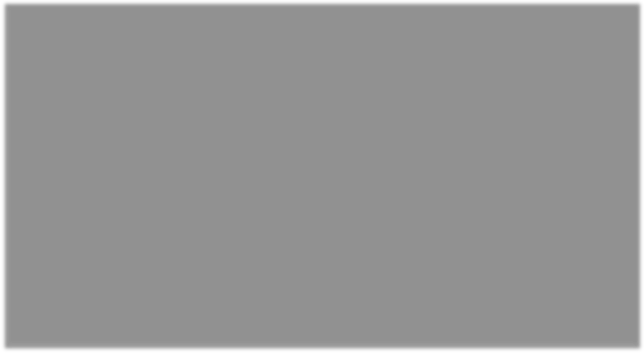
Here we use the command ‘scrapy crawl mobiles’ to crawl the web page.

#### Storing the results of scrapper in a csv file:

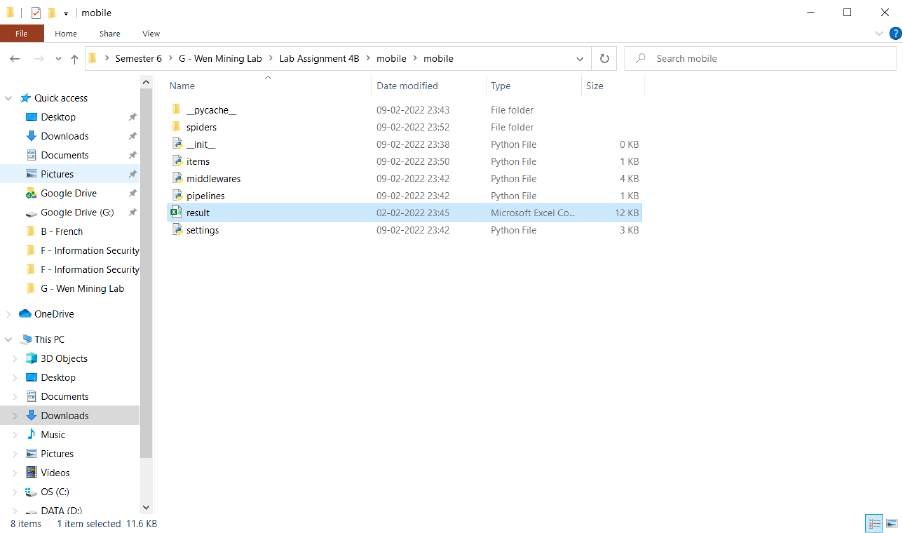
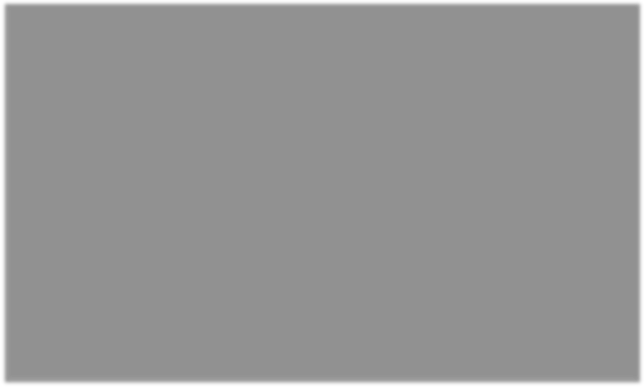


**Results:**

#### result.csv file:



**Directory:**



We can see that the result.csv file is dumped in the directory.

# CSE-3024 WEB MINING LAB ASSIGNMENT 5

**Aim:** Write a python program to perform the following encoding and decoding for the EVEN numbers between 1-20

1. Unary
2. Elias Gamma
3. Elias Delta
4. Golomb (b=10)

#### Procedure:

* Firstly, we import the numpy library to use mathematical functions such as logarithm in our code.
* We create two functions, one to convert integer to binary and the other to create binary to integer.
* We write respective functions for each of the given methods.

The functions are corresponding to Unary Encoding, Unary Decoding, Elias Gamma Encoding, Elias Gamma Decoding, Elias Delta Encoding, Elias Delta Decoding, Golomb Encoding and Golomb Decoding.

* In main program, we run a loop from 2 to 21 with a jump of 2 to get even numbers in the range from 1-20.
* We perform the above functions to each of the iterators in the above loop.

#### Code:

#Importing Library import numpy as np

#Converting Integer to Binary def intToBin(var):

return bin(var).split("0b")[1]

#Converting Binary to Integer def binToInt(var):

return int(var, 2)

#Unary Encoding

def unaryEncoding(var): unary = ""

for i in range(var-1): unary='0'+unary

unary=unary+'1' return unary

#Unary Decoding

def unaryDecoding(var): counter=0 while(var[0]=='0'):

var=var[1:] counter=counter+1

return counter+1

#Elias Gamma Encoding

def eliasGammaEncoding(var): var = intToBin(var) n=len(var)-1

for i in range(n):

var = '0'+var return var

#Elias Gamma Decoding

def eliasGammaDecoding(var): counter=0

while(var[0]=='0'):

var=var[1:] counter=counter+1

var=var[0:counter+1:1] return binToInt(var)

#Elias Delta Encoding

def eliasDeltaEncoding(var):

selector = eliasGammaEncoding(1+int(np.log2(var))) var = intToBin(var)

offset=""

for i in range(1, len(var)): offset=offset+var[i]

return (selector+offset)

#Elias Delta Decoding

def eliasDeltaDecoding(var): Nbits=eliasGammaDecoding(var)-1 ans=""

for i in range(Nbits):

ans=var[-(i+1)]+ans return binToInt('1'+ans)

#Golomb Encoding

def golombEncoding(var, b): quotientunary=unaryEncoding(int(var/b) +1) remainder=var%b

i=int(np.log2(b)) d= (2\*\*(i+1))-b

if (remainder<d):

r = intToBin(remainder) while len(r)<i:

r='0'+r

else:

r=intToBin(remainder+d) while len(r)<i+1:

r='0'+r

return quotientunary+r

#Golomb Decoding

def golombDecoding(var, b): quotient=unaryDecoding(var)-1 i=int(np.log2(b))

d=(2\*\*(i+1))-b

counter=0

while (var[0]=='0'):

var=var[1:] counter=counter+1

var=var[1:] remainder=var[0:i]

remainder=binToInt(remainder) if (remainder>=d):

remainder=intToBin(remainder) remainder=var[0:i+1] remainder=binToInt(remainder)-d

ans=quotient\*b+remainder return ans

for i in range(2,21,2): print("\n\nNumber=",i) UE = unaryEncoding(i)

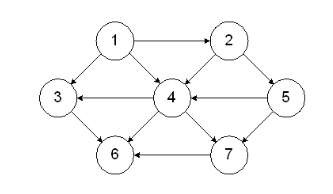
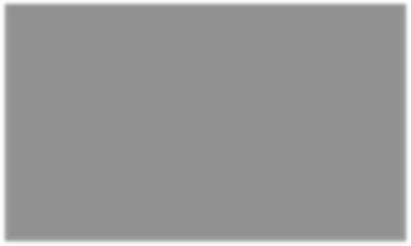
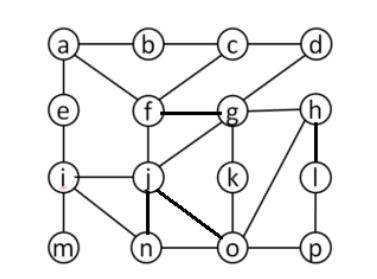
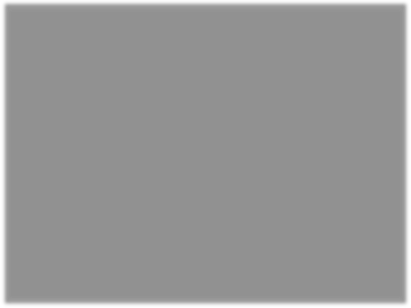
print("\tUnaryEncoding: ", UE) EGE=eliasGammaEncoding(i) print("\tElias Gamma Encoding: ",EGE) EDE=eliasDeltaEncoding(i) print("\tElias Delta Encoding: ",EDE) GE=golombEncoding(i,10) print("\tGoloumb Encoding: ",GE)

print("\tUnary Decoding:", unaryDecoding(UE))

print("\tElias Gamma Decoding:", eliasGammaDecoding(EGE)) print("\tElias Delta Decoding:", eliasDeltaDecoding(EDE)) print("\tGolomb Decoding:", golombDecoding(GE,10))

# CSE-3024 WEB MINING LAB ASSIGNMENT 6

**Aim:** Write a python program to find the centrality (degree, closeness and betweenness) and prestige (degree and proximity) for the given graph:



#### Centrality –

**Procedure:**

* Firstly, we import the networkx and matplotlib libraries.
* Then we instantiate a Graph G, using networkx’s .Graph

method.

* Then using .add\_edge method we create our given graph.
* We then print our graph using matplotlib’s .show method.
* To find the degree of each node, we use degree\_centrality method of networkx.
* To find the closeness of each node, we use closeness\_centrality method of netowrkx.
* To find the betweenness of each node, we use betweenness\_centrality method of networkx.

#### Code:

#Importing Libraries import networkx as nx

import matplotlib.pyplot as plt

#Initiating and desgining graph G = nx.Graph() G.add\_edge('a','b')

G.add\_edge('a','e')

G.add\_edge('a','f')

G.add\_edge('b','c')

G.add\_edge('c','d')

G.add\_edge('c','f')

G.add\_edge('d','g')

G.add\_edge('e','i')

G.add\_edge('f','g')

G.add\_edge('f','j')

G.add\_edge('g','h')

G.add\_edge('g','j')

G.add\_edge('g','k')

G.add\_edge('h','l')

G.add\_edge('h','o')

G.add\_edge('i','j')

G.add\_edge('i','n')

G.add\_edge('i','m')

G.add\_edge('j','n')

G.add\_edge('j','o')

G.add\_edge('k','o')

G.add\_edge('l','p')

G.add\_edge('n','o')

G.add\_edge('o','p') nx.draw(G, with\_labels=True) plt.show()

#Closeness - Degree print("Node\tDegree") for node in G.nodes():

print(node, "\t", nx.degree\_centrality(G)[node])

#Closeness - Centrality print("Node\tCentrality") for node in G.nodes():

print(node,"\t", nx.closeness\_centrality(G)[node])

#Closeness -Betweeness print("Node\tBetweeness") for node in G.nodes():

print(node, "\t", nx.betweenness\_centrality(G)[node])

#### Prestige –

**Procedure:**

* Firstly, we import the networkx and matplotlib libraries.
* Then we instantiate a Graph G, using networkx’s .Graph

method.

* Then using .add\_edge method we create our given graph.
* We then print our graph using matplotlib’s .show method.
* To find degree of each node in our graph, we find the number of incoming edges to each node and then divide it by 6 (number of nodes – 1).
* To find Proximity of each node, we find the distance of that node to each of the other node, then we sum it and finally we divide it by the number of nodes.
* We then print our results.

#### Code:

#Importing Libraries import networkx as nx

import matplotlib.pyplot as plt

#Initiating and designing graph G = nx.DiGraph() G.add\_edge('1','2')

G.add\_edge('1','3')

G.add\_edge('1','4')

G.add\_edge('2','4')

G.add\_edge('2','5')

G.add\_edge('3','6')

G.add\_edge('4','3')

G.add\_edge('4','6')

G.add\_edge('4','7')

G.add\_edge('5','4')

G.add\_edge('5','7')

G.add\_edge('7','6') nx.draw(G, with\_labels=True) plt.show()

#Degree Calculation n\_nodes = 7 print("Node\tDegree") for node in G.nodes():

print(node, "\t", len(G.in\_edges(node))/(n\_nodes-1))

#Proximity Calculation distance = []

temp\_dis = 0

n = 0

for dest in G.nodes: temp\_dis = 0

n = 0

for src in G.nodes:

if (nx.has\_path(G,src,dest) == True):

temp\_dis = temp\_dis + nx.shortest\_path\_length(G,source = src,target = dest) n = n + 1

if temp\_dis == 0:

distance.append([dest, 0]) else:

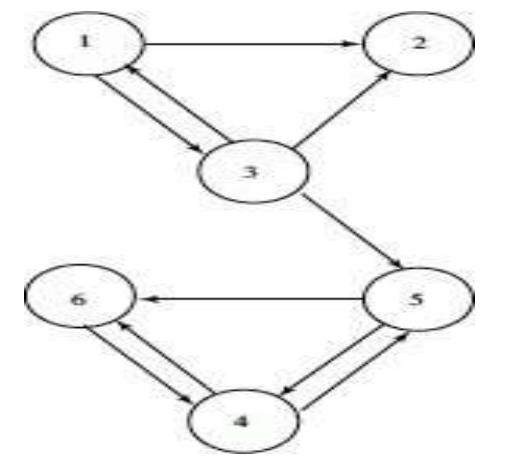
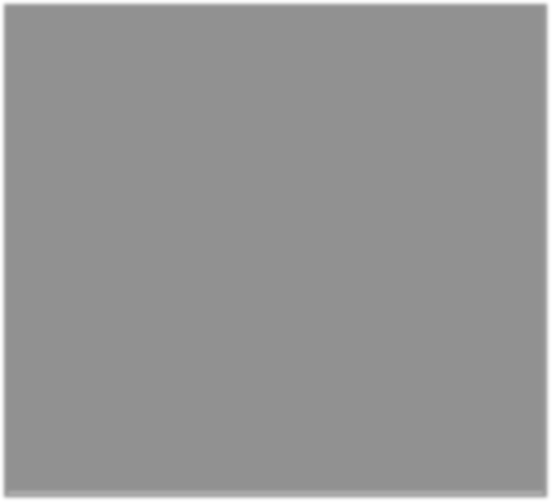
distance.append([dest, temp\_dis/(n - 1)])

print("Node\tProximity") for i in distance:

print(str(i[0]) + " \t " + str(i[1]))

# CSE-3024 WEB MINING LAB ASSIGNMENT 7

**Aim:** Write a python program to find the ranks for the given graph.



Perform **7 iteration** and print the final iteration value only.

#### Procedure:

* Firstly, we import the necessary libraries of numpy, scipy and sparse.
* Then write a compute page rank function, which takes in three input parameters, namely, links, damping factor and number of iterations.
* We initialize the damping factor to a standard 0.85 value and number of iterations to 7 as mentioned in question.
* Then we use that function to compute page rank of each page in our network.

#### Code:

#Importing libraries import scipy

from scipy import sparse import numpy

#Computing Page Rank function

def computePageRank(links, c=0.85, iteration=7): count = 0

ones = numpy.ones(len(links)) sources = [x[0] for x in links] targets = [x[1] for x in links]

n = max(max(sources), max(targets))+1

HT = sparse.coo\_matrix((ones, (targets, sources)), shape=(n,n)) num\_outlinks = numpy.array(HT.sum(axis=0)).flatten() HT.data/=num\_outlinks[sources]

d\_indices = numpy.where(num\_outlinks == 0)[0] r = numpy.ones(n)/n

while True: previous\_r = r

r = c \* (HT \* r + sum(r[d\_indices])/n) + (1.0 - c)/n #r.sum() ≈ 1 but prevent errors from adding up. r /= r.sum()

count = count+1 if(count >iteration):

#if scipy.absolute(r - previous\_r).sum() < epsilon:

return r

print(computePageRank([(0,1), (0,2), (2,0),(2, 1),(2, 4),(3, 4),(3,5),(4,3),(4,5), (5,3) ]))

# CSE-3024 WEB MINING LAB ASSIGNMENT 8

**Aim:** Classify the given network intrusion dataset into normal and anomaly using Decision Tree Classifier. Following things need to be printed along with the classification:

* Confusion Matrix
* Accuracy of model on Test data
* Decision Tree visualization.

**Dataset Used:** The network intrusion dataset from Kaggle. Link to which is:

[https://www.kaggle.com/datasets/sampadab17/network-intrusion-](https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection?select=Train_data.csv) [detection?select=Train\_data.csv](https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection?select=Train_data.csv)

#### Procedure:

* Firstly, we import the necessary libraries of numpy, pandas, matplotlib and tree.
* Next, we import the dataset into our workspace. We also define the set of independent and dependent attribute.
* Next, we split the dataset into training set and test set using a ratio of 7:3.
* Then we train our decision tree model using DecisionTreeClassifier from sklearn.tree
* Next, we find the test set results as predicted by our model.
* Then we print our confusion matrix using predicted result and test set results.
* Similarly, we print the accuracy of our model using test set result and predicted result.
* Finally, using the tree of sklearn, we visualize our model.

#### Code:

#Importing libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd

from sklearn import tree

#Importing dataset

dataset = pd.read\_csv("Train\_data.csv")

X = dataset.iloc[:, 4:41].values y = dataset.iloc[:, -1].values

#Splitting the dataset

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

#Fitting our model

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy' ,random\_state = 0) classifier.fit (X\_train, y\_train)

#Predicting the Test set Results y\_pred = classifier.predict(X\_test)

#Printing the confusion matrix

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred) print(cm)

#Printing the accuracy of our model

from sklearn.metrics import accuracy\_score accuracy = accuracy\_score(y\_test, y\_pred) print(accuracy)

#Defining the labels of our dataset classes = ["Anamoly", "Normal"]

#Printing the visualized decision tree fig = plt.figure(figsize=(25,20))

\_ = tree.plot\_tree(classifier,

feature\_names=dataset.columns, class\_names=classes,

filled=True)

#Printing the feature wise break points of our decision tree test\_representation = tree.export\_text(classifier) print(test\_representation)

# CSE-3024 WEB MINING LAB ASSIGNMENT 9

**Aim:** To verify the performance of decision tree with change in hyper-parameters

**Dataset Used:** The network intrusion dataset from Kaggle. Link to which is:

[https://www.kaggle.com/datasets/sampadab17/network-intrusion-](https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection?select=Train_data.csv) [detection?select=Train\_data.csv](https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection?select=Train_data.csv)

#### Procedure:

* Firstly, we import the necessary libraries of numpy, pandas, matplotlib and tree.
* Next, we import the dataset into our workspace. We also define the set of independent and dependent attribute.
* We use only the first 500 rows of our dataset in to get a better visualisation of small tree.
* Next, we split the dataset into training set and test set using a ratio of 7:3.
* Then we train our decision tree model using DecisionTreeClassifier from sklearn.tree
* Here we specify the criteria to entropy.
* Next, we find the test set results as predicted by our model.
* Then, we print the accuracy of our model using test set result and predicted result.
* We also train a decision tree from sklearn.tree and this time, we don’t use any hyper-parameters because, by default the criteria is set to gini.
* Again, we print the accuracy score of this classifier after predicting the y\_pred variable from X\_test result set.
* Finally, using the tree of sklearn, we visualize both the classifiers and check the difference in their spatial structure.

#### Code:

#Importing libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd

from sklearn import tree

#Importing datasets

dataset = pd.read\_csv('Train\_data.csv')

X = dataset.iloc[1:500, 4:41].values y = dataset.iloc[1:500, -1].values

#Splitting the dataset

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

#Fitting the classifier with entropy criteria from sklearn.tree import DecisionTreeClassifier

classifier\_1 = DecisionTreeClassifier(criterion="entropy", random\_state=0) classifier\_1.fit(X\_train, y\_train)

#Test set predictions

y\_pred\_1 = classifier\_1.predict(X\_test)

#Accuracy score

from sklearn.metrics import accuracy\_score accuracy\_1 = accuracy\_score(y\_test, y\_pred\_1) print("Accuracy with entropy criteria is: ", accuracy\_1)

#Fitting the classifier with gini criteria

classifier\_2 = DecisionTreeClassifier(random\_state=0) classifier\_2.fit(X\_train, y\_train)

#Test set prediction

y\_pred\_2 = classifier\_2.predict(X\_test)

#Accuracy Score

accuracy\_2 = accuracy\_score(y\_test, y\_pred\_2) print("Accuracy with gini criteria is: ", accuracy\_2)

#Printing the decision tree with entropy fig = plt.figure(figsize = (25,20))

\_ = tree.plot\_tree(classifier\_1,

feature\_names= dataset.columns, class\_names = ["anamoly", "normal"], filled =True)

#Printing the decision tree with gini fig = plt.figure(figsize = (25,20))

\_ = tree.plot\_tree(classifier\_2,

feature\_names= dataset.columns, class\_names = ["anamoly", "normal"], filled =True)

CSE3024 – WEB MINING LAB ASSIGNMENT 10

**Ques:** Illustrate the K-means clustering to cluster the data points for at least five epochs properly.

1. Use the elbow method to determine the optimal number of clusters.
2. Visualize the clusters.
3. Plot the centroids of each cluster.

**Dataset Used:** Shopping-data.csv from kaggle

#### Procedure:

* We first import the dataset into our workspace using pandas.
* We next define the set of independent attributes. Since it is

unsupervised learning, we don’t have dependent attribute.

* Next, we plot the graph of elbow method to find the optimal number of clusters.
* We then train our k-means clustering model with the optimal number of clusters as input.
* We can print the results of each input as predicted by our model, that is the cluster they belong to.
* Finally, we visualize our clusters and their centroids by plotting a scatter plot.

#### Code:

#Importing Libraries import pandas as pd import numpy as np

import matplotlib.pyplot as plt

#Importing the Datasets

dataset = pd.read\_csv('shopping-data.csv')

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

#Elbow method to find the optimal number of clusters from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10) kmeans.fit(X)

wcss.append(kmeans.inertia\_) plt.plot(range(1, 11), wcss) plt.title('The Elbow Method') plt.xlabel('Number of Clusters') plt.ylabel('WCSS')

plt.show()

#Applying Kmeans to the dataset

kmeans = KMeans(n\_clusters=5, init='k-means++', max\_iter=300, n\_init=10); y\_kmeans = kmeans.fit\_predict(X)

#Printing out the cluster each input belongs to y\_kmeans

# Visualising the clusters

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Standard Customers') plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Careless Customers') plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'cyan', label = 'Target Customers') plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'magenta', label = 'Sensible Customers')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'green', label = 'Careful Customers') plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title ('Clusters of Clients') plt.xlabel ('Annual Income (k$)') plt.ylabel ('Spending Score (1-100)') plt.legend()

plt.show()

## CSE3024 – WEB MINING LAB ASSIGNMENT 11

**Ques:** The following are the basic steps involved in performing the random forest algorithm:

1. Pick N random records from the dataset.
2. Build a decision tree based on these N records.
3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.
4. In case of regression problem, for a new record, each tree in the forest predicts a value of Y. The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

**Dataset Used:** petrol\_consumption.csv from Kaggle

#### Procedure:

* We first import the dataset into our workspace using pandas.
* Then we define the set of dependent and independent attributes.
* We then import the random forest regressor from skleanrn.ensemble and train our model using the independent and dependent attributes.
* Next, we print the results of independent set as predicted by our regressor.
* Finally, we print all the evaluation metrics to check for the performance of our dataset.

We have not split the dataset into training set and test set as the number of entries in our dataset is only 48 and is relatively insufficient to split and then train with the remaining count of entries.

#### Code:

#Importing Libraries import pandas as pd import numpy as np

import matplotlib.pyplot as plt

#Importing the Dataset

dataset = pd.read\_csv("petrol\_consumption.csv")

#First few rows of our dataset dataset.head(10)

#Checcking for null values print(dataset.info())

#Set of independent and dependent attributes X = dataset.iloc[:, 0:4].values

y = dataset.iloc[:, -1].values

#Training our Random Forest Regression Model

from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n\_estimators=200, random\_state=0) regressor.fit(X, y)

#Predictions by Regressor y\_pred = regressor.predict(X)

#Printing Mean Absolute Error

from sklearn.metrics import mean\_absolute\_error mean\_absolute\_error(y, y\_pred)

#Printing Mean Absolute Error

from sklearn.metrics import mean\_squared\_error mean\_squared\_error(y, y\_pred)

#Printing Root Mean Squared Error np.sqrt(mean\_squared\_error(y, y\_pred))

#Printing Root Mean Sqaured Log Error np.log(np.sqrt(mean\_squared\_error(y, y\_pred)))

#Printing R-square value

from sklearn.metrics import r2\_score r2\_score(y, y\_pred)

**Ques:** The following are the basic steps involved in performing the random forest algorithm:

1. Pick N random records from the dataset.
2. Build a decision tree based on these N records.
3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.
4. In case of regression problem, for a new record, each tree in the forest predicts a value of Y. The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

**Dataset Used:** bill\_authentication.csv from Kaggle

#### Procedure:

* We first import the dataset into our workspace using pandas.
* Then we define the set of dependent and independent attributes.
* We split the dataset in training set and test set with 20% of data in test set and remaining 80% of them being in training set.
* We then import the random forest regressor from skleanrn.ensemble and train our model using the independent and dependent attributes.
* Next, we print the results of independent set as predicted by our classifier.
* Finally, we print all the evaluation metrics to check for the performance of our dataset.

#### Code:

#Importing Libraries import pandas as pd import numpy as np

import matplotlib.pyplot as plt

#Importing the Dataset

dataset = pd.read\_csv("bill\_authentication.csv")

#First few rows of our dataset dataset.head(10)

#Set of independent and dependent attributes X = dataset.iloc[:, 0:4].values

y = dataset.iloc[:, -1].values

#Splitting the dataset into training set and test set from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

#Training our Random Forest Regression Model

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators=200, random\_state=0) classifier.fit(X\_train,y\_train)

#Predictions by Regressor y\_pred = classifier.predict(X)

#Printing Mean Absolute Error

from sklearn.metrics import mean\_absolute\_error mean\_absolute\_error(y, y\_pred)

#Printing Mean Absolute Error

from sklearn.metrics import mean\_squared\_error mean\_squared\_error(y, y\_pred)

#Printing Root Mean Squared Error np.sqrt(mean\_squared\_error(y, y\_pred))

#Printing Root Mean Sqaured Log Error np.log(np.sqrt(mean\_squared\_error(y, y\_pred)))

#Printing the Confusion Matrix

from sklearn.metrics import confusion\_matrix confusion\_matrix(y\_test, y\_pred)

#Printing R-square value

from sklearn.metrics import r2\_score r2\_score(y, y\_pred)