**Practical 1**

AIM**:** Write a program to demonstrate bitwise operation.

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

docs=['why hello there','omg hello pony','she went there?omg']

vec=CountVectorizer()

x=vec.fit\_transform(docs)

df=pd.DataFrame(x.toarray(),columns=vec.get\_feature\_names())

print(df)

w1=input("Enter word1: ")

w2=input("Enter word2: ")

op=input("Enter operator: ")

x=[]

for i in range(df.shape[0]):

if(op=="&"):

a=(list(df.loc[:,w1]))[i]&(list(df.loc[:,w2]))[i]

x.append(a)

if(op=="|"):

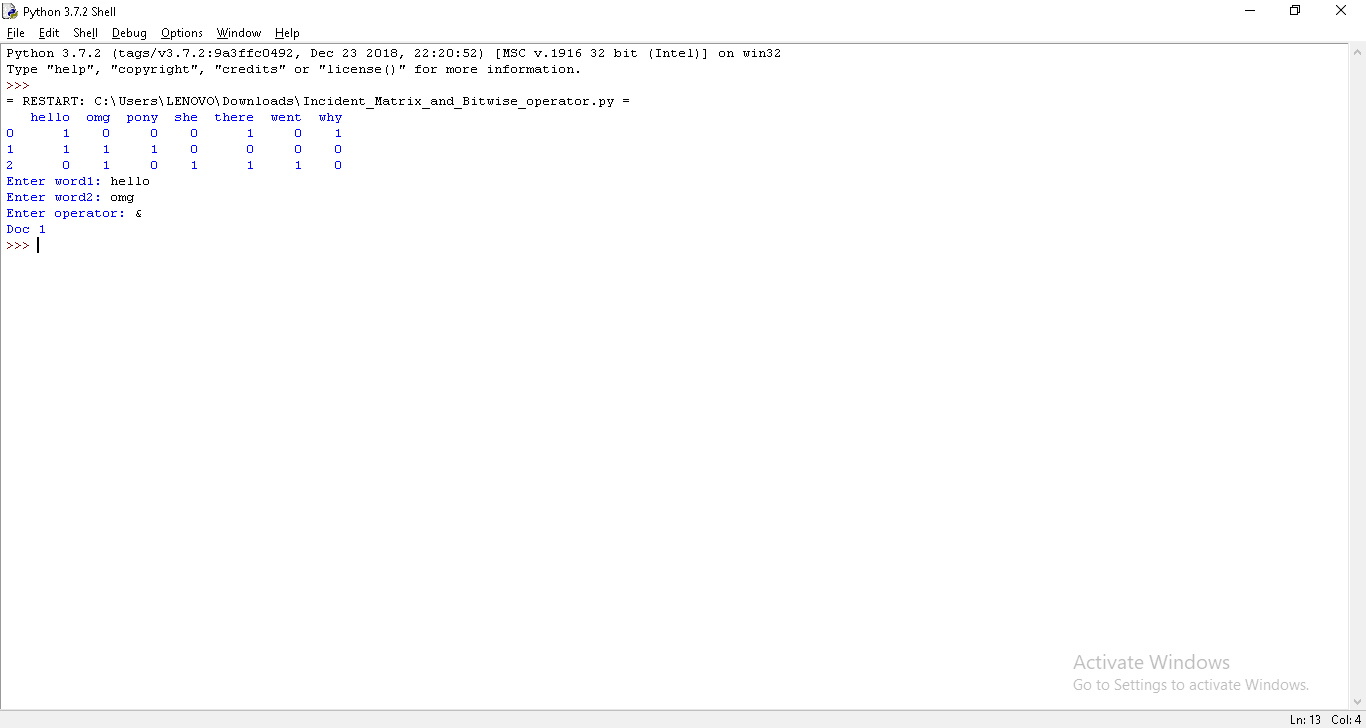
a=(list(df.loc[:,w1]))[i]|(list(df.loc[:,w2]))[i]

x.append(a)

for i in range(df.shape[0]):

if(x[i]==1):

print("Doc",i)



**Practical 2**

AIM: Implement Page Rank Algorithm.

# import some stuff

import numpy as np

from fractions import Fraction

# keep it clean and tidy

def float\_format(vector, decimal):

return np.round((vector).astype(np.float), decimals=decimal)

# we have 3 webpages and probability of landing to each one is 1/3

#(defaultProbability)

dp = Fraction(1,3)

print(dp)

# WWW matrix

M = np.matrix([[0,Fraction(1,2),Fraction(1,2)],

[1,0,0],

[1,0,0]])

print(M)

E = np.zeros((3,3))

E[:] = dp

# taxation

beta = 0.5

# WWW matrix

A = beta \* M + ((1-beta) \* E)

print(A)

# initial vector

r = np.matrix([dp, dp, dp])

r = np.transpose(r)

previous\_r = r

for it in range(1,100):

r = A \* r

print(float\_format(r,3))

#check if converged

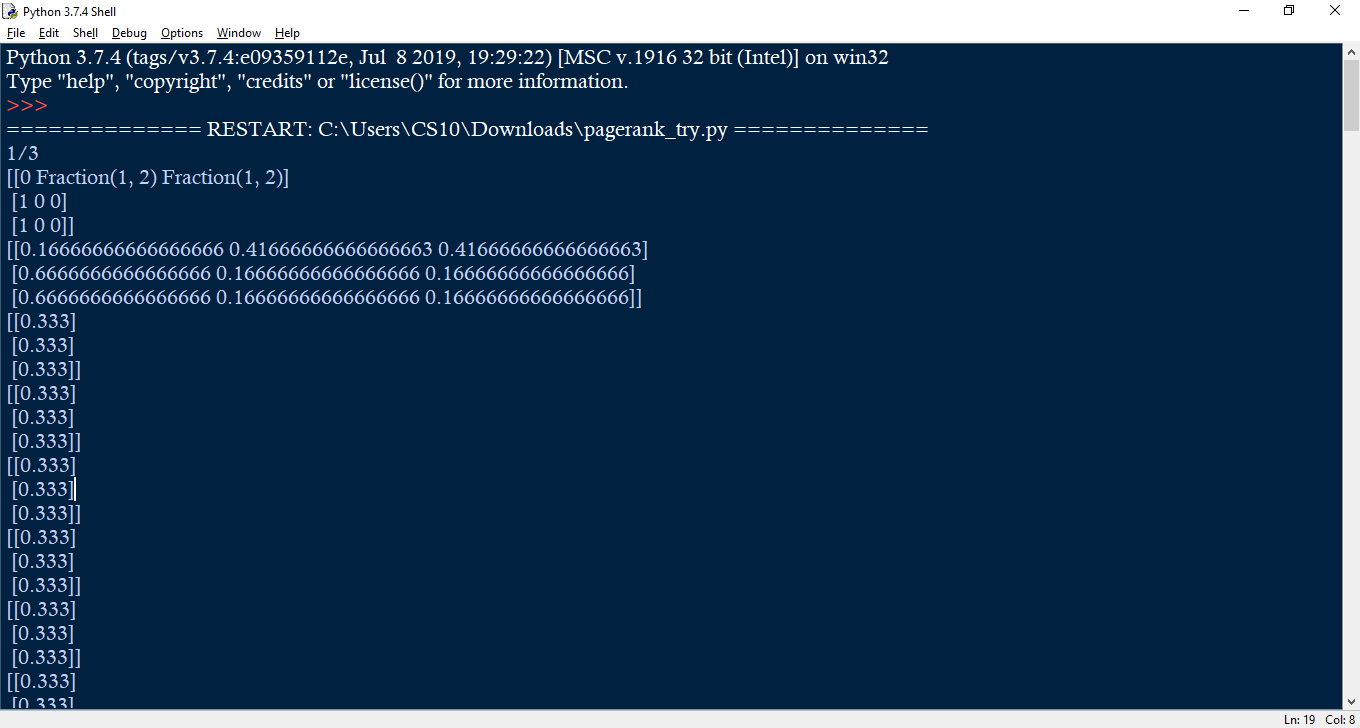
if (previous\_r==r).all():

break

previous\_r = r

print("Final:\n", float\_format(r,3))

print("sum", np.sum(r))





**Practical 3**

AIM:Implement Dynamic programming algorithm for computing the edit distance between

strings s1 and s2. (Hint. Levenshtein Distance)

m=[]

s1=input("enter word1:")

r=len(s1)+1

s2=input("enter word2:")

c=len(s2)+1

if(r==1):

print("edit distance:",c-1)

elif(c==1):

print("edit distance:",r-1)

else:

for i in range(r):

n=[]

for j in range(c):

n.append(0)

m.append(n)

for i in range(r):

for j in range(c):

m[i][0]=i

m[0][j]=j

for i in range(1,r):

for j in range(1,c):

d=m[i-1][j]+1

e=m[i][j-1]+1

if(s1[i-1]!=s2[j-1]):

f=m[i-1][j-1]+1

else:

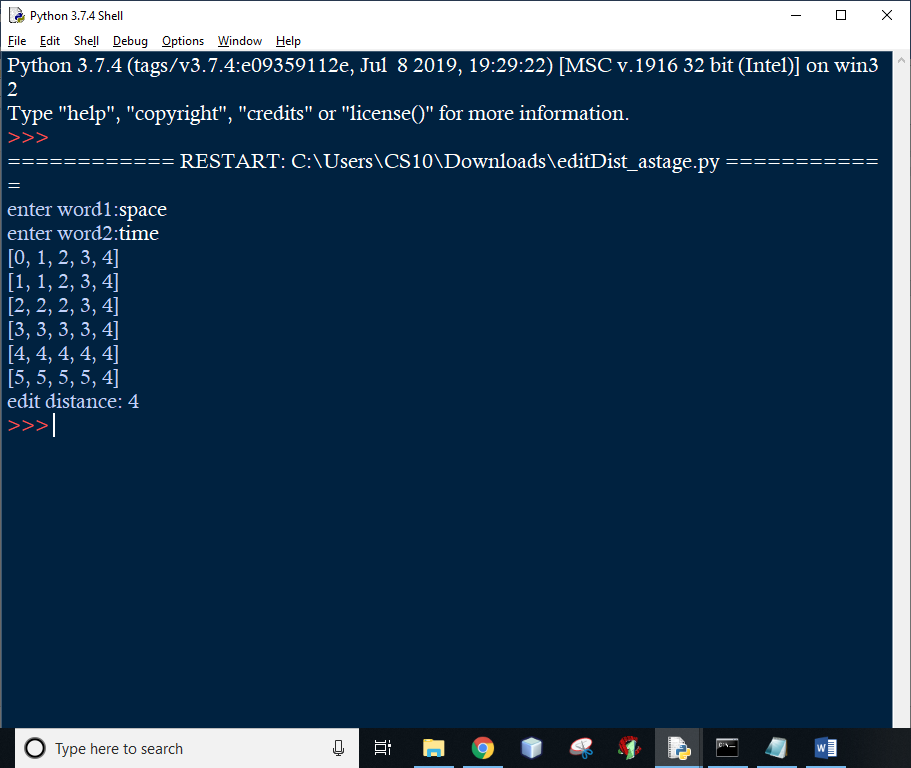
f=m[i-1][j-1]

m[i][j]=min(d,e,f)

for i in range(r):

print(m[i])

print("edit distance:",m[r-1][c-1])



Practical 4

AIM: Write a program to Compute Similarity between two text documents.

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

import numpy as np

import io

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

#word\_tokenize accepts a string as an input, not a file.

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

file1 = open("file.txt")

line = file1.read()# Use this to read file content as a stream:

words = line.split()

appendFile = open('filteredtext.txt','w')

for r in words:

if not r in stop\_words:

appendFile.write(" "+r)

appendFile.close()

file2 = open("C:\\Users\\LENOVO\\Downloads\\fileread.txt")

line = file2.read()# Use this to read file content as a stream:

words = line.split()

appendFile = open('filteredtext1.txt','w')

for r in words:

if not r in stop\_words:

appendFile.write(" "+r)

appendFile.close()

file3 = open("C:\\Users\\LENOVO\\Downloads\\file3.txt")

line = file3.read()# Use this to read file content as a stream:

words = line.split()

appendFile = open('filteredtext2.txt','w')

for r in words:

if not r in stop\_words:

appendFile.write(" "+r)

appendFile.close()

file1=open("filteredtext.txt","r")

file2=open("filteredtext1.txt","r")

file3=open("filteredtext2.txt","r")

doc1=[file1.read(),file2.read(),file3.read()]

vect=CountVectorizer()

X=vect.fit\_transform(doc1).toarray()

X[X>0]=1

df=pd.DataFrame(X,columns=vect.get\_feature\_names())

print(df)

sum\_list=[]

for j in df.index:

sum1=0

for i in range(0,len(df.columns)):

sum1+=df.iloc[j,i]\*\*2

sum\_list.append(sum1)

# print(sum\_list)

# for document 1 and document 2

theta=[]

for j in df.index:

x=0

for i in range(0,len(df.columns)):

x+=np.bitwise\_and(df.iloc[j,i],df.iloc[(j+1)%3,i])

x=x/(np.sqrt(sum\_list[j])\*np.sqrt(sum\_list[(j+1)%3]))

theta.append(x)

print(theta)

for i in range(len(theta)):

if theta[i]>0.5:

print("document ",i," and ",(i+1)%3," have cosine similarity greater than 0.5")



**Practical 5**

Aim : Write a map-reduce program to count the number of occurrences of each alphabetic character in the given dataset. The count for each letter should be case-insensitive (i.e., include both upper-case and lower-case versions of the letter; Ignore non-alphabetic characters).

from collections import Counter

# initializing string

test =input("Enter a String : ")

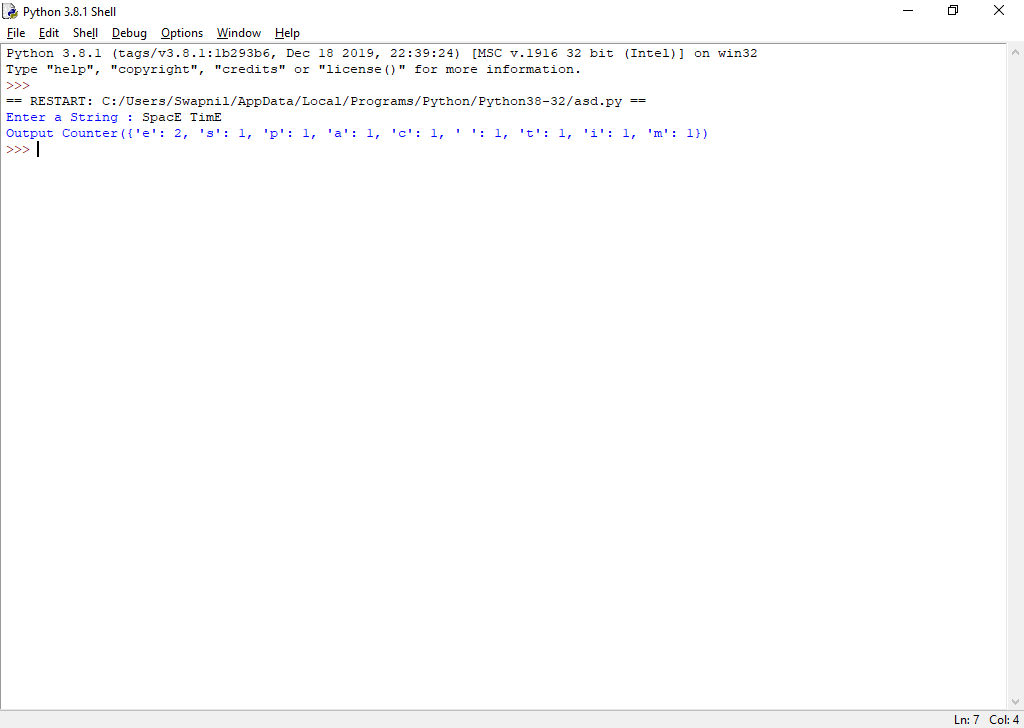
#using collections.Counter() to get

#count of each element in string

res = Counter(test.casefold())

#printing result

print ("Output " + str(res))



**Practical 7**

AIM: Write a program for Pre-processing of a Text Document: stop word removal.

import io

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

#word\_tokenize accepts a string as an input, not a file.

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

file1 = open("file.txt")

line = file1.read()# Use this to read file content as a stream:

words = line.split()

for r in words:

if not r in stop\_words:

appendFile = open('fileread.txt','a')

appendFile.write(" "+r)

appendFile.close()





**Practical 8**

AIM**:** Write a program for mining Twitter to identify tweets for a specific period and identify trends and named entities.

import tweepy

consumer\_key = "AZ3TjHaOv6dS2EJad49vxxSO2"

consumer\_secret = "NxhrLCXNBZuJCicV0yAo0aDRtWNavz4pcPW2TcIBVnci6OjtXi"

access\_token = "1230773194996367360-4qHClU7vJjgmhqntdH0UGeV5avPytt"

access\_token\_secret = "XtcXPAAQl8cudMQW55XcSrMIyF8i3VMFqERBPO6VYa8hm"

# Creating the authentication object

auth = tweepy.OAuthHandler(consumer\_key, consumer\_secret)

# Setting your access token and secret

auth.set\_access\_token(access\_token, access\_token\_secret)

# Creating the API object while passing in auth information

api = tweepy.API(auth)

# The Twitter user who we want to get tweets from

name = "TheNotoriousMMA"

# Number of tweets to pull

tweetCount = 20

# Calling the user\_timeline function with our parameters

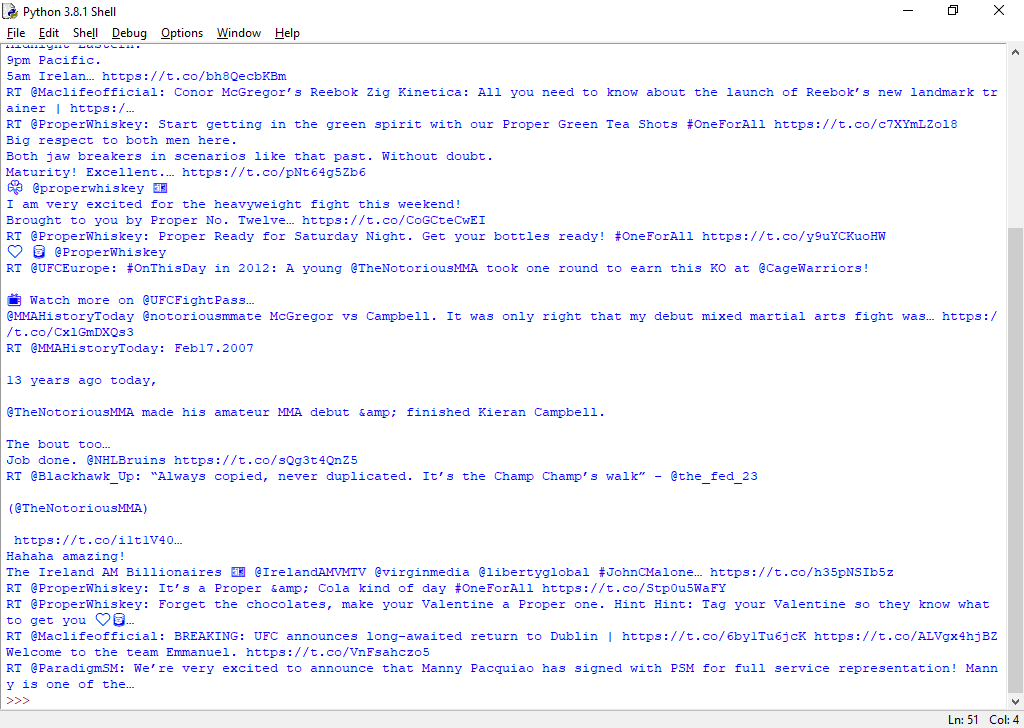
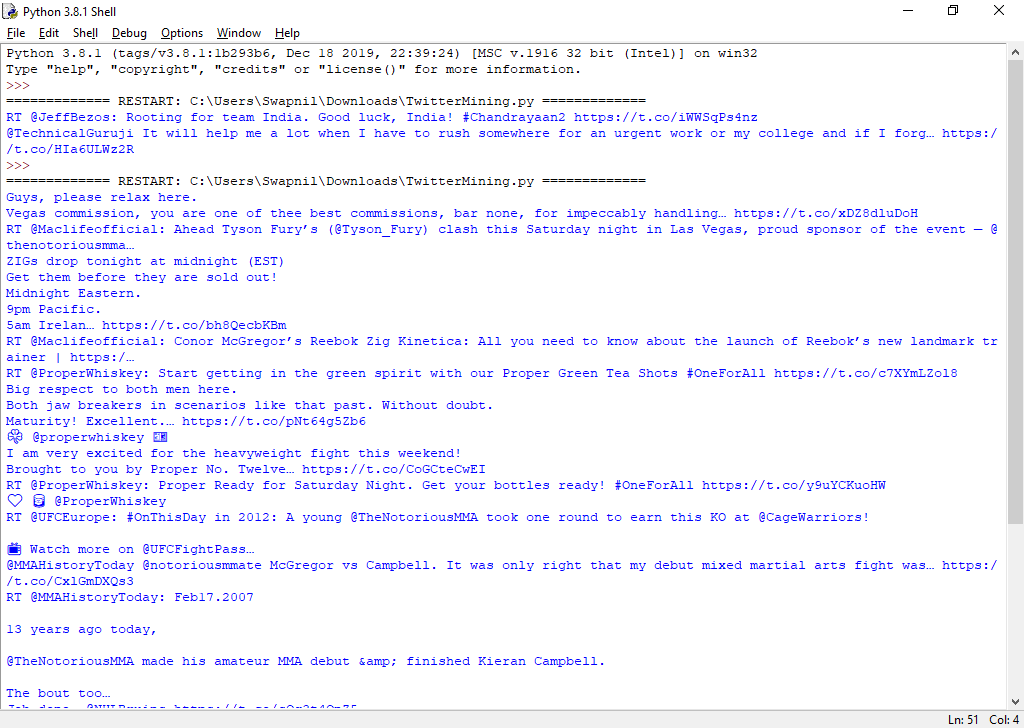
results = api.user\_timeline(id=name, count=tweetCount)

# foreach through all tweets pulled

for tweet in results:

# printing the text stored inside the tweet object

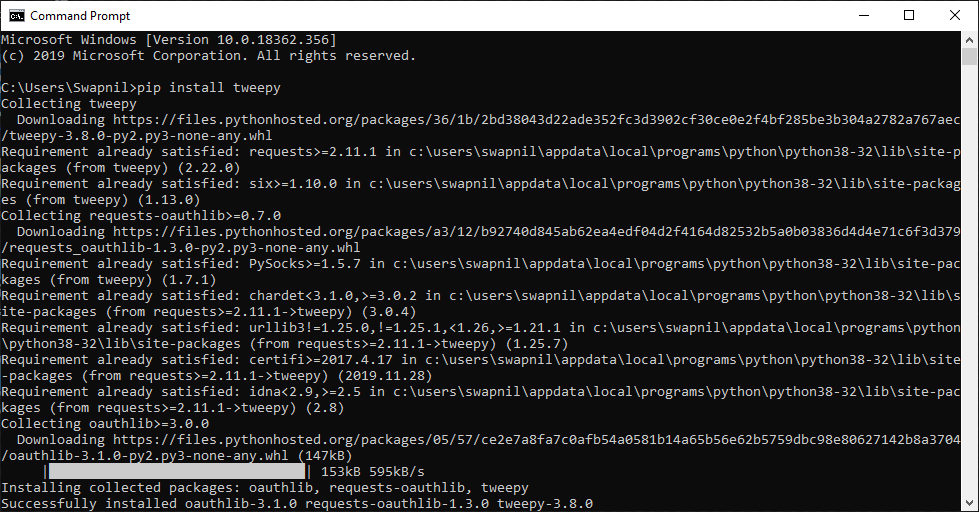
print (tweet.text)



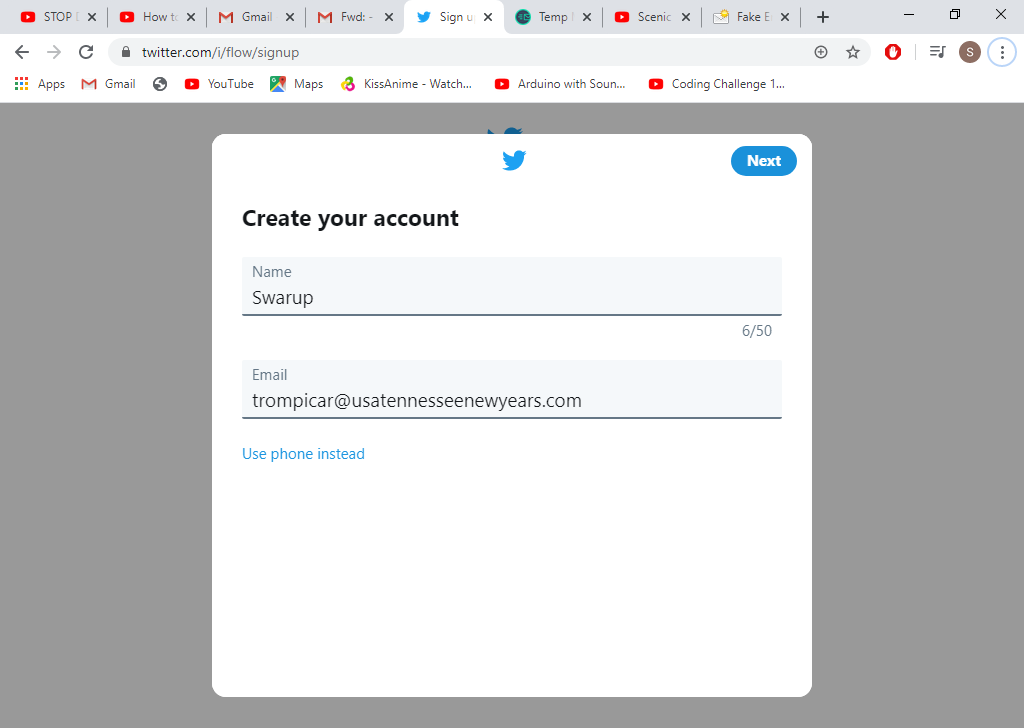


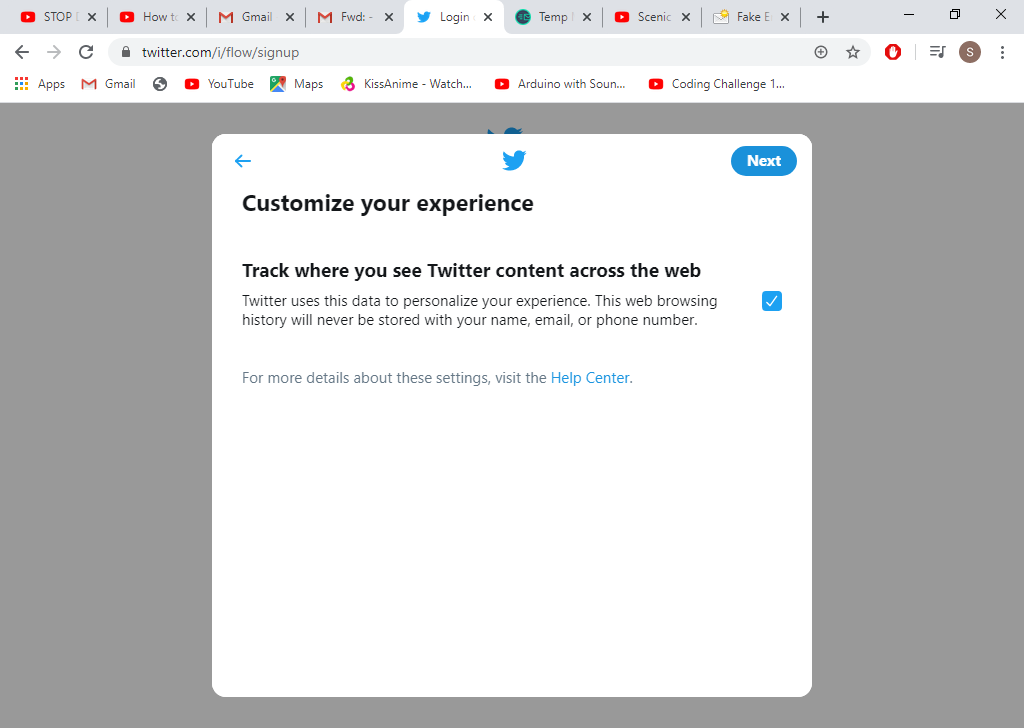
Extra Step(For Knowing):

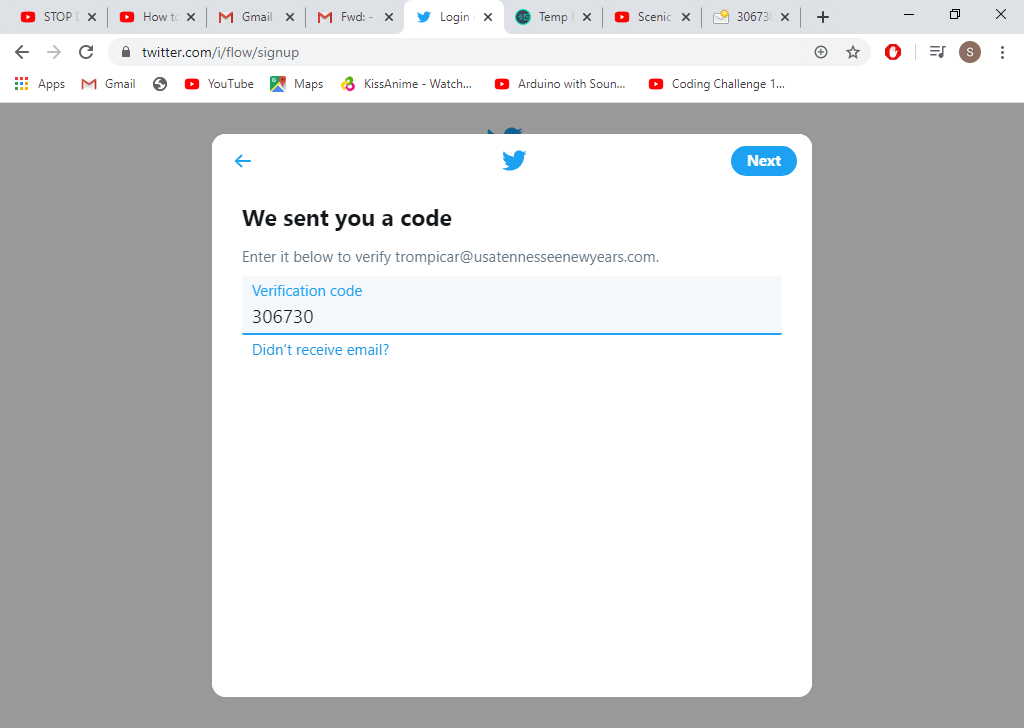
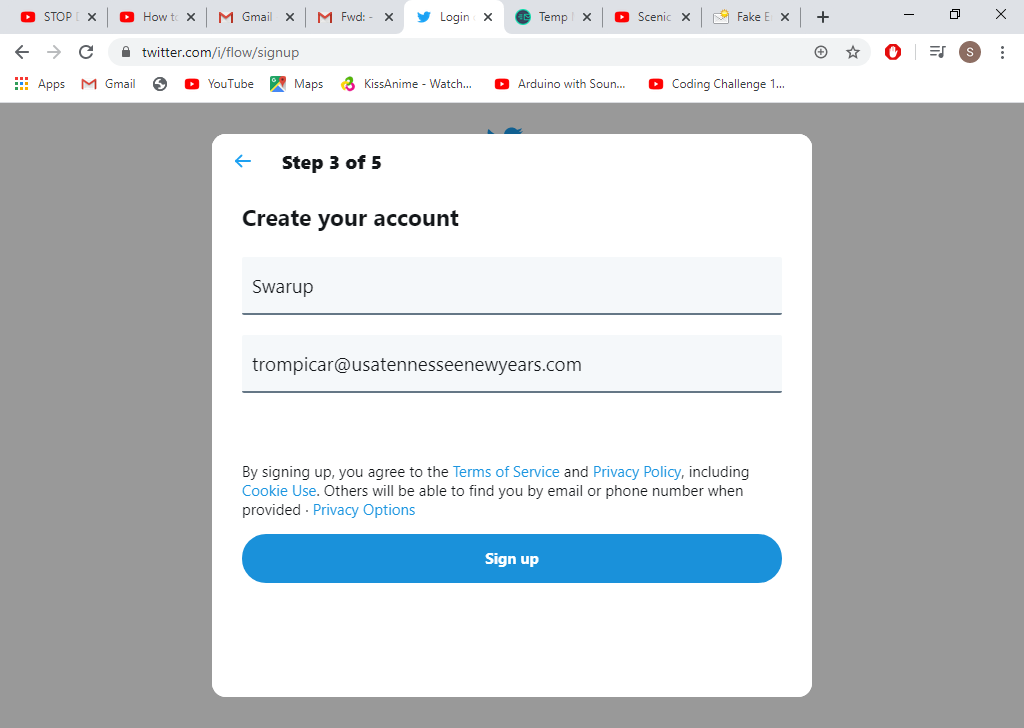
1.install tweepy

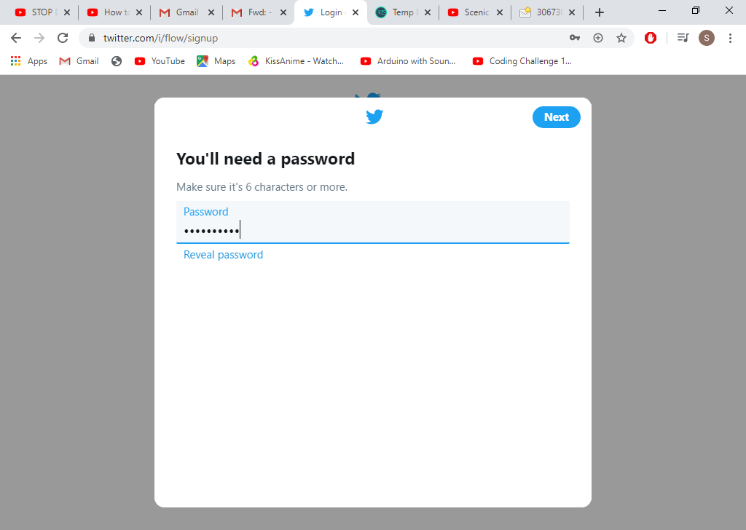


2.Create twitter account.

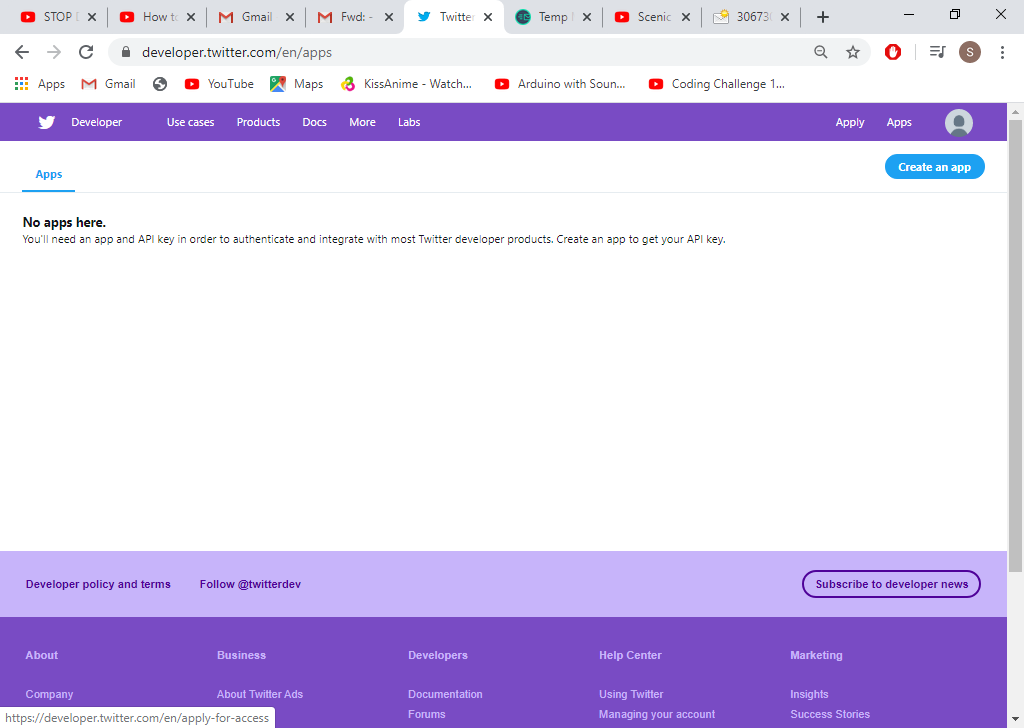




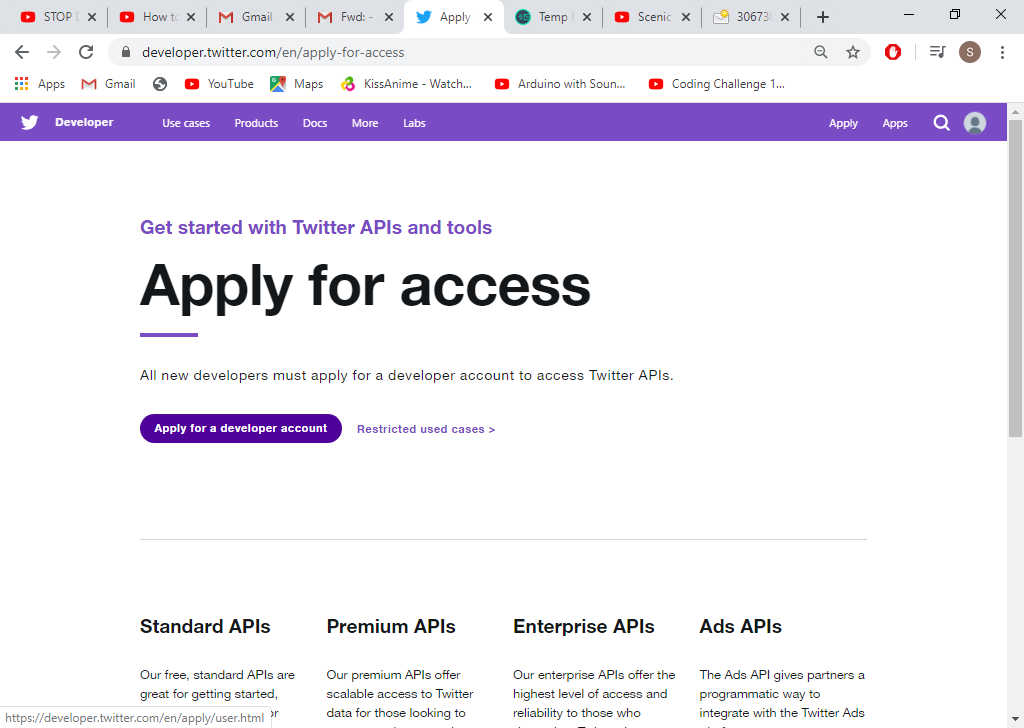


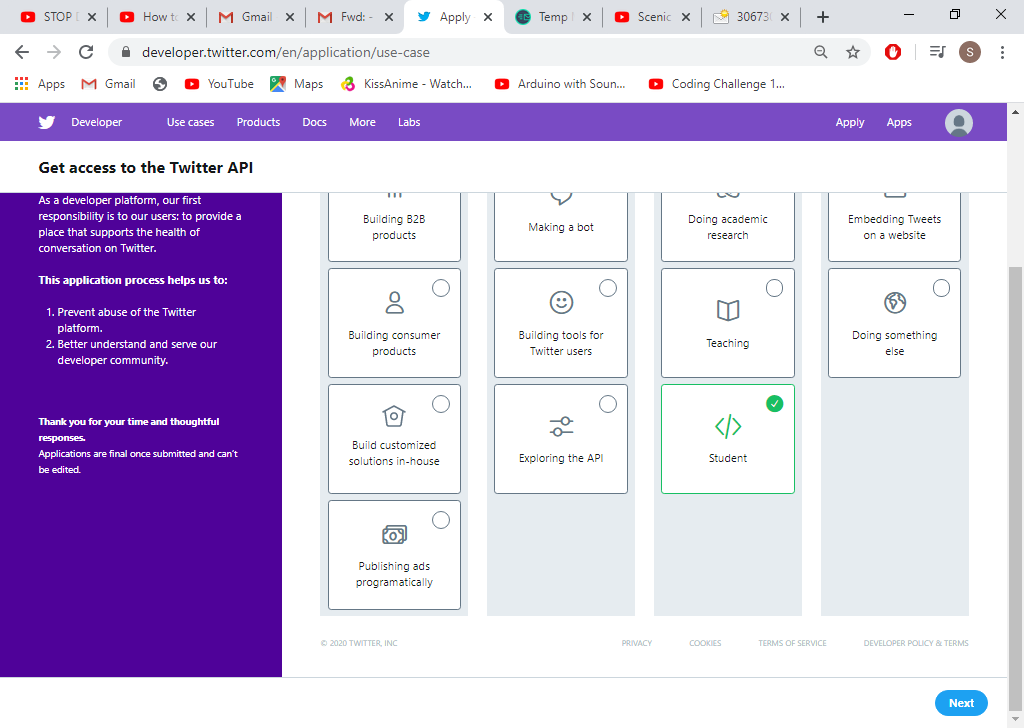


3.Create devloper account. Goto <https://developer.twitter.com/apps> or search apps twitter on google.

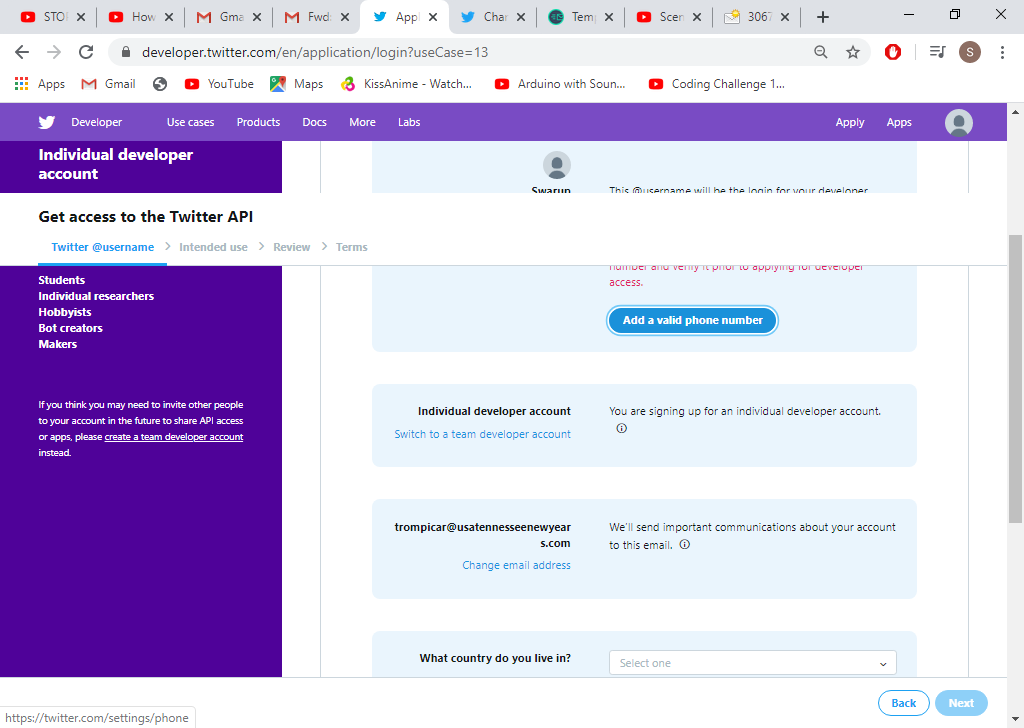


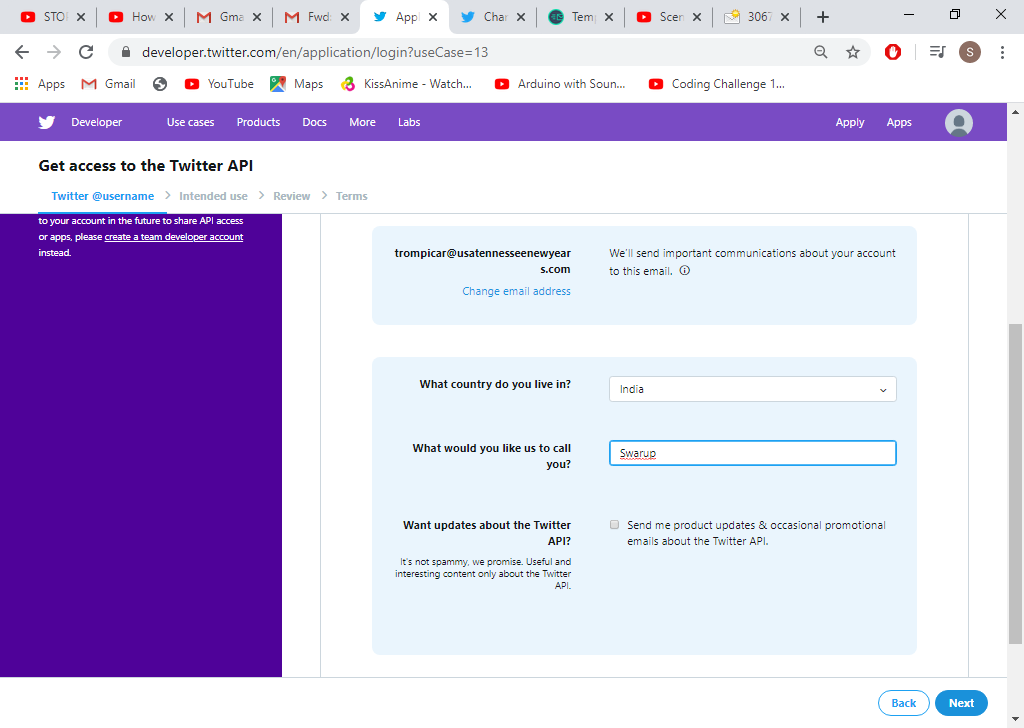
4.Click Apply for devloper account.Select Student.



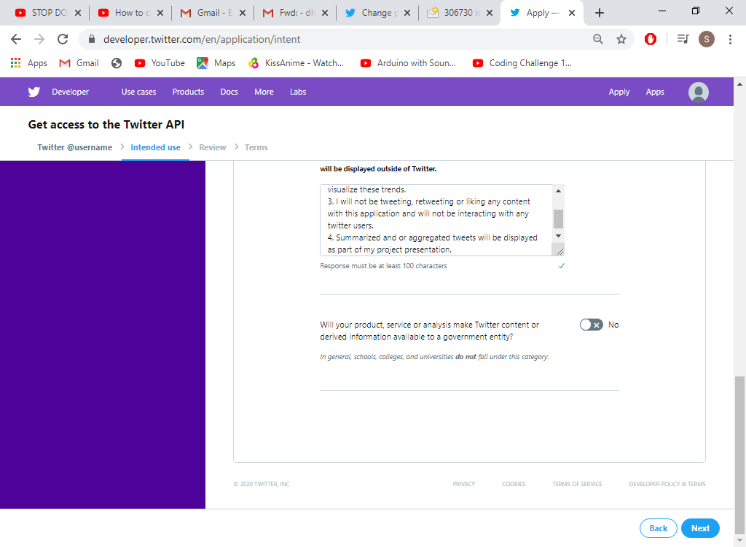
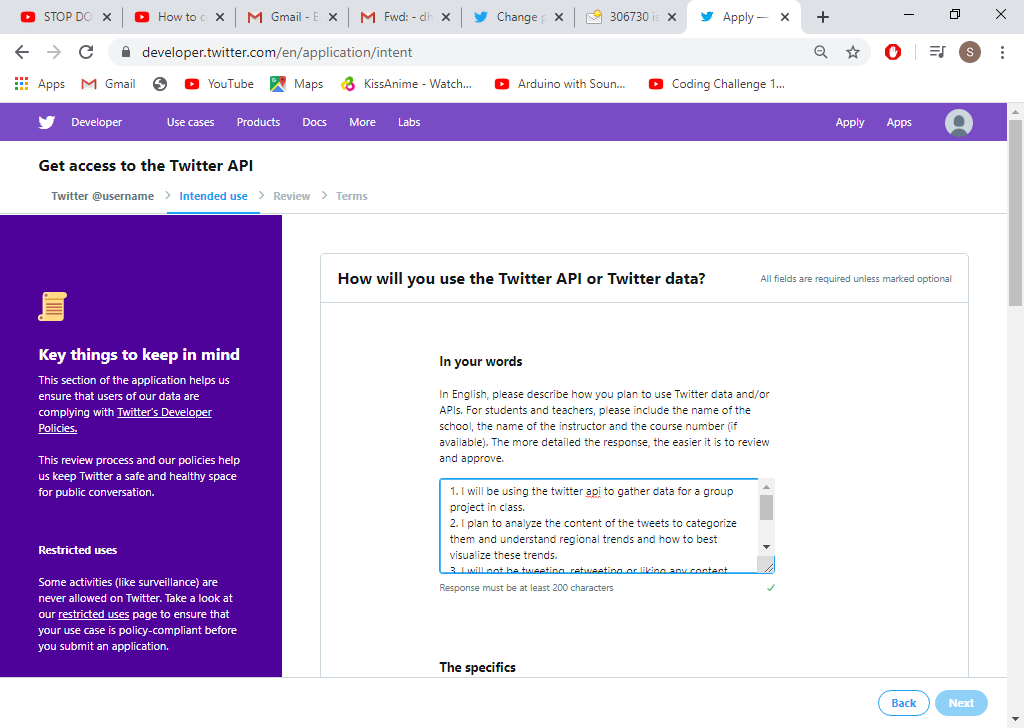


5.Verify phone number if you have not.Give all information.

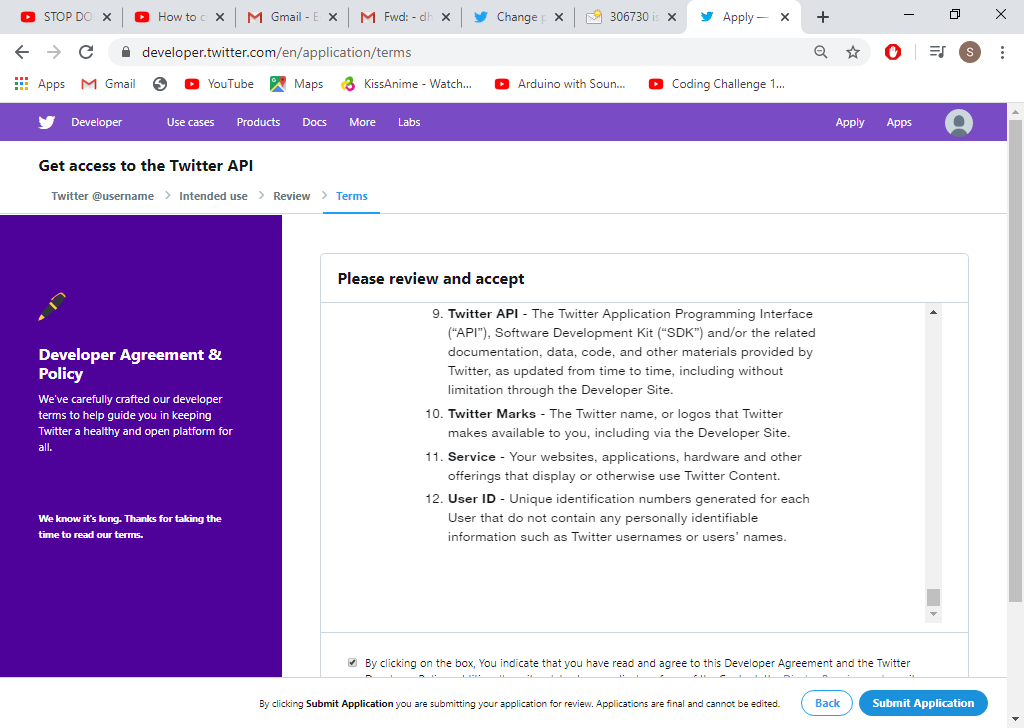


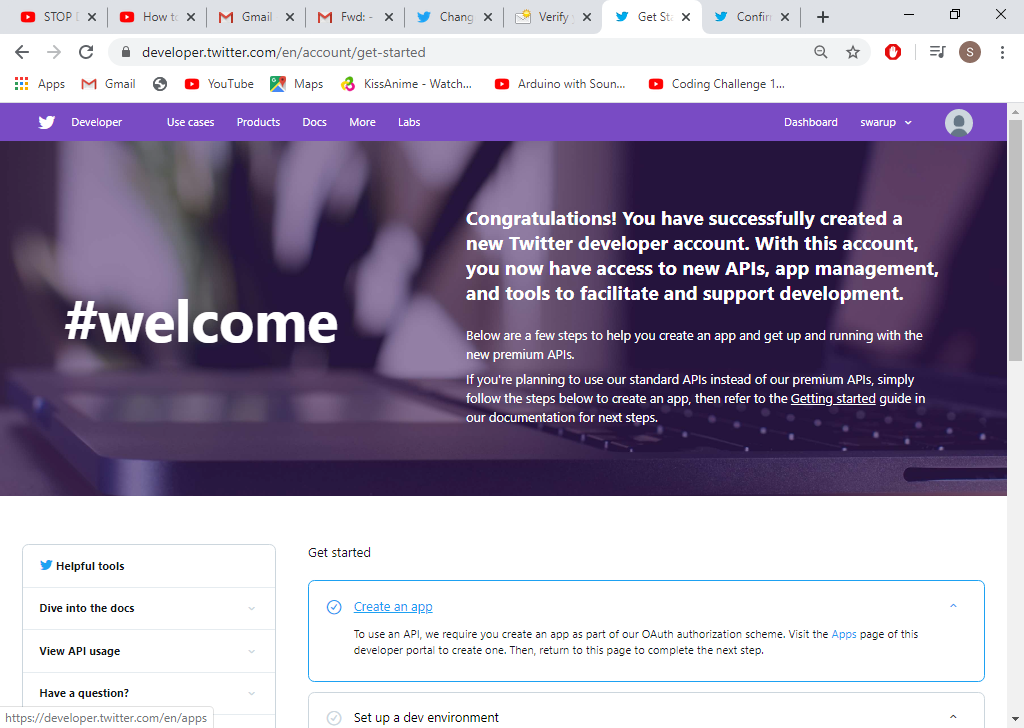


6.Write Something in your words.Deselect last option.

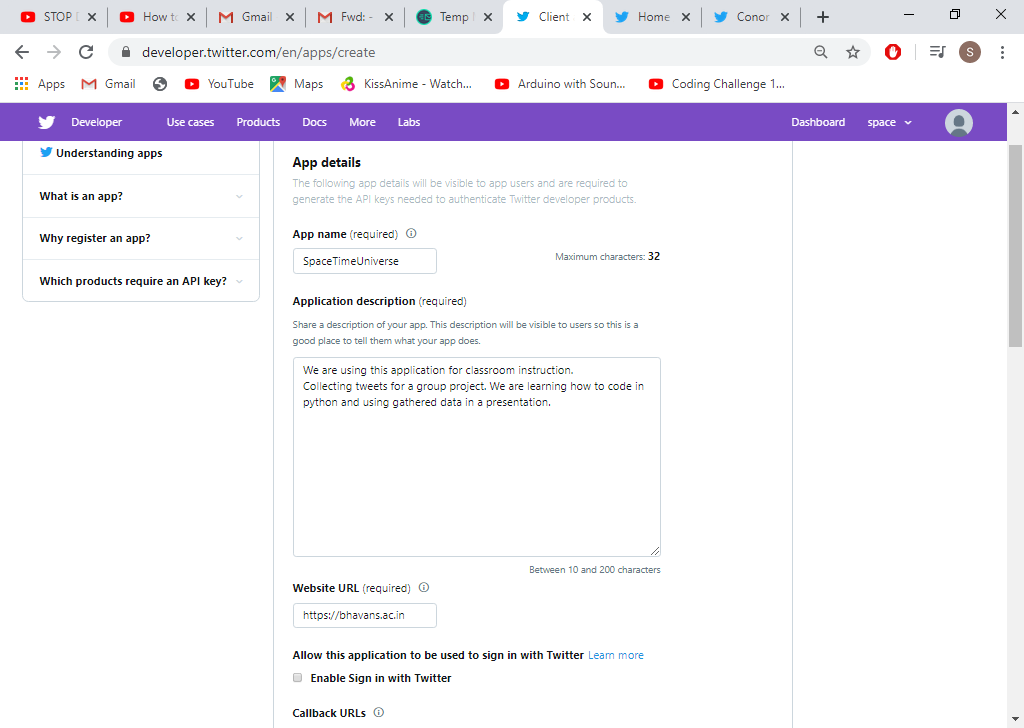


7.Submit application.After verifying on email you will be devloper.Click on Create app.

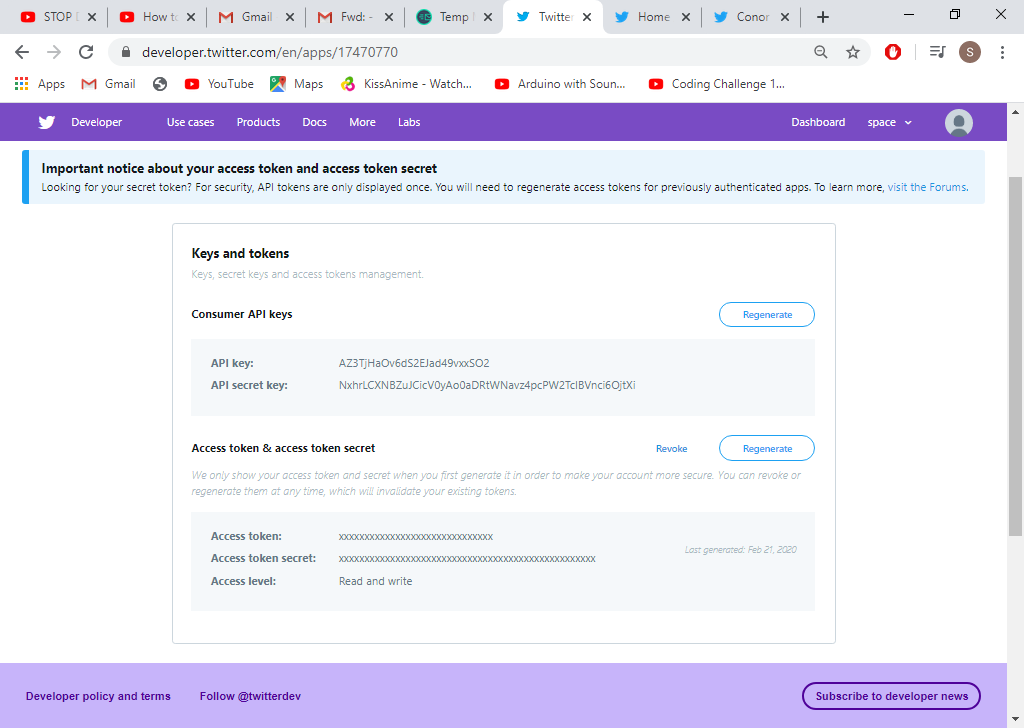




8.Give name and any website url or necessary information.



9.You will have all keys.Warning:The Access token are generated one time so copy and paste them in code.



**Practical 9**

AIM:Write a program to implement simple web crawler.

import requests

from bs4 import BeautifulSoup

def web(page,WebUrl):

if(page>0):

url=WebUrl

code=requests.get(url)

plain=code.text

s=BeautifulSoup(plain,"html.parser")

for link in s.findAll('img'):

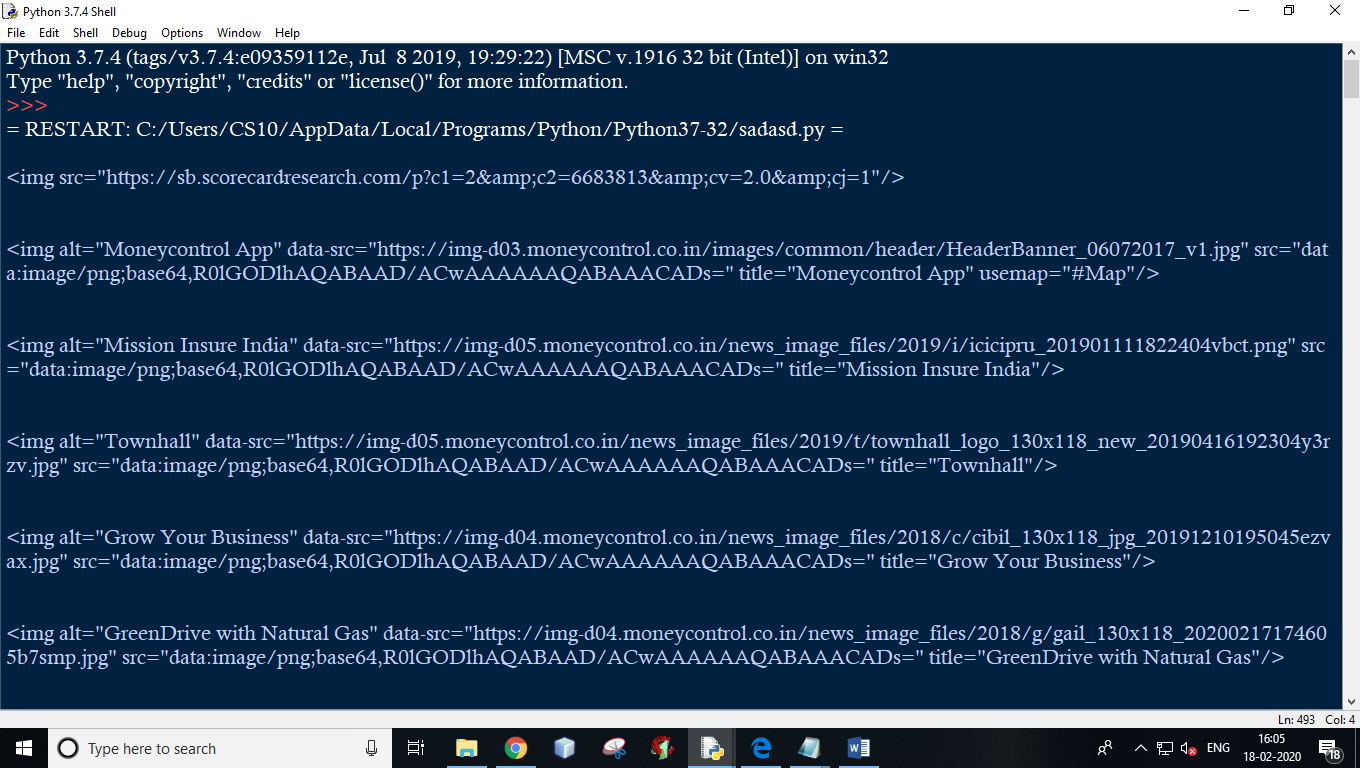
tet=link.getText()

print(tet)

print(link)

print()

web(1,"https://www.moneycontrol.com")



**Practical 10**

AIM: Write a program to parse XML text, generate Web graph and compute topic specific page rank

#Python code to illustrate parsing of XML files

# importing the required modules

import csv

import requests

import xml.etree.ElementTree as ET

def loadRSS():

# url of rss feed

url = 'http://www.hindustantimes.com/rss/topnews/rssfeed.xml'

# creating HTTP response object from given url

resp = requests.get(url)

# saving the xml file

with open('topnewsfeed.xml', 'wb') as f:

f.write(resp.content)

def parseXML(xmlfile):

# create element tree object

tree = ET.parse(xmlfile)

# get root element

root = tree.getroot()

# create empty list for news items

newsitems = []

# iterate news items

for item in root.findall('./channel/item'):

# empty news dictionary

news = {}

# iterate child elements of item

for child in item:

# special checking for namespace object content:media

if child.tag == '{http://search.yahoo.com/mrss/}content':

news['media'] = child.attrib['url']

elif child.text is not None:

news[child.tag] = child.text.encode('utf8')

# append news dictionary to news items list

newsitems.append(news)

# return news items list

return newsitems

def savetoCSV(newsitems, filename):

# specifying the fields for csv file

fields = ['guid', 'title', 'pubDate', 'description', 'link', 'media']

# writing to csv file

with open(filename, 'w') as csvfile:

# creating a csv dict writer object

writer = csv.DictWriter(csvfile, fieldnames = fields)

# writing headers (field names)

writer.writeheader()

# writing data rows

writer.writerows(newsitems)

def main():

# load rss from web to update existing xml file

loadRSS()

# parse xml file

newsitems = parseXML('topnewsfeed.xml')

# store news items in a csv file

savetoCSV(newsitems, 'topnews.csv')

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

# calling main function

main()

