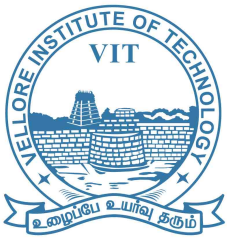


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

April, 2019

CSE-2003

DATA STRUCTURES AND ALGORITHMS

(J COMPONENT)

FINAL PROJECT REPORT

AIR POLLUTION ANALYSIS USING K MEANS CLUSTERING

PROJECT PRESENTED BY:

18BEC0715 - Sanjit C K S

Under the guidance of Prof. Dheeba J

Submitted in partial fulfilment of the degree of B.tech

in

Computer Science and Engineering

TABLE OF CONTENTS

Abstract

The project is aimed at analysing pollution data in various cities in India. The project revolves around the idea of clustering pollution data against population data in 2 Dimensions. By the approach of using K-means clustering algorithm which is highly efficient, cheap, fast and can afford a lot of iterations we cluster the numeric data (pollution and population for each daa point) and cluster the cities based on the ratio. That is, cities with similar population is to pollution will be clustered together. This type of clustering has a lot of potential real time applications and therefore the clusters give us insight about the the population and pollution in various top cities if India.

Introduction

Problem Statement:

There is increasing air pollution and population in India that hinders health and life standards of people. This is a constant threat in big metro cities in India where people want to know about the health standard of the environment they live in. There is currently no provision or application which helps people choose where they live or find a house based on the pollution and population of that region.

Main Objective:

The main objective of this project would be to *create and cluster the set of population/pollution parameters of various cities* so that it is easier to analyse the data and use it for informed decision making and business intelligence. It would enable people to get to know about a place better in terms of population/pollution ratio.

Applications:

Applications include creating an app which would enable people to know the realtime pollution/population level of a region (could be an area within a city also) and make decisions about living in that area. Respiratory patients will also find this useful as they can check if the pollution is within their tolerable range.Other applications include relaying this information to the government pollution control board (CPCB/SPCB) as a means to counter pollution that will help them know which areas to concentrate on next. This will be a form of ‘pollution monitoring’.

Existing Systems

Basically, there are two general approaches to air pollution exposure assessment in air monitoring, which depends on either direct measurements (personal monitors) or indirect measurements (fixed-site monitors combined with data on time-activity patterns).

Indirect Approach : In the indirect exposure assessment approach, average microenvironmental concentrations are multiplied by the total time spent in each microenvironment to give total integrated exposure.

Direct Approach : With study participants (people's activities are monitored) maintaining a log of their activities, then locations where highest concentrations occur as well as the nature of emission sources are inferred based on their health at the time when they were in that place.

Drawbacks:

Indirect Approach : In most cases, there is not enough information to determine which micro-environments are adequately defined, which can be bypassed or lumped with others, which should be subdivided, and which should have their limits altered to ensure accurate exposure estimates.

Direct Approach : MAJOR drawback for this approach is the lack of suitable instruments for the process.

Proposed System

I. Algorithm Adopted:

The type of clustering used in order to perform this pollution analysis, k-means clustering is used.

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters). It is a main task of exploratory data mining and statistical analysis.

K-means Algorithm:

K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem.

Initialisation :The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed appropriately (user defined) .

The main idea is to define k centres, one for each cluster. So, the better choice is to place them as much as possible far away from each other.

Process/Iterating: The next step is to take each point belonging to a given data set and associate it to the nearest centre. When no point is pending, the first step is completed and an early group age is done.

At this point we need to re-calculate k new centroids as barycenter(mean/average) of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centre. A loop has been generated.

Termination: As a result of this loop we may notice that the k centres change their location step by step until no more changes are done or in other words centres do not move any more.

Finally, for centroid reassignment:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\left\| x_i - v_j \right\| \right)^2$$

where,

‘||xi - vj||’ is the Euclidean distance between xi and vj.

‘ci’ is the number of data points in ith cluster.

‘c’ is the number of cluster centres.

Algorithmic steps for k-means clustering :

Let X = {x1,x2,x3,……..,xn} be the set of data points and V = {v1,v2,……..,vc} be the set of centres.

1) Randomly select ‘c’ cluster centres. (Initialise)

2) Calculate the distance between each data point and cluster centers.(Iterate/process)

3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.(Iterate/ process)

4) Recalculate the new cluster center using:

$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i$$

where, ‘ci’ represents the number of data points in ith cluster. (Basically we take mean/average)

5) Recalculate the distance between each data point and new obtained cluster centers.

6) If no data point was reassigned then stop, otherwise repeat from step 3(Termination)

Advantages

1) Fast, robust and easier to understand.

2) Relatively efficient: O(tknd), where n is # objects, k is # clusters, d is # dimension of each object, and t is # iterations. Normally, k, t, d << n.

3) Gives best result when data set are distinct or well separated from each other (Numeric Data Only).

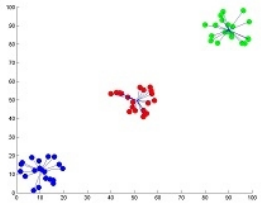


Fig I: Showing the result of k-means for 'N' = 60 and 'c' = 3 for a arbitrary dataset

Air Pollution Analysis as a potential application for k-means:

Air pollution affects body organs and human systems in addition to the environment. Smart air pollution monitoring consists of wireless sensor nodes, server and a database to store the monitored data. Huge amounts of data are generated by gas sensors in air pollution monitoring system.

Traditional methods are too complex to process and analyze the voluminous data. The heterogeneous data are converted into meaningful information by using data mining approaches for decision making. The K-Means algorithm is one of the frequently used clustering method in data mining for clustering massive data sets.

II. Explanation of Code and Methods Adopted

A) Algorithmic Implementation :

- Kmeans algorithm is implemented in such a way that ‘K’ can be defined by the user.
- 2-Dimensional clustering is adopted here because 1 dimensional clustering is not viable and not recommended. The 2 Dimensions adopted here are POPULATION and POLLUTION of the the top 200 most populated cities in India.
- The Algorithm computes and gives the clusters of cities with similar population is to pollution ratio as ‘K clusters’.
- The entire algorithm is coded in a modular way (separate files for each case).
- The algorithm and approach are in three phases:
    1. Input data processing (dataprep)
    2. Data processing (actual k means algorithm implementation)
    3. Output data process (visual output and result)
- Not only are the K clusters on cities with similar ratio shown, they are visually represented using Jupyter notebook, that is with matplotlib from pythonpy.

What is Jupyter?

Jupyter notebook is a web based interactive programming and computing environment that allows to edit and run human readable code while doing data analysis.

B) WORKING OF CODE :

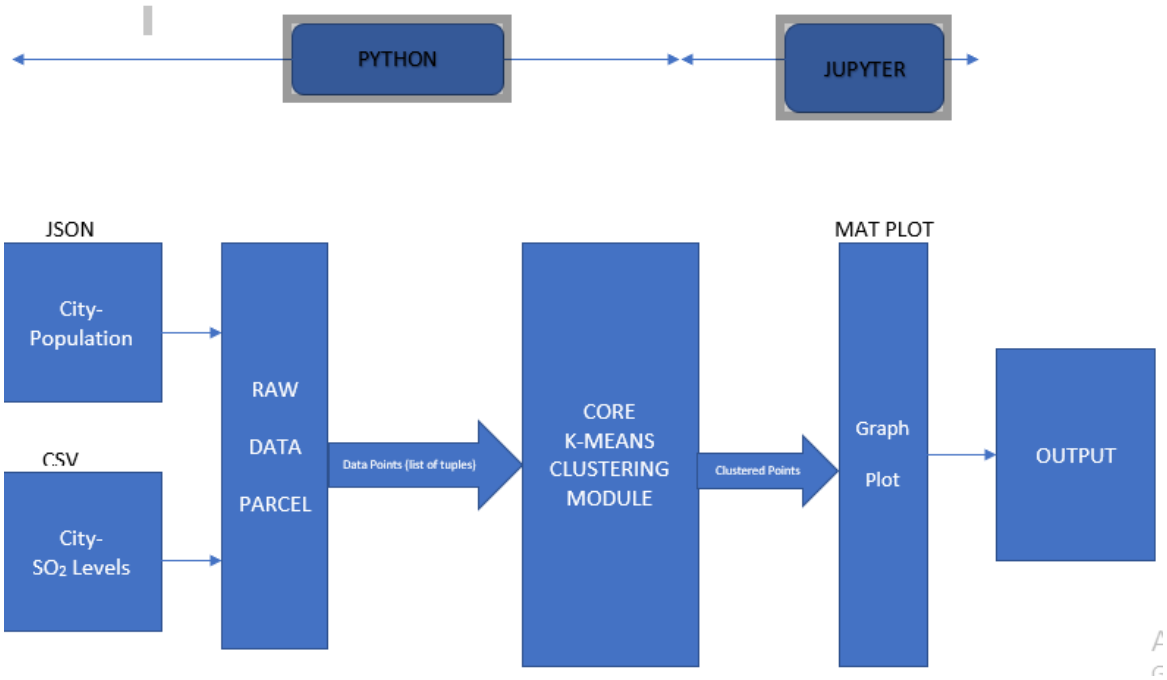
Note: Individual functions working are explained after the code.

- In this project we have attempted to cluster air pollution data (Sulphur dioxide levels in different cities of India - 2012 statistics) that is available open source on the government website data.gov.in in the form of a csv file.
- Here each city is a datapoint and has two parameters - population and SO2 particulate matter level (in μg/m3) (pollution).
- The population data was obtained for the top 200 most populated cities from wikipedia (screen-scraped into a .txt file)and parsed into a text file and then into a json file.
- The the particulate matter statistics was obtained in csv format from data.gov.in.
- Data Parcel : The raw input here was created from this as data points (for each city - pollution and population level) - (a dictionary is created with population as key and pollution s value)

- This input is given to the kmeans clustering algorithm and the the output clusters are generated (explained how in functions explanation). The number of clusters generated can be decided by the user (2 to 8 for effective clustering because of limited number of datapoints used here).

- For better understanding and visualisation these clusters are plotted as a scatter graph in Jupyter notebook using matplotlib in pythonpy.
- The final output will be the cities clusters that have been clustered together.

C) BLOCK DIAGRAM EXPLAINING CODE'S FUNCTIONING:



D) OTHER METHODS INCORPORATED :

Development Environment:

Python 3.7.0

Jupyter notebook for final execution

Source Code Management :

The code has been regularly debugged and updated in an online repository for source code management throughout the project in bitbucket.org .

The link for the repository is pasted below in the hope of proving legitimacy of the project work.

Repository : https://sanjit_77@bitbucket.org/sanjitcks2000/vit-project.git

Unit Testing:

The code's functions have been individually unit tested for the validation of the particular function. The code has been added and edited based on certain exceptions and test cases. This has also been performed in python under clustering_unittest.py.

III. CODE

INPUT (input data process in python) :
named: input_data_processing.py
Note: air_quality_PM10_2012.csv and city_population.json are the input files used.

```
Import csv
import json

def prep_data():
    with open (“city_population.json”) as read_json:
        data  = json.load(read_json)

    with open (‘air_quality_PM10_2012.csv’,encoding=”utf-8”,
errors=”ignore”) as csvfile:
        readCSV = csv.reader(csvfile, delimiter=’,’)
        population_dict = dict()
        pm10_dict = dict()
        for row in readCSV:
            if row[1] in data:
                if row[9] != “Null”:
                    pm10_dict[row[1]] = row[9]
                    population_dict[row[1]] = data[row[1]]

        pop_pm = list()
        for city in population_dict:
            pop_pm.append((int(population_dict[city].replace(“,”,””)),
                           int(pm10_dict[city])))

        return pop_pm,population_dict
```

MAIN MODULE: (named : clustering_main.py)

```
import Constant
import random
import clustering_functions
import input_data_processing

def main(k, pop_pm):
    #clean data for use

    pom_pm = input_data_processing.prep_data();

    centroid=[]

    #Random Centroid Generation
    for I in range(0,k):
        centroid.append(
                (random.randint(Constant.MIN_POP,Constant.MAX_POP),
                random.randint(Constant.MIN_PM,Constant.MAX_PM)))

    #cluster initialisation
```

```
    cluster_dict = {i: [] for I in range(k)}

    #initialise prev centroid list
    prev_centroid=[]

    #call the cluster function

cluster_dict,cen_res=clustering_functions.cluster(prev_centroid,centroid
                                                  ,pop_pm,cluster_dict,k)

    return cluster_dict,cen_res
```

CLUSTER FUNCTIONS MODULE : (named : clustering_functions.py)

```
import math
import copy
import random
import constant

def edist(A,B):
    dist=math.sqrt((A[0] - B[0])**2 + (A[1] - B[1])**2)
    return round(dist,2)


def calc_avg(dp_list):
    #if cluster has no tuples, create a random tuple and move forward
    if(len(dp_list) == 0):
        return(random.randint(Constant.MIN_POP,Constant.MAX_POP),
                 random.randint(Constant.MIN_PM,Constant.MAX_PM))

    sum1=0
    sum2=0
    for I in range(len(dp_list)):
        sum1+=dp_list[i][0]
        sum2+=dp_list[i][1]
    x_avg=sum1/len(dp_list)


    y_avg=sum2/len(dp_list)
    return (round(x_avg),round(y_avg))


def diff(prev,curr):
    for I in range(0,len(prev)):
        x=prev[i][0]-curr[i][0]
        y=prev[i][1]-curr[i][1]

def alloc_dp_to_cluster_no(pop_pm,centroid,cluster_dict,k):
    for I in range(0,len(pop_pm)):
        temp=[]
        for j in range(0,k):
            temp.append(edist(pop_pm[i],centroid[j]))
        cen_number=temp.index(min(temp))
        cluster_dict[cen_number].append(pop_pm[i])
    return cluster_dict
```

```
def calc_new_centroid(centroid,cluster_dict,k):
        for I in range(0,k):
            centroid[i] = calc_avg(cluster_dict[i])
        return centroid


def cluster(prev_centroid,centroid,pop_pm,cluster_dict,k):
    count_while=0
    while(prev_centroid!=centroid):
        #Clean Original Centroid dict
        for I in range(k):
            cluster_dict[i]=[]

        #Euclidean Distance between the Centroid and DataPoints
             cluster_dict=alloc_dp_to_cluster_no(pop_pm,centroid
                                                 ,cluster_dict,k)

        #Prev Centroid temp
        prev_centroid=copy.deepcopy(centroid)

        #Calculate New Centroids from the lists
        centroid=calc_new_centroid(centroid,cluster_dict,k)

        count_while+=1
        diff(prev_centroid,centroid)

    return cluster_dict,centroid
```

CONSTANT (named: Constant.py) :

```
MIN_PM = 5
MAX_PM = 150
MIN_POP = 1
MAX_POP = 20
```

CLUSTERING FINAL OUTPUT CITIES (named: clustering_result.py) :

```
def result(cluster_dict,population_dict):
    population = list(population_dict.values())
    for I in range(len(population)):
        population[i]=int(population[i].replace(“,”,””))
    city = list(population_dict.keys())
    clustered_city={}

    for j in range(len(cluster_dict.keys())):
        curr_cluster_list_tup=cluster_dict[j]
        temp_list=[]
        for I in range(len(curr_cluster_list_tup)):
            curr_cluster_pop=curr_cluster_list_tup[i][0]
            temp_list.append(city[population.index(curr_cluster_pop)])
        clustered_city[j]=temp_list
    return clustered_city
```

JUPYTER MAIN EXECUTION AND VISUAL OUTPUT (named: cluster_plot.ipynb) :

jupyter cluster_plot Last Checkpoint: 8 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

In [1]:
```
import clustering_main
import input_data_processing
import clustering_result
```

In [2]:
```
k=7
```

In [3]:
```
pop_pm,population_dict=input_data_processing.prep_data()
```

In [4]:
```
cluster_dict_1,cen=clustering_main.main(k,pop_pm)
```

In [5]:
```
print(cluster_dict_1)
```

```
{0: [(5577940, 77), (4467797, 95), (4646732, 45), (4496694, 153)], 1: [(169578, 64), (174164, 54), (172878, 57), (220
257, 64), (101520, 70), (184000, 53), (177658, 116), (228832, 149), (200827, 238)], 2: [(957352, 94), (961587, 118),
(1010087, 208), (1286678, 79), (1162472, 206), (1073427, 202), (887446, 57), (1069276, 346), (1055525, 75), (951558,
83), (1132383, 210), (862196, 148), (1033918, 179), (1017865, 49), (831038, 60), (1112544, 287), (1305429, 129), (119
8491, 140), (1077075, 186)], 3: [(1684222, 196), (1670806, 141), (1414050, 170), (1798218, 127), (1964086, 139), (148
6053, 94), (1841488, 60), (1618879, 246), (1585704, 174), (1648643, 244)], 4: [(529308, 101), (631364, 149), (488292,
72), (499486, 31), (601574, 63), (752490, 51), (515215, 54), (646801, 117), (460468, 146), (549283, 126), (506937, 12
1), (537149, 157), (550564, 79), (606007, 57), (603797, 205), (507293, 96), (642381, 139), (703345, 207), (578420, 19
7), (564491, 105), (566937, 108)], 5: [(325985, 58), (430214, 74), (363210, 84), (349033, 43), (432097, 54), (289438,
100), (273357, 120), (321036, 173), (382754, 144), (285349, 123), (291822, 29), (355823, 80), (273217, 97), (405164,
104), (315310, 152), (451735, 212), (349336, 220)], 6: [(2405665, 57), (3124458, 89), (3046163, 234), (2765348, 260),
(2817105, 215)]}
```

jupyter cluster_plot Last Checkpoint: 8 hours ago (autosaved)

Logo

File Edit View Insert Cell Kernel Widgets Help

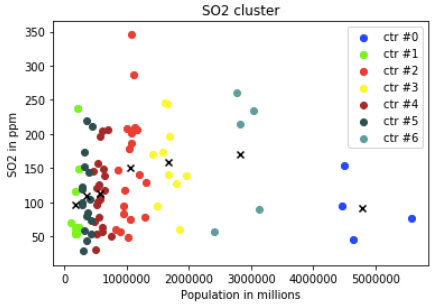
Trusted | Python

Run Code

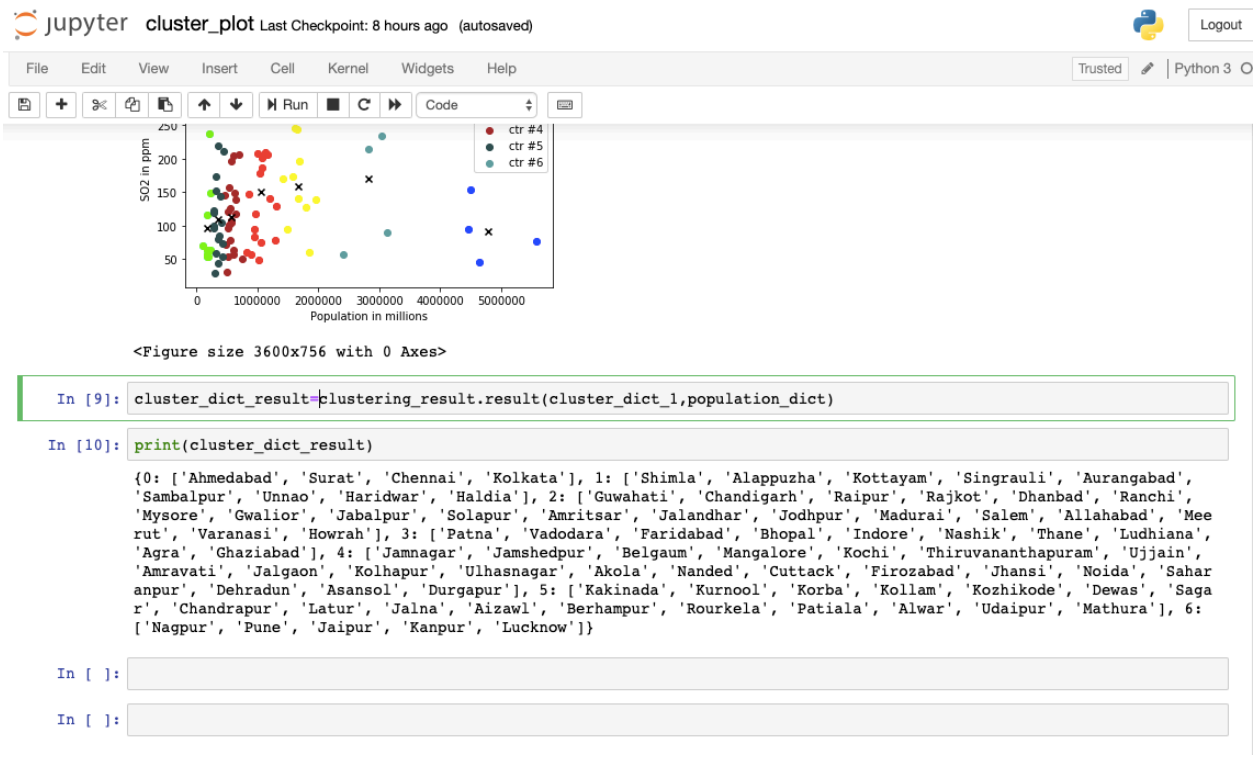
In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [7]:
```
colors=["#0000FF", "#00FF00","#FF0000","#FFFF00","#A52A2A","#2F4F4F","#5F9EA0","#808000","#FFA500","#F9D045","#54FEDA"]
```

In [8]:
```
fig,ax=plt.subplots()
for i in range(k):
    ax.scatter(*zip(*cluster_dict_1[i]), color=colors[i],label='ctr #'+str(i))
plt.scatter(*zip(*cen),marker='x',color="#000000")
ax.legend()
plt.title('SO2 cluster')
plt.xlabel('Population in millions')
plt.ylabel('SO2 in ppm')
plt.show()
fig = plt.gcf()
fig.set_size_inches(50, 10.5)
fig.savefig('test2png.png', dpi=100)
```




```
<Figure size 3600x756 with 0 Axes>
```



UNIT TESTING (all functions are unit tested, file name : clustering_unittest.py):

```
import unittest
import math
from clustering_functions import
edist,calc_avg,alloc_dp_to_cluster_no,calc_new_centroid,cluster
from clustering_result import result

class TestUM(unittest.TestCase):

    def setUp(self):
        pass

    def test_edist(self):
        self.assertEqual(edist((1,2), (3,4)),2.83)

    def test_calc_avg(self):
        list1=[(2,1),(3,1),(4,4)]
        self.assertEqual(calc_avg(list1),(3,2))
        list2=[(1,1),(7,4),(8,1)]
        self.assertEqual(calc_avg(list2),(5,2))


    def test_alloc_dp_to_cluster_no(self):
        pop_pm=[(1,2),(1,3),(3,4),(3,3),(6,1),(6,3),
                                (7,2),(8,3),(2,2),(2,4)]
        centroid=[(2,3),(7,2)]
        cluster_dict={0:[],1:[]}
        dict1={0: [(1, 2), (1, 3), (3, 4), (3, 3), (2
                        , 2), (2, 4)], 1: [(6, 1),
                         (6, 3), (7, 2), (8, 3)]}
        self.assertEqual(alloc_dp_to_cluster_no(pop_pm,centroid,
                                cluster_dict,2),dict1)
```

```
    def test_calc_new_centroid(self):
       centroid=[(2,3),(7,2)]
       cluster_dict={0: [(1, 2), (1, 3),
                         (3, 4), (3, 3), (2, 2), (2, 4)], 1:
                             [(6, 1),(6, 3), (7, 2), (8, 3)]}
       s1=calc_avg([(1, 2), (1, 3), (3, 4), (3, 3), (2, 2), (2, 4)])
       s2=calc_avg([(6, 1), (6, 3), (7, 2), (8, 3)])
       expected_centroid=[s1,s2]
       self.assertEqual(calc_new_centroid(centroid,
                                  cluster_dict,2),expected_centroid)

    def test_cluster_1_iteration(self):
       prev=[(2,2),(8,2)]
                                 10
       centroid=[(2,3),(7,2)]
       pop_pm=[(1,2),(1,3),(3,4),(3,3),
                     (6,1),(6,3),(7,2),(8,3),(2,2),(2,4)]
       cluster_dict={0:[(0,0),(0,1)],1:[(1,1),(2,2)]}
       dict1={0: [(1, 2), (1, 3), (3, 4), (3, 3),
                 (2, 2), (2, 4)], 1: [(6, 1), (6, 3), (7, 2), (8, 3)]}
       self.assertDictEqual(cluster(prev,centroid,pop_pm,
                                       cluster_dict,2)[0],dict1)
       self.assertListEqual(cluster(prev,centroid,pop_pm,
                                cluster_dict,2)[1],centroid)

    def test_cluster_2_iteration(self):
        prev=[(2,2),(8,2)]
        centroid=[(0,1),(0,2)]
        pop_pm=[(1,2),(1,3),(3,4),(3,3),(6,1),(6,3),(7,2),
                      (8,3),(2,2),(2,4)]
        cluster_dict={0:[(0,0),(0,1)],1:[(1,1),(2,2)]}
        dict1={1: [(1, 2), (1, 3), (3, 4), (3, 3), (2, 2),
                       (2, 4)], 0: [(6, 1), (6, 3), (7, 2), (8, 3)]}
        self.assertDictEqual(cluster(prev,centroid,pop_pm,
                                       cluster_dict,2)[0],dict1)
        self.assertListEqual(cluster(prev,centroid,pop_pm,
                                  cluster_dict,2)[1],centroid)

    def test_result(self):
        cluster_dict={0:[(15000000,0),(5000000,1)],
                                1:[(6000000,1),(8000000,2)]}
        population_dict={“chennai”:15000000,”erode”:5000000,
                         “vellore”:6000000,”coimbatore”:8000000}
        list_of_cities_0=[“19oimbat”,”erode”]
        list_of_cities_1=[“vellore”,”19oimbatore”]
        clustered_city={0: [“chennai”, “erode”],
                                1: [‘vellore’, ‘coimbatore’]}
        self.assertDictEqual(result(cluster_dict,population_dict),
                                                        clustered_city)

if __name__ == ‘__main__’:
    unittest.main()
```

Sample input JSON file for population and CSV file for POLLUTION:

city_population.json

```json
{
  "Hyderabad": "6,731,790",
  "Ahmedabad": "5,577,940",
  "Chennai": "4,646,732",
  "Kolkata": "4,496,694",
  "Surat": "4,467,797",
  "Pune": "3,124,458",
  "Jaipur": "3,046,163",
  "Lucknow": "2,817,105",
  "Kanpur": "2,765,348",
  "Nagpur": "2,405,665",
  "Indore": "1,964,086",
  "Thane": "1,841,488",
  "Bhopal": "1,798,218",
  "Visakhapatnam[a][5]": "1,728,128",
  "Pimpri-Chinchwad": "1,727,692",
  "Patna": "1,684,222"
}
```

air_quality_PM10_2012.csv

```
State,City,Location,Station Code,Type,Category of ES,No. of mon. days (n),Min,Max,PM10
Annual average (µg/m3),10 percentile,90 percentile,Standard Deviation,Percentage-
exceedence(24 hourly),Air Quality
Andhra Pradesh,Chitoor,GNC Toll Gate Tirumala,582,RIRuO,NA,103,24,52,40,35,45,4.0,Moderate
Andhra Pradesh,Guntur,"Near Hindu College, Market Road",583,RIRuO,NA,
105,71,79,75,2,73,77,0,High
Andhra Pradesh,Hydrabad,"Tarnaka, NEERI Lab. IICT Campus",150,RIRuO,NA,
88,8,70,26,17,35,10,0,Low
Andhra Pradesh,Hydrabad,"Nacharam, Industrial Estate",151,RIRuO,NA,
87,12,100,30,19,42,15,0,Low
Andhra Pradesh,Hydrabad,ABIDS Circle General Post Office,152,RIRuO,NA,
88,12,58,27,17,43,10,0,Low
Andhra Pradesh,Hydrabad,Balanagar,95,RIRuO,NA,108,42,286,128,72,189,46,69,Critical
Andhra Pradesh,Hydrabad,"Uppal, Modern Foods & Industries IDA",203,RIRuO,NA,
108,30,241,120,58,197,53,59,Critical
Andhra Pradesh,Hydrabad,Jublee Hills,365,RIRuO,NA,109,19,240,85,40,124,38,30,High
Andhra Pradesh,Hydrabad,Paradise,393,RIRuO,NA,107,27,240,98,65,139,33,36,Critical
Andhra Pradesh,Hydrabad,Charminar,394,RIRuO,NA,106,35,233,106,64,150,37,54,Critical
Andhra Pradesh,Hydrabad,Zoo Park,470,RIRuO,NA,108,13,175,75,31,122,35,25,High
Andhra Pradesh,Hydrabad,"Jeedimetla Industrial Estate, Rangareddy Distt.",745,RIRuO,NA,
95,33,215,98,57,143,36,43,Critical
Andhra Pradesh,Kakinada,Office Building Ramanayyapeta,578,RIRuO,NA,Null,
45,70,58,53,62,4,0,Moderate
Andhra Pradesh,Kothagudem,"CER Club, Khamam",581,RIRuO,NA,90,53,81,63,57,70,6,0,High
Andhra Pradesh,Kurnool,Mourya Inn,466,RIRuO,NA,102,44,100,74,56,88,12,0,High
Andhra Pradesh,Nalgonda,AP PCB Nalgonda,577,RIRuO,NA,108,47,145,79,63,97,16,6,High
Andhra Pradesh,Nellore,"Venkatareddy nagar, Vedayapalem",580,RIRuO,NA,
102,45,78,62,54,72,7,0,High
Andhra Pradesh,Patencheru,"Police Station, Medak, Ramachadrapuram",468,RIRuO,NA,
108,74,153,108,89,127,16,67,Critical
```

Functions Explanation:

File : input_data_processing.py
Function : dataprep():

- Gets the input population from the json file (created from screenscraped .txt from wikipedia and converted to json) using json module.
- Also loads the pollution statistics from csv file from data.gov.in.
- Checks if the cities in the csv file are present in the json file and creates and returns a dictionary with keys as population and values as particulate matter pollution

File: clustering_fucntions.py

Function: edist(A,B)
Returns the value of the euclidian distance between 2 points A and B (2 Dimension).

Function: calc_avg(dp_list)
Given the list of points assigned to a centroid it calclulates the average of all the data points assigned to a centroid in the previous iteration and returns the new centroid.

Function: alloc_dp_cluster_no(pop_pm,centroid,cluster_dict,k)
Finds and assigns the closest centroid for each datapoint and returns a dictionary with centroid number (key) and data points (values)

Function: calc_new_centroid(centroid,cluster_dict,k)
Given the list of points assigned to a centroid it calculates the average of all the data points assigned to a centroid in the previous iteration and returns the new centroid.

Function: cluster(prev_centroid,centroid,pop_pm,cluster_dict,k)
Uses the rest of the utility functions to cluster the data points and returns the (population,pollution_level) clusters.

File: clustering_result.py
Fucntion: result(cluster_dict,population_dict)
Uses the the cluster_dictionary and extracts the cities for the respective populations in the cluster and returns clusters of cities.

File: cluster_plot.ipynb (interactive python notebook - Jupiter)
This file calls all the functions and gives us the final output. With marplot lib we get the visual output also.

CORE WORKING USING THESE FUNCTIONS :

Note that in the final code, **dataprep()** is called to get the input data of population to pollution (dict). This is fed as input to the main function which initialises random centroids and calls the cluster function.

The **cluster function** with the help of - **prev_centroid** (dict that has previous iterations centroids), **centroid** (current centroid dict of reallocated centroids with centroid number as key and datapoints as values), the **input population/pollution parcel** and **k** (# clusters) runs a loop and reallocates centroids (using euclidian distance function edist() and alloc_dp_cluster_no() and calc_new_centroid()) until it finds the final centroids and clusters.

All these functions including the main are called in python notebook (Jupyter) so that visual output can be produced.

Also these Core functions have all been unit tested under clustering_unittest.py

PSEUDO CODE OF IMPLEMENTED CODE (input_process, functions, output_process):

Pseudocode of input_data_processing.py:

```
Begin
Define function prepdata
Open file for city population from json
      Read CSV file for air quallity
          Set CSV as delimiter
          Create a dictionary for population
           for row in readCSV:
          if row[1] in data:
              if row[9] != “Null”:
                  pm10_dict[row[1]] = row[9]
end if (1)
    end if (2)
                  population_dict[row[1]] = data[row[1]]
        Create list pop_pm
       for city in population_dict:
          pop_pm.append((int(population_dict[city].replace(“,”,””))  int(pm10_dict[city])))
end for
       Return the list and dictionary
End
```

Pseudocode for clustering functions:

```
import libraries: math copy random Constant

function edist(A,B):
            #Calculated the eucledian distance using dist formula between points A and B.
            dist=sqrt((Ax - Bx)2 - (Ay - 2)
            return (dist{rounded to 2 decimal values})

function calculate average(data point list):
            if (list is empty):
                        #create random tuple
                        return (random.randint(in range min to max population),random.randint(in range min to max pollution))
            for i in datapoint list:
                        sum1= datapoint list [i][0]
                        sum2= datapoint list [i][1]
            end for
            x comp avg = sum1/ len(datapoint list)
            y comp avg = sum2/ len(datapoint list)
            return(round (x comp avg) ,round (y comp avg))

function difference(previous points -current points):
            for i in range 0 - len(prev)):
                        x=previous[i]x - current[i]x
                        y=previous[i]y - current[i]y

function allocate datapoint to cluster(list of points, centroid list, cluster dictionary , k):
            for i in range (0,len(list of points):
                        #create  a temporary list
                        temp=[]
                        for j in range(0,k);
                                    temp.append( euclidean distance( list of point[i] , centroid[j]))
                        end for
                        centroid number = index of (min(temp)) in temp
                        cluster_dict[centroid number].append(pop_pm[i])
            end for
            return cluster_dict

function calculate new centroid(centroid, cluster_dict, k):
            for i in range (0-k):
```

```
                centroid[i]= calculate average (cluster_dict,k):
        return centroid

function cluster(previous centroid, centroid,  list of points, cluster_dict, k):
        initialize while loop count = 0
        while(previous centroid!=centroid):
                #clean original centroid dictionary
                for i in range(k):
                        cluster dict[i]=[]

                #Euclidean Distance between the Centroid and DataPoints
                cluster_dict=allocate datapoint to cluster(list of points,centroid,cluster dictionary,k)

                # current centroid is stored in Previous Centroid for the next iteration
                 previous centroid=copy.deepcopy(centroid)

                # increment the while loop count
                while loop count = while loop count +1

                # check if the previous centroid and new centroid have any difference
                diff(previous centroid,centroid)

        return cluster dictionary, centroid
```

Pseudocode of the clustering_result.py:

```
Begin
Define-> result(cluster_dict,population_dict):
   Assign sign population as list(population_dict.values())
   for i in range(len(population)):
      population[i]=int(population[i].replace(“,”,””))
   end for
    city = list(population_dict.keys())
   for j in range(len(cluster_dict.keys())):
      curr_cluster_list_tup=cluster_dict[j]
    end for
     Create a temporary list, temp_list=[]
      for i in range(len(curr_cluster_list_tup)):
         curr_cluster_pop=curr_cluster_list_tup[i][0]
         temp_list.append(city[population.index(curr_cluster_pop)])
      end for
   return clustered_city
End
```

Time and Space Complexity

For a data-set with n objects, each with m attributes (parameters/ dimensions), the k-means clustering algorithm has the following characteristics:

Time-Complexity:

For every iteration there are:

* Calculation of distances: To calculate the distance from a point to the centroid, we can use the squared Euclidean proximity function. For this, we need two subtractions, one summation, two multiplications and one square - root operations, i.e., 6-operations.

* Comparisons between distances

* Calculation of centroids (cluster centre-points)

Therefore, for every iteration, the number of operations =

6*[k*m*n] operations + [(k-1)*m*n] operations + [k*((m-1) + 1)*n] operations

If the algorithm converges within I iterations then the operations =

6*[I*k*m*n] operations + [I*(k-1)*m*n] operations + [I*k*((m-1) + 1)*n] operations

Therefore, the time complexity is O(I*k*m*n).

where, I is # iterations, k is # centroids/clusters, m is number of dimensions. For large data-sets where k << m & n << m, the complexity is approximately O(m)

However, in a more general case, the liner time complexity of a k means clustering algorithm is $O(n^2)$.

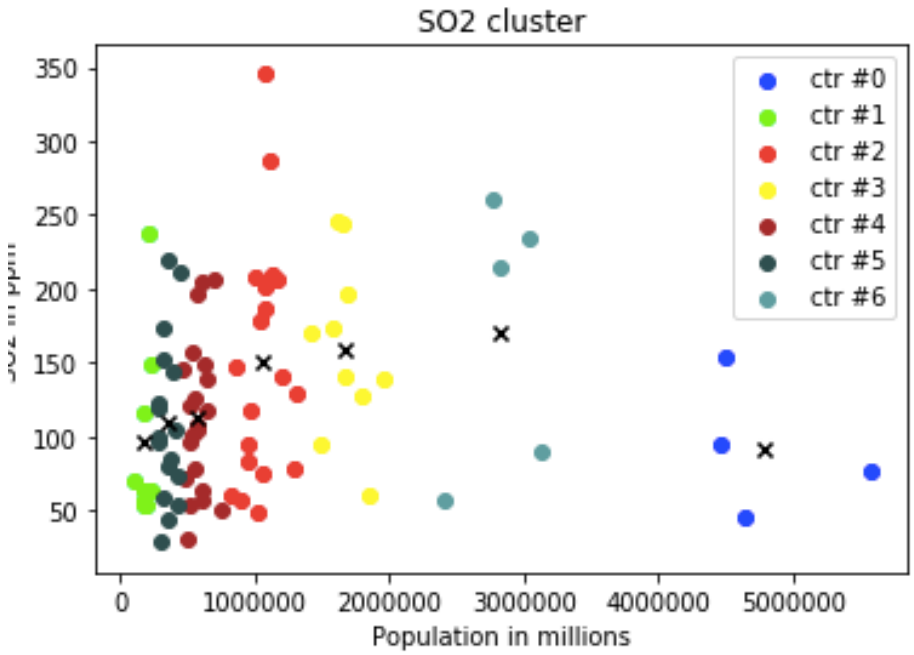
The k-means algorithm is known to have a time complexity of O (n2), where n is the input data size.

Space-Complexity:

You only need to store the data points and centroids. So the complexity is **O((m+k)n)** for the same meanings of m,k,n.

Results and Output

Visual Final Result (Scatterplot from Jupyter notebook):

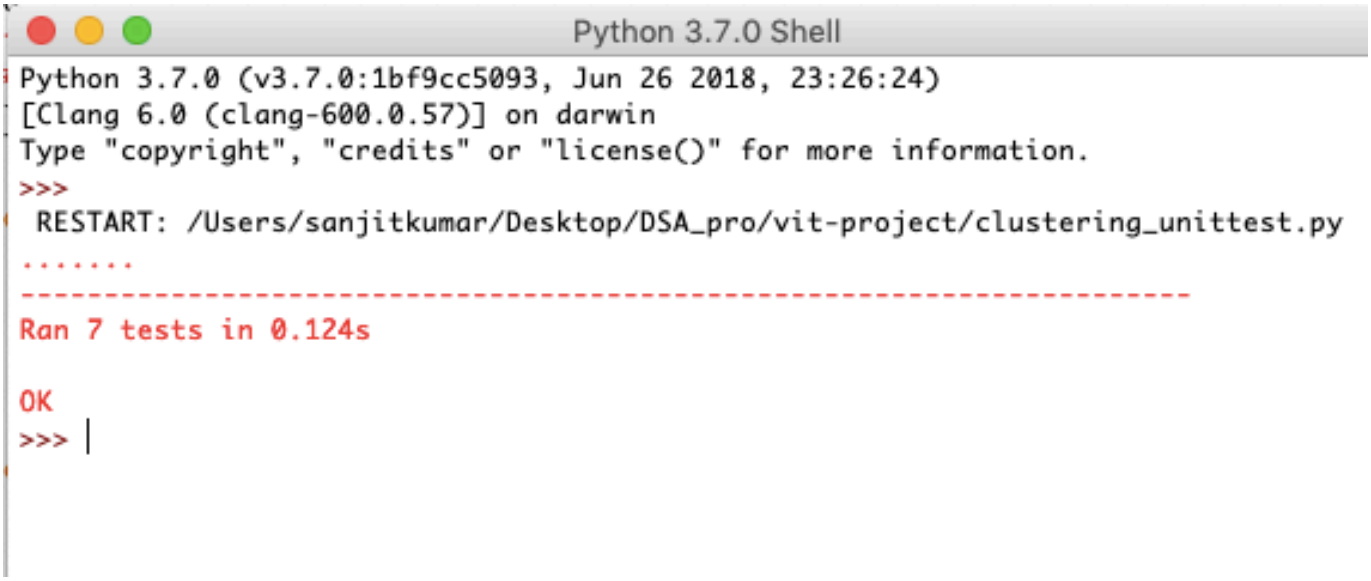


Final clusters of cities (in the form of a dictionary):

```
In [10]: print(cluster_dict_result)
```

```
{0: ['Ahmedabad', 'Surat', 'Chennai', 'Kolkata'], 1: ['Shimla', 'Alappuzha', 'Kottayam', 'Singrauli', 'Aurangabad',
'Sambalpur', 'Unnao', 'Haridwar', 'Haldia'], 2: ['Guwahati', 'Chandigarh', 'Raipur', 'Rajkot', 'Dhanbad', 'Ranchi',
'Mysore', 'Gwalior', 'Jabalpur', 'Solapur', 'Amritsar', 'Jalandhar', 'Jodhpur', 'Madurai', 'Salem', 'Allahabad', 'Mee
rut', 'Varanasi', 'Howrah'], 3: ['Patna', 'Vadodara', 'Faridabad', 'Bhopal', 'Indore', 'Nashik', 'Thane', 'Ludhiana',
'Agra', 'Ghaziabad'], 4: ['Jamnagar', 'Jamshedpur', 'Belgaum', 'Mangalore', 'Kochi', 'Thiruvananthapuram', 'Ujjain',
'Amravati', 'Jalgaon', 'Kolhapur', 'Ulhasnagar', 'Akola', 'Nanded', 'Cuttack', 'Firozabad', 'Jhansi', 'Noida', 'Sahar
anpur', 'Dehradun', 'Asansol', 'Durgapur'], 5: ['Kakinada', 'Kurnool', 'Korba', 'Kollam', 'Kozhikode', 'Dewas', 'Saga
r', 'Chandrapur', 'Latur', 'Jalna', 'Aizawl', 'Berhampur', 'Rourkela', 'Patiala', 'Alwar', 'Udaipur', 'Mathura'], 6:
['Nagpur', 'Pune', 'Jaipur', 'Kanpur', 'Lucknow']}
```

Unit test Output:



CONCLUSION

As we wanted, we have obtained a graph and the clusters for the purpose of analysing the pollution vs population data we have inputted. This idea was inspired by the the unhealthy air quality standards we live in today. As we can see we conclude from the graph that cluster 0 has the minimal population is to average pollution ratio.

Though this program might not solve the problem of air pollution, we really hope the idea of analysing and mining pollution data would be taken seriously and will be a step in the right direction.

The algorithm works properly with the given data (legit/factual data) and effectively clusters the data-points that are in 2 dimension. The functions run in python 3.7.0 fluently and the Jupyter notebook outputs the scatterplot for the clusters.

Cluster analysis and data mining are vast and complex fields but we have tried to implement the K-means on a simple realtime problem.

Future Prospects :

The code can be made more efficient in terms of time and space complexity. Possible real life applications are apps that show daily real time pollution statistics. In fact with proper accurate data, the algorithm can be used for knowing the amount of pollution risk in a particular region of a city at any time.

Reference

www.https://data.gov.in/catalog/ambient-air-quality-respect-particulate-matter-under-national-air-quality-monitoring/

https://en.wikipedia.org/wiki/List_of_cities_in_India_by_population

www.matplotlib.org/

www.pythonspot.com/

www.stackoverflow.com/

www.codecademy.com

www.geeksforgeeks.org

https://realpython.com/python-json/