

PROJECT REPORT

WIA2005 ALGORITHM DESIGN & ANALYSIS

SEMESTER 2, 2021/2022

Tutorial Group : Tutorial 1

Group Name : Group 6

Lecturer : Dr Asmiza Binti Abdul Sani

Group Members :

| **No.** | **Name** | **Matric ID** |
| --- | --- | --- |
| 1. | AMIRAH BINTI ABD NASIR | 17203688/2 |
| 2. | ATHIRAH BINTI AHMAD FAUZI | U2005247/1 |
| 3. | FARISAH ATHIRA BT MD ZAMRI | 17206915/2 |
| 4. | NURSYAZWANI BINTI MOHAMMAD | 17206746/2 |
| 5. | NUHA BINTI MOHAMED YAZID | 17206682/2 |
| 6. | NURUL AIN BT KHAIRUL ANWAR | U2005370/1 |

**TABLE OF CONTENTS**

[**INTRODUCTION**](#_foljukmrnd0k) **3**

[**DESCRIPTION**](#_alvxjjrrh5o9) **4**

[Problem 1](#_v97lgldomfm) 4

[Problem 2](#_udr8pre4129i) 7

[Problem 3](#_qsn8st77mbhp) 12

[**TIME COMPLEXITY**](#_gn3vafxi5f7o) **14**

[Problem 1](#_mgsoefk16om1) 14

[Problem 2](#_lb6qtd7c7wyy) 14

[Problem 3](#_y9b6l01c6hqg) 14

[**PROGRAM CODE & OUTPUT SAMPLE**](#_s3nee2hpd2g3) **15**

[Problem 1](#_avlx488zm0iq) 15

[Problem 2](#_rr9x216xll37) 21

[Problem 3](#_vrty2ez2f11a) 31

[**CONCLUSION**](#_hv0gkby3oy8t) **33**

# INTRODUCTION

An algorithm is a sequence of instructions that execute calculations, data processing, and automated reasoning in order to solve a problem. An effective technique that can be stated in a finite amount of time and space is called an algorithm. The design of an effective algorithm that solves a problem effectively while taking up the least amount of time and space is one of the key elements of algorithm design. Different methods can be used to tackle a problem. In terms of time usage, some of them may be effective, whilst other methods might be memory effective.

The purpose of this project is to do analysis, design an algorithm to solve business problems and provide insights to the management for making business decisions. In order to find a solution, we use the Trie algorithm, K-means algorithm, Nearest Neighbour algorithm and Bubble Sort algorithm. Other than that, different kinds of tools are used such as Plotly to show the graph analysis and Geopy and Folium to map the coordinate of the route.

# 

# DESCRIPTION

## Problem 1

[Brainstorming activity P1](https://docs.google.com/document/d/1Vk5bmOtxQH3ZJ4mSqnQX4FqWqCK2Zu4uQL0VnsPeh14/edit?usp=sharing)

Problem 1 highlights the sentiment analysis technique using a string-matching algorithm. Moonbucks is looking for the possibility of expanding its business by adding a number of stores around the world. Therefore, they have to analyse local economic and social conditions by using sentiment analysis for the selected countries which are Canada, Indonesia, Malaysia, Singapore and the United States to ensure a maximum profit.

**Algorithm**

**Trie Algorithm**

The algorithm used to analyse the word count for stop words, negative, positive and neutral words is Trie Algorithm. We chose Tries because of the time complexity of its worst case, which is O(n) where we thought it is the best choice other than other string matching algorithms like KMP. Here is the step of how we implement our algorithm:

1. First, before the analysis can be done, we have searched 5 articles related to each country’s economic and social status and extracted the words from the webpage into a text version. Then, we save all the words into the text file for each country.
2. Then, we also collect and list all the English negative, positive and stop words in the text file for the string matching algorithm. It is to search for the negative, positive and stop words in the articles.
3. We store the positive and negative words inside trie and find the matching words inside the article.
4. After we sorted out the stop words from the articles and stored them inside the filtered file, we calculated the total of positive and negative words first. After that, in order for us to find the neutral words, we use this calculation:

**Neutral words = total words inside the filtered file - (positive words + negative words)**

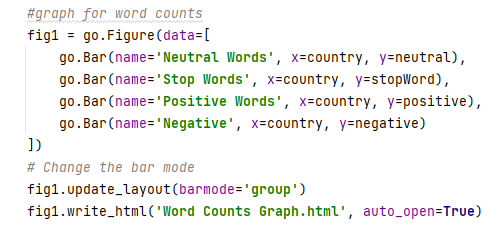
1. For the algorithm, we have created 3 classes which are the class *Trie* for the Trie algorithm, class *Freq* to calculate the frequency of the type of words and the main class.

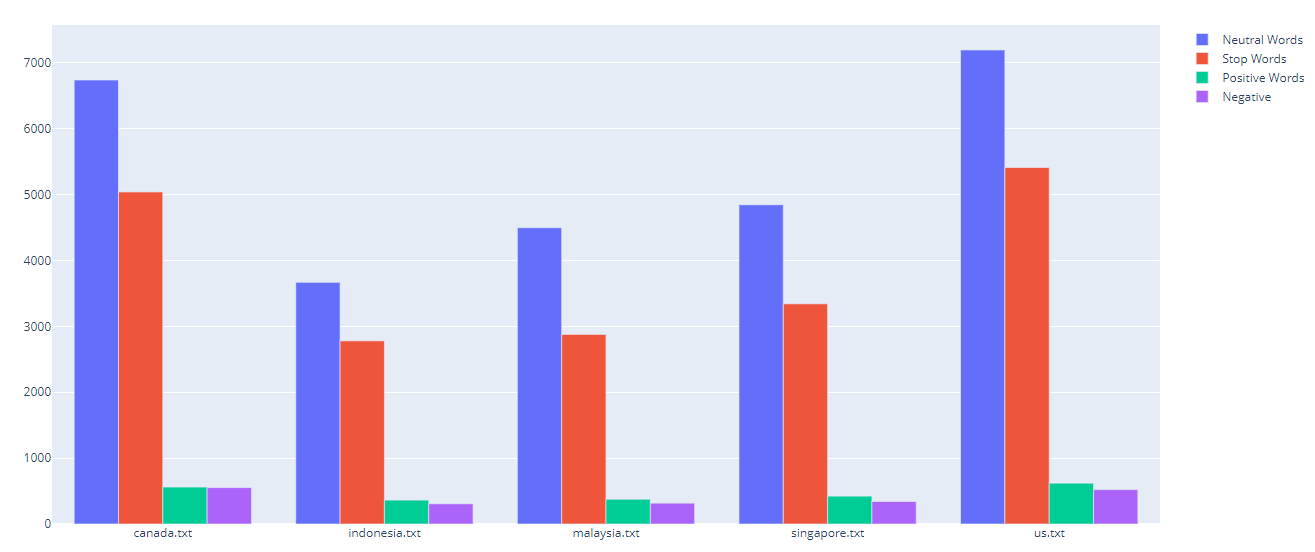
**Tool**

**Plotly**

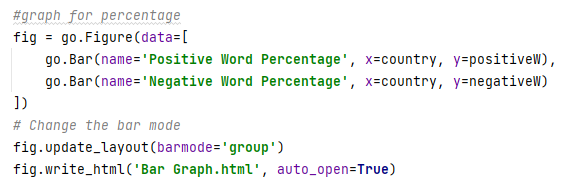
The tool used to show the graph of the analysis is Plotly because it is easy to use and can create different types of graphs using the library. There are 2 group bar charts created to visualise the word counts analysis to show useful information about the countries’ local economic and social conditions.

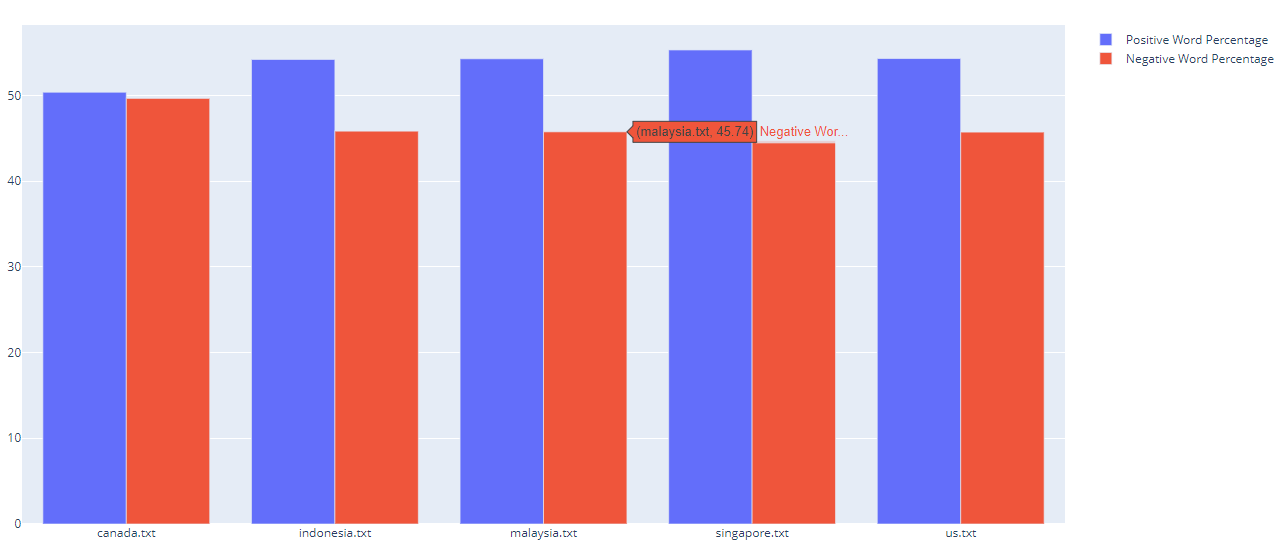
1. As we have to analyse different types of words which are neutral, stop, positive, and negative, the first graph is plotted to show how many words are for each type of word for each country.
2. First, the Plotly Python library is imported to access the library’s script for creating the graph.
3. Then, to plot the graph in Plotly, we simply call the Bar() function from the Plotly Graph Objects instance and provide x and y arguments with valid data.
4. For the first graph, the x argument is the list of countries generated from the text file while the y argument provides the word count for each type of word. Then, we visualised the chart as a grouped bar plot which is grouped by the countries.

***Figure 1*** *code for the first graph*

**

***Figure 2*** *Grouped bar plot for word count*

1. For the second graph, the x argument is the same as the first graph, but for the y arguments, it represents the percentage of positive and negative words for each country. This is to show which country has the highest positive percentage and from that, we can decide which is the best country to have the store expansion. 

***Figure 3*** *Code for second graph*

***Figure 4*** *Grouped bar plot for positive and negative percentage*

## Problem 2

[Brainstorming activity P2](https://docs.google.com/document/d/15hc-eDvdG47L3R1Co5dS-GFiQ05LFCRVFyo_Rz_Q82I/edit?usp=sharing)

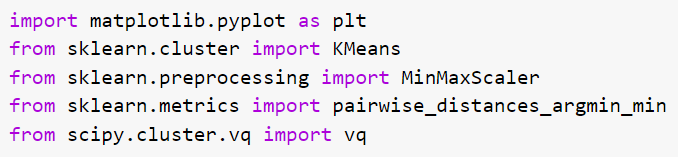
Problem 2 emphasises the shortest route optimization technique. Initially, Moonbucks companies ship their stock only from warehouses in the region. This situation makes the delivery of goods inefficient. Therefore, Moonbucks company wants to have a local central distribution centre in each country to deliver their stock to all stores in the country.

The solution that has been used is by applying a K-Means and Nearest Neighbour algorithm.

**Algorithm**

**K-Means Algorithm**

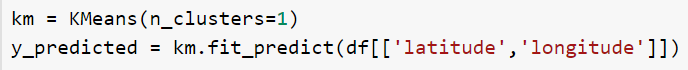
In order to find the best distribution centre among stores, we used K-Means algorithm. We imported some important libraries to imply K-Means.



1. The first step is we read the latitude and longitude of a specific country from the dataset.



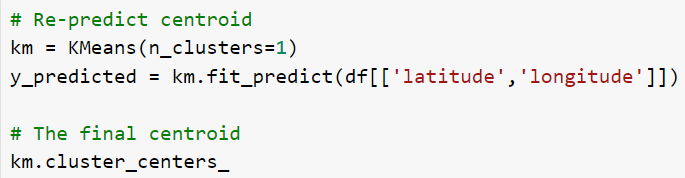
1. Secondly, we set the number of clusters, n\_clusters = 1. This is because we treated a country as a cluster. Then, we define y\_predicted as a predicted centroid (given the list of latitude & longitude of each of the stores.)



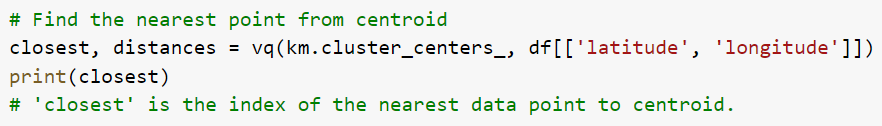
1. Next, we scale the dataframe of columns ‘latitude’ and ‘longitude’ by using MinMaxScaler(). The default range for the feature returned by MinMaxScaler is 0 to 1.



1. Then, re-predict the cluster & centroid. The final centroid is km.cluster\_centers\_.



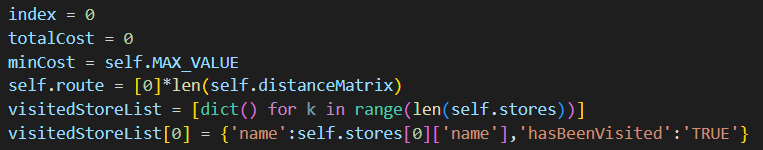
1. In the last step, by gaining the final centroid, we try to find out the nearest point from the centroid. The closest returns the index for the store closest to the centroid.



**Nearest Neighbour Algorithm**

Problem 2 emphasizes the need to find the most optimal delivery cost which the route will start from and end at the distribution centre. We have incorporated Nearest Neighbour Algorithm to find the best sequence in choosing the next store that should be visited. The algorithm prioritizes the shortest path from the node to its neighbour. The shortest route among the neighbouring nodes will be selected and the process continues until all nodes have been visited. Finally, the last node chosen will select the path to the starting node.

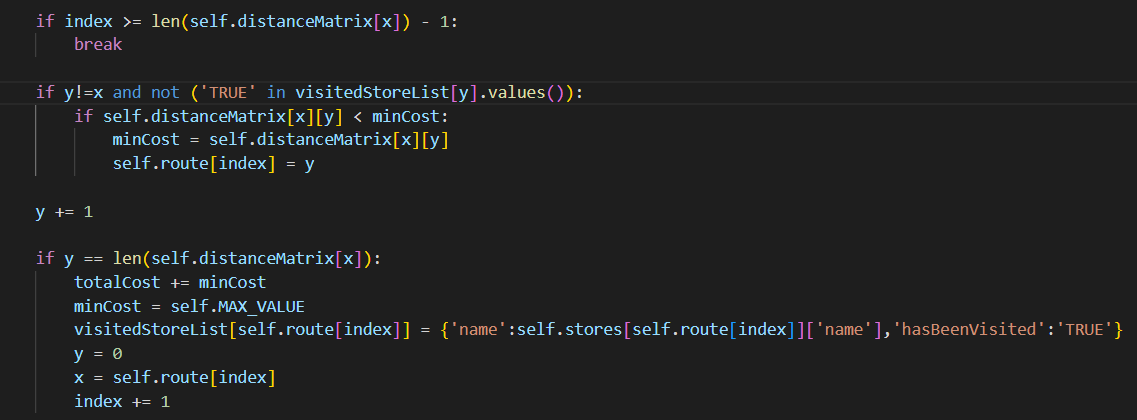
1. First, we initialize int variables as 0 and an array of dictionary visitedStoreList to store the name of the stores (key: ‘name’) and their status whether or not the stores have been visited (key: ‘hasBeenVisited’). The first store, which is also the local distribution centre is initialized and stored in the array visitedStoreList.



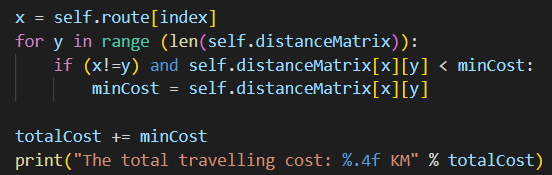
1. In a while loop, two conditions are set the loop should not exceed the size of array distanceMatrix and the size of element in array distanceMatrix.



1. If the current store has not been visited yet, it will find and compare the cost from the minCost. If the cost is lower than the current minCost, the cost value will be set as the new minCost value. At the same time, the array route will store the value of the current store.
2. The process continues until all unvisited stores have been compared.
3. Then, the next store will repeat the same process p.3-p.4 until all stores have been visited. The while loop ends.



1. The last store will choose the route to the starting store and the totalCost will also be added.



**Tool**

**Geopy and Folium**

The tool used to map the coordinate of route is Geopy and also the Folium library. This is because both of the libraries are easy to use. By using Folium, it can visualise an interactive real map. The steps to visualise a path in a map are as follows :-

1. First, create a base map with the centre coordinate of the country.Can choose any of built-in tileset that suit with the use of your program .We chose OpenStreetMap.

m = folium.Map(location=[self.latBase,self.lngBase],tiles="OpenStreetMap",zoom\_start= 5)

1. After creating a base map,we need to plot the coordinates on the map.We use a Folium Marker to plot the point.In addition,we also use a popup parameter to label our marker.By using for loop,we iterate the list of stores coordinate and add it to the map.



**for** i **in** range(len(self.stores)):

lat = self.stores[i]['latitude']

lng = self.stores[i]['longitude']

**if** i == 0:

popup = self.stores[0]['name'] + '<br>' + str(self.stores[0]['street\_address']) + '<br>' + str(self.stores[0]['zip\_code']) + '<br>' + str(self.stores[0]['city'])

folium.Marker(location=[self.stores[0]['latitude'],self.stores[0]['longitude']],popup=popup,icon=folium.Icon(color='green')).add\_to(m)

**else**:

popup = self.stores[i]['name'] + '<br>' + str(self.stores[i]['street\_address']) + '<br>' + str(self.stores[i]['zip\_code']) + '<br>' + str(self.stores[i]['city'])

folium.Marker(location=[lat,lng],popup=popup,icon=folium.Icon(color='blue')).add\_to(m)

3.To connect the point,we need to use the AntPath plugin from Folium.This will create a line from the starting point until the end point. Lastly, save the map in html.

plugins.AntPath(waypoints).add\_to(m)

m.save('D:**\V**S Code Project**\A**LGO\_PROJECT'+self.code+'\_map.html')

## Problem 3

Problem 3 highlights the need of expanding a business in a country by considering the local economic and social situation of the country and also the running cost for delivering logistics.

**Probability**

Weightage: Sentiment score = 75%, Delivery cost = 25%

* Due to huge differences in geographical factor, we decided that it is best that the weightage leans more towards sentiment score to reduce bias selection.

P(S) = (0.75 \* (percentage of positive word/total percentage of positive word)) \* (0.25 \* (1 - (cost/total cost)))

| No. | Country | Sentiment Score (%) | Delivery Cost (KM) | Probability |
| --- | --- | --- | --- | --- |
| 1. | Indonesia | 54.18 | 19413.0755 | 0.036258 |
| 2. | Canada | 50.36 | 3639.1348 | 0.027281 |
| 3. | Malaysia | 54.26 | 3238.0747 | 0.036488 |
| 4. | US | 54.29 | 60033.3097 | 0.011583 |
| 5. | Singapore | 55.29 | 103.6103 | 0.038581 |

**Algorithm**

**Bubble Sort Algorithm**

To rank the country according to its final result, we use the Bubble Sort algorithm because it is efficient to sort the countries based on their probability values.

1. Starting from the first index, compare the first and the second elements.
2. If the first element is greater than the second element, they are swapped.
3. Now, compare the second and the third elements. Swap them if they are not in order.
4. The above process goes on until the last element.
5. The same process goes on for the remaining iterations.
6. After each iteration, the largest element among the unsorted elements is placed at the end.
7. In each iteration, the comparison takes place up to the last unsorted element.
8. The array is sorted when all the unsorted elements are placed at their correct positions.

# 

# TIME COMPLEXITY

## Problem 1

## 

| **Algorithm** | | **Time Complexity** | | |
| --- | --- | --- | --- | --- |
| **Best Case** | **Average Case** | **Worst Case** |
| Trie Algorithm | | O(n) | O(n) | O(n) |

## Problem 2

| **Algorithm** | **Time Complexity** |
| --- | --- |
| **Best Case** |
| KMeans Algorithm | O(n2) |
| Nearest Neighbour Algorithm | O(n2) |

## 

## Problem 3

| **Algorithm** | **Time Complexity** | | |
| --- | --- | --- | --- |
| **Best Case** | **Average Case** | **Worst Case** |
| Bubble Sort Algorithm | O(n) | O(n2) | O(n2) |

## 

## 

## 

# PROGRAM CODE & OUTPUT SAMPLE

## Problem 1

**Source Code**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import plotly.express as px

import plotly.graph\_objects as go

from plotly.subplots import make\_subplots

class Node:

def \_\_init\_\_(self):

self.children = {}

self.last\_letter = False

class Trie:

def \_\_init\_\_(self):

self.root = Node()

def insert(self, word):

cur = self.root

for ch in word:

if ch not in cur.children:

cur.children[ch] = Node()

cur = cur.children[ch]

cur.last\_letter = True

def search(self, word):

cur = self.root

for ch in word:

if ch not in cur.children:

return False

cur = cur.children[ch]

if cur.last\_letter:

return True

else:

return False

def startwithPrefix(self, prefix):

cur = self.root

for ch in prefix:

if ch not in cur.children:

return False

cur = cur.children[ch]

return True

class Freq:

def frequency(country,filter):

neg\_w=open("NEGATIVE WORDS.txt")

T = Trie()

for x in neg\_w:

T.insert(x)

tr=Trie()

pos\_w= open("POSITIVE WORDS.txt")

for z in pos\_w:

tr.insert(z)

st\_count=0

filt\_freq=[]

stop\_words= open("STOP WORDS.txt","r")

stopw= stop\_words.read()

st=stopw.splitlines()

with open(country, encoding="utf-8") as f:

for lines in f:

words = lines.split()

for r in words:

if not r in st:

appendFile = open(filter,'a')

appendFile.write(r+"\n")

filt\_freq.append(r)

else:

st\_count+=1

appendFile.close()

p\_wordfreq=[]

with open(filter) as file1:

for line in file1:

word1 = line.split()

for i in word1:

if tr.startwithPrefix(i):

p\_wordfreq.append(i)

# print(wordfreq)

else:

break

n\_wordfreq=[]

with open(filter) as file2:

for line in file2:

word2 = line.split()

for j in word2:

if T.startwithPrefix(j):

n\_wordfreq.append(j)

# print(wordfreq)

else:

break

positive = len(p\_wordfreq)

negative = len(n\_wordfreq)

neutral = (len(filt\_freq) - (negative + positive))

Ppercent = positive / (positive + negative) \* 100

Npercent = negative / (positive + negative) \* 100

NePercent = neutral / (positive + negative + neutral) \* 100

print("The total stop word in the file is: " + str(st\_count)) # print all stop word in txt files

print("Total positive word frequency: ", positive, "%.2f" % Ppercent)

print("Total negative word frequency: ", negative, "%.2f" % Npercent)

print("Total neutral word frequency: ", neutral)

outputFile = open('output.txt', 'a')

outputFile.write(country + " " + str(neutral) + " " + str(st\_count) + " " + str(positive) + " " + str(negative) + " " + str("%.2f" % Ppercent) + " " + str("%.2f" % Npercent) + " " + str("%.2f" % NePercent) + "\n")

fr=Freq

filter=["filteredtext.txt","filteredtext1.txt","filteredtext2.txt","filteredtext3.txt","filteredtext4.txt"]

line=["canada.txt","indonesia.txt","malaysia.txt","singapore.txt","us.txt"]

for x in filter:

f = open(x, 'r+')

f.truncate(0)

g = open('output.txt', 'r+')

g.truncate(0)

for i in range (5):

print("\nCountry:", line[0])

fr.frequency(line.pop(0),filter.pop(0))

# get data from text file

with open('output.txt', 'r') as data:

country = []

neutral= []

stopWord = []

positive = []

negative = []

positiveW = []

negativeW = []

neutralW = []

for line in data:

column = line.split(' ')

country.append(column[0])

neutral.append(float(column[1]))

stopWord.append(float(column[2]))

positive.append(float(column[3]))

negative.append(float(column[4]))

positiveW.append(float(column[5]))

negativeW.append(float(column[6]))

#plot graph

values = [neutral, stopWord, positive, negative]

fig = px.line(x = country, y = values, title = "Words count")

#fig = go.Figure(data=go.Scatter(y=values))

fig.show()

# first plot with X and Y data

plt.plot(country, neutral, label = "Neutral words")

plt.plot(country, stopWord, label = "Stop words")

plt.plot(country, positive, label = "Positive words")

plt.plot(country, negative, label = "Negative words")

plt.xlabel("Country")

plt.ylabel("Words Count")

#plt.title('multiple plots')

plt.legend(loc="best")

plt.show()

x\_axis = np.arange(len(country))

width1 = 0.2

plt.bar(x\_axis - width1, positiveW, width=0.4, label='Positive Value')

plt.bar(x\_axis + width1, negativeW, width=0.4, label='Negative Value')

plt.xticks(x\_axis, country)

plt.legend(loc='upper center', bbox\_to\_anchor=(0.5, -0.05),

fancybox=True, shadow=True, ncol=5)

plt.show()

print("\n")

# conclusion

print("Summary from the Analysis in Percentage (%)")

a = {'Country': country, 'Positive Words': positiveW, 'Negative Words': negativeW}

df = pd.DataFrame(a)

condition = [df['Positive Words'] > df['Negative Words'], df['Positive Words'] < df['Negative Words']]

choice = ['Positive Sentiment', 'Negative Sentiment']

df['Sentiment'] = np.select(condition, choice)

print(df)

maxPercent = max(positiveW)

print("Based on the result, it can be concluded that article about Singapore"

"\nhas the highest percentage of positive words which is" + " " + str(maxPercent))

"""

inputSize = len(positiveW)

gap = inputSize//2

while gap>0:

for i in range(gap, inputSize):

temp = positiveW[i]

j=i

while j>=gap and positiveW[j-gap]<temp:

positiveW[j]=positiveW[j-gap]

j-=gap

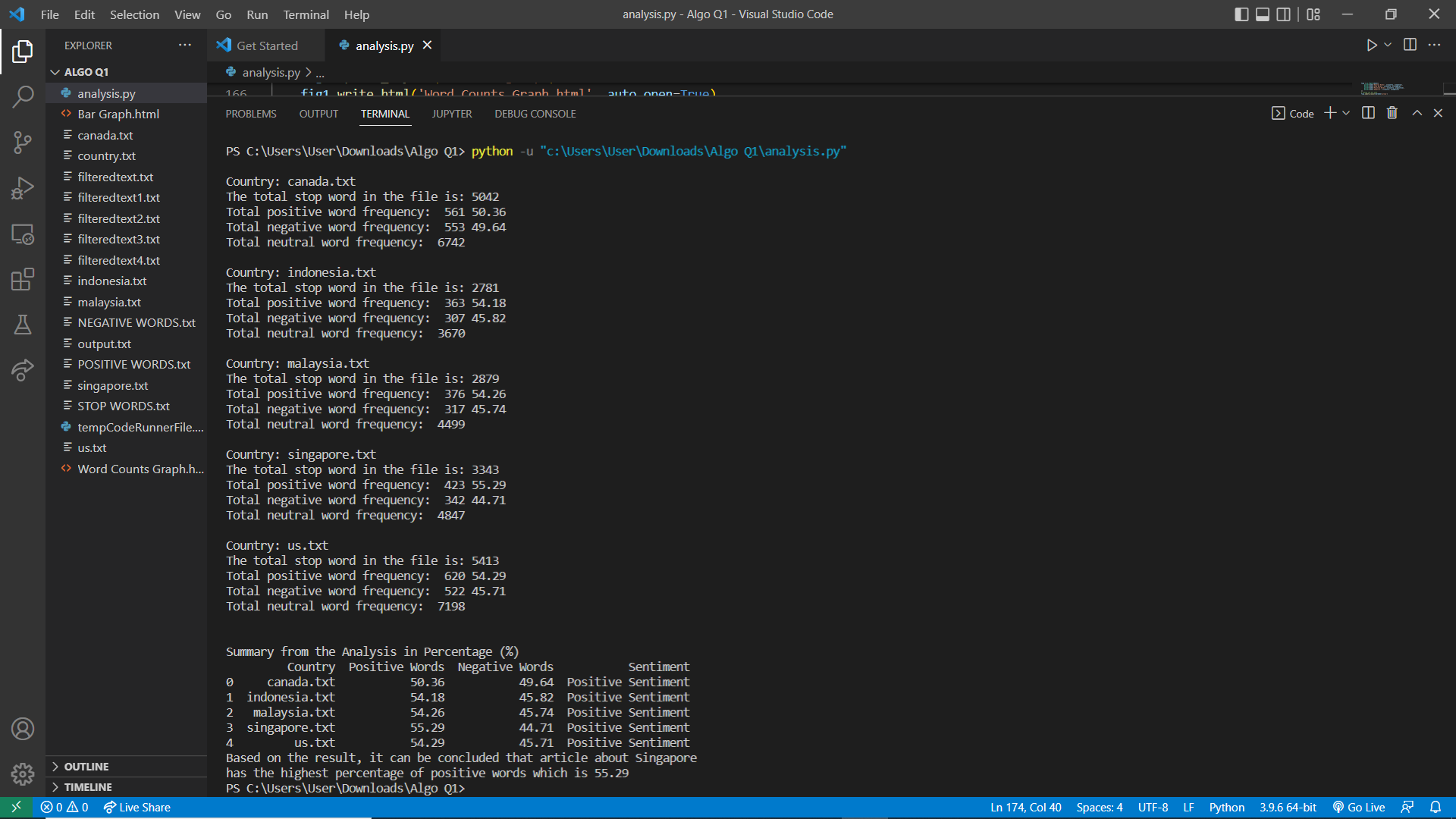
positiveW[j]=temp

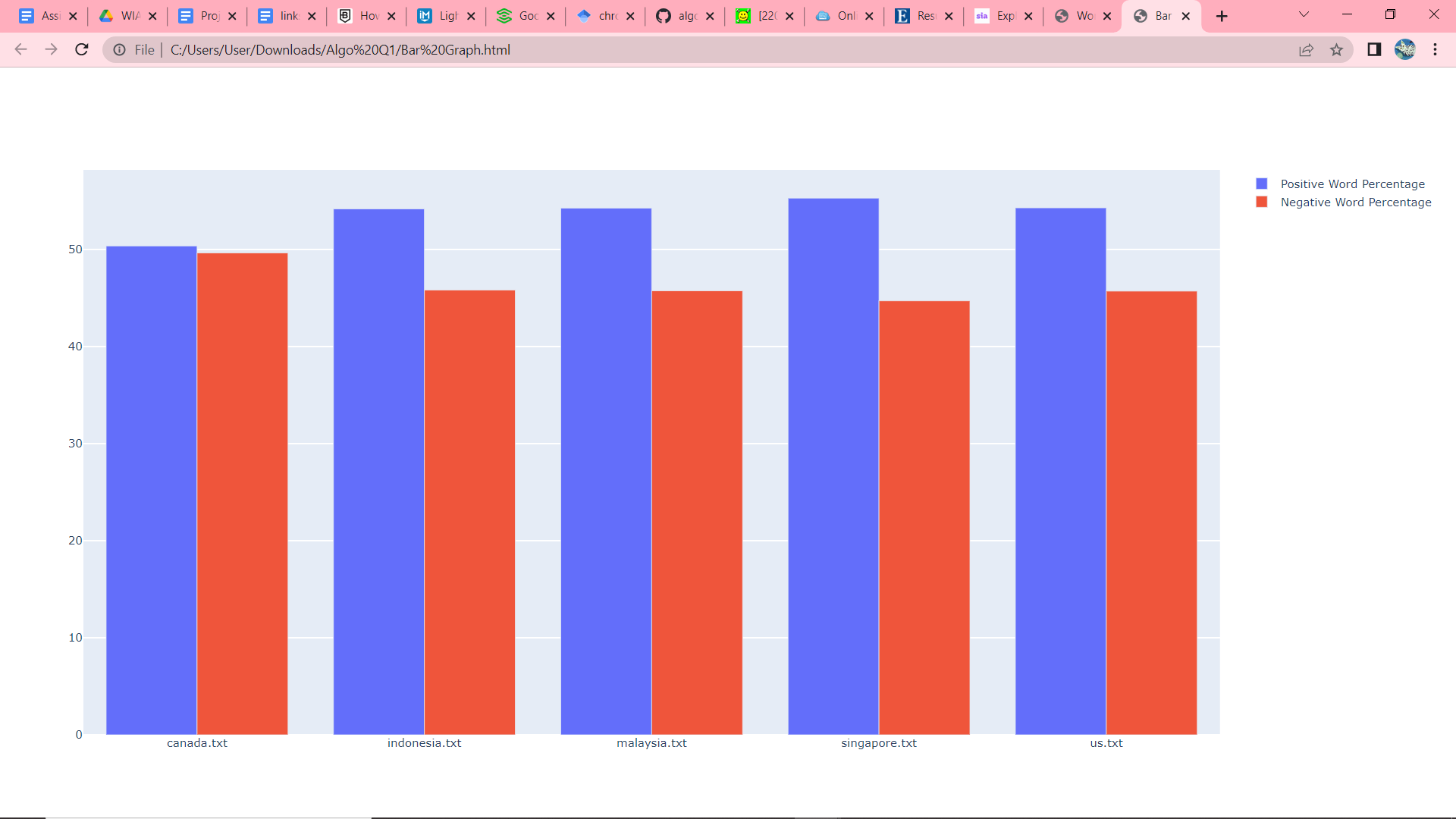
gap = gap//2

print(positiveW)

"""

**Output**





## 

## Problem 2

## 

**Source Code**

import sys

import folium

import folium.plugins as plugins

import geopy.distance

import numpy as np

import pandas as pd

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

from scipy.cluster.vq import vq

class Problem2:

def \_\_init\_\_(self,name,code,latBase,lngBase):

self.name = name

self.code = code

self.latBase = latBase

self.lngBase = lngBase

self.stores = []

self.distanceMatrix = []

self.route = []

self.countryStores = []

self.MAX\_VALUE = sys.float\_info.max

def storeData(self):

df = pd.read\_csv("starbucks.csv")

self.countryStores = df.loc[df['state']==self.code]

self.countryStores = self.countryStores.reset\_index()

self.countryStores.drop('index', inplace=True, axis=1)

self.stores = self.countryStores.head(1000).to\_dict(orient='records')

self.countryStores = self.countryStores.head(6)

def kMeansAlgorithm(self):

pd.set\_option('mode.chained\_assignment', None)

df = self.countryStores[["latitude","longitude"]]

km = KMeans(n\_clusters=1)

y\_predicted = km.fit\_predict(df[['latitude','longitude']])

# Predicting cluster based on number of clusters

df['cluster'] = y\_predicted

# Scaling dataset

scaler = MinMaxScaler()

scaler.fit(df[['latitude']])

df['latitude'] = scaler.transform(df[['latitude']])

scaler.fit(df[['longitude']])

df['longitude'] = scaler.transform(df[['longitude']])

# Re-predict centroid

km = KMeans(n\_clusters=1)

y\_predicted = km.fit\_predict(df[['latitude','longitude']])

# The final centroid

km.cluster\_centers\_

# Find the nearest point from centroid

# 'closest' is the index of the nearest data point to centroid

closest, distances = vq(km.cluster\_centers\_, df[['latitude', 'longitude']])

# Swap center position

self.stores[0],self.stores[closest[0]] = self.stores[closest[0]],self.stores[0]

def setDistanceMatrix(self):

self.distanceMatrix = [[0 for i in range(len(self.stores))] for j in range(len(self.stores))]

for i in range (len(self.stores)):

for j in range (len(self.stores)):

coordinate1 = ((self.stores[i]['latitude']),(self.stores[i]['longitude']))

coordinate2 = ((self.stores[j]['latitude']),(self.stores[j]['longitude']))

self.distanceMatrix[i][j] = geopy.distance.distance(coordinate1,coordinate2).km

self.distanceMatrix[j][i] = geopy.distance.distance(coordinate1,coordinate2).km

def nearestNeighbourAlgorithm(self):

index = 0

totalCost = 0

minCost = self.MAX\_VALUE

self.route = [0]\*len(self.distanceMatrix)

visitedStoreList = [dict() for k in range(len(self.stores))]

visitedStoreList[0] = {'name':self.stores[0]['name'],'hasBeenVisited':'TRUE'}

x,y = (0,0)

while x < len(self.distanceMatrix) and y < len(self.distanceMatrix[x]):

if index >= len(self.distanceMatrix[x]) - 1:

break

if y!=x and not ('TRUE' in visitedStoreList[y].values()):

if self.distanceMatrix[x][y] < minCost:

minCost = self.distanceMatrix[x][y]

self.route[index] = y

y += 1

if y == len(self.distanceMatrix[x]):

totalCost += minCost

minCost = self.MAX\_VALUE

visitedStoreList[self.route[index]] = {'name':self.stores[self.route[index]]['name'],'hasBeenVisited':'TRUE'}

y = 0

x = self.route[index]

index += 1

x = self.route[index]

for y in range (len(self.distanceMatrix)):

if (x!=y) and self.distanceMatrix[x][y] < minCost:

minCost = self.distanceMatrix[x][y]

totalCost += minCost

print("The total travelling cost: %.4f KM" % totalCost)

def display(self):

# Display route sequence

print('\nRoute:',end=' ')

print(self.stores[0]['name'],'->',end=' ')

for i in range(len(self.stores)-1):

print(self.stores[self.route[i]]['name'],'->',end=' ')

print(self.stores[0]['name'],'\n')

latList = [0 for i in range(len(self.stores)+1)]

latList[0] = self.stores[0]['latitude']

lngList = [0 for i in range(len(self.stores)+1)]

lngList[0] = self.stores[0]['longitude']

for p in range(len(self.stores)-1):

latList.append(self.stores[self.route[p]]['latitude'])

lngList.append(self.stores[self.route[p]]['longitude'])

latList.append(self.stores[0]['latitude'])

lngList.append(self.stores[0]['longitude'])

latList = [float(i) for i in latList if i!=0]

lngList = [float(i) for i in lngList if i!=0]

waypoints = np.array(list(zip(latList,lngList)))

# Display map

m = folium.Map(location=[self.latBase,self.lngBase],tiles="OpenStreetMap",zoom\_start= 5)

for i in range(len(self.stores)):

lat = self.stores[i]['latitude']

lng = self.stores[i]['longitude']

if i == 0:

popup = self.stores[0]['name'] + '<br>' + str(self.stores[0]['street\_address']) + '<br>' + str(self.stores[0]['zip\_code']) + '<br>' + str(self.stores[0]['city'])

folium.Marker(location=[self.stores[0]['latitude'],self.stores[0]['longitude']],popup=popup,icon=folium.Icon(color='green')).add\_to(m)

else:

popup = self.stores[i]['name'] + '<br>' + str(self.stores[i]['street\_address']) + '<br>' + str(self.stores[i]['zip\_code']) + '<br>' + str(self.stores[i]['city'])

folium.Marker(location=[lat,lng],popup=popup,icon=folium.Icon(color='blue')).add\_to(m)

plugins.AntPath(waypoints).add\_to(m)

m.save('C:\\Users\\Acer\\Documents\\GitHub\\Moonbucks-Store-Project\\Problem 2\\'+self.code+'\_map.html')

# Create object from class Problem2

Indonesia = Problem2('Indonesia','ID',0.7893,113.9213)

print('\nCountry:',Indonesia.name,'\n')

Indonesia.storeData()

Indonesia.kMeansAlgorithm()

Indonesia.setDistanceMatrix()

Indonesia.nearestNeighbourAlgorithm()

Indonesia.display()

Canada = Problem2('Canada','CA',56.0,-96.0)

print('\nCountry:',Canada.name,'\n')

Canada.storeData()

Canada.kMeansAlgorithm()

Canada.setDistanceMatrix()

Canada.nearestNeighbourAlgorithm()

Canada.display()

Malaysia = Problem2('Malaysia','MY',4.140634,109.6181485)

print('\nCountry:',Malaysia.name,'\n')

Malaysia.storeData()

Malaysia.kMeansAlgorithm()

Malaysia.setDistanceMatrix()

Malaysia.nearestNeighbourAlgorithm()

Malaysia.display()

America = Problem2('USA','US',37.6,-95.665)

print('\nCountry:',America.name,'\n')

America.storeData()

America.kMeansAlgorithm()

America.setDistanceMatrix()

America.nearestNeighbourAlgorithm()

America.display()

Singapore = Problem2('Singapore','SG',1.3146631,103.8454093)

print('\nCountry:',Singapore.name,'\n')

Singapore.storeData()

Singapore.kMeansAlgorithm()

Singapore.setDistanceMatrix()

Singapore.nearestNeighbourAlgorithm()

Singapore.display()

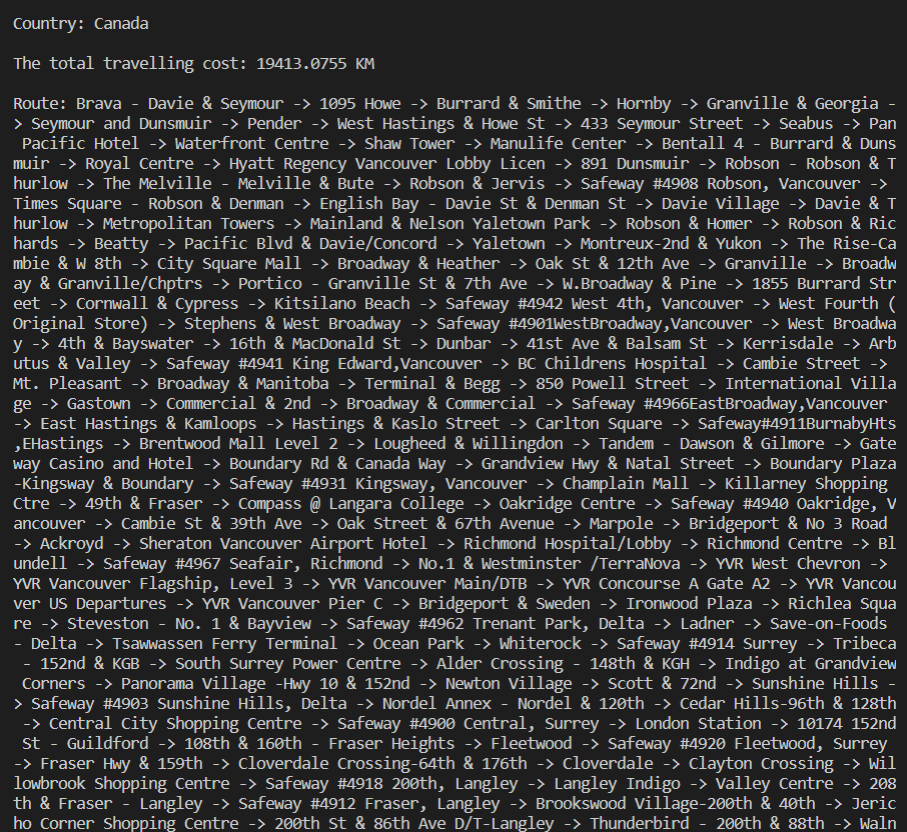
**Output**

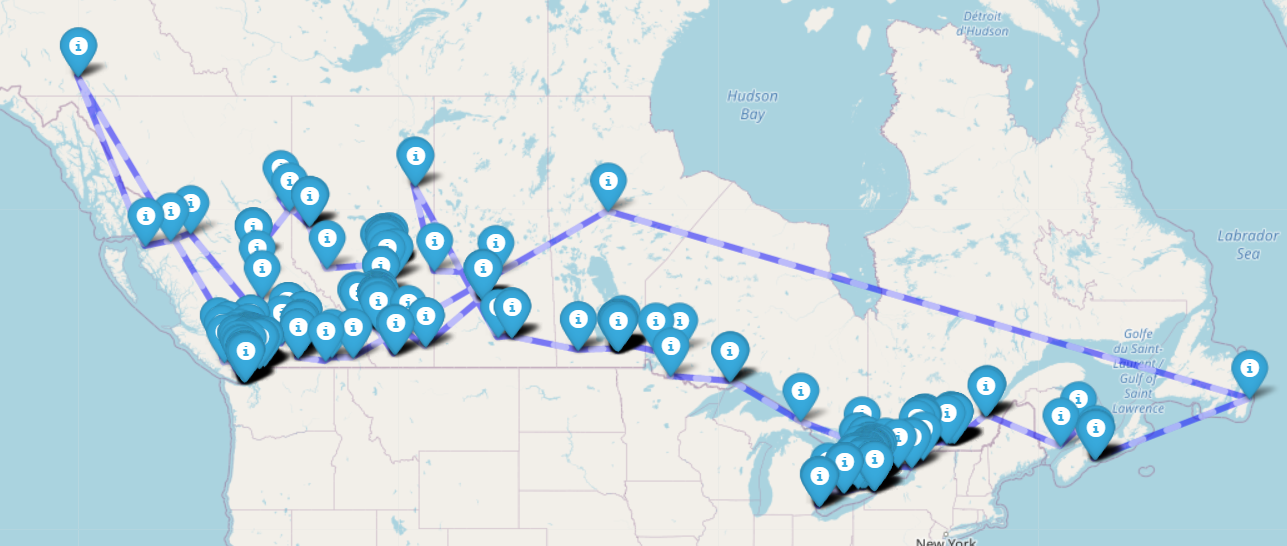
**Indonesia:**



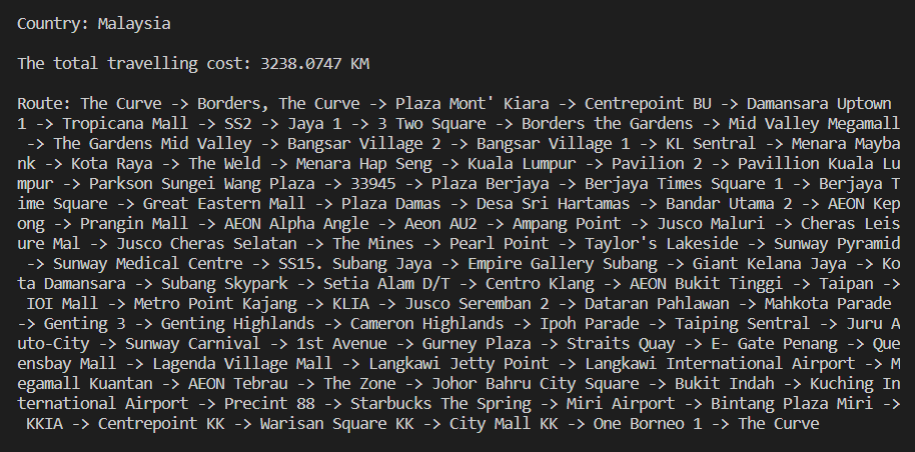


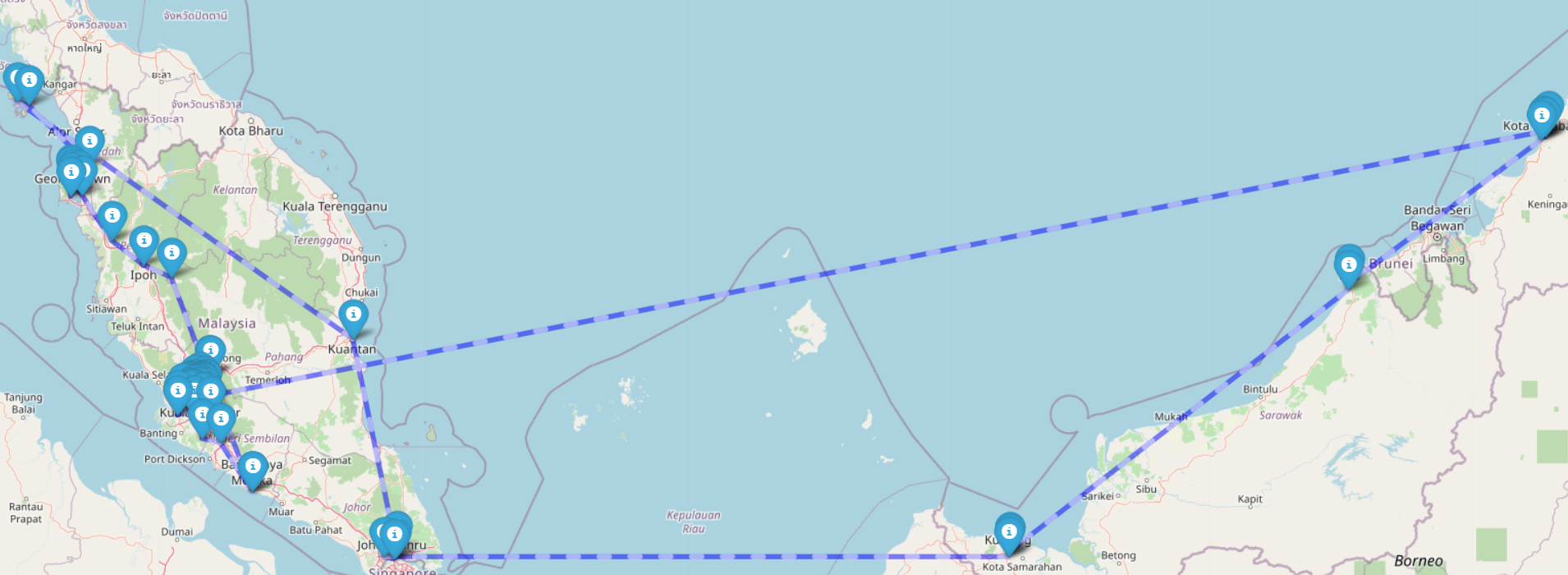
**Canada:**

****

****

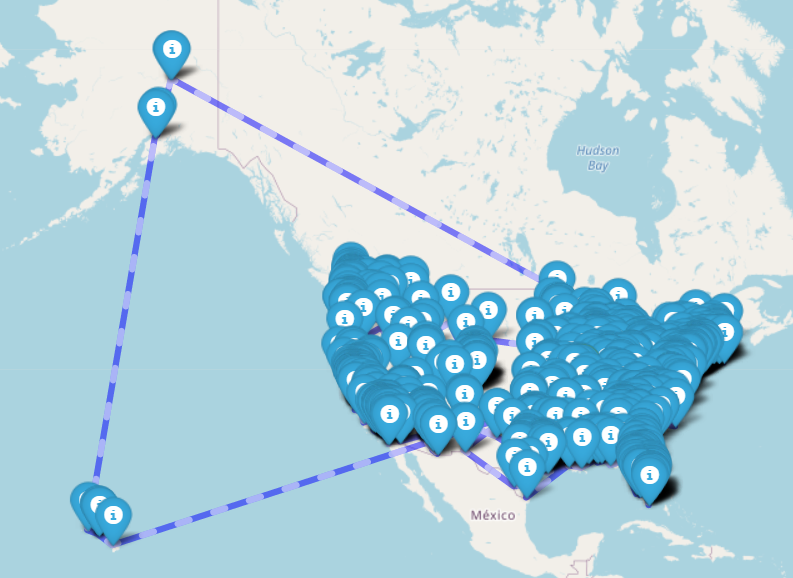
**Malaysia:**

****

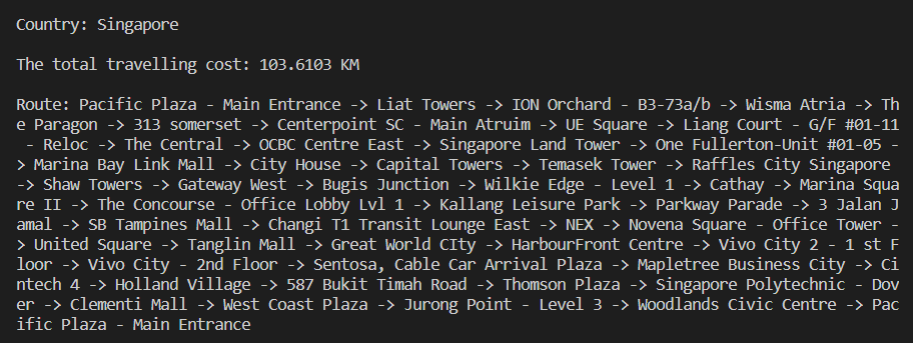
****

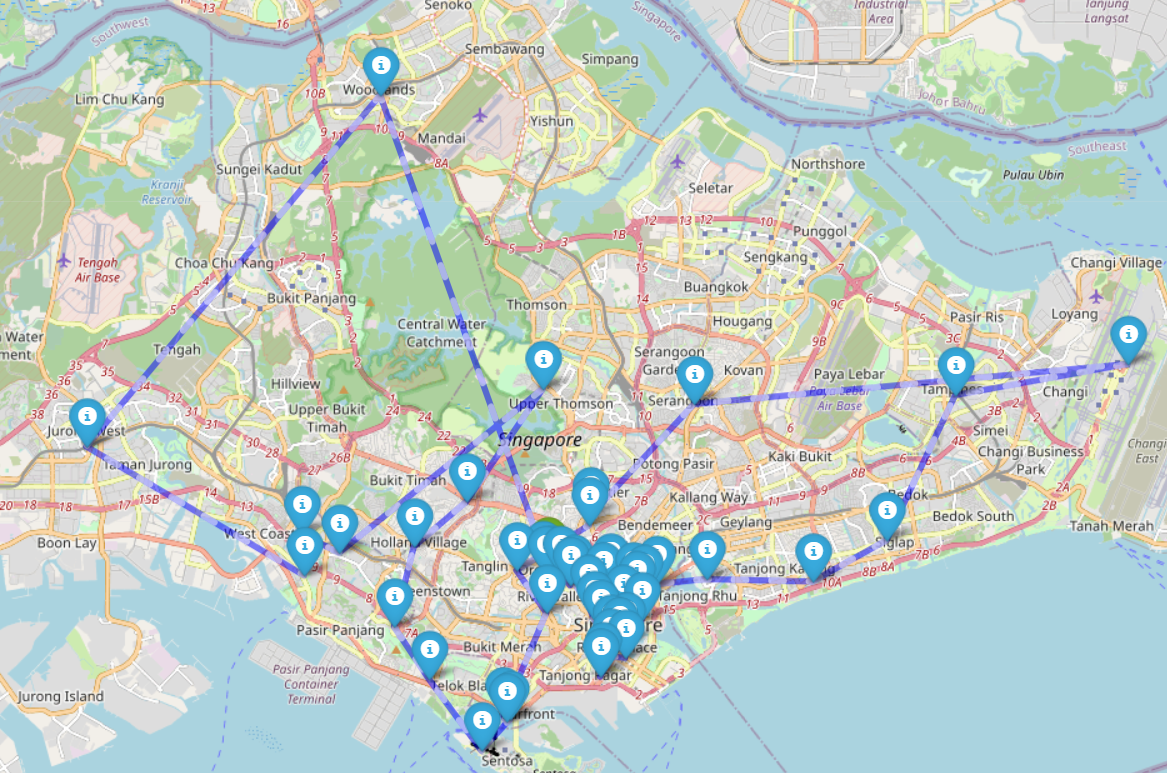
**US:**

****

****

**Singapore:**

****

****

## Problem 3

**Source Code**

# get data from text file

import pandas as pd

def BubbleSort(Probability, n):

swapped = False

# Traverse through all array elements

for i in range(n - 1):

for j in range(0, n - i - 1):

if Probability[j] < Probability[j + 1]:

swapped = True

Probability[j], Probability[j + 1] = Probability[j + 1], Probability[j]

if not swapped:

return

sentiment = []

CountryProb = []

country=[]

output1 = []

positiveList = []

costList = []

probability1 = []

with open('output2.txt', 'r') as data:

for line in data:

cells = line.split(",")

country.append(cells[0])

positiveList.append(float(cells[1]))

costList.append(float(cells[2]))

totalPos = sum(positiveList)

totalCost = sum(costList)

for i in range(len(country)):

probability = ((3/4 \* (positiveList[i] / totalPos)) \* (1/4\*(1-(costList[i] / totalCost))))

probability1.append(probability)

sentiment.append((country[i], positiveList[i], costList[i], probability))

df = pd.DataFrame(sentiment, columns=["Country", "Positive Percentage", "Cost", "Probability"])

print(df)

arr = probability1.copy()

BubbleSort(arr, len(arr))

for x in range(0, len(arr)):

arr[x]

SortedCountry = []

for x in range(0, len(probability1)):

for y in range(0, len(probability1)):

if arr[x] == probability1[y]:

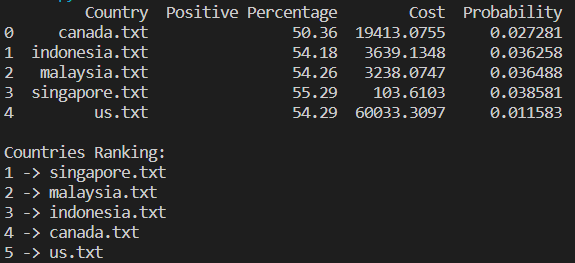
SortedCountry.append(y)

print("\nCountries Ranking:")

for x in range(0, len(SortedCountry)):

print(x + 1, '->', country[SortedCountry[x]])

**Output**



# CONCLUSION

By considering an algorithm for a specific problem, we can begin to design and develop code through python to solve the problem. Algorithms are often quite different from one another, even though the objective of these algorithms are the same. But, the time complexity of those algorithms may differ. Those chosen algorithms help to solve the problem and we can conclude that for Problem 3, the order from the most recommended country to undergo business expansion, to the least recommended one starts from Singapore, Malaysia, Indonesia, Canada and lastly US.

**References**

*Analysis of Algorithms | Set 4 (Analysis of Loops)*. (n.d.). GeeksforGeeks. Retrieved June 23, 2022, from https://www.geeksforgeeks.org/analysis-of-algorithms-set-4-analysis-of-loops/

Badvelu, J. (2020, September 25). *Determining Optimal Distribution Centers locations using Weighted K-Means*. Towards Data Science. Retrieved June 23, 2022, from https://towardsdatascience.com/determining-optimal-distribution-centers-locations-using-weighted-k-means-1dd099726307

Machhar, S. (2019, September 2). *Introduction to DAA(Design And Analysis of Algorithms)*. Medium. Retrieved June 23, 2022, from https://medium.com/@withsnowy2009/introduction-to-daa-design-and-analysis-of-algorithms-60559c7e4ba9

Menghanibhanvi. (2020, August 26). *Folium: All you need for Map Visualization | by Menghanibhanvi | Image Vision*. Medium. Retrieved June 23, 2022, from https://medium.com/image-vision/folium-all-you-need-for-map-visualization-206e88c8ac

*Shell Sort (With Code in Python, C++, Java and C)*. (n.d.). Programiz. Retrieved June 23, 2022, from https://www.programiz.com/dsa/shell-sort

ShivamKumar. (n.d.). *K Means Cluster | K-Means*. GitHub Pages. Retrieved June 23, 2022, from https://shivamkumar-bit.github.io/K-Means/