UNIVERSITI MALAYA

WIA2005 ALGORITHM DESIGN & ANALYSIS

Project Report

LECTURER NAME : DR. ASMIZA BINTI ABDUL SANI
COURSE OCCURRENCE : 1
GROUP NAME : 7

STUDENT NAME	MATRIC NO.
PANG CHONG WEN	U2005402/1
CHEONG YI FONG	U2005327/1
ZHAODONG LI	S2003637/1
CHIA SIAO WEI	U2005267/1
LEONG JING WEI	U2005251/1
NEIL LAI GUAN MING	U2005423/1

Table of Contents

1. Introduction

Our team has assisted the Moonbucks Coffee Chain by applying IT in major business decisions during our tenure. The feasibility of the plan is analyzed, and clear recommendations are provided through the results of the analysis. First of all, we solve the problem of expanding the company, by analyzing the market conditions in many countries and their regions, and recommending the most suitable development sites. Secondly, according to the requirements of the company, we choose the company branch with the best location as the center point, which greatly reduces the transportation cost for the company. Finally, we once again combined many market reports and the results of transportation cost analysis to summarize the best suggestions for company expansion. Following, we will introduce the solutions in detail based on the three main issues raised by the company.

2. Description

Problem 1

Algorithm used: Web Scraping, Natural Language Processing, Trie String Matching
Python Library Applied: BeautifulSoup, nltk, pandas, pickle
API used: None

Description

In this part, we were tasked to analyze local economic and social situations from articles of 5 countries to ensure maximum profit. We are also required to plot related graphs to show the analysis result. The hypothesis applied is if there are more positive words, then the article is giving positive sentiment, if there are more negative words, then it is giving negative sentiment. Based on the sentiment, we have to rank which country is worth having branch expansion.

1. Each article is scraped using the BeautifilSoup library and all the titles, subtitles and each paragraph of the article is extracted and saved into a document file. All positive, negative and stop words are also scraped from websites and saved into document files. This step is to prevent constant scraping of the website but only to access the file in the directory and to reduce the time taken accessing the files.

2. Each word from the article is lemmatized using the nltk library. They are all tagged afterwards according to its part of speech (POS). This step reduces the words into its basic form and it is easier for the following string matching algorithm to identify the positive and negative words.

3. The original word count and distinct word count is calculated. The frequency of each word is also calculated and stored inside a dictionary.

4. Next, stopwords are removed from each article using the word list obtained above and the built-in stopwords from the nltk library.

5. All positive and negative words are inserted into the Tries object and the object is saved into a binary file. We counted all positive and negative words present in an article by a string matching algorithm using the Tries object obtained above.

6. The positive words percentage (PWS) is calculated as positive words count / (positive word count + negative word count). The overall sentiment of an article is graded according to the positive word percentage such that:
   $PWS < 20 \rightarrow$ Very Negative

20 <= PWS < 40 → Negative

40 <= PWS < 60 → Neutral

60 <= PWS < 80 → Positive

PWS >= 80 → Very positive

7. The overall sentiment of a country is calculated as the average PWS of the 5 articles of each country. The overall sentiment of a country is also graded as the grading system above. The result will display the overall score of each country and its grade and is presented in a descending order.

Trie algorithm:

Trie is an ordered tree-like data structure which consists of nodes and edges like a tree. Each trie node except the root node will store an alphabet character and the end of the word node is marked with a special character. When inserting a key into Trie, every character is inserted as an individual node. If the input is an extension to the existing key, we need to construct a new node and append a special character as the end of the node. Each node or branch can have multiple branches. Trie can quickly search for a word as the words may share the common ancestors. When searching for a key in Trie, we compare each character and move down. The search will terminate if the key is not present or when it comes to the end of the node.

Pseudocode for Trie algorithm :

```
Begin
Initialize empty dictionary as root node of Trie

(Inserting)
For each character in input word
        If character does not exist in head node of current node
                Create new head node
Move pointer to head node
Append a special character to the current head node when there is no more input
character to be inserted

(Searching)
Set starting pointer at root node
For each character in input search pattern
        If character is not found in head node of current node
                Pattern does not exist
        Move pointer to head node
        If special character is found in the current head node
                Pattern exist
```

End

Problem 2

Algorithm used: Bellman-Held-Karp Algorithm
Python Library Applied: numpy, pandas, requests, json, itertools, gmplot
API used: Google Distance Matrix API

Description:
In this problem, we were tasked to decide the local central distribution center in each available country and **plot an optimal delivery route** for each delivery truck across all stores possible in the country. As requested, all deliveries have to start from and end at the distribution center. For the final outcome, we should calculate the **total distance** of delivery route for each of the countries.

1. The exact location information, which comprises name, url, street address, city, state, latitude and longitude, is imported using **pandas** libraries.

2. Specifically, we select one country and locate 6 stores by random within the country itself to compute the **distance matrix** in between the stores by using the Google Distance Matrix API and better visualize the dataset retrieved.

3. The concept of locating the local central distribution center will be the store location which has the **shortest total distance from all store points**.

4. The following move will be looking for the optimal delivery route across all stores possible in the country. We consider the problem similar to the typically-known **Traveling Salesman Problem (TSP)** and **Bellman-Held-Karp algorithm** is used to solve the problem.

5. The final outcome will display the optimal and shortest delivery route across the stores within the country, and also the total (shortest) distance of the route the delivery truck has to travel.

Bellman-Held-Karp algorithm:
Bellman-Held-Karp algorithm is a bottom up dynamic programming approach in solving traveling salesman problems. It starts with the smallest function calls, calculates the cost and slowly builds itself up to solve more complex function calls. Since the problem ends up at the start node, the simplest function calls will be computing the cost of each node going back to the start node, each with an empty subset. We use a distance matrix to enumerate all the function calls that lead one node to another. The function calls become complex when the subset size grows bigger. Dynamic programming with memoization helps us to eliminate the repeated work in computing the function calls with the same

subproblems. For each function call, we store the distance in a dictionary C with a set bit as key generated from the node index, and we refer to the value everytime we encounter the same subproblem.

In this program, we represent a set of vertices using bits of an integer. For example, the set {0,3,4} can be represented by $2^0 + 2^3 + 2^4 = 11001_2$. Therefore, in the program, we use 1<<k to shift the bit 1 to the correct position.

Pseudocode for Bellman Held-Karp algorithm:

In this program, our start node is fixed at index = 0

```
Function Held-Karp algorithm (distance matrix) :
        Set n = length of distance matrix
        C = { }
        For each node in range(n) starting from 1:
                #Save the distance from node to start node in dictionary
                C[(set bit, node)] = (distance, 0)
        For subset size from 2 to n-1:
                For each subset in set of node in range(1,n) with the subset size:
                        #Find shortest distance to get to this subset
                        For k in subset:
                                C[(set bit, k)] = min {C[(set bit-k, m )] + distance(m,k) |
m= subset -k }

        #Calculate shortest distance
        res = [ ]
        For k in range 1 to n:
                Append (C[(set bit, k )][0] + distance(k,0), k) to res
        Cost, parent = min(res)

        Backtrack to find full path and total distance using cost and parent
```

Problem 3

Algorithm used: Quicksort Algorithm
Python Library Applied: pandas
API used: None

Description:
In this problem, we are required to determine the final ranking of the countries where new stores can be located based on the local economic and social situation of the country as well as the total journey made for deliveries of the country.

1. Two dictionaries, the sentiment dictionary and the distance dictionary, are initialized with the results obtained in Question 1 and Question 2 respectively.

2. Perform min-max normalization on the average PWS in the sentiment dictionary. Since the country with a higher average PWS will have a higher score, the lowest average PWS will be the min value which is transformed into 0 while the highest average PWS will be the max value which is transformed into 1 and others will get transformed into decimals between 0 and 1.

3. Perform min-max normalization on the distance in the distance dictionary. Since the country with a shorter total distance will have a higher score, the longest distance will be the min value which is transformed into 0 while the shortest total distance will be the max value which is transformed into 1 and others will get transformed into decimals between 0 and 1.

4. The weights for both sentiment and total distance are set to 0.5. Then, the weight is applied to each normalized score in the sentiment dictionary and the distance dictionary. The weighted sentiment score and weighted distance score for each country are summed up to obtain its final score.

5. The probability of each country being chosen to have expansion among the 6 countries is computed by dividing its respective score with the total score of all the 6 countries.

6. The rank of the countries are sorted using a quicksort algorithm based on their respective final score.

7. Eventually, the ranking of the most recommended country to have expansion is displayed in the form of a dataframe along with their respective average PWS, total journey for deliveries, score obtained and probability.

Quicksort algorithm:

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. Generally, there are 4 different versions of quickSort that pick pivot in different ways.

1. Pick the first element as pivot.
2. Pick the last element as pivot.
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is **partition()**. Target of partitions is, given an array and an element x of array as pivot, put x at its correct position and put all smaller elements (< x) before x, and put all greater elements (> x) after x. All this should be done in linear time.

Partition Algorithm. There can be many ways to partition. One of the ways is that we start from the leftmost element and keep track of the index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap the current element with arr[i]. Otherwise we ignore the current element.

Pseudocode for Quicksort algorithm:

```
//low = starting indexp; high = ending index

quickSort(arr[], low, high) {
   if (low < high) {

      //pi is partitioning index, arr[pi] is now at the right place.

       pi = partition(arr, low, high);
       quickSort(arr, low, pi – 1); // Before pi
       quickSort(arr, pi + 1, high); // After pi
   }
}

// This function takes last element as pivot, places the pivot element at its correct position
in sorted array, and places all smaller (smaller than pivot) to left of pivot and all greater
elements to right of pivot
partition (arr[], low, high)
{
// pivot (Element to be placed at right position)
pivot = arr[high];
```

```
 i = (low – 1)  // Index of smaller element and indicates the
// right position  of pivot found so far
for (j = low;  j <= high-  1; j++){
 // If current element is smaller than the pivot
if (arr[j] < pivot){
i++;    // increment index of smaller element
 swap arr[i] and arr[j]
    }
 }

   swap arr[i + 1] and arr[high])
return (i + 1)
}
```

3. Time complexity

Trie (String Matching Algorithm)

The time complexity of inserting in Trie is O(n) where n is the length of the input string. When inserting a key into Trie, we need to insert each key in a single node.
The time complexity of searching in Trie is O(m) where m is the length of the pattern. When searching for a pattern, we have to move down to track each node only once in the trie.

Held Karp Algorithm

The time complexity of the Held-Karp algorithm is $O(n^2 \cdot 2^n)$. For the loop where we compute for each subset in the given subset size from 2 to n, it takes $O(2^n)$. This is calculated by, for every iteration, we decide to in and out for the element, so it's 2, so it will be 2x2x2x2x…. $= 2^n$. For each execution of the cost function, we compute not more than n values based on the values obtained from the previous step, this takes O(n).

Quicksort algorithm

The best and average case time complexity of the Quicksort algorithm is **$O(n\log_2 n)$**. Everytime the partition happens, the pivot will be swapped to the middle of the list and it is the median of the list. Because this algorithm belongs to the divide and conquer algorithm. In result, the partition will happen $\log_2 n$ time. And each time, the index j will go through all the elements which are n times. Therefore it will be $n * \log_2 n$.

The worst case time complexity of the Quicksort algorithm is **$O(n^2)$**. What if the list has already been sorted, and the pivot starts from the most left or right. After the partition, the pivot is always swapped to the next nearby index. Which means each of the partitions, index j will go through the current number of element-1 times. Therefore the formula will be $n+(n-1)+(n-2)+(n-3) \ldots +1 = n(n+1)/2$, after simplify will be n^2.

4. Source Code

Question 1

```
# Web Scraping Lib
import requests
from bs4 import BeautifulSoup
from urllib.request import Request, urlopen
import lxml
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

# Store Object lib
import pickle

# NLP Lib
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

# General Lib
import re
from itertools import chain
import pandas as pd
import plotly.express as px
```

```
# Required Downloads
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
# String Matching Algorithm : Tries
class Trie:
  def __init__(self):
    self.head = {}

  def insert(self, word):
    cur = self
    for char in word:
      if char not in cur.head:
        cur.head[char] = Trie()
      cur = cur.head[char]
    cur.head['*'] = True

  def search(self, pattern):
    cur = self
    for char in pattern:
      if char not in cur.head:
        return False
      cur = cur.head[char]
    if '*' in cur.head:
      return True

  def insertAll(self, words):
    for i in words:
      self.insert(i)
```

```python
# Scrap stop words, positive words and negative words
# Execute once
def getStopWords():
  url = 'https://bit.ly/38uiVQH'
  stop_words = requests.get(url).text.split()
  saveObj(stop_words, "Stop Words")
  return stop_words

def getPosWords():
  url = 'https://bit.ly/3l73BfM'
  html_text = requests.get(url).text

  soup = BeautifulSoup(html_text, 'lxml')
  division = soup.find("div", class_ = 'entry-content')
  paras = list(division.find_all('p'))[3:-2]

  pos_words = [para.text.lower().replace('\xa0', '').split(', ') for para in paras]
  pos_words = list(chain.from_iterable(pos_words))
  pos_words = [word.strip() for word in pos_words]
  return pos_words
```

```python
# Step 2: Lemmatizing sentences
def nltk_pos_tagger(nltk_tag):
  if nltk_tag.startswith('J'):
    return wordnet.ADJ
  elif nltk_tag.startswith('V'):
    return wordnet.VERB
  elif nltk_tag.startswith('N'):
    return wordnet.NOUN
  elif nltk_tag.startswith('R'):
    return wordnet.ADV
  else:
    return None

def lemmatize(text):
  lemmatizer = WordNetLemmatizer()
  nltk_tagged = nltk.pos_tag(nltk.word_tokenize(text.lower()))
  wordnet_tagged = map(lambda x: (x[0], nltk_pos_tagger(x[1])), nltk_tagged)

  lemmed_sent = []
  for word,tag in wordnet_tagged:
    if tag is None:
      lemmed_sent.append(word)
    else:
      lemmed_sent.append(lemmatizer.lemmatize(word, tag))
  return lemmed_sent
```

```python
# Execute once
# Step 1: General Web Scraping
def scrapText(url, header_loc, taglist, filename):
  req = Request(url, headers={'User-Agent': 'Mozilla/5.0'})
  webpage = urlopen(req).read()

  soup = BeautifulSoup(webpage, 'lxml')
  text = ''

  headers = soup.find(header_loc[0], class_ = header_loc[1])
  headers = headers.find_all(re.compile('^h[1-6]$'))
  for header in headers:
    text += header.text + '\n'

  for i in range(len(taglist)):
    divisions = soup.find_all(taglist[i][0], class_ = taglist[i][1])
    for division in divisions:
      paras = division.find_all('p')
```

```python
# Ma
def getNegWords():
  url = 'https://bit.ly/3FKNvBM'
  html_text = requests.get(url).text

  soup = BeautifulSoup(html_text, 'lxml')
  division = soup.find("div", class_ = 'entry-content')
  paras = list(division.find_all('p'))[1:-1]

  neg_words = [para.text.lower().replace('\xa0', '').split(', ') for para in paras]
  neg_words = list(chain.from_iterable(neg_words))
  neg_words = [word.strip() for word in neg_words]
  return neg_words
```

```python
# Step 4: Filter Stop words
def filterStopWords(lst, stop_words_list):
  filtered = [i for i in lst if i[0] not in (stop_words_list and nltk.corpus.stopwords.words('english'))] ) as f:
  stop_word_num = len(lst) - len(filtered)
#scrapText('https://bit.ly/3wATftQ', ['div', 'container page ufonts'], [['div', 'entry mar-b-10 mar-t-10 pad-t-10 clearfix']], 'JP1')
#scrapText('https://bit.ly/3sP89vj', ['article', 'clearfix post-280 encyclopedia type-encyclopedia status-publish hentry'], [['article', 'clearfix post-280 encyclopedia type-encyclopedia status-publish hentry']], 'JP2')
#scrapText('https://bit.ly/3sP8NZY', ['section', 'cardHolder open ga-tracking'], [['section', 'cardHolder open ga-tracking']], 'JP3')
#scrapText('https://bit.ly/3wBkWUA', [None, None], [['div', 'article__content']], 'JP4')
#scrapText('https://bit.ly/3MBnWG0', ['div', 'top_wrapper'], [['section', 'composer_content']], 'JP5')

#stop_words_list = getStopWords()
#pos_words_list = getPosWords()
#neg_words_list = getNegWords()
#savePosNegWords(pos_words_list, neg_words_list)
```

```python
    pos_words_found_num = len(pos_words_found)
    neg_words_found_num = len(neg_words_found)
    pos_word_percentage = float(pos_words_found_num) * 100 / (pos_words_found_num + neg_words_found_num)
    pos_word_percentage = round(pos_word_percentage, 2)
    sum_percentage += pos_word_percentage
    sentiment = grade[int(pos_word_percentage // 20)]
```

```python
# Step 5: Find all positive and negative words in text
def findPosNegWords(lst):
  pos_words_found, neg_words_found = [], []

  with open('/content/drive/MyDrive/Colab Notebooks/Positive Word Tries', 'rb') as pos_file:
    pos_trie = pickle.load(pos_file)

    for i in range(len(lst)):
      if pos_trie.search(lst[i][0]):
        pos_words_found.append(lst[i][0])

  with open('/content/drive/MyDrive/Colab Notebooks/Negative Word Tries', 'rb') as neg_file:
    neg_trie = pickle.load(neg_file)

    for i in range(len(lst)):
      if neg_trie.search(lst[i][0]):
        neg_words_found.append(lst[i][0])
  #print(pos_words_found)
  #print(neg_words_found)
  return pos_words_found, neg_words_found
```

```python
dataframe.sort_values(by = 'Avg PWS (%)', ascending = False, inplace = True, ignore_index = True)
display(dataframe)

fig = px.bar(x = dataframe['Country'], y = dataframe['Avg PWS (%)'], labels = dict(x='Country', y = 'Avg PWS (%)') ,title = f"Bar Chart of comparing average PWS (%) between countries" )
fig.show()

dataframe.to_csv('/content/drive/MyDrive/Colab Notebooks/Overall Output')
```

```python
    df = df.append({'Article' : name,
                    'Original Words Count' : ori_word_count,
                    'Distinct Words Count' : distinct_word_count,
                    'Stop Words Count' : stop_word_num,
                    'Positive Words Count' : pos_words_found_num,
                    'Negative Words Count' : neg_words_found_num,
                    'Positive Words Percentage (%)' : pos_word_percentage,
                    'Overall Sentiment' : sentiment}, ignore_index = True)

  display(df)

  fig = px.bar(x= df['Article'], y=df['Original Words Count'], labels = dict(x='Article', y = 'Count') ,title=f"Bar Chart of Total Word Count" )
  fig.show()
  fig = px.bar(x= df['Article'], y=df['Distinct Words Count'], labels = dict(x='Article', y = 'Count') ,title=f"Bar Chart of Distinct Words Count" )
  fig.show()
  fig = px.bar(x= df['Article'], y=df['Stop Words Count'], labels = dict(x='Article', y = 'Count') ,title=f"Bar Chart of Stop Words Count" )
  fig.show()
  fig = px.bar(x= df['Article'], y=df['Positive Words Count'], labels = dict(x='Article', y = 'Count') ,title=f"Bar Chart of Positive Words Count" )
  fig.show()
  fig = px.bar(x= df['Article'], y=df['Negative Words Count'], labels = dict(x='Article', y = 'Count') ,title=f"Bar Chart of Negative Words Count" )
  fig.show()
  fig = px.bar(x= df['Article'], y=df['Positive Words Percentage (%)'], labels = dict(x='Article', y = 'Count') ,title=f"Bar Chart of Positive Words Percentage" )
  fig.show()

  df.to_csv('/content/drive/MyDrive/Colab Notebooks/' + str(lst[i]) + ' Output')
```

Question 2

2.1 Below shows the code of computing distance matrix

```
def calculate_distance_matrix(stores):
    distance_matrix= []
    for i in range(len(stores)):
        ori = stores.iloc[i]
        des_list = []
        for j in range(len(stores)):
            des = stores.iloc[j]
            #update the parameter to be passed into distance matrix
            param = {
                'origins' :str(ori['latitude'])+","+ str(ori['longitude']),  # pair up first location latitude and longitude
                'destinations' : str(des['latitude'])+","+ str(des['longitude']), # pair up second location latitude and
longitude
                'mode' : 'driving',
                'key' : API_KEY
            }

            response = requests.get(base_url, params = param)
            r = response.json()
            des_list.append(r['rows'][0]['elements'][0]['distance']['value'])
            # only get the distance between the 2 points
        distance_matrix.append(des_list)
    return distance_matrix
```

2.2 Below shows the code of finding distribution center

```
# find local distribution center, the location having shortest total distance from all point is the center

def find_distribution_center(distance_matrix):
  cost = {}

  for i in range(len(distance_matrix)):
      name = stores['name'].iloc[i]
      total_distance= 0
      for x in distance_matrix[i]:
          total_distance +=x

      cost[name]=total_distance

  center = min(cost, key=cost.get)
  # getting the store which has min length from all stores as the distribution center
  distribution_center = stores[stores['name']==center]  # retrieving info from dataset on distribution center

  center_index= distribution_center.index[0]
  print(center)
  # rearrange data so that distribution center is at the first index
  b, c = stores.iloc[0].copy(), stores.iloc[center_index].copy()
```

```
    stores.iloc[0],stores.iloc[center_index] = c,b
    distribution_center = stores[stores.index==0]

    return distribution_center
```

2.3 Below shows the code of held-karp algorithm

```
# dynamic programming with memoization

def held_karp(dists):

    n = len(dists)

    # Maps each subset of the nodes to the cost to reach that subset, as well
    # as what node it passed before reaching this subset.
    # Node subsets are represented as set bits.
    # format in C : '(Set bit,Subset),(Cost,Parent)'
    C = {}

    # Set transition cost from initial state
    for k in range(1, n):
        C[(1 << k, k)] = (dists[0][k], 0)

    # Iterate subsets of increasing length and store intermediate results
    # in classic dynamic programming manner
    for subset_size in range(2, n):
        for subset in itertools.combinations(range(1, n), subset_size):

            # Set bits for all nodes in this subset
            # we encode each set into binary
            # eg : {0,3,4}  = 11001
            bits = 0
            for bit in subset:
                bits |= 1 << bit

            # Find the lowest cost to get to this subset
            for k in subset:
                prev = bits & ~(1 << k)

                res = []
                for m in subset:
                    if m == 0 or m == k:
                        continue
                    res.append((C[(prev, m)][0] + dists[m][k], m))
                C[(bits, k)] = min(res)

    # We're interested in all bits but the least significant (the start state)
    bits = (2**n - 1) - 1
```

```
    # Calculate optimal cost
    res = []
    for k in range(1, n):
        res.append((C[(bits, k)][0] + dists[k][0], k))
    opt, parent = min(res)

    # Backtrack to find full path
    path = []
    path.append(0)
    for i in range(n - 1):
        path.append(parent)
        new_bits = bits & ~(1 << parent)
        _, parent = C[(bits, parent)]
        bits = new_bits

    # Add implicit start state
    path.append(0)

    return opt, list(reversed(path))
```

2.4 Below shows the code of plotting the stores location on map

```
def gmplot_stores(stores,filename):

    map =
gmplot.GoogleMapPlotter(float(distribution_center.latitude),float(distribution_center.longitude),4,apikey=
API_KEY)

    # plot stores
    for i in range(len(stores)):
        s = stores.iloc[i]
        if i ==0:
          map.marker(s['latitude'],s['longitude'],color='red',label=i,info_window=s['name'])
        else:
          map.marker(s['latitude'],s['longitude'],color='blue',label=i,info_window="Center: " + s['name'])


    map.draw( "/content/drive/MyDrive/Algo/"+filename )
```

2.5 Below shows the code of plotting the delivery path on map

```
def gmplot_paths(stores,route,filename):

    map =
gmplot.GoogleMapPlotter(float(distribution_center.latitude),float(distribution_center.longitude),4,apikey=
API_KEY)

    # plot stores
    for i in range(len(stores)):
        s = stores.iloc[i]
        map.marker(s['latitude'],s['longitude'],color='blue',label=i,info_window=s['name'])

    # plot distribution center
    # map.marker(float(distribution_center.latitude),float(distribution_center.longitude),color='orange')


    #plot route

#map.directions([float(distribution_center.latitude),float(distribution_center.longitude)],[float(stores.iloc[4]
['latitude']),float(stores.iloc[4]['longitude'])],color='red')
    for i in range(len(route)-1):
        start_index= route[i]
        end_index= route[i+1]
        starting_point = [float(stores.iloc[start_index]['latitude']),float(stores.iloc[start_index]['longitude'])]
        end_point = [float(stores.iloc[end_index]['latitude']),float(stores.iloc[end_index]['longitude'])]
        map.directions(starting_point,end_point,color='black')

    map.draw( "/content/drive/MyDrive/Algo/"+filename )
```

Question 3

3.1 Below shows the code of to import the Pandas library and initialize the sentiment dictionary obtained in Question 1 as well as the total distance dictionary obtained in Question 2

```
# import pandas library (can remove after integrating)
import pandas as pd

# use the average PWS of each country obtained from Question 1
sentiment_dict = {'AR': 56.28, 'CA': 75.57, 'CN': 63.05, 'JP': 80.77, 'US': 75.69}

# use the total distance of each country obtained from Question 2
distance_dict = {'AR': 22495, 'CN': 5145492, 'JP': 1272783, 'CA': 5475049, 'US': 16583822}

# reuse the lst in Question 1
lst = ['AR', 'CA', 'CN', 'JP', 'US']
```

3.2 Below shows the code of quicksort algorithm

```
# Function to find the partition position
def partition(array, low, high):
  # Choose the rightmost element as pivot
  pivot = array[high]
  # Pointer for greater element
  i = low - 1
  # Traverse through all elements to compare each element with pivot
  for j in range(low, high):
    if array[j] <= pivot:
      # If element is smaller than pivot, swap it with the greater element pointed by i
      i = i + 1
      # Swapping element at i with element at j
      (array[i], array[j]) = (array[j], array[i])
  # Swap the pivot element with the greater element specified by i
  (array[i + 1], array[high]) = (array[high], array[i + 1])
  # Return the position from where partition is done
  return i + 1
# Function to perform quicksort
def quick_sort(array, low, high):
  if low < high:
    # Find pivot element (element < pivot are on the left, element > pivot are on the right)
    pi = partition(array, low, high)
    # Recursive call on the left of pivot
    quick_sort(array, low, pi - 1)
    # Recursive call on the right of pivot
```

```
    quick_sort(array, pi + 1, high)
```

3.3 Below shows the code of min-max normalization

```
# to get the minimum value in the dictionary
def getMin(dict):
  firstKey = str(list(dict.keys())[0])
  min = dict[firstKey]
  for key in dict:
    if (dict[key] < min):
      min = dict[key]
  return min
# to get the maximum value in the dictionary
def getMax(dict):
  max = 0
  for key in dict:
    if (dict[key] > max):
      max = dict[key]
  return max
def minMaxNormalization(min, max, value):
  return ((value - min) / (max - min))
```

3.4 Below shows the code of applying weighted sum model

```
# Step 1: apply weight to each normalized score in the dictionary
def applyWeight(dict, weight):
  weighted_dict = {}
  for key in dict:
    weightedScore = dict[key] * weight
    weighted_dict[key] = weightedScore
  return weighted_dict
# Step 2: sum up the weighted sentiment score and weighted distance score to obtain the final score
for each country
def sumWeightedScore(weightedPWS_dict, weightedDistance_dict):
  finalScore_dict = {}
  for country in weightedPWS_dict:
    finalScore = round((weightedPWS_dict[country] + weightedDistance_dict[country]), 2)
    finalScore_dict[country] = finalScore
  return finalScore_dict
```

3.5 Below shows the code of calculating probability and analyze ranking

```
# calculate the probability of a country to be chosen to have an expansion among the 6 countries
def computeProbability(finalScore_list):
  totalFinalScore = sum(finalScore_list)
```

```
  probability_list = []
  for score in finalScore_list:
    probability = round((score / totalFinalScore), 4)
    probability_list.append(probability)
  return (dict(zip(lst, probability_list)))
# sort the countries based on thier respective final score
def analyseRanking(finalScore_list, finalScore_dict):
  quick_sort(finalScore_list, 0, (len(finalScore_list)-1))
  rank_dict = {}
  for i in range(len(finalScore_list)-1, -1, -1):
    for country in finalScore_dict:
      if (finalScore_list[i] == finalScore_dict[country]):
        rank_dict[country] = finalScore_list[i]
  return rank_dict
# display the ranking of countries where new stores can be located in the form of dataframe
def displayRanking(rank_dict, sentiment_dict, distance_dict, probability_dict):
  rank_num = [1, 2, 3, 4, 5]
  count = 0
  df = pd.DataFrame(columns = ['Ranking', 'Country', 'Avg PWS (%)', 'Total journey for deliveries
(miles)', 'Score', 'Probability'])
  for country in rank_dict:
    df = df.append({'Ranking': rank_num[count],
                    'Country': country,
                    'Avg PWS (%)': sentiment_dict[country],
                    'Total journey for deliveries (miles)': distance_dict[country],
                    'Score': rank_dict[country],
                    'Probability': probability_dict[country]},
                    ignore_index = True)
    count += 1
  df = df.set_index("Ranking")
  print("Ranking of most recommended country to have expansion:\n")
  display(df)
  country_name = {'AR': 'Argentina', 'CA': 'Canada', 'CN': 'China', 'JP': 'Japan', 'US': 'United States'}
  print(f"\nThe most recommended country to have expansion is
{country_name[list(rank_dict.keys())[0]]}.")
  print(f"The least recommended country to have expansion is
{country_name[list(rank_dict.keys())[4]]}.")
```

3.6 Below shows the driver code to solve Question 3

```
# Perform min-max normalization on the average PWS obtained in Question 1
minPWS = getMin(sentiment_dict)
maxPWS = getMax(sentiment_dict)
normalizedPWS_dict = {}
for country in sentiment_dict:
  normalizedPWS = minMaxNormalization(minPWS, maxPWS, sentiment_dict[country])
  normalizedPWS_dict[country] = normalizedPWS

# Perform min-max normalization on the total distance obtained in Question 2
minDistance = getMin(distance_dict)
maxDistance = getMax(distance_dict)
normalizedDistance_dict = {}
for country in distance_dict:
  normalizedDistance = minMaxNormalization(maxDistance, minDistance, distance_dict[country])
  normalizedDistance_dict[country] = normalizedDistance

# Apply weighted sum model
# both weight of the sentiment and distance are the same (0.5)
weight_sentiment = 0.5
weight_distance = 0.5
weightedPWS_dict = applyWeight(normalizedPWS_dict, weight_sentiment)
weightedDistance_dict = applyWeight(normalizedDistance_dict, weight_distance)
finalScore_dict = sumWeightedScore(weightedPWS_dict, weightedDistance_dict)

# Compute probabiltiy of a country to be chosen to have expansion among the 6 countries
finalScore_list = list(finalScore_dict.values())
probability_dict = computeProbability(finalScore_list)

# Analyse ranking of countries using Quicksort Algorithm
rank_dict = analyseRanking(finalScore_list, finalScore_dict)

# Display the ranking
displayRanking(rank_dict, sentiment_dict, distance_dict, probability_dict)
```

5. Results

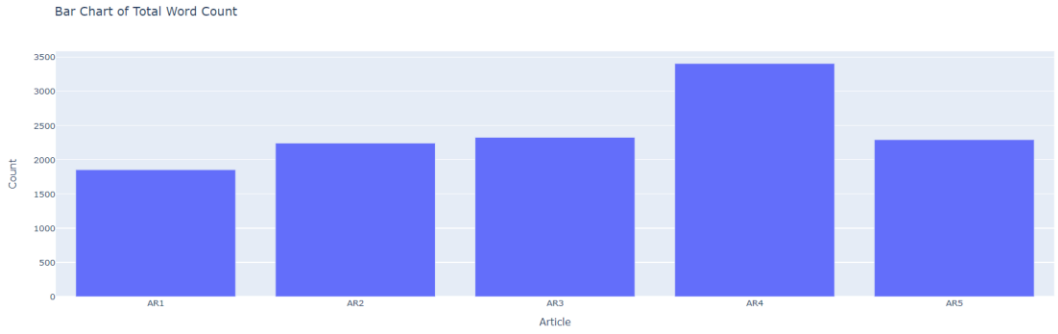
From the sample dataset, we choose Argentina(AR), China(CN), Japan(JP), Canada(CA) and the United States(US) to analyze. Below are our findings for each country.

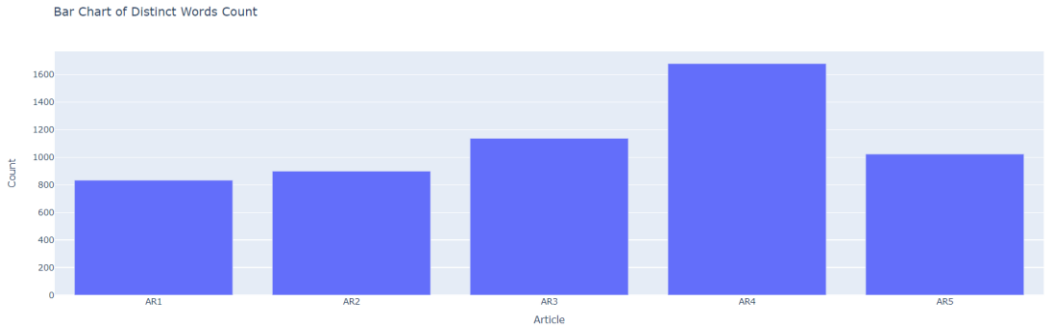
Argentina (AR)

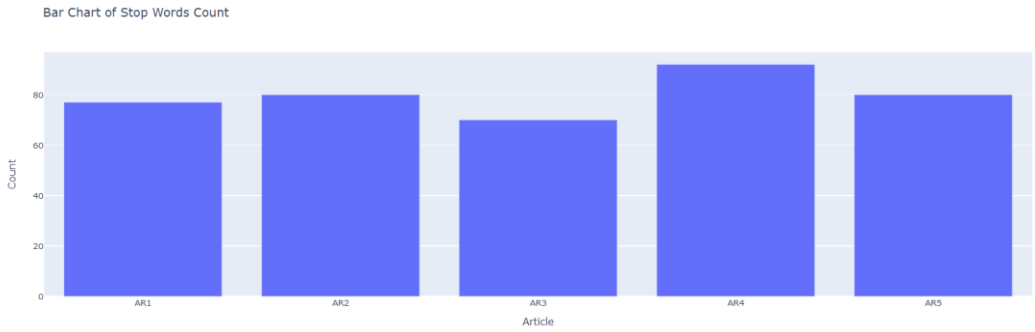
1. Below shows the analysis based on 5 articles about Argentina.

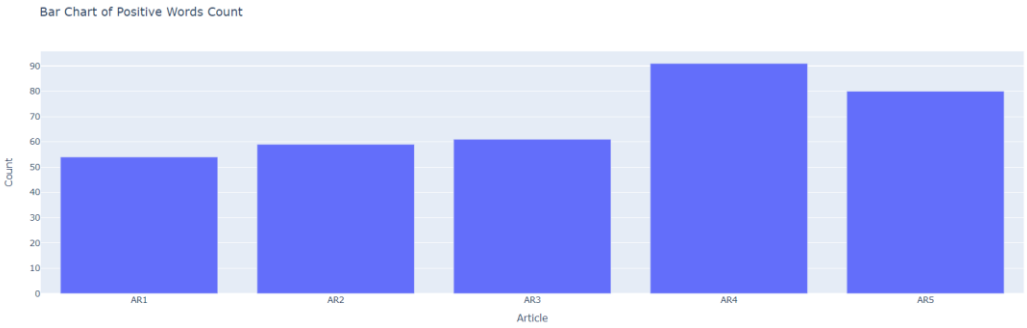
Article	Original Words Count	Distinct Words Count	Stop Words Count	Positive Words Count	Negative Words Count	Positive Words Percentage (%)	Overall Sentiment
AR1	1854	833	77	54	46	54.00	Neutral
AR2	2243	899	80	59	49	54.63	Neutral
AR3	2328	1137	70	61	44	58.10	Neutral
AR4	3407	1679	92	91	63	59.09	Neutral
AR5	2294	1023	80	80	64	55.56	Neutral

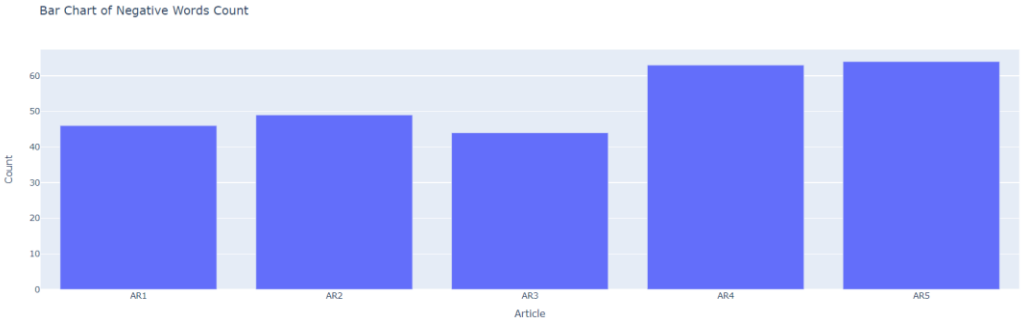
2. Below shows the graphs based on analysis on Argentina.

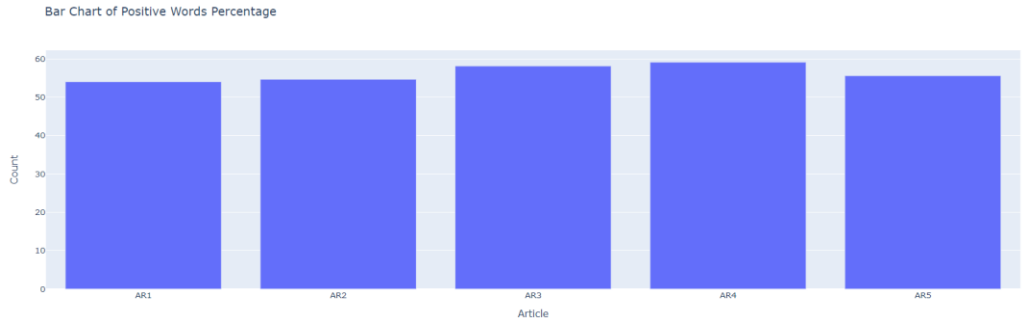






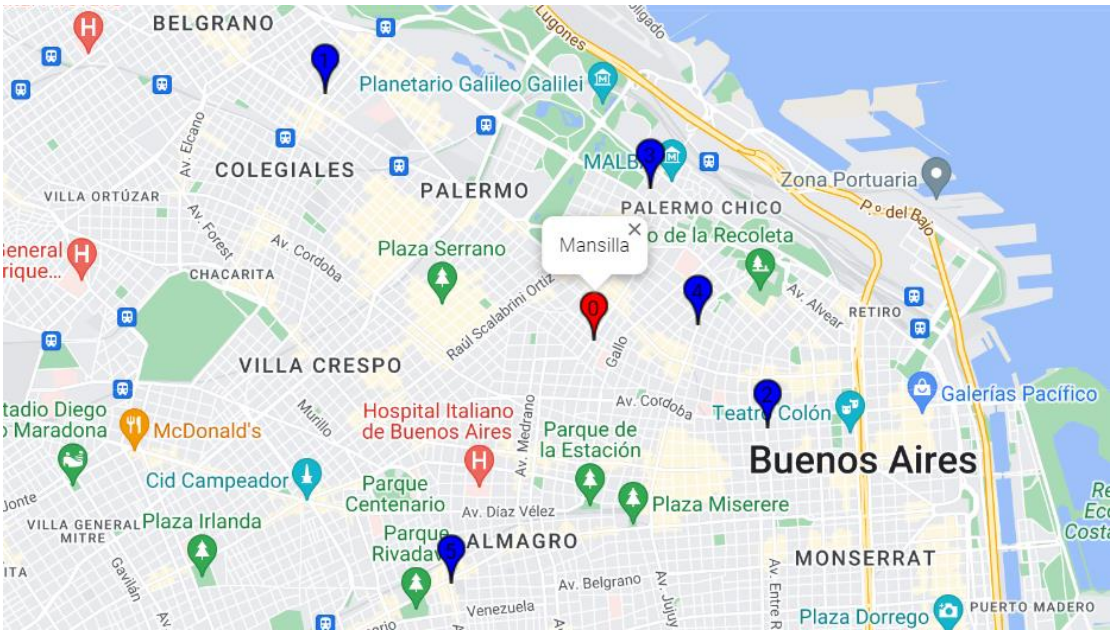






3. Below shows the location of the stores selected in Argentina.

Selected stores:

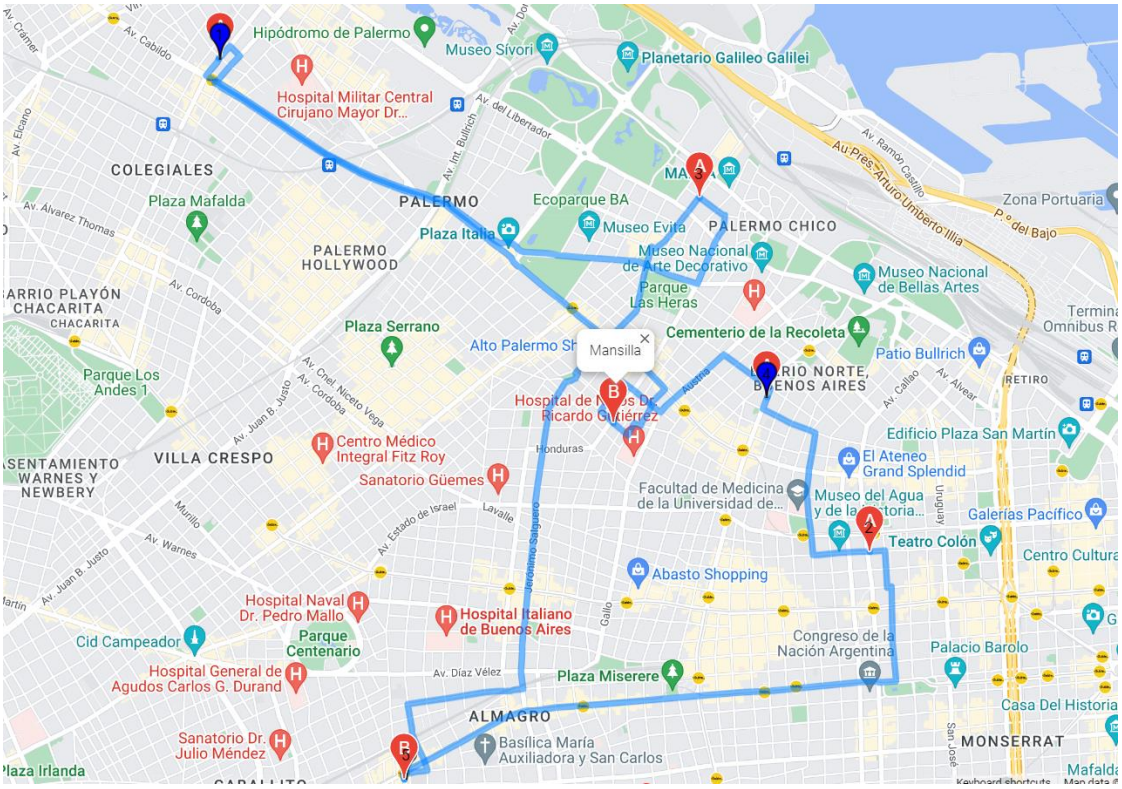


4. Below shows the distance matrix, shortest path and total distance for the delivery.

Distance matrix :
[[0, 4864, 5088, 2315, 2735, 5497],
[4688, 0, 9180, 5043, 6827, 7257],
[4846, 7320, 0, 3752, 2506, 4810],
[2146, 4431, 2536, 0, 1835, 3590],
[2557, 5609, 1821, 1662, 0, 4865],
[5559, 7257, 5080, 4010, 4571, 0]]

Shortest path : [0, 4, 2, 5, 3, 1, 0]
Total distance : 22495 miles

5. Below shows the plotting of the path for the delivery.

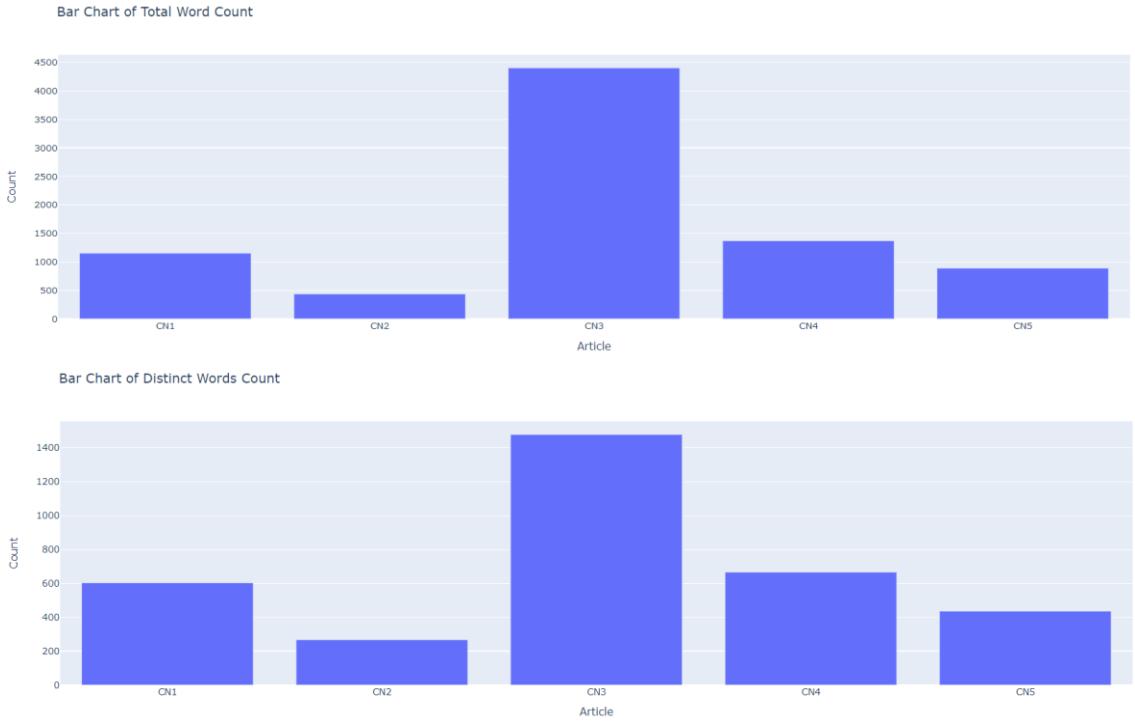

Hipódromo de Palermo
Museo Sívori
Planetario Galileo Galilei
Hospital Militar Central Cirujano Mayor Dr...
COLEGIALES
Plaza Mafalda
PALERMO
Ecoparque BA
Museo Evita
PALERMO CHICO
Plaza Italia
PALERMO HOLLYWOOD
Museo Nacional de Arte Decorativo
Museo Nacional de Bellas Artes
Parque Las Heras
Zona Portuaria
Plaza Serrano
Cementerio de la Recoleta
Mansilla
Patio Bullrich
Parque Los Andes 1
Centro Médico Integral Fitz Roy
VILLA CRESPO
Sanatorio Güemes
Facultad de Medicina de la Universidad de...
Edificio Plaza San Martín
El Ateneo Grand Splendid
Museo del Agua y de la ...toria...
Galerías Pacífico
Teatro Colón
Abasto Shopping
Hospital Naval Dr. Pedro Mallo
Hospital Italiano de Buenos Aires
Cid Campeador
Parque Centenario
Congreso de la Nación Argentina
Palacio Barolo
Hospital General de Agudos Carlos G. Durand
Plaza Miserere
ALMAGRO
Sanatorio Dr. Julio Méndez
Basílica María Auxiliadora y San Carlos
MONSERRAT
Casa Del Historia

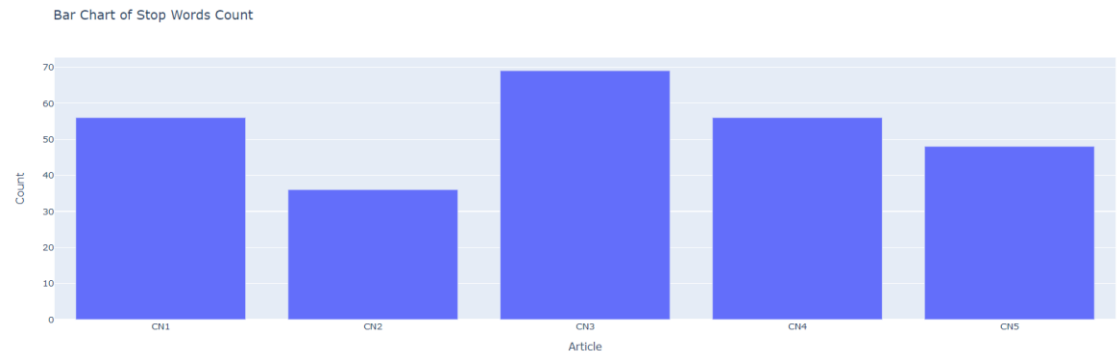
China (CN)

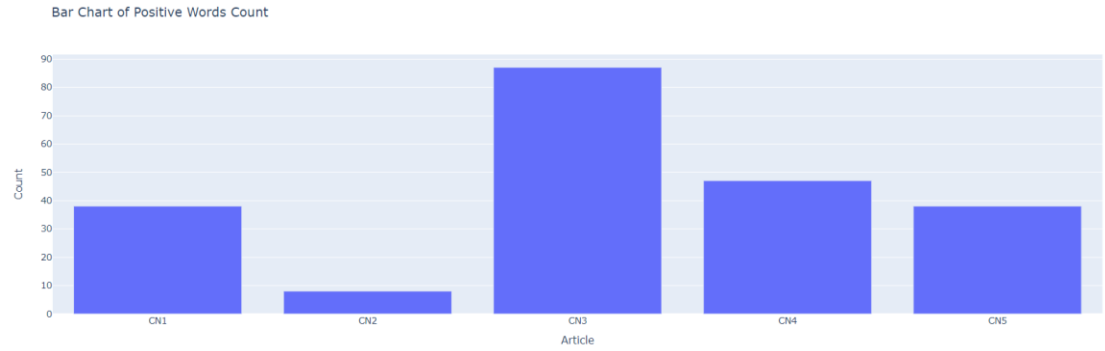
1. Below shows the analysis based on 5 articles about China.

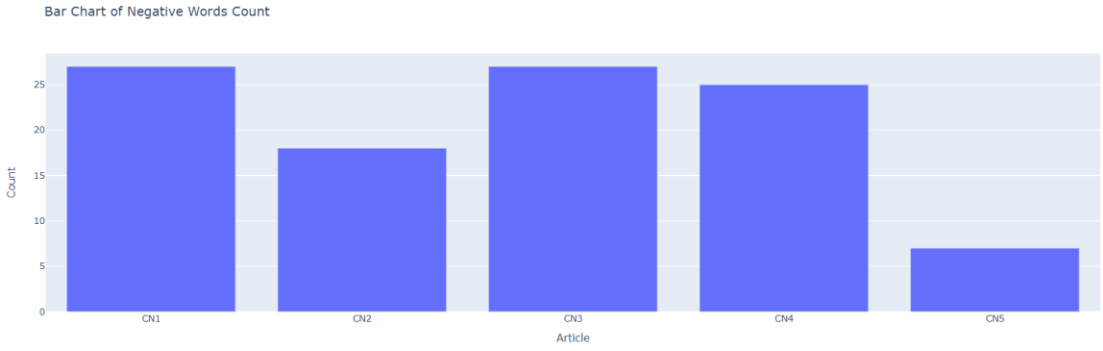
Article	Original Words Count	Distinct Words Count	Stop Words Count	Positive Words Count	Negative Words Count	Positive Words Percentage (%)	Overall Sentiment
CN1	1151	603	56	38	27	58.46	Neutral
CN2	436	267	36	8	18	30.77	Negative
CN3	4401	1479	69	87	27	76.32	Positive
CN4	1369	666	56	47	25	65.28	Positive
CN5	888	436	48	38	7	84.44	Very Positive

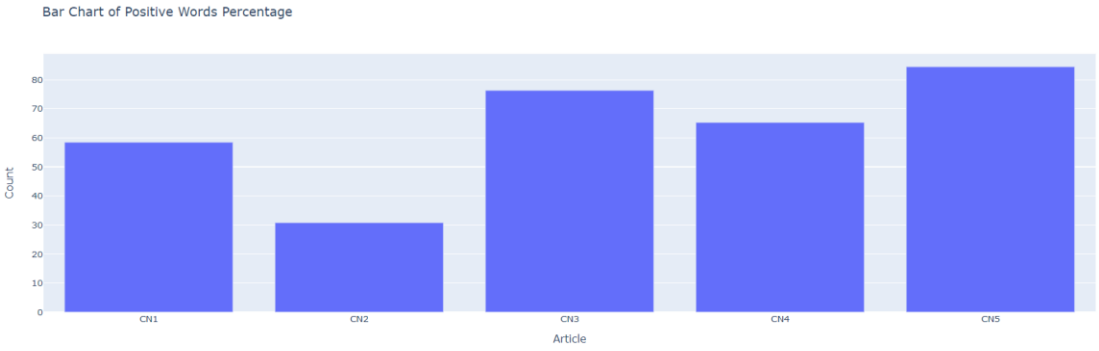
2. Below shows the graphs based on analysis on China.





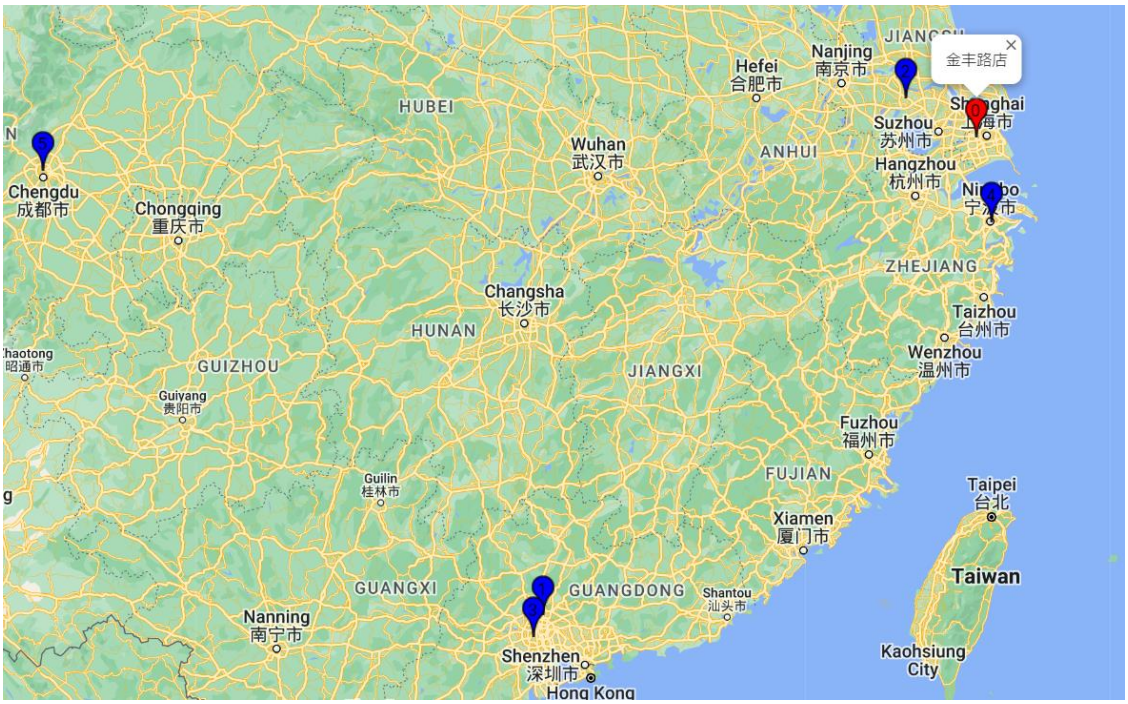






3. Below shows the location of the stores selected in China.

Selected stores:



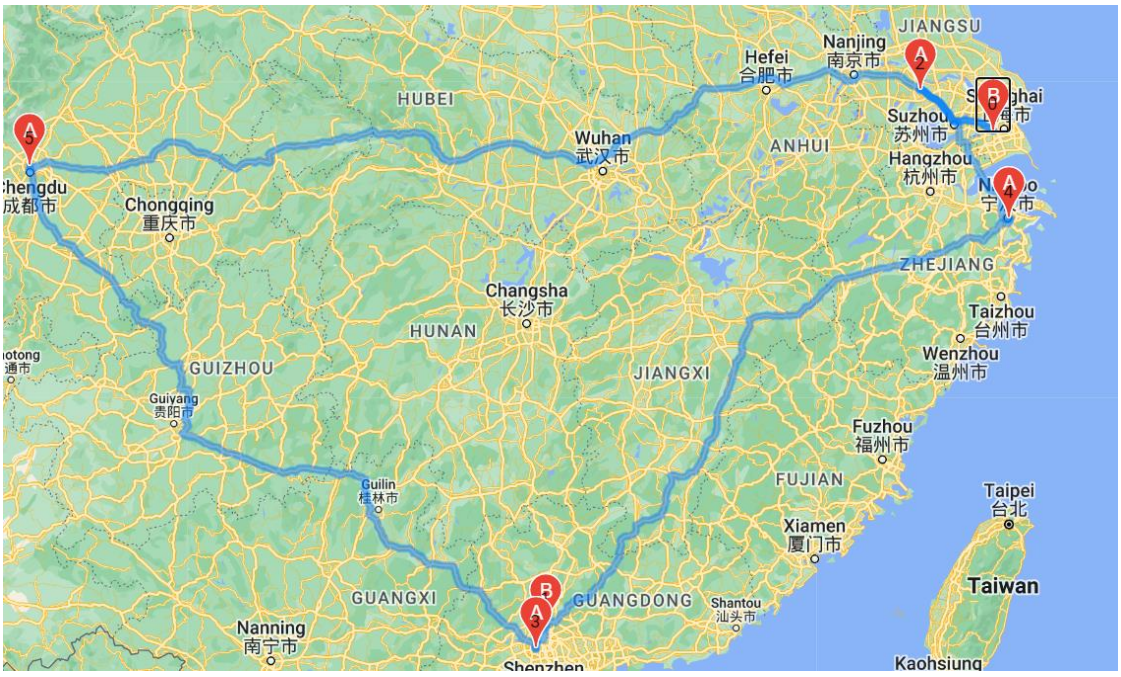
4. Below shows the distance matrix, shortest path and total distance for the delivery.

Distance matrix:

[[0, 1454531, 159398, 1496811, 313464, 1790801],
[1472495, 0, 1442491, 56802, 1376248, 1608345],
[160726, 1408470, 0, 1450750, 213040, 1945721],
[1516630, 57968, 1486625, 0, 1424707, 1590988],
[312993, 1334922, 212592, 1377203, 0, 2040407],
[1790342, 1610486, 1944733, 1591504, 2034859, 0]]

Shortest path : [0, 2, 4, 1, 3, 5, 0]
Total distance : 5145492 miles

5. Below shows the plotting of the path for the delivery.

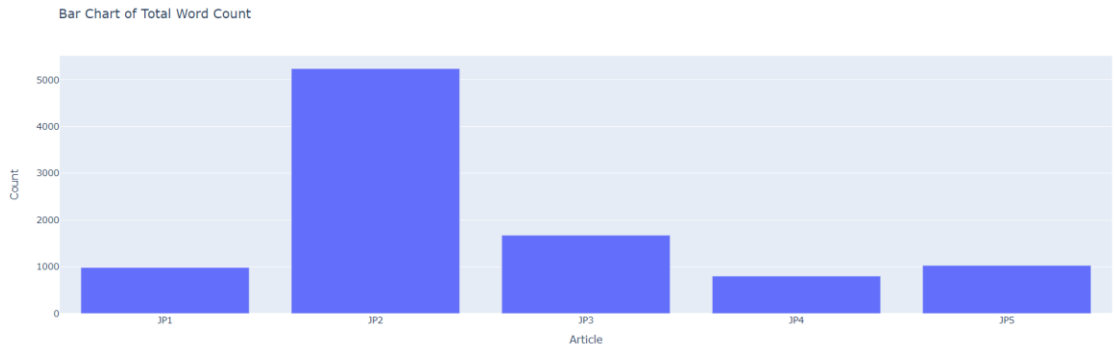

JIANGSU
Nanjing
南京市
Hefei
合肥市
HUBEI
Wuhan
武汉市
ANHUI
Suzhou
苏州市
Hangzhou
杭州市
Chengdu
成都市
Chongqing
重庆市
ZHEJIANG
Changsha
长沙市
HUNAN
Taizhou
台州市
Wenzhou
温州市
GUIZHOU
JIANGXI
Guiyang
贵阳市
Fuzhou
福州市
Guilin
桂林市
FUJIAN
Taipei
台北
Xiamen
厦门市
Taiwan
GUANGXI
GUANGDONG
Shantou
汕头市
Nanning
南宁市
Kaohsiung

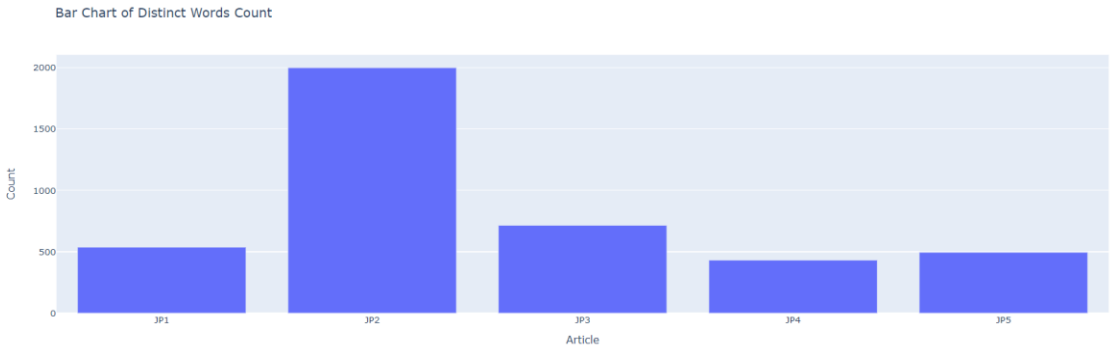
Japan (JP)

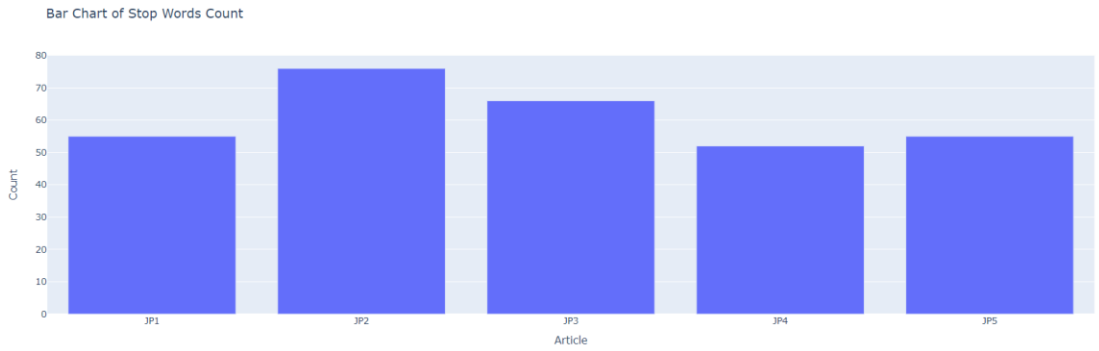
1. Below shows the analysis based on 5 articles about Japan.

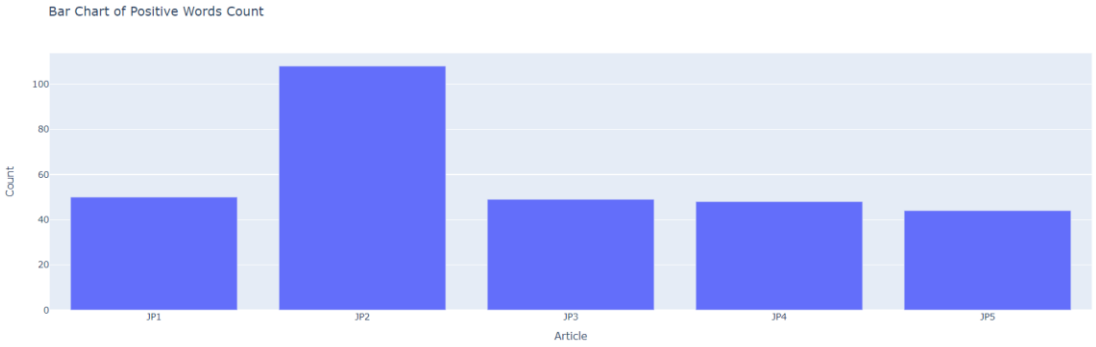
Article	Original Words Count	Distinct Words Count	Stop Words Count	Positive Words Count	Negative Words Count	Positive Words Percentage (%)	Overall Sentiment
JP1	985	537	55	50	9	84.75	Very Positive
JP2	5240	1999	76	108	60	64.29	Positive
JP3	1675	715	66	49	14	77.78	Positive
JP4	800	431	52	48	11	81.36	Very Positive
JP5	1028	495	55	44	2	95.65	Very Positive

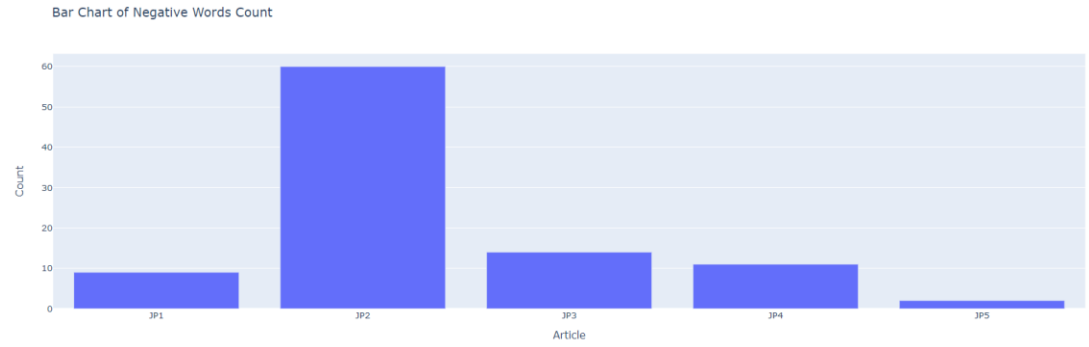
2. Below shows the graphs based on analysis on Japan.

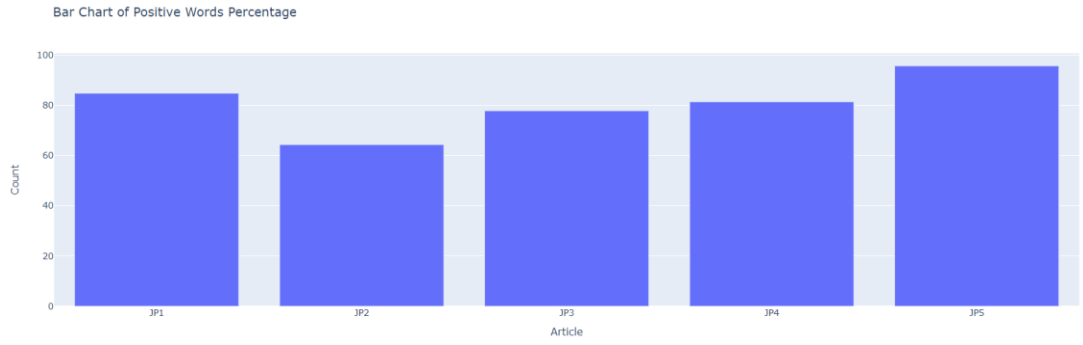




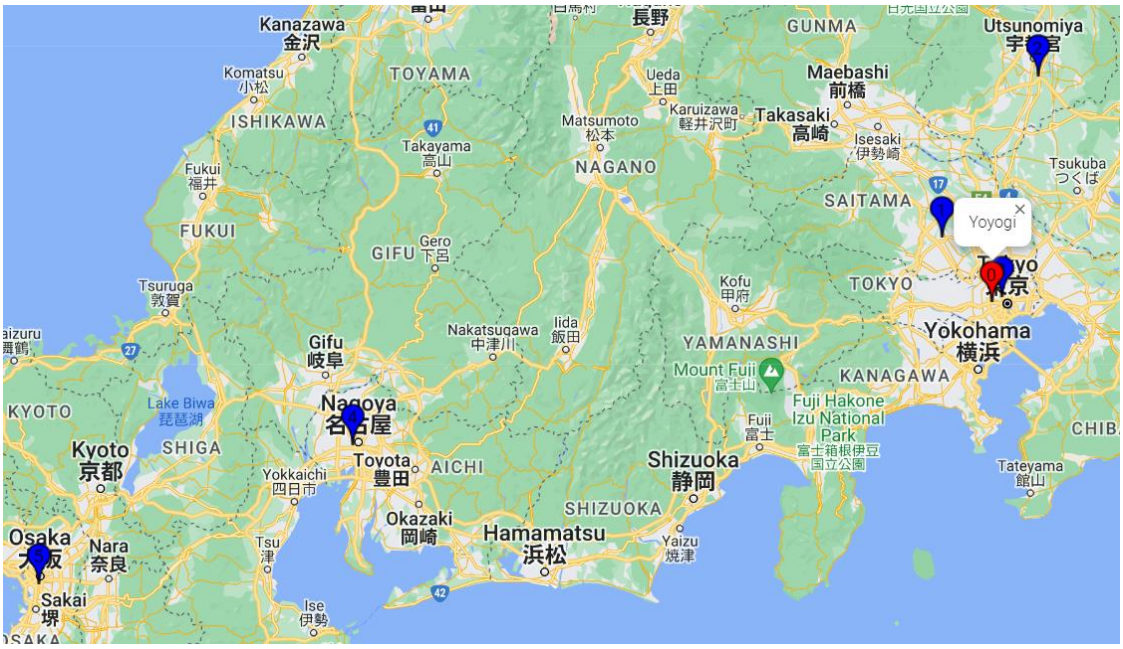








3. Below shows the location of the stores selected in Japan.

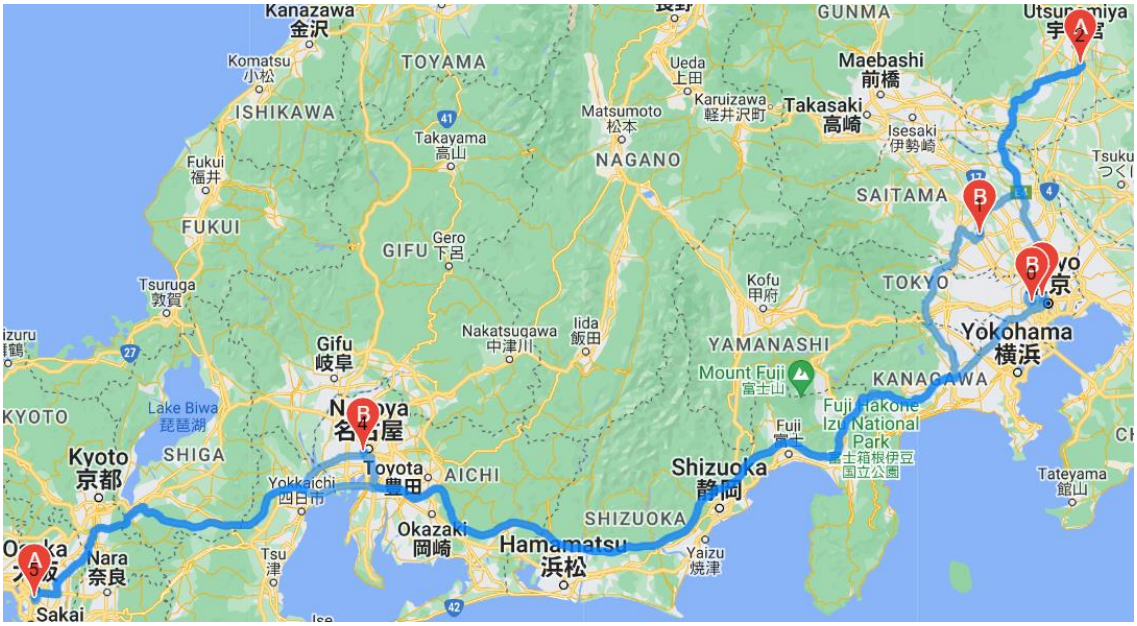


4. Below shows the distance matrix, shortest path and total distance for the delivery.

Distance matrix:
[[0, 52276, 120817, 6305, 349744, 503170],
[51231, 0, 100443, 42880, 377332, 530757],
[121155, 101217, 0, 123828, 467015, 620440],
[5553, 43312, 123568, 0, 344347, 497772],
[350696, 378231, 467420, 345753, 0, 171271],
[501727, 529262, 618451, 496784, 171751, 0]]

Shortest path : [0, 3, 4, 5, 1, 2, 0]
Total distance : 1272783 miles

5. Below shows the plotting of the path for the delivery.

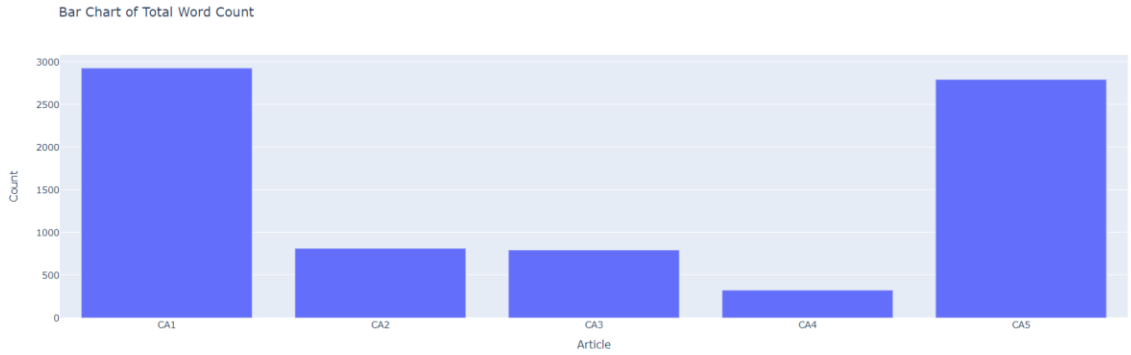

Kanazawa
金沢
GUNMA
TOYAMA
Komatsu
小松
Ueda
上田
Maebashi
前橋
ISHIKAWA
Matsumoto
松本
Karuizawa
軽井沢町
Takasaki
高崎
Takayama
高山
NAGANO
Isesaki
伊勢崎
Fukui
福井
SAITAMA
FUKUI
GIFU
Gero
下呂
Tsuruga
敦賀
Kofu
甲府
TOKYO
Nakatsugawa
中津川
Iida
飯田
Gifu
岐阜
YAMANASHI
Yokohama
横浜
Mount Fuji
富士山
KANAGAWA
KYOTO
Lake Biwa
琵琶湖
Fuji-Hakone-Izu National Park
富士箱根伊豆国立公園
Kyoto
京都
SHIGA
Toyota
豊田
AICHI
Shizuoka
静岡
Tateyama
館山
Yokkaichi
四日市
Okazaki
岡崎
SHIZUOKA
Hamamatsu
浜松
Nara
奈良
Tsu
津
Yaizu
焼津
Sakai

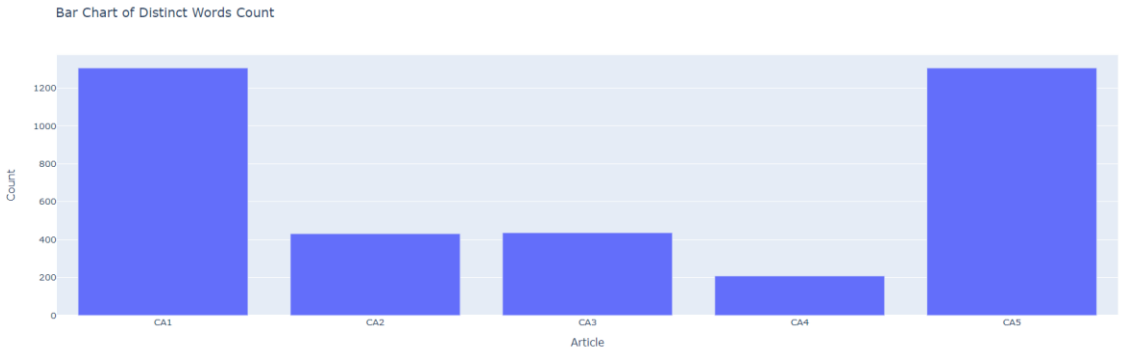
Canada(CA)

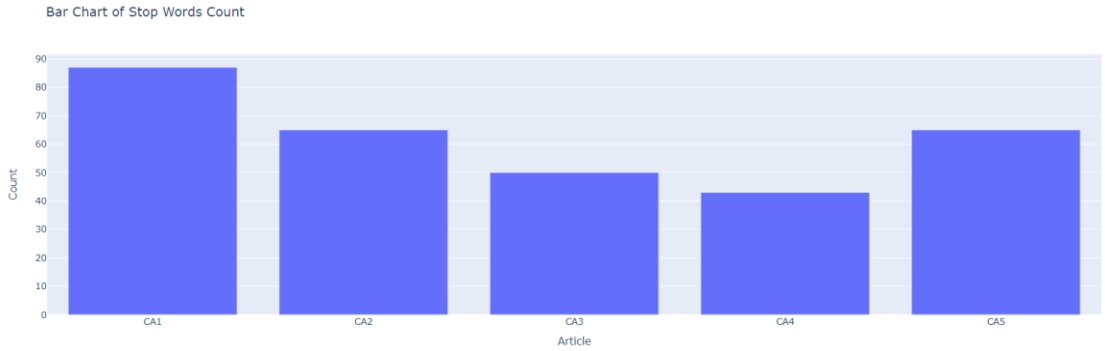
1. Below shows the analysis based on 5 articles about Canada.

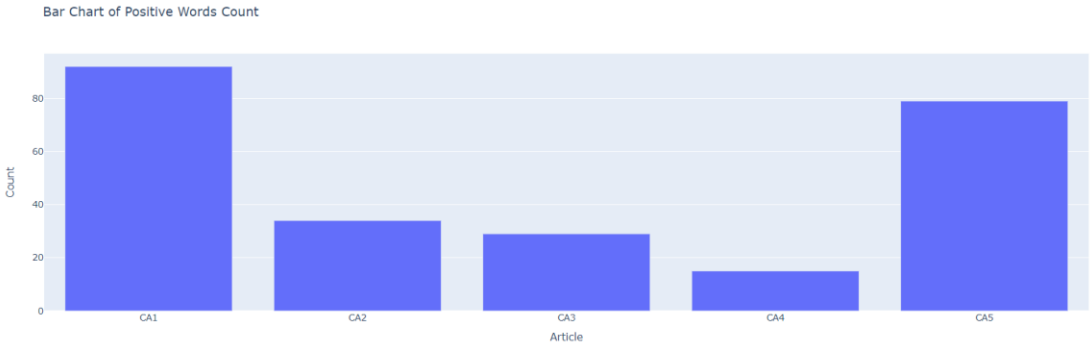
Article	Original Words Count	Distinct Words Count	Stop Words Count	Positive Words Count	Negative Words Count	Positive Words Percentage (%)	Overall Sentiment
CA1	2928	1305	87	92	42	68.66	Positive
CA2	815	431	65	34	8	80.95	Very Positive
CA3	795	436	50	29	11	72.50	Positive
CA4	325	208	43	15	2	88.24	Very Positive
CA5	2794	1305	65	79	38	67.52	Positive

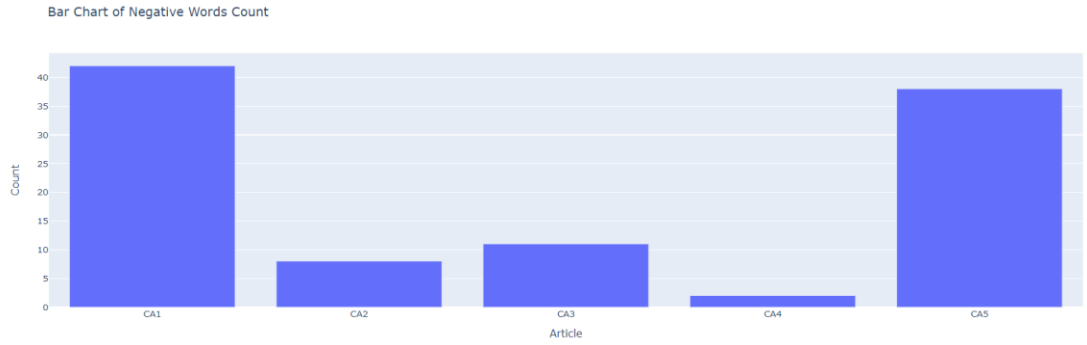
2. Below shows the graphs based on analysis on Canada.

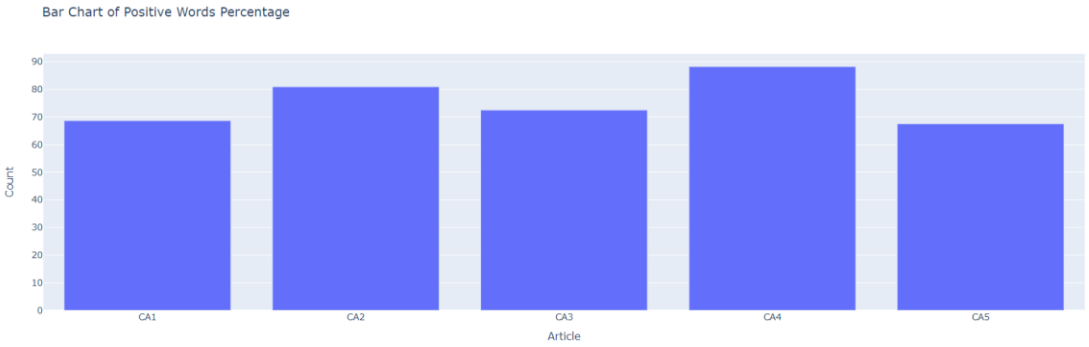






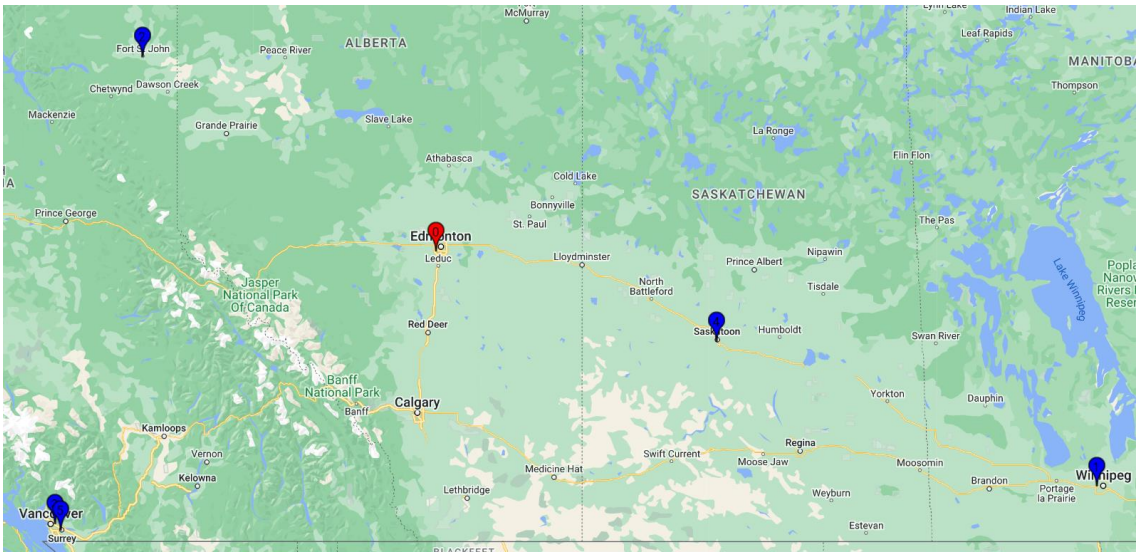






3. Below shows the location of the stores selected in Canada.

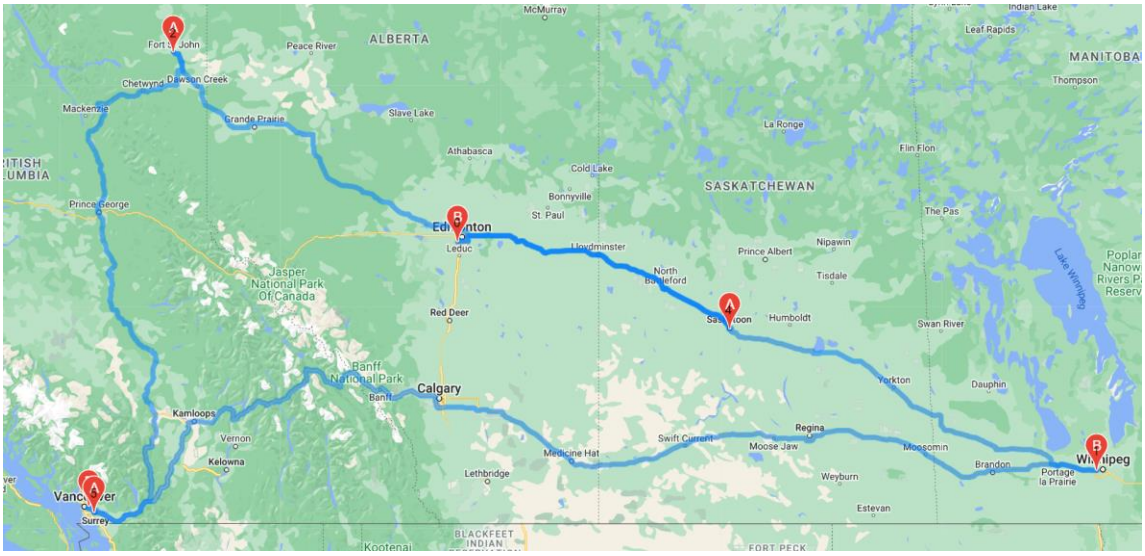
Selected stores:
Red - distribution center
Blue - other stores



4. Below shows the distance matrix, shortest path and total distance for the delivery.
Distance matrix:
[[0, 768288, 1174347, 1554417, 537454, 1536895],
[768452, 0, 1941540, 2285230, 1304647, 2267708],
[1180260, 1948093, 0, 1212592, 666416, 1195070],
[1555481, 2283062, 1212167, 0, 1158667, 26083],
[538128, 1305961, 665135, 1160154, 0, 1142632],
[1537753, 2265333, 1194439, 31806, 1140938, 0]]

Shortest path : [0, 4, 2, 3, 5, 1, 0]
Total distance : 5475049 miles

5. Below shows the plotting of the path for the delivery.

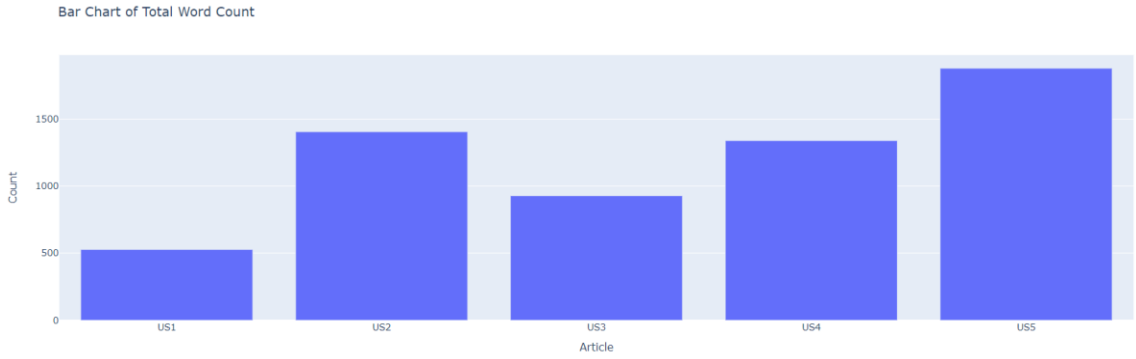


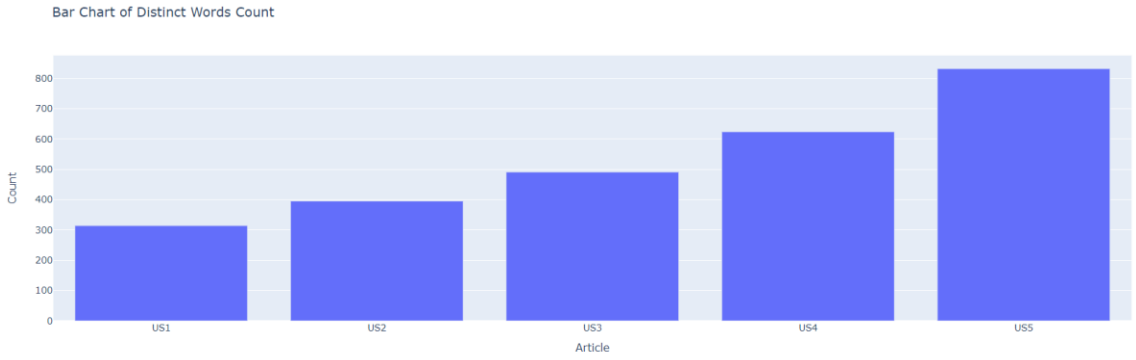
United States (US)

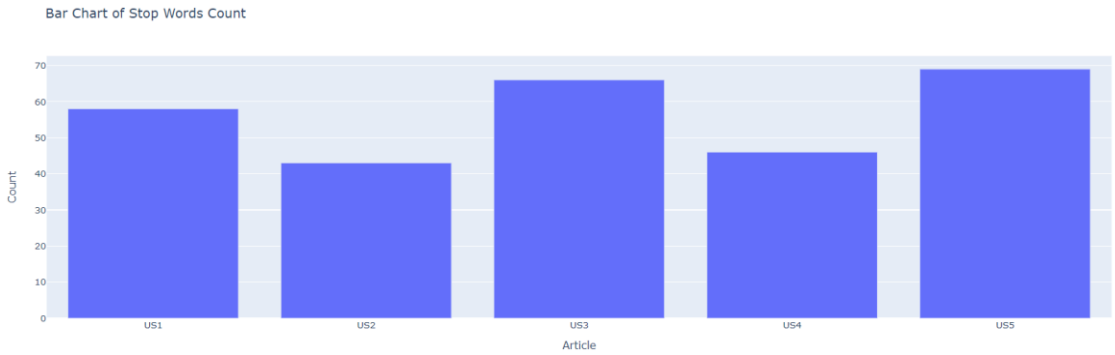
1. Below shows the analysis based on 5 articles about United States.

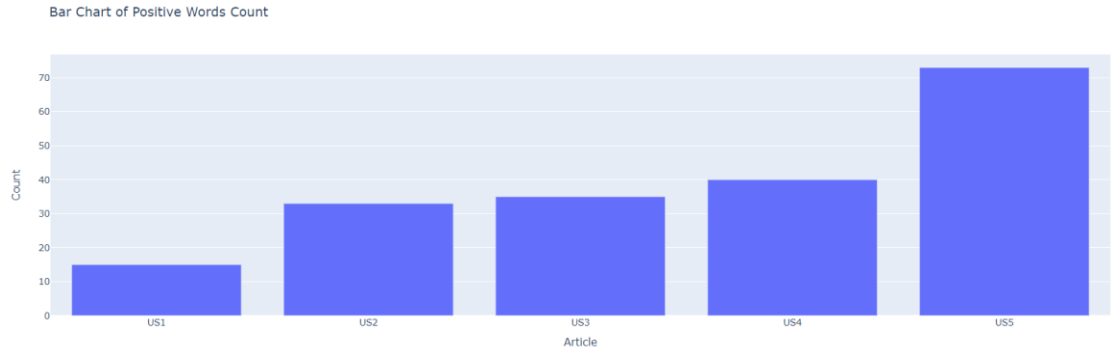
Article	Original Words Count	Distinct Words Count	Stop Words Count	Positive Words Count	Negative Words Count	Positive Words Percentage (%)	Overall Sentiment
US1	527	314	58	15	4	78.95	Positive
US2	1404	395	43	33	12	73.33	Positive
US3	927	491	66	35	22	61.40	Positive
US4	1338	624	46	40	2	95.24	Very Positive
US5	1877	832	69	73	32	69.52	Positive

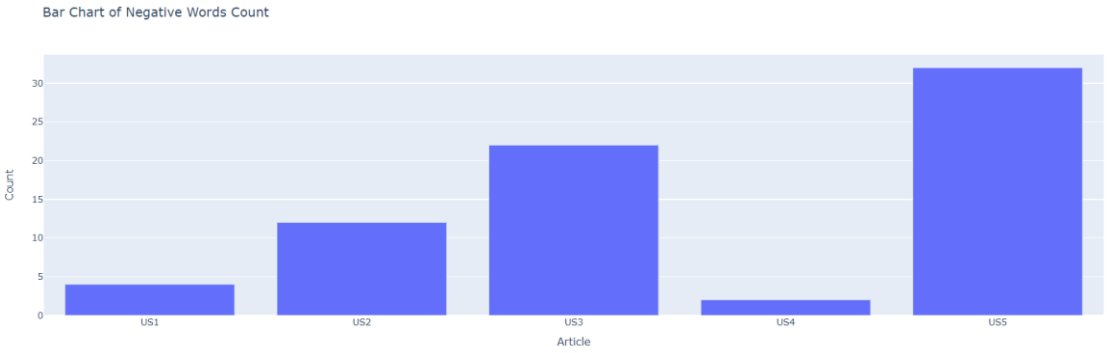
2. Below shows the graphs based on analysis on United States.

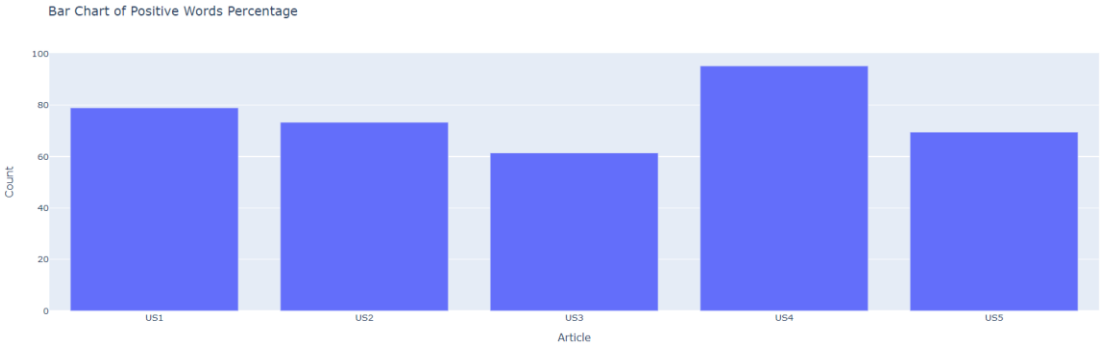






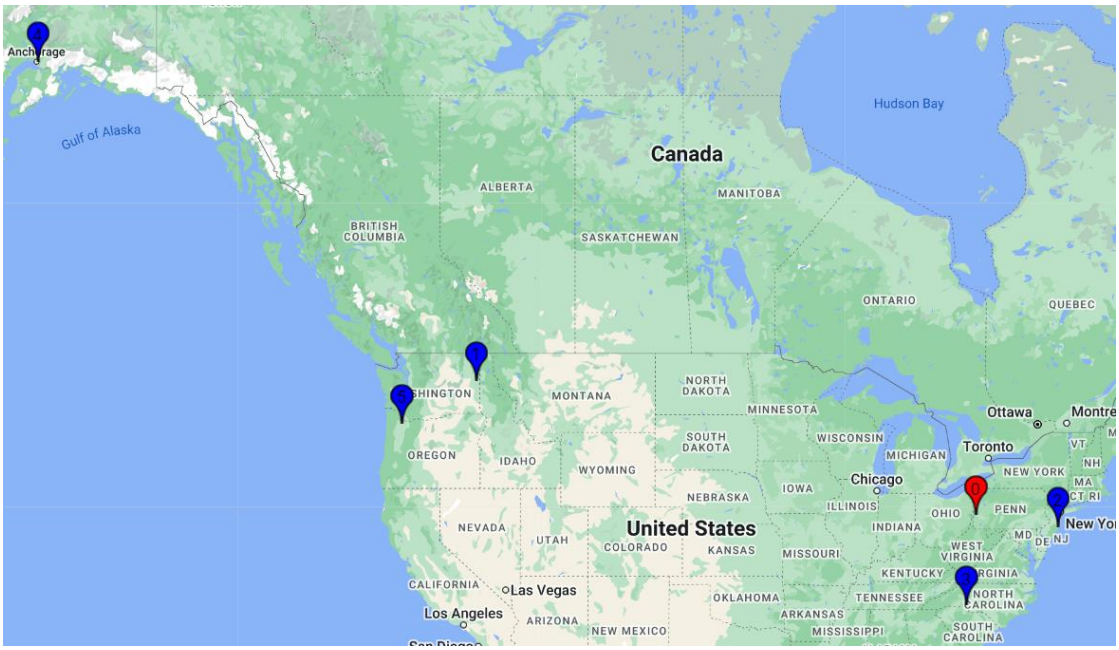






3. Below shows the location of the stores selected in the United States.

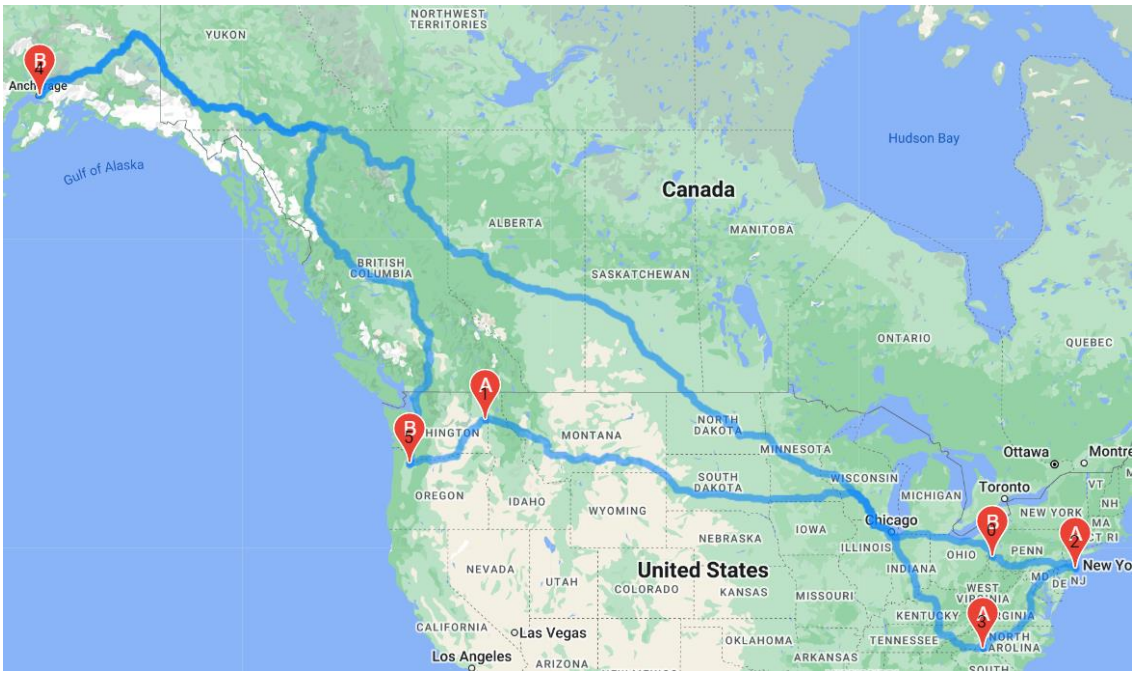
Selected stores:
Red - distribution center
Blue - other stores



4. Below shows the distance matrix, shortest path and total distance for the delivery.
   Distance matrix:
   [[0, 4023687, 952149, 731749, 6970627, 4439804],
   [4023342, 0, 4150928, 3550606, 3834332, 597534],
   [954670, 4184828, 0, 623001, 7059979, 4730504],
   [731725, 3585420, 622627, 0, 6460571, 4131097],
   [6975003, 3837312, 7069030, 6468708, 0, 3929916],
   [4437457, 594843, 4729351, 4129029, 3924529, 0]]

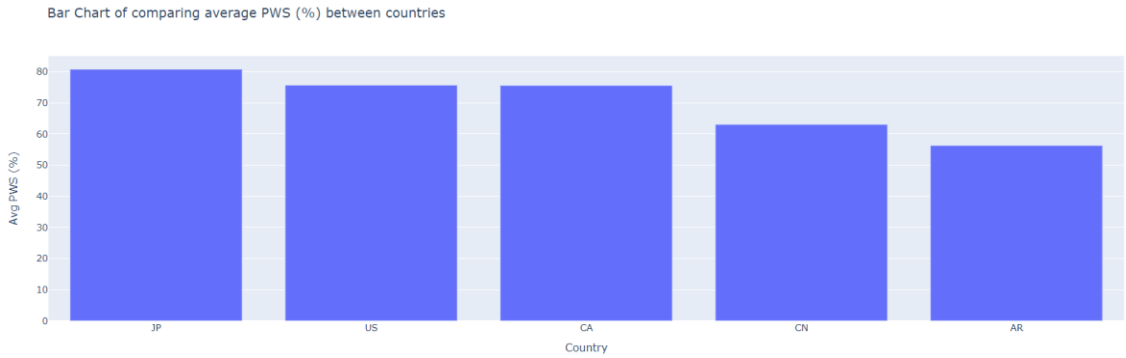
   Shortest path : [0, 2, 3, 4, 5, 1, 0]
   Total distance : 16583822 miles

5. Below shows the plotting of the path for the delivery.



Below shows the overall sentiment for each country calculated from Question 1.

Country	Avg PWS (%)	Overall Sentiment
JP	80.77	Very Positive
US	75.69	Positive
CA	75.57	Positive
CN	63.05	Positive
AR	56.28	Neutral



Below shows the total distance for the delivery in each country calculated from Question 2.

Country	Total Distance (miles)
AR	22495
CN	5145492
JP	1272783
CA	547049
US	16583822

Below shows the probability of a country that has a good local economic and social situation with the lowest optimal delivery calculated from Question 3.

Ranking of most recommended country to have expansion:

Ranking	Country	Avg PWS (%)	Total journey for deliveries (miles)	Score	Probability
1	JP	80.77	1272783	0.96	0.3127
2	CA	75.57	5475049	0.73	0.2378
3	AR	56.28	22495	0.50	0.1629
4	CN	63.05	5145492	0.48	0.1564
5	US	75.69	16583822	0.40	0.1303

The most recommended country to have expansion is Japan.
The least recommended country to have expansion is United States.

6. Conclusion

In short, we use Web Scraping, NLP and Tries algorithm to analyze the local economic and social situations of five countries which are Argentina(AR), China(CN), Japan(JP), Canada(CA) and the United States(US) based on the overall sentiment of online news websites for each country. With this method, we are able to see the feedback and economic trends of a country to maximize the profit of the business. It is best to run a business where the country has a positive social situation and stable economy. From our result, Japan is the country with the best overall sentiment which is very positive while Argentina only has neutral overall sentiment compared to three other countries whose overall sentiment is positive.

Besides, we used Google Distance Matrix API and Held-Karp algorithm to further compute the distance matrix and optimal delivery route necessary to reduce the traveling costs for each delivery truck. It is indeed beneficial for logistics management for business or commercial companies in each country to manage their branch stores well and boost their products' marketing in a more efficient way. By using a dynamic approach, our solution can scale better when the number of delivery locations increases as the country expands in large scales in the future compared to brute force. As for the final outcome shown under the "results" section, we can see that the United States of America has the longest distance delivery route (16583822 kilometers) while Argentina has the shortest path (22495 kilometers) across the stores within the country itself. However, our result changes for every execution as we randomized the stores selected in each country.

According to the final output generated, the ranking of most recommended countries to have expansion is Japan, Canada, Argentina, China and the United States in order. The ranking is sorted using the quicksort algorithm based on the final score computed after doing normalization and applying the weighted sum model. The most recommended country to have expansion is Japan with the score of 0.92 and probability of 0.3127 whereas the least recommended country to have expansion is the United States with the score of 0.40 and the probability of 0.1303. The ranking of the countries is displayed in the data frame format along with their respective average PWS, total journey for deliveries, score as well as probability.

7. Appendix

Brainstorming for question 1:
https://drive.google.com/file/d/1l60vOvKlTJh0AMXbQb3Ve5_6t4jCW9UF/view?usp=sharing

Brainstorming for question 2:
https://drive.google.com/file/d/1L2QSusPoEAERgqOn05Jd8s_ru-QH-faY/view?usp=sharing

Brainstorming for question 3:
https://drive.google.com/file/d/1TkJS9EW7p1EWwr76DsCA2rD0N1zRHuhB/view?usp=sharing

Code for question 1:
https://colab.research.google.com/drive/15qFC6NvemQBXnhTF9Olt0GiS0nI5MRBL?usp=sharing
Code for question 2:
https://colab.research.google.com/drive/19oE_c4rYZRjCqIi9C3202pGzD97okiqE?usp=sharing

Code for question 3:
https://colab.research.google.com/drive/1cdlZKVeY0YwKSC4nIDB5Pzihhcw-AaLR?usp=sharing

8. References

Question 1:
https://bit.ly/3NexlDz
https://bit.ly/3wzao8q
https://on.cfr.org/3wvlziu
https://bit.ly/3MD3gxw
https://bit.ly/3NsEJLE
https://tgam.ca/3FBXg5n
https://bit.ly/3wuToQF
https://bit.ly/3PBHNGS
https://bit.ly/3wIjLTz
https://bit.ly/3NiCElE
https://bit.ly/3yuB6Az
https://bit.ly/3NoEYXW
https://bit.ly/3N8xOH5
https://bit.ly/3sTtt2T
https://bit.ly/3lxy3jj
https://bit.ly/39OWUMR
https://bit.ly/3wlYc9U
https://brook.gs/3G7erMz
https://bit.ly/3PsNHu9
https://bit.ly/39FCS7w
https://bit.ly/39OJLmZ
https://bit.ly/3LBgRUL
https://herit.ag/3wFpY1c
https://reut.rs/3wK5QLv
https://bit.ly/39GUgIU
https://bit.ly/3wATftQ
https://bit.ly/3sP89vj
https://bit.ly/3sP8NZY
https://bit.ly/3wBkWUA
https://bit.ly/3MBnWG0

Question 2:
http://www.math.nagoya-u.ac.jp/~richard/teaching/s2020/Quang1.pdf
https://wuecampus2.uni-wuerzburg.de/moodle/pluginfile.php/1928459/mod_resource/content/2/advalg-ws19-vl01-intro%2Btsp.pdf
https://medium.com/basecs/speeding-up-the-traveling-salesman-using-dynamic-programming-b76d7552e8dd
https://www.youtube.com/watch?v=ihCeYJX7ji8

Question 3:

https://www.youtube.com/watch?v=-qOVVRIZzao&t=2s
https://www.geeksforgeeks.org/quick-sort/
https://towardsdatascience.com/ranking-algorithms-know-your-multi-criteria-decision-solving-techniques-20949198f23e
https://www.geeksforgeeks.org/weighted-sum-method-multi-criteria-decision-making/
https://www.codecademy.com/article/normalization
https://towardsdatascience.com/everything-you-need-to-know-about-min-max-normalization-in-python-b79592732b79