Optimizing of Employee Shuttle Stops: Report

written by Hwanpyo Kim, 09/11/2019

Overview

Let's assume I am a data scientist in a tech company in San Francisco, CA and working to figure out the optimal bus stops for bus-commuters. With thounds of employee-addresses and hundres of potential bus-stops, it is hard to solve this optimization problem with traditional approaches. With K-means clustering, one of unsupervised learning methods to cluster samples, I suggests a new methodology to select the 10 most efficient bus stops.

Keywords - Data exploration, Geocoding, Google Maps Platform, K-means clustering, Euclidean distance

In [1]:

```
import pandas as pd
import re
import googlemaps
from gmplot import gmplot
import matplotlib.pyplot as plt
import numpy as np

# suppress Pandas warning: "SettingWithCopyWarning"
pd.set_option('mode.chained_assignment', None)
```

Data exploration

To start my data-oriented journey, I explore the data with Pandas providing high-level data structures and functions designed to make working with structured data fast and easy. There are two files given for potential bus stops and addresses of employees.

In [2]:

```
employee_addresses = pd.read_csv('./Employee_Addresses.csv')
employee_addresses.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2191 entries, 0 to 2190
Data columns (total 2 columns):
address         2191 non-null object
employee_id     2191 non-null int64
dtypes: int64(1), object(1)
memory usage: 34.3+ KB
```

In [3]:

```python
bus_stops = pd.read_csv('./Bus_Stops.csv')
bus_stops.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119 entries, 0 to 118
Data columns (total 2 columns):
Street_One    119 non-null object
Street_Two    119 non-null object
dtypes: object(2)
memory usage: 1.9+ KB
```

Total number of addresses and bus stops are 2191 and 119, and there isn't any missing value. Considering 10-combinations of bus stops, the total number of combinations is $1.063958304E+14$. IT'S HUGE!!! Even if we could handle the combinatations fortuneatley, we need to assign different bus stops to 2000s employees. It is a extremely intensive job!!

In [4]:

```python
employee_addresses.head()
```

Out[4]:

	address	employee_id
0	98 Edinburgh St, San Francisco, CA 94112, USA	206
1	237 Accacia St, Daly City, CA 94014, USA	2081
2	1835 Folsom St, San Francisco, CA 94103, USA	178
3	170 Cambridge St, San Francisco, CA 94134, USA	50
4	16 Roanoke St, San Francisco, CA 94131, USA	1863

In [5]:

```python
bus_stops.head()
```

Out[5]:

	Street_One	Street_Two
0	MISSION ST	ITALY AVE
1	MISSION ST	NEW MONTGOMERY ST
2	MISSION ST	01ST ST
3	MISSION ST	20TH ST
4	MISSION ST	FREMONT ST

To clean given data, Let's start with employee addresses. I build my own filter to figure out availability of the given addresses. This filter checks wheater the address contains appropriate "House Number", "Street Name", and "City", "State", and "Country".

In [6]:

```python
def address_filter(raw_address):
    street = raw_address.split(',', 1)[0]
    non_street = raw_address.split(',', 1)[1]

    # House Number
    if not re.match("[0-9]", street):
        print("ERROR: House Number: " + raw_address)
        return False

    # Street Name
    if not re.search("[a-z]+", street):
        print("ERROR: Street Name: " + raw_address)
        return False

    # City - San Francisco or Daly City
    if (not re.search("San Francisco", non_street)) and (not re.search(
        print("ERROR: City Name: " + raw_address)
        return False

    # State - CA
    if (not re.search("CA", non_street)):
        print(non_street)
        print("ERROR: State Name: " + raw_address)
        return False

    # Country - USA
    if (not re.search("USA", non_street)):
        print(non_street)
        print("ERROR: Country Name: " + raw_address)
        return False

    return True
```

```
In [7]:  1  employee_addresses_filtered = employee_addresses[(employee_addresses["a
```

```
ERROR: Street Name: 80, San Francisco, CA 94105, USA
ERROR: House Number: St. Luke's Hospital Garage, San Francisco, CA 94110,
USA
ERROR: House Number: St. Luke's Hospital Garage, San Francisco, CA 94110,
USA
ERROR: House Number: San Francisco War Memorial and Performing Arts Cente
r, 301 Van Ness Ave, San Francisco, CA 94102, USA
ERROR: House Number: Essex St, San Francisco, CA 94105, USA
ERROR: House Number: Market Square, 1355 Market St, San Francisco, CA 941
03, USA
ERROR: Street Name: 80, San Francisco, CA 94105, USA
ERROR: House Number: Market Square, 1355 Market St, San Francisco, CA 941
03, USA
ERROR: House Number: Alemany Blvd, San Francisco, CA 94112, USA
ERROR: House Number: Twin Peaks Blvd, San Francisco, CA 94114, USA
ERROR: House Number: St Mary's Park Footbridge, San Francisco, CA, USA
ERROR: House Number: Trainor St, San Francisco, CA 94103, USA
ERROR: House Number: San Jose Avenue, San Francisco, CA 94131, USA
ERROR: Street Name: 101, San Francisco, CA 94124, USA
ERROR: Street Name: 80, San Francisco, CA 94105, USA
ERROR: House Number: Market Square, 1355 Market St, San Francisco, CA 941
03, USA
ERROR: House Number: San Francisco War Memorial and Performing Arts Cente
r, 301 Van Ness Ave, San Francisco, CA 94102, USA
ERROR: House Number: Treat St, San Francisco, CA 94103, USA
ERROR: Street Name: 101, San Francisco, CA 94103, USA
ERROR: Street Name: 101, San Francisco, CA 94103, USA
ERROR: House Number: Twin Peaks Blvd, San Francisco, CA 94114, USA
ERROR: House Number:  B Mission St, San Francisco, CA 94112, USA
ERROR: House Number: Polk St, San Francisco, CA 94102, USA
ERROR: House Number: Plum St, San Francisco, CA 94103, USA
ERROR: House Number: Earl Warren Building, 350 McAllister St, San Francis
co, CA 94102, USA
ERROR: House Number: Earl Warren Building, 350 McAllister St, San Francis
co, CA 94102, USA
```

There are mistaken addresses with wrong "Street Name" or "House Name", and I drop them out.

```
In [8]:  1  employee_addresses_filtered.head()
```

Out[8]:

	address	employee_id
0	98 Edinburgh St, San Francisco, CA 94112, USA	206
1	237 Accacia St, Daly City, CA 94014, USA	2081
2	1835 Folsom St, San Francisco, CA 94103, USA	178
3	170 Cambridge St, San Francisco, CA 94134, USA	50
4	16 Roanoke St, San Francisco, CA 94131, USA	1863

Similar to the mentioned approach, I filter the data of bus stops. However, the format of bus stop data is different from the one of emplyee addresses; it has two columns, "Street_One" and "Street_Two". Right! The potential bus stops are given as intersection. So, we need to check whether these street name are correct or not.

In [9]:
```
def busstop_filter(street_two):

    # Street Number
    if re.match("0[0-9]", street_two):
        print("ERROR: Street two: " + street_two)
        return True

    return False
```

Specially, there are error in "Street_Two" columns. All of single-digit ordinal numbers in "Street_Two" start with zero, and it will cause the significant problem when we calculate their coordinates with `geocode`. So, I strip out the zero and keep the last strings.

In [10]:
```
bus_stops[bus_stops['Street_Two'].apply(busstop_filter)]
```

```
ERROR: Street two: 01ST ST
ERROR: Street two: 02ND ST
ERROR: Street two: 06TH ST
ERROR: Street two: 08TH ST
ERROR: Street two: 04TH ST
ERROR: Street two: 03RD ST
ERROR: Street two: 07TH ST
ERROR: Street two: 09TH ST
ERROR: Street two: 05TH ST
```

Out[10]:

	Street_One	Street_Two
2	MISSION ST	01ST ST
27	MISSION ST	02ND ST
32	MISSION ST	06TH ST
41	MISSION ST	08TH ST
52	MISSION ST	04TH ST
79	MISSION ST	03RD ST
89	MISSION ST	07TH ST
112	MISSION ST	09TH ST
116	MISSION ST	05TH ST

In [11]:
```
bus_stops_filtered = bus_stops.copy()
```

In [12]:
```
bus_stops_filtered['Street_Two'] = bus_stops_filtered['Street_Two'].app
```

Moreoever, There are another typo in "Street_Two", "ANGELOS ALY". I converted it to the original name, "ANGELO'S ALY".

In [13]:
```
bus_stops_filtered['Street_Two'] =  bus_stops_filtered['Street_Two'].ap
```

After filtering, I create a new column, "address", to find the intersection between two roads as following

In [14]:
```
bus_stops_filtered['address'] = bus_stops_filtered['Street_One'] + " &
```

Google Maps - Geocoding API

Geocoding is the process of converting address into geographic coordinates, latitude & longitude, and we can perform this process with Geocoding API in Google Maps Platform. To get started with Google Maps services, I install Python Clent for Google Maps services (https://github.com/googlemaps/google-maps-services-python). we need to manage API key in Google Cloud Platform console and the details are explained in the following website (https://developers.google.com/maps/gmp-get-started). With my API key (erased for security), we can convert employee addresses and bus stop intersections into geocodes.

In [15]:
```
# Please update API Key
APIKey = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
gmaps = googlemaps.Client(key=APIKey)
```

In [16]:
```
employee_addresses_filtered['geocode'] = employee_addresses_filtered['a
```

In [17]:
```
bus_stops_filtered['geocode'] = bus_stops_filtered['address'].apply(gma
```

I check if there is any null return in the geocode.

In [18]:
```
employee_addresses_filtered[employee_addresses_filtered['geocode'].str.
```

Out[18]:

address	employee_id	geocode

In [19]:
```
bus_stops_filtered[bus_stops_filtered['geocode'].str.len() == 0]
```

Out[19]:

Street_One	Street_Two	address	geocode

And, I extract coordinates, latitude and longitude from given geocodes.

In [20]:
```
def coordinates_func(geocode):
    try:
        return (geocode[0]['geometry']['location']['lat'], geocode[0]['
    except AttributeError or IndexError:
        return None
```

In [21]:
```
employee_addresses_filtered['coordinates'] = employee_addresses_filtere
```

In [22]:
```python
bus_stops_filtered['coordinates'] = bus_stops_filtered['geocode'].apply
```

In [23]:
```python
employee_addresses_filtered['lat'], employee_addresses_filtered['lng']
```

In [24]:
```python
ops_filtered['lat'], bus_stops_filtered['lng'] = bus_stops_filtered['coordi
```

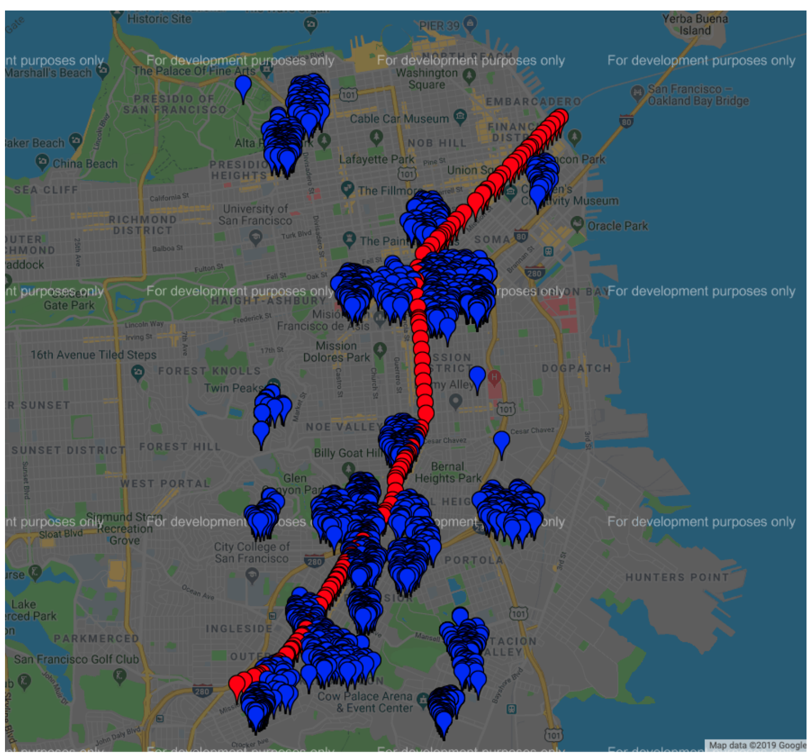
With [gmplot (https://pypi.org/project/gmplot/)](https://pypi.org/project/gmplot/), a matploblib-like interface to generate HTML and javascript to render the data on top of Google maps, I plot all of the employee addresses and proposed bus stops on Google Maps at San Francisco, CA.

In [25]:
```python
# Place map
gmap = gmplot.GoogleMapPlotter(37.766956, -122.438481, 13)

# Marker - Employee Addresses
employee_addresses_filtered.apply(
    lambda x: gmap.marker(x['lat'], x['lng'], title = x['address'], c='
# Marker - Bus Stops
bus_stops_filtered.apply(
    lambda x: gmap.marker(x['lat'], x['lng'], title = x['address'], c='

# Draw
gmap.draw("employee_adresses_bus_stops_filtered.html")
```

Here is the image of the previous draw; the blue markers are the addresses of employee and the red ones are the potential bus stops.



Now, we went through our given data and visualize the on Google maps. To go to the next step, we need **assumptions** to simply this problem as following:

1. **Longitude and Latitude as Cartesian coordinates**: The project is focused on San Francisco, CA and its neighborhoods. So, we can assume the given coordinates as Cartesian coordiantes on flat surface. Therefore, we are able to calculate between two points on the map with latitude and longitude simply.
2. **Walking distance as Euclidean distance (L_2 distance)**: I assume that every employee use Euclidean routing to go bus stops. In this project, it is hard to calculate every walking routes and their distance based on the geographic maps. Estimating Euclidean distance between an employee address and a bus stop is simple and useful metrics .

K-means clustering for grouping employee-addresses

As mentioned, it is difficult to consider all of 10 combinations of bus stops and pairs between bus stops and employee addresses. So, I suggest my solution based on K-means clustering. K-means clustering is a powerful method for partitioning a data set into K distinct clusters. It minimizes the normalized sum of the pairwise squared Euclidean distances between all data in the kth cluster. After clustering, we can compute centorids of each cluster. As my first tryout, I build K-means cluster with 10 clusters for 10 bus stops.

In [26]:
```
from sklearn.cluster import KMeans
```

In [27]:
```
estimator = KMeans(n_clusters=10)
estimator.fit(employee_addresses_filtered[['lat', 'lng']])
```

Out[27]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=10, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [28]:
```
estimator.labels_
```

Out[28]:
```
array([3, 4, 1, ..., 3, 0, 1], dtype=int32)
```

Also, here are their own centroids

In [29]:
```
estimator.cluster_centers_
```

Out[29]:
```
array([[  37.71197141, -122.44176278],
       [  37.76976721, -122.41321875],
       [  37.79250522, -122.44355565],
       [  37.72659155, -122.42651066],
       [  37.70982582, -122.41263268],
       [  37.7381825 , -122.42834067],
       [  37.7351947 , -122.40262495],
       [  37.74349028, -122.44881285],
       [  37.76798878, -122.43002465],
       [  37.78492408, -122.39595134]])
```

In [30]:

```
centers = pd.DataFrame(estimator.cluster_centers_, columns=['lat', 'lng
centers
```

Out[30]:

	lat	lng
0	37.711971	-122.441763
1	37.769767	-122.413219
2	37.792505	-122.443556
3	37.726592	-122.426511
4	37.709826	-122.412633
5	37.738183	-122.428341
6	37.735195	-122.402625
7	37.743490	-122.448813
8	37.767989	-122.430025
9	37.784924	-122.395951

In [31]:

```
employee_addresses_filtered['KMeans_10'] = estimator.labels_
```

In [32]:

```
colorlist = ['#800080', '#FF00FF', '#000080', '#0000FF', '#008080',
             '#00FFFF', '#008000', '#00FF00', '#808000', '#FFFF00',
             '#800000', '#FF0000', '#808080', '#C0C0C0', '#FFFFFF',
             '#483D8B', '#A52A2A', '#B8860B', '#BDB76B', '#EE82EE'] * 3
```

In [33]:

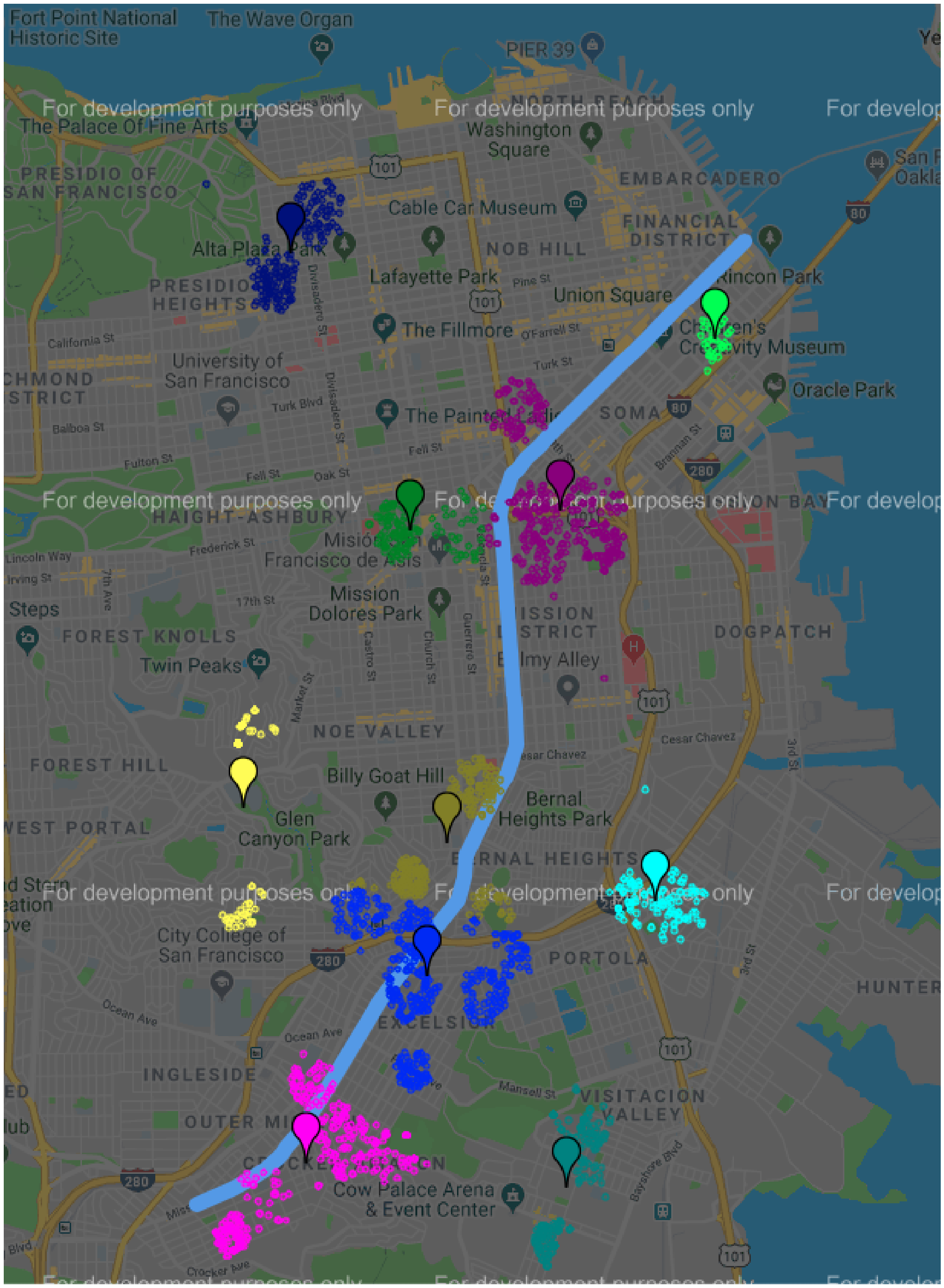
```
# Place map
gmap = gmplot.GoogleMapPlotter(37.766956, -122.438481, 13)

# Routes of bus
bus_stops_filtered_lats, bus_stops_filtered_lngs = zip(*bus_stops_filte
gmap.plot(bus_stops_filtered_lats, bus_stops_filtered_lngs, 'cornflower

for i in employee_addresses_filtered['KMeans_10'].unique():
    top_attraction_lats, top_attraction_lons = zip(*list(
        employee_addresses_filtered[employee_addresses_filtered['KMeans
    gmap.scatter(top_attraction_lats, top_attraction_lons, colorlist[i]
    # centroid of clusters
    gmap.marker(centers['lat'][i], centers['lng'][i], c=colorlist[i])


# Draw
gmap.draw("employee_adresses_filtered_KMeans_10.html")
```

Again, we can plot clustered addresses with their own clusters on Google maps. However, the clustering result is only based on their Euclidean disntaces between other points. Not based on the distances bus stops.


Fort Point National Historic Site
The Wave Organ
PIER 39
The Palace Of Fine Arts
PRESIDIO OF SAN FRANCISCO
Washington Square
EMBARCADERO
Cable Car Museum
FINANCIAL DISTRICT
NOB HILL
Alta Plaza Park
Lafayette Park
PRESIDIO HEIGHTS
Union Square
Rincon Park
The Fillmore
University of San Francisco
Oracle Park
SOMA
HAIGHT-ASHBURY
Mission Dolores Park
DOGPATCH
FOREST KNOLLS
Twin Peaks
NOE VALLEY
FOREST HILL
Billy Goat Hill
Bernal Heights Park
Glen Canyon Park
City College of San Francisco
PORTOLA
EXCELSIOR
INGLESIDE
VISITACION VALLEY
Cow Palace Arena & Event Center
For development purposes only

In [34]:
```
centers = pd.DataFrame(estimator.cluster_centers_, columns=['lat', 'lng
centers
```

Out[34]:

	lat	lng
0	37.711971	-122.441763
1	37.769767	-122.413219
2	37.792505	-122.443556
3	37.726592	-122.426511
4	37.709826	-122.412633
5	37.738183	-122.428341
6	37.735195	-122.402625
7	37.743490	-122.448813
8	37.767989	-122.430025
9	37.784924	-122.395951

In [35]:
```
def closest_node(node, nodes):
    # Euclidean distance
    dist_manhattan = np.sum((nodes - node)**2, axis=1)
    return np.argmin(dist_manhattan)

def distance(node1, node2):
    # Euclidean distance
    return  np.sum((node1 - node2)**2, axis=1)

# def closest_node(node, nodes):
#     # Manhattan distance
#     dist_manhattan = np.sum(np.abs(nodes - node), axis=1)
#     return np.argmin(dist_manhattan)

# def distance(node1, node2):
#     # Manhattan distance
#     return  np.sum(np.abs(node1 - node2), axis=1)
```

To find closest bus stop for each cluster, I assume that the closest bus stop to cluster is the closest one to its centroid with Euclidean distance. Although it is not an accurate approch to calculate the total walking distance, it reduces the computational cost and time.

In [36]:
```
bus_stops_chosen = bus_stops_filtered.iloc[
    centers.apply(lambda x: closest_node(x, bus_stops_filtered[['lat',
```

In [37]:

```
# Place map
gmap = gmplot.GoogleMapPlotter(37.766956, -122.438481, 13)

# Routes of bus
bus_stops_filtered_lats, bus_stops_filtered_lngs = zip(*bus_stops_filte
gmap.plot(bus_stops_filtered_lats, bus_stops_filtered_lngs, 'cornflower

for i in employee_addresses_filtered['KMeans_10'].unique():
    top_attraction_lats, top_attraction_lons = zip(*list(
        employee_addresses_filtered[employee_addresses_filtered['KMeans
    gmap.scatter(top_attraction_lats, top_attraction_lons, colorlist[i]
    # centroid of employee_addresses
#     gmap.marker(centers['lat'][i], centers['lng'][i], c=colorlist[i])
    # nearest bus stop
    gmap.marker(bus_stops_chosen.iloc[i]['lat'], bus_stops_chosen.iloc[

# Draw
gmap.draw("employee_adresses_filtered_KMeans_10_best.html")
```

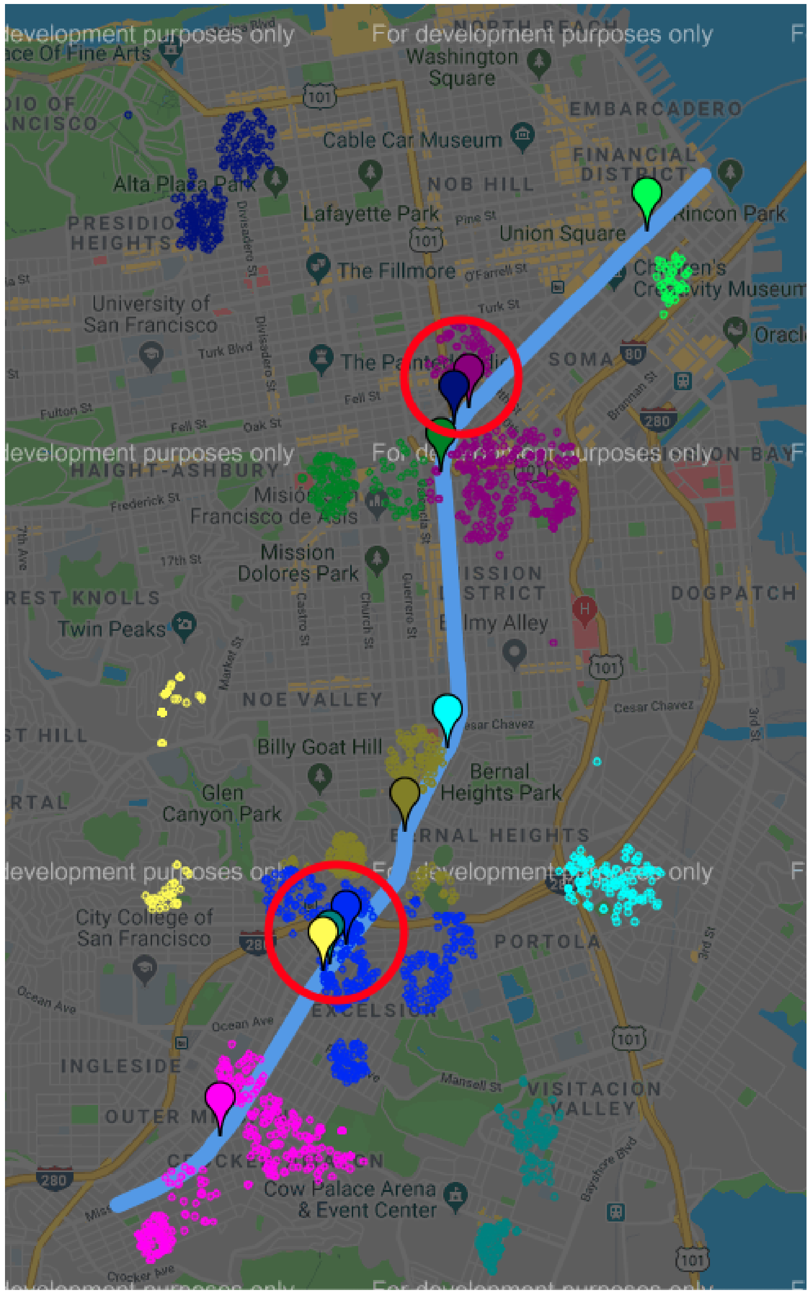
In [38]:

```
bus_stops_chosen
```

Out[38]:

	Street_One	Street_Two	address	geocode	coordinates	lat	l
67	MISSION ST	CONCORD ST	MISSION ST & CONCORD ST San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.7139529, -122.4433752)	37.713953	-122.4433
50	MISSION ST	11TH ST	MISSION ST & 11TH ST San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.7743325, -122.4171372)	37.774332	-122.4171
84	MISSION ST	12TH ST	MISSION ST & 12TH ST San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.7730672, -122.4187201)	37.773067	-122.4187
62	MISSION ST	ADMIRAL AVE	MISSION ST & ADMIRAL AVE San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.729844, -122.4301774)	37.729844	-122.4301
20	MISSION ST	TINGLEY ST	MISSION ST & TINGLEY ST San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.7282519, -122.431806)	37.728252	-122.4318
68	MISSION ST	HIGHLAND AVE	MISSION ST & HIGHLAND AVE San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.7373612, -122.4240484)	37.737361	-122.4240
102	MISSION ST	POWERS AVE	MISSION ST & POWERS AVE San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.7461855, -122.4195241)	37.746186	-122.4195
74	MISSION ST	AVALON AVE	MISSION ST & AVALON AVE San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.727656, -122.4324727)	37.727656	-122.4324
6	MISSION ST	ERIE ST	MISSION ST & ERIE ST San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.7690631, -122.4200723)	37.769063	-122.4200
73	MISSION ST	SHAW ALY	MISSION ST & SHAW ALY San Francisco	[{'address_components': [{'long_name': 'Missio...	(37.7889865, -122.3985861)	37.788987	-122.3985

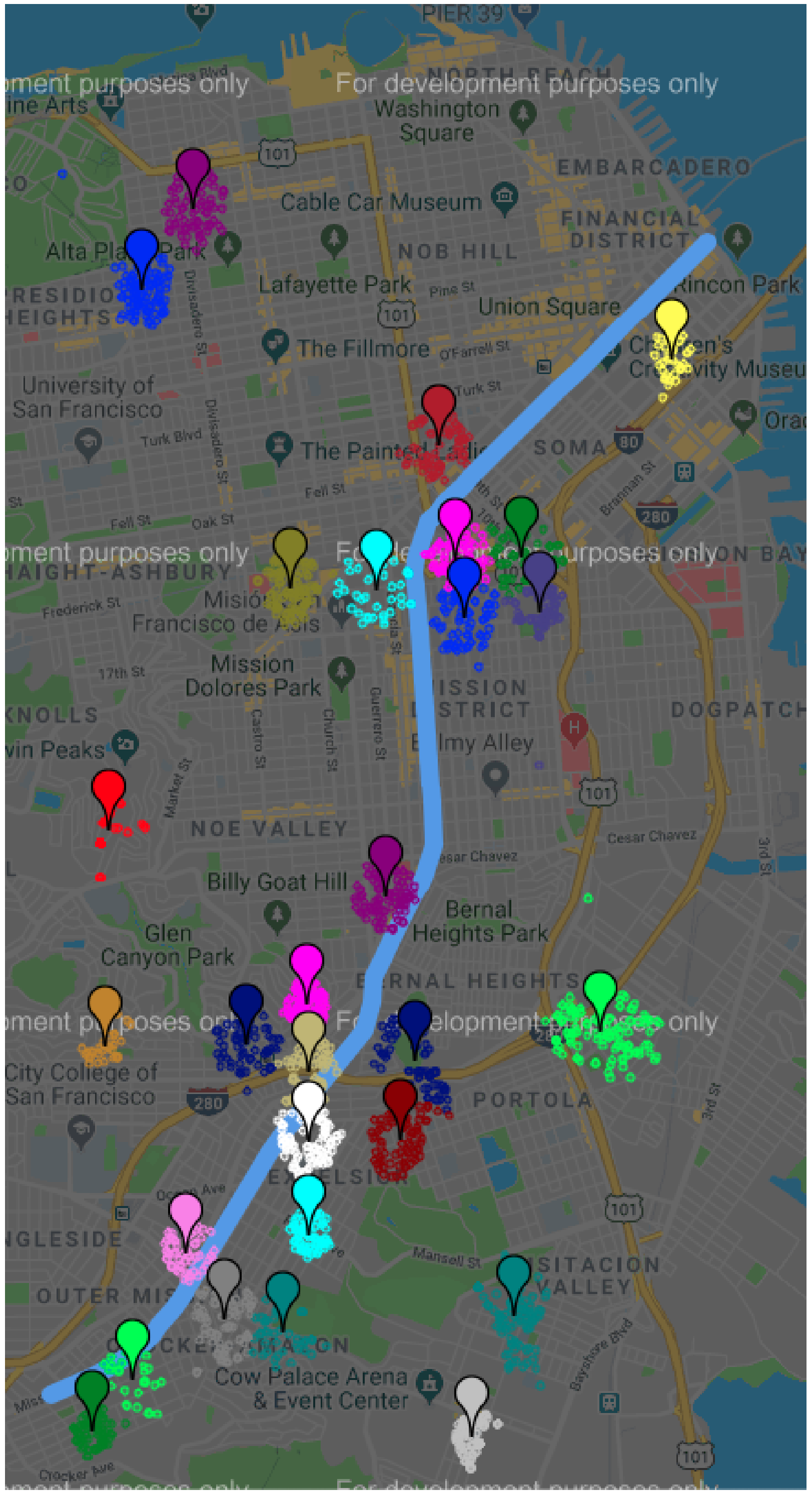
Here is the plot of chosen 10 bus stops with K-means clustering with K=10, drawn as different markers. We can see that some of selected bus stops in the red circles are too close each other. It means these are not efficient to reduce walking distance actually.



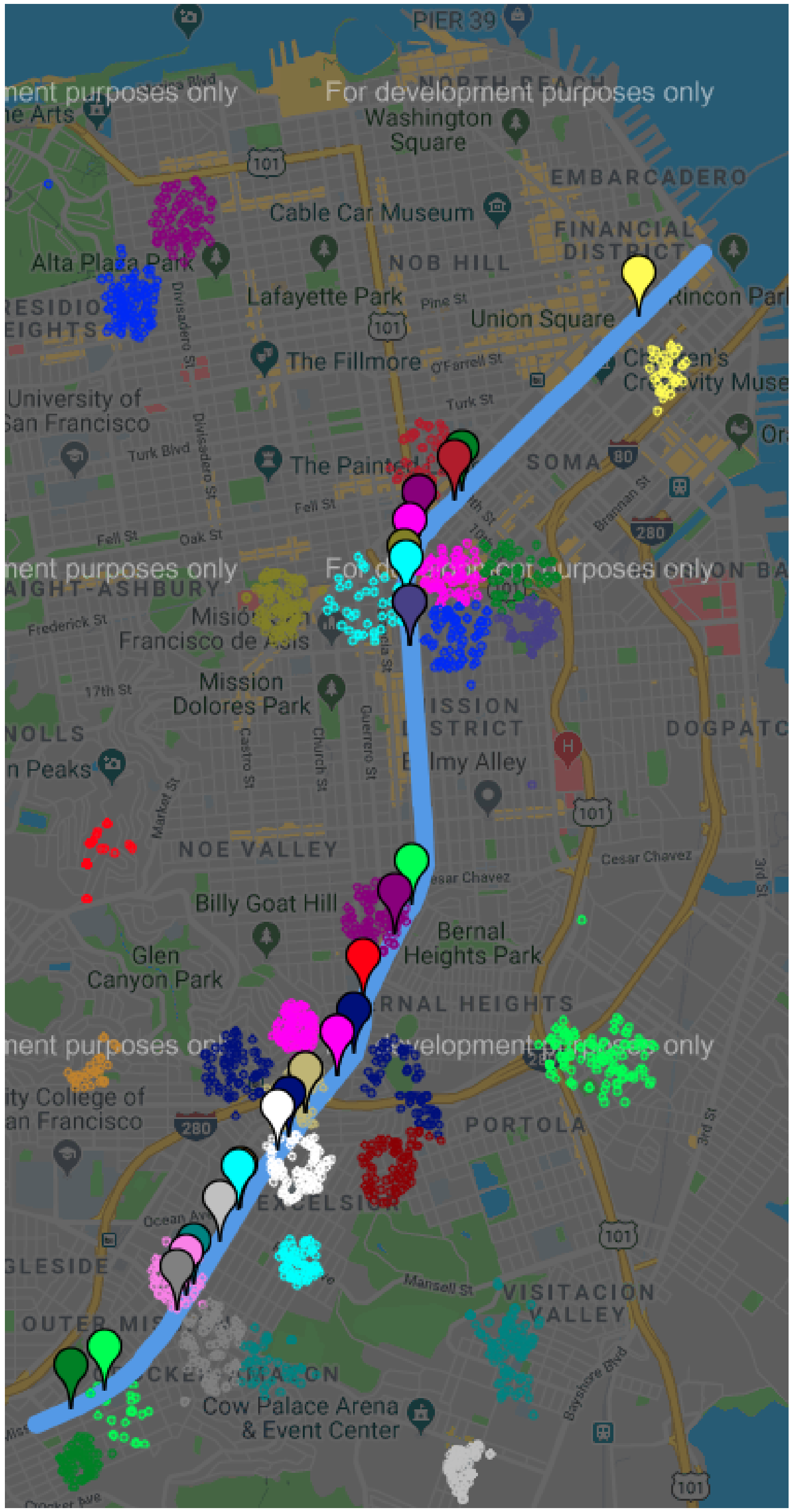
Clustering more than 10 selected bus stops !!

The reason with wrong result in the last approach is coming from K-means clustering. K-means cluster bundles the given points based on Euclidean distance, but not the distance to bus stops. So, I suggested a upgraded method to find optimal bus stops with double layered K-means clusterings. Here is the algorithm:

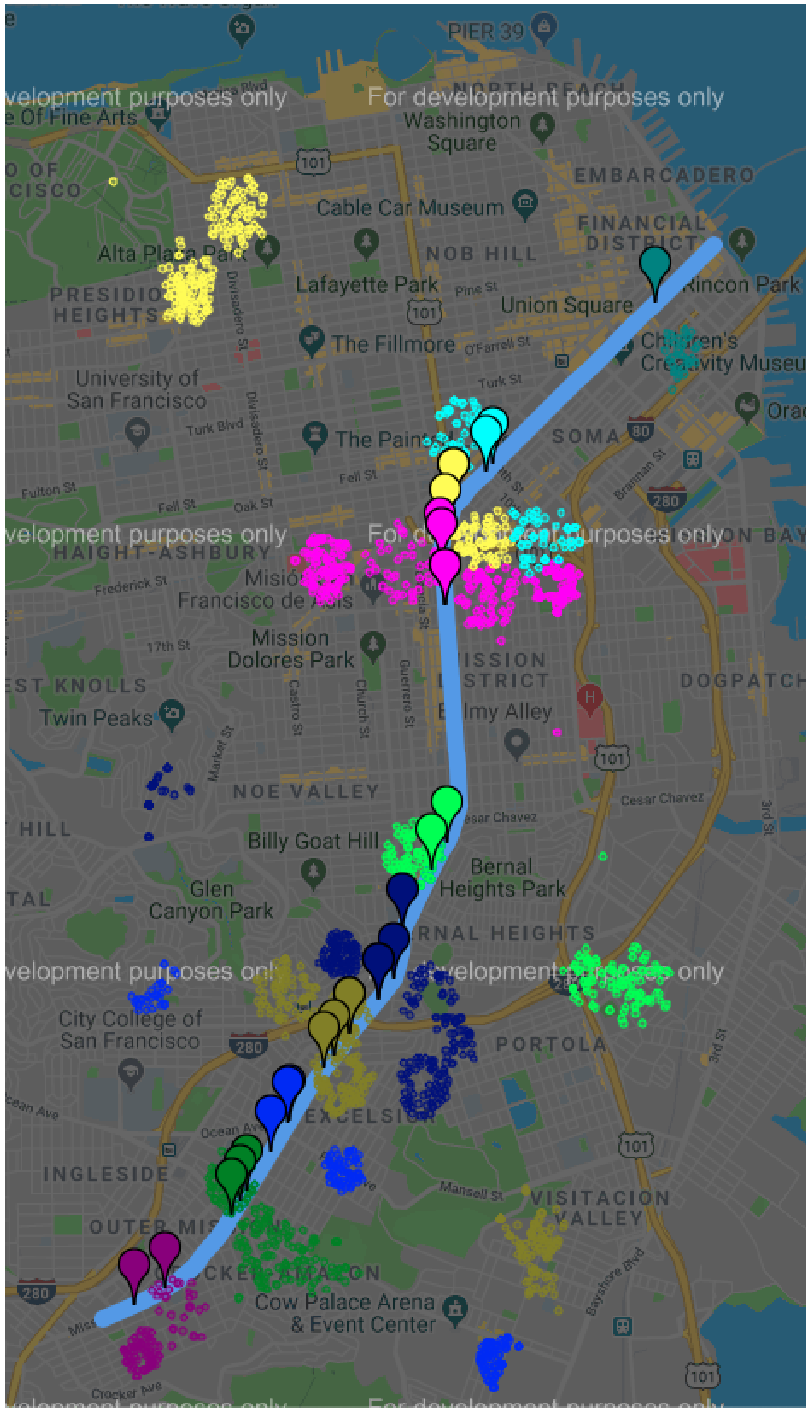
First, cluster employee addresses more than 10 clusters. We need more clusters than 10 to get more freedom to choose bus stops. Here is the first plot with $K = 28$, which has best result in total distance.



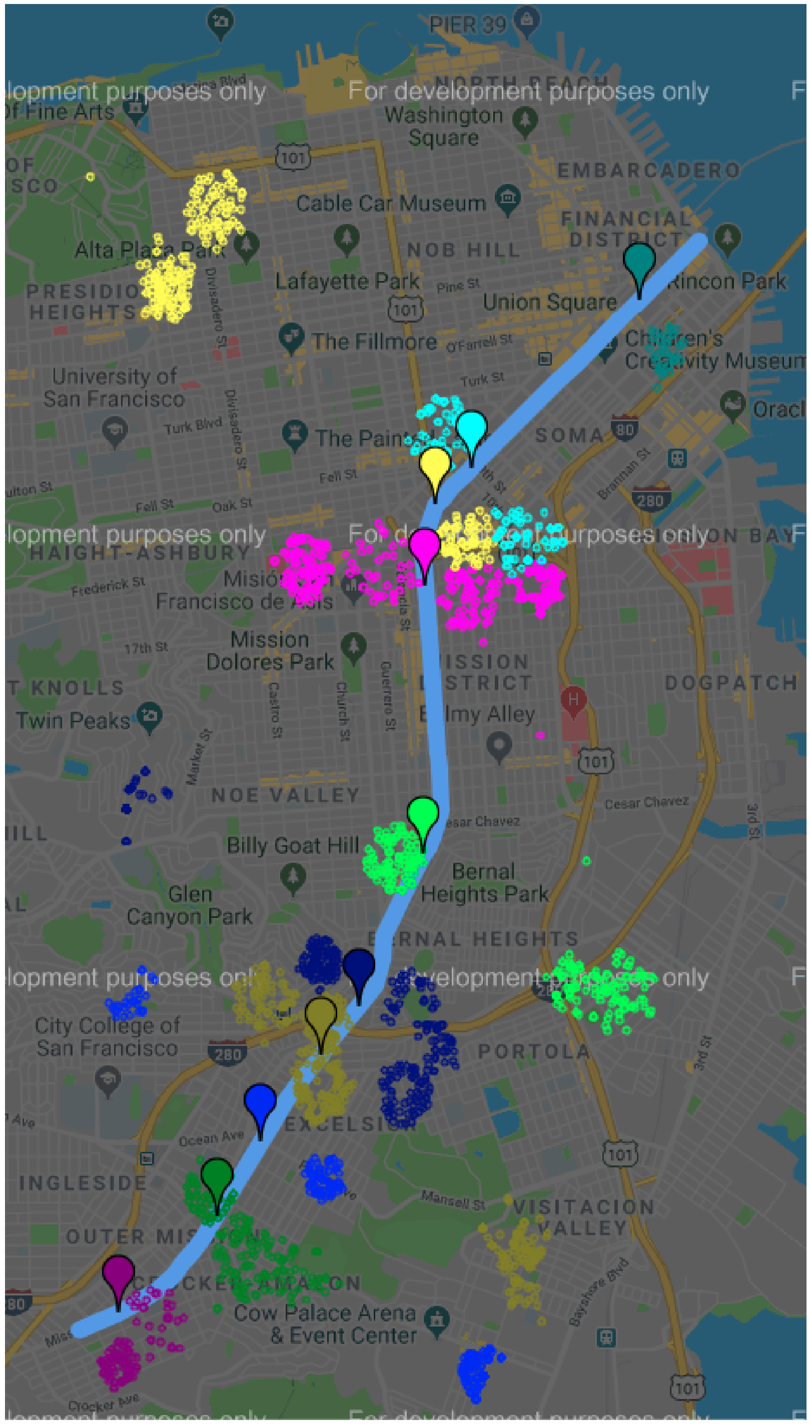
Second, find closest bus stops for each clusters. Up to this step, identical with the previous one.



Third, group the bus stops!! Now, I have more than 10 bus stops and we are not allowed to build more than 10 bus stops. So, we can build another K-means cluster with K=10 for clustering bus stops and their employee groups.



Finally, find new closest bus stops for optimal 10 employee groups. With the updated 10 groups of employee. We can find their new closest bus stops with new centroids of the fresh clusters!! Now, we have the nearest 10 bus stops!!! Yay!!!



We can see that the final clusters are nearly perpendicular to the bus routes at their own bus stop respectivley. Its means every employees can take reach to their bus stops with shortest walking dinstance. :)))

In [39]:

```
def find_centroid(nodes):
    return np.mean(nodes)
```

In [40]:

```
def Kmeans_func(employee_addresses_filtered, bus_stops_filtered_lats, r

    #### Step 1: cluster employee addresses
    label_name = 'KMeans_' + str(n_clusters)

    estimator = KMeans(n_clusters=n_clusters, random_state=0)
    estimator.fit(employee_addresses_filtered[['lat', 'lng']])

    centroids = pd.DataFrame(estimator.cluster_centers_, columns=['lat

    employee_addresses_filtered[label_name] = estimator.labels_

    # Place map
    gmap = gmplot.GoogleMapPlotter(37.766956, -122.438481, 13)

    # Routes of bus
    bus_stops_filtered_lats, bus_stops_filtered_lngs = zip(*bus_stops_
    gmap.plot(bus_stops_filtered_lats, bus_stops_filtered_lngs, 'cornf

    for i in range(len(centroids)):
        top_attraction_lats, top_attraction_lons = zip(*list(
            employee_addresses_filtered[employee_addresses_filtered[la
        gmap.scatter(top_attraction_lats, top_attraction_lons, colorli
        # centroid of clusters
        gmap.marker(centroids['lat'][i], centroids['lng'][i], c=colorl

    # Draw
    gmap.draw("employee_adresses_filtered_" + label_name + ".html")




    #### Step 2: find closest bus stops for each cluster
    bus_stops_chosen = bus_stops_filtered.iloc[centroids.apply(lambda 
    bus_stops_chosen[label_name]= range(len(bus_stops_chosen))
    # Place map
    gmap = gmplot.GoogleMapPlotter(37.766956, -122.438481, 13)

    # Routes of bus
    bus_stops_filtered_lats, bus_stops_filtered_lngs = zip(*bus_stops_
    gmap.plot(bus_stops_filtered_lats, bus_stops_filtered_lngs, 'cornf

    for i in employee_addresses_filtered[label_name].unique():
        top_attraction_lats, top_attraction_lons = zip(*list(
            employee_addresses_filtered[employee_addresses_filtered[la
        gmap.scatter(top_attraction_lats, top_attraction_lons, colorli
        # nearest bus stop
        gmap.marker(bus_stops_chosen.iloc[i]['lat'], bus_stops_chosen.

    # Draw
    gmap.draw("employee_adresses_filtered_" + label_name + "_busstops.



    #### Step 3: group bus stops and employee clusters into 10 groups
    estimator = KMeans(n_clusters=10, random_state=0)
```

```
    estimator.fit(bus_stops_chosen[['lat', 'lng']])

    bus_stops_chosen['clustered'] = estimator.labels_

    employee_addresses_filtered_clustered = pd.merge(
        employee_addresses_filtered, bus_stops_chosen[[label_name, 'clu


    # Place map
    gmap = gmplot.GoogleMapPlotter(37.766956, -122.438481, 13)

    # Routes of bus
    bus_stops_filtered_lats, bus_stops_filtered_lngs = zip(*bus_stops_
    gmap.plot(bus_stops_filtered_lats, bus_stops_filtered_lngs, 'cornf

    for i in employee_addresses_filtered_clustered['clustered'].unique
        top_attraction_lats, top_attraction_lons = zip(*list(
            employee_addresses_filtered_clustered[employee_addresses_f
        gmap.scatter(top_attraction_lats, top_attraction_lons, colorlis

    for i in range(len(bus_stops_chosen)):
        # nearest bus stop
        gmap.marker(bus_stops_chosen.iloc[i]['lat'], bus_stops_chosen.i

    # Draw
    gmap.draw("employee_adresses_filtered_" +  label_name + "_busstops_



    #### Step 4: find new closest bus stops for optimal 10 employee gro
    # New centroids based on last clustering result
    centroids = pd.DataFrame()
    for i in employee_addresses_filtered_clustered['clustered'].unique
        centroid = employee_addresses_filtered_clustered[
            employee_addresses_filtered_clustered['clustered'] == i][[
        centroids = centroids.append(centroid, ignore_index=True)
    centroids = centroids.set_index(employee_addresses_filtered_cluste

    bus_stops_chosen = bus_stops_filtered.loc[centroids.apply(lambda x
    bus_stops_chosen = bus_stops_chosen.set_index(employee_addresses_f
    # Place map
    gmap = gmplot.GoogleMapPlotter(37.766956, -122.438481, 13)

    # Routes of bus
    bus_stops_filtered_lats, bus_stops_filtered_lngs = zip(*bus_stops_
    gmap.plot(bus_stops_filtered_lats, bus_stops_filtered_lngs, 'cornf

    for i in employee_addresses_filtered_clustered['clustered'].unique
        top_attraction_lats, top_attraction_lons = zip(*list(
            employee_addresses_filtered_clustered[employee_addresses_f
        gmap.scatter(top_attraction_lats, top_attraction_lons, colorlis
        # nearest bus stop
        gmap.marker(bus_stops_chosen.loc[i]['lat'], bus_stops_chosen.lo

    # Draw
    gmap.draw("employee_adresses_filtered_" + label_name + "_busstops_
```

```
    # Calculate Total Distance
    employee_temp = pd.merge(employee_addresses_filtered_clustered[['la
                    bus_stops_chosen[['lat', 'lng']].rename(columns={"la
                    left_on='clustered', right_index = True)

    return np.sum(distance(employee_temp[['lat', 'lng']],
                            employee_temp[['lat_bus', 'lng_bus']].renam
```

To optimize hyper-parameter in my algorithm, the number of K of employee addresses clustering in the firt step is considered. I checked the different number of clusters from 10 to 40, and I choose $K = 28$ as my optimal hyper-parameter based on total distances between employee addresses and their assigned bus stops. Its result are drawn in the previous plots.

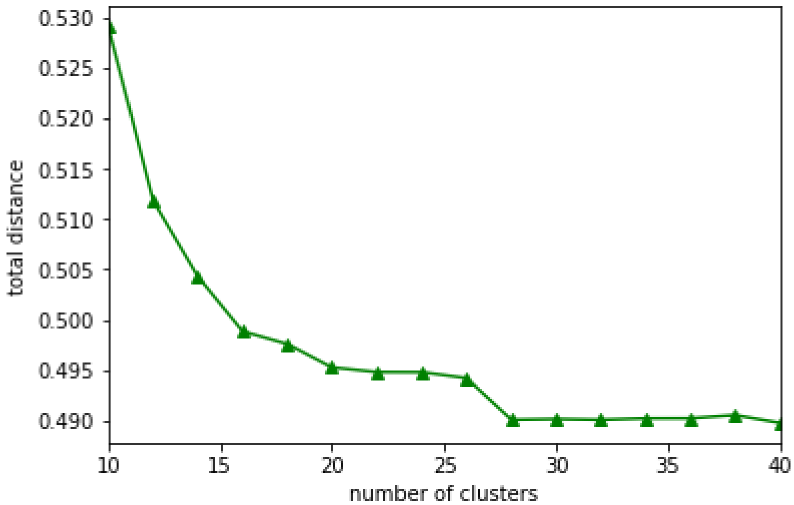
In [41]:

```
total_distances = []
for k in range(10, 41, 2):
    total_distance, _ = Kmeans_func(employee_addresses_filtered, bus_st
    print(str(k) + " clusters - total distance: " + str(total_distance)
    total_distances.append(total_distance)
```

```
10 clusters - total distance: 0.5291219874870465
12 clusters - total distance: 0.5117956550576135
14 clusters - total distance: 0.504266909379683l
16 clusters - total distance: 0.4988321862822756
18 clusters - total distance: 0.49756509934741766
20 clusters - total distance: 0.49523818996920826
22 clusters - total distance: 0.4947738311185496
24 clusters - total distance: 0.4947591860944997
26 clusters - total distance: 0.4941692369576148
28 clusters - total distance: 0.49007245629958873
30 clusters - total distance: 0.49014570657960876
32 clusters - total distance: 0.49007245629958884
34 clusters - total distance: 0.4902104236621086
36 clusters - total distance: 0.49021631797739207
38 clusters - total distance: 0.49054054207316256
40 clusters - total distance: 0.48976495478935694
```

In [42]:

```
plt.plot(range(10, 41, 2), total_distances, 'g^-')
plt.xlim([10,40])
plt.xlabel('number of clusters')
plt.ylabel('total distance')
plt.show()
```



In [46]:

```
total_distance_28, bus_stops_chosen_28 = Kmeans_func(
    employee_addresses_filtered, bus_stops_filtered_lats, 28)
```

YES!! Here are the best 10 bus stops to minimize walking distance and our Data journey ends!!!

In [47]:
```
bus_stops_chosen_34[['Street_One', 'Street_Two']]
```

Out[47]:

	Street_One	Street_Two
7	MISSION ST	ADMIRAL AVE
1	MISSION ST	RUTH ST
8	MISSION ST	15TH ST
5	MISSION ST	BOSWORTH ST
4	MISSION ST	LAURA ST
9	MISSION ST	WASHBURN ST
0	MISSION ST	FAIR AVE
3	MISSION ST	12TH ST
6	MISSION ST	AMAZON AVE
2	MISSION ST	SHAW ALY

Summary

In this mini-project, I explored the data with thounds of employ addresses and hundreds of bus stops and built my own machine learning algorithm to find the best 10 bus stops to minimize walking distances of employees. I utilized two of K-means clusters and its performance is successful. However it is not the end. We simplify this problem with strong assumption on walking distance as Euclidean distance. However, We could consider different metrics including Manhattan distance to figure out different results. Furthermore, if we consider walking routes to bus stops, it would be more realistic.