# Tutorial: Using Google Maps to calculate driving distance between two points in Python

Geospacial analysis has entered the social science mainstream. Spacial economics and political geography are trending thanks to the availability of accessible and user-friendly geospacial analysis software.

As geographic distance is an increasingly popular dependent variable in social science, this tutorial explains how to use the Google Maps API to find driving distance in miles between locations. We simply start with two dataframes containing geocordinates: one with origins, and another with destinations.

This tutorial specifically will teach you to calculate the driving distance between the origin and the *nearest* possible destination — assuming the dataframe containing destinations has more than one observation — which can be used to find the driving distance between survey respondents and their nearest polling place, government officials and 311 calls, or even customers and stores in a commercial chain.

To begin, let's load the required packages: pandas, datetime, geopy.distance and googlemaps.

```
import pandas as pd
from datetime import datetime
from geopy.distance import great_circle
import googlemaps
```

Next, we need to define the Google Maps API. To do this, you first need to obtain an API key, which you can obtain here.

```
# Define the Google Maps API key
gmaps = googlemaps.Client(key='YOUR_KEY', retry_over_query_limit=True)
```

The Google Maps API has usage limits. We can set *retry_over_query_limit* to "True" to avoid rate limit errors when we make our calls to the API.

## Finding the nearest adult-use cannabis dispensary

We can now get to work. For this tutorial, I will be calculating the distance between towns in the Philadelphia suburbs and recreational marijuana dispensaries in New Jersey. New Jersey legalized adult-use cannabis in 2022 Despite U.S. Senator and former-Lieutenant Governor John Fetterman's outspoken support for legalization of recreational marijuana, it has yet to occur in Pennsylvania.

Yet, it has been well-documented that Pennsylvanians in the Philadelphia suburbs have been traveling to New Jersey to purchase legal pot. The *Philadelphia Inquirerer* even published an article about what Pennsylvanians should know before traveling to New Jersey to purchase cannabis. One dispensary in Phillipsburg, New Jersey faces directly across the Delaware River into Easton — it's a modern day Washington's crossing!

We will calculate the driving distance from the centroid of each designated census place in Bucks and Montgomery County to their nearest New Jersey recreational marijuana dispensary!

We can import our dispensary locations from New Jersey's NJOIT Open Data Center, a cleaned version of which I have posted on my GitHub.

```
dat = pd.read_csv('https://raw.githubusercontent.com/BrendanTHartnett/the_road_to_pot/main/New_Jersey_Cannabis_Dispensaries.csv')
# Extract the latitude and longitude values from the DISPENSARY LOCATION column
dat['STORElong'] = dat['DISPENSARY LOCATION'].apply(lambda x: float(x.split(' ')[1][1:]))
dat['STORElat'] = dat['DISPENSARY LOCATION'].apply(lambda x: float(x.split(' ')[2][:-1]))
```

Given that medicinal marijuana has been legal in Pennsylvania since 2016, and our dataframe contains dispensaries in New Jersey which are either medicainal and recreational or just medicinal, we should remove dispensaries which only sell medical marijuana.

```
# create a boolean mask indicating which rows have 'TYPE' equal to "Medicinal cannabis only"
mask = dat['TYPE'] == "Medicinal cannabis only"

# use the mask to drop the relevant rows
dat = dat[~mask]
```

Next, import our "customer" data — a dataframe containing the geocoordinates of each Census designated place in Bucks and Montgomery County.

```
fdat = pd.read_csv('https://raw.githubusercontent.com/BrendanTHartnett/the_road_to_pot/main/bucksmont_subdivisons.csv')
```

Now, with our two dataframes, we need to first match each town with its nearest store. While we could do this using the Google Maps API, the rate limit causes this to be slow and computationally intensive process — and with more observations, these issues will only compound. Instead, we can just find the nearest store by linear distance. There are limitations to doing this, which our region of focus speaks to.

Given all Pennsylvanians must travel via bridge to New Jersey, this algorithm could misidentify the nearest dispensary to each town based on the location of bridges. Ultimately, the trade-off between run time and accuracy forces researchers to make a choice based on what they are willing to sacrifice.

We will define a function to find each town's (in dataframe *fdat*) linear distance to the nearest dispensary (in dataframe *dat*).

```
def nearest_store(row):
    buyer_lat, buyer_long = row['BuyerLat'], row['BuyerLong']
    buyer_loc = (buyer_lat, buyer_long)
    min_dist = float('inf')
    nearest_store = None

    for _, store in dat.iterrows():
        store_lat, store_long = store['STORElat'], store['STORElong']
        store_loc = (store_lat, store_long)
        distance = great_circle(buyer_loc, store_loc).meters

        if distance < min_dist:
            min_dist = distance
            nearest_store = store

    return pd.Series([nearest_store['STORElong'], nearest_store['STORElat']])
```

We can then run the function and append the longitude and latitude of the nearest dispensary to the corresponding Pennsylvania town.

```
# Find the nearest store for each potential customer and add the STORElong and STORELAD to fdat
fdat[['NearestStoreLong', 'NearestStoreLat']] = fdat.apply(nearest_store, axis=1)
```

```
fdat.head()
```

| | GEOID | NAMELSAD | BuyerLat | BuyerLong | NearestStoreLong | NearestStoreLat |
|---|---|---|---|---|---|---|
| 0 | 4201704976 | Bedminster township | 40.426614 | -75.188230 | -75.201923 | 40.690754 |
| 1 | 4201705616 | Bensalem township | 40.106196 | -74.943689 | -74.912595 | 40.040754 |
| 2 | 4201708592 | Bridgeton township | 40.552842 | -75.121723 | -75.201923 | 40.690754 |
| 3 | 4201708760 | Bristol borough | 40.102767 | -74.852347 | -74.912595 | 40.040754 |
| 4 | 4201708768 | Bristol township | 40.123638 | -74.867385 | -74.912595 | 40.040754 |

## Finding driving distances to weed stores

Now we can run a loop that calculates the driving distance from the road nearest the origin (BuyerLat, BuyerLong) and the destination (NearestStoreLat, NearestStoreLat).

One thing to remember is that the coordinates we are working with may not all be accessible via road — particularly the origin, the coordinates of which simply correspond to the centroids of designated Census places. Therefore, we will find the nearest roadway to the listed coordinates before finding the riving distance between the two locations. Then we simply call the Google Maps API and specify that we want the *driving* distance between the two points. After getting the distance in meters, we can convert to miles and append the value to our dataframe.

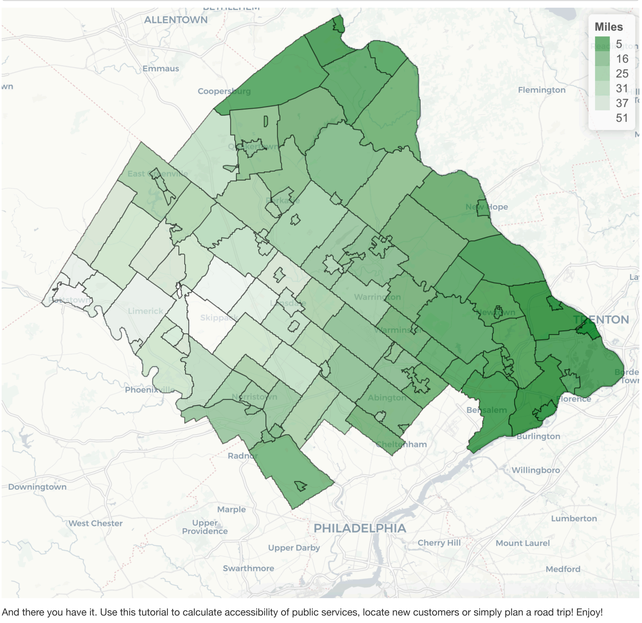
```
# Loop through each row in the fdat dataframe and compute the driving distance
for i in range(0, len(fdat)):
    # Define the coordinates for the two points: buyer (origin) and store (destination)
    origin = (fdat['BuyerLat'][i], fdat['BuyerLong'][i])
    destination = (fdat['NearestStoreLat'][i], fdat['NearestStoreLong'][i])

    # Find the nearest road using the Google Maps API for both the buyer and the store
    origin_geocode_result = gmaps.reverse_geocode(origin)  # get location info for the origin
    origin_nearest_road = origin_geocode_result[0]['formatted_address']  # get address of road nearest to origin
    destination_geocode_result = gmaps.reverse_geocode(destination)  # get location info for the destination
    destination_nearest_road = destination_geocode_result[0]['formatted_address']  # address of road nearest to o
rigin

    # Find driving distance between two points using Google Maps API
    now = datetime.now()  # get the current time
    directions_result = gmaps.directions(origin,  # request driving directions from origin to destination
                                         destination,
                                         mode="driving", # can be changed to other travel form
                                         departure_time=now)
    # If directions result is available, calculate driving distance in miles
    if directions_result:
        distance_meters = directions_result[0]['legs'][0]['distance']['value']  # extract distance in meters
        distance_miles = distance_meters * 0.000621371  # convert meters to miles
        milage = distance_miles
    else:
        milage = None

    # Add  driving distance to fdat
    fdat.at[i, 'driving_distance'] = milage

    # Print the current iteration number to track progress
    # print(f"Completed {i+1}/{len(fdat)}")  #helpful if large N
```




And there you have it. Use this tutorial to calculate accessibility of public services, locate new customers or simply plan a road trip! Enjoy!