Applied Data Science Capstone

|  |
| --- |
| Python Packages for Data Science  A Python library is a collection of functions and methods that allow you to perform lots of actions without writing any code. The libraries usually contain built in modules providing different functionalities which you can use directly. There are extensive libraries offering a broad range of facilities. Libraries are broadly divided in three groups.  Scientifics computing libraries:   * + **Pandas:** offers *data structure and tools* for effective data manipulation and analysis. Offers data structure and tools for effective data manipulation and analysis. It provides fast access to structured data. The primary instrument of Pandas is a two-dimensional table consisting of columns and rows labels which are called a DataFrame. It is designed to provide an easy indexing function   + **Numpy:** Uses arrays as their inputs and outputs. It can be extended to objects for matrices, and with a little change of coding, developers perform fast array processing Arrays & Matrices   + **SciPy:** includes functions for some advanced math problems as listed here, as well as data visualization. Integrals, solving differential equations and optimisation   Visualization libraries: *These libraries enable you to create graphs, charts and maps*   * + **Matplotlib:** The Matplotlib package is the most well-known library for data visualization. The graphs are also highly customizable.   + **Seaborn:** Another high-level visualization library is Seaborn. it is based on Matplotlib. It's very easy to generate various plots such as heat maps, time series and violin plots.   Machine learning algorithms: we're able to develop a model using our data set and obtain predictions. Here we introduce two packages   * + **Scikit-learn:** the Scikit-learn library contains tools statistical modelling, including regression, classification, clustering, and so on. This library is built on NumPy, SciPy and Matplotib.   + **Statsmodels:** is also a Python module that allows users to explore data, estimate statistical models, and perform statistical tests. |
| Learning FourSquare API with Python  *In this lab, you will learn in details how to make calls to the Foursquare API for different purposes. You will learn how to construct a URL to send a request to the API to search for a specific type of venues, to explore a particular venue, to explore a Foursquare user, to explore a geographical location, and to get trending venues around a location. Also, you will learn how to use the visualization library, Folium, to visualize the results.*  Table of Contents   * Foursquare API Search Function * Explore a Given Venue * Explore a User * Foursquare API Explore Function * Get Trending Venues   Import necessary Libraries  import requests # library to handle requests  import pandas as pd # library for data analsysis  import numpy as np # library to handle data in a vectorized manner  import random # library for random number generation  ​  !conda install -c conda-forge geopy --yes  from geopy.geocoders import Nominatim # module to convert an address into latitude and longitude values  ​  # libraries for displaying images  from IPython.display import Image  from IPython.core.display import HTML  # tranforming json file into a pandas dataframe library  from pandas.io.json import json\_normalize  ​  !conda install -c conda-forge folium=0.5.0 --yes  import folium # plotting library  ​  print('Folium installed')  print('Libraries imported.')  Define Foursquare Credentials and Version  *Make sure that you have created a Foursquare developer account and have your credentials handy*  *your-client-secret*  CLIENT\_ID = '0XUWQYJ51LOM4MNDEUUOJ1XPHCV13TQ4PIUE4SW1MADEN2U2' # your Foursquare ID  CLIENT\_SECRET = 'QBLKXICAK1INEP0KGKBYXCFC20HOZVBKDRFS23EC5UJWC0F0' # your Foursquare Secret  VERSION = '20180604'  LIMIT = 30  print('Your credentails:')  print('CLIENT\_ID: ' + CLIENT\_ID)  print('CLIENT\_SECRET:' + CLIENT\_SECRET)  Your credentails:  CLIENT\_ID: 0XUWQYJ51LOM4MNDEUUOJ1XPHCV13TQ4PIUE4SW1MADEN2U2  CLIENT\_SECRET:QBLKXICAK1INEP0KGKBYXCFC20HOZVBKDRFS23EC5UJWC0F0  *Let's again assume that you are staying at the Conrad hotel. So let's start by converting the Contrad Hotel's address to its latitude and longitude coordinates. In order to define an instance of the geocoder, we need to define a user\_agent. We will name our agent foursquare\_agent, as shown below.*  address = '102 North End Ave, New York, NY'  geolocator = Nominatim(user\_agent="foursquare\_agent")  location = geolocator.geocode(address)  latitude = location.latitude  longitude = location.longitude  print(latitude, longitude)  40.7151482 -74.0156573  1. Search for a specific venue category  *Now, let's assume that it is lunch time, and you are craving Italian food. So, let's define a query to search for Italian food that is within 500 metres from the Conrad Hotel.*  [https://api.foursquare.com/v2/venues/search?**client\_id**=’’&**client\_secret**=’’&**ll**=’’,’’&**v**=’’&**query**=’’&**radius**=’’&**limit**=](https://api.foursquare.com/v2/venues/search?client_id=’’&client_secret=’’&ll=’’,’’&v=’’&query=’’&radius=’’&limit=)’’  # Define search query and radius  search\_query = 'Italian'  radius = 500  # Define the corresponding URL  url = 'https://api.foursquare.com/v2/venues/search?client\_id={}&client\_secret={}&ll={},{}&v={}&query={}&radius={}&limit={}'.format(CLIENT\_ID, CLIENT\_SECRET, latitude, longitude, VERSION, search\_query, radius, LIMIT)  url  'https://api.foursquare.com/v2/venues/search?client\_id=0XUWQYJ51LOM4MNDEUUOJ1XPHCV13TQ4PIUE4SW1MADEN2U2&client\_secret=QBLKXICAK1INEP0KGKBYXCFC20HOZVBKDRFS23EC5UJWC0F0&ll=40.7151482,-74.0156573&v=20180604&query=Italian&radius=500&limit=30'  # Send the GET Request and examine the results  results = requests.get(url).json()  results  {  'meta': {'code': 200, 'requestId': '5ec90d59542890001be449c7'},  'response': {'venues': [{'id': '4fa862b3e4b0ebff2f749f06',  'name': "Harry's Italian Pizza Bar", 'location': {'address': '225 Murray St',  'lat': 40.71521779064671, 'lng': -74.01473940209351,  'labeledLatLngs': [{'label': 'display', 'lat': 40.71521779064671, 'lng': -74.01473940209351}, {'label': 'entrance', 'lat': 40.715361, 'lng': -74.014975}],  'distance': 77,  'postalCode': '10282',  'cc': 'US',  'city': 'New York',  'state': 'NY',  'country': 'United States',  'formattedAddress': ['225 Murray St',  'New York, NY 10282',  'United States']},  'categories': [{'id': '4bf58dd8d48988d1ca941735', 'name': 'Pizza Place', 'pluralName': 'Pizza Places', 'shortName': 'Pizza', 'icon': {'prefix':  'https://ss3.4sqi.net/img/categories\_v2/food/pizza\_', 'suffix': '.png'}, 'primary': True}],  'referralId': 'v-1590234484',  'hasPerk': False},  {[…]}  }  # Get relevant part of JSON and transform it into a pandas dataframe  Venues  [{  'id': '4fa862b3e4b0ebff2f749f06',  'name': "Harry's Italian Pizza Bar",  'location': {'address': '225 Murray St', 'lat': 40.71521779064671, 'lng': -74.01473940209351,  'labeledLatLngs': [{'label': 'display', 'lat': 40.71521779064671, 'lng': -74.01473940209351},  {'label': 'entrance', 'lat': 40.715361, 'lng': -74.014975}],  'distance': 77,  'postalCode': '10282',  'cc': 'US',  'city': 'New York',  'state': 'NY',  'country': 'United States',  'formattedAddress': ['225 Murray St', 'New York, NY 10282', 'United States']},  'categories': [{'id': '4bf58dd8d48988d1ca941735', 'name': 'Pizza Place', 'pluralName': 'Pizza Places','shortName': 'Pizza',  'icon': {'prefix': 'https://ss3.4sqi.net/img/categories\_v2/food/pizza\_', 'suffix': '.png'}, 'primary': True}],  'referralId': 'v-1590066875',  'hasPerk': False  }, {  'id': '4f3232e219836c91c7bfde94',  'name': 'Conca Cucina Italian Restaurant',  'location': {'address': '63 W Broadway', 'lat': 40.714484000000006, 'lng': -74.00980600000001,  'labeledLatLngs': [{'label': 'display','lat': 40.714484000000006, 'lng': -74.00980600000001}],  'distance': 499,  'postalCode': '10007',  'cc': 'US',  'city': 'New York',  'state': 'NY',  'country': 'United States',  'formattedAddress': ['63 W Broadway', 'New York, NY 10007', 'United States']},  'categories': [{'id': '4d4b7105d754a06374d81259', 'name': 'Food', 'pluralName': 'Food', 'shortName': 'Food',  'icon': {'prefix': 'https://ss3.4sqi.net/img/categories\_v2/food/default\_', 'suffix': '.png'}, 'primary': True}],  'referralId': 'v-1590066875',  'hasPerk': False  }]  # assign relevant part of JSON to venues  venues = results['response']['venues']  # tranform venues into a dataframe  dataframe = json\_normalize(venues)  dataframe.head()   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | id | name | categories | referralId | hasPerk | location.address | location.lat | location.lng | location.labeledLatLngs | location.distance | location.postalCode | location.cc | location.city | location.state | location.country | location.formattedAddress | | 0 | 4fa862b3e4b0ebff2f749f06 | Harry's Italian Pizza Bar | [{'id': '4bf58dd8d48988d1ca941735', 'name': 'P... | v-1590234484 | False | 225 Murray St | 40.715218 | -74.014739 | [{'label': 'display', 'lat': 40.71521779064671... | 77 | 10282 | US | New York | NY | United States | [225 Murray St, New York, NY 10282, United Sta... | | 1 | 4f3232e219836c91c7bfde94 | Conca Cucina Italian Restaurant | [{'id': '4d4b7105d754a06374d81259', 'name': 'F... | v-1590234484 | False | 63 W Broadway | 40.714484 | -74.009806 | [{'label': 'display', 'lat': 40.71448400000000... | 499 | 10007 | US | New York | NY | United States | [63 W Broadway, New York, NY 10007, United Sta... |   # Define information of interest and filter dataframe  # keep only columns that include venue name, and anything that is associated with location  filtered\_columns = ['name', 'categories'] + [col for col in dataframe.columns if col.startswith('location.')] + ['id']  dataframe\_filtered = dataframe.loc[:, filtered\_columns]  # function that extracts the category of the venue  def get\_category\_type(row):  try:  categories\_list = row['categories']  except:  categories\_list = row['venue.categories']    if len(categories\_list) == 0:  return None  else:  return categories\_list[0]['name']  # filter the category for each row  dataframe\_filtered['categories'] = dataframe\_filtered.apply(get\_category\_type, axis=1)  # clean column names by keeping only last term  dataframe\_filtered.columns = [column.split('.')[-1] for column in dataframe\_filtered.columns]  dataframe\_filtered   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | name | categories | address | lat | lng | labeledLatLngs | distance | postalCode | cc | city | state | country | formattedAddress | id | | 0 | Harry's Italian Pizza Bar | Pizza Place | 225 Murray St | 40.715218 | -74.014739 | [{'label': 'display', 'lat': 40.71521779064671... | 77 | 10282 | US | New York | NY | United States | [225 Murray St, New York, NY 10282, United Sta... | 4fa862b3e4b0ebff2f749f06 | | 1 | Conca Cucina Italian Restaurant | Food | 63 W Broadway | 40.714484 | -74.009806 | [{'label': 'display', 'lat': 40.71448400000000... | 499 | 10007 | US | New York | NY | United States | [63 W Broadway, New York, NY 10007, United Sta... | 4f3232e219836c91c7bfde94 |   # Let's visualize the Italian restaurants that are nearby  dataframe\_filtered.name  0 Harry's Italian Pizza Bar  1 Conca Cucina Italian Restaurant  Name: name, dtype: object  venues\_map = folium.Map(location=[latitude, longitude], zoom\_start=13) # generate map centred ~ the Conrad Hotel  # add a red circle marker to represent the Conrad Hotel  folium.features.CircleMarker(  [latitude, longitude],  radius=10,  color='red',  popup='Conrad Hotel',  fill = True,  fill\_color = 'red',  fill\_opacity = 0.6  ).add\_to(venues\_map)  # add the Italian restaurants as blue circle markers  for lat, lng, label in zip(dataframe\_filtered.lat, dataframe\_filtered.lng, dataframe\_filtered.categories):  folium.features.CircleMarker(  [lat, lng],  radius=5,  color='blue',  popup=label,  fill = True,  fill\_color='blue',  fill\_opacity=0.6  ).add\_to(venues\_map)  # display map  venues\_map  2.Explore a Given Venue  https://api.foursquare.com/v2/venues/VENUE\_ID?**client\_id**=CLIENT\_ID&**client\_secret**=CLIENT\_SECRET&**v**=VERSION  *A. Let's explore the closest Italian restaurant -- Harry's Italian Pizza B*  # ID of Harry's Italian Pizza Bar  venue\_id = '4fa862b3e4b0ebff2f749f06'  # Define URL  url = 'https://api.foursquare.com/v2/venues/{}?client\_id={}&client\_secret={}&v={}'.format(venue\_id, CLIENT\_ID, CLIENT\_SECRET, VERSION)  url  'https://api.foursquare.com/v2/venues/4fa862b3e4b0ebff2f749f06?client\_id=0XUWQYJ51LOM4MNDEUUOJ1XPHCV13TQ4PIUE4SW1MADEN2U2&client\_secret=QBLKXICAK1INEP0KGKBYXCFC20HOZVBKDRFS23EC5UJWC0F0&v=20180604'  # Send GET request for result  result = requests.get(url).json()  print(result['response']['venue'].keys())  result['response']['venue']  dict\_keys(['id', 'name', 'contact', 'location', 'canonicalUrl', 'categories', 'verified', 'stats', 'url', 'price', 'hasMenu', 'likes', 'dislike', 'ok', 'rating', 'ratingColor', 'ratingSignals', 'menu', 'allowMenuUrlEdit', 'beenHere', 'specials', 'photos', 'reasons', 'hereNow', 'createdAt', 'tips', 'shortUrl', 'timeZone', 'listed', 'hours', 'popular', 'seasonalHours', 'defaultHours', 'pageUpdates', 'inbox', 'attributes', 'bestPhoto', 'colors'])  *B. Get the venue's overall rating*  try:  print(result['response']['venue']['rating'])  except:  print('This venue has not been rated yet.')  6.4  *That is not a very good rating. Let's check the rating of the second closest Italian restaurant.*  venue\_id = '4f3232e219836c91c7bfde94' # ID of Conca Cucina Italian Restaurant  url = 'https://api.foursquare.com/v2/venues/{}?client\_id={}&client\_secret={}&v={}'.format(venue\_id, CLIENT\_ID, CLIENT\_SECRET, VERSION)  ​  result = requests.get(url).json()  try:  print(result['response']['venue']['rating'])  except:  print('This venue has not been rated yet.')  This venue has not been rated yet.  *Since this restaurant has no ratings, let's check the third restaurant.*  venue\_id = '3fd66200f964a520f4e41ee3' # ID of Ecco  url = 'https://api.foursquare.com/v2/venues/{}?client\_id={}&client\_secret={}&v={}'.format(venue\_id, CLIENT\_ID, CLIENT\_SECRET, VERSION)  ​  result = requests.get(url).json()  try:  print(result['response']['venue']['rating'])  except:  print('This venue has not been rated yet.')  7.4  *Since this restaurant has a slightly better rating, let's explore it further.*  *C. Get the number of tips*  result['response']['venue']['tips']['count']  19  *D. Get the venue's tips*  https://api.foursquare.com/v2/venues/VENUE\_ID/tips?client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET&v=VERSION&limit=LIMIT  # Create URL and send GET request. Make sure to set limit to get all tips  # Ecco Tips  limit = 15 # set limit to be greater than or equal to the total number of tips  url = 'https://api.foursquare.com/v2/venues/{}/tips?client\_id={}&client\_secret={}&v={}&limit={}'.format(venue\_id, CLIENT\_ID, CLIENT\_SECRET, VERSION, limit)  ​  results = requests.get(url).json()  results  # Get tips and list of associated features  tips = results['response']['tips']['items']  ​tip = results['response']['tips']['items'][0]  tip.keys()  dict\_keys(['id', 'createdAt', 'text', 'type', 'canonicalUrl', 'lang', 'likes', 'logView', 'agreeCount', 'disagreeCount', 'todo', 'user', 'authorIn..Type'])  # Format column width and display all tips  pd.set\_option('display.max\_colwidth', None) # use -1 for no limit  tips\_df = json\_normalize(tips) # json normalize tips  # columns to keep  filtered\_columns = ['text', 'agreeCount', 'disagreeCount', 'id', 'user.firstName', 'user.lastName', 'user.id']  tips\_filtered = tips\_df.loc[:, filtered\_columns]  # display tips  tips\_filtered   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | text | agreeCount | disagreeCount | id | user.firstName | user.lastName | user.id | | 0 | A+ Italian food! Trust me on this: my mom’s side of the family is 100% Italian. I was born and bred to know good pasta when I see it, and Ecco is one of my all-time NYC favorites | 4 | 0 | 5ab1cb46c9a517174651d3fe | Nick | E | 484542633 |   *Now remember that because we are using a personal developer account, then we can access only 2 of the restaurant's tips, instead of all 15 tips.*  *3. Search a Foursquare User*  https://api.foursquare.com/v2/users/USER\_ID?client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET&v=VERSION  # Define URL, send GET request and display features associated with user  user\_id = '484542633' # user ID with most agree counts and complete profile  # Define URL  url = 'https://api.foursquare.com/v2/users/{}?client\_id={}&client\_secret={}&v={}'.format(user\_id, CLIENT\_ID, CLIENT\_SECRET, VERSION)  ​  # Send GET request  results = requests.get(url).json()  user\_data = results['response']['user']  ​  # display features associated with user  user\_data.keys()  print('First Name: ' + user\_data['firstName'])  print('Last Name: ' + user\_data['lastName'])  print('Home City: ' + user\_data['homeCity'])  *Wow! So it turns out that Nick is a very active Foursquare user, with more than 250 tips. (devloper account to can view)*  # *Get User's tips*  # define tips URL  url = 'https://api.foursquare.com/v2/users/{}/tips?client\_id={}&client\_secret={}&v={}&limit={}'.format(user\_id, CLIENT\_ID, CLIENT\_SECRET, VERSION, limit)  print(url)  https://api.foursquare.com/v2/users/484542633/tips?client\_id=0XUWQYJ51LOM4MNDEUUOJ1XPHCV13TQ4PIUE4SW1MADEN2U2&client\_secret=QBLKXICAK1INEP0KGKBYXCFC20HOZVBKDRFS23EC5UJWC0F0&v=20180604&limit=15  # send GET request and get user's tips  results = requests.get(url).json()  tips = results['response']['tips']['items']  ​  # format column width  pd.set\_option('display.max\_colwidth', -1)  ​tips\_df = json\_normalize(tips)  ​  # filter columns  filtered\_columns = ['text', 'agreeCount', 'disagreeCount', 'id']  tips\_filtered = tips\_df.loc[:, filtered\_columns]  ​  # display user's tips  tips\_filtered   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | text | agreeCount | disagreeCount | id | | 0 | They serve coffee!!!!!! | 1 | 0 | 5accc98c0313204c9d7ec157 | | 1 | Quick, cheap lunch that tastes good! Way shorter line than Chipotle, too. | 2 | 0 | 5acbec70a0215b732e264fe8 | | 2 | You’re not a real New Yorker until you’ve shame-ordered Insomnia Cookies for delivery at 3am | 1 | 0 | 5acbbd4eb1538e45373b07f5 | | 3 | Good for you yet still tasty! Clean green protein is my go-to after I hit the gym 💪 | 2 | 0 | 5acbbcda01235808d5d6dc75 | | 4 | Burger game strong 💪 | 1 | 0 | 5ab575fb6bdee65f759da8c1 | | 5 | Great burgers & fries! Also, this place is exactly what it’s like when you go to a bar in the Southwest. Source: I’m from Arizona. | 2 | 0 | 5ab5575d73fe2516ad8f363b | | 6 | Açaí bowl + peanut butter + whey protein = 💪💪💪 | 1 | 0 | 5ab42db53c858d64af2688a4 | | 7 | Highly underrated and way less crowded than Central Park! | 3 | 0 | 5ab42c396f706a29f53ad1a8 | | 8 | Way easier to navigate than the Met proper, plus the Met Breuer focuses on modern art. If I only have a limited amount of time to spend in a museum, I would rather go here than anywhere else! | 6 | 0 | 5ab42b987dc9e17930e5ff5b | | 9 | Get the açaí bowl with peanut butter after your work out and thank me later 👌 | 1 | 0 | 5ab42aca2a7ab6333652b266 | | 10 | When you want a burger, this should be the first thing that comes to mind. A+! | 1 | 0 | 5ab42a28da5e5617d18e3a6a | | 11 | Way less crowded than Central Park! People who live in the neighbourhood rave about Carl Schurz Park. | 3 | 0 | 5ab429db1ffe971b060083f5 | | 12 | The best Mexican food in the Murray Hill / Kips Bay area! | 1 | 0 | 5ab3f53f8496ca57542d5549 | | 13 | Best coffee shop in the neighbourhood! | 1 | 0 | 5ab3f428da5e5617d17d1475 | | 14 | When there’s nice weather, the rooftop at Tonic East is the best place to watch the game. Perfect for March Madness & NBA finals! | 2 | 0 | 5ab3f3fedd70c572de886c9d |   *Let's get the venue for the tip with the greatest number of agree counts*  tip\_id = '5ab5575d73fe2516ad8f363b' # tip id  ​  # define URL  url = 'http://api.foursquare.com/v2/tips/{}?client\_id={}&client\_secret={}&v={}'.format(tip\_id, CLIENT\_ID, CLIENT\_SECRET, VERSION)  ​  # send GET Request and examine results  result = requests.get(url).json()  print(result['response']['tip']['venue']['name'])  print(result['response']['tip']['venue']['location'])  Cowgirl  {'address': '519 Hudson St', 'crossStreet': 'at W 10th St', 'lat': 40.73373338282062, 'lng': -74.0062998849649, 'labeledLatLngs': [{'label': 'display', 'lat': 40.73373338282062, 'lng': -74.0062998849649}], 'postalCode': '10014', 'cc': 'US', 'city': 'New York', 'state': 'NY', 'country': 'United States', 'formattedAddress': ['519 Hudson St (at W 10th St)', 'New York, NY 10014', 'United States']}  # Get User's friends  user\_friends = json\_normalize(user\_data['friends']['groups'][0]['items'])  user\_friends  error! – due account limitation  *Interesting. Despite being very active, it turns out that Nick does not have any friends on Foursquare. This might definitely change in the future.*  Retrieve the User's Profile Image  user\_data  # 1. grab prefix of photo  # 2. grab suffix of photo  # 3. concatenate them using the image size  Image(url='https://igx.4sqi.net/img/user/300x300/484542633\_mK2Yum7T\_7Tn9fWpndidJsmw2Hof\_6T5vJBKCHPLMK5OL-U5ZiJGj51iwBstcpDLYa3Zvhvis.jpg')  4. Explore a location  https://api.foursquare.com/v2/venues/explore?client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET&ll=LATITUDE,LONGITUDE&v=VERSION&limit=LIMIT  So, you just finished your gourmet dish at Ecco, and are just curious about the popular spots around the restaurant. In order to explore the area, let's start by getting the latitude and longitude values of Ecco Restaurant.  latitude = 40.715337  longitude = -74.008848  # Define URL  url = 'https://api.foursquare.com/v2/venues/explore?client\_id={}&client\_secret={}&ll={},{}&v={}&radius={}&limit={}'.format(CLIENT\_ID, CLIENT\_SECRET, latitude, longitude, VERSION, radius, LIMIT)  url  'https://api.foursquare.com/v2/venues/explore?client\_id=0XUWQYJ51LOM4MNDEUUOJ1XPHCV13TQ4PIUE4SW1MADEN2U2&client\_secret=QBLKXICAK1INEP0KGKBYXCFC20HOZVBKDRFS23EC5UJWC0F0&ll=40.715337,-74.008848&v=20180604&radius=500&limit=30'  # Send GET request and examine results  import requests  import requests  results = requests.get(url).json()  'There are {} around Ecco restaurant.'.format(len(results['response']['groups'][0]['items']))  'There are 30 around Ecco restaurant.'  # Get relevant part of JSON  items = results['response']['groups'][0]['items']  items[0]  # Process JSON and convert it to a clean dataframe  dataframe = json\_normalize(items) # flatten JSON  ​  # filter columns  filtered\_columns = ['venue.name', 'venue.categories'] + [col for col in dataframe.columns if col.startswith('venue.location.')] + ['venue.id']  dataframe\_filtered = dataframe.loc[:, filtered\_columns]  ​  # filter the category for each row  dataframe\_filtered['venue.categories'] = dataframe\_filtered.apply(get\_category\_type, axis=1)  ​  # clean columns  dataframe\_filtered.columns = [col.split('.')[-1] for col in dataframe\_filtered.columns]  ​dataframe\_filtered.head(10)   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | name | categories | address | crossStreet | lat | lng | labeledLatLngs | distance | postalCode | cc | city | state | country | formattedAddress | neighbourhood | id | | 1 | Korin | Furniture / Home Store | 57 Warren St | Church St | 40.714824 | -74.009404 | [{'label': 'display', 'lat': 40.71482437714839, 'lng': -74.00940425461492}, {'label': 'entrance', 'lat': 40.714727, 'lng': -74.009399}] | 73 | 10007 | US | New York | NY | United States | [57 Warren St (Church St), New York, NY 10007, United States] | Tribeca | 4af5d65ff964a52091fd21e3 | | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |   *Let's visualize these items on the map around our location*  venues\_map = folium.Map(location=[latitude, longitude], zoom\_start=15) # generate map centred around Ecco  ​​# add Ecco as a red circle mark  folium.features.CircleMarker( [latitude, longitude], radius=10, popup='Ecco', fill=True, color='red', fill\_color='red', fill\_opacity=0.6).add\_to(venues\_map)  ​​# add popular spots to the map as blue circle markers  for lat, lng, label in zip(dataframe\_filtered.lat, dataframe\_filtered.lng, dataframe\_filtered.categories):  folium.features.CircleMarker(  [lat, lng],  radius=5,  popup=label,  fill=True,  color='blue',  fill\_color='blue',  fill\_opacity=0.6  ).add\_to(venues\_map)  ​  # display map  venues\_map  5. Explore Trending Venues  https://api.foursquare.com/v2/venues/trending?client\_id=CLIENT\_ID&client\_secret=CLIENT\_SECRET&ll=LATITUDE,LONGITUDE&v=VERSION  *Now, instead of simply exploring the area around Ecco, you are interested in knowing the venues that are trending at the time you are done with your lunch, meaning the places with the highest foot traffic. So let's do that and get the trending venues around Ecco.*  # define URL  url = 'https://api.foursquare.com/v2/venues/trending?client\_id={}&client\_secret={}&ll={},{}&v={}'.format(CLIENT\_ID, CLIENT\_SECRET, latitude, longitude, VERSION)  ​# send GET request and get trending venues  results = requests.get(url).json()  results  {'meta': {'code': 200, 'requestId': '5ec93455618f43001b014cd5'},'response': {'venues': []}}  # Check if any venues are trending at this time  if len(results['response']['venues']) == 0:  trending\_venues\_df = 'No trending venues are available at the moment!'  else:  trending\_venues = results['response']['venues']  trending\_venues\_df = json\_normalize(trending\_venues)  ​  # filter columns  columns\_filtered = ['name', 'categories'] + ['location.distance', 'location.city', 'location.postalCode', 'location.state', 'location.country', 'location.lat', 'location.lng']  trending\_venues\_df = trending\_venues\_df.loc[:, columns\_filtered]  ​  # filter the category for each row  trending\_venues\_df['categories'] = trending\_venues\_df.apply(get\_category\_type, axis=1)  # display trending venues  trending\_venues\_df  'No trending venues are available at the moment!'  *Now, depending on when you run the above code, you might get different venues since the venues with the highest foot traffic are fetched live.*  # Visualize trending venues  if len(results['response']['venues']) == 0:  venues\_map = 'Cannot generate visual as no trending venues are available at the moment!'​  else:  venues\_map = folium.Map(location=[latitude, longitude], zoom\_start=15) # generate map centred around Ecco​  ​  # add Ecco as a red circle mark  folium.features.CircleMarker( [latitude, longitude], radius=10, popup='Ecco', fill=True, color='red', fill\_color='red', fill\_opacity=0.6).add\_to(venues\_map)  ​  ​ # add the trending venues as blue circle markers  for lat, lng, label in zip(trending\_venues\_df['location.lat'], trending\_venues\_df['location.lng'], trending\_venues\_df['name']):  folium.features.CircleMarker(  [lat, lng], radius=5, poup=label, fill=True, color='blue', fill\_color='blue', fill\_opacity=0.6).add\_to(venues\_map)  # display map  venues\_map  'Cannot generate visual as no trending venues are available at the moment!' |
| k-means Clustering  *There are many models for clustering out there. In this lab, we will be presenting the model that is considered the one of the simplest model among them. Despite its simplicity, k-means is vastly used for clustering in many data science applications, especially useful if you need to quickly discover insights from unlabeled data.*  *Some real-world applications of k-means include:*   * customer segmentation, * understand what the visitors of a website are trying to accomplish, * pattern recognition, and, * data compression.   *In this lab, we will learn k-means clustering with 2 examples:*   * k-means on a randomly generated dataset. * Using k-means for customer segmentation.   *Before we start with the main lab content, let's download all the dependencies that we will need.*  import random # library for random number generation  import numpy as np # library for vectorized computation  import pandas as pd # library to process data as dataframes  ​  import matplotlib.pyplot as plt # plotting library  # backend for rendering plots within the browser  %matplotlib inline  ​  from sklearn.cluster import KMeans  from sklearn.datasets.samples\_generator import make\_blobs  ​  print('Libraries imported.')  1. k-means on a Randomly Generated Dataset  *Let's first demonstrate how k-means works with an example of engineered datapoints.*  *30 data points belonging to 2 different clusters (x1 is the first feature and x2 is the second feature)*  # data  x1 = [-4.9, -3.5, 0, -4.5, -3, -1, -1.2, -4.5, -1.5, -4.5, -1, -2, -2.5, -2, -1.5, 4, 1.8, 2, 2.5, 3, 4, 2.25, 1, 0, 1, 2.5, 5, 2.8, 2, 2]  x2 = [-3.5, -4, -3.5, -3, -2.9, -3, -2.6, -2.1, 0, -0.5, -0.8, -0.8, -1.5, -1.75, -1.75, 0, 0.8, 0.9, 1, 1, 1, 1.75, 2, 2.5, 2.5, 2.5, 2.5, 3, 6, 6.5]  ​print('Datapoints defined!')  # Define a function that assigns each datapoint to a cluster  colors\_map = np.array(['b', 'r'])  def assign\_members(x1, x2, centers):  compare\_to\_first\_center = np.sqrt(np.square(np.array(x1) - centers[0][0]) + np.square(np.array(x2) - centers[0][1]))  compare\_to\_second\_center = np.sqrt(np.square(np.array(x1) - centers[1][0]) + np.square(np.array(x2) - centers[1][1]))  class\_of\_points = compare\_to\_first\_center > compare\_to\_second\_center  colors = colors\_map[class\_of\_points + 1 - 1]  return colors, class\_of\_points  ​print('assign\_members function defined!')  # Define a function that updates the centroid of each cluster  # update means  def update\_centers(x1, x2, class\_of\_points):  center1 = [np.mean(np.array(x1)[~class\_of\_points]), np.mean(np.array(x2)[~class\_of\_points])]  center2 = [np.mean(np.array(x1)[class\_of\_points]), np.mean(np.array(x2)[class\_of\_points])]  return [center1, center2]  ​print('update\_centers function defined!')  # Define a function that plots the data points along with the cluster centroids  def plot\_points(centroids=None, colors='g', figure\_title=None):  # plot the figure  fig = plt.figure(figsize=(10, 6)) # create a figure object  ax = fig.add\_subplot(1, 1, 1)    centroid\_colors = ['bx', 'rx']  if centroids:  for (i, centroid) in enumerate(centroids):  ax.plot(centroid[0], centroid[1], centroid\_colors[i], markeredgewidth=5, markersize=20)  plt.scatter(x1, x2, s=500, c=colors)    # define the ticks; xticks = np.linspace(-6, 8, 15, endpoint=True); yticks = np.linspace(-6, 6, 13, endpoint=True);  ​  # fix the horizontal axis; ax.set\_xticks(xticks); ax.set\_yticks(yticks);  ​  # add tick labels; xlabels = xticks; ax.set\_xticklabels(xlabels); ylabels = yticks; ax.set\_yticklabels(ylabels);  ​  # style the ticks; ax.xaxis.set\_ticks\_position('bottom'); ax.yaxis.set\_ticks\_position('left');  # tick parameter; ax.tick\_params('both', length=2, width=1, which='major', labelsize=15)  # add labels to axes; ax.set\_xlabel('x1', fontsize=20); ax.set\_ylabel('x2', fontsize=20)  # add title to figure; ax.set\_title(figure\_title, fontsize=24)  ​ plt.show()  ​print('plot\_points function defined!')  *Now let’s Initialize k-means - plot data points*  plot\_points(figure\_title='Scatter Plot of x2 vs x1')  # Initialize k-means - randomly define clusters and add them to plot  centers = [[-2, 2], [2, -2]]  plot\_points(centers, figure\_title='k-means Initialization')  # Run k-means (4-iterations only)  number\_of\_iterations = 4  for i in range(number\_of\_iterations):  input('Iteration {} - Press Enter to update the members of each cluster'.format(i + 1))  colors, class\_of\_points = assign\_members(x1, x2, centers)  title = 'Iteration {} - Cluster Assignment'.format(i + 1)  plot\_points(centers, colors, figure\_title=title)  input('Iteration {} - Press Enter to update the centers'.format(i + 1))  centers = update\_centers(x1, x2, class\_of\_points)  title = 'Iteration {} - Centroid Update'.format(i + 1)  plot\_points(centers, colors, figure\_title=title)  *Now, we have visually observed how k-means works, let's look at an example with many more datapoints. For this example, we will use the random library to generate thousands of datapoints. Let’s create our own dataset for this lab!, First, we need to set up a random seed.*  # Use numpy's random.seed() function, where the seed will be set to 0  np.random.seed(0)  *Next, we will be making random clusters of points by using the make\_blobs class. The make\_blobs class can take in many inputs, but we will be using these specific ones.*  # Input   * n\_samples: The total number of points equally divided among clusters.   Value will be: 5000   * centers: The number of centers to generate, or the fixed center locations.   Value will be: [[4, 4], [-2, -1], [2, -3],[1,1]]   * cluster\_std: The standard deviation of the clusters.   Value will be: 0.9  # Output   * X: Array of shape [n\_samples, n\_features]. (Feature Matrix)   The generated samples.   * y: Array of shape [n\_samples]. (Response Vector)   The integer labels for cluster membership of each sample.  X, y = make\_blobs(n\_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1, 1]], cluster\_std=0.9)  # Display the scatter plot of the randomly generated data.  plt.figure(figsize=(8, 6))  plt.scatter(X[:, 0], X[:, 1], marker='.')  # Setting up K-Means  *Now that we have our random data, let's set up our K-Means Clustering.*  *The KMeans class has many parameters that can be used, but we will be using these three:*   * *init: Initialization method of the centroids.*   *Value will be: "k-means++"*  *k-means++: Selects initial cluster canter’s for k-mean clustering in a smart way to speed up convergence.*   * *n\_clusters: The number of clusters to form as well as the number of centroids to generate.*   *Value will be: 4 (since we have 4 centres’)*   * *n\_init: Number of times the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n\_init consecutive runs in terms of inertia.*   *Value will be: 12*  # Initialize KMeans with these parameters, where the output parameter is called k\_means.  k\_means = KMeans(init = "k-means++", n\_clusters = 4, n\_init = 12)  # Now let's fit the KMeans model with the feature matrix we created above, X  k\_means.fit(X)  # Now let's grab the labels for each point in the model using KMeans' .labels\_ attribute and save it as k\_means\_labels  k\_means\_labels = k\_means.labels\_  k\_means\_labels  array([0, 3, 3, ..., 1, 0, 0], dtype=int32)  *We will also get the coordinates of the cluster centers using KMeans' .cluster\_centers\_ and save it as:*  k\_means\_cluster\_centers = k\_means.cluster\_centers\_  k\_means\_cluster\_centers  Creating the Visual Plot  *So now that we have the random data generated and the KMeans model initialized, let's plot them. Please read through the code and comments to understand how to plot the model.*  # Initialize the plot with the specified dimensions.  fig = plt.figure(figsize=(6, 4))  ​  # Colors uses a color map, which will produce an array of colors based on the number of labels there are.  # We use set(k\_means\_labels) to get the unique labels.  colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k\_means\_labels))))  ​  # Create a plot  ax = fig.add\_subplot(1, 1, 1)  ​  # For loop that plots the data points and centroids.  # k will range from 0-3, which will match the possible clusters that each data point is in.  for k, col in zip(range(len([[4,4], [-2, -1], [2, -3], [1, 1]])), colors):  ​  # Create a list of all data points, where the data points that are in the cluster (ex. cluster 0) are labelled as true, else they are labelled as false.  my\_members = (k\_means\_labels == k)    # Define the centroid, or cluster center.  cluster\_center = k\_means\_cluster\_centers[k]    # Plots the datapoints with colour col.  ax.plot(X[my\_members, 0], X[my\_members, 1], 'w', markerfacecolor=col, marker='.')    # Plots the centroids with specified color, but with a darker outline  ax.plot(cluster\_center[0], cluster\_center[1], 'o', markerfacecolor=col, markeredgecolor='k', markersize=6)  ​  ax.set\_title('KMeans') # Title of the plot  # Remove x-axis ticks, y-axis ticks  ax.set\_xticks(())​  ax.set\_yticks(())  ​  plt.show() # Show the plot  2. Customer Segmentation with K-Means  *Imagine that you have a customer dataset, and you need to apply customer segmentation on this historical data. Customer segmentation is the practice of partitioning a customer base into groups of individuals that have similar characteristics. It is a significant strategy as a business can target these specific groups of customers and effectively allocate marketing resources. For example, one group might contain customers who are high-profit and low-risk, that is, more likely to purchase products, or subscribe for a service. A business task is to retaining those customers. Another group might include customers from non-profit organizations. And so on.*  # Let's download the data and save it as a CSV file called customer\_segmentation.csv  !wget -q -O 'customer\_segmentation.csv' https://cocl.us/customer\_dataset  # Now that the data is downloaded, let's read it into a pandas dataframe.  customers\_df = pd.read\_csv("Cust\_Segmentation.csv")  customers\_df.head()  # Pre-processing  *As you can see, Address in this dataset is a categorical variable. k-means algorithm isn't directly applicable to categorical variables because Euclidean distance function isn't really meaningful for discrete variables. So, lets drop this feature and run clustering.*  df = customers\_df.drop('Address', axis=1)  df.head()   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Customer Id | Age | Edu | Years Employed | Income | Card Debt | Other Debt | Defaulted | DebtIncomeRatio | | 0 | 1 | 41 | 2 | 6 | 19 | 0.124 | 1.073 | 0.0 | 6.3 | | 1 | 2 | 47 | 1 | 26 | 100 | 4.582 | 8.218 | 0.0 | 12.8 | | 2 | 3 | 33 | 2 | 10 | 57 | 6.111 | 5.802 | 1.0 | 20.9 | | 3 | 4 | 29 | 2 | 4 | 19 | 0.681 | 0.516 | 0.0 | 6.3 | | 4 | 5 | 47 | 1 | 31 | 253 | 9.308 | 8.908 | 0.0 | 7.2 |   # Normalizing over the standard deviation  *Now let's normalize the dataset. But why do we need normalization in the first place? Normalization is a statistical method that helps mathematical-based algorithms to interpret features with different magnitudes and distributions equally. We use StandardScaler() to normalize our dataset.*  from sklearn.preprocessing import StandardScaler  X = df.values[:,1:]  X = np.nan\_to\_num(X)  Cluster\_dataset = StandardScaler().fit\_transform(X)  Cluster\_dataset  Modelling  *In our example (if we didn't have access to the k-means algorithm), it would be the same as guessing that each customer group would have certain age, income, education, etc, with multiple tests and experiments. However, using the K-means clustering we can do all this process much easier.*  # Let’s apply k-means on our dataset, and take look at cluster labels.  clusterNum = 3  k\_means = KMeans(init = "k-means++", n\_clusters = clusterNum, n\_init = 12)  k\_means.fit(X)  labels = k\_means.labels\_  print(labels)  # Insights, note that each row in our dataset represents a customer, and therefore, each row is assigned a label.  df["labels "] = labels  df.head(5)   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Customer Id | Age | Edu | Years Employed | Income | Card Debt | Other Debt | Defaulted | DebtIncomeRatio | labels | | 0 | 1 | 41 | 2 | 6 | 19 | 0.124 | 1.073 | 0.0 | 6.3 | 0 | | 1 | 2 | 47 | 1 | 26 | 100 | 4.582 | 8.218 | 0.0 | 12.8 | 2 | | 2 | 3 | 33 | 2 | 10 | 57 | 6.111 | 5.802 | 1.0 | 20.9 | 1 | | 3 | 4 | 29 | 2 | 4 | 19 | 0.681 | 0.516 | 0.0 | 6.3 | 0 | | 4 | 5 | 47 | 1 | 31 | 253 | 9.308 | 8.908 | 0.0 | 7.2 | 2 |   *We can easily check the centroid values by averaging the features in each cluster.*  df.groupby('Clus\_km').mean()   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | labels | Cust\_Id | Age | Edu | Yrs Employed | Income | Card Debt | Oth. Debt | Defaulted | DebtIncome  Ratio | | 0 | 426.122905 | 33.817505 | 1.603352 | 7.625698 | 36.143389 | 0.853128 | 1.816855 | 0.000000 | 7.964991 | | 1 | 424.451807 | 31.891566 | 1.861446 | 3.963855 | 31.789157 | 1.576675 | 2.843355 | 0.993939 | 13.994578 | | 2 | 424.408163 | 43.000000 | 1.931973 | 17.197279 | 101.959184 | 4.220673 | 7.954483 | 0.162393 | 13.915646 |   *k-means will partition your customers into three groups since we specified the algorithm to generate 3 clusters. The customers in each cluster are similar to each other in terms of the features included in the dataset.*  *Now we can create a profile for each group, considering the common characteristics of each cluster. For example, the 3 clusters can be:*   * OLDER, HIGH INCOME, AND INDEBTED * MIDDLE AGED, MIDDLE INCOME, AND FINANCIALLY RESPONSIBLE * YOUNG, LOW INCOME, AND INDEBTED   *However, you can devise your own profiles based on the means above and come up with labels that you think best describe each cluster.*  *I hope that you are able to see the power of k-means here. This clustering algorithm provided us with insight into the dataset and lead us to group the data into three clusters. Perhaps the same results would have been achieved but using multiple tests and experiments.* |
| Segmenting and Clustering neighbourhoods in New York City  *In this lab, you will learn how to convert addresses into their equivalent latitude and longitude values. Also, you will use the Foursquare API to explore neighbourhoods in New York City. You will use the explore function to get the most common venue categories in each neighbourhood, and then use this feature to group the neighbourhoods into clusters. You will use the k-means clustering algorithm to complete this task. Finally, you will use the Folium library to visualize the neighbourhoods in New York City and their emerging clusters.*  *Table of Contents*   * *Download and Explore Dataset* * *Explore Neighbourhoods in New York City* * *Analyse Each Neighbourhood* * *Cluster neighbourhoods* * *Examine Clusters*   *Before we get the data and start exploring it, let's download all the dependencies that we will need.*  import numpy as np # library to handle data in a vectorized manner  ​import pandas as pd # library for data analsysis  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.max\_rows', None)  ​import json # library to handle JSON files  ​  !conda install -c conda-forge geopy --yes # uncomment this line if you haven't completed the Foursquare API lab  from geopy.geocoders import Nominatim # convert an address into latitude and longitude values  ​  import requests # library to handle requests  from pandas.io.json import json\_normalize # transform JSON file into a pandas dataframe  ​  # Matplotlib and associated plotting modules  import matplotlib.cm as cm  import matplotlib.colors as colors  ​  # import k-means from clustering stage  from sklearn.cluster import KMeans  ​  #!conda install -c conda-forge folium=0.5.0 --yes # uncomment this line if you haven't completed the Foursquare API lab  import folium # map rendering library  ​  print('Libraries imported.')  1. Download and Explore Dataset  *Neighbourhood has a total of 5 boroughs and 306 neighbourhoods. In order to segment the neighbourhoods and explore them, we will essentially need a dataset that contains the 5 boroughs and the neighbourhoods that exist in each borough as well as the latitude and longitude coordinates of each neighbourhood.*  *Luckily, this dataset exists for free on the web. Feel free to try to find this dataset on your own, but here is the link to the dataset:* [*https://geo.nyu.edu/catalog/nyu\_2451\_34572*](https://geo.nyu.edu/catalog/nyu_2451_34572)  *For your convenience, I downloaded the files and placed it on the server, so you can simply run a wget command and access the data. So, let's go ahead and do that.*  !wget -q -O 'newyork\_data.json' https://cocl.us/new\_york\_dataset  print('Data downloaded!')  *Next, let's load and explore the data.*  # Load the data.  with open('newyork\_data.json') as json\_data:  newyork\_data = json.load(json\_data)  newyork\_data.keys  dict\_keys(['type', 'totalFeatures', 'features', 'crs', 'bbox'])  # Let's take a quick look at the data.  newyork\_data['features'][0]  {'type': 'Feature', 'id': 'nyu\_2451\_34572.1',  'geometry': {'type': 'Point', 'coordinates': [-73.84720052054902, 40.89470517661]}, 'geometry\_name': 'geom', 'properties': {'name': 'Wakefield', 'stacked': 1, 'annoline1': 'Wakefield', 'annoline2': None, 'annoline3': None, 'annoangle': 0.0, 'borough': 'Bronx', 'bbox': [-73.84720052054902, 40.89470517661, -73.84720052054902, 40.89470517661]}}  *Notice how all the relevant data is in the features key, which is basically a list of the neighbourhoods. So, let's define a new variable that includes this data.*  neighbourhoods\_data = newyork\_data['features']  *Let's take a look at the first item in this list.*  neighbourhoods\_data[0]  'type': 'Feature',  'id': 'nyu\_2451\_34572.1',  'geometry': {'type': 'Point', 'coordinates': [-73.84720052054902, 40.89470517661]},  'geometry\_name': 'geom',  'properties': {'name': 'Wakefield', 'stacked': 1, 'annoline1': 'Wakefield', 'annoline2': None, 'annoline3': None, 'annoangle': 0.0, 'borough': 'Bronx', 'bbox': [-73.84720052054902, 40.89470517661, -73.84720052054902, 40.89470517661]}}  *Now,* *let’s transform the data into a pandas dataframe*  *The next task is essentially transforming this data of nested Python dictionaries into a pandas dataframe. So, let's start by creating an empty dataframe.*  # define the dataframe columns  column\_names = ['Borough', 'Neighbourhood', 'Latitude', 'Longitude']  ​  # instantiate the dataframe  neighbourhoods = pd.DataFrame(columns=column\_names)  # take a look at the empty dataframe to confirm that the columns are as intended.  neighbourhoods   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Borough | Neighbourhood | Latitude | Longitude |   *Then let's loop through the data and fill the dataframe one row at a time.*  for data in neighbourhoods\_data:  borough = neighbourhood\_name = data['properties']['borough']  neighbourhood\_name = data['properties']['name']  neighbourhood\_latlon = data['geometry']['coordinates']  neighbourhood\_lat = neighbourhood\_latlon[1]  neighbourhood\_lon = neighbourhood\_latlon[0]    neighbourhoods = neighbourhoods.append({'Borough': borough,  'Neighbourhood': neighbourhood\_name,  'Latitude': neighbourhood\_lat,  'Longitude': neighbourhood\_lon}, ignore\_index=True)  *Quickly examine the resulting dataframe., And make sure that the dataset has all 5 boroughs and 306 neighbourhoods.*  neighbourhoods.head()     |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Borough | Neighbourhood | Latitude | Longitude | | 0 | Bronx | Wakefield | 40.894705 | -73.847201 | | 1 | Bronx | Co-op City | 40.874294 | -73.829939 | | 2 | Bronx | Eastchester | 40.887556 | -73.827806 | | 3 | Bronx | Fieldston | 40.895437 | -73.905643 | | 4 | Bronx | Riverdale | 40.890834 | -73.912585 |   print('The dataframe has {} boroughs and {} neighbourhoods.'.format(  len(neighbourhoods['Borough'].unique()),  neighbourhoods.shape[0]))  The dataframe has 5 boroughs and 306 neighbourhoods.  *Use geopy library to get the latitude and longitude values of New York City.*  *In order to define an instance of the geocoder, we need to define a user\_agent. We will name our agent ny\_explorer, as shown below.*  address = 'New York City, NY'  ​  geolocator = Nominatim(user\_agent="ny\_explorer")  location = geolocator.geocode(address)  latitude = location.latitude  longitude = location.longitude  print('The geographical coordinate of New York City is {}, {}.'.format(latitude, longitude))  The geographical coordinate of New York City is 40.7127281, -74.0060152.  *Create a map of New York with neighbourhoods superimposed on top.*  # create map of New York using latitude and longitude values  map\_newyork = folium.Map(location=[latitude, longitude], zoom\_start=10)  ​  # add markers to map  for lat, lng, borough, neighbourhood in zip(neighbourhoods ['Latitude'], neighbourhoods ['Longitude']  , neighbourhoods ['Borough'], neighbourhoods ['Neighbourhood']):  label = '{}, {}'.format(neighbourhood, borough)  label = folium.Popup(label, parse\_html=True)  folium.CircleMarker(  [lat, lng],  radius=5,  popup=label,  color='blue',  fill=True,  fill\_color='#3186cc',  fill\_opacity=0.7,  parse\_html=False).add\_to(map\_newyork)    map\_newyork  *Folium is a great visualization library. Feel free to zoom into the above map, and click on each circle mark to reveal the name of the neighbourhood and its respective borough.*  *However, for illustration purposes, let's simplify the above map and segment and cluster only the neighbourhoods in Manhattan. So, let's slice the original dataframe and create a new dataframe of the Manhattan data.*  manhattan\_data = neighbourhoods[neighbourhoods['Borough'] == 'Manhattan'].reset\_index(drop=True)  manhattan\_data.head()   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Borough | Neighbourhood | Latitude | Longitude | | 0 | Manhattan | Marble Hill | 40.876551 | -73.910660 | | 1 | Manhattan | Chinatown | 40.715618 | -73.994279 | | 2 | Manhattan | Washington Heights | 40.851903 | -73.936900 | | 3 | Manhattan | Inwood | 40.867684 | -73.921210 | | 4 | Manhattan | Hamilton Heights | 40.823604 | -73.949688 |   *Let's get the geographical coordinates of Manhattan.*  address = 'Manhattan, NY'  ​  geolocator = Nominatim(user\_agent="ny\_explorer")  location = geolocator.geocode(address)  latitude = location.latitude  longitude = location.longitude  print('The geographical coordinate of Manhattan are {}, {}.'.format(latitude, longitude))  The geographical coordinate of Manhattan are 40.7896239, -73.9598939.  As we did with all of New York City, let's visualize Manhattan the neighbourhoods in it.  # create map of Manhattan using latitude and longitude values  map\_manhattan = folium.Map(location=[latitude, longitude], zoom\_start=11)  ​  # add markers to map  for lat, lng, label in zip(manhattan\_data['Latitude'], manhattan\_data['Longitude'], manhattan\_data['Neighbourhood']):  label = folium.Popup(label, parse\_html=True)  folium.CircleMarker(  [lat, lng],  radius=5,  popup=label,  color='blue',  fill=True,  fill\_color='#3186cc',  fill\_opacity=0.7,  parse\_html=False).add\_to(map\_manhattan)    map\_manhattan  *Next, we are going to start utilizing the Foursquare API to explore the neighbourhoods and segment them.*  # Define Foursquare Credentials and Version  CLIENT\_ID = '0XUWQYJ51LOM4MNDEUUOJ1XPHCV13TQ4PIUE4SW1MADEN2U2' # your Foursquare ID  CLIENT\_SECRET = 'F5ZG3S4U0DVS2D0OI1YDYWYM54FBO4X4VXSAXALU2I4KSS0M' # your Foursquare Secret  VERSION = '20200520' # Foursquare API version  ​  print('Your credentails:')  print('CLIENT\_ID: ' + CLIENT\_ID)  print('CLIENT\_SECRET:' + CLIENT\_SECRET)  *Let's explore the first neighbourhood in our dataframe.*  # Get the neighbourhood's name.  manhattan\_data.loc[0, 'Neighbourhood']  'Marble Hill'  # Get the neighbourhood's latitude and longitude values.  neighbourhood\_latitude = manhattan\_data.loc[0, 'Latitude'] # neighbourhood latitude value  neighbourhood\_longitude = manhattan\_data.loc[0, 'Longitude'] # neighbourhood longitude value  ​  neighbourhood\_name = manhattan\_data.loc[0, 'Neighbourhood'] # neighbourhood name  ​  print('Latitude and longitude values of {} are {}, {}.'.format(neighbourhood\_name,  neighbourhood\_latitude,  neighbourhood\_longitude))  Latitude and longitude values of Marble Hill are 40.87655077879964, -73.91065965862981.  *Now, let's get the top 100 venues that are in Marble Hill within a radius of 500 meters.*  # First, let's create the GET request URL.  # define query LIMIT and radius  radius = 500  LIMIT = 100  # define the corresponding URL  url = 'https://api.foursquare.com/v2/venues/explore?client\_id={}&client\_secret={}&ll={},{}&v={}&radius={}&limit={}'.format(  CLIENT\_ID, CLIENT\_SECRET, neighbourhood\_latitude, neighbourhood\_longitude, VERSION, radius, LIMIT)  url  'https://api.foursquare.com/v2/venues/explore?client\_id=0XUWQYJ51LOM4MNDEUUOJ1XPHCV13TQ4PIUE4SW1MADEN2U2&client\_secret=F5ZG3S4U0DVS2D0OI1YDYWYM54FBO4X4VXSAXALU2I4KSS0M&ll=40.87655077879964,-73.91065965862981&v=20200520&radius=500&limit=100'  # Send the GET request and examine the results  results = requests.get(url).json()  results['response']['groups'][0]['items']#[0]  *From the Foursquare lab in the previous module, we know that all the information is in the items key. Before we proceed, let's borrow the get\_category\_type function from the Foursquare lab.*  # function that extracts the category of the venue  def get\_category\_type(row):  try:  categories\_list = row['categories']  except:  categories\_list = row['venue.categories']    if len(categories\_list) == 0:  return None  else:  return categories\_list[0]['name']  *Now we are ready to clean the json and structure it into a pandas dataframe.*  venues = results['response']['groups'][0]['items']  nearby\_venues = json\_normalize(venues) # flatten JSON  ​  # filter columns  filtered\_columns = ['venue.name', 'venue.categories', 'venue.location.lat', 'venue.location.lng']  nearby\_venues =nearby\_venues.loc[:, filtered\_columns]  ​  # filter the category for each row  nearby\_venues['venue.categories'] = nearby\_venues.apply(get\_category\_type, axis=1)  ​  # clean columns  nearby\_venues.columns = [col.split(".")[-1] for col in nearby\_venues.columns]  nearby\_venues.head()   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | name | categories | lat | lng | | 0 | Arturo's | Pizza Place | 40.874412 | -73.910271 | | 1 | Bikram Yoga | Yoga Studio | 40.876844 | -73.906204 | | 2 | Tibbett Diner | Diner | 40.880404 | -73.908937 | | 3 | Starbucks | Coffee Shop | 40.877531 | -73.905582 | | 4 | Dunkin' | Donut Shop | 40.877136 | -73.9066 |   *And how many venues were returned by Foursquare?*  print('{} venues were returned by Foursquare.'.format(nearby\_venues.shape[0]))  26 venues were returned by Foursquare.  2. Explore Neighbourhoods in Manhattan  *Let's create a function to repeat the same process to all the neighbourhoods in Manhattan*  def getNearbyVenues(names, latitudes, longitudes, radius=500):    venues\_list=[]  for name, lat, lng in zip(names, latitudes, longitudes):  print(name)    # create the API request URL  url = 'https://api.foursquare.com/v2/venues/explore?&client\_id={}&client\_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(  CLIENT\_ID, CLIENT\_SECRET, VERSION, lat, lng, radius, LIMIT)    # make the GET request  results = requests.get(url).json()["response"]['groups'][0]['items']    # return only relevant information for each nearby venue  venues\_list.append([  (name, lat, lng, v['venue']['name'], v['venue']['location']['lat'], v['venue']['location']['lng'], v['venue']['categories'][0]['name'])  for v in results])  ​ nearby\_venues = pd.DataFrame([item for venue\_list in venues\_list for item in venue\_list])  nearby\_venues.columns =  ['Neighbourhood', 'Neighbourhood Latitude', 'Neighbourhood Longitude', 'Venue', 'Venue Latitude', 'Venue Longitude', 'Venue Category']    return(nearby\_venues)  *Now write the code to run the above function on each neighbourhood and create a new dataframe called manhattan\_venues.*  # type your answer here  ​  manhattan\_venues = getNearbyVenues(names=manhattan\_data['Neighbourhood'],  latitudes=manhattan\_data['Latitude'],  longitudes=manhattan\_data['Longitude'])  ​  *Let's check the size of the resulting dataframe*  print(manhattan\_venues.shape)  manhattan\_venues.head()  (3071, 7)   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Neighbourhood | Neig.. Latitude | Neig.. Longitude | Venue | Venue Latitude | Venue Longitude | Venue Category | | 0 | Marble Hill | 40.876551 | -73.91066 | Arturo's | 40.874412 | -73.910271 | | 1 | Marble Hill | 40.876551 | -73.91066 | Bikram Yoga | 40.876844 | -73.906204 | | 2 | Marble Hill | 40.876551 | -73.91066 | Tibbett Diner | 40.880404 | -73.908937 | | 3 | Marble Hill | 40.876551 | -73.91066 | Starbucks | 40.877531 | -73.905582 | | 4 | Marble Hill | 40.876551 | -73.91066 | Dunkin' | 40.877136 | -73.906666 |   *Let's check how many venues were returned for each neighbourhood*  manhattan\_venues.groupby('Neighbourhood').count()  *Let's find out how many unique categories can be curated from all the returned venues*  print('There are {} unique categories.'.format(len(manhattan\_venues['Venue Category'].unique())))  There are 329 unique categories.  3. Analyse Each Neighbourhood  # one hot encoding  manhattan\_onehot = pd.get\_dummies(manhattan\_venues[['Venue Category']], prefix="", prefix\_sep="")  ​  # add neighbourhood column back to dataframe  manhattan\_onehot['Neighbourhood'] = manhattan\_venues['Neighbourhood']  ​  # move neighbourhood column to the first column  fixed\_columns = [manhattan\_onehot.columns[-1]] + list(manhattan\_onehot.columns[:-1])  manhattan\_onehot = manhattan\_onehot[fixed\_columns]  ​  manhattan\_onehot.head()  #manhattan\_onehot.head()[list(manhattan\_onehot.columns[:15])+ [manhattan\_onehot.columns[-1]]]   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Neighborhood | Accessories Store | Adult Boutique | Afghan Restaurant | African Restaurant | American Restaurant | Antique Shop | … | Arepa Restaurant | Argentinian Restaurant | Art Gallery | Art Museum | Arts & Crafts Store | Asian Restaurant | Athletics & Sports | Yoga Studio | | 0 | Marble Hill | 0 | 0 | 0 | 0 | 0 | 0 | … | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | Marble Hill | 0 | 0 | 0 | 0 | 0 | 0 | … | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | 2 | Marble Hill | 0 | 0 | 0 | 0 | 0 | 0 | … | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 3 | Marble Hill | 0 | 0 | 0 | 0 | 0 | 0 | … | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 4 | Marble Hill | 0 | 0 | 0 | 0 | 0 | 0 | … | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   *And let's examine the new dataframe size.*  manhattan\_onehot.shape  (3071, 330)  *Next, let's group rows by neighbourhood and by taking the mean of the frequency of occurrence of each category*  manhattan\_grouped = manhattan\_onehot.groupby('Neighbourhood').mean().reset\_index()  manhattan\_grouped ##.head()[manhattan\_grouped.columns[0:10]]   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Neighbourhood | Accessories Store | Adult Boutique | Afghan Restaurant | African Restaurant | … | American Restaurant | Antique Shop | Arcade | Arepa Restaurant | Argentinian Restaurant | | 0 | Battery Park City | 0.0 | 0.0 | 0.0 | 0.000000 | … | 0.000000 | 0.0 | 0.0 | 0.0 | 0.000000 | | 1 | Carnegie Hill | 0.0 | 0.0 | 0.0 | 0.000000 | … | 0.012195 | 0.0 | 0.0 | 0.0 | 0.012195 | | 2 | Central Harlem | 0.0 | 0.0 | 0.0 | 0.068182 | … | 0.045455 | 0.0 | 0.0 | 0.0 | 0.000000 | | 3 | Chelsea | 0.0 | 0.0 | 0.0 | 0.000000 | … | 0.030000 | 0.0 | 0.0 | 0.0 | 0.000000 | | 4 | Chinatown | 0.0 | 0.0 | 0.0 | 0.000000 | … | 0.030000 | 0.0 | 0.0 | 0.0 | 0.000000 |   # Let's confirm the new size  manhattan\_grouped.shape  (40, 330)  # Let's print each neighbourhood along with the top 5 most common venues  num\_top\_venues = 5  ​  for hood in manhattan\_grouped['Neighbourhood']:  print("----"+hood+"----")  temp = manhattan\_grouped[manhattan\_grouped['Neighbourhood'] == hood].T.reset\_index()  temp.columns = ['venue','freq']  temp = temp.iloc[1:]  temp['freq'] = temp['freq'].astype(float)  temp = temp.round({'freq': 2})  print(temp.sort\_values('freq', ascending=False).reset\_index(drop=True).head(num\_top\_venues))  print('\n')  ----Battery Park City----  venue freq  0 Park 0.11  1 Coffee Shop 0.08  …  4 Wine Shop 0.05  ----Carnegie Hill----  venue freq  …  # Let's put that into a pandas dataframe  *First, let's write a function to sort the venues in descending order.*  def return\_most\_common\_venues(row, num\_top\_venues):  row\_categories = row.iloc[1:]  row\_categories\_sorted = row\_categories.sort\_values(ascending=False)    return row\_categories\_sorted.index.values[0:num\_top\_venues]  *Now let's create the new dataframe and display the top 10 venues for each neighbourhood.*  manhattan\_grouped.iloc[0, :][1:].sort\_values(ascending=False)  num\_top\_venues = 10  ​  indicators = ['st', 'nd', 'rd']  ​  # create columns according to number of top venues  columns = ['Neighbourhood']  for ind in np.arange(num\_top\_venues):  try:  columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))  except:  columns.append('{}th Most Common Venue'.format(ind+1))  ​  # create a new dataframe  neighbourhoods\_venues\_sorted = pd.DataFrame(columns=columns)  neighbourhoods\_venues\_sorted['Neighbourhood'] = manhattan\_grouped['Neighbourhood']  ​  for ind in np.arange(manhattan\_grouped.shape[0]):  neighbourhoods\_venues\_sorted.iloc[ind, 1:] = return\_most\_common\_venues(manhattan\_grouped.iloc[ind, :], num\_top\_venues)  ​  neighbourhoods\_venues\_sorted.head()   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Neighbour-hood | 1st Most Common Venue | 2nd Most Common Venue | 3rd Most Common Venue | 4th Most Common Venue | 5th Most Common Venue | 6th Most Common Venue | 7th Most Common Venue | 8th Most Common Venue | 9th Most Common Venue | | 0 | Battery Park City | Park | Coffee Shop | Hotel | Wine Shop | Memorial Site | Gym | Shopping Mall | Plaza | Playground | | 1 | Carnegie Hill | Coffee Shop | Pizza Place | Café | Japanese Restaurant | Gym | Grocery Store | Yoga Studio | Bookstore | Bar | | 2 | Central Harlem | Chinese Restaurant | African Restaurant | Bar | American Restaurant | Cosmetics Shop | French Restaurant | Seafood Restaurant | Dessert Shop | Boutique | | 3 | Chelsea | Art Gallery | Coffee Shop | Café | Ice Cream Shop | American Restaurant | Market | Seafood Restaurant | Boutique | Cupcake Shop | | 4 | Chinatown | Chinese Restaurant | Bakery | Bubble Tea Shop | Cocktail Bar | Spa | Coffee Shop | American Restaurant | Optical Shop | Salon / Barbershop |   4. Cluster Neighbourhoods  *Run k-means to cluster the neighbourhood into 5 clusters*.  # set number of clusters  kclusters = 5  ​  manhattan\_grouped\_clustering = manhattan\_grouped.drop('Neighbourhood', 1)  ​  # run k-means clustering  kmeans = KMeans(n\_clusters=kclusters, random\_state=0).fit(manhattan\_grouped\_clustering)  ​  # check cluster labels generated for each row in the dataframe  kmeans.labels\_[0:10]  *Let's create a new dataframe that includes the cluster as well as the top 10 venues for each neighbourhood.*  # add clustering labels  neighbourhoods\_venues\_sorted.insert(0, 'Cluster Labels', kmeans.labels\_)  ​  manhattan\_merged = manhattan\_data  ​  # merge toronto\_grouped with toronto\_data to add latitude/longitude for each neighbourhood  manhattan\_merged = manhattan\_merged.join(neighbourhoods\_venues\_sorted.set\_index('Neighbourhood'), on='Neighbourhood')  ​  manhattan\_merged.head() # check the last columns!   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Borough | Neighbo-urhood | Latitude | Longitude | Cluster Labels | 1st Most Common Venue | 2nd Most Common Venue | 3rd Most Common Venue | 4th Most Common Venue | 5th Most Common Venue | 6th Most Common Venue | 7th Most Common Venue | 8th Most Common Venue | 9th Most Common Venue | 10th Most Common Venue | | 0 | Manhattan | Marble Hill | 40.876551 | -73.910660 | 1 | Sandwich Place | Gym | Coffee Shop | Yoga Studio | Deli / Bodega | Supplement Shop | Steakhouse | Shopping Mall | Seafood Restaurant | Pizza Place |   *Finally, let's visualize the resulting clusters*  # create map  map\_clusters = folium.Map(location=[latitude, longitude], zoom\_start=11)  ​  # set color scheme for the clusters  x = np.arange(kclusters)  ys = [i + x + (i\*x)\*\*2 for i in range(kclusters)]  colors\_array = cm.rainbow(np.linspace(0, 1, len(ys)))  rainbow = [colors.rgb2hex(i) for i in colors\_array]  ​  # add markers to the map  markers\_colors = []  for lat, lon, poi, cluster in zip(manhattan\_merged['Latitude'], manhattan\_merged['Longitude']  , manhattan\_merged['Neighbourhood'], manhattan\_merged['Cluster Labels']):  label = folium.Popup(str(poi) + ' Cluster ' + str(cluster), parse\_html=True)  folium.CircleMarker(  [lat, lon],  radius=5,  popup=label,  color=rainbow[cluster-1],  fill=True,  fill\_color=rainbow[cluster-1],  fill\_opacity=0.7).add\_to(map\_clusters)    map\_clusters  5. Examine Clusters  *Now, you can examine each cluster and determine the discriminating venue categories that distinguish each cluster. Based on the defining categories, you can then assign a name to each cluster.*  *I will leave this exercise to you.*  # Cluster 1  manhattan\_merged.loc[manhattan\_merged['Cluster Labels'] == 0,  manhattan\_merged.columns[[1] + list(range(5, manhattan\_merged.shape[1]))]]   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Neighbour-hood | 1st Most Common Venue | 2nd Most Common Venue | 3rd Most Common Venue | 4th Most Common Venue | 5th Most Common Venue | 6th Most Common Venue | 7th Most Common Venue | 8th Most Common Venue | 9th Most Common Venue | 10th Most Common Venue | | 2 | Washington Heights | Café | Bakery | Deli / Bodega | Mobile Phone Shop | Pizza Place | Grocery Store | Chinese \*\* | Latin American \*\* | Mexican \*\* | Park | | 3 | Inwood | Mexican \*\* | Café | Lounge | Pizza Place | \*\* | Park | Chinese \*\* | Spanish \*\* | Frozen Yogurt Shop | Bakery | | 4 | Hamilton Heights | Pizza Place | Coffee Shop | Café | Mexican \*\* | Deli / Bodega | Cocktail Bar | Sushi \*\* | Park | Yoga Studio | Chinese \*\* | | 5 | Manhattanville | Coffee Shop | Seafood \*\* | Italian \*\* | Mexican \*\* | Sushi \*\* | Park | Deli / Bodega | Food & Drink Shop | Music School | Japanese Curry \*\* | | 7 | East Harlem | Mexican \*\* | Bakery | Thai \*\* | Latin American \*\* | Deli / Bodega | Gas Station | Gym | Grocery Store | Cocktail Bar | Beer Bar | | 25 | Manhattan Valley | Coffee Shop | Pizza Place | Bar | Mexican \*\* | Yoga Studio | Grocery Store | Playground | Park | Latin American \*\* | Korean \*\* | | 36 | Tudor City | Park | Café | Mexican \*\* | Deli / Bodega | Sushi \*\* | Thai \*\* | Coffee Shop | Gym / Fitness Center | Dog Run | Asian \*\* |   \*\* Restaurant  # Cluster 2  manhattan\_merged.loc[manhattan\_merged['Cluster Labels'] == 1,  manhattan\_merged.columns[[1] + list(range(5, manhattan\_merged.shape[1]))]]   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Neighbour-hood | 1st Most Common Venue | 2nd Most Common Venue | 3rd Most Common Venue | 4th Most Common Venue | 5th Most Common Venue | 6th Most Common Venue | 7th Most Common Venue | 8th Most Common Venue | 9th Most Common Venue | 10th Most Common Venue | | 0 | Marble Hill | Sandwich Place | Gym | Coffee Shop | Yoga Studio | Deli / Bodega | Supplement Shop | Steakhouse | Shopping Mall | Seafood \*\* | Pizza Place | | 8 | Upper East Side | Italian \*\* | Bakery | Gym / Fitness Center | Spa | Coffee Shop | Exhibit | Hotel | Juice Bar | Yoga Studio | Wine Shop | | 9 | Yorkville | Coffee Shop | Italian \*\* | Gym | Sushi \*\* | Bar | Deli / Bodega | Diner | Japanese \*\* | Mexican \*\* | Wine Shop | | 10 | Lenox Hill | Coffee Shop | Italian \*\* | Pizza Place | Café | Cocktail Bar | Sushi \*\* | Gym | Gym / Fitness Center | Burger Joint | Thai \*\* | | 11 | Roosevelt Island | Park | Cosmetics Shop | Plaza | Farmers Market | Soccer Field | Supermarket | School | Scenic Lookout | Outdoors & Recreation | Liquor Store | | 14 | Clinton | Theater | Gym / Fitness Center | Coffee Shop | Hotel | Gym | Italian \*\* | Sandwich Place | Pizza Place | Spa | American \*\* | | 15 | Midtown | Coffee Shop | Bakery | Theater | Hotel | Clothing Store | Japanese \*\* | Pizza Place | Cuban \*\* | Bookstore | Sporting Goods Shop | | 16 | Murray Hill | Coffee Shop | Sandwich Place | Japanese \*\* | Hotel | Pizza Place | Gym / Fitness Center | Grocery Store | Mediterranean \*\* | Juice Bar | Indian \*\* | | 27 | Gramercy | Bagel Shop | Italian \*\* | Bar | Pizza Place | Coffee Shop | Playground | Mexican \*\* | Thai \*\* | Grocery Store | American \*\* | | 28 | Battery Park City | Park | Coffee Shop | Hotel | Wine Shop | Memorial Site | Gym | Shopping Mall | Plaza | Playground | Food Court | | 29 | Financial District | Coffee Shop | Hotel | American \*\* | Pizza Place | Café | Park | Falafel \*\* | Cocktail Bar | Steakhouse | Sandwich Place | | 30 | Carnegie Hill | Coffee Shop | Pizza Place | Café | Japanese \*\* | Gym | Grocery Store | Yoga Studio | Bookstore | Bar | Wine Shop | | 31 | Noho | Coffee Shop | Pizza Place | Italian \*\* | Japanese \*\* | Grocery Store | Sandwich Place | Mexican \*\* | Wine Shop | Greek \*\* | Taco Place | | 32 | Civic Center | Coffee Shop | American \*\* | French \*\* | Hotel | Cocktail Bar | Park | Spa | Gym / Fitness Center | Yoga Studio | Sushi \*\* | | 33 | Midtown South | Korean \*\* | Hotel | Burger Joint | Japanese \*\* | Café | Dessert Shop | Hotel Bar | Gym / Fitness Center | Coffee Shop | New American \*\* | | 34 | Sutton Place | Coffee Shop | Italian \*\* | Gym / Fitness Center | Gym | Park | Bagel Shop | Hotel | Pizza Place | Grocery Store | Smoke Shop | | 35 | Turtle Bay | Coffee Shop | Italian \*\* | Park | Café | Deli / Bodega | French \*\* | Seafood \*\* | Sushi \*\* | Karaoke Bar | Garden | | 38 | Flatiron | Gym / Fitness Center | Italian \*\* | Gym | Park | Outdoor Sculpture | Mediterranean \*\* | Juice Bar | Japanese \*\* | Salon / Barbershop | Café | | 39 | Hudson Yards | Gym / Fitness Center | Italian \*\* | American \*\* | Café | Hotel | Coffee Shop | Park | Gym | Dog Run | \*\* |   \*\* Restaurant  # Cluster 3  manhattan\_merged.loc[manhattan\_merged['Cluster Labels'] == 2,  manhattan\_merged.columns[[1] + list(range(5, manhattan\_merged.shape[1]))]]   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Neigh | 1st | 2nd | 3rd | 4th | 5th | 6th | 7th | 8th | 9th | 10th | | 1 | Chinatown | Chinese \*\* | Bakery | Bubble Tea Sh | Cocktail Bar | Spa | Coffee Sh | American \*\* | Optical Sh | Salon | Mexican \*\* | | 6 | Central Harlem | Chinese \*\* | African \*\* | Bar | American \*\* | Cosmetics Sh | French \*\* | Seafood \*\* | Dessert Sh | Boutique | Gym / Fitness | | 12 | Upper West Side | Italian \*\* | Dessert Sh | Wine Bar | Bar | Coffee Sh | Indian \*\* | Mexican \*\* | Middle Eastern \*\* | Pizza Place | Sushi \*\* | | 13 | Lincoln Square | Italian \*\* | Plaza | Café | Concert Hall | Theater | Performing Arts | American \*\* | Wine Sh | French \*\* | Gym / Fitness | | 17 | Chelsea | Art Gallery | Coffee Sh | Café | Ice Cream Sh | American \*\* | Market | Seafood \*\* | Boutique | Cupcake Sh | Cycle Studio | | 18 | Greenwich Village | Italian \*\* | Café | Pizza Place | Gym | Sushi \*\* | Coffee Sh | Ice Cream Sh | Bakery | Pilates Studio | Sandwich Place | | 19 | East Village | Cocktail Bar | Pizza Place | Mexican \*\* | Bar | Coffee Sh | Wine Bar | Juice Bar | Ice Cream Sh | Japanese \*\* | Ramen \*\* | | 20 | Lower East Side | Chinese \*\* | Café | Pharmacy | Art Gallery | Bakery | Park | Cocktail Bar | Mediterranean \*\* | Women's Store | Japanese \*\* | | 21 | Tribeca | Italian \*\* | Park | American \*\* | Bakery | Café | Spa | Wine Bar | Skate Park | Scenic Lookout | Hotel | | 22 | Little Italy | Chinese \*\* | Bubble Tea Sh | Café | Spa | Bakery | Mediterranean \*\* | Hotel | Thai \*\* | Pizza Place | Ice Cream Sh | | 23 | Soho | Italian \*\* | Mediterranean \*\* | Café | Sandwich Place | Gym | Coffee Sh | Sushi \*\* | Ice Cream Sh | French \*\* | Clothing Store | | 24 | West Village | Italian \*\* | Wine Bar | American \*\* | Jazz Club | Pizza Place | New American \*\* | Coffee Sh | Bakery | Park | Cocktail Bar |   \*\* Restaurant, Sh. = Shop  # Cluster 4  manhattan\_merged.loc[manhattan\_merged['Cluster Labels'] == 3,  manhattan\_merged.columns[[1] + list(range(5, manhattan\_merged.shape[1]))]]   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Neighbour-hood | 1st Most Common Venue | 2nd Most Common Venue | 3rd Most Common Venue | 4th Most Common Venue | 5th Most Common Venue | 6th Most Common Venue | 7th Most Common Venue | 8th Most Common Venue | 9th Most Common Venue | 10th Most Common Venue | | 37 | Stuyvesant Town | Playground | Park | Bar | Pet Service | Gas Station | Boat or Ferry | Farmers Market | Bistro | Gym / Fitness Center | Baseball Field |   # Cluster 5  manhattan\_merged.loc[manhattan\_merged['Cluster Labels'] == 4,  manhattan\_merged.columns[[1] + list(range(5, manhattan\_merged.shape[1]))]]   |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | Neighbour-hood | 1st Most Common Venue | 2nd Most Common Venue | 3rd Most Common Venue | 4th Most Common Venue | 5th Most Common Venue | 6th Most Common Venue | 7th Most Common Venue | 8th Most Common Venue | 9th Most Common Venue | 10th Most Common Venue | | 26 | Morningside Heights | Park | Coffee Shop | American Restaurant | Bookstore | Pizza Place | Deli / Bodega | Burger Joint | Tennis Court | Sandwich Place | New American Restaurant |   *Thank you for completing this lab!* |

Final Assignment: Applied Data Science Capstone

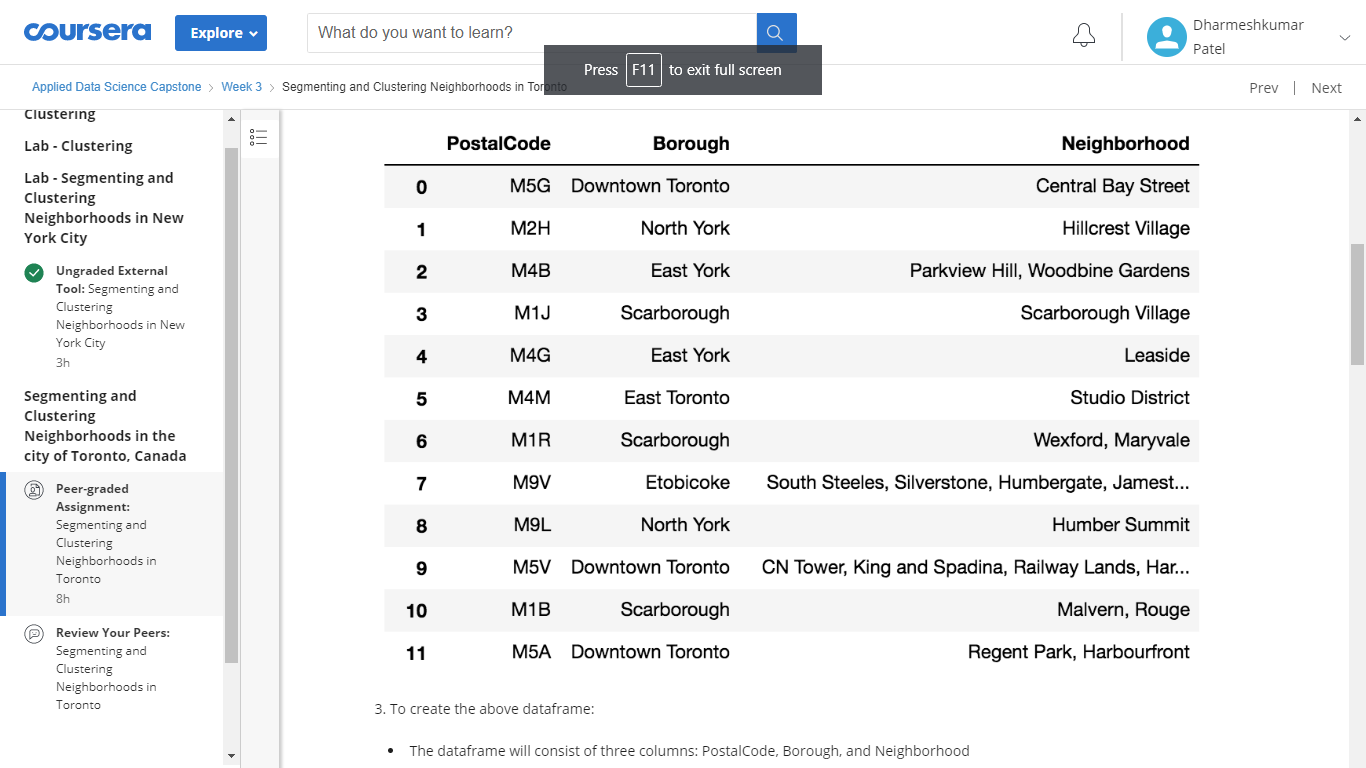
## Instructions

You can prepare one notebook for all three parts, but please use Markdown to clearly label your work for each part in order to make it easy for your peers to grade your work. However, you will have to submit the notebook three times since a submission has to be associated with each question. Sorry about that.

Project Title : \*Give your project a descriptive title

## Part I

For this assignment, you will be required to explore and cluster the neighbourhoods in Toronto.

1. Start by creating a new Notebook for this assignment.
2. Use the Notebook to build the code to scrape the following Wikipedia page, <https://en.wikipedia.org/wiki/List_of_postal_codes_of_Canada:_M> , in order to obtain the data that is in the table of postal codes and to transform the data into a pandas dataframe like the one shown below:
3. To create the above dataframe:
   * + The dataframe will consist of three columns: PostalCode, Borough, and Neighbourhood
     + Only process the cells that have an assigned borough. Ignore cells with a borough that is Not assigned.
     + More than one neighbourhood can exist in one postal code area. For example, in the table on the Wikipedia page, you will notice that M5A is listed twice and has two neighbourhoods: Harbourfront and Regent Park. These two rows will be combined into one row with the neighbourhoods separated with a comma as shown in row 11 in the above table.
     + If a cell has a borough but a Not assigned neighbourhood, then the neighbourhood will be the same as the borough.
     + Clean your Notebook and add Markdown cells to explain your work and any assumptions you are making.
     + In the last cell of your notebook, use the shape method to print the number of rows of your dataframe.
4. Submit a link to your Notebook on your Github repository. (10 marks)

Note: There are different website scraping libraries and packages in Python. For scraping the above table, you can simply use pandas to read the table into a pandas dataframe.

Another way, which would help to learn for more complicated cases of web scraping is using the BeautifulSoup package. Here is the package's main documentation page: <http://beautiful-soup-4.readthedocs.io/en/latest/>

Use pandas, or the BeautifulSoup package, or any other way you are comfortable with to transform the data in the table on the Wikipedia page into the above pandas dataframe.

**Submit Notebook link on Github repository:**

## Part II

Now that you have built a dataframe of the postal code of each neighborhood along with the borough name and neighborhood name, in order to utilize the Foursquare location data, we need to get the latitude and the longitude coordinates of each neighborhood.

In an older version of this course, we were leveraging the Google Maps Geocoding API to get the latitude and the longitude coordinates of each neighborhood. However, recently Google started charging for their API: <http://geoawesomeness.com/developers-up-in-arms-over-google-maps-api-insane-price-hike/>, so we will use the Geocoder Python package instead: https://geocoder.readthedocs.io/index.html.

The problem with this Package is you have to be persistent sometimes in order to get the geographical coordinates of a given postal code. So you can make a call to get the latitude and longitude coordinates of a given postal code and the result would be None, and then make the call again and you would get the coordinates. So, in order to make sure that you get the coordinates for all of our neighborhoods, you can run a while loop for each postal code. Taking postal code M5G as an example, your code would look something like this:

import geocoder # import geocoder

# initialize your variable to None

lat\_lng\_coords = None

# loop until you get the coordinates

while(lat\_lng\_coords is None):

g = geocoder.google('{}, Toronto, Ontario'.format(postal\_code))

lat\_lng\_coords = g.latlng

latitude = lat\_lng\_coords[0]

longitude = lat\_lng\_coords[1]

Given that this package can be very unreliable, in case you are not able to get the geographical coordinates of the neighborhoods using the Geocoder package, here is a link to a csv file that has the geographical coordinates of each postal code: <http://cocl.us/Geospatial_data>

Use the Geocoder package or the csv file to create the following dataframe:

Important Note: There is a limit on how many times you can call geocoder.google function. It is 2500 times per day. This should be way more than enough for you to get acquainted with the package and to use it to get the geographical coordinates of the neighborhoods in the Toronto.

Once you are able to create the above dataframe, submit a link to the new Notebook on your Github repository. (2 marks)

**Submit Notebook link on Github repository:**

## Part III

Explore and cluster the neighborhoods in Toronto. You can decide to work with only boroughs that contain the word Toronto and then replicate the same analysis we did to the New York City data. It is up to you.

Just make sure:

1. to add enough Markdown cells to explain what you decided to do and to report any observations you make.
2. to generate maps to visualize your neighborhoods and how they cluster together.

Once you are happy with your analysis, submit a link to the new Notebook on your Github repository. (3 marks)