Najmin Ahmed, Mai Nakhala, Virginia Pollock

Professor Barbara Ericson

SI 206

Github repo: <https://github.com/pollockv/SI206-API-Project>

API Till We Die Final Report

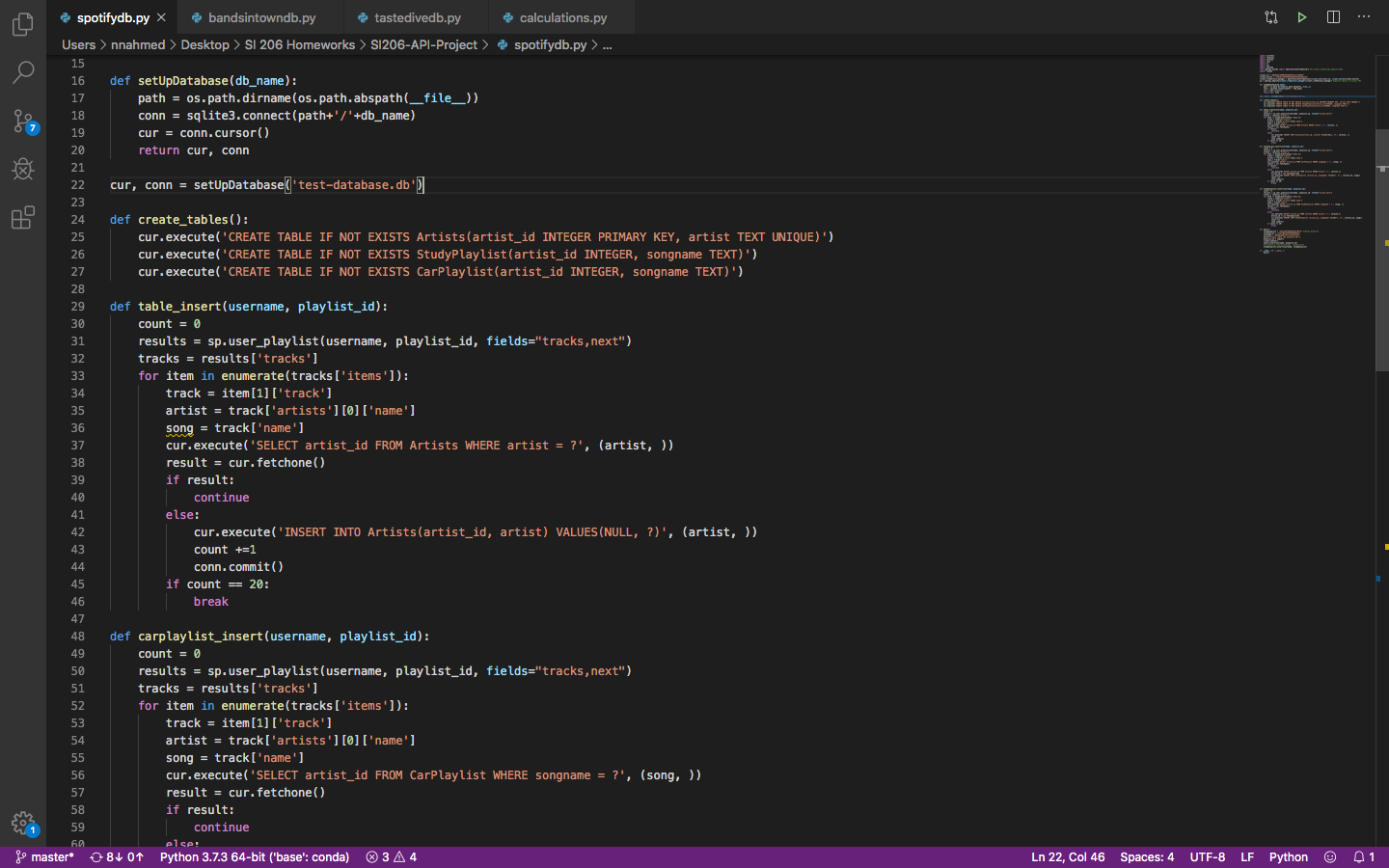
Original and Achieved Goals

The goal of our project was to find new artists based on what we all currently listen to and find upcoming events or concerts from those recommendations. We created two different playlists on our Spotify account, a study playlist and a car playlist. Those playlists were full of our own favorite music that we like listening to while in our favorite study spot or on the road. We wanted to do that by finding genre information, most listened to information, etc. We were able to pull most of this information, but ran into some trouble collecting everything.

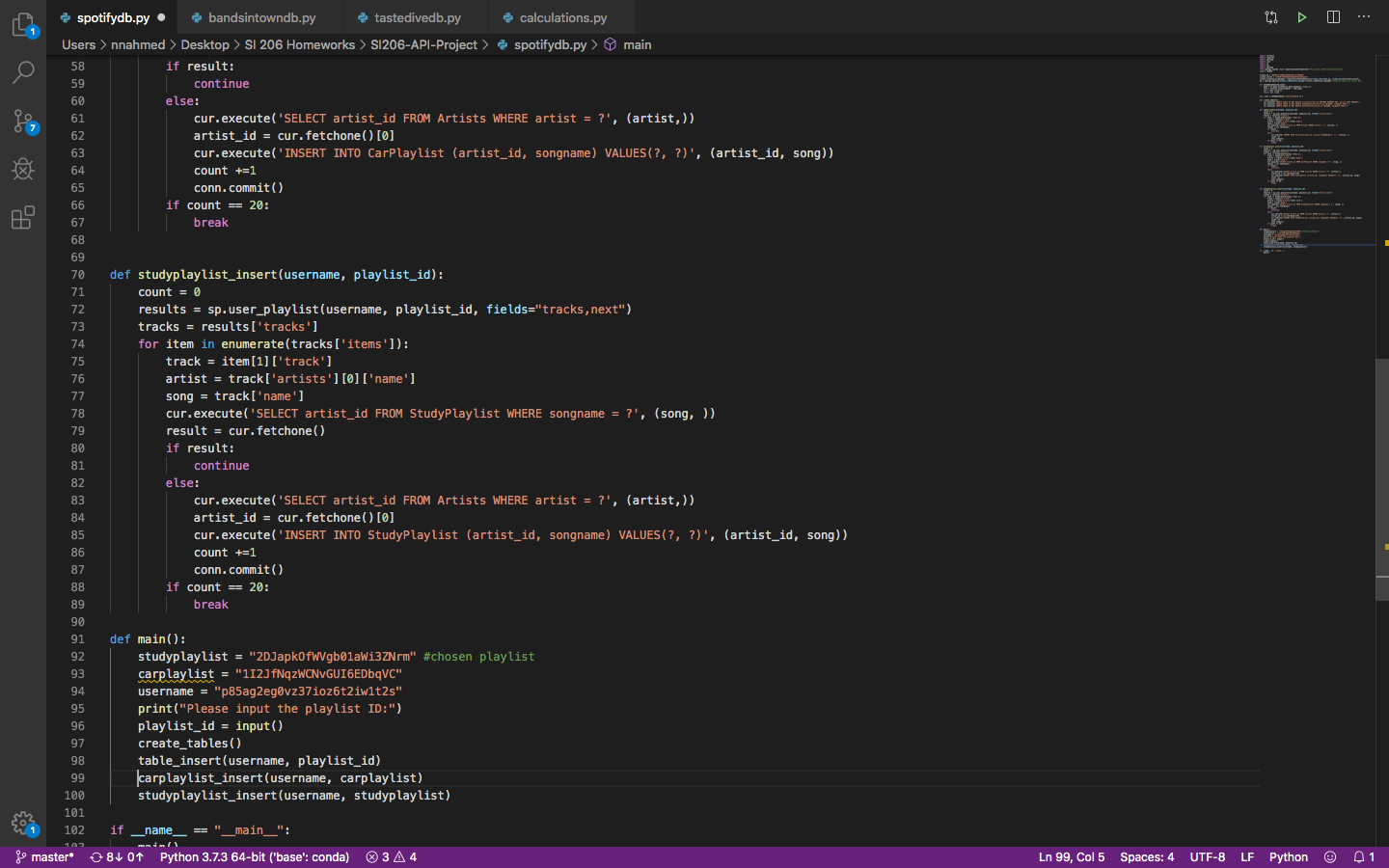
Problems We Faced

While we were able to achieve our original goals, we had to go about doing it a different way than planned. Spotify’s API did not have the option to pull genre information from various songs, which is something we really wanted to include in our database to base our calculations and visualizations off of. Some other problems we faced was finding enough unique characteristics of the songs, since we could not find genre, to include in our database along with figuring out how to use the Spotipy documentation to utilize the Spotify API. Our biggest problem was trying to comprehend the rate limit and how to write code for that in our project, but that was solved by going to office hours and receiving help for that. What we ended up doing instead was using artist id, artist name, and song name throughout our project to then use towards the TasteDive and BandsInTown APIs. We decided to focus on TasteDive and BandsInTown results using the information we gained from Spotify’s playlists.

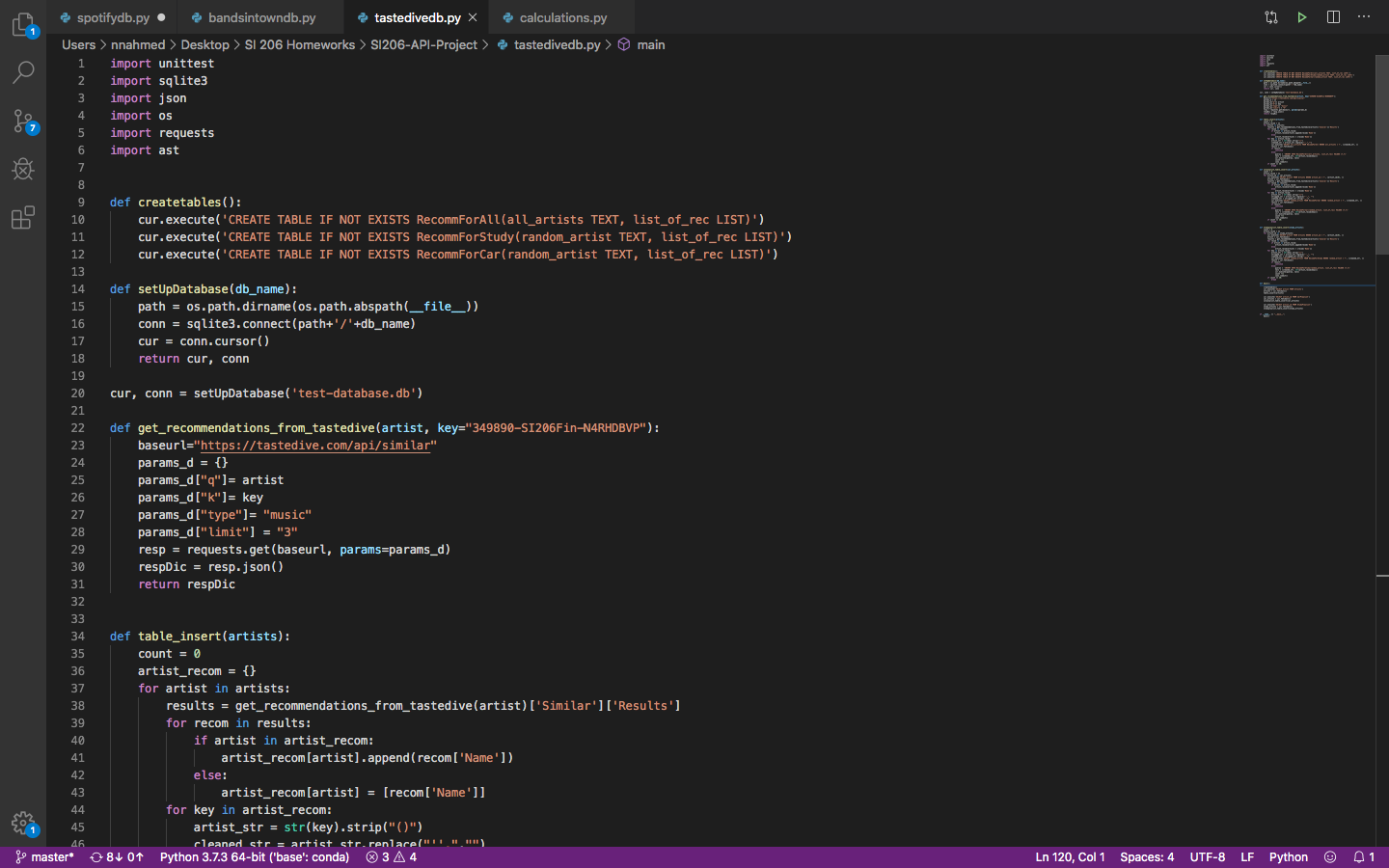
Functions to Create Database:



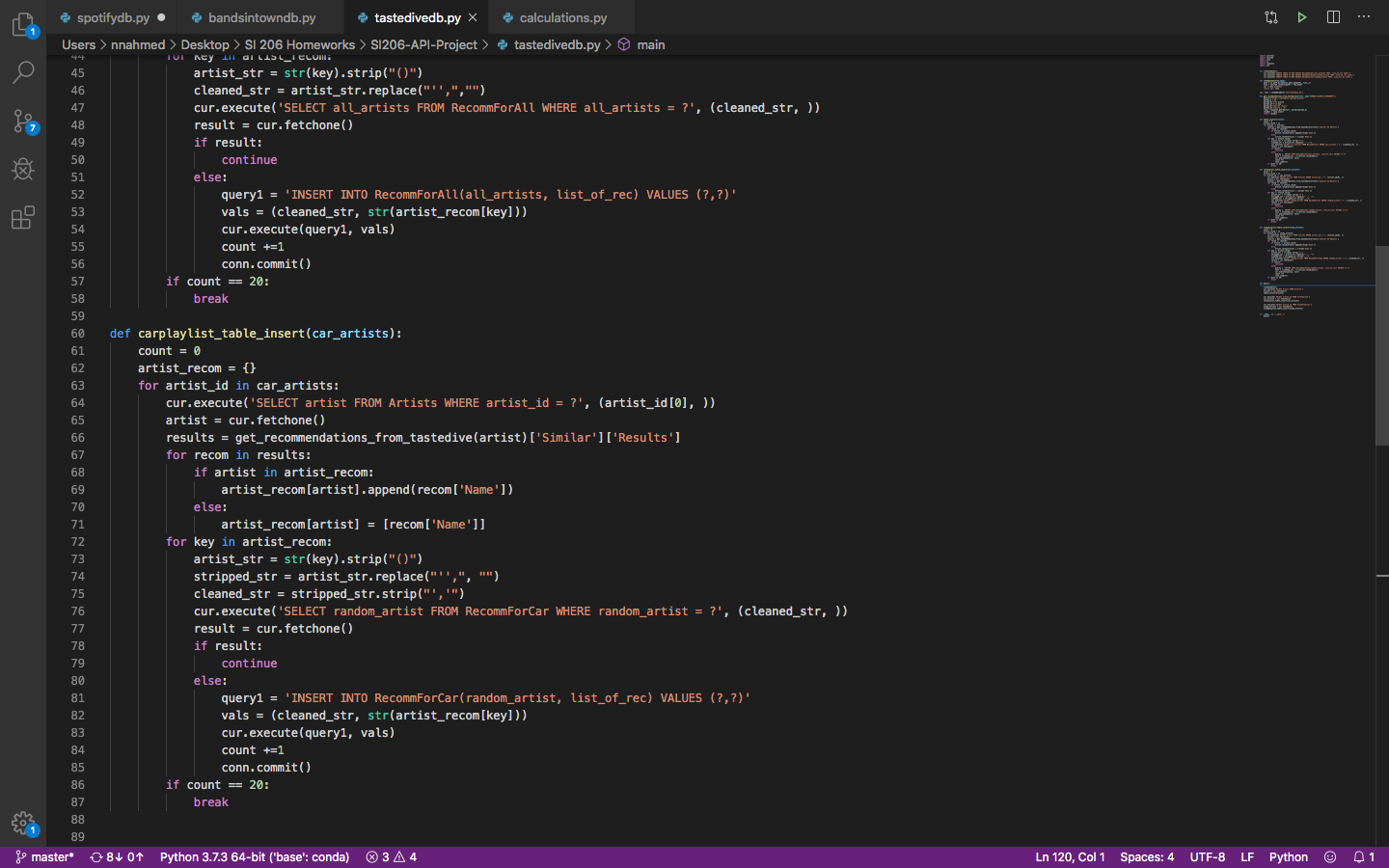
setUpDatabase function creates our database and creates the cur and conn objects for us to use. Create\_tables function creates the tables we are going to use with data pulled from Spotify API, if they haven’t been created already. table\_insert function creates the Artists table where we insert all artists from our two playlists, with an artist\_id and without duplicates while running 20 at a time. Carplaylist\_insert function inserts artist\_id taken from Artists table and the specific song name from our car playlist from Spotify account, without duplicates while running 20 at a time.



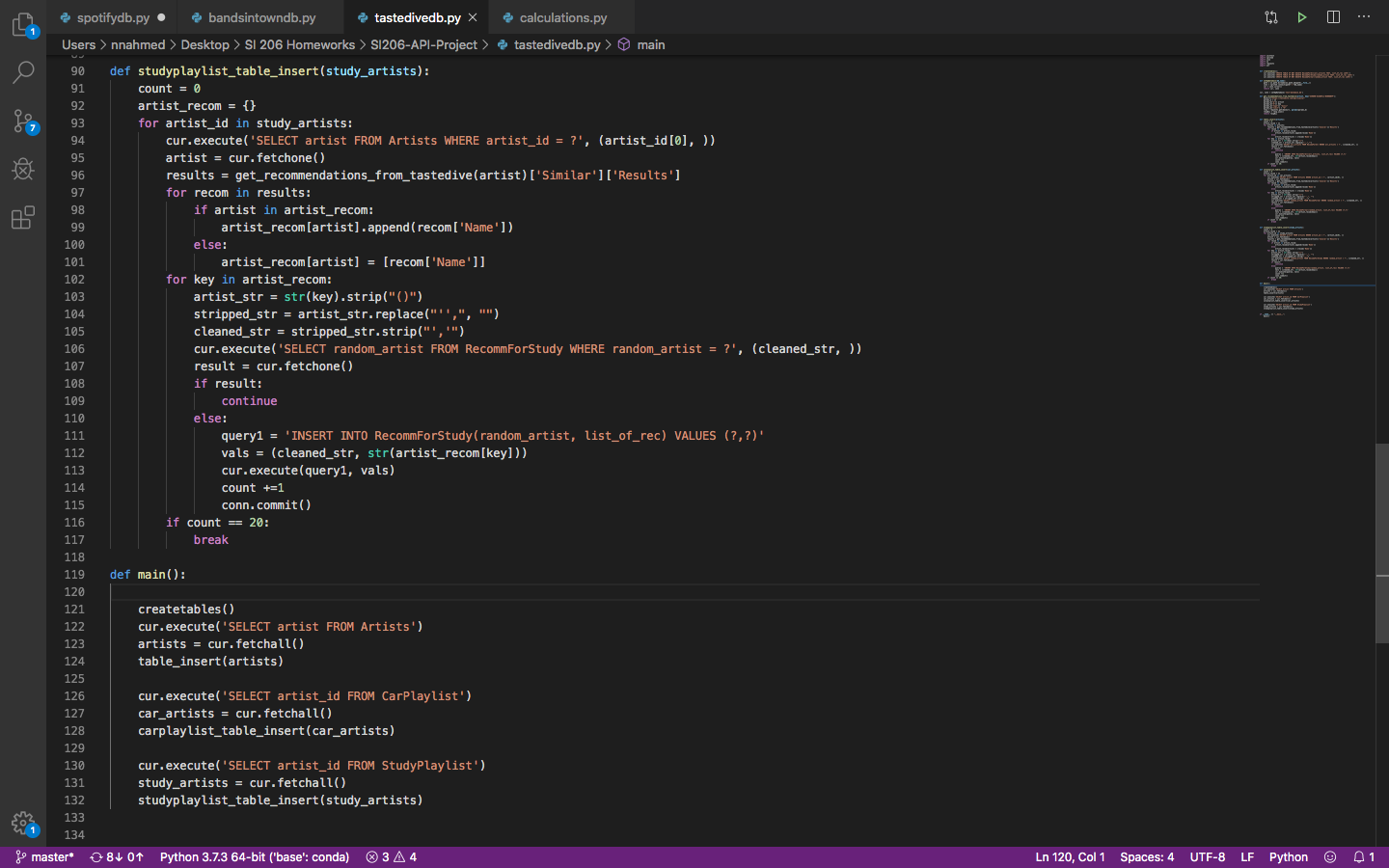
Studyplaylist\_insert function inserts artist\_id taken from Artists table and the specific song name from our study playlist from Spotify account, without duplicates while running 20 at a time. The main function establishes the study playlist and car playlist IDs, the username of our Spotify account, and calls all the functions we previously created to add all Spotify tables to database.

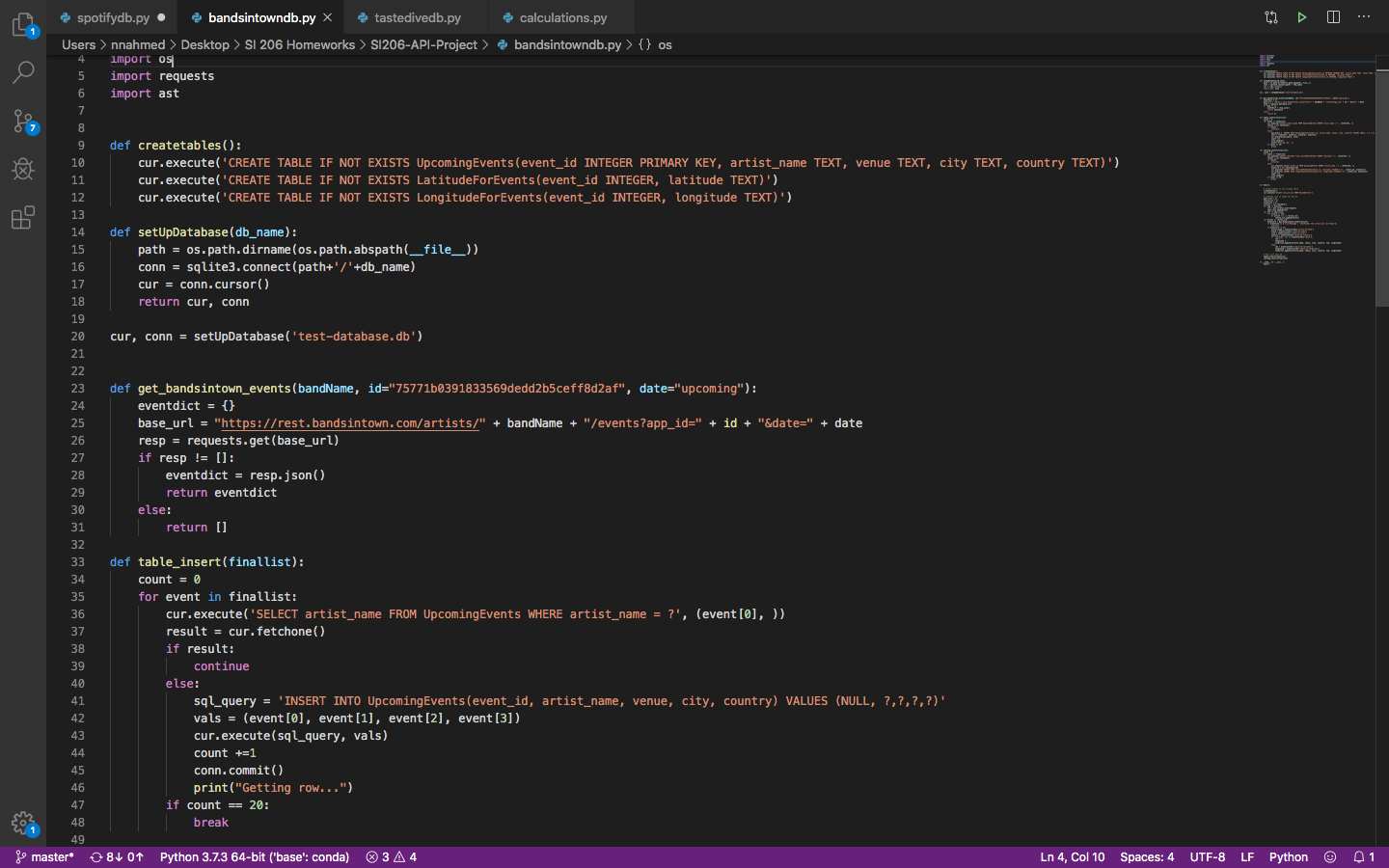


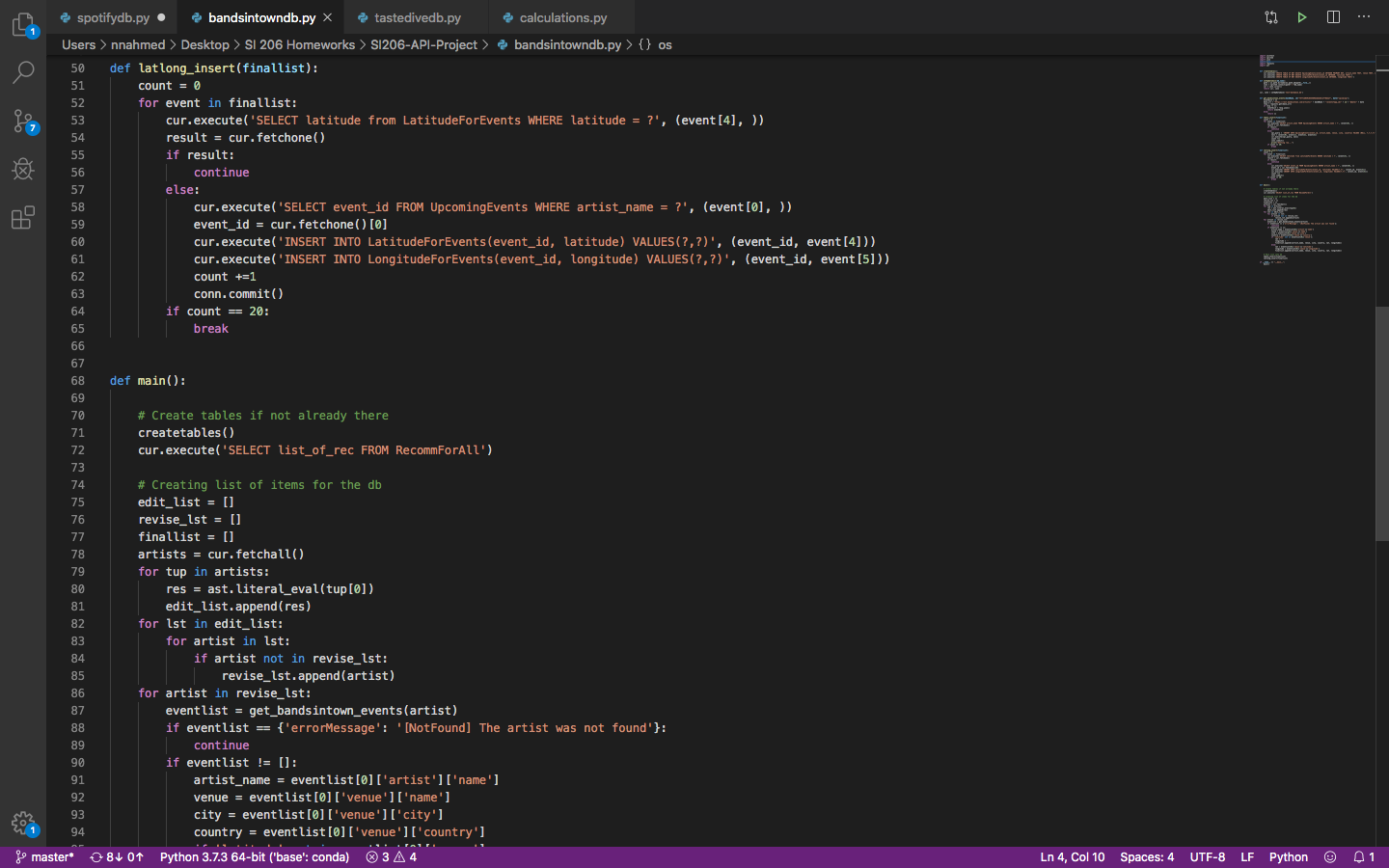
Create\_tables function creates the tables we are going to use with data pulled from TasteDive API. setUpDatabase function creates our database and creates the cur and conn objects for us to use. Get\_recommendations\_from\_tastedive function returns a JSON dictionary of artist recommendations of an artist you input. The table\_insert function takes a list of artists you input and inserts a list of recommended artists with the artist's name in the RecommForAll table, without duplicates while running 20 at a time.



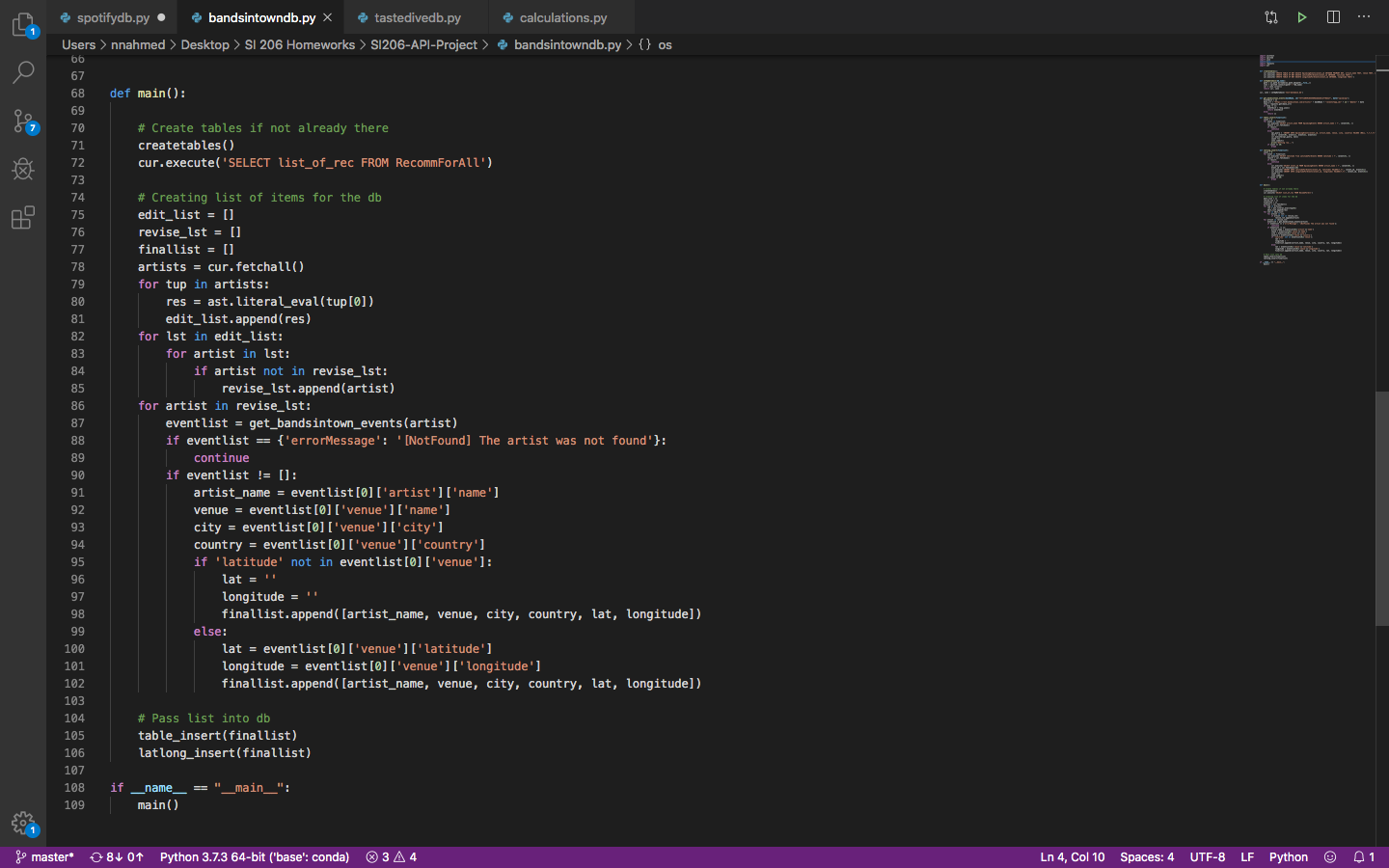
Carplaylist\_table\_insert function takes a list of artist IDs from the car playlist and selects the artist's name from the Artists table, then inserting a list of recommended artists with the artist's name identified from the artist ID in the RecommForCar table, without duplicates while running 20 at a time.

studyplaylist\_table\_insert function takes a list of artist IDs from the study playlist and selects the artist's name from the Artists table, then inserting a list of recommended artists with the artist's name identified from the artist ID in the RecommForStudy table, without duplicates while running 20 at a time. The main function calls the createtables function and configures the list of artists and artist IDs that would act as the input for the table\_insert, carplaylist\_table\_insert, and studyplaylist\_table\_insert functions.

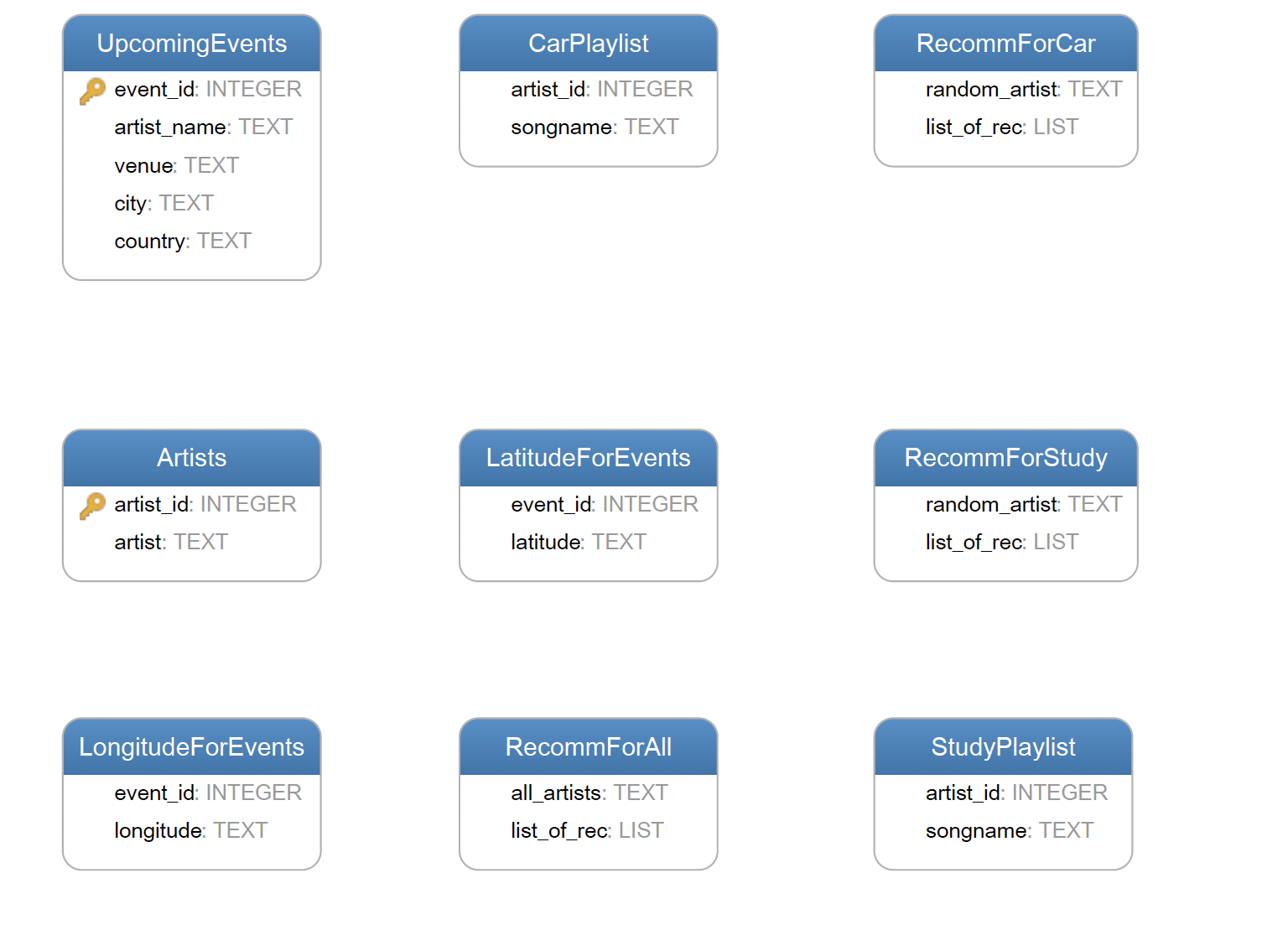
Createtables function creates the tables we are going to use with data pulled from the BandsInTown API. setUpDatabase function creates the cur and conn objects for us to use. Get\_bandsintown\_events function creates a JSON dictionary with the results of upcoming events from our recommended artists. Table\_insert function creates the Upcoming Events table where we have the event\_id, recommended artist, venue, city, country of an upcoming event of theirs, without duplicates while running 20 at a time. 



Latlong\_insert function selects the event\_id from Upcoming Events table and inserts it into the LatitudeForEvents table and the LongitudeForEvents table at the same time. The associated event\_id is put into the tables with the appropriate latitude and longitude respectively, without duplicates while running 20 at a time.



The main function selects all of our artists from the RecommForAll table so that we can go through them and input each artist into the get\_bandsintown\_events mentioned previously. We were coming across some errors as some artists were not found in bandsintown so we had to deal with that separately in the main function. This is also where we pulled specific data from the JSON dictionary and called the table\_insert and latlong\_insert functions.



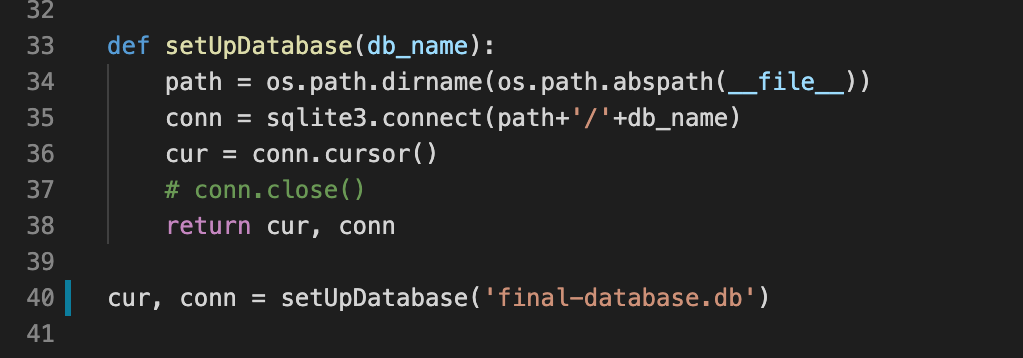


saveTextFile Function:

Simple function designed to take data and a filename and write the data to the file. We used saveTextFile() within our calculation functions to write our data into text files.

readDataFromFile Function:

This function takes a filename and reads the contents of the file, returning a json file.



setUpDatabase Function:

This function was used to establish a connection with the database so that we could use SELECT statements to gather data for our calculations. It takes a database name as an input and returns cur (cursor) and conn (connection to the database) which allows us to read data.

calcBandsInTown Function:

def calcBandsInTown(cur, conn):

cur.execute("SELECT count(country) FROM UpcomingEvents WHERE country = 'United States';")

us\_number = cur.fetchall()[0][0]

cur.execute("SELECT count(country) FROM UpcomingEvents;")

total\_number = cur.fetchall()[0][0]

percent\_us = us\_number / total\_number \* 100

info = "This is a file containing percentage data for USA events from our UpcomingEvents table.\n"

us\_str = "Number of events in USA: " +str(us\_number) + "\n"

total\_str = "Total Number of events: " +str(total\_number) + "\n"

percent\_str = "Percentage for events in USA: " +str(percent\_us) + "\n"

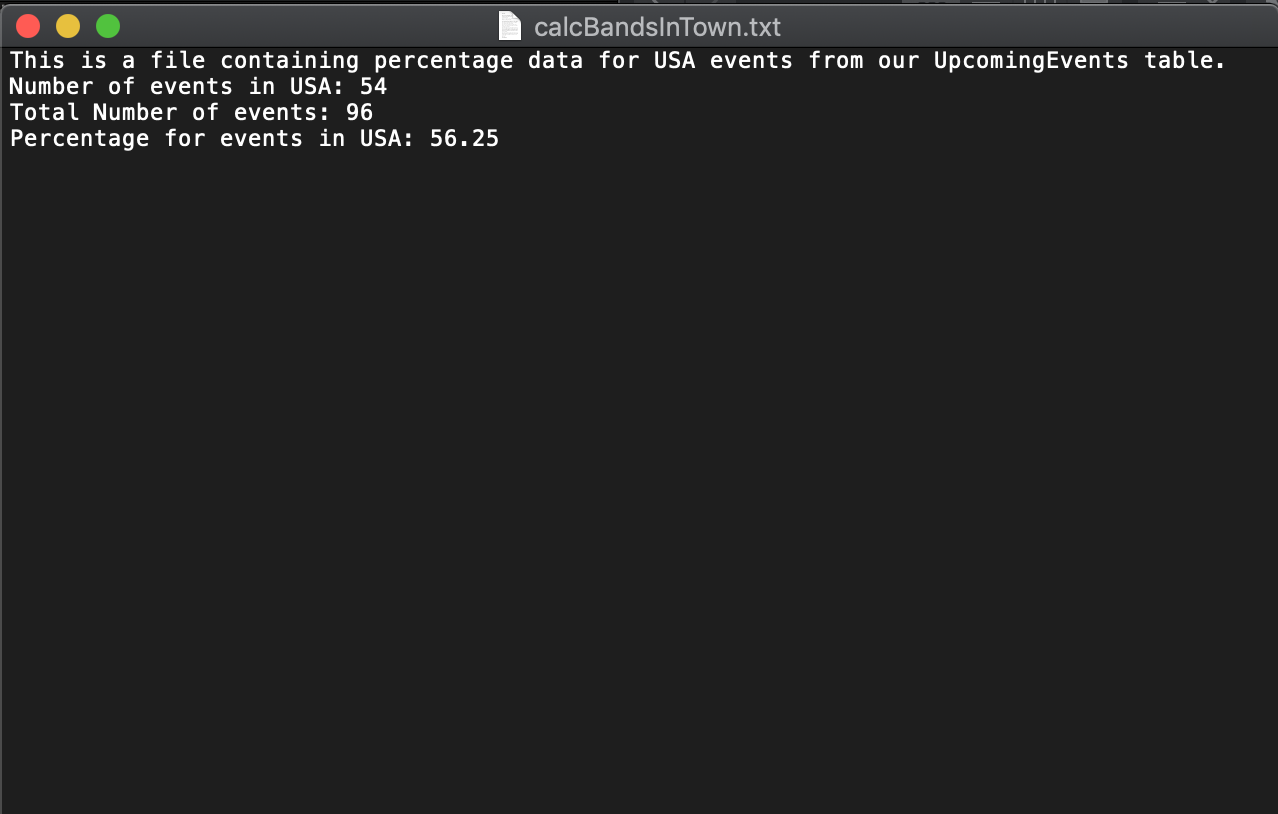
combined\_str = info + us\_str + total\_str + percent\_str

saveTextFile(combined\_str, 'calcBandsInTown.txt')

visualizationBandsInTownPie(us\_number,total\_number)

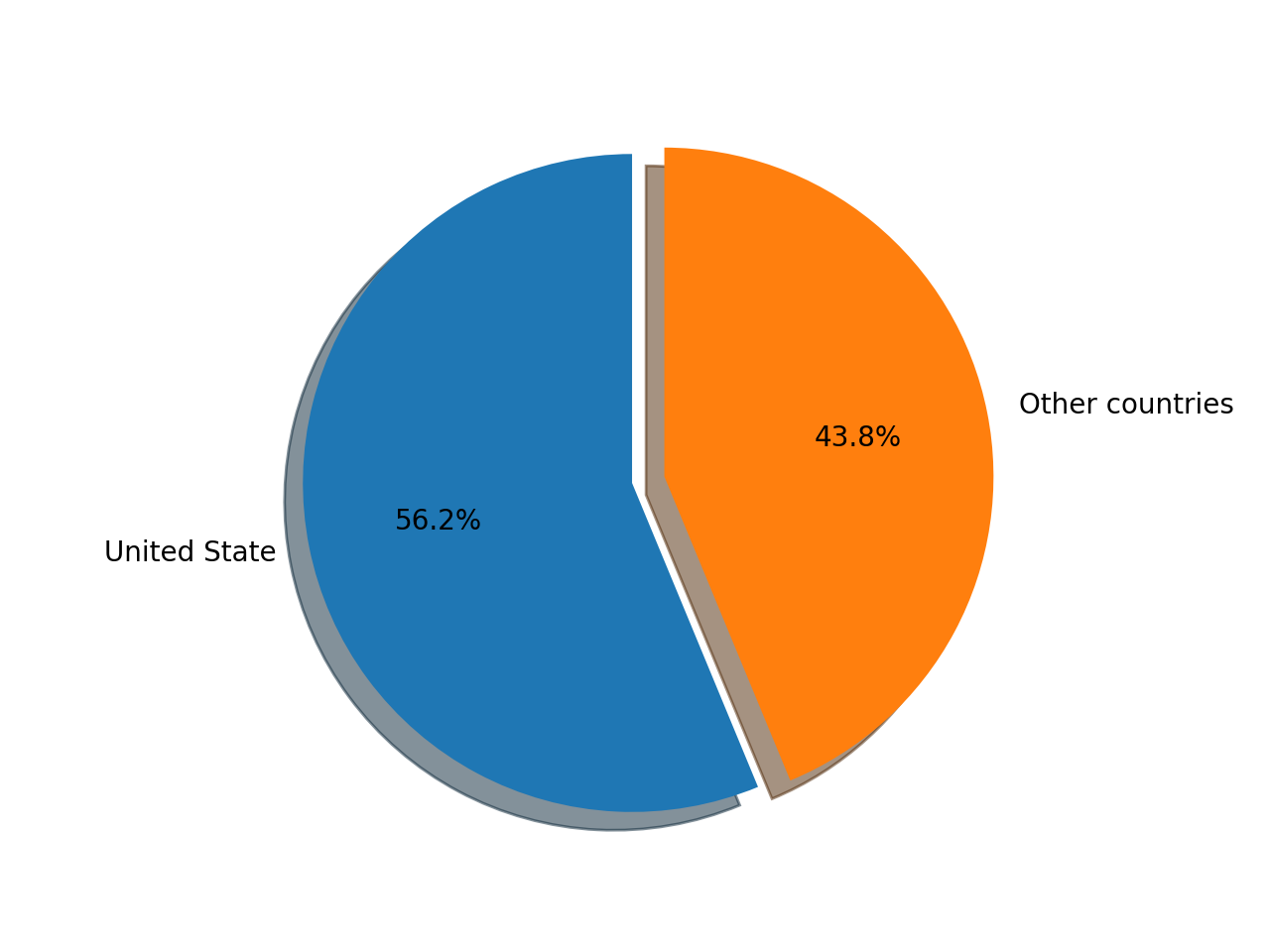
The calcBandsInTown function will Select from UpcomingEvents Table the number of all events occurring in the USA and total number of events around the world and it will calculate the percentage of USA events. Percent\_us = us\_number/ total\_number \*100

Then the result will be exported to a text file “calcBandsInTown.txt”



visualizationBandsInTownPie function:

We used matplotlib library to display the results from calcBandsInTown Function into a pie chart.



get\_events\_by\_country Function:

def get\_events\_by\_country(cur, conn):

cur.execute("SELECT country, count(country) from UpcomingEvents GROUP BY country")

events\_by\_country = cur.fetchall()

print("Events by country:")

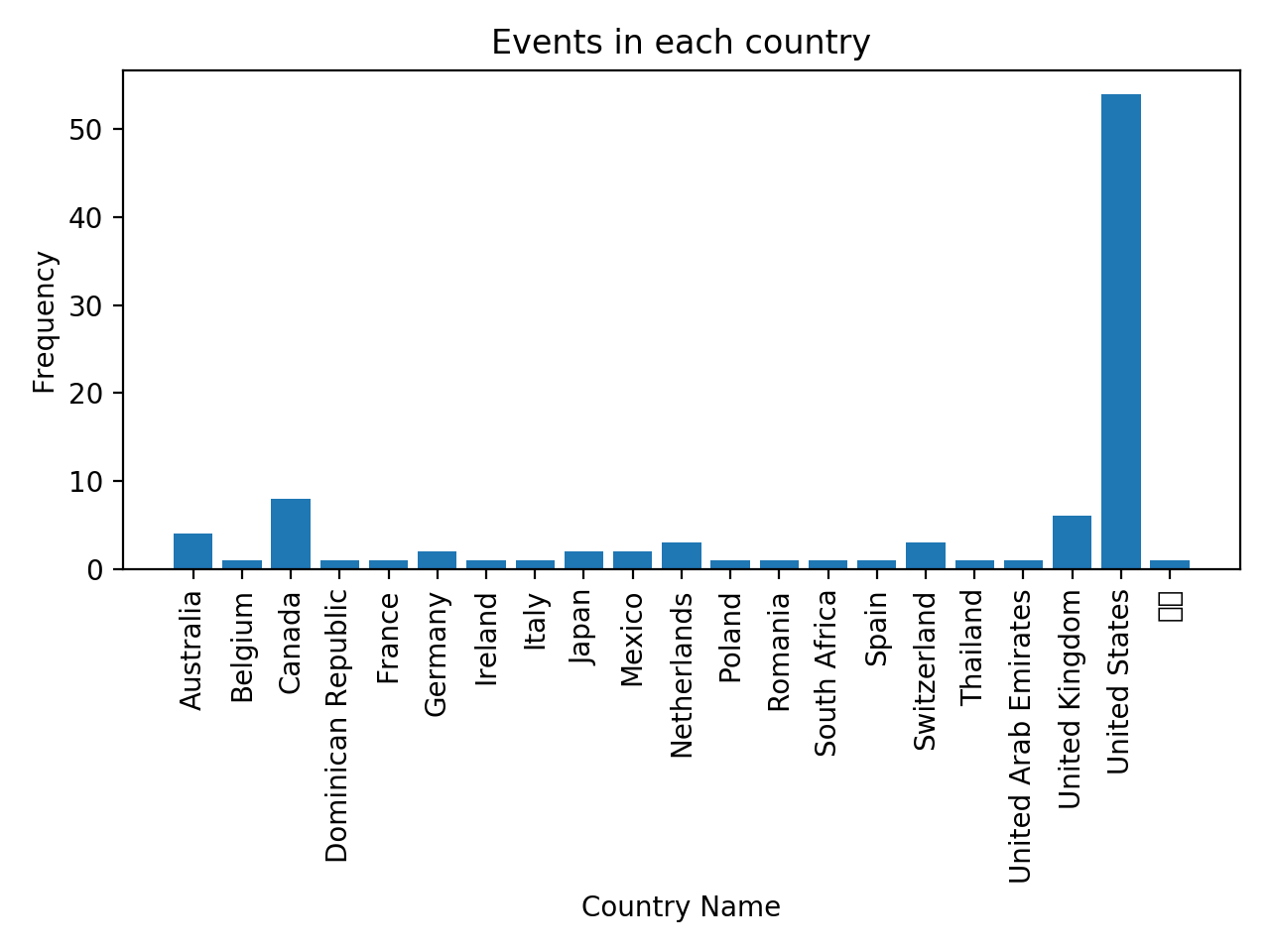
print(events\_by\_country)

draw\_barchart(events\_by\_country)

This function will select from the UpcomingEvents Table the number of events in each country.

draw\_barchart:

We used matplotlib library to display the results from get\_events\_by\_country function into a barchart.



get\_Long\_Lat Function:

def get\_Long\_Lat(cur, conn):

info = "This function will calculate the distance between concert venue and University of Michigan/n"

saveTextFile(info, 'calcDistance.txt')

venue = input("Enter venue : ")

cur.execute("SELECT latitude, longitude FROM LongitudeForEvents JOIN LatitudeForEvents JOIN UpcomingEvents WHERE venue=(?) and LatitudeForEvents.event\_id = UpcomingEvents.event\_id and LongitudeForEvents.event\_id = UpcomingEvents.event\_id", (venue,))

results = cur.fetchall()

result\_txt = "The Distance between " +venue+ "and University of Michigan is: "

saveTextFile(result\_txt, 'calcDistance.txt')

Distance = calDistance(float(results[0][0]), float(results[0][1]))

saveTextFile(str(Distance), 'calcDistance.txt')

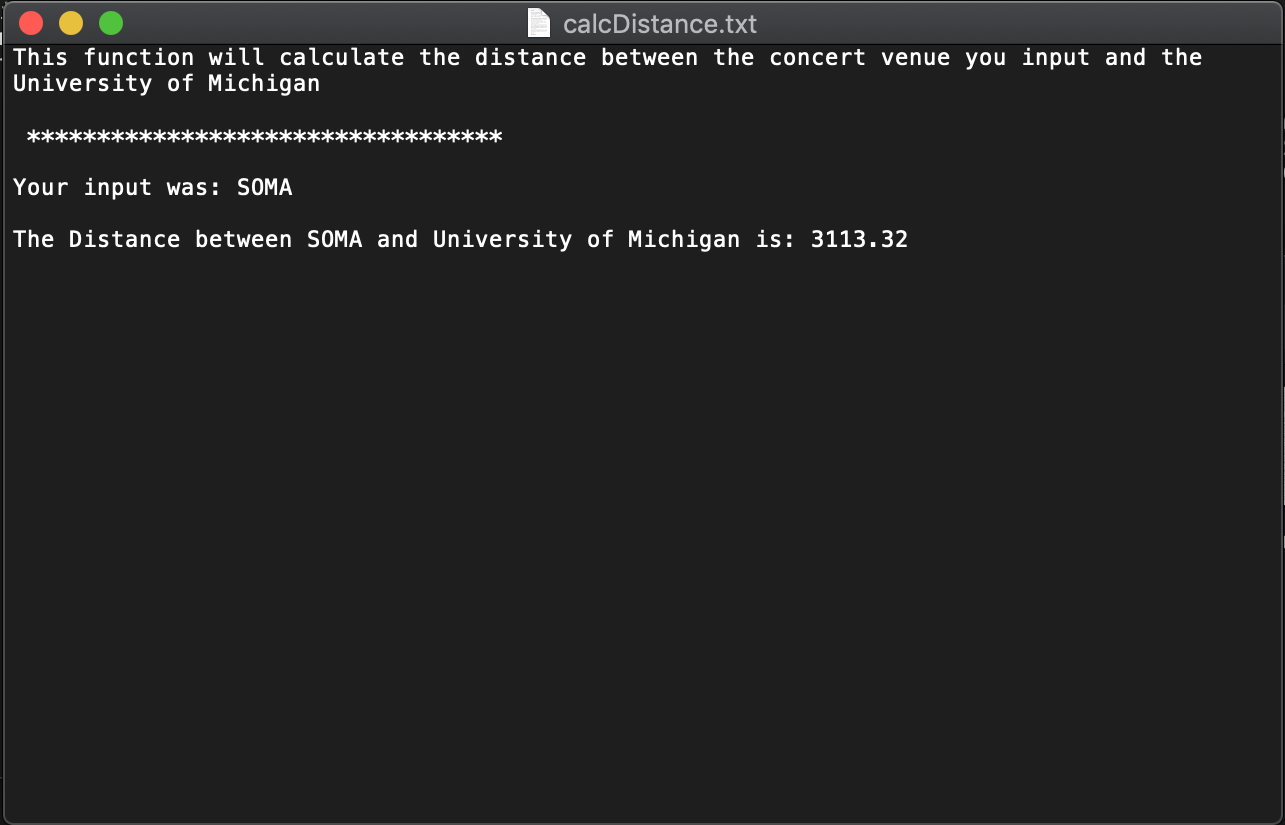
DistanceVisual(float(results[0][0]), float(results[0][1]),venue,Distance)

This function will select Latitude and Longitude from LongtitudeForEvents table and LatitudeForEvents table join UpcomingEvents table

This function will prompt the user to enter an event venue, then the function will call calDistance function and will pass the longitude latitude of the entered venue into the function.



The function will also produce a text file called calcDistance.txt with the input and result



calDistance Function:

This function will calculate the distance between the entered venue by user and University of Michigan.

def calDistance(lat, lon):

#calculate distance between University of Michigan and event venue in km

R = 6373.0

lat1 = radians(lat)

lon1 = radians(lon)

coord1 = "The first set of coordinates is ({},{})\n".format(lat1,lon1)

lat2 = radians(42.2780)

lon2 = radians(-83.7382)

coord1 = "The second set of coordinates is ({},{})\n".format(lat2,lon2)

dlon = lon2 - lon1

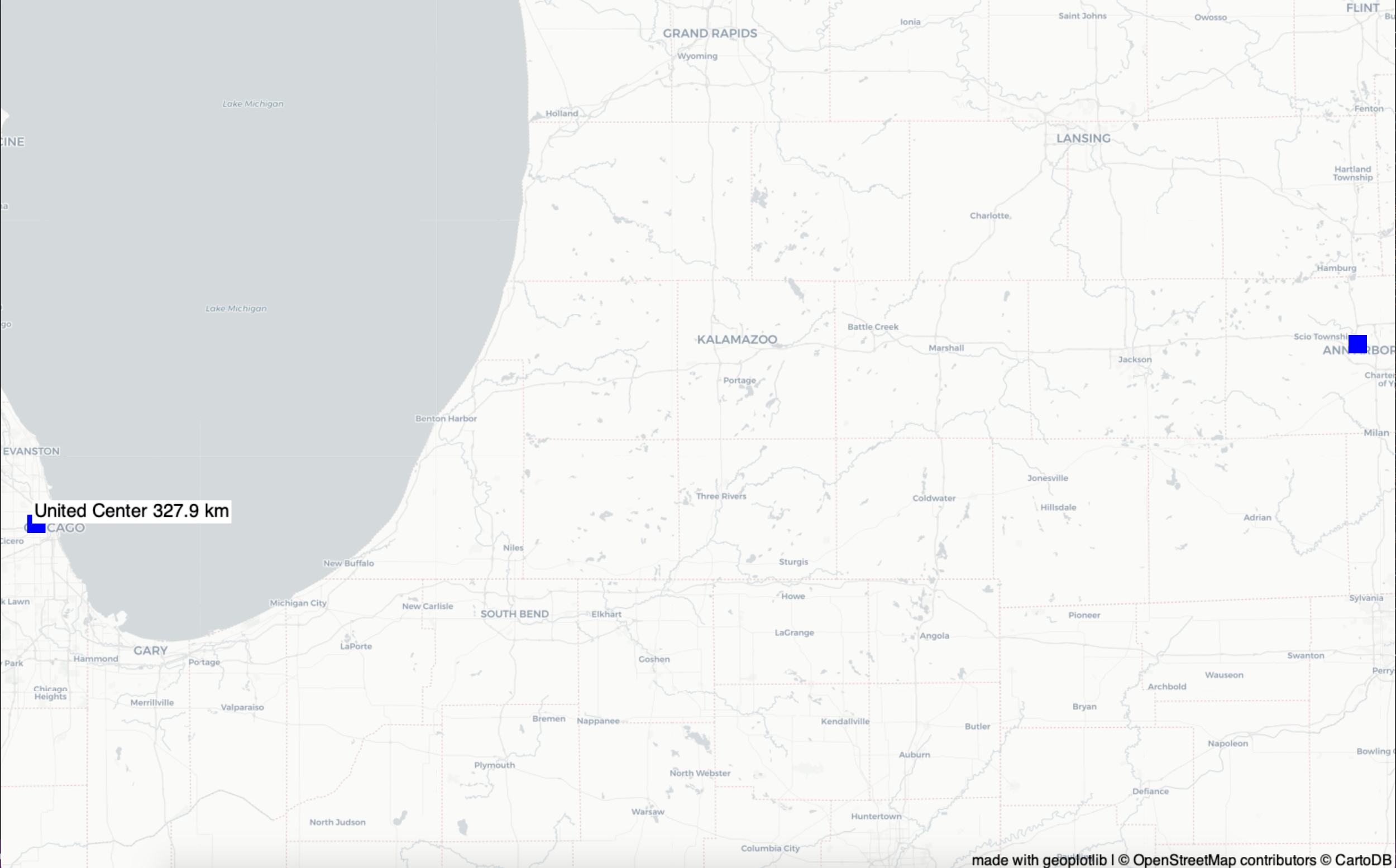
dlat = lat2 - lat1

a = sin(dlat / 2)\*\*2 + cos(lat1) \* cos(lat2) \* sin(dlon / 2)\*\*2

c = 2 \* atan2(sqrt(a), sqrt(1 - a))

distance = round((R \* c),2)

return distance

DistanceVisual Function:

In this function we used GeoPlotLib to visualize the distance between the entered venue and University of Michigan.

get\_artist\_playList:

def get\_artist\_playList(cur, conn):

info = "This will calculate how many times an artist appeared in a playlist\n"

cur.execute("SELECT artist, COUNT(StudyPlaylist.artist\_id) FROM StudyPlaylist JOIN Artists WHERE StudyPlaylist.artist\_id = Artists.artist\_id GROUP BY StudyPlaylist.artist\_id")

Study\_results = cur.fetchall()

print("Study PlayList")

study\_str = "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\nStudy Playlist Frequency:\n\n" + str(Study\_results)

cur.execute("SELECT artist, COUNT(CarPlaylist.artist\_id) FROM CarPlaylist JOIN Artists WHERE CarPlaylist.artist\_id = Artists.artist\_id GROUP BY CarPlaylist.artist\_id")

Car\_results = cur.fetchall()

car\_str = "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\nCar Playlist Frequency:\n\n" + str(Car\_results)

print(Car\_results)

combined\_str = info + study\_str + car\_str

saveTextFile(combined\_str, 'calcArtistFrequency.txt')

calc\_Artistfreq\_PlayList(Study\_results, Car\_results)

This function will get artist name from StudyPlayList and CarPlayList and pass them to calc\_Artistfreq function. The result will be saved into calcArtistFrequency.txt

def calc\_Artistfreq\_PlayList(list1, list2):

freq\_list = []

for item1 in list1:

for item2 in list2:

if item1[0] == item2[0]:

newValue = item1[1] + item2[1]

freq\_list.append((item1[0], newValue))

words = ""

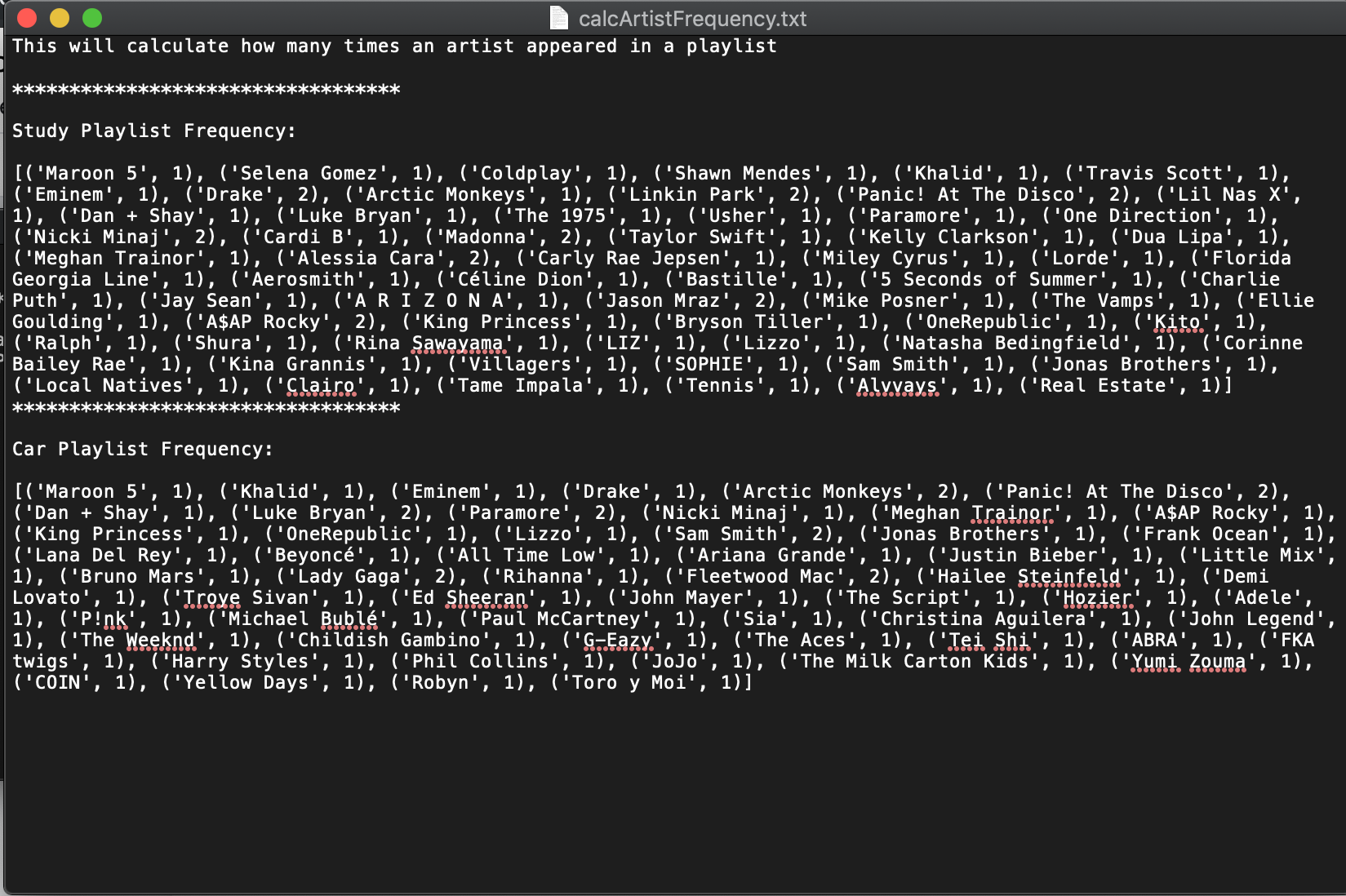
for item in freq\_list:

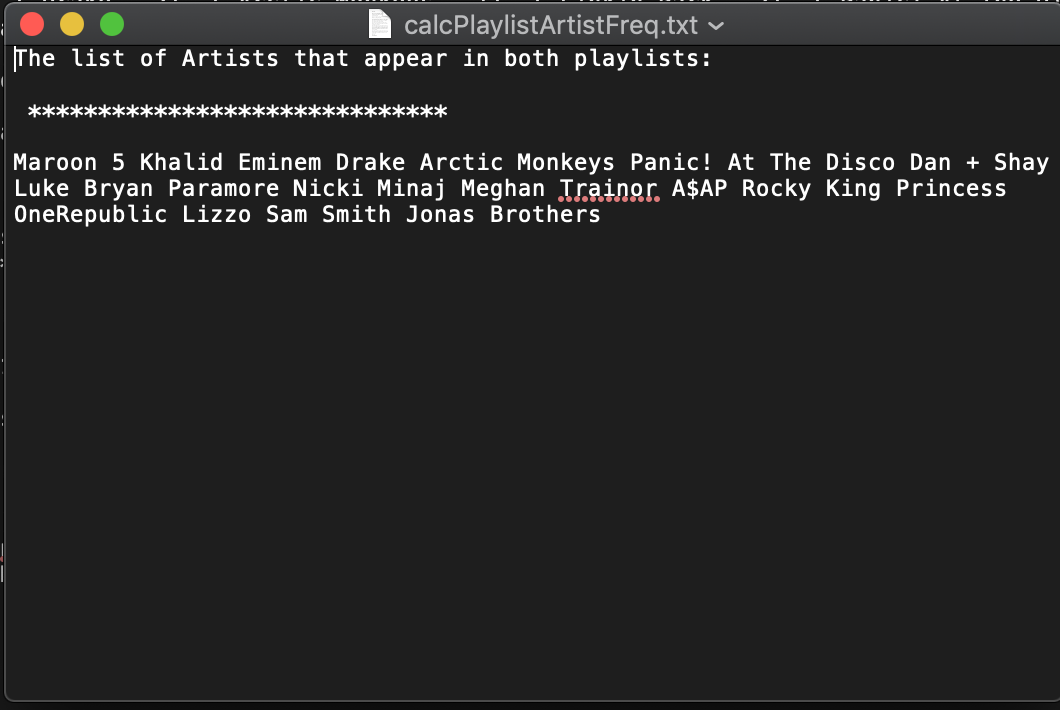
words = words+item[0]+" "

info = "The list of Artists that appear in both playlists:\n\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n" + words

saveTextFile(info, "calcPlaylistArtistFreq.txt")

draw\_wordcloud(words)





This function will get the list of Artists that appear in both car and study PlayList and will pass a string of Artist names to draw\_wordcloud function.

draw\_wordcloud function:

This function will take the the list of Artists that appear in both car and study PlayList and display the results in a wordcloud.



Visualization\_recommendation Function:

def visualization\_recommendation(cur, conn):

artist = input("Enter artist name : ")

cur.execute("SELECT all\_artists, list\_of\_rec FROM RecommForAll WHERE all\_artists=?", (artist,))

results = cur.fetchall()

Recommendation = ast.literal\_eval(results[0][1])

artist\_name = results[0][0]

g = nx.DiGraph()

g.add\_edge(artist\_name,Recommendation[0])

g.add\_edge(artist\_name,Recommendation[1])

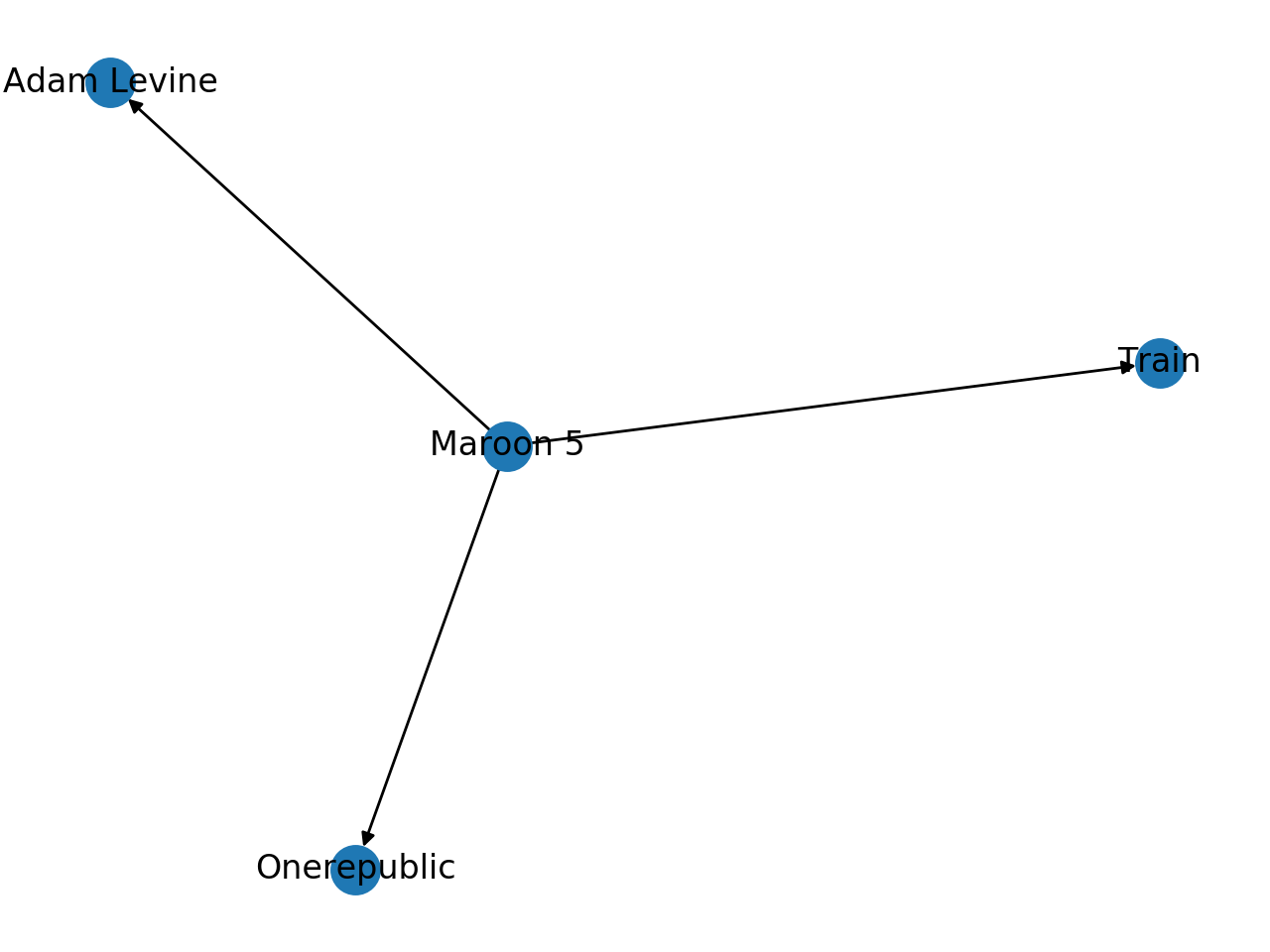
g.add\_edge(artist\_name,Recommendation[2])

nx.draw(g,with\_labels=True)

plt.draw()

plt.show()

This function will prompt the user to enter an Artist name and will return three artists recommendations in a networkx visualization.

****

Resources Used

Spotipy Documentation: used around 11/18/19, needed to use Spotify API using client credentials, found on Google, it definitely solved the issue we were facing when trying to figure out how to use Spotify API.

Link: <https://spotipy.readthedocs.io/en/latest/>

Spotify API: used around 11/18/19, needed to use Spotify API to pull playlist information from our Spotify account, found on Google, it definitely solved the issue of how to utilize the Spotipy documentation in correlation with the Spotify API information.

Link: <https://developer.spotify.com/documentation/web-api/>

TasteDive API: used around 11/18/19, needed to use TasteDive API to pull recommended artists from the artists we had in our playlists, found on Google, it definitely solved the issue of what we wanted to do in our project and to pull different artists to put in our database.

Link: <https://tastedive.com/read/api>

BandsInTown API: used around 11/18/19, needed to use BandsInTown API to pull upcoming events from the recommended artists we pulled from TasteDive, found on Google, it definitely solved the issue of what we wanted to do in our project and to pull a lot of different data, like event venue and event location, to put in our database.

Link: <https://app.swaggerhub.com/apis-docs/Bandsintown/PublicAPI/3.0.0>

MatplotLib: used around 12/1/19, needed to use MatplotLib to visualize a bar chart and pie chart, found on Google, it definitely solved the issue of how we wanted to visualize the percentage data for USA events from our UpcomingEvent table.

Link: <https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.html>

WordCloud: used around 12/1/19, needed to use WordCloud to visualize a cloud of most frequent artists in our playlists, found on Google, it definitely solved the issue of how we wanted to visualize the data we received from the Spotify API.

Link: <https://pypi.org/project/wordcloud/>

GeoPlotLib: used around 12/1/19, needed to use GeoPlotLib to visualize the map for the distance between the UOFM campus and the event location, found on Google, it definitely solved the issue of how we wanted to visualize the data from BandsinTown in a sort of interactive manner.

Link: <https://andrea-cuttone.github.io/geoplotlib/genindex.html>