Max Chalfin and Andrew Nguyen
Musical Meteorologists
SI 206
12/19/2019
***Github Repository: https://github.com/arnguyen/SI206-FinalProject

Final Report

Goals:

- Track the frequency that a genre of music was played depending on the current weather.
- Create a playlist of songs that correlated to the weather:
    - Sad playlist = rainy
    - Happy playlist = sunny
- Collect weather data from the OpenWeatherMap API
- Collect music data from the Spotify API
- Originally planned to find out which songs were being played the most and when, map it to specific days with forecasts, and wanted to create playlists of most played songs from those days based on mood (forecasts, i.e. rainy = sad, etc.)

Achieved Goals:

- Spotify api however doesn't provide the data we wanted, there was no way to get a history of when songs were being played
- As a result, we ended up finding the most popular genres from spotify api
- Form weather api, we gathered the most frequent forecasts, as well as found the average weather lows and highs for each time period given from the api
- We were able to visualize these and get a rough idea of the most popular genres versus weather forecasts, as both apis draw from recent data

Problems:

- As stated, we were unable to find the data that we originally wanted to get
- In addition, using the spotify api was incredibly confusing, we honestly did not understand how the spotify oauth connection worked at all, the documentations were very confusing
- As a result, the spotifyOauth.py file used takes care of that, but was essentially pulled from a public Github repository with some minor tweaks to make it work for our purposes

File that contains calculations from the data in the database:

- Screenshots of databases are provided below, as well as code to process data
- Files with data outputs are provided in zipped folder

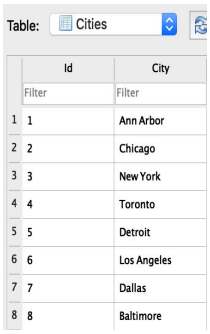


Table: Cities

	Id	City
	Filter	Filter
1	1	Ann Arbor
2	2	Chicago
3	3	New York
4	4	Toronto
5	5	Detroit
6	6	Los Angeles
7	7	Dallas
8	8	Baltimore

Table: Genres

	genre_id	Title
	Filter	Filter
1	1	happy
2	2	rainy-day
3	3	sad
4	4	summer
5	5	holidays
6	6	ambient

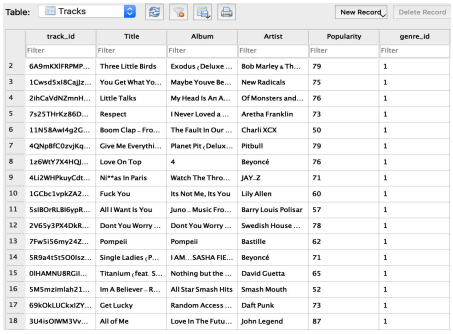


Table: Tracks

	track_id	Title	Album	Artist	Popularity	genre_id
	Filter	Filter	Filter	Filter	Filter	Filter
2	6A9mKXlFRPMP…	Three Little Birds	Exodus ‹ Deluxe …	Bob Marley & Th…	79	1
3	1Cwsd5xI8CajJz…	You Get What Yo…	Maybe Youve Be…	New Radicals	75	1
4	2ihCaVdlNZmmH…	Little Talks	My Head Is An A…	Of Monsters and…	76	1
5	7s25THrKz86D…	Respect	I Never Loved a …	Aretha Franklin	73	1
6	11N58AWI4g2G…	Boom Clap - Fro…	The Fault In Our …	Charli XCX	50	1
7	4QNpBfC0zvjKq…	Give Me Everythi…	Planet Pit ‹ Delux…	Pitbull	79	1
8	1z6WtY7X4HQJ…	Love On Top	4	Beyoncé	76	1
9	4Li2WHPkuyCdt…	Ni**as In Paris	Watch The Thro…	JAY..Z	71	1
10	1GCbc1vpkZA2…	Fuck You	Its Not Me, Its You	Lily Allen	60	1
11	5sI8OrRLBI6ypR…	All I Want Is You	Juno - Music Fro…	Barry Louis Polisar	57	1
12	2V65y3PX4DkR…	Dont You Worry …	Dont You Worry …	Swedish House …	78	1
13	7FwSI56my24Z…	Pompeii	Pompeii	Bastille	62	1
14	5R9a4t5ISOOlsz…	Single Ladies ‹ P…	I AM…SASHA FIE…	Beyoncé	71	1
15	0IHAMNU8RGII…	Titanium ‹ feat. S…	Nothing but the …	David Guetta	65	1
16	5MSmzimlah21…	Im A Believer - R…	All Star Smash Hits	Smash Mouth	52	1
17	69kOkLUCkxIZY…	Get Lucky	Random Access …	Daft Punk	73	1
18	3U4isOWM3Vv…	All of Me	Love In The Futu…	John Legend	87	1

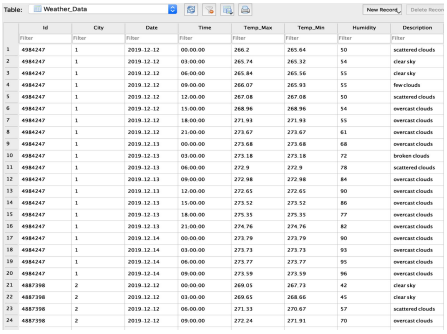
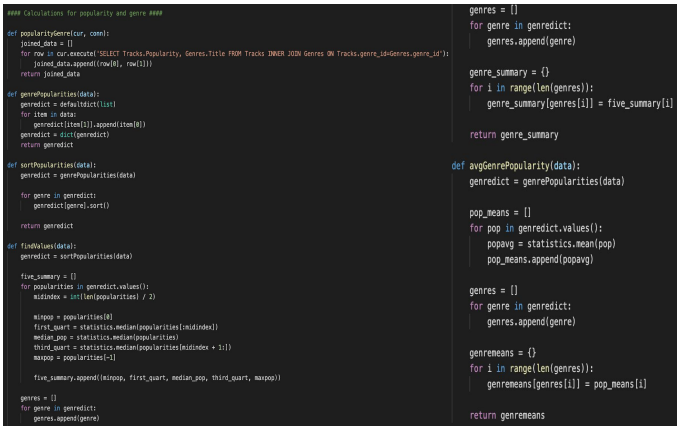
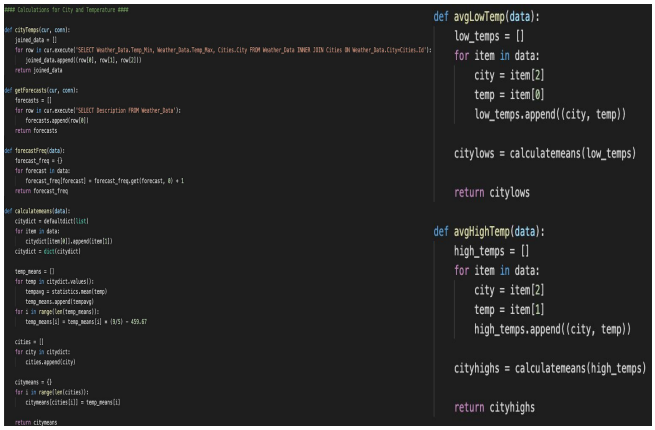


Table: Weather_Data

	Id	City	Date	Time	Temp_Max	Temp_Min	Humidity	Description
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	4984247	1	2019.12.12	00:00:00	266.2	265.64	50	scattered clouds
2	4984247	1	2019.12.12	03:00:00	265.74	265.32	54	clear sky
3	4984247	1	2019.12.12	06:00:00	265.84	265.56	55	clear sky
4	4984247	1	2019.12.12	09:00:00	266.07	265.93	55	few clouds
5	4984247	1	2019.12.12	12:00:00	267.08	267.08	50	scattered clouds
6	4984247	1	2019.12.12	15:00:00	268.96	268.96	54	overcast clouds
7	4984247	1	2019.12.12	18:00:00	271.93	271.93	55	overcast clouds
8	4984247	1	2019.12.12	21:00:00	273.67	273.67	61	overcast clouds
9	4984247	1	2019.12.13	00:00:00	273.68	273.68	68	overcast clouds
10	4984247	1	2019.12.13	03:00:00	273.18	273.18	72	broken clouds
11	4984247	1	2019.12.13	06:00:00	272.9	272.9	78	scattered clouds
12	4984247	1	2019.12.13	09:00:00	272.98	272.98	84	overcast clouds
13	4984247	1	2019.12.13	12:00:00	272.65	272.65	90	overcast clouds
14	4984247	1	2019.12.13	15:00:00	273.52	273.52	86	overcast clouds
15	4984247	1	2019.12.13	18:00:00	275.35	275.35	77	overcast clouds
16	4984247	1	2019.12.13	21:00:00	274.76	274.76	82	overcast clouds
17	4984247	1	2019.12.14	00:00:00	273.79	273.79	90	overcast clouds
18	4984247	1	2019.12.14	03:00:00	273.73	273.73	93	overcast clouds
19	4984247	1	2019.12.14	06:00:00	273.77	273.77	95	overcast clouds
20	4984247	1	2019.12.14	09:00:00	273.59	273.59	96	overcast clouds
21	4887398	2	2019.12.12	00:00:00	269.05	267.73	42	clear sky
22	4887398	2	2019.12.12	03:00:00	269.65	268.66	45	clear sky
23	4887398	2	2019.12.12	06:00:00	271.33	270.67	57	scattered clouds
24	4887398	2	2019.12.12	09:00:00	272.24	271.91	70	overcast clouds



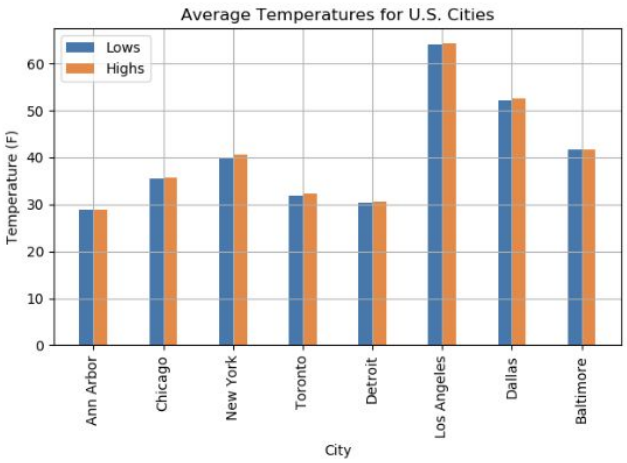
```
#### Calculations for popularity and genre ####

def popularityGenre(cur, conn):
    joined_data = []
    for row in cur.execute('SELECT Tracks.Popularity, Genres.Title FROM Tracks INNER JOIN Genres ON Tracks.genre_id=Genres.genre_id'):
        joined_data.append((row[0], row[1]))
    return joined_data

def genrePopularities(data):
    genredict = defaultdict(list)
    for item in data:
        genredict[item[1]].append(item[0])
    genredict = dict(genredict)
    return genredict

def sortPopularities(data):
    genredict = genrePopularities(data)

    for genre in genredict:
        genredict[genre].sort()

    return genredict

def findValues(data):
    genredict = sortPopularities(data)

    five_summary = []
    for popularities in genredict.values():
        midindex = int(len(popularities) / 2)

        minpop = popularities[0]
        first_quart = statistics.median(popularities[:midindex])
        median_pop = statistics.median(popularities)
        third_quart = statistics.median(popularities[midindex + 1:])
        maxpop = popularities[-1]

        five_summary.append((minpop, first_quart, median_pop, third_quart, maxpop))

    genres = []
    for genre in genredict:
        genres.append(genre)
    genres = []
    for genre in genredict:
        genres.append(genre)

    genre_summary = {}
    for i in range(len(genres)):
        genre_summary[genres[i]] = five_summary[i]

    return genre_summary

def avgGenrePopularity(data):
    genredict = genrePopularities(data)

    pop_means = []
    for pop in genredict.values():
        popavg = statistics.mean(pop)
        pop_means.append(popavg)

    genres = []
    for genre in genredict:
        genres.append(genre)

    genremeans = {}
    for i in range(len(genres)):
        genremeans[genres[i]] = pop_means[i]

    return genremeans
```

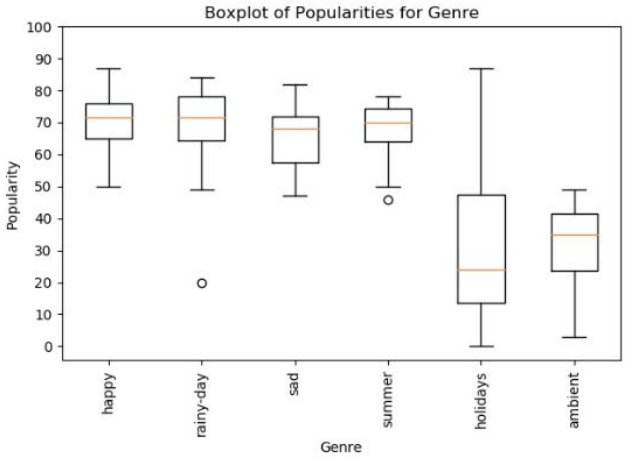


```
#### Calculations for City and Temperature ####

def cityTemps(cur, conn):
    joined_data = []
    for row in cur.execute('SELECT Weather_Data.Temp_Min, Weather_Data.Temp_Max, Cities.City FROM Weather_Data INNER JOIN Cities ON Weather_Data.CityeCities.Id'):
        joined_data.append((row[0], row[1], row[2]))
    return joined_data

def getForecasts(cur, conn):
    forecasts = []
    for row in cur.execute('SELECT Description FROM Weather_Data'):
        forecasts.append(row[0])
    return forecasts

def forecastFreq(data):
    forecast_freq = {}
    for forecast in data:
        forecast_freq[forecast] = forecast_freq.get(forecast, 0) + 1
    return forecast_freq

def calculatemeans(data):
    citydict = defaultdict(list)
    for item in data:
        citydict[item[0]].append(item[1])
    citydict = dict(citydict)

    temp_means = []
    for temp in citydict.values():
        tempavg = statistics.mean(temp)
        temp_means.append(tempavg)
    for i in range(len(temp_means)):
        temp_means[i] = temp_means[i] * (9/5) - 459.67

    cities = []
    for city in citydict:
        cities.append(city)

    citymeans = {}
    for i in range(len(cities)):
        citymeans[cities[i]] = temp_means[i]

    return citymeans
```

```
def avgLowTemp(data):
    low_temps = []
    for item in data:
        city = item[2]
        temp = item[0]
        low_temps.append((city, temp))

    citylows = calculatemeans(low_temps)

    return citylows

def avgHighTemp(data):
    high_temps = []
    for item in data:
        city = item[2]
        temp = item[1]
        high_temps.append((city, temp))

    cityhighs = calculatemeans(high_temps)

    return cityhighs
```

Visualizations:





Instructions for running code:

1. Run main.py
2. There will be several prompts provided throughout the process, just follow the prompts as instructed
3. ***Important Note***
    a. If trying to collect data, the access token may be expired. If that's the case, you will see a message that looks like this:

```
Access code expired, getting new access code
Bottle v0.12.18 server starting up (using WSGIRefServer())...
Listening on http://:8080/
Hit Ctrl-C to quit.
```

You will need to open up a browser and go to "http://localhost:8080/". This will prompt you to login to spotify and get an access token. If successful, a message like this should pop up:

```
Found cached token!
Access token available! Writing access token...
127.0.0.1 - - [19/Dec/2019 13:07:13] "GET / HTTP/1.1" 200 0
127.0.0.1 - - [19/Dec/2019 13:07:13] "GET /favicon.ico HTTP/1.1" 404 742
```

At this point, you can hit Ctrl-C and everything else will run normally.

Documentation for each function

weather.py:

1. getWeather(city_name):
   This function takes in a city provided by the user to generate a complete_url to make a request to the OpenWeatherMap API and converts the data into a JSON object.
2. getCityTable(data):
   This function takes in the JSON object from the getWeather(city_name) function and loads all the weather in the JSON object into a table called Cities. The table has the following columns: Id and City.
3. getWeatherTable(data):

This function takes in the JSON object from the getWeather(city_name) function and loads all the weather in the JSON object into a table called Weather_Data. The table has the following columns: Id, City, Date, Time, Temp_Max, Temp_Min, Humidity, and Description. Also, there is a counter that only allows 20 rows of weather data to be added every time the code is run.

4. user_response():
   This function gives the user a yes or no option to enter another city. It also notifies them if the city they want to input is already added to the database. If the user does not want to input another city then they are thanked for their time.
5. main():
    a. Runs processes

spotifyOauth.py:

1. htmlForLoginButton():
    a. Creates a login button for users to login to spotify
2. getSPOauthURI():
    a. Gets the uri
3. write_access_token(access_token):
    a. Writes the access token to a seperate text file for later processing
4. index():
    a. Runs the whole process of getting the access token

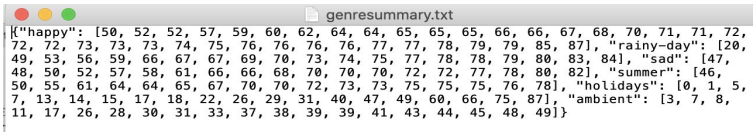
spotifyCollection.py:

1. read_access_token():
    a. Reads the access token written by write_access_token
2. check_access_token(access_token):
    a. Checks to see if the current access token works or if it has already expired
3. setUpDatabase(db_name):
    a. Sets up database
4. setUpGenreTable(data, cur, conn): and setUpTrackTable(data, cur, conn):
    a. Sets up tables
5. user_response():
    a. Interface for users to choose options
6. Class DataCollection:
    a. __init__(self, genre, access_token):
        i. Constructor for DataCollection object
    b. getData(self):
        i. Returns self.data
    c. collectData(self):
        i. Connects to API and connects data to put into self.data list

7. main():
    a. Runs full process

dataProcessor.py:

1. dataCollection():
   This function asks the user if they would like to collect data. If the user responds yes to the prompted question, they will be asked more specific questions in the function dataCollectionHandler. If they say no, nothing is returned.
2. dataCollectionHandler():
   This function is called when the user responds yes to the question asked within the function dataCollection(), and prompts the user with a more specific question on what kind of data they would like to collect (track data, weather data, or both).
3. user_response():
   This function lets the user know that the data has been processed successfully and asks the user if they would like to process again, which would bring them back to dataCollection().
4. databaseConnection():
   This function connects to the spotifyweather.db database.
5. writeData(filename, data):
   This function takes the JSON object returned by the API request and writes it to the desired filename as a Python string.
6. popularityGenre(cur, conn):
   This function takes in the database cursor and the database connection object and grabs data from the database
7. genrePopularities(data):
    a. Takes the data passed in and returns a dictionary of genre and list of popularity pairs
8. sortPopularities(data):
    a. Sorts the popularities



9. findValues(data):
    a. Calculates the min, 1st quartile, median, 3rd quartile and max for use in a box plot
    b. ***Note***: This function is actually not used, after we realized that matplotlib does all that automatically in making a box plot
10. avgGenrePopularity(data):

    a. Finds the average popularity of each genre

genrepopularity.txt
{"happy": 69.57894736842105, "rainy-day": 67.85, "sad": 65.47368421052632, "summer": 67.16666666666667, "holidays": 31.1, "ambient": 30.85}

11. cityTemps(cur, conn):
    a. Grabs data from database relating to the weather and city
12. getForecasts(cur, conn):
    a. Grabs all the forecast descriptions from the table
13. forecastFreq(data):
    a. Creates a dictionary of forecast and frequency
14. calculatemeans(data):
    a. Given a list, it creates a list dictionary of average temperature per city
15. avgLowTemp(data):
    a. Creates a list of low temps, which it passes into calculatemeans() to return a dictionary

avglowtemp.txt
{"Ann Arbor": 28.849099999999964, "Chicago": 35.434399999999998, "New York": 40.09820000000002, "Toronto": 31.860500000000002, "Detroit": 30.364699999999997, "Los Angeles": 64.06790000000007, "Dallas": 52.18160000000006, "Baltimore": 41.71100000000001}

16. avgHighTemp(data):
    a. Same as avgLowTemp but with a list of highs

avghightemp.txt
{"Ann Arbor": 28.97509999999994, "Chicago": 35.73140000000001, "New York": 40.58600000000001, "Toronto": 32.38969999999995, "Detroit": 30.71029999999996, "Los Angeles": 64.28209999999996, "Dallas": 52.65320000000003, "Baltimore": 41.77760000000001}

17. main():
    a. Runs full process

visualizations.py:

1. readData(filename):
    a. Reads the data written by dataProcessor.py
2. generateWordcloud(data, filename):
    a. Creates wordcloud of the given data
3. boxplotGenrePops(data, filename):
    a. Creates a boxplot of the given data
4. scatterCityTemps(data1, data2, filename):
    a. Creates a scatter plot of the given data, in which data1 is low temp and data2 is high temp
5. welcomeUser():
    a. Just creates a basic interface where a user can select which visualization they wish to see
6. user_response():

    a. Contributes to user interface by allowing user to run program again and see another visualization
7. main():
    a. Runs process

main.py:

1. menu():
   This function prompts the user with a menu of options: Collect and Process Data, View Visualizations, or Exit.
2. main():
    a. Runs everything

Document Resources:

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
12/4/19	Finding/Using OpenWeathermap API	https://openweathermap.org/appid	Yes: I was able to find all of the API documentation and parameters
12/6/19	Limiting the amount of data stored to 20 or less each time code runs	https://www.w3schools.com/python/python_mysql_limit.asp	No: This document showed me that a LIMIT statement could be used, but was not useful for me code
12/7/19	Repetitive data	https://stackoverflow.com/questions/43737613/how-to-find-if-a-key-already-exists-in-my-mysql-database	This helped me solve my issue partially
12/7/19	Issues with spotify	https://github.com/per	Changed it slightly to

	oauth	elin/spotipy_oauth_demo	fit our needs, worked like a charm. The only resource that actually solved my issue
12/7/19	Confused with how to actually use spotify api	https://medium.com/analytics-vidhya/build-your-own-playlist-generator-with-spotifys-api-in-python-ceb883938ce4	Great resource, was able to follow this to figure out how to use the spotify api to pull data
12/12/19	Unsure as to how to make a wordcloud	https://stackoverflow.com/questions/43145199/create-wordcloud-from-dictionary-values	Worked well
12/12/19	Unsure as to how to make a boxplot and what data to provide	http://blog.bharatbhole.com/creating-boxplots-with-matplotlib/	Worked well