Sydney Sella & Hazel Rosenberg

APIs, SQL, and Visualizations

https://github.com/hazelrosenberg/206-final

Original Goals

We initially planned to compare the algorithms used in Apple Music to those used in Spotify using the Apple Music API and the Spotify Web API with Spotipy. We were going to do so by gathering song recommendation data from each platform and assessing how similar the recommendations were.

Achieved Goals

APIs/websites used: Spotify Web API (with Spotipy), Kworb.net (with Beautiful Soup)

After reading the documentation for the Apple Music API and Spotipy more thoroughly, we decided that our original plan was going to be more difficult to implement than we initially thought. In order to make requests to the Apple Music API, one must be an official member of the Apple Developer Program, which neither of us are. We still wanted to use the Spotipy and the Spotify Web API, so we created a new plan to compare the most played songs on Spotify (a streaming platform) to the Billboard Hot 100, which is based on sales, radio play, and online streaming. Our comparison would be based on the proportions of each genre that were represented in the top 50 songs on Spotify and the Billboard Hot 100 by country. Due to difficulties using Beautiful Soup on Billboard.com (see Problem 2), we decided to adjust our plan once more to compare the same data from Spotify against the Apple Music streaming platform's top charts by scraping Kworb.net using Beautiful Soup.

We ended up looking at the genre distribution of the top 50 songs on Spotify to the top 50 songs on Apple Music in North America. Using the Spotify Web API through Spotipy, we were able to get song names, artist ids, and artist genres for the top songs on Spotify with Spotipy's playlist_items and artist methods. Then, using Beautiful Soup, we were able to get song names and artists for the top songs on Apple Music. To get information about the genres for the Apple Music songs, we used Spotipy again with its search method to search for each song's artist and get their genre from the search results. It was interesting to compare data between the two streaming platforms because there is often a lot of debate on which is better. The visualizations we created serve as an idea of what types of music are preferred by each platform's users. Completing this project also allowed us to enhance our programming and analytical skills through a deeper understanding of APIs, using SQL to create databases, Beautiful Soup, and designing visualizations.

Problems

1. A major problem we discovered trying to use Beautiful Soup on Billboard.com was that our lists would come back empty when we tried to specifically scrape the information for the song names and artists. We tried using different tags, but we would either output way more information than needed from the tags (more than just the song title and artist) or an empty list. The information that was retrieved when the tags worked was far too complex to extract parts of information from.

    Solution: After spending hours trying to make the Billboard site work, we decided to try and find a different solution. We looked at many different website options to scrape from, like Apple Music, but we were still having issues. Our hypothesis is that the websites that allowed users to play music from them had some sort of underlying feature on the backend that made it very difficult to scrape specific information from, like titles. We ended up coming across a site, Kworb.net, that had information about Apple Music's top charts. To use this site instead of Billboard, we had to change our research question a bit, but the website was very simple and made it much easier to scrape from. Overall, it was worth it to use Kworb instead of Billboard in completing this project due to the ease it provided.

2. Another second problem we encountered was with extracting genre information from Spotify.

    a. We used Spotipy's playlist_items method to get the information about all of the songs in the top charts playlists, but upon looking through the information that was returned we found that Spotify does not associate genres with individual songs. After doing a little bit of research, though, we learned that Spotify does associate genres with albums and artists.

        Solution: Some of the songs on the playlists were singles and did not belong to a specific album, so we decided to use each song's artist to associate the song with a genre. To do this we extracted the artist id for each song from the information that was returned by Spotipy's playlist_items method. Then we used the artist id as the parameter for Spotipy's artist method, which returns information about an artist, including genres.

    b. After we were able to get genres using Spotipy's artist method, we discovered that each artist had up to three associated genres.

        Solution: In order to narrow each song down to one genre, we outputted all of the associated genres for each song in the playlist and examined each set of genres. By doing this, we determined that there were no vast differences between the genres, and we decided to use the first listed genre in each cluster.

    c. When examining all of the genres, we also found that many of the genres themselves were extremely specific and niche. For example, rather than a genre being “hip hop”, it would be “kentucky underground hip hop”. Due to the specificity of the genres (as listed on Spotify), it would be very difficult to achieve our project goals.

Solution: To standardize the genres extracted from Spotify, we did some research and created a list of broad categories of genres, including “Other” to account for any outliers. We used this list to create the Genres table in our database that we would use to associate with the other tables through a primary key in the Genres table and foreign keys in the other tables. We then wrote code to select the genres using a database cursor object and then to loop through the genres in a list to determine if the genre name (as listed in our list of broad genres) was within each song’s specific genre from Spotify. If it was, we broke out of the loop and reassigned the song’s genre to the general name, otherwise the loop would continue to the next genre in the list, and if none of the general genres were within the more specific ones then we assigned the genre to “Other”. This also allowed us to examine the number of sub genres within each genre and create an additional visualization displaying our results.

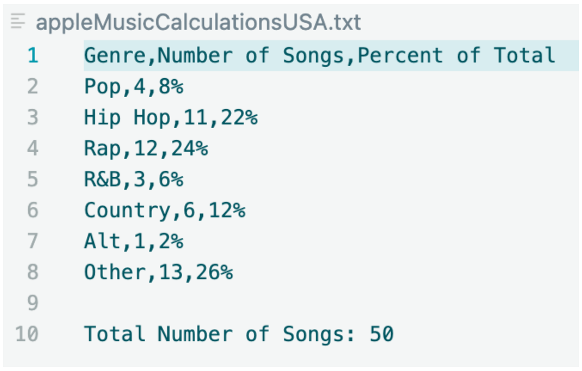
3. Another problem that we faced had to do with comparing the distribution of genres across the spotify and billboard charts. After we had already begun the project, we found that Kworb does not contain any genre data (i.e., it is not a part of its HTML). We had already begun writing the code to scrape the Kworb website and finished writing the code for Spotipy when we realized this, so we decided to try and find a solution to this problem rather than come up with a new research question and project plan.

   Solution: The solution that we figured out was to still scrape the website for the songs, and then use the data we scraped in conjunction with Spotipy’s search method. Since Spotify does not associate genres with individual songs, we scraped the artists of each song from the website, and then used the artists (in string format) as the query in Spotipy’s search function. The search function returns results from searching for something (the query) in Spotify. We limited the search results to return one item and then extracted the artist’s genre from that result.

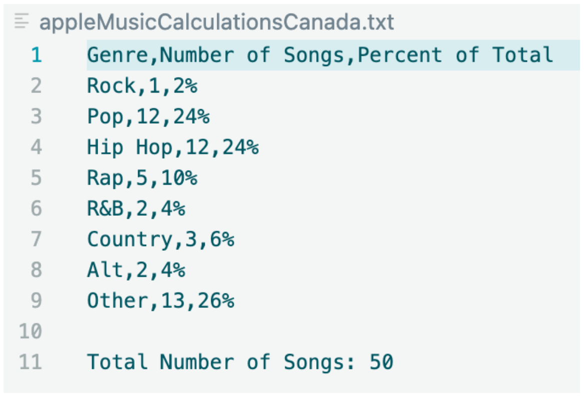
4. The final problem that we faced (when all of our code had been working properly) was that our Spotipy object stopped working. When our code got to a line that called a method with the Spotipy object we had created, it would stall indefinitely and return no error. This was particularly an issue because we had only stored half of our final data in our database at this point.

   Solution: After doing some research, our hypothesis was that this was occurring due to a rate-limiting problem with how many requests we had made using our original Spotipy credentials (client id and client secret). To work around this, we created a new app in the Spotify dashboard, which generated a new client id and new client secret. We then used these new credentials to read in the rest of our data to our database.

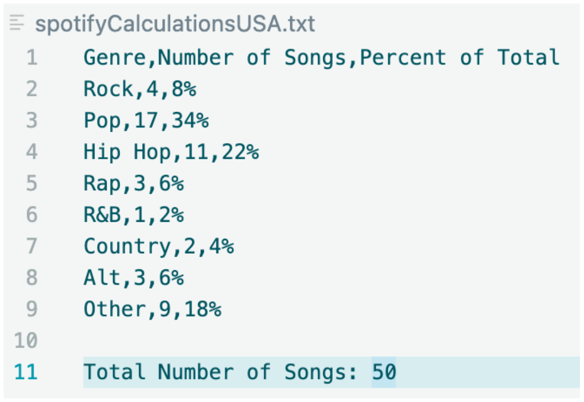
Calculations


appleMusicCalculationsUSA.txt

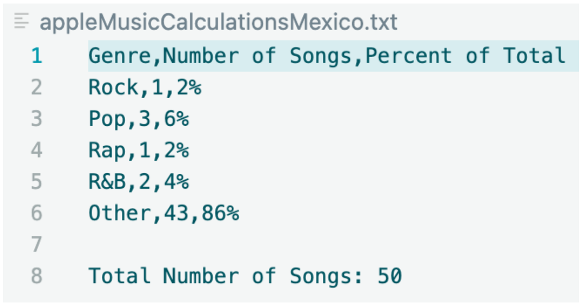
```
Genre,Number of Songs,Percent of Total
Pop,4,8%
Hip Hop,11,22%
Rap,12,24%
R&B,3,6%
Country,6,12%
Alt,1,2%
Other,13,26%

Total Number of Songs: 50
```


appleMusicCalculationsCanada.txt

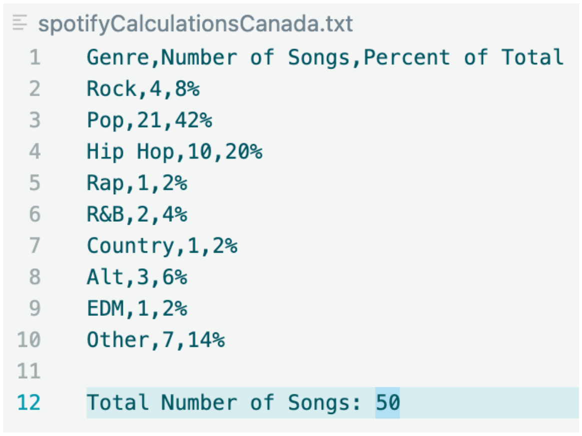
```
Genre,Number of Songs,Percent of Total
Rock,1,2%
Pop,12,24%
Hip Hop,12,24%
Rap,5,10%
R&B,2,4%
Country,3,6%
Alt,2,4%
Other,13,26%

Total Number of Songs: 50
```


spotifyCalculationsUSA.txt

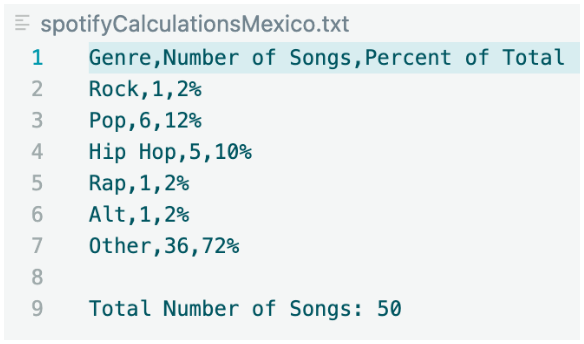
```
Genre,Number of Songs,Percent of Total
Rock,4,8%
Pop,17,34%
Hip Hop,11,22%
Rap,3,6%
R&B,1,2%
Country,2,4%
Alt,3,6%
Other,9,18%

Total Number of Songs: 50
```


appleMusicCalculationsMexico.txt

```
Genre,Number of Songs,Percent of Total
Rock,1,2%
Pop,3,6%
Rap,1,2%
R&B,2,4%
Other,43,86%

Total Number of Songs: 50
```

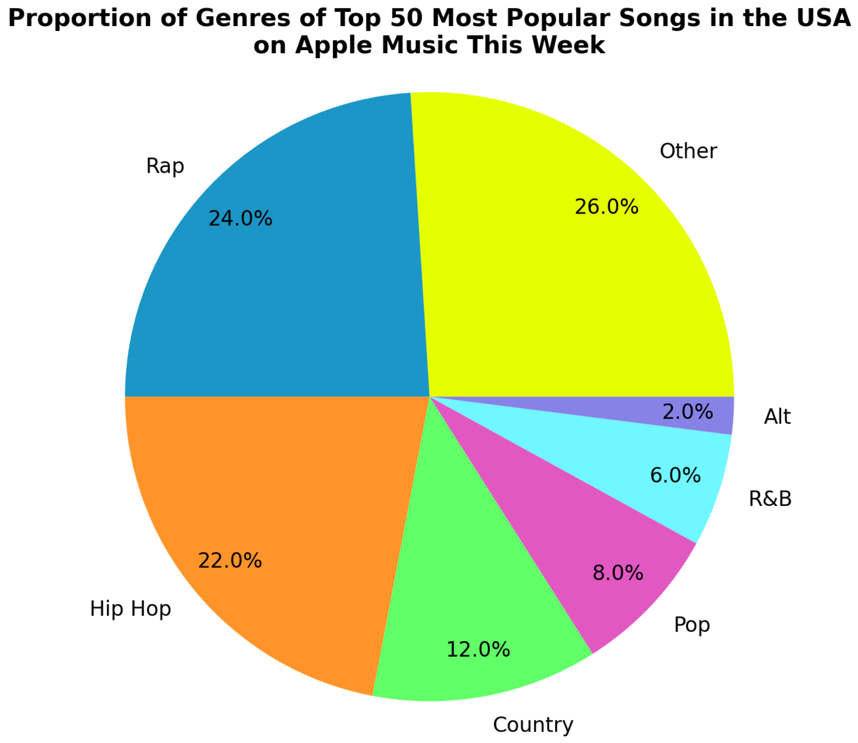

spotifyCalculationsCanada.txt

```
Genre,Number of Songs,Percent of Total
Rock,4,8%
Pop,21,42%
Hip Hop,10,20%
Rap,1,2%
R&B,2,4%
Country,1,2%
Alt,3,6%
EDM,1,2%
Other,7,14%

Total Number of Songs: 50
```

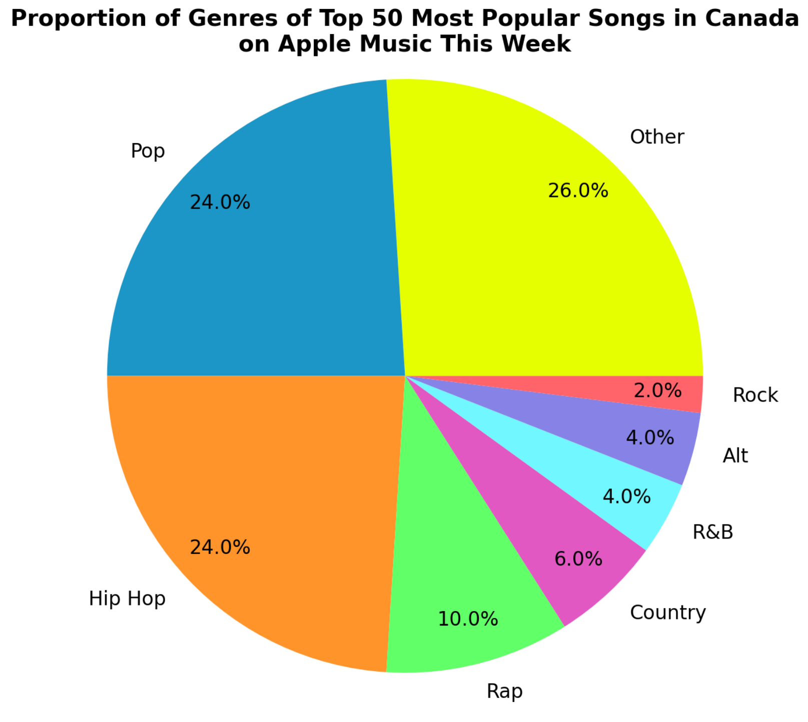

spotifyCalculationsMexico.txt

```
Genre,Number of Songs,Percent of Total
Rock,1,2%
Pop,6,12%
Hip Hop,5,10%
Rap,1,2%
Alt,1,2%
Other,36,72%

Total Number of Songs: 50
```

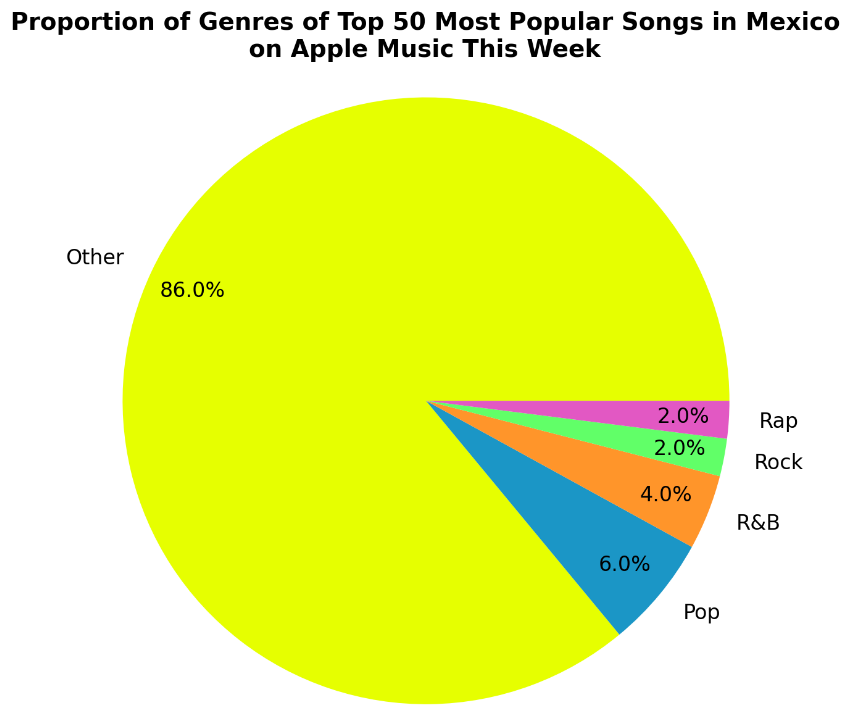
Visualizations



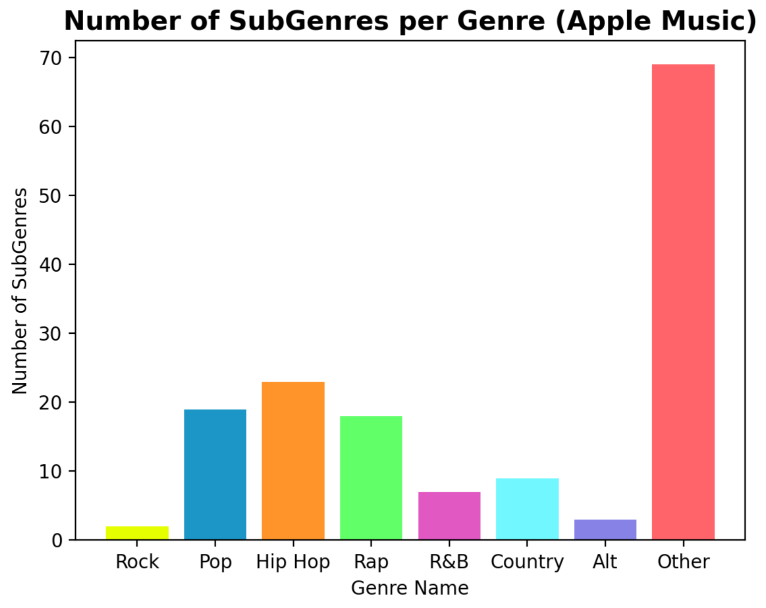
Genres with no songs in the top 50 this week: Rock, Classical, EDM, Jazz



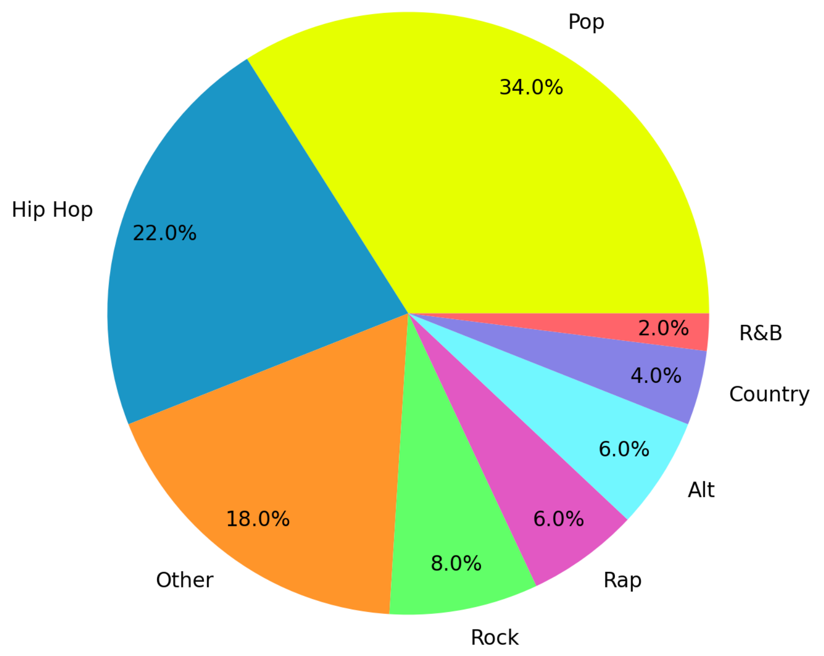
Genres with no songs in the top 50 this week: Classical, EDM, Jazz



Genres with no songs in the top 50 this week: Hip Hop, Country, Alt, Classical, EDM, Jazz

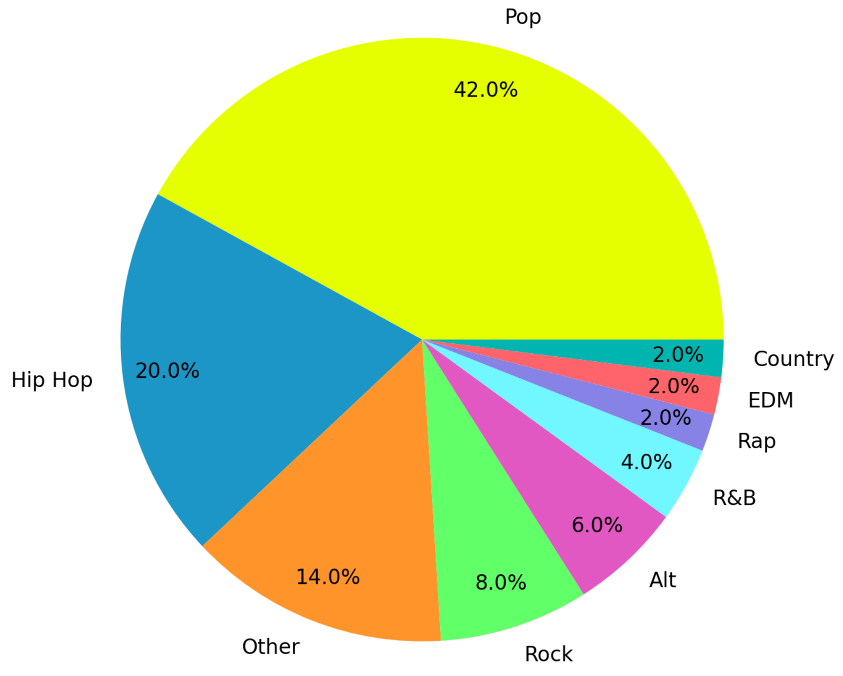


Proportion of Genres of Top 50 Most Popular Songs in the USA on Spotify This Week

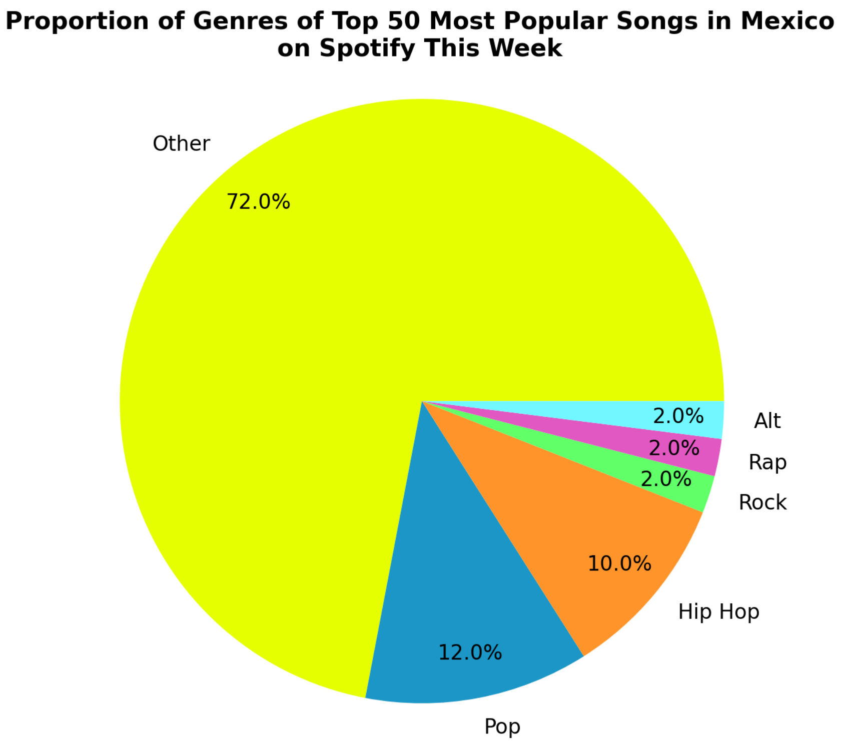


Genres with no songs in the top 50 this week: Classical, EDM, Jazz

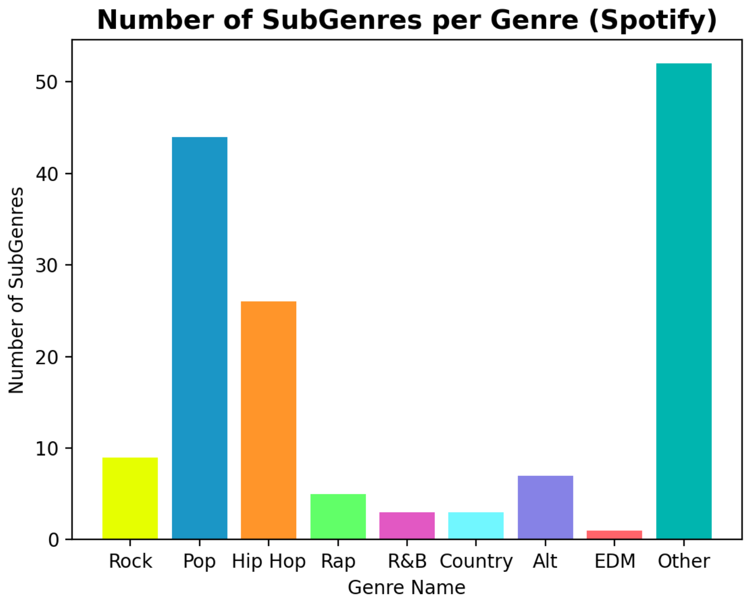
Proportion of Genres of Top 50 Most Popular Songs in Canada on Spotify This Week



Genres with no songs in the top 50 this week: Classical, Jazz



Genres with no songs in the top 50 this week: R&B, Country, Classical, EDM, Jazz



Instructions For Running Code

Order to Follow Instructions:

1. Go through spotify.py instructions to run spoitfy.py first
2. Go through apple_music.py instructions to run apple_music.py second
3. Go through genre_counts.py instructions to run genre_counts.py last

**NOTE: In order to run spotify.py and apple_music.py to collect the data from their original sources, you must have a separate file within your directory called ‘secrets.txt’ that contains a client id for a spotify application on the first line, and a client secret on the second line.

spotify.py

1. Scroll to the bottom of the file to access the main() function
2. To collect and begin to store data in the database, comment out the last two sections of the main() function that begin with “#WRITE CALCULATED DATA…” and “#CREATE PIE CHARTS…” (lines 247-261)
3. Run the file at least 12 times to collect and store all of the data in the database
4. To write the data calculations to txt files, uncomment the penultimate section of the main() function that begins with “#WRITE CALCULATED DATA…” (lines 247-253)
5. Run the file once to write the calculations to files
6. To see the visualizations, uncomment the last section of the main() function that begins with “#CREATE PIE CHARTS…” (lines 255-261)
7. Run the file once to create the visualizations

apple_music.py

1. Scroll to the bottom of the file to access the main() function
2. To collect and begin to store data in the database, comment out the last two sections of the main() function that begin with “#WRITE CALCULATED DATA…” and “#CREATE PIE CHARTS…” (lines 233-247)
3. Run the file at least 12 times to collect and store all of the data in the database
4. To write the data calculations to txt files, uncomment the penultimate section of the main() function that begins with “#WRITE CALCULATED DATA…” (lines 233-239)
5. Run the file once to write the calculations to files
6. To see the visualizations, uncomment the last section of the main() function that begins with “#CREATE PIE CHARTS…” (lines 241-247)
7. Run the file once to create the visualizations

genre_counts.py

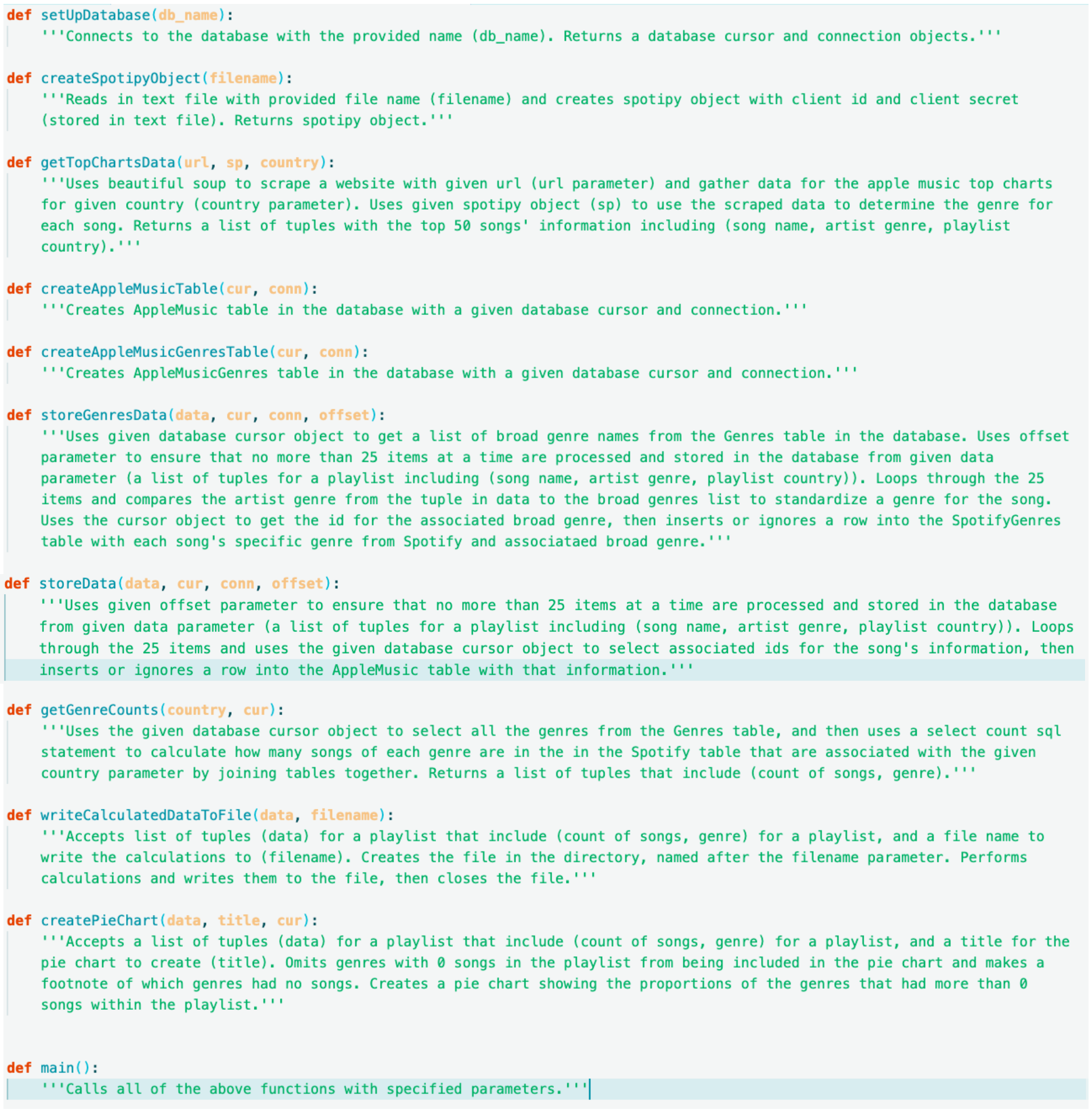
1. Run the file once to create the visualizations

Documentation

spotify.py

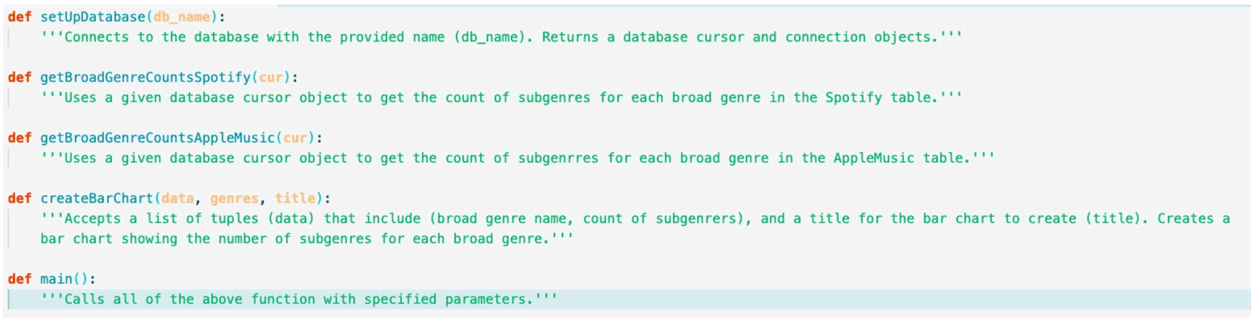
```python
def setUpDatabase(db_name):
    '''Sets up the database with the provided name (db_name). Returns a database cursor and connection objects.'''

def createSpotipyObject(filename):
    '''Reads in text file with provided file name (filename) and creates spotipy object with client id and client secret
    (stored in text file). Returns spotipy object.'''

def getPlaylistData(pid, sp):
    '''Uses spotipy object passed in as a parameter (sp) to get information about which country a playlist is for using
    playlist id passed in as a parameter (pid). Uses spotipy object to get information about all the songs in a playlist.
    Returns a list of tuples that include (song name, artist genre, playlist country).'''

def createGenresTable(genres, cur, conn):
    '''Creates Genres table in the database with a given list of genres, database connection, and cursor.'''

def createSpotifyGenresTable(cur, conn):
    '''Creates SpotifyGenres table in the database with a given database cursor and connection. '''

def createCountriesTable(countries, cur, conn):
    '''Creates Countries table in the database with a given list of countries, database connection, and cursor.'''

def createSpotifyTable(cur, conn):
    '''Creates Spotify table in the database with a given database cursor and connection.'''

def storeGenresData(data, cur, conn, offset):
    '''Uses given database cursor object to get a list of broad genre names from the Genres table in the database. Uses offset
    parameter to ensure that no more than 25 items at a time are processed and stored in the database from given data
    parameter (a list of tuples for a playlist including (song name, artist genre, playlist country)). Loops through the 25
    items and compares the artist genre from the tuple in data to the broad genres list to standardize a genre for the song.
    Uses the cursor object to get the id for the associated broad genre, then inserts or ignores a row into the SpotifyGenres
    table with each song's specific genre from Spotify and associataed broad genre.'''

def storeData(data, cur, conn, offset):
    '''Uses given offset parameter to ensure that no more than 25 items at a time are processed and stored in the database
    from given data parameter (a list of tuples for a playlist including (song name, artist genre, playlist country)). Loops
    through the 25 items and uses the given database cursor object to select associated ids for the song's information, then
    inserts or ignores a row into the Spotify table with that information.'''

def getGenreCounts(country, cur):
    '''Uses the given database cursor object to select all the genres from the Genres table, and then uses a select count sql
    statement to calculate how many songs of each genre are in the in the Spotify table that are associated with the given
    country parameter by joining tables together. Returns a list of tuples that include (count of songs, genre).'''

def writeCalculatedDataToFile(data, filename):
    '''Accepts list of tuples (data) for a playlist that include (count of songs, genre) for a playlist, and a file name to
    write the calculations to (filename). Creates the file in the directory, named after the filename parameter. Performs
    calculations and writes them to the file, then closes the file.'''

def createPieChart(data, title, cur):
    '''Accepts a list of tuples (data) for a playlist that include (count of songs, genre) for a playlist, and a title for the
    pie chart to create (title). Omits genres with 0 songs in the playlist from being included in the pie chart and makes a
    footnote of which genres had no songs. Creates a pie chart showing the proportions of the genres that had more than 0
    songs within the playlist.'''

def main():
    '''Calls all of the above functions with specified parameters.'''
```

apple_music.py



```python
def setUpDatabase(db_name):
    '''Connects to the database with the provided name (db_name). Returns a database cursor and connection objects.'''

def createSpotipyObject(filename):
    '''Reads in text file with provided file name (filename) and creates spotipy object with client id and client secret
    (stored in text file). Returns spotipy object.'''

def getTopChartsData(url, sp, country):
    '''Uses beautiful soup to scrape a website with given url (url parameter) and gather data for the apple music top charts
    for given country (country parameter). Uses given spotipy object (sp) to use the scraped data to determine the genre for
    each song. Returns a list of tuples with the top 50 songs' information including (song name, artist genre, playlist
    country).'''

def createAppleMusicTable(cur, conn):
    '''Creates AppleMusic table in the database with a given database cursor and connection.'''

def createAppleMusicGenresTable(cur, conn):
    '''Creates AppleMusicGenres table in the database with a given database cursor and connection.'''

def storeGenresData(data, cur, conn, offset):
    '''Uses given database cursor object to get a list of broad genre names from the Genres table in the database. Uses offset
    parameter to ensure that no more than 25 items at a time are processed and stored in the database from given data
    parameter (a list of tuples for a playlist including (song name, artist genre, playlist country)). Loops through the 25
    items and compares the artist genre from the tuple in data to the broad genres list to standardize a genre for the song.
    Uses the cursor object to get the id for the associated broad genre, then inserts or ignores a row into the SpotifyGenres
    table with each song's specific genre from Spotify and associataed broad genre.'''

def storeData(data, cur, conn, offset):
    '''Uses given offset parameter to ensure that no more than 25 items at a time are processed and stored in the database
    from given data parameter (a list of tuples for a playlist including (song name, artist genre, playlist country)). Loops
    through the 25 items and uses the given database cursor object to select associated ids for the song's information, then
    inserts or ignores a row into the AppleMusic table with that information.'''

def getGenreCounts(country, cur):
    '''Uses the given database cursor object to select all the genres from the Genres table, and then uses a select count sql
    statement to calculate how many songs of each genre are in the in the Spotify table that are associated with the given
    country parameter by joining tables together. Returns a list of tuples that include (count of songs, genre).'''

def writeCalculatedDataToFile(data, filename):
    '''Accepts list of tuples (data) for a playlist that include (count of songs, genre) for a playlist, and a file name to
    write the calculations to (filename). Creates the file in the directory, named after the filename parameter. Performs
    calculations and writes them to the file, then closes the file.'''

def createPieChart(data, title, cur):
    '''Accepts a list of tuples (data) for a playlist that include (count of songs, genre) for a playlist, and a title for the
    pie chart to create (title). Omits genres with 0 songs in the playlist from being included in the pie chart and makes a
    footnote of which genres had no songs. Creates a pie chart showing the proportions of the genres that had more than 0
    songs within the playlist.'''


def main():
    '''Calls all of the above functions with specified parameters.'''
```

genre_counts.py



```python
def setUpDatabase(db_name):
    '''Connects to the database with the provided name (db_name). Returns a database cursor and connection objects.'''

def getBroadGenreCountsSpotify(cur):
    '''Uses a given database cursor object to get the count of subgenres for each broad genre in the Spotify table.'''

def getBroadGenreCountsAppleMusic(cur):
    '''Uses a given database cursor object to get the count of subgenrres for each broad genre in the AppleMusic table.'''

def createBarChart(data, genres, title):
    '''Accepts a list of tuples (data) that include (broad genre name, count of subgenrers), and a title for the bar chart to create (title). Creates a
    bar chart showing the number of subgenres for each broad genre.'''

def main():
    '''Calls all of the above function with specified parameters.'''
```

Resources Used

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
4-13	Needed to have a Spotify developer app client id and a client secret to utilize Spotipy	https://developer.spotify.com/documentation/general/guides/authorization/app-settings/	Yes, it provided instructions on how to set up a Spotify app and get the client id and client secret
4-13	Needed to identify ids from Spotify for playlists and artists	https://support.tunecore.com/hc/en-us/articles/360040325651-How-to-Find-my-Spotify-Artist-ID#:~:text=Your%20Spotify%20artist%20ID%20is,to%20the%20connect%20artist%20page.	Yes, it pointed to where the Spotify ids can be found in the URLs on the Spotify Web Player
4-14	Couldn't access Spotify Web API with Spotipy using get_access_token() method (like IA demonstrated)	https://medium.com/@maxtingle/getting-started-with-spotifys-api-spotipy-197c3dc6353b	Yes, used client_credentials_manager=SpotifyClientCredentials(client_id=cid, client_secret=secret) sp=spotipy.Spotify(client_credentials_manager=client_credentials_manager)
4-16	Wanted to sort the pie chart slices based on size	https://www.nxn.se/valent/making-nicer-looking-pie-charts-with-matplotlib (sorting the sizes)	No, because it messed up the coordination between sizes and labels. Instead I sorted the list of tuples including (label, size) by size before constructing the pie chart
4-17	Needed to stop labels on pie chart from overlapping, and wanted to remove genres with 0 songs from data being displayed in pie chart to do so	https://note.nkmk.me/en/python-list-clear-pop-remove-del/ (list comprehension)	Yes, it provided information on how to perform list comprehension to get a new list with only genres and counts with at least 1 song
4-17	Wanted to create a string with the genres that had 0 songs to include as a footnote in the pie chart	https://www.simplilearn.com/tutorials/python-tutorial/list-to-string-in-python#:~:text=To%20convert%20a%20list%20to%20a%20string%2C%20use%20Python%20List,and%20retu	Yes, it provided information on how to use the .join() function on lists

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
		rn%20it%20as%20output. (using join function)	
4-17	Wanted to add footnote with genres that had 0 songs in the top 50 (rather than having them be a slice in the pie)	https://matplotlib.org/3.5.0/api/_as_gen/matplotlib.pyplot.annotate.html	Yes, it provided documentation for the .annotate() function to add a footnote to a Matplotlib visualization
4-17	Needed to develop color palette for the visualizations	https://learnui.design/tools/data-color-picker.html#palette http://colormind.io/	Yes, both links gave recommendations for hex code colors for a palette given two end hex code colors
4-17	Needed to stop the displaying percentages on the pie chart slices from overlapping a lot (especially on smaller slices)	https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html	Yes, it provided documentation for the .pie() function's pctdistance attribute to change the amount of distance between the displaying percent and the center of each pie slice
4-17	Wanted the footnote on the pie chart to have a smaller font than the rest of the labels	https://stackoverflow.com/questions/14643891/different-font-sizes-in-the-same-annotation-of-matplotlib	Yes, it provided an example of how to use the fontsize attribute with the .annotate() function
4-17	Needed to stop the title from overflowing the window of the visualizations	https://www.discoverbits.in/492/python-how-to-wrap-title-in-matplotlib-plot-to-fit-long-title	Yes, used from textwrap import wrap plt.title('\n'.join(wrap(title,60)), fontsize=14)
4-17	Wanted to increase size of pie chart figure window	https://pythonguides.com/matplotlib-pie-chart/ (Matplotlib pie chart increase size)	Yes, used matplotlib.pyplot.figure(figsize=(x, y))
4-17	Wanted to increase size of labels on pie chart	https://stackoverflow.com/questions/7082345/how-to-set-the-labels-size-on-a-pie-chart-in-python (textprops)	Yes, it provided an example on how to use the textprops attribute for the .pie() function for fontsize
4-17	Wanted to make the visualizations' titles bold	https://pythonguides.com/matplotlib-title-font-size/ (Matplotlib title font size bold)	Yes, it gave example on how to use the fontweight attribute in the .title() function

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
4-17	Needed to save data scraped with Beautiful Soup to a csv file to ensure code was working because there were complications saving to database	https://www.youtube.com/watch?v=kH2IgDUilcs&list=PLI1n_4fW81108IGLS3GxDA6Qs7fDkrUdu&index=2	Yes, it provided a tutorial for creating .txt files. We ended up changing this later, though, to keep the code more consistent across files
4-20	Needed to link tables created with csv to main database	https://www.youtube.com/watch?v=kzgZAd5_wt8&list=PLI1n_4fW81108IGLS3GxDA6Qs7fDkrUdu&index=3	No, this became too complicated of a solution. We figured out an easier way to create our tables and store data in the database without using csv
4-23	Wanted to create a grouped bar chart as another visualization	https://matplotlib.org/3.5.0/gallery/lines_bars_and_markers/barchart.html	Yes, it provided an example of how to create a grouped bar chart with labels using Matplotlib, but we ultimately ended up creating a different type of additional visualization
4-23	Wanted to round decimal float number to whole number	https://www.w3schools.com/python/ref_func_round.asp	Yes, used round(number, digits)
4-23	Needed to use Spotipy to find genres for Beautiful Soup scraped songs	https://spotipy.readthedocs.io/en/2.19.0/	Yes, it provided the documentation for Spotipy's .search() function
4-24	Needed to join multiple tables together in a SQL statement	https://www.sqlshack.com/learn-sql-join-multiple-tables/	Yes, it provided a description and example of how to construct INNER JOIN statements
4-24	Needed to count occurrences of ids in a table	https://www.codegrepper.com/code-examples/sql/count+occurrences+sql	Yes, it provided an example of how to use a SELECT COUNT statement with GROUP BY
4-24	Needed further information on how to create a bar graph using Matplotlib	https://datatofish.com/bar-chart-python-matplotlib/	Yes, it provided steps on how to make a bar graph with Matplotlib.

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
4-24	Wanted to make all the bars on our bar graph different colors that match the color scheme used in our pie charts	https://www.tutorialkart.com/matplotlib-tutorial/matplotlib-pyplot-bar-plot-different-colors-for-bars/#:~:text=To%20set%20different%20colors%20for,parameter%20of%20bar()%20function.&text=Of%20course%2C%20there%20are%20other,only%20color%20parameter%20is%20given.	Yes, it provided an example of how to set a list of color codes and then set the color attribute in the .bar() function to equal that list.