CS 181 Final Project

Spotify API and Billboard Webscraping

Alex Tubbs and Hieu Nguyen Notebook 2

In this notebook, we want to use the token to get the data from provider, organize it appropriately using the tidy data knowledge we learned at the beginning of the semester, and finally put it into an SQL table using third normal form constraints

First, we import necessary libraries. Json, requests, etree, io, re, lxml.html and pandas libraries are used for getting the data from Spotify using tokens, whereas, sqlalchemy library is used to put dataframes into SQL database

In [1]:
```
import json
import requests
from IPython.core.debugger import set_trace
from lxml import etree
import pandas as pd
import re
import sqlalchemy as sa
with open("creds.json", "r") as file:
    creds = json.load(file)
import io
import lxml.html as lh
import pandas as pd
```

Getting data from Spotify API

In the following functions, we would like to use our creds.json file, which stored the refresh token of users, to get the access token of users

In [2]:
```
def getRefreshToken():
    """
    This function will use the creds dictionary of user to retrieve their refresh token
    Parameter: None
    Return: codemap: a dictionary of users to be keys and their refresh tokens to be value
    """
    codemap = {}
    for key in creds['spotify']['users']:
        codevalue = creds['spotify']['users'][key][0]
        codemap[key] = codevalue
    return codemap

codemap = getRefreshToken()
codemap
```

Out[2]:
```
{'Alex': 'AQCG8g28_14TzLvnQ6hn3zqkB-5PzRXfPTaFIBvcHNcDx4TpCvVcg-WCFRhcJHv3iAUUg7-nWoV_Pva8DKvPO5_ThIFQV
hBXHY_R-_KT7LvT-RhLMou1bqdEqzCgVv1fB90',
 'AlexB': 'AQA3tbU6NEslw2HMPoZkO13QAcYnm_Ugcze-1R0kC9_6hfLFjrlhrORD9gFXbfegHC4PVdhUhrty8opkhXSl9eSptRbj
6H2u_9gJihY7z65o0V9kvJqMf5aq9azrBWhBA3U',
 'Danish': 'AQDPJm8BUVz4qYqaV3leg6SAqYtOFQtu3yFCaEWRpDbA-Jqe_ZzYgyCny1H57VBwmo5fB80eTs3HFbs7eEGx99zUm_o
VFg5IpM8DvyDlG-3c3Jv9obGPkEmtaHHKJQK8e6s',
 'Hieu': 'AQA0ZnCmatRTcseIjmaStfbTdEArZYuyULrg7IOTBxv2QJp-75Tn4Bhk17zkXwiMxBVHoLmCfJKTQLHhtpYVJZufNJPmK
RGE59XcnkenYzGWl2IbABooXxjpHaJAwAhYpyo',
 'Josh': 'AQCsm-gD6IIhUNsPukJoTzxxb2tLFLw-CM9d1VEV-C-DgGmqk7u-i56kUCtB6JvRXJ-8S2WjizQT1I8IQfZ1Y0QGx7LFF
3jYZ85mRCCUjkLKEnqCNcK7xyJoxojd8ZzPgso',
 'Kush': 'AQBr7OAYUoCjnlKJ18Qa0Njg_Nw2pp7YgW2CnAdM4E3ZEaOV3DBTFAGxilBLhbSxAVkYS_xxxIk5CFli8o_fEHrIUc_Lg
Bqd05QVGrdtfQ3aGXCD1zGXkcQfSTCOL3lULI',
 'Noah': 'AQBTCuso2UWETiZbpZiBhHm25BV5Hg5dHVpBBFz_VN6aF73IrIqBpRv7HF6uMYJ8XubHZzUE2v1ET_Bf6dLb6CdvA-LNK
_SpTJCeVbzoR6ddhaS6gIUEPXcEktIYgxYH7zI'}
```

In [3]:
```
def getTokenmap(codemap):
    """
    This function will use the codemap achieved above to get user's access token for data acquiring later
    Parameter: codemap: a dictionary of users to be keys and their refresh tokens to be value
    Return: tokenmap: a dictionary of users to be keys and access token to be value
    """
    protocol = "https"
    location = "accounts.spotify.com"
    auth_resource = "/api/token"
    access_fmt = "{}://{}{}"
    accessurl = access_fmt.format(protocol, location, auth_resource)

    dataD = {}
    dataD['client_id'] = creds['spotify']['appid']
    dataD['client_secret'] = creds['spotify']['appsecret']
    dataD['redirect_uri'] = creds['spotify']['redirect_uri']
    dataD['grant_type'] = "refresh_token"
    resp = requests.post(accessurl, data=dataD)

    tokenmap = {}
    for user, code in codemap.items():
        dataD['refresh_token'] = code
        resp = requests.post(accessurl, data = dataD)
        if resp.status_code == 200:
            retval = resp.json()
            if 'access_token' in retval:
                tokenmap[user] = [code, retval['access_token']]
            else:
                print('No access token found in result:', str(retval))
        else:
            print("HTTP error on exhange of code for token", str(resp.status_code), str(resp.text))
    return tokenmap

tokenmap = getTokenmap(codemap)
tokenmap
```

```
HTTP error on exhange of code for token 400 {"error":"invalid_grant","error_description":"Invalid refre
sh token"}
```

Out[3]:
```
{'Alex': ['AQCG8g28_14TzLvnQ6hn3zqkB-5PzRXfPTaFIBvcHNcDx4TpCvVcg-WCFRhcJHv3iAUUg7-nWoV_Pva8DKvPO5_ThIFQ
VhBXHY_R-_KT7LvT-RhLMou1bqdEqzCgVv1fB90',
  'BQCH0bFmNwMMleOef9rPUCYOml9NvtMGz6r8fIYe1_3Xp9DyReRTrXi741ykOwQzcwM3TrYwYYLjpXIQlbwses7oYSG4W8N03yp4
VYnk1jsOcuKPMgBg3Ez6WwsWnjx1b0DEJ5h0DRvPm_L2knWWse84dQsdjU_0ADwIpEuj'],
 'AlexB': ['AQA3tbU6NEslw2HMPoZkO13QAcYnm_Ugcze-1R0kC9_6hfLFjrlhrORD9gFXbfegHC4PVdhUhrty8opkhXSl9eSptRb
j6H2u_9gJihY7z65o0V9kvJqMf5aq9azrBWhBA3U',
  'BQBJ3B6_9dddafhyUt6l6LzH7P2s3GiduPYo28dn9ioDBseWEDYs97HwFGCypSkwjzcaLE-dURDD2Lm436qRVNFOB5YOFwQsMhz-
fUeBjerMv1oEWbceXUDYP2IFQTP_cTBotMQ2A5604xZFrviDvNOxPZhrLZ4V6bU'],
 'Danish': ['AQDPJm8BUVz4qYqaV3leg6SAqYtOFQtu3yFCaEWRpDbA-Jqe_ZzYgyCny1H57VBwmo5fB80eTs3HFbs7eEGx99zUm_
oVFg5IpM8DvyDlG-3c3Jv9obGPkEmtaHHKJQK8e6s',
  'BQCOgmnFMb51JndVbnym3Y9KN7gLoHYfgxGpETqwcO3-WNLweh1UoaftsoahIbd2JFPu2Dh7g-iq0xQVyULg2aZBnvybyzWsRCf5
5w_zbnsOPbDFxYOffnHjhkrgvrKI5ujDTKZ6ZHPZhmz__1L1UeLGsX0tZ9Eo3eG5gq0b2vT9uJyVzpYxuTtyNCU'],
 'Hieu': ['AQA0ZnCmatRTcseIjmaStfbTdEArZYuyULrg7IOTBxv2QJp-75Tn4Bhk17zkXwiMxBVHoLmCfJKTQLHhtpYVJZufNJPm
KRGE59XcnkenYzGWl2IbABooXxjpHaJAwAhYpyo',
  'BQAscC4_8z7CgcutAd122XSBpO0zLpvoSnR-jQBFCELLq2eqP8zgxpYybzwOR_dHaGqAn0sb4W7YzXZW0mDLuPPialeYPL_kApHc
hx39w7hbS-SzLkYmxYiDUNdmzcPtPq2Z-QAGWgnlaoZecvFfU7487TO4wj1rb2_fOY3VE0c8Tux_mjkEiT87414'],
 'Josh': ['AQCsm-gD6IIhUNsPukJoTzxxb2tLFLw-CM9d1VEV-C-DgGmqk7u-i56kUCtB6JvRXJ-8S2WjizQT1I8IQfZ1Y0QGx7LF
F3jYZ85mRCCUjkLKEnqCNcK7xyJoxojd8ZzPgso',
  'BQABvFkhewmk6v6e0ViFG4YL-FA6tyYLM9gtvTQjVktU4NxsUDaJdV4ER51VIRRgRDrwDD6jfOkY-Ti7fM7xbI2ws90fKeiTfJl0
s5KqRz99bSknYc-LXvQ_aPdE7wD05SB7HsJtcpeOozLnzvJYYO255GIF4tbGJds-FA0'],
 'Noah': ['AQBTCuso2UWETiZbpZiBhHm25BV5Hg5dHVpBBFz_VN6aF73IrIqBpRv7HF6uMYJ8XubHZzUE2v1ET_Bf6dLb6CdvA-LN
K_SpTJCeVbzoR6ddhaS6gIUEPXcEktIYgxYH7zI',
  'BQAOTPqXBE898KPX84EqXZuEyNP4SG0L-lMJVpjq7FGobWvCYUBKzWAUuUt1i12SU0BpwgoTsOCLnE0pmjXNMYPfJazIwSofn5-X
g_VKQk7IbzXUdR5FjWZPoK4boVU5GHD655OFbD9rHs-AIiSvoLaLspQ3xKTZ0KOi']}
```

To use the refresh token it was only minor adjustments from getting the original token. We had to change one part of the body of our post to reflect that we were using a refresh token rather than a code. We also always had the same refresh token, so we made sure to save that first in the token map, and then had the token as the second item in the list.

In [4]:
```
def listToString(array):
    """
    This function will turn a list of elements into a string of these elements, seperated by commas
    Parameter: array: a list of elements
    Return: s: a string of these elements separated by commas
    """
    s = ''
    for i in range(len(array)):
        s = s + array[i]+ ', '
    return s[:-2]
```

Once we have the access token of users, we can go ahead to use these access token to retrieve data from provider. The only challenging part we faced in this step was the process to figure out how to correctly format the access token, in which we didn't think about the Bearer token type at first. Moreover, we also used the token as access URL parameter, which was not true. Then, by using the curl to correctly fetch the data in command line, we were able to find out that the problem was on our code but not on the data provider. By doing researches, we found out that the token is a Bearer type and the token needs to be specified under URL header, not URL parameter.Once we solved this problem, we were able to get the data in JSON format.

Our next challenge was that the JSON-formatted data was too overwhelming and messy. Therefore, we need to carefully examine what is the composite of the data and what we think would be necessary for our analysis later. Then, by traversing the JSON data, we were able to correctly put the data into pandas dataframes

The getTopArtist and the getTopTracks function below are composed of three parts, which are using tokens to get data, traversing JSON format data, and turning them into dataframes. The getTopArtist function below return 2 dataframes because when we examined the JSON, we realized that artist genre was put under the format of a list and we wanted them to be single strings for later analysis. Therefore, the second dataframes served as a melted dataframe with overlapped artist with multiple genres

In [23]:
```
def getTopArtist(tokenmap):
    """
    This function will take a dictionary of access tokens and use them for URL header to request data fro
    It will also traverse JSON format data and turn it into dataframes
    Parameter: tokenmap: a dictionary of users to be keys and access token to be value
    Return: df1, df2: the two dataframes
    """

    protocol = "https"
    location = "api.spotify.com"
    auth_resource = "/v1/me/top/artists"
    access_fmt = "{}://{}{}"
    accessurl = access_fmt.format(protocol, location, auth_resource)

    final = []

    for key in tokenmap:
        token = tokenmap[key][1]
        urlquery = {'Authorization': 'Bearer '+token}
        paramD = {'limit': 50}
        session = requests.Session()
        p = requests.get(accessurl, headers=urlquery,params = paramD)
        jdict = p.json()


        for item in jdict:
            #total,limit,offset,href,previous

            if item == 'items':
                for i in range(len(jdict[item])):
                    row = []
                    row.append(jdict[item][i]['followers']['total'])
                    row.append(listToString(jdict[item][i]['genres']))
                    row.append(jdict[item][i]['id'])
                    row.append(jdict[item][i]['name'])
                    row.append(jdict[item][i]['popularity'])
                    final.append(row)

        ArtistGenre = []
        for j in range(len(final)):
            temp = final[j][1].split(',')
            for item in temp:
                row = []
                row.append(final[j][2])
                row.append(final[j][3])
                row.append(item)
                ArtistGenre.append(row)

    cols = ['ArtistiD','ArtistName','ArtistGenre']
    df2 = pd.DataFrame(ArtistGenre, columns = cols)

    colName = ['Followers','ArtistGenres','ArtistID','ArtistName','ArtistPopularity']
    df1 = pd.DataFrame(final,columns = colName)

    return df1,df2

TopArtist, ArtistGenre = getTopArtist(tokenmap)
TopArtist.head()
```

Out[23]:

	Followers	ArtistGenres	ArtistID	ArtistName	ArtistPopularity
0	3050233	dance pop, etherpop, indie poptimism, pop, pos...	26VFTg2z8YR0cCuwLzESi2	Halsey	85
1	502679	dance pop, indie poptimism, pop, post-teen pop	3LjhVl7GzYsza1biQjTpaN	Hayley Kiyoko	74
2	23365205	pop	6eUKZXaKkcviH0Ku9w2n3V	Ed Sheeran	94
3	9039430	modern rock, vegas indie	53XhwfbYqKCa1cC15pYq2q	Imagine Dragons	89
4	3934414	emo, modern rock, pop punk, vegas indie	20JZFwl6HVl6yg8a4H3ZqK	Panic! At The Disco	83

In [6]:
```
#maybe turn auth resource into
def getTopTracks(tokenmap):
    """
    This function will take a dictionary of access tokens and use them for URL header to request data fro
    It will also traverse JSON format data and turn it into dataframes
    Parameter: tokenmap: a dictionary of users to be keys and access token to be value
    Return: df: the returned dataframe
    """

    protocol = "https"
    location = "api.spotify.com"
    auth_resource = "/v1/me/top/tracks"
    access_fmt = "{}://{}{}"
    accessurl = access_fmt.format(protocol, location, auth_resource)

    final = []

    for key in tokenmap:
        token = tokenmap[key][1]
        urlquery = {'Authorization': 'Bearer '+token}
        paramD = {'limit': 50}
        session = requests.Session()
        p = requests.get(accessurl, headers=urlquery,params = paramD)

        if p.headers['Content-Type'][:16] == 'application/json' or p.headers['Content-Type'][:9] == 'text
            js = p.text
            jdict = p.json()


        for item in jdict:
            #total,limit,offset,href,previous
            if item == 'items':
                for i in range(len(jdict[item])):
                    row = []
                    row.append(jdict[item][i]['id'])
                    row.append(jdict[item][i]['artists'][0]['name'])
                    row.append(jdict[item][i]['artists'][0]['id'])
                    row.append(jdict[item][i]['name'])
                    row.append(jdict[item][i]['duration_ms'])
                    row.append(jdict[item][i]['popularity'])
                    row.append(jdict[item][i]['album']['id'])
                    row.append(jdict[item][i]['album']['name'])
                    row.append(jdict[item][i]['album']['release_date'])
                    final.append(row)

    colName = ['TrackID','ArtistName','ArtistID','TrackName','TrackDuration','TrackPopularity','AlbumID',
    df = pd.DataFrame(final,columns = colName)

    return df


TopTracks = getTopTracks(tokenmap)
TopTracks
```

	TrackID	ArtistName	ArtistID	TrackName	TrackDuration	TrackPopularity
3	4vTsOYAocjaslUONkx2YS3	Halsey	26VFTg2z8YR0cCuwLzESi2	Angel On Fire	194904	58
4	2WQn7Yvs728KZmmY6tgWqH	Halsey	26VFTg2z8YR0cCuwLzESi2	Eyes Closed	202438	66
5	3wqPinf9whHeT7y9EApaPM	Hayley Kiyoko	3LjhVl7GzYsza1biQjTpaN	Wanna Be Missed	195773	66
6	5btaVjrLBxTvXNmCv5DrW2	Hayley Kiyoko	3LjhVl7GzYsza1biQjTpaN	Curious	183280	68
7	11EDhDAVDtGPoSar6ootYA	Halsey	26VFTg2z8YR0cCuwLzESi2	Strangers	221205	67
8	5k38wzpLb15YgncyWdTZE4	G-Eazy	02kJSzxNuaWGqwubyUba0Z	Him & I (with Halsey)	268866	88

9	03PM8jkLPwI6cDcZBvwCSL	Hayley Kiyoko	3LjhVl7GzYsza1biQjTpaN	He'll Never Love You (UNLY)	231293	65

Getting data by webscraping Billboard

Once we are finished with getting data from Spotify API, we moved on to webscrappe Billboard

In [7]:
```
def getBillboardTree(chart):
    """Given a string locale giving a city and state, perform a Yelp search for a ranking
       of restaurants in that area.  If start is not specified, then perform the search
       for the default top ten.  If start is specified, add it as a search parameter to
       get the next 10 listings starting at index start.

       On success, the return should be an lxml html-parsed tree represented by the root element
       of that tree.

       On failure, None should be returned.
    """
    protocol = 'https'
    location = 'www.billboard.com'
    resource = '/charts'
    chart = chart

    urlfmt = "{}://{}{}/{}"
    url = urlfmt.format(protocol, location, resource, chart)

    resp = requests.get(url)
    if resp.status_code != 200:
        return None

    return lh.parse(io.BytesIO(resp.content)).getroot()
```

At first, retrieving the songs and artists from the billboard website appeared to be very easy. It was a simple x-path traversal to get to artists and songs, because they both fell under the div with a class chart-row_title. The problem arose when our lists weren't turning out the same length and we couldn't figure out why. The problem was if it was multiple artists for one song, the artist's name would be stored in a different tag (span instead of a) than single artists, so we were only getting single artists. To fix this, instead of specifying the tag, we just got the text of all children of the chart-row_title path. If it was only an artist, thats all it returned so we were good for the getTopArtists function. If there was both a song and an artist, like in the hot100, we had to return every other entry, starting at index 1, because we were getting song, artist, song, artist and we only wanted the artists.

In [8]:
```
def getSongArtists(tree):
    """
    This function uses xpath query to traverse the tree to get the song of artists
    Parameter: tree: an XML tree
    Return: TopA: a list of song titles
    """
    top100 = tree.xpath('//div[@class = "chart-row__title"]/*/text()')
    TopA = []
    for i in top100:
        i = i.replace("\n", "")
        TopA.append(i)
    return TopA[1::2]

def getTopArtists(tree):
    """
    This function uses xpath query to traverse the tree to get top artists
    Parameter: tree: an XML tree
    Return: TopA: a list of song artist
    """
    top100 = tree.xpath('//div[@class = "chart-row__title"]/*/text()')
    TopA = []
    for i in top100:
        i = i.replace("\n", "")
        TopA.append(i)
    return TopA

def getTop100Songs(tree):
    """
    This function uses xpath query to traverse the tree to get top 100 songs
    Parameter: tree: an XML tree
    Return: TopA: a list of top 100 songs
    """
    top100 = tree.xpath('//div[@class = "chart-row__title"]/h2/text()')
    TopA = []
    for i in top100:
        i = i.replace("\n", "")
        TopA.append(i)
    return TopA

def getTop100Rank(tree):
    """
    This function uses xpath query to traverse the tree to get top 100 songs by rank
    Parameter: tree: an XML tree
    Return: TopA: a list of top 100 songs by rank
    """
    top100 = tree.xpath('//div[@class = "chart-row__rank"]/span[@class = "chart-row__current-week"]/text(
    TopA = []
    for i in top100:
        i = i.replace("\n", "")
        TopA.append(i)
    return TopA
```

One other weird part of the html was when we got back the text of each artist or song, it would look like '\nHalsey\n', so for each of the above loops we also implemented a replace function, to get rid of those new line characters so that it would return just the artist, and make it easier to compare to Spotify.

In [9]:
```
def ArtistDF(rank, artist):
    '''
    This function takes in a list of ranks and artists in order, and returns a dataframe of the artists w
    '''
    D = {'rank':rank,'Artist':artist}
    dfTopArtists = pd.DataFrame.from_dict(D, orient='columns')
    return dfTopArtists

artiststree = getBillboardTree('greatest-hot-100-artists')
artistRank = getTop100Rank(artiststree)
artists = getTopArtists(artiststree)
dfTopArtists = ArtistDF(artistRank, artists)
dfTopArtists.head(10)
```

Out[9]:

	Artist	rank
0	The Beatles	1
1	Madonna	2
2	Elton John	3
3	Elvis Presley	4
4	Mariah Carey	5
5	Stevie Wonder	6
6	Janet Jackson	7
7	Michael Jackson	8
8	Whitney Houston	9
9	The Rolling Stones	10

Put data in SQL database

In [10]:
```
def getCreds(filename, subset, defaults = {}):
    """ Use `filename` to look for a file containing a json-encoded dictionary
        of credentials.  If the file is successfully found and contains valid
        json, return the sub-dictionary based on `subset`.  If the file is not
        found, is not accessible, has improper encoding, or if the subset is
        not present in the dictionary, return the given defaults.
    """
    try:
        with open(filename, 'r') as file:
            D = json.load(file)
            if D[subset]:
                return D[subset]
            else:
                return defaults
    except:
        return defaults

creds = getCreds("sql_creds.json", "mysql", defaults={'user': 'studen_j1',
                                                      'password': 'studen_j1'})
def db_setup(user, password, database):
    template = 'mysql+mysqlconnector://{}:{}@hadoop2.mathsci.denison.edu/{}'
    cstring = template.format(user, password, database)
    e = sa.create_engine(cstring)
    c = e.connect()
    return e, c, cstring
```

In [11]:
```
try:
    connection.close()
    del engine
except:
    pass
database = creds['user']
engine, connection, cstring = db_setup(creds['user'], creds['password'], database)
%load_ext sql

%sql $cstring

%sql USE $database
```

```
0 rows affected.
```

Out[11]: []

We used the function given to us earlier this year to establish the connection and link to our MySQL servers. We then used the format that we developped in our XMLtoSQL homework to make all of the drop/create functions to create the table, as well as the insert functions to add in all of our data to the tables.

In [12]:
```
artiststree = getBillboardTree('greatest-hot-100-artists')
artistRank = getTop100Rank(artiststree)
artists = getTopArtists(artiststree)
dfTopArtists = ArtistDF(artistRank, artists)

def dropcreate_TopArtists(conn):
    drop = 'DROP TABLE IF EXISTS TopArtists'
    create = 'CREATE TABLE TopArtists(Artist VARCHAR(50) NOT NULL, Rank INT NOT NULL, PRIMARY KEY (Rank))

    # Execute the query and fetch all rows of the result
    resultproxy = conn.execute(drop)
    results = conn.execute(create)
    return results


def insert_TopArtists(conn, df):
    for row in df.iterrows():
        insert = sa.sql.text('INSERT INTO TopArtists(Artist, Rank) VALUES (:x, :y)')
        boundTopArtists = insert.bindparams(x=str(row[1][0]), y=str(row[1][1]))
        resultproxy = connection.execute(boundTopArtists)

dropcreate_TopArtists(connection)
insert_TopArtists(connection, dfTopArtists)
```

In [25]:
```
%sql SELECT * FROM TopArtists LIMIT 5
```

```
5 rows affected.
```

Out[25]:

Artist	Rank
The Beatles	1
Madonna	2
Elton John	3
Elvis Presley	4
Mariah Carey	5

In [13]:
```
newartisttree = getBillboardTree('artist-100')
newartistrank = getTop100Rank(newartisttree)
newartists = getTopArtists(newartisttree)
dfArtist100 = ArtistDF(newartistrank, newartists)

def dropcreate_Artist100(conn):
    drop = 'DROP TABLE IF EXISTS Artist100'
    create = 'CREATE TABLE Artist100(Artist VARCHAR(50) NOT NULL, Rank INT NOT NULL, PRIMARY KEY (Rank))'

    # Execute the query and fetch all rows of the result
    resultproxy = conn.execute(drop)
    results = conn.execute(create)
    return results

def insert_Artist100(conn, df):
    for row in df.iterrows():
        insert = sa.sql.text('INSERT INTO Artist100(Artist, Rank) VALUES (:x, :y)')
        boundArtist100 = insert.bindparams(x=str(row[1][0]), y=str(row[1][1]))
        resultproxy = connection.execute(boundArtist100)

dropcreate_Artist100(connection)
insert_Artist100(connection, dfArtist100)
```

In [26]:
```
%sql SELECT * FROM Artist100 LIMIT 5
```

5 rows affected.

Out[26]:

Artist	Rank
J. Cole	1
Drake	2
Cardi B	3
Avicii	4
Imagine Dragons	5

In [14]:
```
hot100tree = getBillboardTree('hot-100')
hot100 = getTop100Songs(hot100tree)
hot100A = getSongArtists(hot100tree)
hot100rank = getTop100Rank(hot100tree)

def SongDF(rank, songs, artists):
    '''
    Given 3 lists of song rank, the song title, and song artist, it will return a dataframe in order of ra
    '''

    D = {'rank':rank,'Song':songs,'Artist':artists}
    dfHot100 = pd.DataFrame.from_dict(D, orient='columns')
    return dfHot100

dfHot100 = SongDF(hot100rank, hot100, hot100A)
```

In [15]:
```
def dropcreate_hot100(conn):
    drop = 'DROP TABLE IF EXISTS Hot100'
    create = 'CREATE TABLE Hot100(Artist VARCHAR(200) NOT NULL, Song VARCHAR(100) NOT NULL, Rank INT NOT N

    # Execute the query and fetch all rows of the result
    resultproxy = conn.execute(drop)
    results = conn.execute(create)
    return results


def insert_Hot100(conn, df):
    for row in df.iterrows():
        insert = sa.sql.text('INSERT INTO Hot100(Artist, Song, Rank) VALUES (:x, :y, :z)')
        boundHot100 = insert.bindparams(x=str(row[1][0]), y=str(row[1][1]), z=str(row[1][2]))
        resultproxy = connection.execute(boundHot100)

dropcreate_hot100(connection)
insert_Hot100(connection, dfHot100)
```

In [27]:
```
%sql SELECT * FROM Hot100 LIMIT 5
```

```
5 rows affected.
```

Out[27]:

Artist	Song	Rank
Drake	Nice For What	1
Drake	God's Plan	2
Ariana Grande	No Tears Left To Cry	3
Bebe Rexha & Florida Georgia Line	Meant To Be	4
Post Malone Featuring Ty Dolla $ign	Psycho	5

The genre data below was only available in pdf form. Other websites referenced it, but the only source with the data was a pdf, so we coud not webscrape from it. Instead, we decided to just copy the data by hand from the table into a list of lists, and transform it into a dataframe from there, because the data was necessary for our analysis but we couldn't get it in html form.

In [16]:
```
def getTopGenres():
    headers = ['Rank', 'Genre', 'Percent']
    LoL = [[1, ' r&b', 24.5], [2, ' rock', 20.8], [3, ' pop', 12.7], [4, ' country', 7.7 ], [5, ' latin',
    dfGenre = pd.DataFrame(LoL, columns = headers)
    return dfGenre
dfTopGenres = getTopGenres()
```

In [17]:
```
def dropcreate_TopGenres(conn):
    drop = 'DROP TABLE IF EXISTS TopGenres'
    create = 'CREATE TABLE TopGenres(Rank INT NOT NULL, Genre VARCHAR(25) NOT NULL, Percent DECIMAL(5,1) N

    # Execute the query and fetch all rows of the result
    resultproxy = conn.execute(drop)
    results = conn.execute(create)
    return results

def insert_TopGenres(conn, df):
    for row in df.iterrows():
        insert = sa.sql.text('INSERT INTO TopGenres(Rank, Genre, Percent) VALUES (:x, :y, :z)')
        boundTopGenres = insert.bindparams(x=str(row[1][0]), y=str(row[1][1]), z=str(row[1][2]))
        resultproxy = connection.execute(boundTopGenres)


dropcreate_TopGenres(connection)
insert_TopGenres(connection, dfTopGenres)
```

In [28]:
```
%sql SELECT * FROM TopGenres LIMIT 5
```

```
5 rows affected.
```

Out[28]:

Rank	Genre	Percent
1	r&b	24.5
2	rock	20.8
3	pop	12.7
4	country	7.7
5	latin	5.9

In [18]:
```
def insert_Albums(conn, albumID, albumName, albumReleaseDate):

    stmt = sa.sql.text("INSERT INTO Albums VALUES (:x,:y,:z)")

    resultproxy = conn.execute(stmt,x= albumID, y= albumName, z= albumReleaseDate)

def create_Albums_Table(conn):
    drop = 'DROP TABLE IF EXISTS Albums'
    create = """
    CREATE TABLE Albums(
       ALBUMID CHAR(225)          NOT NULL,
       ALBUMNAME CHAR(255)   ,
       ALBUMRELEASEDATE CHAR(255)   ,
       PRIMARY KEY (ALBUMID)
    );
    """
    resultproxy = conn.execute(drop)
    resultproxy = conn.execute(create)

    dicta = {}
    for i in range(len(TopTracks)):
        albumID = TopTracks.iloc[i,6] #albumID
        albumName = TopTracks.iloc[i,7] #album name
        albumReleaseDate = TopTracks.iloc[i,8] #album release date
        if albumID not in dicta:
            dicta[albumID] = []
            dicta[albumID].append(albumName)
            dicta[albumID].append(albumReleaseDate)

    for key in dicta:
        ID = key
        name = dicta[key][0]
        date = dicta[key][1]
        insert_Albums(conn, ID, name, date)

create_Albums_Table(connection)
```

In [24]:
```
%sql SELECT * FROM Albums LIMIT 5
```

```
5 rows affected.
```

Out[24]:

ALBUMID	ALBUMNAME	ALBUMRELEASEDATE
01sfgrNbnnPUEyz6GZYlt9	Dua Lipa (Deluxe)	2017-06-02
05CVGFPlWVC3WDwBfTzjca	Glow Like Dat	2017-08-15
06haetPrpbIFCY1FUWzVel	Salute (The Deluxe Edition)	2013-11-08
0829Pk9WEro3oPVnWT2B4B	Danger (with Migos & Marshmello) [From Bright: The Album]	2017-12-08
08ipn1MH7xqgoqhUbtvCTy	Caracal (Deluxe)	2015-09-25

In [19]:
```
def insert_Tracks(conn, trackID, trackName, trackPop, trackDur, artistID, albumID):

    stmt = sa.sql.text("INSERT INTO Tracks VALUES (:a,:b,:c,:d,:e,:f)")

    resultproxy = conn.execute(stmt,a= trackID, b= trackName, c= trackPop, d=trackDur, e = artistID, f =

def create_Tracks_Table(conn):
    drop = 'DROP TABLE IF EXISTS Tracks'
    query = """
    CREATE TABLE Tracks(
       TRACKID  CHAR(225)          NOT NULL,
       TRACKNAME CHAR(255)   ,
       TRACKPOP INT(64)   ,
       TRACKDUR INT(64) ,
       ARTISTID CHAR(225) NOT NULL,
       ALBUMID CHAR(225) NOT NULL,
       PRIMARY KEY (TRACKID)
    );
    """
    resultproxy = conn.execute(drop)
    resultproxy = conn.execute(query)

    dicta = {}
    for i in range(len(TopTracks)):
        trackID = TopTracks.iloc[i,0]
        artistID = TopTracks.iloc[i,2]
        trackName = TopTracks.iloc[i,3]
        trackDur = str(TopTracks.iloc[i,4])
        trackPop = str(TopTracks.iloc[i,5])
        albumID = TopTracks.iloc[i,6] #albumID
        if trackID not in dicta:
            dicta[trackID] = []
            dicta[trackID].append(trackName)
            dicta[trackID].append(trackPop)
            dicta[trackID].append(trackDur)
            dicta[trackID].append(artistID)
            dicta[trackID].append(albumID)

    for key in dicta:
        trackID = key
        trackName = dicta[key][0]
        trackPop = dicta[key][1]
        trackDur = dicta[key][2]
        artistID = dicta[key][3]
        albumID = dicta[key][4]
        #print(trackID,trackName,trackPop,trackDur,artistID,albumID)
        insert_Tracks(conn, trackID, trackName, trackPop, trackDur, artistID, albumID)

create_Tracks_Table(connection)
```

In [29]:
```
%sql SELECT * FROM Tracks LIMIT 5
```

5 rows affected.

Out[29]:

TRACKID	TRACKNAME	TRACKPOP	TRACKDUR	ARTISTID	ALBUMID
00CqEmnPLFKDhAb3cuu6Cs	Halo	53	177960	26T3LtbuGT1Fu9m0eRq5X3	4EK8gtQfdVsmDTji7gBFlz
02R2z7JWV0G8VuU1xs58OB	Come	62	162360	2HHmvvSQ44ePDH7IKVzgK0	2rb6C1wUwk7hFOVmfgt19k
03PM8jkLPwI6cDcZBvwCSL	He'll Never Love You (HNLY)	65	231293	3LjhVl7GzYsza1biQjTpaN	2oRkkW6ZudviRBd6mx4CfL
03W4Ya6D0isiTvODjf0Afb	Brother Be Wise	3	210949	00jxQK5hfu4xTLhatLBggp	3uThhPJ9YHIQ21y3VjOE2C
03xWMkKEbeO4SnyIA53ipj	When Will My Life Begin - From "Tangled" / Soundtrack Version	65	152333	2LJxr7Pt3JnP60eLxwbDOu	1l0aFrH24oPrQSqGtfeFyE

In [20]:
```
def insert_Artists(conn, artistID, artistName, artistGenre, artistPop, follower):

    stmt = sa.sql.text("INSERT INTO Artists VALUES (:a,:b,:c,:d,:e)")

    resultproxy = conn.execute(stmt,a= artistID, b= artistName, c= artistGenre, d=artistPop, e = follower

def create_Artists_Table(conn):
    drop = 'DROP TABLE IF EXISTS Artists'
    query = """
    CREATE TABLE Artists(
       ARTISTID CHAR(225) NOT NULL,
       ARTISTNAME CHAR(225),
       ARTISTGENRE CHAR(225),
       ARTISTPOPULARITY INT(64),
       FOLLOWER INT(64),
       PRIMARY KEY (ARTISTID)
    );
    """
    resultproxy = conn.execute(drop)
    resultproxy = conn.execute(query)

    dicta = {}
    for i in range(len(TopArtist)):
        artistID = TopArtist.iloc[i,2]
        artistName = TopArtist.iloc[i,3]
        artistGenre = TopArtist.iloc[i,1]
        artistPop = str(TopTracks.iloc[i,4])
        follower = str(TopTracks.iloc[i,0])
        if artistID not in dicta:
            dicta[artistID] = []
            dicta[artistID].append(artistName)
            dicta[artistID].append(artistGenre)
            dicta[artistID].append(artistPop)
            dicta[artistID].append(follower)

    for key in dicta:
        artistID = key
        artistName = dicta[key][0]
        artistGenre = dicta[key][1]
        artistPop = dicta[key][2]
        follower = dicta[key][3]
        #print(artistID,artistName,artistGenre,artistPop,follower)
        insert_Artists(conn, artistID, artistName, artistGenre, artistPop, follower)

create_Artists_Table(connection)
```

In [30]:
```
%sql SELECT * FROM Artists LIMIT 5
```

```
5 rows affected.
```

Out[30]:

ARTISTID	ARTISTNAME	ARTISTGENRE	ARTISTPOPULARITY	FOLLOWER
00FQb4jTyendYWaN8pK0wa	Lana Del Rey	dance pop, pop	270013	7
00jxQK5hfu4xTLhatLBggp	Harry Jay Smith & the Bling		197213	38
00Z3UDoAQwzvGu13HoAM7J	Skizzy Mars	indie pop rap, pop rap, rap	191251	3
02kJSzxNuaWGqwubyUba0Z	G-Eazy	indie pop rap	176320	7
04gDigrS5kc9YWfZHwBETP	Maroon 5	pop	216546	27

In [21]:
```
def insert_ArtistGenre(conn, num, artistID, artistGenre):

    stmt = sa.sql.text("INSERT INTO ArtistGenre VALUES (:a,:b,:c)")

    resultproxy = conn.execute(stmt,a= num,b= artistID, c= artistGenre)

def create_ArtistGenre_Table(conn):
    drop = 'DROP TABLE IF EXISTS ArtistGenre'
    query = """
    CREATE TABLE ArtistGenre(
       NUM INT(64) NOT NULL,
       ARTISTID CHAR(225) NOT NULL,
       ARTISTGENRE CHAR(225),
       PRIMARY KEY (NUM)
    );
    """
    resultproxy = conn.execute(drop)
    resultproxy = conn.execute(query)


    dicta = {}
    for i in range(len(TopArtist)):
        num = i + 1
        artistID = ArtistGenre.iloc[i,0]
        artistGenre = ArtistGenre.iloc[i,2]
        insert_ArtistGenre(conn,num, artistID, artistGenre)

create_ArtistGenre_Table(connection)
```

In [31]:
```
%sql SELECT * FROM ArtistGenre LIMIT 5
```

5 rows affected.

Out[31]:

NUM	ARTISTID	ARTISTGENRE
1	26VFTg2z8YR0cCuwLzESi2	dance pop
2	26VFTg2z8YR0cCuwLzESi2	etherpop
3	26VFTg2z8YR0cCuwLzESi2	indie poptimism
4	26VFTg2z8YR0cCuwLzESi2	pop
5	26VFTg2z8YR0cCuwLzESi2	post-teen pop

In [22]:
```
%sql SHOW TABLES
```

8 rows affected.

Out[22]:

Tables_in_nguyen_h2
Albums
Artist100
ArtistGenre
Artists
Hot100
TopArtists
TopGenres
Tracks