data_collection

January 6, 2022

1 Data Collection from Spotify Web API

1.0.1 Method

This code is used for all data collection. Data is collected using the following method: 1. Authentication and http client setup 2. Use the 'categories' endpoint to get the id and name of all categories.
The category is used to identify the genre of a track. 3. Use the {category_id}/playlists endpoint to get the name and id of each playlist in the category 4. Use the playlists/{playlist_id}/tracks endpoint to get - Track id and name - album information - names of artists 5. Use the audio-features endpoint to get all available audio features for each track
At this point we have all data stored in a nested json file, which needs to be flattened, in order to create a dataframe that can be used for further processing and data exploration 6. Flatten json and create dataframe ### Running this code - Place a .env file in the project root directory, which contains the following variables

```
CLIENT_ID=
CLIENT_SECRET=
```

To obtain these credentials - create a Spotify account - go to the developer dashboard - create an app - copy the apps credentials into the .env file

1.0.2 Filtering

You might not want to get data for all categories, as this needs several thousand calls to the api.
We implemented a category filter system which you can use to filter by category id.

1.0.3 Continuing with data from file

Each step of the method can be run on its own and the data used as input can be given in form of a json file.
If no file is used, results from the previous step is directly passed in.
Alternatively, each step can output its result to a file. The path is specified in the function arguments.
`(write_to_file=True, path_to_file=%path%)`

How to use input file: - uncomment code to load file into data variable - run all the setup code up to and including the point where data is loaded from file - run blocks from the step you want to continue at

1.1 Imports, authentification and http setup

```
[ ]: # imports
import http
from dotenv import load_dotenv
import os
import json
import requests
import math
from copy import copy
import csv

#Get environment variables from ".env" file and read credentials
load_dotenv('.env')
client_id = os.environ.get('CLIENT_ID')
client_secret = os.environ.get('CLIENT_SECRET')

# Authenticate and get an API Token from Spotify using a Client ID and secret
def getAuthTokenFromCredentials(id, secret):

    url = "https://accounts.spotify.com/api/token"

    payload =␣
 ↪f'grant_type=client_credentials&client_id={id}&client_secret={secret}'
    headers = {
    'Content-Type': 'application/x-www-form-urlencoded',
    }

    response = requests.request("POST", url, headers=headers, data=payload)

    return response.json()["access_token"]

auth_token = getAuthTokenFromCredentials(client_id, client_secret)
```

1.1.1 HTTP Setup

```
[ ]: from requests.adapters import HTTPAdapter
from urllib3.util import Retry

DEFAULT_TIMEOUT = 10 # seconds

# This is used to configure timeouts and retries if the API takes a long time␣
 ↪to respond to the call
# It's crucial that theres some room for slow responses, as one failed request␣
 ↪will exit out of the whole function
class TimeoutHTTPAdapter(HTTPAdapter):
    def __init__(self, *args, **kwargs):
        self.timeout = DEFAULT_TIMEOUT
        if "timeout" in kwargs:
            self.timeout = kwargs["timeout"]
            del kwargs["timeout"]
        super().__init__(*args, **kwargs)

    def send(self, request, **kwargs):
        timeout = kwargs.get("timeout")
        if timeout is None:
            kwargs["timeout"] = self.timeout
        return super().send(request, **kwargs)

def setupRequestsSession():
    http = requests.Session()
    assert_status_hook = lambda response, *args, **kwargs: response.
 ↪raise_for_status()
    http.hooks["response"] = [assert_status_hook]

    retries = Retry(total=5, backoff_factor=1, status_forcelist=[429, 500, 502,␣
 ↪503, 504])
    adapter = TimeoutHTTPAdapter(max_retries=retries)
    http.mount("https://", adapter)
    http.mount("http://", adapter)

    return http
```

[ ]:
```
# get http session
http = setupRequestsSession()
```

1.2 Filter

[ ]:
```
category_filter = None

# comment this out if you don't want to set a filter
category_filter = ["hiphop", "pop", "country", "rock", "latin", "rnb", "mood",␣
 ↪"indie_alt",
                    "regional_mexican", "edm_dance", "inspirational", "chill",␣
 ↪"party", "roots",
                    "kpop", "instrumental", "ambient", "alternative",␣
 ↪"classical", "jazz", "soul",
                    "punk", "blues", "arab", "afro", "metal", "caribbean",␣
 ↪"funk"]

# tells data collection functions to not use filter if it's not set
if category_filter is not None:
    use_filter = True
```

```
else:
    use_filter = False
```

1.3 Data from file

```
[ ]: data = {}

     #To load data_object from file instead of rerunning the scripts, uncomment this:

     #file = open(os.path.join("data_collection", "json", "tracks_full.json"))
     #data = json.load(file)
```

1.4 Get Categories

```
[ ]: # function definition

     def getAllCategories(requests_session, auth_token, data_object,␣
      ↪use_category_filter=False, category_filter=None, write_to_file=False,␣
      ↪path_to_file=''):

         # Establishing the requests session
         http = requests_session

         # Establishing given data object
         data = data_object

         # First API call used to get the total amount of categories
         headers = { 'Authorization': f'Bearer {auth_token}' }
         url = "https://api.spotify.com/v1/browse/categories?
      ↪country=US&locale=en_US&limit=1"

         try:
             response = http.request("GET", url, headers=headers, data={})
             if response.status_code != requests.codes.ok:
                 raise Exception
         except Exception as e:
             raise SystemExit(e)

         response.raise_for_status()
         categoryAmount = response.json()["categories"]["total"]

         # API only returns 50 items at a time. Offset can be used to gradually get␣
      ↪all items
         # Calculate number of pages with 50 items
         pages = int(math.ceil(categoryAmount/50))
         data = {"categories": []}
```

```
    # Second call gets all categories
    for x in range(pages):
        url = f"https://api.spotify.com/v1/browse/categories?
↪country=US&locale=en_US&limit=50&offset={x * 50}"

        try:
            response = http.request("GET", url, headers=headers, data={})
            if response.status_code != requests.codes.ok:
                raise Exception
        except Exception as e:
            raise SystemExit(e)

        # categories are stored in the data dictionary
        for el in response.json()["categories"]["items"]:
            if (use_category_filter == True and el["id"] in category_filter) or␣
↪use_category_filter == False:
                data["categories"].append({
                    "id": el["id"],
                    "name": el["name"]
                })

    if write_to_file == True:
        with open(path_to_file, 'w') as outfile:
            json.dump(data, outfile, indent=2)

    return data
```

```
[ ]: # execution
data = getAllCategories(
    requests_session=http,
    auth_token=auth_token,
    data_object=data,
    use_category_filter=use_filter,
    category_filter=category_filter,
    write_to_file=True,
    path_to_file=os.path.join("json", "01_categories.json"))

print("got categories")
```

```
got categories
```

1.5 Get Playlists

[ ]:
```python
# function definition

def getPlaylistsForCategories(requests_session, auth_token, data_object,␣
 ↪write_to_file=False, path_to_file=''):

    # Establishing the requests session
    http = requests_session

    # Establishing given object
    data = data_object

    for category in data["categories"]:

        category_id = category["id"]
        url = f"https://api.spotify.com/v1/browse/categories/{category_id}/
 ↪playlists?country=US&limit=1&offset=0"
        headers = { 'Authorization': f'Bearer {auth_token}' }

        try:
            response = http.request("GET", url, headers=headers, data={})
            if response.status_code != requests.codes.ok:
                raise Exception
        except Exception as e:
            raise SystemExit(e)

        categoryAmount = response.json()["playlists"]["total"]

        #Calculate number of pages with 50 items
        pages = int(math.ceil(categoryAmount/50))

        #Initialize playlist attribute
        category["playlists"] = []

        # Get 50 playlists at a time and increase offset by 50
        for page in range(pages):
            url = f"https://api.spotify.com/v1/browse/categories/{category_id}/
 ↪playlists?country=US&limit=50&offset={page * 50}"
            headers = { 'Authorization': f'Bearer {auth_token}' }
            try:
                response = http.request("GET", url, headers=headers, data={})
                if response.status_code != requests.codes.ok:
                    raise Exception
            except Exception as e:
                raise SystemExit(e)
            # Store playlists for each category
            i = 0
            for playlist in response.json()["playlists"]["items"]:
                if playlist["type"] == "playlist":
                    category["playlists"].append({
                        "id": playlist["id"],
                        "name": playlist["name"]
                    })
                i += 1

        if write_to_file == True:
            with open(path_to_file, 'w') as outfile:
                json.dump(data, outfile, indent=2)

    return data
```

[ ]:
```
# execution

data = getPlaylistsForCategories(
    requests_session=http,
    auth_token=auth_token,
    data_object=data,
    write_to_file=True,
    path_to_file=os.path.join("json", "02_playlists.json"))

print("got playlists")
```

```
got playlists
```

1.6 Get Tracks

[ ]:
```
# function definition

def getTracksOfPlaylists(requests_session, auth_token, data_object,␣
 ↪write_to_file=False, path_to_file=''):

    # Establishing the requests session
    http = requests_session

    # Establishing given object
    data = data_object

    for category in data["categories"]:

        # Get all tracks for all playlists in all categories
        for playlist in category["playlists"]:
```

```
        # First we call the API once to learn how many tracks are in the␣
↪playlist. This is indicated in the field "total"
        playlist_id = playlist["id"]
        url = f"https://api.spotify.com/v1/playlists/{playlist_id}/tracks?
↪market=US&limit=2&offset=0&fields=items(track(name,id,album(name,id),artists)),total&additi
        headers = { 'Authorization': f'Bearer {auth_token}' }

        try:
            response = http.request("GET", url, headers=headers, data={})
            if response.status_code != requests.codes.ok:
                raise Exception
        except Exception as e:
            raise SystemExit(e)

        trackAmount = response.json()["total"]

        #Calculate number of pages with 50 items based on the number of␣
↪total tracks
        pages = int(math.ceil(trackAmount/50))

        #Initialize playlist attribute
        playlist["tracks"] = []

        # Get tracks on each page
        for page in range(pages):
            url = f"https://api.spotify.com/v1/playlists/{playlist_id}/
↪tracks?market=US&limit=50&offset={page * 50}&fields=items(track(name, id,␣
↪album(name, id), artists)), total&additional_types=track"
            headers = { 'Authorization': f'Bearer {auth_token}' }

            try:
                response = http.request("GET", url, headers=headers,␣
↪data={})
                if response.status_code != requests.codes.ok:
                    raise Exception
            except Exception as e:
                raise SystemExit(e)

            i = 0
            for item in response.json()['items']:
                track = item["track"]

                # Some track elements will have value null, this throws␣
↪exception
                if track is None:
                    continue
                    artists = []
                    # Contains all artists and their ids for each track
                    for artist in track["artists"]:
                        artists.append({
                            "id" : artist["id"],
                            "name" : artist["name"]
                        })

                    # This is all of the metadata saved for each track
                    playlist["tracks"].append({
                        "id": track["id"],
                        "name": track["name"],
                        "album" : {
                            "id" : track["album"]["id"],
                            "name" : track["album"]["name"]
                            },
                        "artists" : artists
                     })
                    i += 1

        if write_to_file == True:
            with open(path_to_file, 'w') as outfile:
                json.dump(data, outfile, indent=2)

    return data
```

[ ]:
```
# execution

data = getTracksOfPlaylists(
    requests_session=http,
    auth_token=auth_token,
    data_object=data,
    write_to_file=True,
    path_to_file=os.path.join("json", "03_tracks.json"))

print("got tracks")
```

```
got tracks
```

1.7 Get Features

[ ]:
```
# function definition

def getFeaturesOfTracks(requests_session, auth_token, data_object,␣
 ↪write_to_file=False, path_to_file=''):
    #Establishing the requests session
    http = requests_session

    #Establishing given object
    data = data_object

    for category in data["categories"]:

        for playlist in category["playlists"]:

            track_ids = [[]]

            # API endpoint is called using all track ids seperated by comma
            # This creates arrays of arrays containing 99 track ids
            # This is done because a maximum of 99 tracks can be requested at a␣
↪time
            for track in playlist["tracks"]:

                track_ids[len(track_ids)-1].append(track["id"])

                if len(track_ids[len(track_ids)-1]) > 99:
                    track_ids.append([])

            all_track_features = []

            # Get all features for all tracks for all playlists in all␣
↪categories
            for page in track_ids :
                # Each subarray is joined by comma and used for a get request
                comma_seperated_ids = ",".join(page)

                url = f"https://api.spotify.com/v1/audio-features?
↪ids={comma_seperated_ids}"
                headers = { 'Authorization': f'Bearer {auth_token}' }

                try:
                    response = http.request("GET", url, headers=headers,␣
↪data={})
                    if response.status_code != requests.codes.ok:
                        raise Exception
                except Exception as e:
                    raise SystemExit(e)

                # Combine lists
                all_track_features = all_track_features + response.
↪json()["audio_features"]
```

```
            i = 0
            for track in playlist["tracks"]:

                # Removing unneeded features to save a bit of space
                for entry in all_track_features:
                    if entry is not None and track["id"] == entry["id"] and␣
↪entry["type"] == "audio_features":
                        track["features"] = copy(entry)
                        del track["features"]["id"]
                        del track["features"]["type"]
                        del track["features"]["uri"]
                        del track["features"]["track_href"]
                        del track["features"]["analysis_url"]
                        break
                i += 1

        if write_to_file == True:
            with open(path_to_file, 'w') as outfile:
                json.dump(data, outfile, indent=2)

    return data
```

[ ]:
```
# execution

data = getFeaturesOfTracks(
    requests_session=http,
    auth_token=auth_token,
    data_object=data,
    write_to_file=True,
    path_to_file=os.path.join("json", "04_features.json"))

print("got features")
```

```
got features
```

1.8 Flatten JSON

This does not mutate any of the data collected except artist information.
To maintain one table row per track and remove redundancy, we collapse multiple artists into a comma seperated list.
This does not matter, as we won't use "artist" as a feature

[ ]:
```
# function definition

def flatten_json(data_object, write_to_file=True, path_to_file=''):

    data = data_object

    # variables
    temp = {}
    result = []

    # loop over categories
    for category in data["categories"]:
        path = "categories"

        for item in category:
                if item != "playlists":
                    key     = f"{path}.{item}"
                    value   = f"{category[item]}"
                    temp[key] = value
                else:
                    # loop over playlist
                    for playlist in category["playlists"]:
                        path = "categories.playlists"

                        for item in playlist:
                            if item != "tracks":
                                key     = f"{path}.{item}"
                                value   = f"{playlist[item]}"
                                temp[key] = value
                            else:
                                # loop over tracks
                                for track in playlist["tracks"]:
                                    path = "categories.playlists.tracks"
                                    for item in track:
                                        if item != "album" and item !=␣
↪"artists" and item != "features":
                                            key     = f"{path}.{item}"
                                            value   = f"{track[item]}"
                                            temp[key] = value

                                        # album data
                                        elif item == "album":
                                            for album in track["album"]:
                                                key     = f"{path}.album.
↪{album}"
                                                value   =␣
↪f"{track['album'][album]}"
                                                temp[key] = value

                                        # artist data (just name)
                                        # at this point, multiple datafields␣
↪are collapsed into a comma seperated list to maintain one table row per␣
↪track and remove redundancy
```

```
                                            # this does not matter for our␣
↪purposes, as we are not using artist names as features
                                            elif item == "artists":
                                                value = ""

                                                for artist in track["artists"]:
                                                    if not value:
                                                        value = artist["name"]
                                                    else:
                                                        value =␣
↪f"{value},{artist['name']}"

                                                key = f"{path}.artists"
                                                temp[key] = value

                                            # track features
                                            elif item == "features":

                                                for feature in track["features"]:
                                                    key     = f"{path}.features.
↪{feature}"
                                                    value   =␣
↪f"{track['features'][feature]}"
                                                    temp[key] = value

                                        # At this point, temp contains a flat␣
↪dictionary with all nested fields (which represents one row in a table)
                                        # This is appended to the result, which is␣
↪an array containing all table rows
                                        result.append(copy(temp))

    if write_to_file == True:
        with open(path_to_file, 'w') as outfile:
            json.dump(result, outfile, indent=2)

    return result
```

[ ]:
```
# execution

print("flattening json...")
data = flatten_json(data, True,  os.path.join("json", "05_flat_data.json"))
print("done flattening json")
```

```
flattening json…
done flattening json
```

1.9 Create CSV from flat JSON data

[ ]:
```
# function definition

def json_to_csv(data_object, path_to_file=''):

    # open file to write to
    f = open(path_to_file, 'w')

    # create the csv writer object
    csv_writer = csv.writer(f)

    count = 0
    for line in data_object:
        if count == 0:

            # Writing headers of CSV file
            header = line.keys()
            csv_writer.writerow(header)
            count += 1

        # Writing data of CSV file
        csv_writer.writerow(line.values())

    f.close()
```

[ ]:
```
# execution

json_to_csv(data, os.path.join("final_result.csv"))
print("created csv")
```

```
created csv
```