📝

Spotify Data Extraction and CSV Conversion Using Python

This tutorial caters to both beginners and experienced Python developers looking to integrate Spotify's Web API into their projects. Throughout this guide, you will be meticulously guided through each step of setting up your Python environment, installing necessary libraries, authenticating with the Spotify API, and efficiently extracting data from Spotify. Whether you are aiming to enhance your data handling skills or seeking to understand the intricacies of working with APIs, this document serves as a comprehensive resource to guide you through the process seamlessly.

Install the Required Tools

This section provides step-by-step instructions for installing the necessary tools. If you have already installed these tools, you may skip this section.

Python Installation

Python is essential for running the code. Follow these steps to install Python:

1. Visit the Python Official Website.
2. Navigate to the Downloads page and select the version suitable for your operating system.
3. After downloading, open the installer (.exe file for Windows).
4. Follow the installation prompts, accepting the software license agreement.
5. Choose an installation location, then complete the installation process.
6. Verify the installation by checking the Python version in the command line (type `python --version`).

Jupyter Notebook Installation

Jupyter Notebook is an interactive coding environment. Follow these steps to install Jupyter Notebook:

1. Open Command Prompt (Windows) or Terminal (Mac).
    - Windows: Search for "Command Prompt" in the start menu.
    - Mac: Press F4 and search for "Terminal".
2. Ensure `pip`, the package installer for Python, is up-to-date by running:
    ```
    pip install --upgrade pip
    ```
3. Install Jupyter Notebook using:
    ```
    pip install jupyter
    ```
4. To launch Jupyter Notebook, type:
    ```
    jupyter notebook
    ```

Setting Up Spotify Developer Account

To access the Spotify API, a Spotify Developer account is required. Follow these steps to obtain a Spotify Developer account:

1. Go to the Spotify Developer Dashboard.
2. Log in with your Spotify credentials, or create a new Spotify account if necessary.

   > 💡 **Note:** A premium Spotify account is not required for developer access.

3. Accept the Terms of Service.
4. Click on "Create an App" and provide a name and description for your application.
5. Your dashboard will display your **Client ID**.
6. Reveal your **Client Secret** by clicking on "Show Client Secret".
7. Keep these credentials secure, as they are necessary for API requests.

Installing NumPy

NumPy is used for scientific computing and working with arrays. Follow these steps to install NumPy:

1. Open Command Prompt (Windows) or Terminal (Mac).
2. Install NumPy with the command:

   ```
   pip install numpy
   ```

Installing Pandas

Pandas is a library for data manipulation and analysis. Follow these steps to install Pandas:

1. Open Command Prompt (Windows) or Terminal (Mac).
2. Run the installation command:

   ```
   pip install pandas
   ```

Installing Spotipy

Spotipy is a Python library for the Spotify Web API. Follow these steps to install Spotipy:

1. Open Command Prompt (Windows) or Terminal (Mac).
2. Install Spotipy using:

   ```
   pip install spotipy
   ```

Import Packages and Libraries

The foundation of working with Spotify's data in Python involves setting up your environment with the necessary libraries. Follow these steps to get started:

1. **Launch Jupyter Notebook:**
   - Open Jupyter Notebook by typing `jupyter notebook` in your Command Prompt (Windows) or Terminal (Mac). This will open Jupyter in your default web browser.
2. **Importing Libraries:**
   - In a new Jupyter notebook, you'll start by importing all the required packages and libraries. This step is crucial for accessing and processing Spotify data. Here's an example of what your import statements might look like:

     ```
     import spotipy
     import numpy as np
     ```

```
import pandas as pd
```

- **Note:** Refer to the previous step to ensure all necessary tools and libraries are installed.

3. **Execute the Code Cell:**
    - To import these libraries into your Jupyter notebook, run the cell by pressing `Shift + Enter/Return` or by clicking the 'Run' icon in the toolbar.
    - This action executes the code within the cell.
4. **Verify Successful Import:**
    - Upon running the cell, you should not see any output. This is normal for import statements. However, it's crucial to ensure there are no error messages. If the cell runs without errors, it confirms the successful import of the libraries.
    - If you encounter an error, it usually indicates a missing package or an issue with the installation. Refer back to the installation section to resolve any issues.
5. **Preparing for Coding:**
    - With the libraries successfully imported, you're now set to proceed with writing the code to extract and manipulate data from Spotify.

Create a List of Album Names and Album URIs

This section demonstrates how to create a list of album names and their corresponding Spotify URIs (Uniform Resource Indicators). This guide will use two sample albums, "Melodrama" and "Pure Heroine" as examples.

1. Setting up Spotify API Authentication

First, you need to authenticate your application with Spotify:

- Declare variables for your Spotify Client ID and Client Secret. You can find these on your Spotify Developer Account dashboard.
- Replace 'YOUR-CLIENT-ID' and 'YOUR-CLIENT-SECRET' with your actual Client ID and Client Secret.

```
client_id = 'YOUR-CLIENT-ID'
client_secret = 'YOUR-CLIENT-SECRET'
```

- Use these credentials to authenticate your requests to the Spotify API. Run the cell to ensure there are no errors.

2. Finding and Storing Album URIs

A Spotify URI uniquely identifies a Spotify resource (album, artist, playlist, etc.).

- To find an album's URI:
  a. Open Spotify and navigate to the album of choice.
  b. Click the three dots beneath the album name.
  c. Choose "Share" and hover over "Copy Album Link".
  d. Press the Option (Alt) key on your keyboard.
  e. Select "Copy Album URI".
- Store the album URIs in a list. For example:

```
album_uris = ['spotify:album:6rnzvZhe3PA57xKcKLRtJ6', 'spotify:album:4oCGmYsAQOWt2ACWTpN
U']
```

3. Retrieving Album Data with Spotipy

Use Spotipy's `albums` function to fetch album data:

- Initialize the Spotify client with your credentials:

```
from spotipy.oauth2 import SpotifyClientCredentials
import spotipy

client_credentials = SpotifyClientCredentials(client_id, client_secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials)
```

- Retrieve album data by passing the list of album URIs to the `albums` function:

```
get_album = sp.albums(album_uris)
```

- This function will return a wealth of information about the albums, which you can then process and use in your data frame.

4. Preparing Lists for Album Data

Create two empty lists to store album names and their URIs:

```
lorde_album_names = []  # List for album names
lorde_album_uris = []   # List for album URIs
```

5. Extracting Album Names and URIs

Iterate over each album in the album URIs list, extracting and appending the album name and URI to the respective lists:

```
for i, album in enumerate(get_album['albums']):
    # Extracting the album name
    album_name = album['name']
    lorde_album_names.append(album_name)

    # Extracting the album URI
    album_uri = album['uri']
    lorde_album_uris.append(album_uri)
```

- The Spotify data is provided in JSON format. The album name is located under the key `"name"`.
- To access the album name, we use `get_album['albums'][i]['name']`.
- The same approach is applied to retrieve the album's URI.

6. Verifying the Extraction Process

To ensure that the extraction process works correctly, you can run the following code:

```
print("Album Names:", lorde_album_names)
print("Album URIs:", lorde_album_uris)
```

- This code will display the lists of album names and URIs, confirming that the data has been successfully extracted and stored.

Note on Viewing Album Data

To explore the complete dataset returned by Spotify, you can run the following code:

```
import json
print(json.dumps(get_album, indent=4))  # This will print the data in a formatted JSON struct
ure
```

- This will display the entire JSON data structure returned by Spotify, allowing you to see all the values and details of each album.

Extract All the Tracks From the Album List

The objective of this step is to construct a dictionary that encapsulates all albums, including each track and its associated data. We will develop a function to extract and store tracks from a given list of albums.

1. Initializing the Storage Structure

Start by creating an empty dictionary to hold the album data and its tracks:

```
lorde_albums = {}
```

2. Defining the Extraction Function

Construct a function to extract track data from an album URI:

- Define a function named `lorde_songs` which takes `album` as a parameter.
- Within the function, create a dictionary structure for each album to store track details like album name, track number, ID, name, and URI.

  ```
  def lorde_songs(album):
      lorde_albums[album] = {
          'album': [],
          'track_number': [],
          'id': [],
          'name': [],
          'uri': []
      }
  ```

3. Extracting Track Data

Use Spotipy's `album_tracks` function to fetch the album's tracks:

- Assign the result of `sp.album_tracks(album_id)` to a variable `tracks`. This function retrieves the album's track catalog.

  ```
  tracks = sp.album_tracks(album)
  ```

- Iterate over the `tracks['items']` to extract and append the necessary data to the corresponding lists in the dictionary.

  ```
  for track in tracks['items']:
      lorde_albums[album]['album'].append(lorde_album_names[album_count])
      lorde_albums[album]['track_number'].append(track['track_number'])
      lorde_albums[album]['id'].append(track['id'])
      lorde_albums[album]['name'].append(track['name'])
      lorde_albums[album]['uri'].append(track['uri'])
  ```

- Note that `tracks['items']` refers to the array containing track information in the Spotify album JSON data.

4. Implementing the Album Count

Introduce an `album_count` variable to track the current album index in the list:

```
album_count = 0
```

- This variable is incremented in the loop to align the album name with the corresponding tracks.

5. Applying the Function to Each Album URI

Now, iterate over the album URIs, applying the `lorde_songs` function to each:

```
for album_uri in lorde_album_uris:
    lorde_songs(album_uri)
```

```
    album_count += 1
```

- Ensure that the variable used in the `lorde_songs` call matches the parameter in the function definition.

6. Testing the Functionality

Execute the cell to confirm that the function operates correctly and the data is being stored as intended.

Extract Audio Features

In this step, we aim to enrich our dictionary with the audio features of each track. Spotify's API provides access to a variety of audio features. We will extract the following 10 features for each track:

- Acousticness
- Danceability
- Energy
- Instrumentalness
- Liveness
- Loudness
- Speechiness
- Tempo
- Valence
- Popularity

Note: For more detailed information on these audio features, visit Spotify's Web API Documentation.

1. Defining the Feature Extraction Function

Start by defining a function, `song_features`, which takes `album` as its parameter. Within this function, initialize empty lists for each audio feature:

```
def song_features(album):
    lorde_albums[album]['acousticness'] = []
    lorde_albums[album]['danceability'] = []
    lorde_albums[album]['energy'] = []
    lorde_albums[album]['instrumentalness'] = []
    lorde_albums[album]['liveness'] = []
    lorde_albums[album]['loudness'] = []
    lorde_albums[album]['speechiness'] = []
    lorde_albums[album]['tempo'] = []
    lorde_albums[album]['valence'] = []
    lorde_albums[album]['popularity'] = []
```

2. Looping Through Tracks and Extracting Features

Inside the function, iterate over the track URIs in the dictionary and extract each track's audio features and popularity:

- Utilize Spotipy's `audio_features` and `track` functions to gather data.

```
for track_uri in lorde_albums[album]['uri']:
    features = sp.audio_features(track_uri)[0]
    popularity = sp.track(track_uri)['popularity']

    # Appending each feature to its corresponding list
    lorde_albums[album]['acousticness'].append(features['acousticness'])
    lorde_albums[album]['danceability'].append(features['danceability'])
    lorde_albums[album]['energy'].append(features['energy'])
    lorde_albums[album]['instrumentalness'].append(features['instrumentalness'])
    lorde_albums[album]['liveness'].append(features['liveness'])
    lorde_albums[album]['loudness'].append(features['loudness'])
    lorde_albums[album]['speechiness'].append(features['speechiness'])
    lorde_albums[album]['tempo'].append(features['tempo'])
    lorde_albums[album]['valence'].append(features['valence'])
    lorde_albums[album]['popularity'].append(popularity)
```

3. Applying the Function to Each Album

With the function defined, apply it to each album in your dictionary:

```
for album in lorde_albums:
    song_features(album)
```

- This loop ensures that the `song_features` function processes each album and extracts the desired audio features for every track.

4. Testing the Extraction Process

Run the cell to verify the successful execution of the feature extraction process. This will populate your dictionary with the audio features of each track from the albums.

Transform the Dictionary into a DataFrame

The final step involves converting the dictionary containing all our track data into a well-structured DataFrame. This transformation is necessary to facilitate better visualization and analysis of the data.

1. Initializing the DataFrame Structure

Begin by creating an empty dictionary to serve as the foundation for our DataFrame. Then, initialize empty lists for each column we plan to include:

```
lorde_df = {
    'name': [],
    'album': [],
    'track_number': [],
    'id': [],
    'uri': [],
    'acousticness': [],
    'danceability': [],
    'energy': [],
    'instrumentalness': [],
    'liveness': [],
    'loudness': [],
    'speechiness': [],
    'tempo': [],
    'valence': [],
    'popularity': []
}
```

2. Populating the DataFrame

Use nested loops to iterate through each album and its features in the `lorde_albums` dictionary:

```
for album in lorde_albums:
    for i in range(len(lorde_albums[album]['name'])):
        lorde_df['name'].append(lorde_albums[album]['name'][i])
        lorde_df['album'].append(album)
        lorde_df['track_number'].append(lorde_albums[album]['track_number'][i])
        lorde_df['id'].append(lorde_albums[album]['id'][i])
        lorde_df['uri'].append(lorde_albums[album]['uri'][i])
        lorde_df['acousticness'].append(lorde_albums[album]['acousticness'][i])
        lorde_df['danceability'].append(lorde_albums[album]['danceability'][i])
        lorde_df['energy'].append(lorde_albums[album]['energy'][i])
        lorde_df['instrumentalness'].append(lorde_albums[album]['instrumentalness'][i])
        lorde_df['liveness'].append(lorde_albums[album]['liveness'][i])
        lorde_df['loudness'].append(lorde_albums[album]['loudness'][i])
        lorde_df['speechiness'].append(lorde_albums[album]['speechiness'][i])
        lorde_df['tempo'].append(lorde_albums[album]['tempo'][i])
        lorde_df['valence'].append(lorde_albums[album]['valence'][i])
        lorde_df['popularity'].append(lorde_albums[album]['popularity'][i])
```

3. Creating the DataFrame

Convert the dictionary into a pandas DataFrame and adjust the index to start from 1:

```
import pandas as pd

df = pd.DataFrame(lorde_df)
df.index += 1
```

4. Displaying the DataFrame

To view the DataFrame, simply execute:

```
print(df)
```

5. Exporting to CSV

To export the DataFrame to a CSV file, use pandas' `to_csv` function:

```
df.to_csv('lorde_spotify_data.csv')
```

Congratulations! You have successfully created a DataFrame containing your chosen artist's Spotify data. This DataFrame is now ready for further analysis or visualization.