Exploring and Transforming JSON Schemas

Introduction ¶

In this lesson, you'll formalize your knowledge for how to explore a JSON file whose structure and schema is unknown to you. This often happens in practice when you are handed a file or stumble upon one with little documentation.

Objectives

You will be able to:

- Use the JSON module to load and parse JSON documents
- Explore and extract data using unknown JSON schemas
- Convert JSON to a pandas dataframe

Loading the JSON file

As before, you'll begin by importing the `json` package, opening a file with python's built-in function, and then loading that data in.

In [17]:
```
import json
with open('output.json') as f:
    data = json.load(f)
```

Exploring JSON Schemas

Recall that JSON files have a nested structure. The most granular level of raw data will be individual numbers (float/int) and strings. These, in turn, will be stored in the equivalent of Python lists and dictionaries. Because these can be combined, you'll start exploring by checking the type of our root object and start mapping out the hierarchy of the JSON file.

In [18]:
```
type(data)
print(data)
```

```
{'albums': {'href': 'https://api.spotify.com/v1/browse/new-releases?country=SE&
offset=0&limit=20', 'items': [{'album_type': 'single', 'artists': [{'external_u
rls': {'spotify': 'https://open.spotify.com/artist/2RdwBSPQiwcmiDo9kixcl8'}, 'h
ref': 'https://api.spotify.com/v1/artists/2RdwBSPQiwcmiDo9kixcl8', 'id': '2RdwB
SPQiwcmiDo9kixcl8', 'name': 'Pharrell Williams', 'type': 'artist', 'uri': 'spot
ify:artist:2RdwBSPQiwcmiDo9kixcl8'}], 'available_markets': ['AD', 'AR', 'AT',
'AU', 'BE', 'BG', 'BO', 'BR', 'CA', 'CH', 'CL', 'CO', 'CR', 'CY', 'CZ', 'DE',
'DK', 'DO', 'EC', 'EE', 'ES', 'FI', 'FR', 'GB', 'GR', 'GT', 'HK', 'HN', 'HU',
'ID', 'IE', 'IS', 'IT', 'JP', 'LI', 'LT', 'LU', 'LV', 'MC', 'MT', 'MX', 'MY',
'NI', 'NL', 'NO', 'NZ', 'PA', 'PE', 'PH', 'PL', 'PT', 'PY', 'SE', 'SG', 'SK',
'SV', 'TR', 'TW', 'US', 'UY'], 'external_urls': {'spotify': 'https://open.spoti
fy.com/album/5ZX4m5aVSmWQ5iHAPQpT71'}, 'href': 'https://api.spotify.com/v1/albu
ms/5ZX4m5aVSmWQ5iHAPQpT71', 'id': '5ZX4m5aVSmWQ5iHAPQpT71', 'images': [{'heigh
t': 640, 'url': 'https://i.scdn.co/image/e6b635ebe3ef4ba22492f5698a7b5d417f78b8
8a', 'width': 640}, {'height': 300, 'url': 'https://i.scdn.co/image/92ae5b0fe64
870c09004dd2e745a4fb1bf7de39d', 'width': 300}, {'height': 64, 'url': 'https://
i.scdn.co/image/8a7ab6fc2c9f678308ba0f694ecd5718dc6bc930', 'width': 64}], 'nam
e': "Runnin'", 'type': 'album', 'uri': 'spotify:album:5ZX4m5aVSmWQ5iHAPQpT71'},
{'album_type': 'single', 'artists': [{'external_urls': {'spotify': 'https://ope
n.spotify.com/artist/3TVXtAsR1Inumwj472S9r4'}, 'href': 'https://api.spotify.co
m/v1/artists/3TVXtAsR1Inumwj472S9r4', 'id': '3TVXtAsR1Inumwj472S9r4', 'name':
'Drake', 'type': 'artist', 'uri': 'spotify:artist:3TVXtAsR1Inumwj472S9r4'}], 'a
vailable_markets': ['AD', 'AR', 'AT', 'AU', 'BE', 'BG', 'BO', 'BR', 'CH', 'CL',
'CO', 'CR', 'CY', 'CZ', 'DE', 'DK', 'DO', 'EC', 'EE', 'ES', 'FI', 'FR', 'GB',
'GR', 'GT', 'HK', 'HN', 'HU', 'ID', 'IE', 'IS', 'IT', 'JP', 'LI', 'LT', 'LU',
'LV', 'MC', 'MT', 'MY', 'NI', 'NL', 'NO', 'NZ', 'PA', 'PE', 'PH', 'PL', 'PT',
'PY', 'SE', 'SG', 'SK', 'SV', 'TR', 'TW', 'UY'], 'external_urls': {'spotify':
'https://open.spotify.com/album/0geTzdk2InlqIoB16fW9Nd'}, 'href': 'https://api.
spotify.com/v1/albums/0geTzdk2InlqIoB16fW9Nd', 'id': '0geTzdk2InlqIoB16fW9Nd',
'images': [{'height': 640, 'url': 'https://i.scdn.co/image/d40e9c3d22bde2fbdb2e
cc03cccd7a0e77f42e4c', 'width': 640}, {'height': 300, 'url': 'https://i.scdn.c
o/image/dff06a3375f6d9b32ecb081eb9a60bbafecb5731', 'width': 300}, {'height': 6
4, 'url': 'https://i.scdn.co/image/808a02bd7fc59b0652c9df9f68675edbffe07a79',
'width': 64}], 'name': 'Sneakin’', 'type': 'album', 'uri': 'spotify:album:0geTz
dk2InlqIoB16fW9Nd'}], 'limit': 20, 'next': 'https://api.spotify.com/v1/browse/n
ew-releases?country=SE&offset=20&limit=20', 'offset': 0, 'previous': None, 'tot
al': 500}}
```

As you can see, in this case, the first level of the hierarchy is a dictionary. Let's explore what keys are within this:

In [19]:
```
data.keys()
```

Out[19]:
```
dict_keys(['albums'])
```

In this case, there is only a single key, `'albums'` , so can continue exploring linearly without branching out.

Once again, start by checking the type of this nested data structure.

In [20]:
```
type(data['albums'])
```

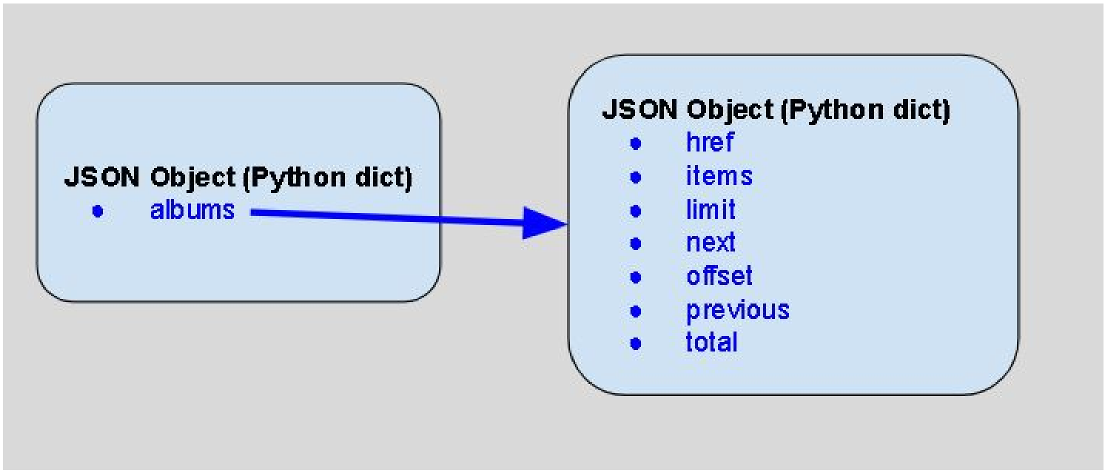
Out[20]:
```
dict
```

Another dictionary! So thus far, you have a dictionary within a dictionary. Once again, investigate what's within this dictionary.

In [21]:
```
data['albums'].keys()
```

Out[21]:
```
dict_keys(['href', 'items', 'limit', 'next', 'offset', 'previous', 'total'])
```

At this point, things are starting to look something like this:

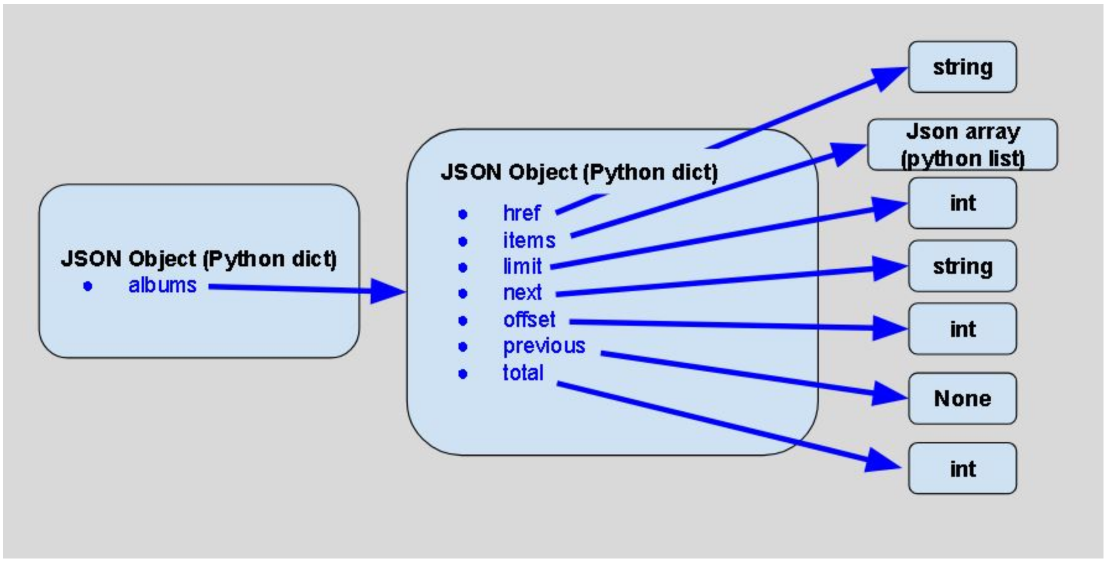


At this point, if you were to continue checking individual data types, you have a lot to go through. To simplify this, you can use a for loop:

In [22]:
```
for key in data['albums'].keys():
    print(key, type(data['albums'][key]))
```

```
href <class 'str'>
items <class 'list'>
limit <class 'int'>
next <class 'str'>
offset <class 'int'>
previous <class 'NoneType'>
total <class 'int'>
```

Adding this to our diagram we now have something like this:



Normally, you may not draw out the full diagram as done here, but it's a useful picture to have in mind, and in complex schemas, can be useful to map out. At this point, you also probably have a good idea of the general structure of the JSON file. However, there is still the list of items, which we could investigate further:

```
In [23]: type(data['albums']['items'])
```

```
Out[23]: list
```

```
In [24]: len(data['albums']['items'])
```

```
Out[24]: 2
```

```
In [25]: type(data['albums']['items'][0])
```

```
Out[25]: dict
```

```
In [26]: data['albums']['items'][0].keys()
```

```
Out[26]: dict_keys(['album_type', 'artists', 'available_markets', 'external_urls', 'hre
f', 'id', 'images', 'name', 'type', 'uri'])
```

Converting JSON to Alternative Data Formats

As you can see, the nested structure continues on: our list of items is only 2 long, but each item is a dictionary with a large number of key-value pairs. To add context, this is actually the data that you're probably after from this file: its that data providing details about what albums were recently released. The entirety of the JSON file itself is an example response from the Spotify API (more on that soon). So while the larger JSON provides us with many details about the response itself, our primary interest may simply be the list of dictionaries within data -> albums -> items. Preview this and see if you can transform it into our usual pandas DataFrame.

```
In [27]: import pandas as pd
```

On first attempt, you might be tempted to pass the whole object to Pandas. Try and think about what you would like the resulting dataframe to look like based on the schema we are mapping out. What would the column names be? What would the rows represent?

In [28]:
```
df = pd.DataFrame(data['albums']['items'])
df
```

Out[28]:

	album_type	artists	available_markets	external_urls	
0	single	[{'external_urls': {'spotify': 'https://open.s...	[AD, AR, AT, AU, BE, BG, BO, BR, CA, CH, CL, C...	{'spotify': 'https://open.spotify.com/album/5Z...	https://api.spotify.c
1	single	[{'external_urls': {'spotify': 'https://open.s...	[AD, AR, AT, AU, BE, BG, BO, BR, CH, CL, CO, C...	{'spotify': 'https://open.spotify.com/album/0g...	https://api.spc

Pandas DataFrames are mostly useful when we have fairly flat, tabular data. In this case, with a list of only two albums, we don't gain much information by displaying our data this way.

Extracting Data

Now that we have some sense of what is in our dataset, we can extract some information that might be useful as part of a larger analysis.

What are the artist names?

Although we don't have a schema available, we can see that each album contains a key `'artists'` . Let's explore that further.

In [29]:
```
first_album = data['albums']['items'][0]

first_album['artists']
```

Out[29]:
```
[{'external_urls': {'spotify': 'https://open.spotify.com/artist/2RdwBSPQiwcmiDo
9kixcl8'},
  'href': 'https://api.spotify.com/v1/artists/2RdwBSPQiwcmiDo9kixcl8',
  'id': '2RdwBSPQiwcmiDo9kixcl8',
  'name': 'Pharrell Williams',
  'type': 'artist',
  'uri': 'spotify:artist:2RdwBSPQiwcmiDo9kixcl8'}]
```

That's a list of dictionaries. To convert it to a list of strings, that would be something like this:

In [30]:
```
first_album_artists = [artist['name'] for artist in first_album['artists']]
first_album_artists
```

Out[30]:
```
['Pharrell Williams']
```

If we wanted to do the same for all albums in the dataset, that would be something like this:

In [31]:
```
artists_by_album = [[artist['name'] for artist in album['artists']] for album in
artists_by_album
```

Out[31]:
```
[['Pharrell Williams'], ['Drake']]
```

That same logic, without list comprehensions:

In [32]:
```
# Make an empty list to hold the overall data
artists_by_album = []

# Loop over the list of dictionaries containing album info
for album in data['albums']['items']:
    # Make a list to contain the artist names for this album
    artist_names = []
    # Loop over the list of dictionaries containing artist info
    for artist in album['artists']:
        # Add the artist name to the list of artist names
        artist_names.append(artist['name'])
    # Add the list of artists for this album to the overall list
    artists_by_album.append(artist_names)

artists_by_album
```

Out[32]:
```
[['Pharrell Williams'], ['Drake']]
```

That same logic using the dataframe would be:

In [33]:
```
def extract_artist_names(record):
    return [artist['name'] for artist in record]
```

In [35]:
```
df["artists"].apply(extract_artist_names)
```

Out[35]:
```
0    [Pharrell Williams]
1                [Drake]
Name: artists, dtype: object
```

How many available markets are there per album?

We see that one of the keys each album has is `'available_markets'` . Let's look at it:

In [37]:
```
first_album['available_markets']
```

Out[37]:
```
['AD',
 'AR',
 'AT',
 'AU',
 'BE',
 'BG',
 'BO',
 'BR',
 'CA',
 'CH',
 'CL',
 'CO',
 'CR',
 'CY',
 'CZ',
 'DE',
 'DK',
 'DO',
 'EC',
 'EE',
 'ES',
 'FI',
 'FR',
 'GB',
 'GR',
 'GT',
 'HK',
 'HN',
 'HU',
 'ID',
 'IE',
 'IS',
 'IT',
 'JP',
 'LI',
 'LT',
 'LU',
 'LV',
 'MC',
 'MT',
 'MX',
 'MY',
 'NI',
 'NL',
 'NO',
 'NZ',
 'PA',
 'PE',
 'PH',
 'PL',
 'PT',
 'PY',
 'SE',
 'SG',
 'SK',
 'SV',
 'TR',
 'TW',
 'US',
 'UY']
```

It appears we have a list of strings. So all we need to do is to count the length of that list for each album.

In [38]:
```
available_market_counts = [len(album['available_markets']) for album in data['alb
available_market_counts
```

Out[38]:
```
[60, 57]
```

Again, it would be possible to do this using pandas instead:

In [42]:
```
df["available_markets"].apply(len)
```

Out[42]:
```
0    60
1    57
Name: available_markets, dtype: int64
```

What is the medium-sized image associated with each album?

We see that the `'images'` key is associated with a list of dictionaries:

In [43]:
```
first_album['images']
```

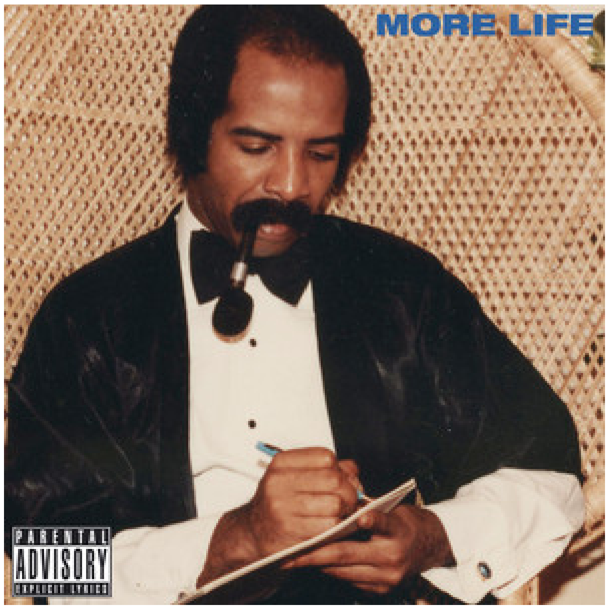
Out[43]:
```
[{'height': 640,
  'url': 'https://i.scdn.co/image/e6b635ebe3ef4ba22492f5698a7b5d417f78b88a',
  'width': 640},
 {'height': 300,
  'url': 'https://i.scdn.co/image/92ae5b0fe64870c09004dd2e745a4fb1bf7de39d',
  'width': 300},
 {'height': 64,
  'url': 'https://i.scdn.co/image/8a7ab6fc2c9f678308ba0f694ecd5718dc6bc930',
  'width': 64}]
```

Let's use IPython to display the image with `'height'` 300 for each album.

In [44]:
```python
from IPython.display import Image

for album in data['albums']['items']:
    for image in album['images']:
        if image['height'] == 300:
            loaded_image = Image(url=image['url'])
            display(loaded_image)
```

HIDDEN
FIGURES
NASA

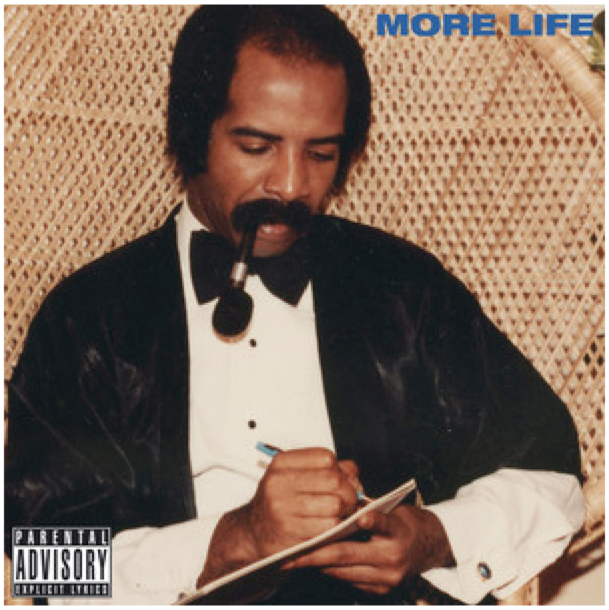


Once again, the same logic would be possible with pandas:

In [24]:
```
def extract_medium_images(record):
    images = pd.DataFrame(record)
    medium_image = images[images["height"] == 300]["url"].values[0]
    return medium_image

def display_image(record):
    loaded_image = Image(url=record)
    display(loaded_image)
```

```
In [25]: df["images"].apply(extract_medium_images).apply(display_image);
```



As you can see, once we have explored the schema somewhat, there are a lot of different things we can extract from this dataset. You will get more practice with these skills in the upcoming lab!

Summary

JSON files often have a deep, nested structure that can require initial investigation into the schema hierarchy in order to become familiar with how data is stored. Once done, it is important to identify what data you are looking to extract and then develop a strategy to transform it using your standard workflow.