11.1 The JSON Data Format

May 9, 2019

1 Chapter 11. Hierarchical Data

A lot of data in the real world is naturally hierarchical. For example, consider a data set of concert programs by the New York Philharmonic, one of the world's leading orchestras. Each program consists of one or more works of music and is performed at one or more concerts. Furthermore, each work of music may feature any number of soloists.

How would we represent this information in a single `DataFrame`? If each row represents a single program, then we need one column for each concert that the program appeared in. This is wasteful because some programs may have only appeared in one concert. We still need to keep around M "concert" columns, where M is the maximum number of concerts that any program appeared in.

concert1	concert2	...	concertM	work1	work2	...	workN
2016-12-11	`NaN`	...	`NaN`	Violin Concerto No. 2	Symphony No. 5	...	`NaN`
2016-12-13	2016-12-14	...	2016-12-17	Messiah	`NaN`	...	`NaN`
...	...	...	...	...	...	...	...

Similarly, we need one column for each work in the program. The number of "work" columns has to be equal to the maximum number of works on any program, even though most programs may have had far fewer works.

Hopefully, it is clear that a single `DataFrame` is an inefficient way to represent hierarchical data—and we haven't even tried to include information about the soloists who performed in each work. This chapter is about efficient ways to represent hierarchical data, like the New York Philharmonic data set described above.

2 Chapter 11.1 The JSON Data Format

The JavaScript Object Notation, or **JSON**, data format is a popular way to represent hierarchical data. Despite its name, its application extends far beyond JavaScript, the language for which it was originally designed.

Let's take a look at the first 1000 characters of a JSON file. (*Warning:* Never try to print the entire contents of a JSON file in a Jupyter notebook; this will freeze the notebook if the file is large!)

```
In [1]: !head -c 1000 /data301/data/nyphil/complete.json
```

```
{
  "programs": [
    {
      "id": "00646b9f-fec7-4ffb-9fb1-faae410bd9dc-0.1",
      "programID": "3853",
      "orchestra": "New York Philharmonic",
      "season": "1842-43",
      "concerts": [
        {
          "eventType": "Subscription Season",
          "Location": "Manhattan, NY",
          "Venue": "Apollo Rooms",
          "Date": "1842-12-07T05:00:00Z",
          "Time": "8:00PM"
        }
      ],
      "works": [
        {
          "ID": "52446*",
          "composerName": "Beethoven,  Ludwig  van",
          "workTitle": "SYMPHONY NO. 5 IN C MINOR, OP.67",
          "conductorName": "Hill, Ureli Corelli",
          "soloists": []
        },
        {
          "ID": "8834*4",
          "composerName": "Weber,  Carl  Maria Von",
          "workTitle": "OBERON",
          "movement": "\"Ozean, du Ungeheuer\" (Ocean, thou mighty monster), Reiza (Scene and A
          "conductorName": "Timm, Henry C.",
          "soloists": [
            {
              "sol
```


Hopefully, this notation is familiar. It is just the notation for a Python dictionary! Although there are a few cosmetic differences between Python dicts and JSON, they are the same for the most part, and we will use the terms "dict" and "JSON" interchangeably.

The `json` library in Python allows you to read a JSON file directly into a Python dict.

```
In [2]: import json

        with open("/data301/data/nyphil/complete.json") as f:
            nyphil = json.load(f)
```

Let's take a look at this Python dict that we just created, again being careful not to print out the entire dict. Let's just take a look at the first two programs in the data set. This should hopefully be enough to give you a sense of how the data is structured.

```
In [3]: nyphil["programs"][:2]
```

```
Out[3]: [{'id': '00646b9f-fec7-4ffb-9fb1-faae410bd9dc-0.1',
          'programID': '3853',
          'orchestra': 'New York Philharmonic',
          'season': '1842-43',
          'concerts': [{'eventType': 'Subscription Season',
            'Location': 'Manhattan, NY',
            'Venue': 'Apollo Rooms',
            'Date': '1842-12-07T05:00:00Z',
            'Time': '8:00PM'}],
          'works': [{'ID': '52446*',
            'composerName': 'Beethoven,  Ludwig  van',
            'workTitle': 'SYMPHONY NO. 5 IN C MINOR, OP.67',
            'conductorName': 'Hill, Ureli Corelli',
            'soloists': []},
           {'ID': '8834*4',
            'composerName': 'Weber,  Carl  Maria Von',
            'workTitle': 'OBERON',
            'movement': '"Ozean, du Ungeheuer" (Ocean, thou mighty monster), Reiza (Scene and A
            'conductorName': 'Timm, Henry C.',
            'soloists': [{'soloistName': 'Otto, Antoinette',
              'soloistInstrument': 'Soprano',
              'soloistRoles': 'S'}]},
           {'ID': '3642*',
            'composerName': 'Hummel,  Johann',
            'workTitle': 'QUINTET, PIANO, D MINOR, OP. 74',
            'soloists': [{'soloistName': 'Scharfenberg, William',
              'soloistInstrument': 'Piano',
              'soloistRoles': 'A'},
             {'soloistName': 'Hill, Ureli Corelli',
              'soloistInstrument': 'Violin',
              'soloistRoles': 'A'},
             {'soloistName': 'Derwort, G. H.',
              'soloistInstrument': 'Viola',
              'soloistRoles': 'A'},
             {'soloistName': 'Boucher, Alfred',
              'soloistInstrument': 'Cello',
              'soloistRoles': 'A'},
             {'soloistName': 'Rosier, F. W.',
              'soloistInstrument': 'Double Bass',
              'soloistRoles': 'A'}]},
           {'ID': '0*', 'interval': 'Intermission', 'soloists': []},
           {'ID': '8834*3',
            'composerName': 'Weber,  Carl  Maria Von',
            'workTitle': 'OBERON',
            'movement': 'Overture',
            'conductorName': 'Etienne, Denis G.',
            'soloists': []},
           {'ID': '8835*1',
```

```
   'composerName': 'Rossini,  Gioachino',
   'workTitle': 'ARMIDA',
   'movement': 'Duet',
   'conductorName': 'Timm, Henry C.',
   'soloists': [{'soloistName': 'Otto, Antoinette',
     'soloistInstrument': 'Soprano',
     'soloistRoles': 'S'},
    {'soloistName': 'Horn, Charles Edward',
     'soloistInstrument': 'Tenor',
     'soloistRoles': 'S'}]},
  {'ID': '8837*6',
   'composerName': 'Beethoven,  Ludwig  van',
   'workTitle': 'FIDELIO, OP. 72',
   'movement': '"In Des Lebens Fruhlingstagen...O spur ich nicht linde," Florestan (a
   'conductorName': 'Timm, Henry C.',
   'soloists': [{'soloistName': 'Horn, Charles Edward',
     'soloistInstrument': 'Tenor',
     'soloistRoles': 'S'}]},
  {'ID': '8336*4',
   'composerName': 'Mozart,  Wolfgang  Amadeus',
   'workTitle': 'ABDUCTION FROM THE SERAGLIO,THE, K.384',
   'movement': '"Ach Ich liebte," Konstanze (aria)',
   'conductorName': 'Timm, Henry C.',
   'soloists': [{'soloistName': 'Otto, Antoinette',
     'soloistInstrument': 'Soprano',
     'soloistRoles': 'S'}]},
  {'ID': '5543*',
   'composerName': 'Kalliwoda,  Johann  W.',
   'workTitle': 'OVERTURE NO. 1, D MINOR, OP. 38',
   'conductorName': 'Timm, Henry C.',
   'soloists': []}]},
{'id': '1118e84e-eb59-46cc-9119-d903375e65e6-0.1',
 'programID': '5178',
 'orchestra': 'New York Philharmonic',
 'season': '1842-43',
 'concerts': [{'eventType': 'Subscription Season',
   'Location': 'Manhattan, NY',
   'Venue': 'Apollo Rooms',
   'Date': '1843-02-18T05:00:00Z',
   'Time': '8:00PM'}],
 'works': [{'ID': '52437*',
   'composerName': 'Beethoven,  Ludwig  van',
   'workTitle': 'SYMPHONY NO. 3 IN E FLAT MAJOR, OP. 55 (EROICA)',
   'conductorName': 'Hill, Ureli Corelli',
   'soloists': []},
  {'ID': '8838*2',
   'composerName': 'Bellini,  Vincenzo',
   'workTitle': 'I PURITANI',
```

```
 'movement': 'Elvira (aria): "Qui la voce...Vien, diletto"',
 'conductorName': 'Hill, Ureli Corelli',
 'soloists': [{'soloistName': 'Otto, Antoinette',
   'soloistInstrument': 'Soprano',
   'soloistRoles': 'S'}]},
{'ID': '3659*',
 'composerName': 'Romberg,  Bernhard',
 'workTitle': 'CELEBRATED ELEGIE',
 'conductorName': 'Hill, Ureli Corelli',
 'soloists': [{'soloistName': 'Boucher, Alfred',
   'soloistInstrument': 'Cello',
   'soloistRoles': 'S'}]},
{'ID': '0*', 'interval': 'Intermission', 'soloists': []},
{'ID': '8839*2',
 'composerName': 'Rossini,  Gioachino',
 'workTitle': 'WILLIAM TELL',
 'movement': 'Overture',
 'conductorName': 'Alpers, William',
 'soloists': []},
{'ID': '53076*2',
 'composerName': 'Rossini,  Gioachino',
 'workTitle': 'STABAT MATER',
 'movement': 'Inflammatus et Accensus (Aria with Chorus)',
 'conductorName': 'Alpers, William',
 'soloists': [{'soloistName': 'Otto, Antoinette',
   'soloistInstrument': 'Soprano',
   'soloistRoles': 'S'}]},
{'ID': '51568*2',
 'composerName': 'Hummel,  Johann',
 'workTitle': 'CONCERTO, PIANO, A-FLAT MAJOR, OP. 113',
 'movement': 'Romanza: Larghetto con moto',
 'conductorName': 'Alpers, William',
 'soloists': [{'soloistName': 'Timm, Henry C.',
   'soloistInstrument': 'Piano',
   'soloistRoles': 'S'}]},
{'ID': '51568*3',
 'composerName': 'Hummel,  Johann',
 'workTitle': 'CONCERTO, PIANO, A-FLAT MAJOR, OP. 113',
 'movement': 'Rondo alla spagniola: Allegro moderato',
 'conductorName': 'Alpers, William',
 'soloists': [{'soloistName': 'Timm, Henry C.',
   'soloistInstrument': 'Piano',
   'soloistRoles': 'S'}]},
{'ID': '6709*16',
 'composerName': 'Weber,  Carl  Maria Von',
 'workTitle': 'FREISCHUTZ, DER',
 'movement': 'Overture',
 'conductorName': 'Alpers, William',
```

```
        'soloists': []}]}]
```

The top-level variables in each "program" are:

- concerts
- id
- orchestra
- programID
- season
- works

Most of these variables are fairly standard; the only interesting ones are "concerts" and "works", which are both lists. A variable that is a list is called a **repeated field**. A repeated field might itself consist of several variables (for example, each "work" has a composer, a conductor, and soloists), thus creating a hierarchy of variables. Repeated fields are what makes a data set hierarchical.

3 Flattening Hierarchical Data

How many distinct works by Ludwig van Beethoven has the New York Philharmonic performed? Answering this question from the Python dict is irritating, as it involves writing multiple nested "for" loops to traverse the JSON data. Shown below is the code to do this, although we will see an easier way shortly.

```
In [4]: # Spaghetti Code (Don't do this --- see below for an easier way.)
        beethoven = set()
        for program in nyphil["programs"]:
            for work in program["works"]:
                if "composerName" in work and work["composerName"] == "Beethoven,  Ludwig  van"
                    beethoven.add(work["workTitle"])

        len(beethoven)

Out[4]: 144
```

The only data that we really need to answer the above question is a `DataFrame` of works that the New York Philharmonic has performed. To obtain such a `DataFrame`, we need to **flatten** the JSON data at the level of "work" to produce a `DataFrame` with one row per work. The `json_normalize()` in `pandas.io.json` is a function that allows us to flatten JSON data at any desired level. The first argument to `json_normalize()` is the JSON data (i.e., a Python dict), and the second argument specifies the level at which to flatten.

```
In [5]: import pandas as pd
        from pandas.io.json import json_normalize
        pd.options.display.max_rows = 10

        works = json_normalize(nyphil["programs"], "works")
        works.head()
```

```
Out[5]:        ID             composerName        conductorName      interval  \
        0  52446*  Beethoven,  Ludwig  van  Hill, Ureli Corelli           NaN
        1  8834*4  Weber,  Carl  Maria Von       Timm, Henry C.           NaN
        2   3642*          Hummel,  Johann                  NaN           NaN
        3      0*                     NaN                  NaN  Intermission
        4  8834*3  Weber,  Carl  Maria Von    Etienne, Denis G.           NaN

                                                 movement  \
        0                                             NaN
        1  "Ozean, du Ungeheuer" (Ocean, thou mighty mons...
        2                                             NaN
        3                                             NaN
        4                                        Overture

                                                 soloists  \
        0                                              []
        1  [{'soloistName': 'Otto, Antoinette', 'soloistI...
        2  [{'soloistName': 'Scharfenberg, William', 'sol...
        3                                              []
        4                                              []

                             workTitle
        0  SYMPHONY NO. 5 IN C MINOR, OP.67
        1                           OBERON
        2   QUINTET, PIANO, D MINOR, OP. 74
        3                              NaN
        4                           OBERON
```

Note that this flattening operation resulted in some loss of information. We no longer have information about the program that each work appeared in. We can partly alleviate this problem by specifying "metadata" from parent levels to append. For example, "season" and "orchestra" are properties of "program", which is the parent of "work". If we want to include these variables with each work, then we pass them to the `meta=` argument of `json_normalize()`.

```
In [6]: json_normalize(nyphil["programs"], "works", meta=["season", "orchestra"])
```

```
Out[6]:           ID               composerName        conductorName      interval  \
        0      52446*    Beethoven,  Ludwig  van  Hill, Ureli Corelli           NaN
        1      8834*4    Weber,  Carl  Maria Von       Timm, Henry C.           NaN
        2       3642*            Hummel,  Johann                  NaN           NaN
        3          0*                       NaN                  NaN  Intermission
        4      8834*3    Weber,  Carl  Maria Von    Etienne, Denis G.           NaN
        ...       ...                       ...                  ...           ...
        83314  52446*    Beethoven,  Ludwig  van        Gilbert, Alan           NaN
        83315  53976*  Handel,  George  Frideric        Manze, Andrew           NaN
        83316      0*                       NaN                  NaN  Intermission
        83317  53976*  Handel,  George  Frideric        Manze, Andrew           NaN
        83318      0*                       NaN                  NaN  Intermission
```

```
                                                  movement  \
0                                                      NaN
1      "Ozean, du Ungeheuer" (Ocean, thou mighty mons...
2                                                      NaN
3                                                      NaN
4                                                 Overture
...                                                    ...
83314                                                  NaN
83315                                                  NaN
83316                                                  NaN
83317                                                  NaN
83318                                                  NaN

                                                  soloists  \
0                                                       []
1      [{'soloistName': 'Otto, Antoinette', 'soloistI...
2      [{'soloistName': 'Scharfenberg, William', 'sol...
3                                                       []
4                                                       []
...                                                    ...
83314                                                   []
83315  [{'soloistName': 'Harvey, Joelle [Joélle]', 's...
83316                                                   []
83317  [{'soloistName': 'Harvey, Joelle [Joélle]', 's...
83318                                                   []

                             workTitle   season              orchestra
0      SYMPHONY NO. 5 IN C MINOR, OP.67  1842-43  New York Philharmonic
1                                OBERON  1842-43  New York Philharmonic
2       QUINTET, PIANO, D MINOR, OP. 74  1842-43  New York Philharmonic
3                                   NaN  1842-43  New York Philharmonic
4                                OBERON  1842-43  New York Philharmonic
...                                 ...      ...                   ...
83314  SYMPHONY NO. 5 IN C MINOR, OP.67  2017-18  New York Philharmonic
83315                           MESSIAH  2017-18  New York Philharmonic
83316                               NaN  2017-18  New York Philharmonic
83317                           MESSIAH  2017-18  New York Philharmonic
83318                               NaN  2017-18  New York Philharmonic

[83319 rows x 9 columns]
```

However, there is still some loss of information. For example, there is no way to tell from this flattened `DataFrame` which works appeared together on the same program. (In the case of this particular data set, there is a "programID" that could be used to preserve information about the program, but not all data sets will have such an ID.)

Note also that repeated fields that are nested within "work", such as "soloist", remain unflattened. They simply remain as a list of JSON objects embedded within the `DataFrame`. They are

not particularly accessible to analysis.

But now that we have a `DataFrame` with one row per work, we can determine the number of unique Beethoven works that the Philharmonic has performed by subsetting the `DataFrame` and grouping by the title of the work.

```
In [7]: beethoven = works[works.composerName == "Beethoven,  Ludwig  van"]
        len(beethoven.groupby("workTitle")["ID"].count())
```

```
Out[7]: 144
```

What if we wanted to know how many works Benny Goodman has performed with the New York Philharmonic? We could flatten the data at the level of the "soloist". Since "soloists" is nested within "works", we specify a path (i.e., `["works", "soloists"]`) as the flattening level.

```
In [8]: soloists = json_normalize(nyphil["programs"], ["works", "soloists"])
        soloists
```

```
Out[8]:       soloistInstrument                 soloistName soloistRoles
        0               Soprano            Otto, Antoinette            S
        1                 Piano       Scharfenberg, William            A
        2                Violin         Hill, Ureli Corelli            A
        3                 Viola              Derwort, G. H.            A
        4                 Cello             Boucher, Alfred            A
        ...                 ...                         ...          ...
        56926           Soprano     Harvey, Joelle [Joélle]            S
        56927     Mezzo-Soprano      Johnson Cano, Jennifer            S
        56928             Tenor                  Bliss, Ben            S
        56929          Baritone               Duncan, Tyler            S
        56930            Chorus  Westminster Symphonic Choir            S

        [56931 rows x 3 columns]
```

Now we can use this flattened `DataFrame` to easily answer the question.

```
In [9]: (soloists["soloistName"] == "Goodman, Benny").sum()
```

```
Out[9]: 25
```

If we wanted to know how many works by Mozart that Goodman performed, we need to additionally store the "composerName" from the "works" level. We do this by specifying the path to "composerName" (i.e., `["works", "soloists"]`) in the `meta=` argument. But there is a catch. There are some works where the "composerName" field is missing. `json_normalize()` will fail if it cannot find the "composerName" key for even a single work. So we have to manually go through the JSON object and manually add "composerName" to the object, setting its value to `None`, if it does not exist.

```
In [10]: for program in nyphil["programs"]:
             for work in program["works"]:
                 if "composerName" not in work:
                     work["composerName"] = None
```

```
In [11]: soloists = json_normalize(
             nyphil["programs"],
             ["works", "soloists"],
             meta=[["works", "composerName"], "season"]
         )
         soloists
```

Out[11]:

	soloistInstrument	soloistName	soloistRoles	works.composerName	season
0	Soprano	Otto, Antoinette	S	Weber, Carl Maria Von	1842-43
1	Piano	Scharfenberg, William	A	Hummel, Johann	1842-43
2	Violin	Hill, Ureli Corelli	A	Hummel, Johann	1842-43
3	Viola	Derwort, G. H.	A	Hummel, Johann	1842-43
4	Cello	Boucher, Alfred	A	Hummel, Johann	1842-43
...	...	...	...	...	...
56926	Soprano	Harvey, Joelle [Joélle]	S	Handel, George Frideric	2017-18
56927	Mezzo-Soprano	Johnson Cano, Jennifer	S	Handel, George Frideric	2017-18
56928	Tenor	Bliss, Ben	S	Handel, George Frideric	2017-18
56929	Baritone	Duncan, Tyler	S	Handel, George Frideric	2017-18
56930	Chorus	Westminster Symphonic Choir	S	Handel, George Frideric	2017-18

[56931 rows x 5 columns]

```
In [12]: soloists[soloists["soloistName"] == "Goodman, Benny"]["works.composerName"].value_cou
```

Out[12]:

```
Mozart, Wolfgang Amadeus      3
Weber, Carl Maria Von         3
Sauter, Eddie                 2
Gershwin, George              2
Confrey, Zez                  1
                             ..
Baxter, Phil                  1
Cannon, Hughie                1
Sampson, Edgar                1
Handy, William Christopher    1
Debussy, Claude               1
Name: works.composerName, Length: 19, dtype: int64
```

4 RESTful Web Services

One way that organizations expose their data to the public is through RESTful web services. In a typical RESTful service, the user specifies the kind of data they want in the URL, and the server returns the desired data. JSON is a common format for returning data.

For example, the Star Wars API is a RESTful web service that returns data about the Star Wars universe, including characters, spaceships, and planets. To look up information about characters named "Skywalker", we would issue an HTTP request to the URL http://swapi.co/api/people/?search=skywalker. Notice that this returns data in JSON format.

To issue the HTTP request within Python (so that we can further process the JSON), we can use the `requests` library in Python.

```
In [13]: import requests
         resp = requests.get("http://swapi.co/api/people/?search=skywalker")
         resp

Out[13]: <Response [200]>
```

The response object contains the JSON and other metadata. To extract the JSON in the form of a Python dict, we call `.json()` on the response object.

```
In [14]: skywalker = resp.json()
         skywalker

Out[14]: {'count': 3,
          'next': None,
          'previous': None,
          'results': [{'name': 'Luke Skywalker',
            'height': '172',
            'mass': '77',
            'hair_color': 'blond',
            'skin_color': 'fair',
            'eye_color': 'blue',
            'birth_year': '19BBY',
            'gender': 'male',
            'homeworld': 'https://swapi.co/api/planets/1/',
            'films': ['https://swapi.co/api/films/2/',
             'https://swapi.co/api/films/6/',
             'https://swapi.co/api/films/3/',
             'https://swapi.co/api/films/1/',
             'https://swapi.co/api/films/7/'],
            'species': ['https://swapi.co/api/species/1/'],
            'vehicles': ['https://swapi.co/api/vehicles/14/',
             'https://swapi.co/api/vehicles/30/'],
            'starships': ['https://swapi.co/api/starships/12/',
             'https://swapi.co/api/starships/22/'],
            'created': '2014-12-09T13:50:51.644000Z',
            'edited': '2014-12-20T21:17:56.891000Z',
            'url': 'https://swapi.co/api/people/1/'},
```

```
 {'name': 'Anakin Skywalker',
  'height': '188',
  'mass': '84',
  'hair_color': 'blond',
  'skin_color': 'fair',
  'eye_color': 'blue',
  'birth_year': '41.9BBY',
  'gender': 'male',
  'homeworld': 'https://swapi.co/api/planets/1/',
  'films': ['https://swapi.co/api/films/5/',
   'https://swapi.co/api/films/4/',
   'https://swapi.co/api/films/6/'],
  'species': ['https://swapi.co/api/species/1/'],
  'vehicles': ['https://swapi.co/api/vehicles/44/',
   'https://swapi.co/api/vehicles/46/'],
  'starships': ['https://swapi.co/api/starships/59/',
   'https://swapi.co/api/starships/65/',
   'https://swapi.co/api/starships/39/'],
  'created': '2014-12-10T16:20:44.310000Z',
  'edited': '2014-12-20T21:17:50.327000Z',
  'url': 'https://swapi.co/api/people/11/'},
 {'name': 'Shmi Skywalker',
  'height': '163',
  'mass': 'unknown',
  'hair_color': 'black',
  'skin_color': 'fair',
  'eye_color': 'brown',
  'birth_year': '72BBY',
  'gender': 'female',
  'homeworld': 'https://swapi.co/api/planets/1/',
  'films': ['https://swapi.co/api/films/5/', 'https://swapi.co/api/films/4/'],
  'species': ['https://swapi.co/api/species/1/'],
  'vehicles': [],
  'starships': [],
  'created': '2014-12-19T17:57:41.191000Z',
  'edited': '2014-12-20T21:17:50.401000Z',
  'url': 'https://swapi.co/api/people/43/'}]}
```

```
In [15]: from pandas.io.json import json_normalize
```

Now we can process this data just like we did with the JSON data that we read in from a file.

```
In [16]: json_normalize(skywalker, "results")
```

```
Out[16]:   birth_year                     created                      edited  \
         0      19BBY  2014-12-09T13:50:51.644000Z  2014-12-20T21:17:56.891000Z
         1    41.9BBY  2014-12-10T16:20:44.310000Z  2014-12-20T21:17:50.327000Z
         2      72BBY  2014-12-19T17:57:41.191000Z  2014-12-20T21:17:50.401000Z
```

```
  eye_color                                              films  gender  \
0      blue  [https://swapi.co/api/films/2/, https://swapi...    male
1      blue  [https://swapi.co/api/films/5/, https://swapi...    male
2     brown  [https://swapi.co/api/films/5/, https://swapi...  female

  hair_color height                         homeworld     mass  \
0      blond    172  https://swapi.co/api/planets/1/       77
1      blond    188  https://swapi.co/api/planets/1/       84
2      black    163  https://swapi.co/api/planets/1/  unknown

               name skin_color                               species  \
0    Luke Skywalker       fair  [https://swapi.co/api/species/1/]
1  Anakin Skywalker       fair  [https://swapi.co/api/species/1/]
2    Shmi Skywalker       fair  [https://swapi.co/api/species/1/]

                                          starships  \
0  [https://swapi.co/api/starships/12/, https://s...
1  [https://swapi.co/api/starships/59/, https://s...
2                                                 []

                              url  \
0   https://swapi.co/api/people/1/
1  https://swapi.co/api/people/11/
2  https://swapi.co/api/people/43/

                                           vehicles
0  [https://swapi.co/api/vehicles/14/, https://sw...
1  [https://swapi.co/api/vehicles/44/, https://sw...
2                                                 []
```

5 Ethical Enlightenment: Staggering Requests

Suppose you want information about the starships associated with the Skywalkers you found above. If we flatten the JSON object at the "starships" level, then we get a list of URLs that we can query to get information about each starship.

```
In [17]: starship_urls = json_normalize(skywalker, ["results", "starships"])
         starship_urls
```

```
Out[17]:                                    0
         0  https://swapi.co/api/starships/12/
         1  https://swapi.co/api/starships/22/
         2  https://swapi.co/api/starships/59/
         3  https://swapi.co/api/starships/65/
         4  https://swapi.co/api/starships/39/
```

It is straightforward enough to write a loop that queries each of these URLs and saves the corresponding JSON object. However, a script can easily issue hundreds, even thousands, of

queries per second, and we want to avoid spamming the server. (In fact, if a website detects many requests coming from the same IP address, it may think it is being attacked and block the IP address.)

To respect the host, who is providing this information for free, we stagger the queries by inserting a delay. This can be done using `time.sleep()`, which will suspend execution of the script for the given number of seconds. We will add a half second delay (so that we make no more than 2 queries per second) between requests.

```
In [18]: import time

         starships = []
         for starship_url in starship_urls[0]:

             # get the JSON for the starship from the REST API
             resp = requests.get(starship_url)
             starships.append(resp.json())

             # add a 0.5 second delay between each query
             time.sleep(0.5)

         starships
```

```
Out[18]: [{'name': 'X-wing',
           'model': 'T-65 X-wing',
           'manufacturer': 'Incom Corporation',
           'cost_in_credits': '149999',
           'length': '12.5',
           'max_atmosphering_speed': '1050',
           'crew': '1',
           'passengers': '0',
           'cargo_capacity': '110',
           'consumables': '1 week',
           'hyperdrive_rating': '1.0',
           'MGLT': '100',
           'starship_class': 'Starfighter',
           'pilots': ['https://swapi.co/api/people/1/',
            'https://swapi.co/api/people/9/',
            'https://swapi.co/api/people/18/',
            'https://swapi.co/api/people/19/'],
           'films': ['https://swapi.co/api/films/2/',
            'https://swapi.co/api/films/3/',
            'https://swapi.co/api/films/1/'],
           'created': '2014-12-12T11:19:05.340000Z',
           'edited': '2014-12-22T17:35:44.491233Z',
           'url': 'https://swapi.co/api/starships/12/'},
          {'name': 'Imperial shuttle',
           'model': 'Lambda-class T-4a shuttle',
           'manufacturer': 'Sienar Fleet Systems',
```

```
 'cost_in_credits': '240000',
 'length': '20',
 'max_atmosphering_speed': '850',
 'crew': '6',
 'passengers': '20',
 'cargo_capacity': '80000',
 'consumables': '2 months',
 'hyperdrive_rating': '1.0',
 'MGLT': '50',
 'starship_class': 'Armed government transport',
 'pilots': ['https://swapi.co/api/people/1/',
  'https://swapi.co/api/people/13/',
  'https://swapi.co/api/people/14/'],
 'films': ['https://swapi.co/api/films/2/', 'https://swapi.co/api/films/3/'],
 'created': '2014-12-15T13:04:47.235000Z',
 'edited': '2014-12-22T17:35:44.795405Z',
 'url': 'https://swapi.co/api/starships/22/'},
{'name': 'Trade Federation cruiser',
 'model': 'Providence-class carrier/destroyer',
 'manufacturer': 'Rendili StarDrive, Free Dac Volunteers Engineering corps.',
 'cost_in_credits': '125000000',
 'length': '1088',
 'max_atmosphering_speed': '1050',
 'crew': '600',
 'passengers': '48247',
 'cargo_capacity': '50000000',
 'consumables': '4 years',
 'hyperdrive_rating': '1.5',
 'MGLT': 'unknown',
 'starship_class': 'capital ship',
 'pilots': ['https://swapi.co/api/people/10/',
  'https://swapi.co/api/people/11/'],
 'films': ['https://swapi.co/api/films/6/'],
 'created': '2014-12-20T19:40:21.902000Z',
 'edited': '2014-12-22T17:35:45.195165Z',
 'url': 'https://swapi.co/api/starships/59/'},
{'name': 'Jedi Interceptor',
 'model': 'Eta-2 Actis-class light interceptor',
 'manufacturer': 'Kuat Systems Engineering',
 'cost_in_credits': '320000',
 'length': '5.47',
 'max_atmosphering_speed': '1500',
 'crew': '1',
 'passengers': '0',
 'cargo_capacity': '60',
 'consumables': '2 days',
 'hyperdrive_rating': '1.0',
 'MGLT': 'unknown',
```

```
 'starship_class': 'starfighter',
 'pilots': ['https://swapi.co/api/people/10/',
  'https://swapi.co/api/people/11/'],
 'films': ['https://swapi.co/api/films/6/'],
 'created': '2014-12-20T19:56:57.468000Z',
 'edited': '2014-12-22T17:35:45.272349Z',
 'url': 'https://swapi.co/api/starships/65/'},
{'name': 'Naboo fighter',
 'model': 'N-1 starfighter',
 'manufacturer': 'Theed Palace Space Vessel Engineering Corps',
 'cost_in_credits': '200000',
 'length': '11',
 'max_atmosphering_speed': '1100',
 'crew': '1',
 'passengers': '0',
 'cargo_capacity': '65',
 'consumables': '7 days',
 'hyperdrive_rating': '1.0',
 'MGLT': 'unknown',
 'starship_class': 'Starfighter',
 'pilots': ['https://swapi.co/api/people/11/',
  'https://swapi.co/api/people/60/',
  'https://swapi.co/api/people/35/'],
 'films': ['https://swapi.co/api/films/5/', 'https://swapi.co/api/films/4/'],
 'created': '2014-12-19T17:39:17.582000Z',
 'edited': '2014-12-22T17:35:45.079452Z',
 'url': 'https://swapi.co/api/starships/39/'}]
```

6 Exercises

Exercises 1-3 deal with the New York Philharmonic data set from above.

Exercise 1. Answer the Benny Goodman question above ("How many works has Benny Goodman performed with the New York Philharmonic?") by writing nested for loops that traverse the structure of the JSON object. Check that your answer agrees with the one we obtained above by first flattening the JSON object to a `DataFrame`.

```
In [19]: works.head()
```

```
Out[19]:        ID             composerName        conductorName      interval  \
         0  52446*  Beethoven, Ludwig van  Hill, Ureli Corelli           NaN
         1  8834*4  Weber, Carl Maria Von       Timm, Henry C.           NaN
         2   3642*          Hummel, Johann                  NaN           NaN
         3      0*                     NaN                  NaN  Intermission
         4  8834*3  Weber, Carl Maria Von    Etienne, Denis G.           NaN

                                                 movement  \
         0                                             NaN
         1  "Ozean, du Ungeheuer" (Ocean, thou mighty mons...
```

```
2                                                    NaN
3                                                    NaN
4                                               Overture

                                                soloists  \
0                                                     []
1  [{'soloistName': 'Otto, Antoinette', 'soloistI...
2  [{'soloistName': 'Scharfenberg, William', 'sol...
3                                                     []
4                                                     []

                          workTitle
0  SYMPHONY NO. 5 IN C MINOR, OP.67
1                           OBERON
2   QUINTET, PIANO, D MINOR, OP. 74
3                              NaN
4                           OBERON
```

```
In [20]: count = 0
         for program in nyphil["programs"]:
             for work in program["works"]:
                 for soloist in work["soloists"]:
                     if soloist["soloistName"] == "Goodman, Benny":
                         count += 1

         count
```

```
Out[20]: 25
```

Exercise 2. What is the most frequent start time for New York Philharmonic concerts?

```
In [21]: concerts = json_normalize(nyphil["programs"], "concerts")
         concerts.head()
         concerts.Time.value_counts()
```

```
Out[21]: 8:30PM    4584
         8:00PM    4443
         3:00PM    2133
         7:30PM    2075
         2:30PM    1618
                   ...
         1:45PM       1
         3:20PM       1
         1:15PM       1
         8:36PM       1
         2:00AM       1
         Name: Time, Length: 70, dtype: int64
```

Exercise 3. How many total concerts did the New York Philharmonic perform in the 2014-15 season?

```
In [22]: concerts["Year"] = concerts.Date.str[0:4]

In [23]: concerts["Year"].value_counts().loc["2014"]

Out[23]: 226

In [24]: concerts = json_normalize(nyphil["programs"], "concerts", meta="season")
         concerts[concerts["season"] == "2014-15"]

Out[24]:                         Date           Location    Time  \
         20962  2014-09-04T04:00:00Z    Shanghai, CHINA    None
         20963  2014-09-10T04:00:00Z    Shanghai, CHINA  8:00PM
         20964  2014-09-12T04:00:00Z    Shanghai, CHINA  7:45PM
         20965  2014-09-16T04:00:00Z      Manhattan, NY  7:30PM
         20966  2014-09-17T04:00:00Z      Manhattan, NY  7:30PM
         ...                      ...                ...     ...
         21174  2015-07-29T04:00:00Z           Vail, CO  6:00PM
         21175  2015-07-30T04:00:00Z           Vail, CO  6:00PM
         21176  2015-07-31T04:00:00Z           Vail, CO  6:00PM
         21177  2015-08-02T04:00:00Z  Santa Barbara, CA  7:00PM
         21178  2015-08-03T04:00:00Z  Santa Barbara, CA  7:00PM

                                              Venue                   eventType  \
         20962                                 None  Tour - Concert for Patrons
         20963              U.S. Consulate Shanghai         Residency - Chamber
         20964  Shanghai Symphony Hall--Chamber Hall                   Residency
         20965                   Avery Fisher Hall            Non-Subscription
         20966                   Avery Fisher Hall            Non-Subscription
         ...                                   ...                         ...
         21174          Gerald R. Ford Amphitheater                        Tour
         21175          Gerald R. Ford Amphitheater                        Tour
         21176          Gerald R. Ford Amphitheater                        Tour
         21177                   Santa Barbara Bowl           Reading Rehearsal
         21178                   Santa Barbara Bowl                        Tour

                season
         20962  2014-15
         20963  2014-15
         20964  2014-15
         20965  2014-15
         20966  2014-15
         ...        ...
         21174  2014-15
         21175  2014-15
         21176  2014-15
         21177  2014-15
         21178  2014-15

         [217 rows x 6 columns]
```

To answer Exercises 4-6, you will need to issue HTTP requests to the Open States API, which contains information about state legislatures. You will need to include an API key with every request. You can register for an API key here. Once you have an API key, enter your API key below. If your API key works, then the code below should produce a `DataFrame` of all of the committees in the California State Assembly (the lower chamber).

```
In [25]: # This is just a sample request to test that your API key is working.
         apikey = "a74a5fb1-5f5e-4816-8f56-fdf92738fc2c"
         resp = requests.get(
             "https://openstates.org/api/v1/committees/?state=ca&chamber=lower&apikey=%s" % api
         )
         pd.DataFrame(resp.json())
```

```
Out[25]: Empty DataFrame
         Columns: []
         Index: []
```

To answer the questions below, you will need to issue your own HTTP requests to the API. To understand how to construct URLs, you will need to refer to the documentation for this API.

Exercise 4. Legislators typically have offices in both the Capitol building and in their districts. Among the active legislators in the California Assembly (lower chamber), which legislators have the most offices (and how many do they have)?

```
In [26]: resp = requests.get(
             "https://openstates.org/api/v1/legislators/?state=ca&chamber=lower&apikey=%s" % ap

         legislators_json = resp.json()
         legislators_df = json_normalize(legislators_json, meta=["full_name"])
         legislators_df.head()
```

```
Out[26]:    active                all_ids chamber country           created_at  \
         0    True             [CAL000410]   lower      us  2018-10-18 14:35:11
         1    True  [CAL000508, CAL000534]   lower      us  2018-10-18 14:35:11
         2    True             [CAL000517]   lower      us  2018-10-18 14:35:11
         3    True             [CAL000458]   lower      us  2018-10-18 14:35:11
         4    True             [CAL000463]   lower      us  2018-10-18 14:35:12

           district                                    email  first_name  \
         0       75       assemblymember.waldron@assembly.ca.gov       Marie
         1       68          assemblymember.choi@assembly.ca.gov   Steven S.
         2        4  assemblymember.aguiar-curry@assembly.ca.gov  Cecilia M.
         3        3     assemblymember.gallagher@assembly.ca.gov       James
         4       26        assemblymember.mathis@assembly.ca.gov    Devon J.

                        full_name         id           ...            middle_name  \
         0            Marie Waldron  CAL000410           ...
         1    Steven S. Choi, Ph.D.  CAL000508           ...
         2  Cecilia M. Aguiar-Curry  CAL000517           ...
         3          James Gallagher  CAL000458           ...
```

```
4          Devon J. Mathis  CAL000463           ...

                                                offices       party  \
0  [{'name': 'Capitol Office', 'fax': None, 'phon...  Republican
1  [{'name': 'Capitol Office', 'fax': None, 'phon...  Republican
2  [{'name': 'Capitol Office', 'fax': None, 'phon...  Democratic
3  [{'name': 'Capitol Office', 'fax': None, 'phon...  Republican
4  [{'name': 'Capitol Office', 'fax': None, 'phon...  Republican

                                             photo_url  \
0  https://assembly.ca.gov/sites/assembly.ca.gov/...
1  https://assembly.ca.gov/sites/assembly.ca.gov/...
2  https://assembly.ca.gov/sites/assembly.ca.gov/...
3  https://assembly.ca.gov/sites/assembly.ca.gov/...
4  https://assembly.ca.gov/sites/assembly.ca.gov/...

                                               roles  \
0  [{'term': '20172018', 'district': '75', 'chamb...
1  [{'term': '20172018', 'district': '68', 'chamb...
2  [{'term': '20172018', 'district': '4', 'chambe...
3  [{'term': '20172018', 'district': '3', 'chambe...
4  [{'term': '20172018', 'district': '26', 'chamb...

                                             sources state suffix  \
0  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
1  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
2  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
3  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
4  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca

            updated_at                      url
0  2018-12-11 01:42:38  https://ad75.asmrc.org/
1  2018-12-11 01:43:08  https://ad68.asmrc.org/
2  2018-12-11 01:42:56   https://a04.asmdc.org/
3  2018-12-11 01:42:17   http://ad03.asmrc.org/
4  2018-12-11 01:42:02  https://ad26.asmrc.org/

[5 rows x 23 columns]
```

In [27]:
```
def office_count(offices):
    return len(offices)

indeces = list(legislators_df["offices"].apply(len).sort_values(ascending=False).inde
legislators_df.loc[indeces]
```

Out[27]:
```
    active                all_ids chamber country           created_at  \
35    True             [CAL000443]   lower      us  2018-10-18 14:35:22
2     True             [CAL000517]   lower      us  2018-10-18 14:35:11
```

```
64    True             [CAL000381]   lower      us  2018-10-18 14:35:33
36    True  [CAL000341, CAL000429]   lower      us  2018-10-18 14:35:23
30    True             [CAL000461]   lower      us  2018-10-18 14:35:21
..     ...                     ...     ...     ...                  ...
52    True             [CAL000520]   lower      us  2018-10-18 14:35:28
53    True             [CAL000366]   lower      us  2018-10-18 14:35:28
54    True             [CAL000503]   lower      us  2018-10-18 14:35:29
55    True             [CAL000527]   lower      us  2018-10-18 14:35:29
0     True             [CAL000410]   lower      us  2018-10-18 14:35:11

   district                                    email  first_name  \
35        2          assemblymember.wood@assembly.ca.gov         Jim
2         4  assemblymember.aguiar-curry@assembly.ca.gov  Cecilia M.
64       10        assemblymember.levine@assembly.ca.gov        Marc
36        5       assemblymember.bigelow@assembly.ca.gov    Franklin
30       56        assemblymember.garcia@assembly.ca.gov     Eduardo
..      ...                                          ...         ...
52        6         assemblymember.kiley@assembly.ca.gov       Kevin
53       58        assemblymember.garcia@assembly.ca.gov    Cristina
54       34         assemblymember.vince@assembly.ca.gov       Vince
55       78        assemblymember.gloria@assembly.ca.gov        Todd
0        75       assemblymember.waldron@assembly.ca.gov       Marie

                full_name         id           ...            middle_name  \
35                Jim Wood  CAL000443           ...
2   Cecilia M. Aguiar-Curry  CAL000517           ...
64              Marc Levine  CAL000381           ...
36            Frank Bigelow  CAL000341           ...
30           Eduardo Garcia  CAL000461           ...
..                      ...        ...           ...                    ...
52              Kevin Kiley  CAL000520           ...
53          Cristina Garcia  CAL000366           ...
54               Vince Fong  CAL000503           ...
55              Todd Gloria  CAL000527           ...
0             Marie Waldron  CAL000410           ...

                                                 offices       party  \
35  [{'name': 'Capitol Office', 'fax': None, 'phon...  Democratic
2   [{'name': 'Capitol Office', 'fax': None, 'phon...  Democratic
64  [{'name': 'Capitol Office', 'fax': None, 'phon...  Democratic
36  [{'name': 'Capitol Office', 'fax': None, 'phon...  Republican
30  [{'name': 'Capitol Office', 'fax': None, 'phon...  Democratic
..                                                ...         ...
52  [{'name': 'Capitol Office', 'fax': None, 'phon...  Republican
53  [{'name': 'Capitol Office', 'fax': None, 'phon...  Democratic
54  [{'name': 'Capitol Office', 'fax': None, 'phon...  Republican
55  [{'name': 'Capitol Office', 'fax': None, 'phon...  Democratic
0   [{'name': 'Capitol Office', 'fax': None, 'phon...  Republican
```

```
                                                photo_url  \
35  https://assembly.ca.gov/sites/assembly.ca.gov/...
2   https://assembly.ca.gov/sites/assembly.ca.gov/...
64  https://assembly.ca.gov/sites/assembly.ca.gov/...
36  https://assembly.ca.gov/sites/assembly.ca.gov/...
30  https://assembly.ca.gov/sites/assembly.ca.gov/...
..                                                ...
52  https://assembly.ca.gov/sites/assembly.ca.gov/...
53  https://assembly.ca.gov/sites/assembly.ca.gov/...
54  https://assembly.ca.gov/sites/assembly.ca.gov/...
55  https://assembly.ca.gov/sites/assembly.ca.gov/...
0   https://assembly.ca.gov/sites/assembly.ca.gov/...

                                                  roles  \
35  [{'term': '20172018', 'district': '2', 'chambe...
2   [{'term': '20172018', 'district': '4', 'chambe...
64  [{'term': '20172018', 'district': '10', 'chamb...
36  [{'term': '20172018', 'district': '5', 'chambe...
30  [{'term': '20172018', 'district': '56', 'chamb...
..                                                ...
52  [{'term': '20172018', 'district': '6', 'chambe...
53  [{'term': '20172018', 'district': '58', 'chamb...
54  [{'term': '20172018', 'district': '34', 'chamb...
55  [{'term': '20172018', 'district': '78', 'chamb...
0   [{'term': '20172018', 'district': '75', 'chamb...

                                               sources state suffix  \
35  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
2   [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
64  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
36  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
30  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
..                                                ...   ...    ...
52  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
53  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
54  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
55  [{'url': 'http://assembly.ca.gov/assemblymembe...    ca
0   [{'url': 'http://assembly.ca.gov/assemblymembe...    ca

            updated_at                      url
35  2019-01-02 10:16:40    https://a02.asmdc.org
2   2018-12-11 01:42:56   https://a04.asmdc.org/
64  2018-12-11 01:42:55    https://a10.asmdc.org
36  2019-01-02 10:16:41   https://ad05.asmrc.org
30  2018-12-11 01:41:51   https://a56.asmdc.org/
..                  ...                      ...
52  2018-12-11 01:43:17  https://ad06.asmrc.org/
```

```
53  2018-12-11 01:42:09   https://a58.asmdc.org/
54  2018-12-11 01:42:31  https://ad34.asmrc.org/
55  2018-12-11 01:42:39   https://a78.asmdc.org/
0   2018-12-11 01:42:38  https://ad75.asmrc.org/

[80 rows x 23 columns]
```

```
In [28]: offices_df = json_normalize(legislators_json, "offices", meta="full_name")
         offices_df.groupby("full_name")["full_name"].count().sort_values()
```

```
Out[28]: full_name
         Kansen Chu                 2
         Philip Y.  Ting           2
         Miguel Santiago            2
         Melissa A. Melendez        2
         Marie Waldron              2
                                   ..
         Rudy Salas, Jr.            3
         Cecilia M. Aguiar-Curry    4
         Marc Levine                4
         Frank Bigelow              4
         Jim Wood                   4
         Name: full_name, Length: 80, dtype: int64
```

Exercise 5. Get all of the *constitutional amendments* in the California State Senate (upper house) from the current legislative session. How many amendments have there been?

(*Hint:* "Constitutional amendment" is a type of bill.)

```
In [38]: resp = requests.get(
             "https://openstates.org/api/v1/bills/?state=ca&chamber=upper&type=amendment:failed
```

```
In [39]: df_json = resp.json()
```

```
In [40]: df_json
```

```
Out[40]: 'Bad Request: request too large, try narrowing your search by adding more filters or u
```

Exercise 6. Look up the votes on the constitutional amendments you found in Exercise 5. Calculate the number of "yes" and "no" votes for each legislator on these amendments. Which legislator had the most total votes on constitutional amendments in the current session? Which legislator had the most total negative votes?

```
In [ ]: # ENTER YOUR CODE HERE.
```