# Using APIs with Python

*Margaret Gratian*


8/17/2023

# Agenda

- Understanding APIs at a High Level
- Understanding HTTP at a High Level
- The Requests Library in Python
- The JSON Data Format
- The RePORTER API

# Agenda

- **Understanding APIs at a High Level**
- Understanding HTTP at a High Level
- The Requests Library in Python
- The JSON Data Format
- The RePORTER API

# What is an API?

- API == Application Programming Interface
- A method for enabling two software programs to communicate
- The interface helps abstract away the details of how the software works and provides standard rules for how the two programs will communicate
- Common analogy is that of a contract between two programs – 'I will send you a request in our agreed format, and you will respond in our agreed format'

For more:

- https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces
- https://aws.amazon.com/what-is/api/#:~:text=API%20stands%20for%20Application%20Programming,other%20using%20requests%20and%20responses.
- https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/

# What is an API? (continued)

- Typical API use case is enabling one software program to *request* data, and the other software program to *respond* with the data
- Companies, governments, and other data owners/providers typically use APIs to make their data accessible to third-parties

## Intro


### Purpose

NIH RePORTER APIs are designed to programmatically expose relevant scientific awards data from both NIH and non-NIH federal agencies for the consumption of project teams or external 3rd party applications to support reporting, data analysis, data integration or to satisfy other business needs as deemed pertinent.

# An API in Practice

- You check the weather forecast for your current location on your phone using a weather app and get back the weather

*Behind the scenes:*

- The weather app sends a structured *request* (a dictionary-like structure of keys and their values) to the The National Weather Service's API
- This structured request contains information about your current location and indicates the particular API *endpoint* (a URL)
- This endpoint (URL) is the location on the *server* (NWS' weather database) where the forecast information can be found
- When the request is sent to the forecast endpoint, it provides a *response* (a dictionary-like structure of fields and their values) back to the *client* (the weather app) with the requested weather forecast information

# How do I use an API?

- The Python Requests library provides methods to send requests and interact with an API
  - https://requests.readthedocs.io/en/latest/
- There may also be specific libraries available for a particular API

# Some Things to Keep in Mind

- Some APIs will require you to set up an API key or username and password to authenticate your use
- API documentation will often specify a limit to your request rate
  - RePORTER asks for no more than one request per second
- API documentation may also specify times they prefer you to run certain types of queries
  - The RePORTER API asks that large queries be run after 5 pm and before 9 am on weekdays or on weekends

# Agenda

- Understanding APIs at a High Level
- **Understanding HTTP at a High Level**
- The Requests Library in Python
- The JSON Data Format
- The RePORTER API

# What is HTTP?

- HTTP – Hypertext Transfer Protocol:
  - How most information/data is transferred on the Internet
  - How a client makes a request to a server, and the server sends the request back (i.e., how APIs work)
- HTTP messages are typically either *requests* or *responses*

- Note that HTTPS is secure HTTP– requests and responses are encrypted

For more:

- https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/
- https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/

# HTTP **Request** Structure

- The previous slides have mentioned that data is sent as a structured request (both when using an API or just in the course of communications on the Internet)
- What is in a typical request? Some key pieces:
  - URL (e.g., an API endpoint)
  - HTTP method
  - HTTP header
  - HTTP body (optional)

# HTTP Methods: GET and POST

- Two common HTTP methods (also called “verbs”)
- GET – request some information/data from a server
  - Example: from your browser you click a link and a get request is issued, returning the webpage to you
- POST – request information/data from a server and/or submit information to a server
  - Example: to get information from the RePORTER API, you issue a POST request with information about the award data you are interested in, and the server processes the public facing award data into the data subset matching your search criteria

# HTTP Header

- Headers are part of every HTTP request and contain key information to make the request possible
- Information is stored in a dictionary-like structure (keys and their values)

Example of HTTP request headers from Google Chrome's network tab:

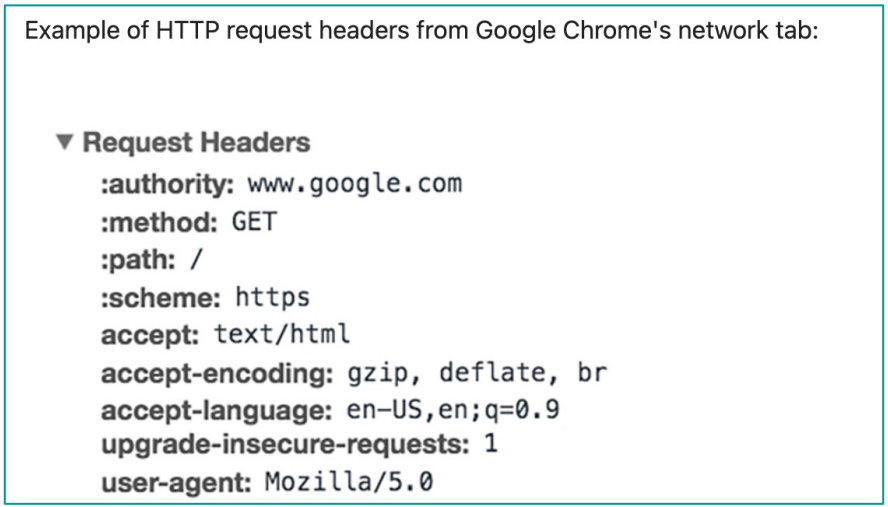



Image source: https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/



For more: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

# HTTP Body

- The body, also called a payload, is often optional and contains information being submitted to a server
- When requesting data from the RePORTER API, the HTTP body is required, as it is where you include information about what award data you want, such as the activity codes, fiscal years, administering agencies, etc.

NIH RePORTER

**Org Name Search:**
The Following criteria payload filters the projects by given Organization Name list and returns 25 records sorted by Project Start Date in descending order.

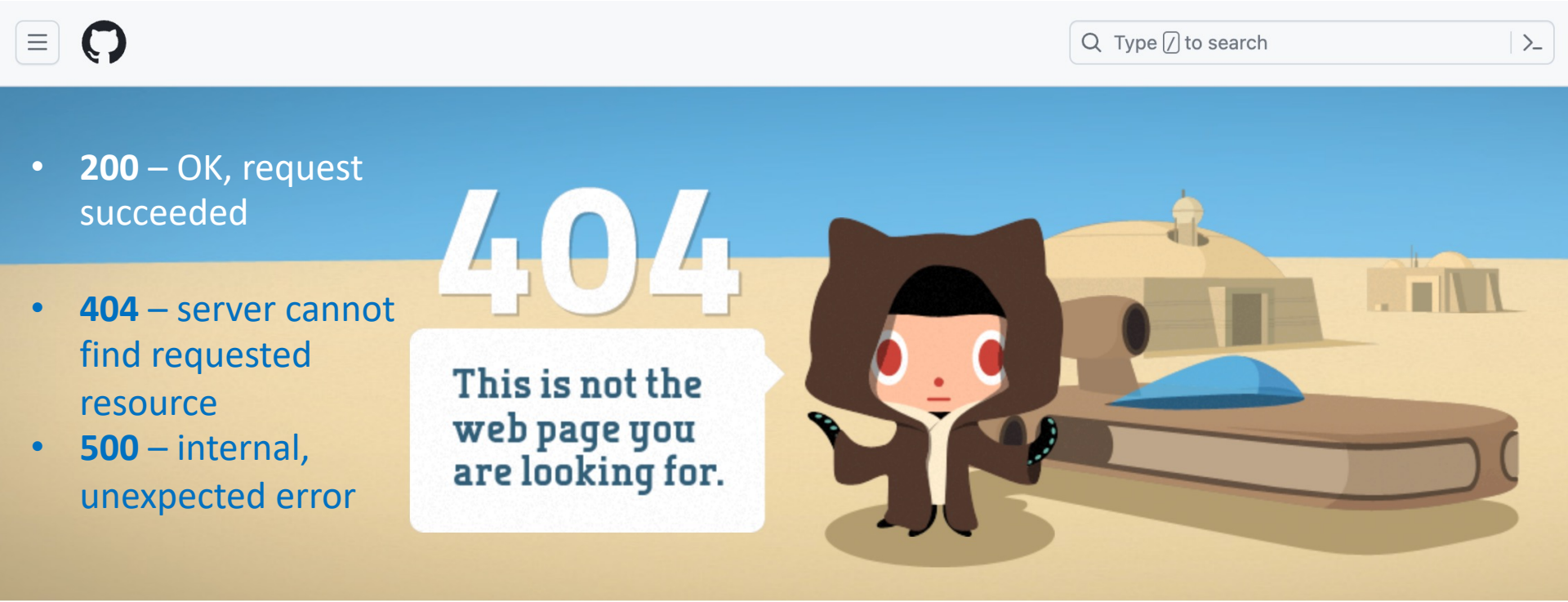
```
{
      "criteria":
      {
          "org_names": ["HARVARD MEDICAL SCHOOL"]
      },
      "include_fields": [
           "ApplId","SubprojectId","FiscalYear","Organization", "ProjectNum",
           "ProjectNumSplit","ContactPiName","AllText","FullStudySection",
           "ProjectStartDate","ProjectEndDate"
       ],
       "offset":0,
       "limit":25,
       "sort_field":"project_start_date",
```

# HTTP **Response** Structure

- Information/data requested from a server is returned as an HTTP response
- A response includes:
  - A status code (numerical codes that tell you if an HTTP request worked)
  - HTTP response header (similar to a request header, always required, contains key information)
  - HTTP body (similar to a request body, optional, may contain the data sent in response)

# Response Status Codes

- **200** – OK, request succeeded
- **404** – server cannot find requested resource
- **500** – internal, unexpected error




For the 5 major categories of status codes and more info: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

# Response Body – RePORTER Documentation Example

NIH RePORTER

## API Endpoints

For the endpoints supported by the NIH RePORTER APIs, the interface below lists the payload criteria that can be used to retrieve the relevant data in JSON format. You can enter a valid value for one or more criteria parameter and click the "Try it Out" button. System will return:

- Curl script for executing the API using a command line interface (CLI).
- The Response Body showing the actual data.
- Request URL, Request Headers, and Response Headers with relevant information.

The Model (Example Value) section shows the output fields along with their data types in a format based on the Response Content Type selection.

Select a version V2.0

## Search

POST /v2/projects/search Retrieves Projects against specified criteria payload.

### Implementation Notes

The input is a criteria payload in JSON format. Only required filter criteria fields can be provided to get the desired project records.

**Response includes:**

- Total Count of matches
- Offset
- Limit
- Total number of pages (based on the Limit per request)
- One or more matching projects.

Source: https://api.reporter.nih.gov/

# Agenda

- Understanding APIs at a High Level
- Understanding HTTP at a High Level
- **The Requests Library in Python**
- The JSON Data Format
- The RePORTER API

# The Requests Library

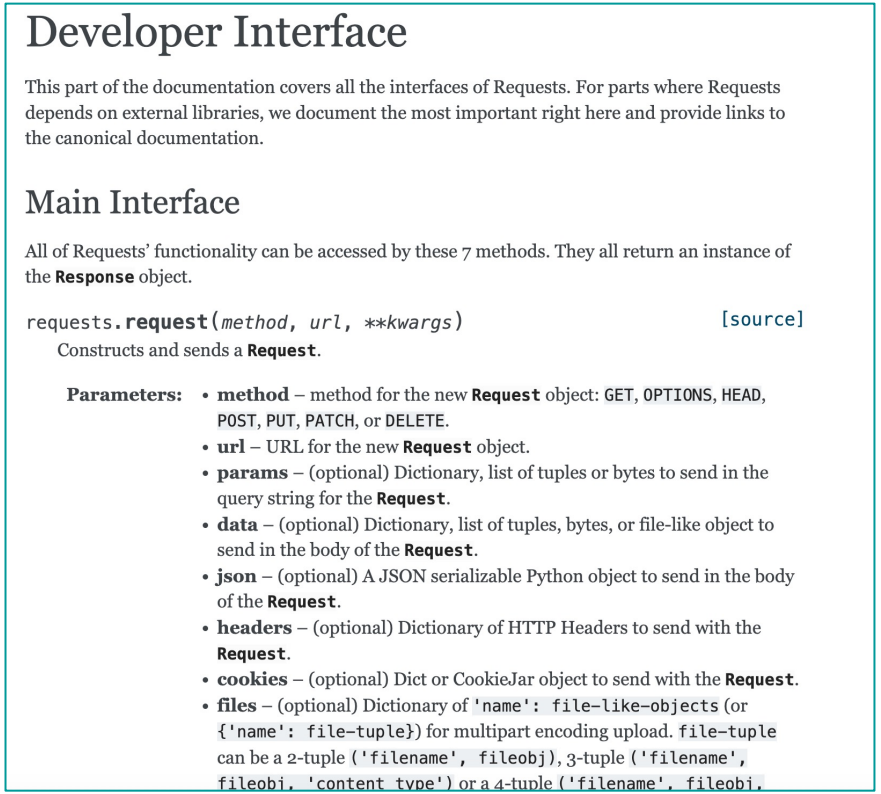
- Requests is a Python library that simplifies the process of using HTTP
- Requests implements HTTP methods such as GET and POST that are essential for requesting and receiving information from an API
- Must be installed and then imported (it is not one of the default Python libraries)
  - See installation document accompanying this tutorial for instructions on how to do so in Anaconda


For more:

- https://pypi.org/project/requests/
- https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/

# Key Components of the Requests Library

- The Requests library is composed of 7 main methods, such as ‘request’, ‘get’, and ‘post’
- These methods enable you to pass in required information for the request as parameters
- All of these methods return a *Response object*, with available methods to extract the information you requested and additional metadata



## Developer Interface

This part of the documentation covers all the interfaces of Requests. For parts where Requests depends on external libraries, we document the most important right here and provide links to the canonical documentation.

### Main Interface

All of Requests’ functionality can be accessed by these 7 methods. They all return an instance of the **Response** object.

`requests.request(method, url, **kwargs)` [source]

Constructs and sends a **Request**.

**Parameters:**

- **method** – method for the new **Request** object: `GET`, `OPTIONS`, `HEAD`, `POST`, `PUT`, `PATCH`, or `DELETE`.
- **url** – URL for the new **Request** object.
- **params** – (optional) Dictionary, list of tuples or bytes to send in the query string for the **Request**.
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
- **json** – (optional) A JSON serializable Python object to send in the body of the **Request**.
- **headers** – (optional) Dictionary of HTTP Headers to send with the **Request**.
- **cookies** – (optional) Dict or CookieJar object to send with the **Request**.
- **files** – (optional) Dictionary of `'name': file-like-objects` (or `{'name': file-tuple}`) for multipart encoding upload. `file-tuple` can be a 2-tuple `('filename', fileobj)`, 3-tuple `('filename', fileobj, 'content_type')` or a 4-tuple `('filename', fileobj,`

Image source: https://requests.readthedocs.io/en/latest/api/

# GET and POST using Python Requests

## Make a Request

Making a request with Requests is very simple.

Begin by importing the Requests module:

```
>>> import requests
```

Now, let's try to get a webpage. For this example, let's get GitHub's public timeline:

```
>>> r = requests.get('https://api.github.com/events')
```

Now, we have a **`Response`** object called `r`. We can get all the information we need from this object.

Requests' simple API means that all forms of HTTP request are as obvious. For example, this is how you make an HTTP POST request:

```
>>> r = requests.post('https://httpbin.org/post', data={'key': 'value'})
```

Source: https://requests.readthedocs.io/en/latest/user/quickstart/

# Agenda

- Understanding APIs at a High Level
- Understanding HTTP at a High Level
- The Requests Library in Python
- **The JSON Data Format**
- The RePORTER API

# JSON

- JavaScript Object Notation, a “lightweight data-interchange format”[1]
- JSON is very similar to a dictionary structure, where there are keys and values
- JSON can be very nested (dictionaries within dictionaries) which can sometimes make it tricky to work with
- API results are often returned in a JSON structure
- JSON objects can sometimes be tricky to flatten if they are very nested e.g.,

  {level 1: {level 2: {level 3: {}}}

[1]https://www.json.org/json-en.html



For more: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

# Python Resources for Working with JSON

- Requests has a .json() method that enables you to see the raw JSON associated with a request (if it returned data in JSON format)
  - https://requests.readthedocs.io/en/latest/user/quickstart/#json-response-content
- Pandas has several different functions for working with JSON (converting to and from DataFrames)
  - https://pandas.pydata.org/docs/reference/api/pandas.json_normalize.html
  - https://pandas.pydata.org/docs/reference/api/pandas.read_json.html
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_json.html
- Python's standard library also comes with a json library
  - https://docs.python.org/3/library/json.html

# JSON Example

```
# We data is contained in response?
# We can use the .json() method from requests to see the raw search result
response.json()
```


```
{'meta': {'search_id': 'SZsY8Em8lkm7G-SPI28Eiw',
  'total': 12935,
  'offset': 0,
  'limit': 25,
  'sort_field': 'project_start_date',
  'sort_order': 'desc',
  'sorted_by_relevance': False,
  'properties': {'URL': 'https:/reporter.nih.gov/search/SZsY8Em8lkm7G-SPI28Eiw/projects'}},
 'results': [{'appl_id': 10795597,
   'subproject_id': None,
   'fiscal_year': 2022,
   'org_name': 'BROWN UNIVERSITY',
   'org_city': 'PROVIDENCE',
```

# Agenda

- Understanding APIs at a High Level
- Understanding HTTP at a High Level
- The Requests Library in Python
- The JSON Data Format
- **The RePORTER API**

# NIH RePORTER API

- Provides a programmatic way to request data from NIH (and some non-NIH federal agencies)
- Two main endpoints:
  - To search projects: https://api.reporter.nih.gov/v2/projects/search
  - To search publications: https://api.reporter.nih.gov/v2/publications/search




Source: https://api.reporter.nih.gov/

# Benefits of Using the API

- The API can help simplify accessing large amounts of data
- The API can also make it easier others to replicate your query (share the payload vs. sharing fields to fill out on the web form)

# Constraints and Limitations when Using the API

## Constraints/Limitations

- NIH RePORTER APIs do not support fuzzy, proximity, or range based searches except where specified.
- The API service will be available except during a pre-scheduled maintenance window. Appropriate maintenance messages will be posted in advance on the API site.
- To provide quality service for all users of NIH RePORTER web and API servers, it is recommended that users post no more than one URL request per second and limit large jobs to either weekends or weekdays between 9:00 PM and 5:00 AM EST.
- Failure to comply with this policy may require administrators to block your IP address from accessing the API service. For any questions regarding use of the NIH RePORTER API, please reach out to the support team at RePORT@mail.nih.gov.

| | |
|---|---|
| offset | Users can set the starting counter for the items (projects). By default, it starts at 0. Offset doesn't support negative number.Maximum allowed value is 14,999. In addition, the requested Offset value shall not exceed total records count.<br><br>*Example:* Out of a total of 100 items, if you want to retrieve items starting from #60, you must set the offset value to 59. |

Note the constraint of requests with more than 14,999 expected records. You will need to split up your request to handle this.



Source: https://api.reporter.nih.gov/

# Other Cautions When Using RePORTER

- NIH data is dynamic!
- The RePORTER documentation cautions that data is updated weekly and can differ between weeks.

# Requesting Data from RePORTER using Requests

**Step 1**: Import required packages

```
1 import requests # library for working with APIs
2 import json # library in Python's Standard Library for handling json
3 import pandas as pd
```

# Requesting Data from RePORTER using Requests

**Step 2**: Define the Request fields

```python
# Set the endpoint url of the API
# We are using the endpoint associated with projects
url = "https://api.reporter.nih.gov/v2/projects/search"

# Set the headers based on the RePORTER documentation
# Use these headers each time
headers = {'accept': 'application/json', 'Content-Type': 'application/json'}
```

```python
# Set up the data payload - this is what we're going to tell RePORTER to get for us
# We specify details about the records to return after a field called "criteria"
# We'll also use a few additional fields to limit our results to 25 and how we want them sorted
# We use field names provided in the data elements documentation: https://api.reporter.nih.gov/documents/Data%20Ele

data = {
    "criteria": {
        "fiscal_years": [2022],
        "agencies": ["NCI"],
    },
    "limit": 25,
    "sort_field":"project_start_date",
    "sort_order":"desc"
}
```

# Requesting Data from RePORTER using Requests

**Step 3**: Send the request using requests.post()

```
# Send the request to the API, using the requests library's POST method
# We pass the url, headers,and data as parameters
# Note that we use a function from the json library called .dumps() to convert our data to valid json
# Assign the response to a variable called response
response = requests.post(url, data=json.dumps(data), headers=headers)
```

# Requesting Data from RePORTER using Requests

**Step 4**: Check the status code of the response. Was it 200 (OK)?

```
1 # First, we can check our status code
2 # 200 indicates the request was error free
3 response.status_code
```

```
200
```

# Requesting Data from RePORTER using Requests

**Step 5**: Take a look at the raw JSON using Requests .json() function. Note that two top-level fields are ‘meta’ (which tells us helpful information about such as how many records there are) and ‘results’ (which contains our actual data)

```
# We data is contained in response?
# We can use the .json() method from requests to see the raw search result
response.json()
```


```
{'meta': {'search_id': 'SZsY8Em8lkm7G-SPI28Eiw',
  'total': 12935,
  'offset': 0,
  'limit': 25,
  'sort_field': 'project_start_date',
  'sort_order': 'desc',
  'sorted_by_relevance': False,
  'properties': {'URL': 'https:/reporter.nih.gov/search/SZsY8Em8lkm7G-SPI28Eiw/projects'}},
 'results': [{'appl_id': 10795597,
   'subproject_id': None,
   'fiscal_year': 2022,
   'org_name': 'BROWN UNIVERSITY',
   'org_city': 'PROVIDENCE',
```

# Requesting Data from RePORTER using Requests

**Step 6**: Take the 'results' portion of the response and save as a variable

```
# Assign the results portion of the data to a variable called results
results = response.json()['results']

# Preview - now we just have the data
results
```


```
[{'appl_id': 10795597,
  'subproject_id': None,
  'fiscal_year': 2022,
  'org_name': 'BROWN UNIVERSITY',
  'org_city': 'PROVIDENCE',
  'org_state': 'RI',
  'org_state_name': None,
  'dept_type': 'PATHOLOGY',
  'project_num': '7R01CA262106-02',
  'project_serial_num': 'CA262106',
  'org_country': 'UNITED STATES',
  'organization': {'org_name': 'BROWN UNIVERSITY',
   'city': None,
   'country': None,
   'org_city': 'PROVIDENCE',
   'org_country': 'UNITED STATES',
   'org_state': 'RI',
   'org_state_name': None,
```

# Requesting Data from RePORTER using Requests

**Step 7**: Use Panda's .json_normalize() function to flatten the results into a DataFrame

```
# Use the Pandas function .json_normalize on the results data
results_df = pd.json_normalize(results)

# See the shape (should match our limit of 25)
print(results_df.shape)

# Preview the data
results_df.head()
```

```
(25, 88)
```

| | appl_id | subproject_id | fiscal_year | org_name | org_city | org_state | org_state_name | dept_type | project_num | project_serial_num | ... | agenc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 10795597 | None | 2022 | BROWN UNIVERSITY | PROVIDENCE | RI | None | PATHOLOGY | 7R01CA262106-02 | CA262106 | ... | Natio |
| 1 | 10537799 | None | 2022 | SALK INSTITUTE FOR BIOLOGICAL STUDIES | La Jolla | CA | None | None | 1F32CA275206-01 | CA275206 | ... | Natio |
| 2 | 10731166 | None | 2022 | LOMA LINDA UNIVERSITY | LOMA LINDA | CA | None | OTHER BASIC SCIENCES | 7R21CA259911-02 | CA259911 | ... | Natio |

# Large Queries

- Recall that earlier slides noted both a limit on the number of records per request (“limit”:500)
- You will need to use “limit” field, “offset” field, and the “total_records” field contained in the metadata (“meta”) to handle a large query
- Depending on the size of the data you are requesting, the process may also involve splitting up your query criteria
  - For example, to query 20 years of data, you may split your query into 4 different queries that are incrementing sets of 5 years

*Questions?*



NATIONAL CANCER INSTITUTE

www.cancer.gov

www.cancer.gov/espanol