

▼ Everything is Better with Friends

Using SAS in Python Applications with SASPy and Open-Source Tooling (Beyond the Basics)

▼ Setup for Part 2

Getting setup to use Google Colab

1. To execute code cells, you'll need credentials for the following:
 - Google. (If you're not already signed in, you should see a **Sign In** button in the upper right corner. You can also visit <https://accounts.google.com/signup> to create an account for free.)
2. We recommend enabling line numbers using the Tools menu: **Tools -> Settings -> Editor -> Show line numbers -> Save**
3. We also recommend enabling the Table of Contents using the View menu: **View -> Table of contents**
4. To save a copy of this notebook, along with any edits you make, please use the File menu: **File -> Save a copy in Drive**
5. Some useful Zoom Reactions:
 - 👍 (Thumbs Up) when you're done with a section
 - 🙋 (Raise Hand) when you need tech support
 - 🍵 (I'm Away) to let us know you've stepped away
6. Looking for "extra credit"? Please let us know if you spot any typos!

▼ Install and import packages

```
1 # Install the rich module for colorful printing, limiting the version for compatibility with Colab
2 !pip install 'rich<13.3'
3
4 # We'll use IPython to display DataFrames or HTML content
5 from IPython.display import display, HTML
6
7 # We'll use the pandas package to create and manipulate DataFrame objects
8 import pandas
9
10 # We'll use the requests package to call a web API
11 import requests
12
13 # We're overwriting the default print function with rich.print
14 from rich import print
15
16 # We're also setting the maximum line width of rich.print to be a bit wider (to avoid line wrapping)
17 from rich import get_console
18 console = get_console()
19 console.width = 165
```

```

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting rich<13.3
  Downloading rich-13.2.0-py3-none-any.whl (238 kB)
    238.9/238.9 KB 3.0 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions<5.0,>=4.0.0 in /usr/local/lib/python3.8/dist-packages (from rich<13.3) (4.5.0)
Requirement already satisfied: pygments<3.0.0,>=2.6.0 in /usr/local/lib/python3.8/dist-packages (from rich<13.3) (2.6.1)
Collecting markdown-it-py<3.0.0,>=2.1.0
  Downloading markdown_it_py-2.2.0-py3-none-any.whl (84 kB)
    84.5/84.5 KB 4.7 MB/s eta 0:00:00
Collecting mdurl~=0.1
  Downloading mdurl-0.1.2-py3-none-any.whl (10.0 kB)
Installing collected packages: mdurl, markdown-it-py, rich
Successfully installed markdown-it-py-2.2.0 mdurl-0.1.2 rich-13.2.0

```

▼ Part 2. Rectangularizing unstructured data in Python applications

▼ Section 2.1. Create pharm_class_response

```

1 # Let's explore one of the many endpoints of the openFDA API!
2
3 # This might not be the most exciting ways to get data, but it's one of the few public API endpoints
4 # that doesn't require an access token, and it'll give us a chance to practice working with JSON.
5
6 # Use an open web API to get the number of drugs available by pharmacologic class.
7 # Note: By default, only the first 100 results are provided, sorted in descending order by count.
8 # To retrieve more than the first 100 results, a combination of limit and skip parameters can be
9 # used, as described at https://open.fda.gov/apis/paging/
10 pharm_class_response = requests.get('https://api.fda.gov/drug/ndc.json?count=pharm_class.exact')
11
12 # Check the resulting status code to make sure the API call was successful, with 200 = "OK".
13 http_status = pharm_class_response.status_code
14 http_status_info = f'https://httpstatus.com/{http_status}'
15 if http_status == 200:
16     print('API call successful!\n')
17 print(f'See {http_status_info} for more information about HTTP status code {http_status}.')

```

API call successful!

See <https://httpstatus.com/200> for more information about HTTP status code 200.

Concept Check 2.1

1. True or False: Changing Line 15 to a single-equals (=) would have the same effect.
 2. True or False: Removing the indentation on Line 16 would have the same effect.
- Fun Fact: The FDA provides many open APIs. Examples queries can be found at <https://open.fda.gov/apis/drug/ndc/example-api-queries/>

Solution: False! Single-equals (=) is only used for variable assignment, and double-equals (==) is only used to test for equality.

▼ Section 2.2. Explore pharm_class_response

```
1 # What exactly makes up an HTTP response?
2 print(dir(pharm_class_response))
3 print('\n')
4
5 # Let's take a look at the HTTP headers used behind the scenes
6 print(dict(pharm_class_response.headers))
```

```
[ ]
```



Concept Check 2.2

- Try this, and see what happens: Find the type of `dir(pharm_class_response)` by using the `type` function, e.g., using `print(type(dir(pharm_class_response)))`
- Try this, and see what happens: Find the type of `dict(pharm_class_response.headers)` by using the `type` function, e.g., using `print(type(dict(pharm_class_response.headers)))`
- Fun Facts:
 - Lists are a fundamental Python data structure and are similar to SAS DATA step arrays.
 - Dictionaries are another fundamental Python data structure and are related to SAS formats and DATA step hash tables.
 - In the `requests` module, the `headers` attribute is a special dictionary with type `<class 'requests.structures.CaseInsensitiveDict'>`, so the `dict` function is used to convert it to a regular dictionary.
 - The `dir` and `type` functions are examples of Python's introspection features, which allow us to ask an object to tell us about itself. Other introspection functions include `help` (to get an object's built-in documentation) and `id` (to get an object's behind-the-scenes unique id).

```
from reduce import *
```

```
1 print(type(dir(pharm_class_response)))
2
3 print(type(dict(pharm_class_response.headers)))
```

```
<class 'list'>
<class 'dict'>
```

```
_, content, _ = ...
```

▼ Section 2.3. Create `pharm_class_json`

```
'connection'
```

```
1 # Extract and print the JSON-formatted list of counts of drugs by pharmacologic class.
2 pharm_class_json = pharm_class_response.json()
3 print(pharm_class_json)
```

```
{
  'meta': {
    'disclaimer': 'Do not rely on openFDA to make decisions regarding medical care. While we make every effort to ensure that all results are unvalidated. We may limit or otherwise restrict your access to the API in line with our Terms of Service.',
    'terms': 'https://open.fda.gov/terms/',
    'license': 'https://open.fda.gov/license/',
    'last_updated': '2023-02-27'
  },
  'results': [
    {'term': 'Cell-mediated Immunity [PE]', 'count': 6032},
    {'term': 'Increased Histamine Release [PE]', 'count': 6018},
    {'term': 'Allergens [CS]', 'count': 6011},
    {'term': 'Increased IgG Production [PE]', 'count': 3840},
    {'term': 'Anti-Inflammatory Agents [MoA]', 'count': 3721},
    {'term': 'Cyclooxygenase Inhibitors [MoA]', 'count': 3721},
    {'term': 'Non-Steroidal [CS]', 'count': 3721},
    {'term': 'Nonsteroidal Anti-inflammatory Drug [EPC]', 'count': 3721},
    {'term': 'Histamine H1 Receptor Antagonists [MoA]', 'count': 2397},
    {'term': 'Corticosteroid Hormone Receptor Agonists [MoA]', 'count': 2383},
    {'term': 'Corticosteroid [EPC]', 'count': 2383},
    {'term': 'Histamine-1 Receptor Antagonist [EPC]', 'count': 2314},
    {'term': 'Osmotic Activity [MoA]', 'count': 2221},
    {'term': 'Increased Large Intestinal Motility [PE]', 'count': 2185},
    {'term': 'Osmotic Laxative [EPC]', 'count': 2143},
    {'term': 'Sigma-1 Agonist [EPC]', 'count': 2063},
    {'term': 'Sigma-1 Receptor Agonists [MoA]', 'count': 2063},
    {'term': 'Uncompetitive N-methyl-D-aspartate Receptor Antagonist [EPC]', 'count': 2063},
    {'term': 'Uncompetitive NMDA Receptor Antagonists [MoA]', 'count': 2063},
    {'term': 'Inhibition Large Intestine Fluid/Electrolyte Absorption [PE]', 'count': 2040},
    {'term': 'Decreased Central Nervous System Disorganized Electrical Activity [PE]', 'count': 1958},
    {'term': 'Adrenergic alpha1-Agonists [MoA]', 'count': 1879},
    {'term': 'alpha-1 Adrenergic Agonist [EPC]', 'count': 1879},
    {'term': 'Opioid Agonist [EPC]', 'count': 1649},
    {'term': 'Dietary Proteins [CS]', 'count': 1607},
    {'term': 'Non-Standardized Food Allergenic Extract [EPC]', 'count': 1602},
    {'term': 'Plant Proteins [CS]', 'count': 1598},
    {'term': 'Pollen [CS]', 'count': 1583},
    {'term': 'Antiarrhythmic [EPC]', 'count': 1551},
    {'term': 'Non-Standardized Plant Allergenic Extract [EPC]', 'count': 1519},
  ]
}
```

Concept Check 2.3

- Short Answer: What types of standard Python objects appear in the output of `pharm_class_json`?
- Fun Fact: In Python, it's common to work with deeply nested objects (like a Russian nested doll, or a Turducken).

```
{'term': 'Atypical Antipsychotic [EPC]', 'count': 1341},
```

Solution: We see instances of `int`, `str`, `dict`, and `list`.

```
{'term': 'Adrenergic alpha1-Agonists [MoA]', 'count': 1879},
```

▼ Section 2.4. Create `pharm_class_list`

```
{'term': 'Full Opioid Agonists [MoA]', 'count': 11521}
```

```
1 # When an API returns a nested collection of dicts and lists like this, we need to match the
2 # structure recursively using
3 # (a) dict-indexing to get values corresponding to specific keys and
4 # (b) for-loops to loop over lists.
5
6 # Accumulate pharmacologic classes and counts in a list of lists called pharm_class_list.
7 pharm_class_list = []
8 for pharm_class_count in pharm_class_json['results']:
9     pharm_class_list.append(
10         [
11             pharm_class_count['term'],
```

```
12         pharm_class_count['count'],
13     ]
14 )
15
16 # In case we want to track when these API results were obtained, let's also extract the date.
17 pharm_class_date = pharm_class_json['meta']['last_updated']
18
19 # Now let's print the date.
20 print(f'Date of API results: {pharm_class_date}')
21 print('\n')
22
23 # And then let's print pharm_class_list.
24 print(pharm_class_list)
```

Date of API results: 2023-02-27

```
[
  ['Cell-mediated Immunity [PE]', 6032],
  ['Increased Histamine Release [PE]', 6018],
  ['Allergens [CS]', 6011],
  ['Increased IgG Production [PE]', 3840],
  ['Anti-Inflammatory Agents', 3721],
  ['Cyclooxygenase Inhibitors [MoA]', 3721]
```

Concept Check 2.4

1. True or False: Changing Line 8 to `pharm_class_json['RESULTS']` (i.e., changing the dictionary key to all caps) would have the same effect.
 2. Short Answer: What types of standard Python objects appear in the definition of `pharm_class_list`?
- Fun Fact: Instead of bothering with a list of lists, we could have instead built a DataFrame row-by-row inside the for-loop. However, DataFrame operations inside a for-loop tend to be slow.

```
[ 'Uncompetitive NMDA Receptor Antagonists [MoA]', 2063 ],
```

Solution: False! In general, dictionary keys are case-sensitive in Python, just like variable names.

```
[ 'Ligands of Adrenergic Receptor [Term]', 10701 ]
```

▼ Section 2.5. Create `pharm_class_df`

```
[ 'Plant Proteins [CS]', 15981.
```

```
1 # Now that we've finish looping, we can put the definitions in a DataFrame called pharm_class_df.
2 pharm_class_df = pandas.DataFrame(pharm_class_list, columns = ['term', 'count'])
3
4 # We can also inspect the size of pharm_class_df.
5 print(f'The size of pharm_class_df: {pharm_class_df.shape}')
6 print('\n')
7
8 # In addition, we can get a sense of the average size pharmacologic class.
9 print(f"The median size pharmacologic class in pharm_class_df: {pharm_class_df['count'].median()}")
10 print('\n')
11
12 # Finally, we can display pharm_class_df.
13 print(f'The contents of pharm_class_df:')
14 display(pharm_class_df)
```

The size of `pharm_class_df`: (100, 2)

The median size pharmacologic class in `pharm_class_df`: 1079.5

The contents of `pharm_class_df`:

	term	count
0	Cell-mediated Immunity [PE]	6032
1	Increased Histamine Release [PE]	6018

Concept Check 2.5

- Short Answer: Other than the median, what are some descriptive statistics we might consider using to better understand the contents of `pharm_class_df`?
- Fun Fact: JSON-formatted data is useful because of how flexibly information can be nested. However, to actually work with the information inside, it's common to first rectangularize the JSON object.

Solution: The following section of the "Getting Started" guide for `pandas` gives a good overview:

https://pandas.pydata.org/docs/getting_started/intro_tutorials/06_calculate_statistics.html

```
30 generic_name_response = requests.get('https://api.fda.gov/drug/ndc.json?count=generic_name.exact')
31
```

▼ Section 2.6. Additional Exercises

For practice, we recommend the following:

- Run the code cell below.
- Repeat the steps in Sections 2.4-5 with the following two changes:
 - Form a `DataFrame` whose first column is `count`.
 - Calculate a statistic other than `median`.

```
1 # Let's try a different openFDA endpoint.
2 generic_name_response = requests.get('https://api.fda.gov/drug/ndc.json?count=generic_name.exact')
3
4 # Check the resulting status code to make sure the API call was successful, with 200 = "OK".
5 if generic_name_response.status_code == 200:
6     print('API call successful!\n')
7
8 # Finally, let's extract and print the JSON-formatted return value.
9 generic_name_json = generic_name_response.json()
10 print('Here\'s the resulting data structure:')
11 print(generic_name_json)
```


API call successful!

Here's the resulting data structure:

```
{
  'meta': {
    'disclaimer': 'Do not rely on openFDA to make decisions regarding medical care. While we make every effort to ensure that data is accurate, yo
all results are unvalidated. We may limit or otherwise restrict your access to the API in line with our Terms of Service.',
    'terms': 'https://open.fda.gov/terms/',
    'license': 'https://open.fda.gov/license/',
    'last_updated': '2023-02-27'
  },
  'results': [
    {'term': 'ALCOHOL', 'count': 1756},
    {'term': 'Alcohol', 'count': 1058},
    {'term': 'Ibuprofen', 'count': 940},
    {'term': 'Acetaminophen', 'count': 921},
    {'term': 'Zinc Oxide', 'count': 783},
    {'term': 'Ethyl Alcohol', 'count': 713},
    {'term': 'Benzalkonium Chloride', 'count': 636},
    {'term': 'BENZALKONIUM CHLORIDE', 'count': 584},
    {'term': 'Menthol', 'count': 582},
    {'term': 'Sodium Fluoride', 'count': 511},
    {'term': 'Salicylic Acid', 'count': 464},
    {'term': 'Isopropyl Alcohol', 'count': 428},
    {'term': 'Oxygen', 'count': 406},
    {'term': 'ZINC OXIDE', 'count': 391},
    {'term': 'Benzocaine', 'count': 381},
    {'term': 'Nicotine Polacrilex', 'count': 381},
    {'term': 'Avobenzone, Homosalate, Octisalate, Octocrylene', 'count': 372},
    {'term': 'Gabapentin', 'count': 355},
    {'term': 'Aspirin', 'count': 336},
    {'term': 'MENTHOL', 'count': 320},
    {'term': 'SALICYLIC ACID', 'count': 309},
    {'term': 'Diphenhydramine HCl', 'count': 302},
    {'term': 'Titanium Dioxide', 'count': 302},
    {'term': 'Titanium Dioxide, Zinc Oxide', 'count': 274},
    {'term': 'Lisinopril', 'count': 268},
    {'term': 'Hydrocortisone', 'count': 263},
    {'term': 'Naproxen Sodium', 'count': 260},
    {'term': 'Loratadine', 'count': 258},
    {'term': 'Famotidine', 'count': 257},
    {'term': 'TITANIUM DIOXIDE', 'count': 257},
    {'term': 'ETHYL ALCOHOL', 'count': 256},
    {'term': 'Levothyroxine Sodium', 'count': 249},
    {'term': 'Simethicone', 'count': 243},
    {'term': 'Aripiprazole', 'count': 242},
    {'term': 'Levetiracetam', 'count': 234},
    {'term': 'Benzalkonium chloride', 'count': 233},
    {'term': 'Guaifenesin', 'count': 233},
    {'term': 'OCTINOXATE and TITANIUM DIOXIDE', 'count': 231},
  ]
}
```

```
1 # Accumulate generic names and counts in a list of lists called generic_name_list.
2 generic_name_list = []
3 for generic_name_count in generic_name_json['results']:
4     generic_name_list.append(
5         [
6             generic_name_count['count'],
7             generic_name_count['term'],
8         ]
9     )
10
11 # In case we want to track when these API results were obtained, let's also extract the date.
12 generic_name_date = generic_name_json['meta']['last_updated']
13
14 # Now let's print the date.
15 print(f'Date of API results: {generic_name_date}')
```

```
16 print('\n')
17
18 # And then let's print generic_name_list.
19 print(generic_name_list)
20
21 # Now that we've finish looping, we can put the definitions in a DataFrame called generic_name_df.
22 generic_name_df = pandas.DataFrame(generic_name_list, columns = ['count', 'term'])
23
24 # We can also inspect the size of generic_name_df.
25 print(f'The size of generic_name_df: {generic_name_df.shape}')
26 print('\n')
27
28 # In addition, we can get a sense of the average size generic type.
29 print(f"The mean generics count in generic_name_df: {generic_name_df['count'].mean()}")
30 print('\n')
31
32 # Finally, we can display generic_name_df.
33 print(f'The contents of generic_name_df:')
34 display(generic_name_df)
```

Date of API results: 2023-02-27

```
[
  [1756, 'ALCOHOL'],
  [1058, 'Alcohol'],
  [940, 'Ibuprofen'],
  [921, 'Acetaminophen'],
  [783, 'Zinc Oxide'],
  [713, 'Ethyl Alcohol'],
  [636, 'Benzalkonium Chloride'],
  [584, 'BENZALKONIUM CHLORIDE'],
  [582, 'Menthol'],
  [511, 'Sodium Fluoride'],
  [464, 'Salicylic Acid'],
  [428, 'Isopropyl Alcohol'],
  [406, 'Oxygen'],
```

▼ Notes and Resources

```
[355, 'Gabanentin']
```

Want some ideas for what to do next? Here are our suggestions:

1. For more about the `pandas` package, including the methods used above, see the following:
 - <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shape.html>
 - <https://pandas.pydata.org/docs/reference/api/pandas.Series.median.html>
 - https://pandas.pydata.org/docs/reference/api/pandas.Series.value_counts.html
2. For more about the `requests` package, see <https://docs.python-requests.org/>
3. For more about the `rich` package, see <https://rich.readthedocs.io/>
4. For more about some of the Python features used, such as dictionaries, lists, and control flow with if-then-else conditionals and for-loops, we recommend the following chapters of [A Whirlwind Tour of Python](#):
 - <https://jakevdp.github.io/WhirlwindTourOfPython/06-built-in-data-structures.html>
 - <https://jakevdp.github.io/WhirlwindTourOfPython/07-control-flow-statements.html>
5. For more information on f-strings (i.e., Python strings like `f'https://httpstatuses.com/{http_status}'`), see <https://realpython.com/python-f-strings/>.
6. For background on the HTTP Request/Response Cycle, we recommend the following:
 - Brief Overview: https://backend.turing.edu/module2/lessons/how_the_web_works_http
 - Deeper Overview: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>
 - Summary of HTTP Status Codes: <https://httpstatuses.com/>
 - Google's Implementation of HTTP Status Code 418: <https://www.google.com/teapot>
7. For more practice with open web APIs, we recommend looking through <https://github.com/public-apis/public-apis> and trying to parse the output from <http://deckofcardsapi.com/>
8. For more about the complexity of parsing JSON in SAS, see <https://blogs.sas.com/content/sasdummy/2016/12/02/json-libname-engine-sas/>
9. We welcome follow-up conversations. You can connect with us on LinkedIn or email us at isaiah.lankham@gmail.com and matthew.t.slaughter@gmail.com

10. If you have a GitHub account (or don't mind creating one), you can also chat with us on Gitter at <https://gitter.im/saspy-bffs/community>

```
[155, 'Quetiapine Fumarate'],
[155, 'Titanium Dioxide and Zinc Oxide'],
[154, 'Carvedilol'],
[154, 'Clotrimazole'],
[154, 'Minoxidil'],
[153, 'Dimethicone'],
[153, 'Lidocaine Hydrochloride'],
[152, 'Diclofenac Sodium'],
[152, 'Rosuvastatin Calcium'],
[146, 'Diltiazem Hydrochloride'],
[145, 'Fluconazole'],
[144, 'DIPHENHYDRAMINE HYDROCHLORIDE'],
[142, 'Naproxen'],
[141, 'ACETAMINOPHEN'],
[136, 'Hydrochlorothiazide'],
[136, 'nicotine polacrilex'],
[134, 'Amoxicillin and Clavulanate Potassium'],
[134, 'Glycopyrrolate'],
[133, 'Buspirone Hydrochloride'],
[132, 'Acyclovir'],
[132, 'Glipizide'],
[131, 'LIDOCAINE HYDROCHLORIDE'],
[129, 'CHLOROXYLENOL'],
[129, 'Cephalexin'],
[128, 'ATORVASTATIN CALCIUM'],
[128, 'Labetalol Hydrochloride'],
[127, 'Aluminum Zirconium Tetrachlorohydrate GLY'],
[127, 'Doxycycline Hyclate'],
```

[Colab paid products](#) - [Cancel contracts here](#)