▾ 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
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/
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (13.3.4)
Requirement already satisfied: markdown-it-py<3.0.0,>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py<3.0
```

▾ 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! This will give us a chance to practice
2 # working with real-live JSON --- without needing to supply any type of access token or credentials.
3
```

```
 4 # Use an open web API to get the number of drugs available by pharmacologic class.
 5 # Note: By default, only the first 100 results are provided, sorted in descending order by count.
 6 # To retrieve more than the first 100 results, a combination of limit and skip parameters can be
 7 # used, as described at https://open.fda.gov/apis/paging/
 8 pharm_class_response = requests.get('https://api.fda.gov/drug/ndc.json?count=pharm_class.exact')
 9
10 # Check the resulting status code to make sure the API call was successful, with 200 = "OK".
11 http_status = pharm_class_response.status_code
12 http_status_info = f'https://httpstatuses.com/{http_status}'
13 if http_status == 200:
14     print('API call successful!\n')
15 print(f'See {http_status_info} for more information about HTTP status code {http_status}.')
```

    API call successful!

    See https://httpstatuses.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

```
[
    '__attrs__',
    '__bool__',
    '__class__',
    '__delattr__',
    '__dict__',
    '__dir__',
    '__doc__',
    '__enter__',
    '__eq__',
    '__exit__',
    '__format__',
    '__ge__',
    '__getattribute__',
    '__getstate__',
    '__gt__',
    '__hash__',
    '__init__',
    '__init_subclass__',
    '__iter__',
    '__le__',
    '__lt__',
    '__module__',
    '__ne__',
    '__new__',
    '__nonzero__',
    '__reduce__',
    '__reduce_ex__',
    '__repr__',
    '__setattr__',
    '__setstate__',
    '__sizeof__',
    '__str__',
    '__subclasshook__',
    '__weakref__',
    '_content',
    '_content_consumed',
    '_next',
    'apparent_encoding',
    'close',
    'connection',
    'content',
    'cookies',
```

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).

    `'Access-Control-Allow-Credentials': 'true'`

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

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

Section 2.3. Create pharm_class_json

`'X-Xss-Protection': '1: mode=block'`

```
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)
```

```json
{
    'meta': {
        'disclaimer': 'Do not rely on openFDA to make decisions regarding medical care. While we make every e
all results are unvalidated. We may limit or otherwise restrict your access to the API in line with our Terms
        'terms': 'https://open.fda.gov/terms/',
        'license': 'https://open.fda.gov/license/',
        'last_updated': '2023-05-10'
    },
    'results': [
        {'term': 'Cell-mediated Immunity [PE]', 'count': 6128},
        {'term': 'Increased Histamine Release [PE]', 'count': 6113},
        {'term': 'Allergens [CS]', 'count': 6106},
        {'term': 'Increased IgG Production [PE]', 'count': 3881},
        {'term': 'Anti-Inflammatory Agents', 'count': 3808},
        {'term': 'Cyclooxygenase Inhibitors [MoA]', 'count': 3808},
        {'term': 'Non-Steroidal [CS]', 'count': 3808},
        {'term': 'Nonsteroidal Anti-inflammatory Drug [EPC]', 'count': 3808},
        {'term': 'Histamine H1 Receptor Antagonists [MoA]', 'count': 2476},
        {'term': 'Corticosteroid Hormone Receptor Agonists [MoA]', 'count': 2452},
        {'term': 'Corticosteroid [EPC]', 'count': 2452},
        {'term': 'Histamine-1 Receptor Antagonist [EPC]', 'count': 2392},
        {'term': 'Osmotic Activity [MoA]', 'count': 2322},
        {'term': 'Increased Large Intestinal Motility [PE]', 'count': 2280},
        {'term': 'Osmotic Laxative [EPC]', 'count': 2235},
        {'term': 'Sigma-1 Agonist [EPC]', 'count': 2140},
        {'term': 'Sigma-1 Receptor Agonists [MoA]', 'count': 2140},
        {'term': 'Uncompetitive N-methyl-D-aspartate Receptor Antagonist [EPC]', 'count': 2140},
        {'term': 'Uncompetitive NMDA Receptor Antagonists [MoA]', 'count': 2140},
        {'term': 'Inhibition Large Intestine Fluid/Electrolyte Absorption [PE]', 'count': 2128},
        {'term': 'Decreased Central Nervous System Disorganized Electrical Activity [PE]', 'count': 2002},
        {'term': 'Adrenergic alpha1-Agonists [MoA]', 'count': 1928},
        {'term': 'alpha-1 Adrenergic Agonist [EPC]', 'count': 1928},
        {'term': 'Dietary Proteins [CS]', 'count': 1659},
        {'term': 'Non-Standardized Food Allergenic Extract [EPC]', 'count': 1654},
        {'term': 'Plant Proteins [CS]', 'count': 1643},
        {'term': 'Opioid Agonist [EPC]', 'count': 1642},
        {'term': 'Antiarrhythmic [EPC]', 'count': 1605},
        {'term': 'Pollen [CS]', 'count': 1585},
        {'term': 'Non-Standardized Plant Allergenic Extract [EPC]', 'count': 1561},
        {'term': 'Non-Standardized Pollen Allergenic Extract [EPC]', 'count': 1511},
        {'term': 'Local Anesthesia [PE]', 'count': 1487},
        {'term': 'Serotonin Uptake Inhibitors [MoA]', 'count': 1444},
        {'term': 'Amide Local Anesthetic [EPC]', 'count': 1441},
        {'term': 'Amides [CS]', 'count': 1441},
        {'term': 'beta-Adrenergic Blocker [EPC]', 'count': 1438},
        {'term': 'Atypical Antipsychotic [EPC]', 'count': 1394},
```

```
{'term': 'Central Nervous System Stimulation [PE]', 'count': 1376}.
```

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': 'Decreased Prostaglandin Production [PE]', 'count': 1173]
```

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

```
{'term': 'Decreased Respiratory Secretion Viscosity [PE]', 'count': 1118},
```

Section 2.4. Create pharm_class_list

```
{ term : Angiotensin 2 Receptor Blocker [EPC] , count : 1091},
```

```python
 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: **2023-05-10**

```
[
    ['Cell-mediated Immunity [PE]', 6128],
    ['Increased Histamine Release [PE]', 6113],
    ['Allergens [CS]', 6106],
    ['Increased IgG Production [PE]', 3881],
    ['Anti-Inflammatory Agents', 3808],
    ['Cyclooxygenase Inhibitors [MoA]', 3808],
    ['Non-Steroidal [CS]', 3808],
    ['Nonsteroidal Anti-inflammatory Drug [EPC]', 3808],
    ['Histamine H1 Receptor Antagonists [MoA]', 2476],
    ['Corticosteroid Hormone Receptor Agonists [MoA]', 2452],
    ['Corticosteroid [EPC]', 2452],
    ['Histamine-1 Receptor Antagonist [EPC]', 2392],
    ['Osmotic Activity [MoA]', 2322],
    ['Increased Large Intestinal Motility [PE]', 2280],
    ['Osmotic Laxative [EPC]', 2235],
    ['Sigma-1 Agonist [EPC]', 2140],
    ['Sigma-1 Receptor Agonists [MoA]', 2140],
    ['Uncompetitive N-methyl-D-aspartate Receptor Antagonist [EPC]', 2140],
    ['Uncompetitive NMDA Receptor Antagonists [MoA]', 2140],
    ['Inhibition Large Intestine Fluid/Electrolyte Absorption [PE]', 2128],
    ['Decreased Central Nervous System Disorganized Electrical Activity [PE]', 2002],
    ['Adrenergic alpha1-Agonists [MoA]', 1928],
    ['alpha-1 Adrenergic Agonist [EPC]', 1928],
    ['Dietary Proteins [CS]', 1659],
    ['Non-Standardized Food Allergenic Extract [EPC]', 1654],
    ['Plant Proteins [CS]', 1643],
    ['Opioid Agonist [EPC]', 1642],
    ['Antiarrhythmic [EPC]', 1605],
    ['Pollen [CS]', 1585],
    ['Non-Standardized Plant Allergenic Extract [EPC]', 1561],
    ['Non-Standardized Pollen Allergenic Extract [EPC]', 1511],
    ['Local Anesthesia [PE]', 1487],
    ['Serotonin Uptake Inhibitors [MoA]', 1444],
    ['Amide Local Anesthetic [EPC]', 1441],
    ['Amides [CS]', 1441],
    ['beta-Adrenergic Blocker [EPC]', 1438],
    ['Atypical Antipsychotic [EPC]', 1394],
    ['Central Nervous System Stimulation [PE]', 1376],
    ['Stimulation Large Intestine Fluid/Electrolyte Secretion [PE]', 1345],
    ['Calculi Dissolution Agent [EPC]', 1332],
    ['Central Nervous System Stimulant [EPC]', 1329],
    ['Adrenergic beta-Antagonists [MoA]', 1301],
    ['Anti-epileptic Agent [EPC]', 1271],
```

```
['Copper Absorption Inhibitor [EPC]', 1257],
```

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.

```
['Inhibition Small Intestine Fluid/Electrolyte Absorption [PE]', 1007],
['M                 A   i   [M Al  1067]
```

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

```
['Standardized Chemical Allergen [EPC]', 1009],
```

Section 2.5. Create pharm_class_df

```
['Calcium [CS]', 948],
```

```
 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: **1108.0**

The contents of pharm_class_df:

	term	count
0	Cell-mediated Immunity [PE]	6128
1	Increased Histamine Release [PE]	6113
2	Allergens [CS]	6106
3	Increased IgG Production [PE]	3881
4	Anti-Inflammatory Agents	3808
...	...	...
95	Calcium Channel Blocker [EPC]	699
96	Tricyclic Antidepressant [EPC]	683
97	Cholinergic Nicotinic Agonist [EPC]	670
	Nicotine [CS]	670

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

▾ 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 e
all results are unvalidated. We may limit or otherwise restrict your access to the API in line with our Terms
        'terms': 'https://open.fda.gov/terms/',
        'license': 'https://open.fda.gov/license/',
        'last_updated': '2023-05-10'
    },
    'results': [
        {'term': 'ALCOHOL', 'count': 1736},
        {'term': 'Alcohol', 'count': 1063},
        {'term': 'Acetaminophen', 'count': 948},
        {'term': 'Ibuprofen', 'count': 947},
        {'term': 'Zinc Oxide', 'count': 816},
        {'term': 'Ethyl Alcohol', 'count': 718},
        {'term': 'Benzalkonium Chloride', 'count': 644},
        {'term': 'Menthol', 'count': 601},
        {'term': 'BENZALKONIUM CHLORIDE', 'count': 575},
        {'term': 'Sodium Fluoride', 'count': 526},
        {'term': 'Salicylic Acid', 'count': 479},
        {'term': 'Isopropyl Alcohol', 'count': 430},
        {'term': 'Oxygen', 'count': 412},
        {'term': 'ZINC OXIDE', 'count': 407},
        {'term': 'Avobenzone, Homosalate, Octisalate, Octocrylene', 'count': 399},
        {'term': 'Benzocaine', 'count': 388},
        {'term': 'Nicotine Polacrilex', 'count': 386},
        {'term': 'Gabapentin', 'count': 361},
        {'term': 'Aspirin', 'count': 342},
        {'term': 'MENTHOL', 'count': 333},
        {'term': 'SALICYLIC ACID', 'count': 315},
        {'term': 'Diphenhydramine HCl', 'count': 305},
        {'term': 'Titanium Dioxide', 'count': 299},
        {'term': 'Lisinopril', 'count': 284},
        {'term': 'Titanium Dioxide, Zinc Oxide', 'count': 278},
        {'term': 'Levothyroxine Sodium', 'count': 273},
        {'term': 'Famotidine', 'count': 272},
        {'term': 'Hydrocortisone', 'count': 270},
        {'term': 'Loratadine', 'count': 269},
        {'term': 'Naproxen Sodium', 'count': 269},
        {'term': 'TITANIUM DIOXIDE', 'count': 258},
        {'term': 'Simethicone', 'count': 253},
        {'term': 'Pyrithione Zinc', 'count': 252},
        {'term': 'ETHYL ALCOHOL', 'count': 249},
```

```
        {'term': 'Aripiprazole'. 'count': 247}.
 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)
```

```
[
    [1736, 'ALCOHOL'],
    [1063, 'Alcohol'],
    [948, 'Acetaminophen'],
    [947, 'Ibuprofen'],
    [816, 'Zinc Oxide'],
    [718, 'Ethyl Alcohol'],
    [644, 'Benzalkonium Chloride'],
    [601, 'Menthol'],
    [575, 'BENZALKONIUM CHLORIDE'],
    [526, 'Sodium Fluoride'],
    [479, 'Salicylic Acid'],
    [430, 'Isopropyl Alcohol'],
    [412, 'Oxygen'],
    [407, 'ZINC OXIDE'],
    [399, 'Avobenzone, Homosalate, Octisalate, Octocrylene'],
    [388, 'Benzocaine'],
    [386, 'Nicotine Polacrilex'],
    [361, 'Gabapentin'],
    [342, 'Aspirin'],
    [333, 'MENTHOL'],
    [315, 'SALICYLIC ACID'],
    [305, 'Diphenhydramine HCl'],
    [299, 'Titanium Dioxide'],
    [284, 'Lisinopril'],
    [278, 'Titanium Dioxide, Zinc Oxide'],
    [273, 'Levothyroxine Sodium'],
    [272, 'Famotidine'],
    [270, 'Hydrocortisone'],
    [269, 'Loratadine'],
    [269, 'Naproxen Sodium'],
    [258, 'TITANIUM DIOXIDE'],
    [253, 'Simethicone'],
    [252, 'Pyrithione Zinc'],
    [249, 'ETHYL ALCOHOL'],
    [247, 'Aripiprazole'],
    [246, 'Guaifenesin'],
    [244, 'Pregabalin'],
    [237, 'Lamotrigine'],
    [237, 'Levetiracetam'],
    [234, 'Benzalkonium chloride'],
    [231, 'OCTINOXATE and TITANIUM DIOXIDE'],
    [230, 'Calcium Carbonate'],
    [229, 'Omeprazole'],
```

```
 [227, 'Diphenhydramine Hydrochloride'],
```

▾ Notes and Resources

```
 [209. 'PREGABALIN'],
```

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

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

The mean generics count in generic_name_df: 278.99

The contents of generic_name_df:
```

	count	term
0	1736	ALCOHOL
1	1063	Alcohol
2	948	Acetaminophen
3	947	Ibuprofen
4	816	Zinc Oxide
...	...	...
95	131	Alprazolam