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 with SAS OnDemand for Academics (ODA)

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. Looking for "extra credit"? Please let us know if you spot any typos!

Install and import packages

```
# Install the rich module for colorful printing
!pip install rich

# We'll use IPython to display DataFrames or HTML content
from IPython.display import display, HTML

# We'll use the pandas package to create and manipulate DataFrame objects
```

```
import pandas

# We'll use the requests package to call a web API
import requests

# We're overwriting the default print function with rich.print
from rich import print

# We're also setting the maximum line width of rich.print to be a bit wider (to avoid line wrapping)
from rich import get_console
console = get_console()
console.width = 165
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting rich
  Downloading rich-12.5.1-py3-none-any.whl (235 kB)
     |████████████████████████████████| 235 kB 5.2 MB/s
Requirement already satisfied: typing-extensions<5.0,>=4.0.0 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pygments<3.0.0,>=2.6.0 in /usr/local/lib/python3.7/dist-packages (from rich) (
Collecting commonmark<0.10.0,>=0.9.0
  Downloading commonmark-0.9.1-py2.py3-none-any.whl (51 kB)
     |████████████████████████████████| 51 kB 7.3 MB/s
Installing collected packages: commonmark, rich
Successfully installed commonmark-0.9.1 rich-12.5.1
```

Part 2. Rectangularizing unstructured data in Python applications

Section 2.1. Create pharm_class_response

```
# Let's explore one of the many endpoints of the openFDA API!

# Use an open web API to get the number of drugs available by pharmacologic class.
# Note: By default, only the first 100 results are provided, sorted in descending order by count.
# To retrieve more than the first 100 results, a combination of limit and skip parameters can be
```

```
# used, as described at https://open.fda.gov/apis/paging/
pharm_class_response = requests.get('https://api.fda.gov/drug/ndc.json?count=pharm_class.exact')

# Check the resulting status code to make sure the API call was successful, with 200 = "OK".
http_status = pharm_class_response.status_code
http_status_info = f'https://httpstatuses.com/{http_status}'
if http_status == 200:
    print('API call successful!\n')
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 12 to a single-equals (=) would have the same effect.
2. True or False: Removing the indentation on Line 13 would have the same effect.

- Fun Fact: The FDA provides many open APIs. Examples for the APIs related specifically to the National Drug Code (NDC) database, including the API used above, 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

```
# What exactly makes up an HTTP response?
print(dir(pharm_class_response))
print('\n')

# Let's take a look at the HTTP headers used behind the scenes
print(dict(pharm_class_response.headers))
```

```
[
    '__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',
    'elapsed',
    'encoding',
    'headers',
```

```
'history',
'is_permanent_redirect',
'is_redirect',
'iter_content',
'iter_lines',
'json',
'links',
'next',
'ok',
'raise_for_status',
'raw',
'reason',
'request',
'status_code',
'text',
```

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.

```
x-cache : MISS ,
```

```
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

```
# Extract and print the JSON-formatted list of counts of drugs by pharmacologic class.
pharm_class_json = pharm_class_response.json()
print(pharm_class_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': '2022-09-02'
    },
    'results': [
        {'term': 'Cell-mediated Immunity [PE]', 'count': 6415},
        {'term': 'Increased Histamine Release [PE]', 'count': 6392},
        {'term': 'Allergens [CS]', 'count': 6384},
        {'term': 'Anti-Inflammatory Agents', 'count': 4157},
        {'term': 'Cyclooxygenase Inhibitors [MoA]', 'count': 4157},
        {'term': 'Non-Steroidal [CS]', 'count': 4157},
        {'term': 'Nonsteroidal Anti-inflammatory Drug [EPC]', 'count': 4157},
        {'term': 'Increased IgG Production [PE]', 'count': 4136},
        {'term': 'Corticosteroid Hormone Receptor Agonists [MoA]', 'count': 2693},
        {'term': 'Corticosteroid [EPC]', 'count': 2693},
        {'term': 'Histamine H1 Receptor Antagonists [MoA]', 'count': 2622},
        {'term': 'Histamine-1 Receptor Antagonist [EPC]', 'count': 2515},
        {'term': 'Osmotic Activity [MoA]', 'count': 2255},
        {'term': 'Increased Large Intestinal Motility [PE]', 'count': 2243},
        {'term': 'Sigma-1 Agonist [EPC]', 'count': 2185},
        {'term': 'Sigma-1 Receptor Agonists [MoA]', 'count': 2185},
        {'term': 'Uncompetitive N-methyl-D-aspartate Receptor Antagonist [EPC]', 'count': 2185},
```


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': 'Pollen [CS]', 'count': 1721},
```


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


```
        {'term': 'Non-Standardized Pollen Allergenic Extract [EPC]', 'count': 1639},
```


Section 2.4. Create pharm_class_list


```
        {'term': 'Serotonin Uptake Inhibitors [MoA]', 'count': 1593},
        {'term': 'Beta-Adrenergic Blocker [EPC]', 'count': 1553},
```


```
# When an API returns a nested collection of dicts and lists like this, we need to match the
# structure recursively using
# (a) dict-indexing to get values corresponding to specific keys and
```

```
# (b) for-loops to loop over lists.

# Accumulate pharmacologic classes and counts in a list of lists called pharm_class_list.
pharm_class_list = []
for pharm_class_count in pharm_class_json['results']:
    pharm_class_list.append(
        [
            pharm_class_count['term'],
            pharm_class_count['count'],
        ]
    )

# In case we want to track when these API results were obtained, let's also extract the date.
pharm_class_date = pharm_class_json['meta']['last_updated']

# Now let's print the date.
print(f'Date of API results: {pharm_class_date}')
print('\n')

# And then let's print pharm_class_list.
print(pharm_class_list)
```

```
Date of API results: 2022-09-02

[
    ['Cell-mediated Immunity [PE]', 6415],
    ['Increased Histamine Release [PE]', 6392],
    ['Allergens [CS]', 6384],
    ['Anti-Inflammatory Agents', 4157],
    ['Cyclooxygenase Inhibitors [MoA]', 4157],
    ['Non-Steroidal [CS]', 4157],
    ['Nonsteroidal Anti-inflammatory Drug [EPC]', 4157],
    ['Increased IgG Production [PE]', 4136],
    ['Corticosteroid Hormone Receptor Agonists [MoA]', 2693],
    ['Corticosteroid [EPC]', 2693],
    ['Histamine H1 Receptor Antagonists [MoA]', 2622],
    ['Histamine-1 Receptor Antagonist [EPC]', 2515],
    ['Osmotic Activity [MoA]', 2255],
    ['Increased Large Intestinal Motility [PE]', 2243],
    ['Sigma-1 Agonist [EPC]', 2185],
    ['Sigma-1 Receptor Agonists [MoA]', 2185],
    ['Uncompetitive N-methyl-D-aspartate Receptor Antagonist [EPC]', 2185],
    ['Uncompetitive NMDA Receptor Antagonists [MoA]', 2185],
    ['Osmotic Laxative [EPC]', 2174],
    ['Decreased Central Nervous System Disorganized Electrical Activity [PE]', 2086],
    ['Inhibition Large Intestine Fluid/Electrolyte Absorption [PE]', 2075],
    ['Adrenergic alpha1-Agonists [MoA]', 1993],
    ['alpha-1 Adrenergic Agonist [EPC]', 1993],
    ['Opioid Agonist [EPC]', 1802],
    ['Pollen [CS]', 1721],
    ['Plant Proteins [CS]', 1680],
    ['Dietary Proteins [CS]', 1642],
    ['Non-Standardized Pollen Allergenic Extract [EPC]', 1639],
    ['Non-Standardized Food Allergenic Extract [EPC]', 1638],
    ['Non-Standardized Plant Allergenic Extract [EPC]', 1594],
    ['Antiarrhythmic [EPC]', 1587],
    ['Serotonin Uptake Inhibitors [MoA]', 1553],
    ['beta-Adrenergic Blocker [EPC]', 1553],
    ['Local Anesthesia [PE]', 1462],
    ['Amide Local Anesthetic [EPC]', 1412],
    ['Amides [CS]', 1412],
    ['Adrenergic beta-Antagonists [MoA]', 1400],
    ['Stimulation Large Intestine Fluid/Electrolyte Secretion [PE]', 1390],
    ['Anti-epileptic Agent [EPC]', 1365],
    ['Calculi Dissolution Agent [EPC]', 1361]
```

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]', 1104],
```

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

Section 2.5. Create pharm_class_df

```
['Calcium [CS]', 1012]
```

```
# Now that we've finish looping, we can put the definitions in a DataFrame called pharm_class_df.
pharm_class_df = pandas.DataFrame(pharm_class_list, columns = ['term', 'count'])

# We can also inspect the size of pharm_class_df.
print(f'The size of pharm_class_df: {pharm_class_df.shape}')
print('\n')

# In addition, we can get a sense of the average size pharmacologic class.
print(f"The median size pharmacologic class in pharm_class_df: {pharm_class_df['count'].median()}")
print('\n')

# Finally, we can display pharm_class_df.
print(f'The contents of pharm_class_df:')
display(pharm_class_df)
```

```
The size of pharm_class_df: (100, 2)
The median size pharmacologic class in pharm_class_df: 1199.0
The contents of pharm_class_df:
```

	term	count
0	Cell-mediated Immunity [PE]	6415
1	Increased Histamine Release [PE]	6392
2	Allergens [CS]	6384
3	Anti-Inflammatory Agents	4157
4	Cyclooxygenase Inhibitors [MoA]	4157
...	...	...
95	Nicotine [CS]	656
96	Tricyclic Antidepressant [EPC]	652

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`.

```
# Let's try a different openFDA endpoint.
generic_name_response = requests.get('https://api.fda.gov/drug/ndc.json?count=generic_name.exact')

# Check the resulting status code to make sure the API call was successful, with 200 = "OK".
if generic_name_response.status_code == 200:
    print('API call successful!\n')

# Finally, let's extract and print the JSON-formatted return value.
generic_name_json = generic_name_response.json()
print('Here\'s the resulting data structure:')
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': '2022-09-02'
    },
    'results': [
        {'term': 'ALCOHOL', 'count': 2622},
        {'term': 'Alcohol', 'count': 1359},
        {'term': 'Ibuprofen', 'count': 1039},
        {'term': 'Acetaminophen', 'count': 1012},
        {'term': 'Ethyl Alcohol', 'count': 902},
        {'term': 'Benzalkonium Chloride', 'count': 814},
        {'term': 'Zinc Oxide', 'count': 745},
        {'term': 'BENZALKONIUM CHLORIDE', 'count': 734},
        {'term': 'Menthol', 'count': 619},
        {'term': 'Sodium Fluoride', 'count': 570},
        {'term': 'Isopropyl Alcohol', 'count': 510},
        {'term': 'Salicylic Acid', 'count': 502},
        {'term': 'Oxygen', 'count': 452},
        {'term': 'Benzocaine', 'count': 421},
        {'term': 'ETHYL ALCOHOL', 'count': 410},
        {'term': 'Aspirin', 'count': 399},
        {'term': 'Gabapentin', 'count': 393},
        {'term': 'ZINC OXIDE', 'count': 388},
        {'term': 'Nicotine Polacrilex', 'count': 379},
        {'term': 'Diphenhydramine HCl', 'count': 346},
        {'term': 'Avobenzone, Homosalate, Octisalate, Octocrylene', 'count': 336},
        {'term': 'SALICYLIC ACID', 'count': 334},
        {'term': 'Lisinopril', 'count': 308},
        {'term': 'MENTHOL', 'count': 307},
        {'term': 'Levothyroxine Sodium', 'count': 293},
        {'term': 'Naproxen Sodium', 'count': 288},
        {'term': 'Hydrocortisone', 'count': 284},
        {'term': 'Loratadine', 'count': 274},
        {'term': 'TITANIUM DIOXIDE', 'count': 267}
```

```
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']:
```

```
    generic_name_list.append(
        [
            generic_name_count['count'],
            generic_name_count['term'],
        ]
    )

# In case we want to track when these API results were obtained, let's also extract the date.
generic_name_date = generic_name_json['meta']['last_updated']

# Now let's print the date.
print(f'Date of API results: {generic_name_date}')
print('\n')

# And then let's print generic_name_list.
print(generic_name_list)

# Now that we've finish looping, we can put the definitions in a DataFrame called generic_name_df.
generic_name_df = pandas.DataFrame(generic_name_list, columns = ['count','term'])

# We can also inspect the size of generic_name_df.
print(f'The size of generic_name_df: {generic_name_df.shape}')
print('\n')

# In addition, we can get a sense of the average size generic type.
print(f"The mean generics count in generic_name_df: {generic_name_df['count'].mean()}")
print('\n')

# Finally, we can display generic_name_df.
print(f'The contents of generic_name_df:')
display(generic_name_df)
```

Date of API results: 2022-09-02

```
[
    [2622, 'ALCOHOL'],
    [1359, 'Alcohol'],
    [1039, 'Ibuprofen'],
    [1012, 'Acetaminophen'],
    [902, 'Ethyl Alcohol'],
    [814, 'Benzalkonium Chloride'],
    [745, 'Zinc Oxide'],
    [734, 'BENZALKONIUM CHLORIDE'],
    [619, 'Menthol'],
    [570, 'Sodium Fluoride'],
    [510, 'Isopropyl Alcohol'],
    [502, 'Salicylic Acid'],
    [452, 'Oxygen'],
    [421, 'Benzocaine'],
    [410, 'ETHYL ALCOHOL'],
    [399, 'Aspirin'],
    [393, 'Gabapentin'],
    [388, 'ZINC OXIDE'],
    [379, 'Nicotine Polacrilex'],
    [346, 'Diphenhydramine HCl'],
    [336, 'Avobenzone, Homosalate, Octisalate, Octocrylene'],
    [334, 'SALICYLIC ACID'],
    [308, 'Lisinopril'],
    [307, 'MENTHOL'],
    [293, 'Levothyroxine Sodium'],
    [288, 'Naproxen Sodium'],
    [284, 'Hydrocortisone'],
    [274, 'Loratadine'],
    [267, 'TITANIUM DIOXIDE'],
    [264, 'Aripiprazole'],
    [264, 'Simethicone'],
    [263, 'Famotidine'],
    [258, 'OCTINOXATE, TITANIUM DIOXIDE'],
    [252, 'Titanium Dioxide and Zinc Oxide'],
    [250, 'Lamotrigine'],
    [248, 'Benzalkonium chloride'],
    [248, 'Guaifenesin'],
    [244, 'OCTINOXATE and TITANIUM DIOXIDE'],
    [240, 'Omeprazole'],
    [238, 'Prednisone'],
    [237, 'Amoxicillin'],
    [236, 'Levetiracetam'],
    [236, 'PREGABALIN'],
```

```
[236, 'Titanium Dioxide'],
[235, 'Titanium Dioxide, Zinc Oxide'],
[233, 'Diphenhydramine Hydrochloride'],
[228, 'Calcium Carbonate'],
[226, 'Lidocaine'],
[219, 'Cetirizine Hydrochloride'],
[219, 'Pyrithione Zinc'],
```

Notes and Resources

```
[214, 'Hand Sanitizer'],
```

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

0	2622	ALCOHOL
1	1359	Alcohol
2	1039	Ibuprofen
3	1012	Acetaminophen
4	902	Ethyl Alcohol
...	...	...
95	142	Cephalexin
96	142	Venlafaxine Hydrochloride
97	139	Methocarbamol
98	139	Metronidazole
99	138	CHLOROXYLENOL

100 rows × 2 columns