Everything is Better with Friends

Using SAS in Python Applications with SASPy and Open-Source Tooling (Getting Started)

A few notes before we get started...

1. Please enable line numbers using the Tools menu: **Tools** -> **Settings** -> **Editor** -> **Show line numbers** -> **Save**
2. To execute code examples, you'll need credentials for the following accounts:
   - 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.)
   - SAS OnDemand for Academics. (You can create an account for free at https://welcome.oda.sas.com/ using an existing SAS Profile account. If you don't already have a SAS Profile account, you can create one for free using the "Don't have a SAS Profile?" link on the ODA login page.)
3. To save a copy of this notebook, along with any edits you make, please use the File menu: **File** -> **Save a copy in Drive**
4. We also recommend enabling the Table of Contents using the View menu: **View** -> **Table of contents**
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!

Section 0. Setup and Connect to SAS OnDemand for Academics (ODA)

Example 0.1. Install the SASPy package

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter.

```
!pip install saspy
```

```
Collecting saspy
  Downloading saspy-3.7.5.tar.gz (7.8 MB)
     |████████████████████████████████| 7.8 MB 3.8 MB/s
Building wheels for collected packages: saspy
  Building wheel for saspy (setup.py) ... done
  Created wheel for saspy: filename=saspy-3.7.5-py3-none-any.whl size=7858579
  Stored in directory: /root/.cache/pip/wheels/8f/1d/b2/2dcc8c22c9fa58dce6ad65
Successfully built saspy
Installing collected packages: saspy
Successfully installed saspy-3.7.5
```

Line-by-Line Code Explanation:

- Line 1: Install the Python package `saspy` inside the current Google Colab session.

Notes about Example 0.1:

1. Google Colab is based on JupyterLab, which is a popular open source platform for programming in Python and other languages.
2. The exclamation mark is used in JupyterLab to pass a command to the underlying operating system, which is Ubuntu Linux for Google Colab sessions.
3. `pip` is the standard command line tool for installing Python packages. (Fun fact: The name "pip" is a recursive acronym meaning "pip installs packages.")

Example 0.2. Connect to SAS ODA and start a SAS session

Instructions:

1. Determine the Region for your ODA account by logging into https://welcome.oda.sas.com/. You should see the value `Asia Pacific`, `Europe`, or `United States` next to your username in the upper-right corner.
2. If your ODA account is associated with a Region other than `United States`, edit the code cell below by adding a number sign (`#`) at the beginning of Line 8, and then do the following:
   - If your ODA account is associated with the Region `Europe`, remove the number sign (`#`) at the beginning of Line 11.
   - If your ODA account is associated with the Region `Asia Pacific`, remove the number sign (`#`) at the beginning of Line 14.
3. Click anywhere in the code cell, and run the cell using Shift-Enter.

Notes: The number sign (`#`) is used to comment out a single line of text in Python. The value of `iomhost` must correspond to the Region for your ODA account, with only one of Line 8, 11, and 14 left uncommented. For more information about Regions, please see the ODA documentation.

```
import saspy
sas = saspy.SASsession(
    java='/usr/bin/java',
    iomport=8591,
    encoding='utf-8',

    # The following line should be uncommented if, and only if, your ODA account is
    iomhost = ['odaws01-usw2.oda.sas.com', 'odaws02-usw2.oda.sas.com', 'odaws03-usw

    # The following line should be uncommented if, and only if, your ODA account is
    #iomhost = ['odaws01-euw1.oda.sas.com','odaws02-euw1.oda.sas.com'],

    # The following line should be uncommented if, and only if, your ODA account is
    #iomhost = ['odaws01-apse1.oda.sas.com','odaws02-apse1.oda.sas.com'],

)
print(sas)
```

```
    Using SAS Config named: default
    Please enter the IOM user id: isaiah.lankham@ucop.edu
    Please enter the password for IOM user : ••••••••••
    SAS Connection established. Subprocess id is 170

    Access Method         = IOM
    SAS Config name       = default
    SAS Config file       = /usr/local/lib/python3.7/dist-packages/saspy/sascfg.py
    WORK Path             = /saswork/SAS_work7B9900016AF7_odaws01-usw2.oda.sas.com
    SAS Version           = 9.04.01M6P11072018
    SASPy Version         = 3.7.5
    Teach me SAS          = False
    Batch                 = False
    Results               = Pandas
    SAS Session Encoding  = utf-8
    Python Encoding value = utf-8
    SAS process Pid value = 92919
```

Line-by-Line Code Explanation:

- Line 1: Load the `saspy` module, which was installed into your Google Colab session in Example 0.1 above.
- Lines 2-16: Connect to SAS ODA and establish a SAS session. A python object named `sas` is created, which will be used in most subsequent examples.
- Line 17: The `print` function is used to display attributes of the `sas` object.

Notes about Example 0.2:

1. If your SAS session times out or terminates (e.g., by closing this notebook or using the `sas.endsas()` command), you'll need to run this cell again and re-enter your ODA login credentials.
2. In this notebook, we're using the "IOM using Java" method to connect to ODA. The SASPy documentation lists methods for several other scenarios, including a local Python installation connecting to SAS running either on the same machine or a remote server.
3. If an error is displayed, an incompatible kernel has been chosen. This Notebook was developed using the Python 3.7 kernel provided in Google Colab as of November 2021.

Example 0.3. Test SAS connection

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter.

```
sas.submitLST("ods text='Hello, SAS ODA!';")
```

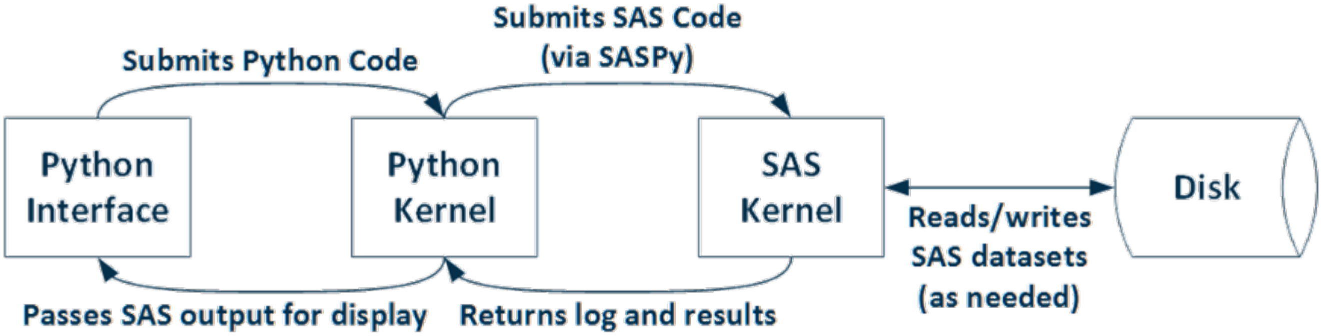
```
                                   The SAS System

    Hello, SAS ODA!
```

Line-by-Line Code Explanation:

- Line 1: Use the `submitLST` method to run some SAS code and display the results. There should be a short message displayed as SAS HTML output.

Notes about Example 0.3:

1. If everything runs successfuly up to this point, it proves that SAS and Python are communicating!
2. This brief example demonstrates the basic purpose of `saspy`, which is to use Python to submit SAS code to a SAS Kernel and get SAS logs and output back. The following figure illustrates the basic architecture, with Google Colab providing the Python Interface/Kernel and SAS OnDemand for Academics providing both the SAS Kernel and the Disk to store SAS datasets:



Section 1. Python Code Conventions and Data Structures

Example 1.1. Meet the Python environment

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
import platform
print(platform.dist())
print()
print(platform.sys.version)
print()
print(sorted(list(platform.sys.modules)))
```

```
('Ubuntu', '18.04', 'bionic')

3.7.12 (default, Sep 10 2021, 00:21:48)
[GCC 7.5.0]

['IPython', 'IPython.core', 'IPython.core.alias', 'IPython.core.application',
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWar
```

Line-by-Line Code Explanation:

- Line 1: Load the `platform` module.
- Lines 2-3: Print information about the underlying operating system, followed by a blank line.
- Lines 4-5: Print information about the Python version, followed by a blank line.
- Line 6: Print a sorted list of all modules currently available to be loaded by the Python kernel.

Exercise 1.1.1. True or False: Changing Line 1 to `IMPORT PLATFORM` would result in an execution error.

True. Because Python is case-sensitive, `IMPORT PLATFORM` would result in an error.

Exercise 1.1.2. True or False: The example code should result in an execution error because there are no terminating semicolons.

False. Semicolons are not required to terminate a Python statement.

Notes about Example 1.1:

1. This example illustrates four ways Python syntax differs from SAS:
    - Unlike SAS, capitalization matters in Python. Changing Line 1 to `IMPORT PLATFORM` would produce an error.
    - Unlike SAS, semicolons are optional in Python, and they are typically only used to separate multiple statements placed on the same line. E.g., Lines 1-2 could be combined into `import platform; print(platform.dist())`
    - Unlike SAS, dot-notation has a consistent meaning in Python and can be used to reference objects nested inside each other at any depth. E.g., on Line 4, the `platform` module object invokes the sub-module object `sys` nested inside of it, and `sys` invokes the object `version` nested inside of it. (Think Russian nesting dolls or turduckens.)
2. To increase performance, only a small number of modules in Python's standard library are available to use directly by default, which is why the `platform` module needs to be explicitly loaded before use.
3. Python comes with a large standard library because of its "batteries included" philosophy, and numerous third-party modules are also actively developed and made freely available through sites like https://github.com/ and https://pypi.org/. For the examples in this notebook, we'll need these third-party modules:
    - `IPython`, which stands for "Interactive Python." JupyterLab builds upon `IPython`, so it's already available by default in Google Colab.
    - `pandas`, which provided `DataFrame` objects. DataFrames can be found in other languages, like R, and are similar to SAS datasets. Because `pandas` is a fundamental package for working with data in Python, it's already available by default in Google Colab.
    - `saspy`, which is a Python package developed by the SAS Institute for connecting to a SAS kernel. Because `saspy` doesn't come pre-installed in Google Colab sessions, we had to manually install it in Section 0 above.

Example 1.2. Hello, data!

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
hello_world_str = 'Hello, Colab!'
print(hello_world_str)
print()
if hello_world_str == 'Hello, Colab!':
    print(type(hello_world_str))
else:
    print("Error: The string doesn't have the expected value!")
```

```
Hello, Colab!

<class 'str'>
```

Line-by-Line Code Explanation:

- Lines 1-3: Create a string object (`str` for short) named `hello_world_str`, and print it's value, followed by a blank line.
- Lines 4-7: Check to see if `hello_world_str` has the expected value. If so, print it's type. Otherwise, print an error message.

Exercise 1.2.1. Which of the following changes to the above example would result in an error? (Select all that apply.)

a. Removing an equal sign (`=`) so that Line 4 becomes `if hello_world_str = 'Hello, Jupyter!'`
b. Removing Line 3 (`print()`)
c. Unindenting Line 5 (`print(type(hello_world_str))`)

Changes a and c would result in errors.

Exercise 1.2.2. Write several lines of Python code to produce the following output:

```
42

<class 'int'>
```

```
hello_world_int = 42
print(hello_world_int)
print()
if hello_world_int == 42:
    print(type(hello_world_int))
else:
    print("Error: The int doesn't have the expected value!")
```

```
42

<class 'int'>
```

Notes about Example 1.2:

1. This example illustrates four more ways Python differs from SAS:
    - Unlike SAS, variables are dynamically typed in Python. After Line 1 has been used to create `hello_world_str`, it can be assigned a new value later with a completely different type. E.g., we could change Line 3 to be `hello_world_str = 42` so that `type(hello_world_str)` becomes `<class 'int'>`.
    - Unlike SAS, single-equals (`=`) only ever means assignment, and double-equals (`==`) only ever tests for equality, in Python. E.g., changing Line 4 to `if hello_world_str = 'Hello, Colab!'` would produce an error.
    - Unlike SAS, indentation is significant and used to determine scope in Python. E.g., unindenting Line 5 would produce an error since the `if` statement would no longer have a body.
    - Unlike SAS, single and double quotes always have identical behavior in Python. E.g., `'Hello, Colab!'` is treated exactly the same as `"Hello, Colab!"`.

Example 1.3. Python lists and indexing

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
hello_world_list = ['Hello', 'list']
print(hello_world_list)
print()
print(type(hello_world_list))
```

```
    ['Hello', 'list']

    <class 'list'>
```

Line-by-Line Code Explanation:

- Line 1: Create a list object named `hello_world_list`, which contains two strings.
- Lines 2-4: Print the contents of `hello_world_list`, followed by a blank line and its type.

Exercise 1.3.1. Would the Python statement `print(hello_world_list[1])` display the value `'Hello'` or `'list'`?

The value `'list'` would be displayed.

Exercise 1.3.2. True or False: A Python list may only contain values of the same type.

False. A list may contain values of different types.

Notes about Example 1.3.

1. Values in lists are always kept in insertion order, meaning the order they appear in the list's definition, and they can be individually accessed using numerical indexes within bracket notation:
    - `hello_world_list[0]` returns `'Hello'`
    - `hello_world_list[1]` returns `'list'`.
2. The left-most element of a list is always at index `0`. Unlike SAS, customized indexing is only available for more sophisticated data structures in Python (e.g., a dictionary, as in Example 1.4 below).
3. Lists are the most fundamental Python data structure and are related to SAS data-step arrays. However, unlike a SAS data-step array, a Python list object may contain values with different types, such as `str` and `int`. (Processing the values of a list without checking their types, though, may cause errors if the list contains unexpected values.)

Example 1.4. Python dictionaries

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
hello_world_dict = {
        'salutation'      : ['Hello'       , 'dict'],
        'valediction'     : ['Goodbye'     , 'list'],
        'part of speech'  : ['interjection', 'noun'],
}
print(hello_world_dict)
print()
print(type(hello_world_dict))
```

```
{'salutation': ['Hello', 'dict'], 'valediction': ['Goodbye', 'list'], 'part of

<class 'dict'>
```

Line-by-Line Code Explanation:

- Lines 1-5 : Create a dictionary object (`dict` for short) named `hello_world_dict`, which contains three key-value pairs, where each key is a string and each value is a list of two strings.
- Lines 6-8: Print the contents of `hello_world_dict`, followed by a blank line and its type.

Exercise 1.4.1. What would be displayed by executing the statement `print(hello_world_dict['salutation'])`?

The value `['Hello', 'dict']` would be displayed.

Exercise 1.4.2. Write a single line of Python code to print the initial element of the list associated with the key `valediction`.

```
print(hello_world_dict['valediction'][0])
```

```
    Goodbye
```

Notes about Example 1.4:

1. Dictionaries are another fundamental Python data structure, which map keys (appearing before the colons in Lines 2-4) to values (appearing after the colons in Lines 2-4). The value associated with each key can be accessed using bracket notation:
    - `hello_world_dict['salutation']` returns `['Hello', 'dict']`
    - `hello_world_dict['valediction']` returns `['Goodbye', 'list']`
    - `hello_world_dict['part of speech']` returns `['interjection', 'noun']`
2. Whenever indexable data structures are nested in Python, indexing methods can be combined. E.g., `hello_world_dict['salutation'][0] == ['Hello', 'dict'][0] == 'Hello'`.
3. Dictionaries are more generally called *associative arrays* or *maps* and are related to SAS formats and data-step hash tables.

Example 1.5. Introduction to DataFrames

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
from pandas import DataFrame
hello_world_df = DataFrame(
    {
        'salutation'      : ['Hello'      , 'DataFrame'],
        'valediction'     : ['Goodbye'    , 'dict'],
        'part of speech'  : ['exclamation', 'noun'],
    }
)
print(hello_world_df)
print()
print(hello_world_df.shape)
print()
hello_world_df.info()
```

```
  salutation valediction part of speech
0      Hello     Goodbye    exclamation
1  DataFrame        dict           noun

(2, 3)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   salutation      2 non-null      object
 1   valediction     2 non-null      object
 2   part of speech  2 non-null      object
dtypes: object(3)
memory usage: 176.0+ bytes
```

Line-by-Line Code Explanation:

- Line 1: Load the definition of a `DataFrame` object from the `pandas` module. (Think of a DataFrame as a rectangular array of values, like a SAS dataset, with all values in a column having the same type.)
- Lines 2-8: Create a DataFrame object (`df` for short) named `hello_world_df` with dimensions 2x3 (2 rows by 3 columns), with each key-value pair in the dictionary in Lines 3-7 becoming a column that is labelled by its key.
- Lines 9-14: Print the contents of `hello_world_df`, following by the number of rows and columns it has, and then some additional information about it.

Exercise 1.5.1. Write a single line of Python code to print the column labelled by `salutation`.

```
print(hello_world_df['salutation'])
```

```
    0        Hello
    1    DataFrame
    Name: salutation, dtype: object
```

Exercise 1.5.2. Write a single line of Python code to print the final element of the column labeled by `valediction`.

```
print(hello_world_df['valediction'][1])
```

```
    dict
```

Notes About Example 1.5:

1. The `DataFrame` object type is not built into Python, which is why we first have to import its definition from the `pandas` module.
2. The columns in a DataFrames can be indexed like the keys in a dictionary. E.g., `hello_world_df['salutation'][0] == ['Hello', 'dict'][0] == 'Hello'`
3. A DataFrame is a tabular data structure with rows and columns, similar to a SAS data set. However, while SAS datasets are typically accessed from disk and processed row-by-row, DataFrames are loaded into memory all at once. This means values in DataFrames can be randomly accessed, but it also means the size of DataFrames can't grow beyond available memory.
4. The dimensions of the DataFrame are determined as follows:
    - The keys `'salutation'`, `'valediction'`, and `'part of speech'` of the dictionary passed to the `DataFrame` constructor function become column labels.
    - Because each key maps to a list of length two, each column will be two elements tall (with an error occurring if the lists aren't all the same length).
5. The `DataFrame` constructor function can also accept many other object types, including another `DataFrame`. Please see pandas documentation for more information.

Section 2. SASPy Data Round Trip

Example 2.1. Load a SAS dataset into a pandas DataFrame

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
fish_df_smelt_only = sas.sasdata2dataframe(
    table='fish',
    libref='sashelp',
    dsopts={
        'where' : ' Species = "Smelt" ',
        'obs'   : 10,
    },
)
print(type(fish_df_smelt_only))
print()
print(fish_df_smelt_only.head())
```

```
<class 'pandas.core.frame.DataFrame'>

  Species  Weight  Length1  Length2  Length3  Height   Width
0   Smelt     6.7      9.3      9.8     10.8  1.7388  1.0476
1   Smelt     7.5     10.0     10.5     11.6  1.9720  1.1600
2   Smelt     7.0     10.1     10.6     11.6  1.7284  1.1484
3   Smelt     9.7     10.4     11.0     12.0  2.1960  1.3800
4   Smelt     9.8     10.7     11.2     12.4  2.0832  1.2772
```

Line-by-Line Code Explanation:

- Lines 1-8: Create a DataFrame object named `fish_df_smelt_only` with dimensions 10x7 (10 rows and 7 columns) from the SAS dataset sashelp.fish by subsetting to rows where the column `Species` has the value `Smelt`.
- Line 9: Print the type of the object `fish_df_smelt_only`.
- Line 10: Print a blank line.
- Line 11: Print the first 5 rows of `fish_df_smelt_only`.

Exercise 2.1.1. By default, the `head` method returns the first ________ rows in a dataset.

By default, the `head` method returns the first five rows in a dataset.

Exercise 2.1.2. True or False: The `head` method (without an argument) always returns the same number of rows in a dataset.

False. If there are fewer than five rows in a dataset, the `head` method may not return as many rows as expected.

Exercise 2.1.3. Write several lines of Python code to create a DataFrame object from the SAS dataset sashelp.fish, but limiting the rows using a different value for `species`. (Hint: If Lines 4-7 in the Example were commented out or removed, you'd be able to view a different part of the dataset.)

```
fish_df_bream_only = sas.sasdata2dataframe(
    table='fish',
    libref='sashelp',
    dsopts={
      'where' : ' Species = "Bream" ',
      'obs'   : 10,
    },
)
print(type(fish_df_bream_only))
print()
print(fish_df_bream_only.head())
```

```
<class 'pandas.core.frame.DataFrame'>

  Species  Weight  Length1  Length2  Length3   Height   Width
0   Bream   242.0     23.2     25.4     30.0  11.5200  4.0200
1   Bream   290.0     24.0     26.3     31.2  12.4800  4.3056
2   Bream   340.0     23.9     26.5     31.1  12.3778  4.6961
3   Bream   363.0     26.3     29.0     33.5  12.7300  4.4555
4   Bream   430.0     26.5     29.0     34.0  12.4440  5.1340
```

Notes About Example 2.1:

1. `sasdata2dataframe` is a method of a `SASsession` object, which allows the contents of a SAS dataset (meaning a physical file living on disk) to be copied into memory as a DataFrame object. The resulting DataFrame has rows labelled by non-negative integers: The first row is labelled as 0, the second as 1, and so on, just like elements in a list. However, as we'll see below, the row labels can also be given by an index column.
2. The `dsopts` argument for `sasdata2dataframe` allows dataset options to be passed to SAS, which subset the dataset before it's converted to a DataFrame. In the above example, we've only used the dataset options `where` and `obs`, but it's also possible to pass through additional options like `keep` and `drop`, which accept lists of columns. Notice that each option is specified as a key-value pair inside a dictionary.
3. When used without an argument, the `head` method returns the first 5 rows of a DataFrame (or the entire DataFrame, if there are 5 or fewer rows). We can also control the number of rows; e.g., `fish_df_smelt_only.head(3)` returns the first 3 rows.
4. A parallel method called `tail` can also be used to return the last few rows in a DataFrame.
5. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

Example 2.2. Manipulate a DataFrame

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
fish_df = sas.sasdata2dataframe(table='fish',libref='sashelp')
fish_df_g   = fish_df.groupby('Species')
fish_df_gs  = fish_df_g['Weight']
fish_df_gsa = fish_df_gs.agg(['count', 'std', 'mean', 'min', 'max'])
print(fish_df_gsa)
```

```
           count         std        mean    min     max
Species
Bream         34  206.604585  626.000000  242.0  1000.0
Parkki        11   78.755086  154.818182   55.0   300.0
Perch         56  347.617717  382.239286    5.9  1100.0
Pike          17  494.140765  718.705882  200.0  1650.0
Roach         20   88.828916  152.050000    0.0   390.0
Smelt         14    4.131526   11.178571    6.7    19.9
Whitefish      6  309.602972  531.000000  270.0  1000.0
```

Line-by-Line Code Explanation:

- Line 1: Create a DataFrame object named `fish_df` with dimensions 159x7, comprising all 159 rows and 7 columns of the SAS dataset sashelp.fish.
- Line 2: Group the rows of `fish_df` by the values in column `Species`. (This can be thought of as follows: For each possible value of `Species`, create a DataFrame having just the corresponding rows.)
- Line 3: Subset to just the column `Weight` in each grouping.
- Line 4: Apply aggregation functions for counting number of records, standard deviation, mean, minimum, and maximum to the values of `Weight` that have been grouped by values of `Species`.
- Line 5: Print the results of the aggregations, which uses `Species` as a row index.

Exercise 2.2.1. The DataFrame `groupby` method is like a ____________________ statement in a PROC MEANS step in SAS.

The DataFrame `groupby` method is like a CLASS statement in a PROC MEANS step in SAS.

Exercise 2.2.2. After a `groupby` method has been used on a DataFrame, subsetting to a specific column is like a ____________________ statement in a PROC MEANS step in SAS.

After a `groupby` method has been used on a DataFrame, subsetting to a specific column is like a VAR statement in a PROC MEANS step in SAS.

Exercise 2.2.3. Write several lines of Python code to create a DataFrame object from the SAS dataset sashelp.class, and then imitate a PROC MEANS step to get the median of `Height` when grouped by `Sex`.

```
class_df = sas.sasdata2dataframe(table='class',libref='sashelp')
class_df_g   = class_df.groupby('Sex')
class_df_gs  = class_df_g['Height']
class_df_gsa = class_df_gs.agg(['count', 'std', 'mean', 'min', 'max'])
print(class_df_gsa)
```

```
         count       std       mean   min   max
    Sex
    F        9  5.018328  60.588889  51.3  66.5
    M       10  4.937937  63.910000  57.3  72.0
```

Notes about Example 2.2:

1. When the `sasdata2dataframe` method is used without an `dsopts` argument, the entire SAS dataset is copied into memory as a DataFrame.
2. In the output, notice that the left-most column `Species` is actually an index column, which is a byproduct of using the `groupby` method. In other words, we can think of the rows of `fish_df_gsa` as being labelled by values of `Species`. This is different from the output in Example 2.1, where the rows of `fish_df_smelt_only` were labelled by non-negative integers since `fish_df_smelt_only` doesn't have an index column.
3. The SAS equivalent of this example is as follows:

   ```
   proc means data=sashelp.fish std mean min max;
       class species;
       var Weight;
   run;
   ```

   However, while PROC MEANS operates on SAS datasets row-by-row from disk, DataFrames are stored entirely in main memory. This allows any number of DataFrame operations to be combined for on-the-fly reshaping using "method chaining." In other words, `fish_df_gsa` could have instead been created with the following one-liner, which avoids the need for intermediate DataFrames (and thus executes much more quickly):

   ```
    fish_df_gsa = fish_df.groupby('Species')['Weight'].agg(['count', 'std', '
   mean', 'min', 'max'])
   ```

4. Example 2.3 below assumes `fish_df_gsa` exists.
5. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

Example 2.3. Load a DataFrame into a SAS dataset

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
from IPython.display import display, HTML

sas.dataframe2sasdata(
    fish_df_gsa.reset_index(),
    table="fish_sds_gsa",
    libref="Work"
)
sas_submit_return_value = sas.submit(
    '''
        PROC PRINT DATA=fish_sds_gsa;
        RUN;
    '''
)
sas_submit_log = sas_submit_return_value['LOG']
print(sas_submit_log)
sas_submit_results = sas_submit_return_value['LST']
display(HTML(sas_submit_results))
```

```
288        ods listing close;ods html5 (id=saspy_internal) file=_tomods1 optic
288      ! ods graphics on / outputfmt=png;
289
290
291                PROC PRINT DATA=fish_sds_gsa;
292                RUN;
293
294
295
296        ods html5 (id=saspy_internal) close;ods listing;
297
```

```
298
```

The SAS System

Obs	Species	count	std	mean	min	max
1	Bream	34	206.605	626.000	242.0	1000.0
2	Parkki	11	78.755	154.818	55.0	300.0
3	Perch	56	347.618	382.239	5.9	1100.0
4	Pike	17	494.141	718.706	200.0	1650.0
5	Roach	20	88.829	152.050	0.0	390.0
6	Smelt	14	4.132	11.179	6.7	19.9
7	Whitefish	6	309.603	531.000	270.0	1000.0

Line-by-Line Code Explanation:

- Line 1: Load the `display` and `HTML` functions from the `IPython` package.
- Lines 3-7: Convert the pandas DataFrame `fish_df_gsa` (a memory-resident rectangular array of values created in Example 2.2 above) into a SAS dataset (a physical file on disk), and store the result in the Work library (a physical location on disk in the remote SAS OnDemand for Academics session). Only columns are transferred, not indexes, which is why the `reset_index` method is used to convert `Species` values to a column.
- Lines 8-13: Apply the SAS PRINT procedure to SAS dataset `fish_df_gsa`. (Note: In Lines 3-6, triple quote marks are used to create a single string object with embedded line breaks.)
- Lines 14-15: Extract the SAS log from the dictionary returned by the `submit` method, and display it using the `print` function.
- Lines 16-17: Extract the SAS output from the dictionary returned by the `submit` method, and display it using the `display` and `HTML` functions.

Exercise 2.3.1. True or False: If `reset_index` were not used in example 2.3, information could be lost when the `dataframe2sasdata` method is used to transform a DataFrame into a SAS dataset.

True, `dataframe2sasdata` only transfers regular DataFrame columns, not indexes.

Exercise 2.3.2. True or False: The `submit` method of a `SASsession` object allows arbitrary SAS code to be submitted directly to the SAS kernel.

True. The `submit` method submits the value of any string to the SAS kernel for execution.

Exercise 2.3.3. Write several lines of Python code to convert the DataFrame `fish_df` created in Example 2.2 to a SAS dataset, and then use the SAS CONTENTS procedure directly on the result.

```
sas.dataframe2sasdata(
    fish_df,
    table="fish_sds",
    libref="Work"
)
sas_submit_return_value = sas.submit(
    '''
```

```
        PROC CONTENTS DATA=fish_sds;
        RUN;
    '''
)
sas_submit_log = sas_submit_return_value['LOG']
print(sas_submit_log)
sas_submit_results = sas_submit_return_value['LST']
display(HTML(sas_submit_results))
```

```
83                                                      The SAS System

709        ods listing close;ods html5 (id=saspy_internal) file=_tomods1 opt
709      ! ods graphics on / outputfmt=png;
710
711
712                PROC CONTENTS DATA=fish_sds;
713                RUN;
714
715
716
717        ods html5 (id=saspy_internal) close;ods listing;
718

84                                                      The SAS System

719
```

The SAS System

The CONTENTS Procedure

Data Set Name	WORK.FISH_SDS	**Observations**	159
Member Type	DATA	**Variables**	7
Engine	V9	**Indexes**	0
Created	11/25/2021 01:01:01	**Observation Length**	64
Last Modified	11/25/2021 01:01:01	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	utf-8 Unicode (UTF-8)		

Engine/Host Dependent Information	
Data Set Page Size	131072

Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	2043
Obs in First Data Page	159
Number of Data Set Repairs	0
Filename	/saswork/SAS_work7C7C0000B0E0_odaws01-usw2.oda.sas.com/SAS_work27450000B0E0_odaws01-usw2.oda.sas.com/fish_sds.sas7bdat
Release Created	9.0401M6
Host Created	Linux
Inode Number	1610731589
Access Permission	rw-r--r--
Owner Name	matthew.t.slaugh
File Size	256KB
File Size (bytes)	262144

Alphabetic List of Variables and Attributes

Notes about Example 2.3:

1. If `reset_index` is not used when exporting to SAS, the index column `Species` will be lost. Properties of a DataFrame without a SAS equivalences are not preserved when the method `dataframe2sasdata` is used to convert a pandas DataFrame to a SAS dataset.
2. Python strings can be defined with one of four quoting conventions:

   - single quotes, as in `'Hello, World!'`
   - double quotes, as in `"Hello, World!"`
   - triple quotes, as in `'''Hello, World!'''` or `"""Hello, World!"""`

   All four styles are interchangeable for single-line strings. However, unlike single- and double-quoted strings, triple-quoted strings can contain embedded line breaks.
3. The `submit` method can be used to pass arbitrary SAS code directly to a SAS kernel. After the SAS kernel executes the code, a dictionary is returned with the following two key-value pairs:

   - `sas_submit_return_value['LST']` is a string comprising the results of executing the PROC PRINT step. Because SAS returns HTML by default, the `HTML` function needs to be used to render the results, and the `display` function needs to be used to display the rendered HTML. (You could also use `print(sas_submit_return_value['LST'])` to view the raw, underlying HTML.)
   - `sas_submit_return_value['LOG']` is a string comprising the plain-text log resulting from executing the PROC PRINT step, which can be displayed with the `print` function.
4. Alternatively, adding the argument `results='TEXT'` to the `submit` method would replace the HTML output with plain-text viewable using the `print` function.
5. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

Section 3. Executing SAS Procedures with Convenience Methods

Example 3.1. Connect directly to a SAS dataset

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
from IPython.display import display

fish_sds = sas.sasdata(table='fish', libref='sashelp')
print(type(fish_sds))
print()
display(fish_sds.columnInfo())
display(fish_sds.describe())
```

```
<class 'saspy.sasdata.SASdata'>
```

	Member	Num	Variable	Type	Len	Pos
0	SASHELP.FISH	6.0	Height	Num	8.0	32.0
1	SASHELP.FISH	3.0	Length1	Num	8.0	8.0
2	SASHELP.FISH	4.0	Length2	Num	8.0	16.0
3	SASHELP.FISH	5.0	Length3	Num	8.0	24.0
4	SASHELP.FISH	1.0	Species	Char	9.0	48.0
5	SASHELP.FISH	2.0	Weight	Num	8.0	0.0
6	SASHELP.FISH	7.0	Width	Num	8.0	40.0

	Variable	N	NMiss	Median	Mean	StdDev	Min	P25	P50
0	Weight	158.0	1.0	272.5000	398.695570	359.086204	0.0000	120.0000	272.5000
1	Length1	159.0	0.0	25.2000	26.247170	9.996441	7.5000	19.0000	25.2000
2	Length2	159.0	0.0	27.3000	28.415723	10.716328	8.4000	21.0000	27.3000
3	Length3	159.0	0.0	29.4000	31.227044	11.610246	8.8000	23.1000	29.4000
4	Height	159.0	0.0	7.7860	8.970994	4.286208	1.7284	5.9364	7.7860
5	Width	159.0	0.0	4.2485	4.417486	1.685804	1.0476	3.3756	4.2485

Line-by-Line Code Explanation:

- Line 1: Load the `display` function from the `IPython` package.
- Lines 3-4: Create a file pointer to the SAS dataset sashelp.fish (a physical file on disk) and print its type.
- Line 6: Use the `columnInfo` method to view metadata about the variables in sashelp.fish, and use the `display` function to render it as HTML output.
- Lines 7: Use the `describe` method to view summary statistics for the numeric variables in the sashelp.fish, and use the `display` function to render it as HTML output.

Exercise 3.1.1. True or False: The `sasdata` method imports a SAS dataset and returns a Python DataFrame.

False. the `sasdata` method creates a pointer to a SAS dataset, and by itself does not import or export data.

Exercise 3.1.2. Can you guess which SAS procedures are invoked by the `columnInfo` and `describe` methods?

The `columnInfo` method invokes PROC CONTENTS, while `describe` invokes PROC MEANS.

Notes About Example 3.1:

1. The `sasdata` method creates a file pointer, meaning a direct connection to a disk-based SAS dataset, whereas the `sasdata2dataframe` method used in Section 2 examples loads a SAS dataset into memory as a pandas DataFrame.
2. `columnInfo` and `describe` are examples of "convenience methods" that implicitly invoke SAS's CONTENTS procedure and MEANS procedure, respectively.
3. Additional convenience methods are listed in the SASPy documentation at https://sassoftware.github.io/saspy/api.html#sas-data-object.
4. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

Example 3.2 Display generated SAS code

Instructions: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
sas.teach_me_SAS(True)
fish_sds.describe()
sas.teach_me_SAS(False)
```

```
    proc means data=sashelp.'fish'n  stackodsoutput n nmiss median mean std min p2
```

Line-by-Line Code Explanation:

- Line 1: Set `teach_me_SAS` to `True`. (This is a global effect that will apply to all `saspy` method calls submitted from this point forward.)
- Line 2: Invoke the `describe` method. (Because `teach_me_SAS` is set to `True`, the SAS code generated by the `describe` method is displayed, but not executed.)
- Line 3: Set `teach_me_SAS` to `False`, reversing the effect in Line 1.

Exercise 3.2.1. Imagine `teach_me_SAS` had been set to True when the `columnInfo` method was called in Example 3.1. What SAS code might have been displayed instead?

```
proc contents data=sashelp.fish;
run;
```

Exercise 3.2.2. Write several lines of Python code to execute your SAS code from Exercise 3.2.1 and display the results.

```
sas_submit_return_value = sas.submit(
    '''
        PROC CONTENTS DATA=sashelp.fish;
        RUN;
    '''
)
sas_submit_results = sas_submit_return_value['LST']
```

```
display(HTML(sas_submit_results))
```

The SAS System

The CONTENTS Procedure

Data Set Name	SASHELP.FISH	**Observations**	159
Member Type	DATA	**Variables**	7
Engine	V9	**Indexes**	0
Created	04/25/2019 07:10:25	**Observation Length**	64
Last Modified	04/25/2019 07:10:25	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label	Measurements of 159 Fish Caught in Lake Laengelmavesi, Finland		
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	us-ascii ASCII (ANSI)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	1021
Obs in First Data Page	159
Number of Data Set Repairs	0
Filename	/pbr/sfw/sas/940/SASFoundation/9.4/sashelp/fish.sas7bdat
Release Created	9.0401M6
Host Created	Linux
Inode Number	6430737
Access Permission	rw-r--r--
Owner Name	odaowner
File Size	128KB
File Size (bytes)	131072

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
6	Height	Num	8
3	Length1	Num	8

4	Length2	Num	8
5	Length3	Num	8
1	Species	Char	9
2	Weight	Num	8

Notes About Example 3.2:

1. Lines 1 and 3 can be thought of as a "Teach Me SAS" sandwich, similar to how an "ODS Sandwich" can be used to toggle output to a specific destination.
2. `True` and `False` are standard Python objects. Like their SAS equivalents, they are interchangeable with the values `1` and `0`, respectively. They are also case sensitive; e.g., `False` is not the same as `false`.
3. The `teach_me_SAS` method allows us to extract (and modify) the SAS code generated by convenience methods. For example, the `describe` convenience method doesn't allow us to set classification variables for PROC MEANS. However, we can use `teach_me_SAS` to generate the underlying SAS code, add a CLASS statement, and then execute the modified SAS code using the `submit` method (see Example 2.3).
4. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

Example 3.3 Don't worry! You have options...

__Instructions__: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercises that follow, only looking at the explanatory notes for hints when needed.

```
from IPython.display import display

class_input_dsopts = {
    'where' : ' Age > 13',
    'keep'  : ['Name','Age','Height','Weight'],
}
class_sds = sas.sasdata(
    table='class',
    libref='sashelp',
    dsopts=class_input_dsopts
)
```

```
class_output_dsopts = {'keep' : ['Name','bmi']}
class_bmi_sds = sas.sasdata(
    table='class_bmi',
    libref='work',
    dsopts=class_output_dsopts
)
class_sds.add_vars(vars = {'bmi':'(Weight/Height**2)*703'}, out = class_bmi_sds)

display(class_bmi_sds.head())
display(class_bmi_sds.means())
```

```
Table work.class_bmi does not exist. This SASdata object will not be useful ur

88                                                          The SAS System

559
560          data work.'class_bmi'n (keep=Name bmi ); set sashelp.'class'n (wher
561          bmi = (Weight/Height**2)*703;
562          ; run;
563
564
565

89                                                          The SAS System

566
```

	Name	bmi
0	Alfred	16.611531
1	Carol	18.270898
2	Henry	17.870296
3	Janet	20.246400
4	Judy	15.302976

	Variable	N	NMiss	Median	Mean	StdDev	Min	P25	P5
0	bmi	9.0	0.0	17.870296	18.342336	1.831753	15.302976	17.804511	17.87029

Line-by-Line Code Explanation:

- Line 1: Load the `display` function from the `IPython` package.
- Lines 3-6: Create a dictionary named `class_input_dsopts` with two keys. Each key-value pair specifies a dataset option.
- Lines 7-11: Create a file pointer to the SAS dataset sashelp.class on disk, specifying dataset options to be used whenever the dataset is read.
- Line 13: Create a dictionary named `class_ouptut_dsopts`, which specifies the `keep` dataset option.
- Lines 14-18: Create a file pointer to a SAS dataset that does not exist (yet), name the SAS dataset work.class_bmi, and specify that the dataset option in Line 13 should be used when the dataset is created.
- Line 19: Starting with the SAS dataset represented by class_sds (i.e., sashelp.class), calculate the new variable BMI, and store the resulting dataset in the SAS dataset represented by class_bmi_sds (i.e., work.class_bmi). (Note: The `add_vars` method also prints the log of the corresponding SAS DATA step, by default.)
- Lines 21-22: Display the first 5 rows of the result, along with some summary statistics.

Exercise 3.3.1. True or False: In this example, specifying `dsopts=class_input_dsopts` on Line 10 modifies the underlying data on disk.

False. These options only alter how the dataset will be processed when read.

Exercise 3.3.2. True or False: In this example, specifying `dsopts=class_output_dsopts` on Line 17 modifies the underlying data on disk.

True. These options affect the contents of the dataset written to disk by the `add_vars` method on line 19.

Exercise 3.3.2. Copy the code from Example 3.3, paste it below, and create one or more additional variables in the SAS dataset work.class_bmi by adding additional key-value pairs to the dictionary used for the `vars=` parameter of the `add_vars` method.

```
class_input_dsopts = {
    'where' : ' Age > 13',
    'keep'  : ['Name','Age','Height','Weight'],
}
class_sds = sas.sasdata(
    table='class',
    libref='sashelp',
    dsopts=class_input_dsopts
)

class_output_dsopts = {'keep' : ['Name','first_initial','bmi']}
class_bmi_sds = sas.sasdata(
    table='class_bmi',
    libref='work',
    dsopts=class_output_dsopts
)
class_sds.add_vars(
    vars = {
        'bmi':'(Weight/Height**2)*703',
        'first_initial':'substr(Name,1,1)'
    },
    out = class_bmi_sds
)

display(class_bmi_sds.head())
display(class_bmi_sds.means())
```

```
128                                                         The SAS System

801
802         data work.'class_bmi'n (keep=Name first_initial bmi ); set sashelp.
802       ! Weight );
803         bmi = (Weight/Height**2)*703;
804         first_initial = substr(Name,1,1);
805         ; run;
806
807
808

129                                                         The SAS System

809
```

	Name	bmi	first_initial
0	Alfred	16.611531	A
1	Carol	18.270898	C
2	Henry	17.870296	H
3	Janet	20.246400	J
4	Judy	15.302976	J

	Variable	N	NMiss	Median	Mean	StdDev	Min	P25	P50
0	bmi	9.0	0.0	17.870296	18.342336	1.831753	15.302976	17.804511	17.870296

Notes About Example 3.3:

1. The `sasdata` method can be used to create a pointer to a dataset that does not exist. This can still be useful if you intend to create the dataset later.
2. The `add_vars` method uses a SAS DATA step to assign values to new variables. If no `out=` argument is specified, the dataset will be modified in place.
3. Just like the `sasdata2dataframe` method used in Section 2 examples, the `sasdata` has a `dsopts` argument allowing dataset options to be passed to SAS. However, because `sasdata` only creates a file pointer, the dataset options only affect the values returned when the SAS dataset is read from disk. The underlying SAS dataset itself will not be changed unless `dsopts` is specified for an output dataset.
4. Just like its pandas counterpart, the `head` method returns the first 5 rows of a `sasdata` object (or the corresponding dataset in its entirety, if there are 5 or fewer observations). We can also control the number of rows; e.g., `class_bmi_sds.head(3)` returns the first 3 rows.
5. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

Section 4. Staying D.R.Y.

Example 4.1. Imitate the SAS Macro Processor

<u>**Instructions**</u>: Click anywhere in the code cell immediately below, and run the cell using Shift-Enter. Then attempt the Exercise that follows, only looking at the explanatory notes for hints when needed.

```
from IPython.display import display, HTML

sas_code_fragment = 'proc means data=sashelp.{data}; run;'
for dsn in ['fish', 'class', 'iris']:
    sas_submit_return_value = sas.submit(
        sas_code_fragment.format(data=dsn)
    )
    print(sas_submit_return_value['LOG'])
    display(HTML(sas_submit_return_value['LST']))
```

```
158                                                        The SAS System

989         ods listing close;ods html5 (id=saspy_internal) file=_tomods1 opt
989       ! ods graphics on / outputfmt=png;
990
991         proc means data=sashelp.fish; run;
992
993
994         ods html5 (id=saspy_internal) close;ods listing;
995

159                                                        The SAS System

996
```

The SAS System

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
Weight	158	398.6955696	359.0862037	0	1650.00
Length1	159	26.2471698	9.9964412	7.5000000	59.0000000
Length2	159	28.4157233	10.7163281	8.4000000	63.4000000
Length3	159	31.2270440	11.6102458	8.8000000	68.0000000
Height	159	8.9709937	4.2862076	1.7284000	18.9570000
Width	159	4.4174855	1.6858039	1.0476000	8.1420000

```
160                                                        The SAS System

999         ods listing close;ods html5 (id=saspy_internal) file=_tomods1 opt
999       ! ods graphics on / outputfmt=png;
1000
1001        proc means data=sashelp.class; run;
1002
1003
1004        ods html5 (id=saspy_internal) close;ods listing;
1005

161                                                        The SAS System

1006
```

The SAS System

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
Age	19	13.3157895	1.4926722	11.0000000	16.0000000
Height	19	62.3368421	5.1270752	51.3000000	72.0000000
Weight	19	100.0263158	22.7739335	50.5000000	150.0000000

```
    162                                                     The SAS System

    1009          ods listing close;ods html5 (id=saspy_internal) file=_tomods1 opt
    1009        ! ods graphics on / outputfmt=png;
```

Line-by-Line Code Explanation:

- Line 1: Load the `display` and `HTML` functions from the `IPython` package.
- Line 3: Create a string object named `sas_code_fragment` with a templating placeholder `{data}` in curly brackets. The portion in brackets will be replaced with other strings in subsequent uses of `sas_code_fragment`.
- Line 4: Initiate a for-loop over the three values in the list `['fish', 'class','iris']`. (In other words, the body of the for-loop, meaning all subsequent lines that are indented, will be executed three time. The first time, the value of the index variable `dsn` will be `'fish'`, the second time `dsn` will be `'class'`, and the third time `dsn` will be `'iris'`)
- Lines 5-9: Define the body of the for-loop to use the `submit` method on `sas_code_fragment`. The `format` method is used to replace the placeholder `{data}` with the current value of the index varaible `dsn`. SAS logs and output are also printed on each iteration of the loop, per Lines 8-9.

Exercise 4.1.1. Write several lines of Python code to accomplish the following: Using the above example as a model, print out the results of applying the SAS CONTENTS procedure to the SAS datasets sashelp.steel and sashelp.tourism, and output the results.

```
sas_code_fragment = 'proc contents data=sashelp.{data}; run;'
for dsn in ['steel','tourism']:
    sas_submit_return_value = sas.submit(
        sas_code_fragment.format(data=dsn)
    )
    print(sas_submit_return_value['LOG'])
    display(HTML(sas_submit_return_value['LST']))
```

```
    243                                                     The SAS System

    1686          ods listing close;ods html5 (id=saspy_internal) file=_tomods1 opt
    1686        ! ods graphics on / outputfmt=png;
    1687
    1688          proc contents data=sashelp.steel; run;
    1689
    1690
    1691          ods html5 (id=saspy_internal) close;ods listing;
    1692
```

```
244                                                       The SAS System

1693
```

The SAS System

The CONTENTS Procedure

Data Set Name	SASHELP.STEEL	**Observations**	44
Member Type	DATA	**Variables**	2
Engine	V9	**Indexes**	0
Created	10/25/2018 02:15:47	**Observation Length**	16
Last Modified	10/25/2018 02:15:47	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label	iron/steel exports (yearly: 1937-1980)		
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	us-ascii ASCII (ANSI)		

Engine/Host Dependent Information	
Data Set Page Size	65536
Number of Data Set Pages	1
First Data Page	1
Max Obs per Page	4061
Obs in First Data Page	44
Number of Data Set Repairs	0
Filename	/pbr/sfw/sas/940/SASFoundation/9.4/sashelp/steel.sas7bdat
Release Created	9.0401M6
Host Created	Linux
Inode Number	6430333
Access Permission	rw-r--r--
Owner Name	odaowner
File Size	128KB
File Size (bytes)	131072

Notes About Example 4.1.

1. The end result of this Example is to construct and submit the following SAS code to the SAS kernel:

```
proc means data=sashelp.fish; run;
proc means data=sashelp.class; run;
proc means data=sashelp.iris; run;
```

While it may have been fewer keystrokes to submit this directly, the Python code illustrates the general software engineering principle "Don't Repeat Yourself" (aka D.R.Y.). Think about how much easier it would be to extend the Example to a list of one hundred datasets --- and how much less error prone it would be.

2. The same outcome could also have been achieved with the following SAS macro code:

   ```
   %macro loop(dsn_list);
       %let list_length = %sysfunc(countw(&dsn_list));
       %do i = 1 %to &list_length;
             %let dsn = %scan(&dsn_list.,&i.);
             proc means data=sashelp.&dsn.;
             run;
       %end;
   %mend;
   %loop(fish class iris)
   ```

   However, note the following differences:

   - Python allows us to concisely repeat an arbitrary block of code by iterating over a list using a for-loop. In other words, the body of the for-loop (meaning everything indented underneath it, since Python uses indentation to determine scope) is repeated for each string in the list `['fish','class','iris']`.
   - The SAS macro facility only provides do-loops based on numerical index variables (the macro variable `&i.` above), so clever tricks like implicitly defined arrays (macro parameter `dsn_list` above) need to be used together with functions like `%SCAN` to extract a sequence of values. A combination of the macro function `%SYSFUNC` and the data-step function `countw` also need to be used to determine the "length" of `dsn_list`.

3. The `sas` object represents a connection to a SAS session, and was created in section 0 above.

Release Created	[illegible]
Host Created	Linux

Wrapping Up: Call to Action!

Owner Name	odaowner

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

1. Continue learning Python.
    - For general programming, we recommend starting with these:
        - Automate the Boring Stuff with Python, a free online book with numerous beginner-friendly hands-on projects
        - Fluent Python, which provided a deep dive into Intermediate to Advanced Python concepts
    - For data science, we recommend starting with these:
        - A Whirlwind Tour of Python, a free online book with coverage of essential Python features commonly used in data science projects
        - Python for Data Analysis, which provided a deep dive into the `pandas` package by its creator, Wes McKinney
    - For web development in Python, we recommend starting with this:
        - The Flask Mega-Tutorial, a freely accessible series of blog posts covering essential features of developing dynamic websites with the `flask` web framework
2. Consider taking our Beyond the Basics class on 06DEC2021.
3. Try using Python outside of Google Colab. For example, if you're interested in setting up a local SASPy enviroment in order to have Python talk to a commercial SAS installation, you're welcome to follow setup instructions (see page 2) from a previous iteration of this course.
4. Keep in touch for follow-up questions/discussion (one of our favorite parts of teaching!) using isaiah.lankham@gmail.com and matthew.t.slaughter@gmail.com
5. 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