Everything is better with friends: Executing SAS code in Python scripts with SASPy

Hands-on Workshop • Western Users of SAS Software (WUSS) 2019

Example 0. [Python] Get version number (IL)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
import platform
print(platform.sys.version)
```

Notes:

1. Assuming a Python 3 kernel is associated with this Notebook, the following should be printed:
    - the Python version
    - operating-system information
2. To increase performance, only a small number of modules in Python's standard library are available by default, so the `platform` module needs to be explicitly loaded.
3. This example illustrates three ways Python syntax differs from SAS:
    - We don't need semicolons at the end of each statement. Unlike SAS, semicolons are optional in Python, and they are typically only used to separate multiple statements placed on the same line (e.g., this example could be written on one line as follows: `import platform; print(platform.sys.version)`).
    - The code `IMPORT PLATFORM` would produce an error. Unlike SAS, capitalization matters in Python.
    - The `platform` object module invokes the sub-module object `sys` nested inside of it, and `sys` invokes the object `version` nested inside of it. Unlike SAS, dot-notation has a consistent meaning in Python and can be used to reference objects nested inside each other at any depth. (Think Russian nesting dolls or turduckens.)
4. If an error is displayed, an incompatible kernel has been chosen. This Notebook was developed using the Python 3.5 kernel provided with SAS University Edition as of August 2019.

In [1]:
```
import platform
print(platform.sys.version)
```

```
3.5.5 (default, Nov 28 2018, 13:42:21)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-16)]
```

Example 1. [Python] Display available modules (IL)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
help('modules')
```

Notes:

1. All Python modules available to be loaded by the Notebook's kernel should be printed, including
   - standard library modules (e.g., `platform`, which was used above),
   - and any third-party modules that have been installed (e.g., `pandas` and `saspy`, which will be used below).
2. Python has a large standard library because of its "batteries included" philosophy. In addition, numerous third-party modules are actively developed and made freely available through sites like https://github.com/ (https://github.com/) and https://pypi.org/ (https://pypi.org/).
3. This example illustrates another way Python syntax differs from SAS:
   - `help("modules")` would produce identical output. Unlike SAS, single and double quotes always have identical behavior in Python.
4. The modules `pandas` and `saspy` will need to appear for the remaining examples in this Notebook to work, and `saspy` will need to be pre-configured to connect to a SAS kernel with access to the `sashelp` library. Depending on the versions of the modules installed, warnings or errors might also appear.

In [2]:
```
help('modules')
```

```
Please wait a moment while I gather a list of all available modules...


/usr/lib64/python3.5/site-packages/IPython/kernel/__init__.py:13: ShimWarning: The `IPython.kerne
l` package has been deprecated since IPython 4.0.You should import from ipykernel or jupyter_clie
nt instead.
```

```
  "You should import from ipykernel or jupyter_client instead.", ShimWarning)

/usr/lib64/python3.5/site-packages/sas_kernel/data

/usr/lib64/python3.5/pkgutil.py:104: VisibleDeprecationWarning: zmq.eventloop.minitornado is depr
ecated in pyzmq 14.0 and will be removed.
    Install tornado itself to use zmq with the tornado IOLoop.

  yield from walk_packages(path, name+'.', onerror)

2019-04-08%20RandomSelectionFunction atexit              jedi                 saspy
CDROM                audioop              jinja2               sched
DLFCN                autoreload           json                 select
IN                   backcall             jsonschema           selectors
IPython              backports            jupyter              send2trash
SGF2019-HOW-Everything_Is_Better_With_Friends-examples base64              jupyter_client       se
tuptools
TYPES                bdb                  jupyter_core         shelve
__future__           binascii             jupyterlab           shlex
_ast                 binhex               jupyterlab_server    shutil
_bisect              bisect               keyword              signal
_bootlocale          bleach               lib2to3              simplegeneric
_bz2                 builtins             linecache            site
_codecs              bz2                  locale               six
_codecs_cn           cProfile             logging              smtpd
_codecs_hk           calendar             lzma                 smtplib
_codecs_iso2022      cgi                  macpath              sndhdr
_codecs_jp           cgitb                macurl2path          socket
_codecs_kr           chunk                mailbox              socketserver
_codecs_tw           cmath                mailcap              spwd
_collections         cmd                  markupsafe           sqlite3
_collections_abc     code                 marshal              sre_compile
_compat_pickle       codecs               math                 sre_constants
_compression         codeop               metakernel           sre_parse
_crypt               collections          mimetypes            ssl
_csv                 colorsys             mistune              stat
_ctypes              compileall           mmap                 statistics
_ctypes_test         concurrent           modulefinder         storemagic
_datetime            configparser         multiprocessing      string
_dbm                 contextlib           nbconvert            stringprep
_decimal             copy                 nbformat             struct
```

```
_dummy_thread
_functools
_gdbm
_hashlib
_heapq
_imp
_io
_json
_locale
_lsprof
_lzma
_markupbase
_md5
_multibytecodec
_multiprocessing
_opcode
_operator
_osx_support
_pickle
_posixsubprocess
_pydecimal
_pyio
_random
_sha1
_sha256
_sha512
_signal
_sitebuiltins
_socket
_sqlite3
_sre
_ssl
_stat
_string
_strptime
_struct
_symtable
_sysconfigdata
_testbuffer
_testcapi
copyreg
crypt
csv
ctypes
curses
cythonmagic
datetime
dateutil
dbm
decimal
decorator
defusedxml
difflib
dis
distutils
doctest
dummy_threading
easy_install
email
encodings
ensurepip
entrypoints
enum
errno
faulthandler
fcntl
filecmp
fileinput
fnmatch
formatter
fractions
ftplib
functools
gc
genericpath
getopt
getpass
gettext
glob
grp
netrc
nis
nntplib
notebook
ntpath
nturl2path
numbers
numpy
opcode
operator
optparse
os
ossaudiodev
pandas
pandocfilters
parser
parso
pathlib
pdb
pexpect
pickle
pickleshare
pickletools
pip
pipes
pkg_resources
pkgutil
platform
plistlib
poplib
posix
posixpath
pprint
profile
prometheus_client
prompt_toolkit
pstats
pty
ptyprocess
pwd
subprocess
sunau
symbol
sympyprinting
symtable
sys
sysconfig
syslog
tabnanny
tarfile
telnetlib
tempfile
terminado
termios
test
testpath
textwrap
this
threading
time
timeit
tkinter
token
tokenize
tornado
trace
traceback
tracemalloc
traitlets
tty
turtle
types
typing
unicodedata
unittest
urllib
uu
uuid
venv
warnings
```

```
_testimportmultiple gzip                py_compile          wave
_testmultiphase     hashlib             pyclbr              wcwidth
_thread             heapq               pydoc               weakref
_threading_local    hmac                pydoc_data          webbrowser
_tkinter            html                pyexpat             webencodings
_tracemalloc        http                pygments            widgetsnbextension
_warnings           imaplib             pytz                wsgiref
_weakref            imghdr              queue               xdrlib
_weakrefset         imp                 quopri              xml
abc                 importlib           random              xmlrpc
aifc                inspect             re                  xxlimited
antigravity         io                  readline            xxsubtype
argparse            ipaddress           reprlib             zipapp
array               ipykernel           resource            zipfile
ast                 ipykernel_launcher  rlcompleter         zipimport
asynchat            ipython_genutils    rmagic              zlib
asyncio             ipywidgets          runpy               zmq
asyncore            itertools           sas_kernel

Enter any module name to get more help.  Or, type "modules spam" to search
for modules whose name or summary contain the string "spam".
```

Example 2. [Python] Define a `str` object (IL)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

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

Notes:

1. A string (`str` for short) object named `hello_world_str` is created, and the following are printed with a blank line between them:

   - the value of the string
   - its type (which is `<class 'str'>` , reflecting Python primarily being an object-oriented language with class-based inheritance)
2. This example illustrates three more ways Python syntax differs from SAS:
   - `hello_world_str` can be assigned a value virtually anywhere, and it could be reassigned a value later with a completely different type (e.g., `hello_world_str = 42` would could `type(hello_world_str)` to become `<class 'int'>`). Unlike SAS, variables are dynamically typed in Python.
   - The code `if hello_world_str = 'Hello, Jupyter!'` would produce an error. Unlike SAS, single-equals (=) only ever means assignment, and double-equals (==) only ever tests for equality, in Python.
   - Removing indentation would also produce errors. Unlike SAS, indentation is significant and used to determine scope in Python.
3. For extra credit, try any or all of the following:
   - Change the value of `hello_world_str` when it's created.
   - Remove the line `print()` , and look at how the output changes.
   - Change the value that `hello_world_str` is compared against in the if-statement.

In [3]:

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

```
Hello, Jupyter!

<class 'str'>
```

Example 3. [Python] Define a `list` object (IL)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

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

Notes:

1. A list object named `hello_world_list` with two values is created, and the following are printed with a blank line between them:
   - the value of the list
   - its type (which is `<class 'list'>`)
2. Lists are the most fundamental Python data structure and are related to SAS data-step arrays. 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'`
3. This example illustrates another way Python syntax differs from SAS:
   - 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 the next example).
4. For extra credit, try any or all of the following:
   - Print out the initial element of the list.
   - Print out the final element of the list.
   - Create a list of length five, and print its middle elements.

In [4]:
```
hello_world_list = ['Hello', 'list']
print(hello_world_list)
print()
print(type(hello_world_list))
```

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

<class 'list'>
```

Example 4. [Python] Define a `dict` object (IL)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

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

Notes:

1. A dictionary (`dict` for short) object named `hello_world_dict` with three key-value pairs is created, and the following are printed with a blank line between them:
    - the value of the dictionary
    - its type (which is `<class 'dict'>`)
2. Dictionaries are another fundamental Python data structure and are related to SAS formats and data-step hash tables. Dictionaries are more generally called *associative arrays* or *maps* because they map keys (appearing before the colons) to values (appearing after the colons). In other words, 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']`

3. Whenever indexable data structures are nested in Python, indexing methods can be combined. E.g., `hello_world_dict['salutation'][0] == ['Hello', 'dict'][0] == 'Hello'`.
4. In Python 3.5, the print order of key-value pairs may not match insertion order, meaning the order key-value pairs are listed when the dictionary is created. As of Python 3.7 (released in June 2018), insertion order is preserved.
5. For extra credit, try any or all of the following:
   - Print out the list with key `'salutation'`.
   - Print out the initial element in the list associated with key `'valediction'`.
   - Print out the final element in the list associated with key `'part of speech'`.

In [5]:
```
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'], 'part of speech': ['interjection', 'noun'], 'valediction': ['Go
odbye', 'list']}

<class 'dict'>
```

Example 5. [Python w/ `pandas`] Define a `DataFrame` object (IL)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```python
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()
print(hello_world_df.info())
```

Notes:

1. A DataFrame (`df` for short) object named `hello_world_df` with dimensions 2x3 (2 rows by 3 columns) is created, and the following are printed with blank lines between them:
    - the value of the DataFrame
    - the number of rows and columns in `hello_world_df`
    - some information about it, which is obtained by `hello_world_df` calling its `info` method (meaning a function whose definition is nested inside it)
2. Since DataFrames are not built into Python, we must first import their definition from the `pandas` module. Like their R counterpart, DataFrames are two-dimensional arrays of values that can be thought of like SAS datasets. However, while SAS datasets are typically only 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.
3. 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 are not of non-uniform length).
4. This example gives one option for building a DataFrame, but the constructor function can also accept many other object types, including another DataFrame.

5. For extra credit, try any or all of the following (keeping in mind that DataFrames can be indexed like dictionaries):
   - Print out the column with key `'salutation'`.
   - Print out the initial element in the column with key `'valediction'`.
   - Print out the final element in the column with key `'part of speech'`.

In [6]:

```
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()
print(hello_world_df.info())
```

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

(2, 3)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
part of speech    2 non-null object
salutation        2 non-null object
valediction       2 non-null object
dtypes: object(3)
memory usage: 128.0+ bytes
None
```

Example 6. [Python w/ `saspy`] Connect to a SAS kernel (MS)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```python
from saspy import SASsession
sas = SASsession()
print(type(sas))
```

Notes:

1. A SASsession object named `sas` is created, and the following are printed with a blank line between them:
    - confirmation a SAS session has been established
    - the type of object `sas` (which is `saspy.sasbase.SASsession`)
2. As with the DataFrame object type above, SASsession is not built into Python, so we first need to import its definition from the `saspy` module.
3. The code `from saspy import SASsession; sas = SASsession()` only needs to be used to establish a SAS kernel connection once within a given Python session. All subsequent cells in this Notebook will assume these lines have been executed.

In [7]:

```python
from saspy import SASsession
sas = SASsession()
print(type(sas))
```

```
Using SAS Config named: default
SAS Connection established. Subprocess id is 10690

<class 'saspy.sasbase.SASsession'>
```

Example 7. [Python w/ `pandas` & `saspy`] Load a SAS dataset into a `DataFrame` (MS)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
fish_df = sas.sasdata2dataframe(table='fish', libref='sashelp')
print(type(fish_df))
print()
print(fish_df.describe())
print()
print(fish_df.head())
```

Notes:

1. A DataFrame object named `fish_df` with dimensions 159x7 (159 rows and 7 columns) is created from the SAS dataset `fish` in the `sashelp` library, and the following are printed with blank lines between them:

    - the type of object `fish_df` (which is `<class 'pandas.core.frame.DataFrame'>`)
    - the first five rows of `fish_df`, which are at row indices 0 through 4 since Python uses zero-based indexing
    - summary information about its 6 numerical columns, which is obtained by `fish_df` calling its `describe` method (the pandas equivalent of the SAS MEANS procedure)
2. The `sas` object represents a connection to a SAS session and was created when a previous cell was run. Here, `sas` calls its `sasdata2dataframe` method to access the SAS library `sashelp` defined within this SAS session and to load the entire contents of SAS dataset `sashelp.fish` into the DataFrame `fish_df`.
3. For extra credit, try the following:
    - Pass a numerical parameter to the `head` method to see a different number of rows (e.g., `fish_df.head(42)`).
    - Change the `head` method to `tail` to see a different part of the dataset.
    - To view other portions of `fish_df`, explore the more advanced indexing methods `loc` and `iloc` explained at [https://brohrer.github.io/dataframe_indexing.html (https://brohrer.github.io/dataframe_indexing.html)](https://brohrer.github.io/dataframe_indexing.html).

In [8]:
```python
fish_df = sas.sasdata2dataframe(table='fish', libref='sashelp')
print(type(fish_df))
print()
print(fish_df.describe())
print()
print(fish_df.head())
```

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

            Weight    Length1    Length2    Length3     Height      Width
count   158.000000 159.000000 159.000000 159.000000 159.000000 159.000000
mean    398.695570  26.247170  28.415723  31.227044   8.970994   4.417486
std     359.086204   9.996441  10.716328  11.610246   4.286208   1.685804
min       0.000000   7.500000   8.400000   8.800000   1.728400   1.047600
25%     120.000000  19.050000  21.000000  23.150000   5.944800   3.385650
50%     272.500000  25.200000  27.300000  29.400000   7.786000   4.248500
75%     650.000000  32.700000  35.500000  39.650000  12.365900   5.584500
max    1650.000000  59.000000  63.400000  68.000000  18.957000   8.142000

  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
```

Example 8. [Python w/ `pandas`] Manipulate a `DataFrame` (MS)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```python
%%time
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)
```

Notes:

1. The DataFrame `fish_df` , which was created in a cell above from the SAS dataset `sashelp.fish` , is manipulated, and the following is printed:

    * a table giving the number of rows, standard deviation, mean, min, and max of `Weight` in `fish_df` when aggregated by `Species`
    * execution time information obtained with the Jupyter magic cell command `%%time` (which must appear as the first line in a cell)
2. This is accomplished by creating a series of new DataFrames:
    * The DataFrame `fish_df_g` is created from `fish_df` using the `groupby` method to group rows by values in column `'Species'` .
    * The DataFrame `fish_df_gs` is created from `fish_df_g` by extracting the `'Weight'` column using bracket notation.
    * The DataFrame `fish_df_gsa` is created from `fish_df_gs` using the `agg` method to aggregate by the functions in the list `['count', 'std', 'mean', 'min', 'max']` .
3. Identical results could be obtained using the following SAS code:

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

4. For extra credit, try the following:
    * Move around and/or remove functions used for aggregation, and see how the output changes.
    * Change the variable whose values are summarized to `'Width'` .
    * Obtain execution time for the one-liner version.

In [9]:
```
%%time
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
CPU times: user 3 ms, sys: 13 ms, total: 16 ms
Wall time: 51.6 ms
```

Example 9. [Python w/ `pandas` & `saspy`] Load a `DataFrame` into a SAS dataset; execute SAS code (MS)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
sas.dataframe2sasdata(fish_df_gsa, table="fish_sds_gsa", libref="Work")
sas_submit_return_value = sas.submit(
    '''
        PROC PRINT DATA=fish_sds_gsa;
        RUN;
    ''',
    results='TEXT'
)
sas_submit_results = sas_submit_return_value['LST']
print(sas_submit_results)
```

Notes:

1. The DataFrame `fish_df_gsa` , which was created in a cell above from the SAS dataset `sashelp.fish` , is used to create

the new SAS dataset `Work.fish_sds_gsa`. The SAS PRINT procedure is then called, and the following is printed:

   - the output returned by PROC PRINT

2. The `sas` object, which was created in a cell above, is a persistent connection to a SAS session, and two of its methods are used as follows:

   - The `dataframe2sasdata` method writes the contents of the DataFrame `fish_df_gsa` to the SAS dataset `fish_sds_gsa` stored in the `Work` library. (Note: The row indexes of the DataFrame `fish_df_gsa` are lost when the SAS dataset `fish_sds_gsa` is created.)
   - The `submit` method is used to submit the PROC PRINT step to the SAS kernel, and a dictionary is returned with the following two key-value pairs:
     - `sas_submit_return_value['LST']` is a string comprising the results from executing PROC PRINT, which will be in plain text because the `results='TEXT'` was used
     - `sas_submit_return_value['LOG']` is a string comprising the plain-text log resulting from executing PROC PRINT
3. Python strings surrounded by single quotes (e.g., `'Hello, World!'`) cannot be written across multiple lines of code, whereas strings surrounded by triple quotes (e.g., the argument to the `submit` method) can.
4. For extra credit, try the following:
   - Print the SAS log instead.
   - Change the SAS procedure used to interact with SAS dataset `Work.fish_sds_gsa` (e.g., try PROC CONTENTS).
   - Print the usual HTML output from PROC PRINT by adding `IPython.display import HTML` at the beginning of the cell, removing the `results='TEXT'` option from the `submit` method, and using `HTML(sas_submit_results)` instead of `print(sas_submit_results)`. (`IPython` is short for *Interactive Python* and is one of the main third-party modules the Jupyter system is built on.)

In [10]:

```
sas.dataframe2sasdata(fish_df_gsa, table="fish_sds_gsa", libref="Work")
sas_submit_return_value = sas.submit(
    '''
        PROC PRINT DATA=fish_sds_gsa;
        RUN;
    ''',
    results='TEXT'
)
sas_submit_results = sas_submit_return_value['LST']
print(sas_submit_results)
```

```
                                                       The SAS System                    Saturd
ay, August 24, 2019 06:15:00 AM   1

                                    Obs    count      std       mean       min       max

                                     1       34     206.605    626.000    242.0    1000.0
                                     2       11      78.755    154.818     55.0     300.0
                                     3       56     347.618    382.239      5.9    1100.0
                                     4       17     494.141    718.706    200.0    1650.0
                                     5       20      88.829    152.050      0.0     390.0
                                     6       14       4.132     11.179      6.7      19.9
                                     7        6     309.603    531.000    270.0    1000.0
```

Example 10. [Python w/ `saspy`] Connect directly to a SAS dataset (MS)

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

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

Notes:

1. The SASdata object `fish_sds` (meaning a direct connection to the disk-based SAS dataset `sashelp.fish` , not an in-memory DataFrame) is created, and the following are printed with a blank line between them:

   - the type of object `fish_sds`
   - the column-information portion of PROC CONTENTS applied to the SAS dataset `sashelp.fish`
   - summary information about the 7 columns in SAS dataset
2. The `sas` object, which was created in a cell above, is a persistent connection to a SAS session, and its `sasdata` method is used to create the connection to `sashelp.fish` .
3. The `fish_sds` object calls its *convenience method* `means` , which implicitly invokes PROC MEANS on `sashelp.fish` .
4. For extra credit, try the following:
   - Explore the additional convenience methods listed at [https://sassoftware.github.io/saspy/api.html#sas-data-object (https://sassoftware.github.io/saspy/api.html#sas-data-object)](https://sassoftware.github.io/saspy/api.html#sas-data-object).

```
In [11]: fish_sds = sas.sasdata(table='fish', libref='sashelp')
         print(type(fish_sds))
         print()
         print(fish_sds.columnInfo())
         print()
         print(fish_sds.means())
```

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

         Member  Num Variable  Type  Len  Pos
0  SASHELP.FISH    6   Height   Num    8   32
1  SASHELP.FISH    3  Length1   Num    8    8
2  SASHELP.FISH    4  Length2   Num    8   16
3  SASHELP.FISH    5  Length3   Num    8   24
4  SASHELP.FISH    1  Species  Char    9   48
5  SASHELP.FISH    2   Weight   Num    8    0
6  SASHELP.FISH    7    Width   Num    8   40

  Variable    N  NMiss    Median        Mean      StdDev     Min       P25  \
0   Weight  158      1  272.5000  398.695570  359.086204  0.0000  120.0000
1  Length1  159      0   25.2000   26.247170    9.996441  7.5000   19.0000
2  Length2  159      0   27.3000   28.415723   10.716328  8.4000   21.0000
3  Length3  159      0   29.4000   31.227044   11.610246  8.8000   23.1000
4   Height  159      0    7.7860    8.970994    4.286208  1.7284    5.9364
5    Width  159      0    4.2485    4.417486    1.685804  1.0476    3.3756

        P50       P75       Max
0  272.5000  650.0000  1650.000
1   25.2000   32.7000    59.000
2   27.3000   36.0000    63.400
3   29.4000   39.7000    68.000
4    7.7860   12.3778    18.957
5    4.2485    5.5890     8.142
```

Extra Credit Example 1. [Python w/ `saspy`] Get SAS code generated by a convenience method

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

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

Notes:

1. The SASdata object `fish_sds` , which was created in a cell above as a direct connection to the SAS dataset `sashelp.fish` , calls its *convenience method* `means` within a "Teach Me SAS" sandwich, and the following is printed:
   - the SAS code for the PROC MEANS step implicitly generated by the `means` convenience method
2. The `sas` object, which was created in a cell above, is a persistent connection to a SAS session, and its `teach_me_SAS` method is used as follows:
   - When called with argument `True` , SAS output is suppressed for all subsequent `saspy` convenience methods, and the SAS code generated is returned instead.
   - When `teach_me_SAS` is called with argument `False` , this behavior is turned off.
3. `True` and `False` are standard Python objects. Like their SAS equivalents, they are interchangeable with the values `1` and `0` , respectively.
4. One benefit of this process is being able to extract and modify the SAS code. For example, if a convenience method doesn't offer an option like a class statement for PROC MEANS, we can manually add it to the code generated by the `teach_me_SAS` method and then execute the modified SAS code using either the `submit` method (as in Example 9 above) or the `%%SAS` Jupyter magic (as in Extra Credit Example 2 below).

In [12]:
```
sas.teach_me_SAS(True)
fish_sds.means()
sas.teach_me_SAS(False)
```

```
proc means data=sashelp.fish stackodsoutput n nmiss median mean std min p25 p50 p75 max;run;
```

Extra Credit Example 2. [Python w/ `saspy` and Jupyter magic command] Execute arbitrary SAS code

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

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

Notes:

1. The Jupyter magic command `%%SAS` is used to redirect all code in the cell to the SAS kernel associated with SASsession object `sas` created when a previous cell was run.
2. Magic commands like `%%SAS` allow code in different languages to be intermixed within the same Notebook and are particularly helpful when options aren't provided by SASPy convenience methods (like the `class` statement for PROC MEANS). However, if both SAS and Python code should be intermixed within the same cell of a Notebook connected to a Python kernel, the `sas.submit` method will need to be used to submit SAS code to a SAS kernel.
3. The magic command `%lsmagic` can be used to list all magic commands available within a Notebook session, including both built-in commands and commands made available when packages are loaded.

In [13]:
```
%%SAS sas
proc means data=sashelp.fish std mean min max;
    class species;
    var Weight;
run;
```

Out[13]:

The SAS System

The MEANS Procedure

Analysis Variable : Weight

Species	N Obs	Std Dev	Mean	Minimum	Maximum
Bream	**35**	206.6045850	626.0000000	242.0000000	1000.00
Parkki	**11**	78.7550864	154.8181818	55.0000000	300.0000000
Perch	**56**	347.6177172	382.2392857	5.9000000	1100.00
Pike	**17**	494.1407650	718.7058824	200.0000000	1650.00
Roach	**20**	88.8289160	152.0500000	0	390.0000000
Smelt	**14**	4.1315258	11.1785714	6.7000000	19.9000000
Whitefish	**6**	309.6029716	531.0000000	270.0000000	1000.00

Extra Credit Example 3. [Python w/ `saspy`] Imitate the SAS Macro Processor

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
sas_code_fragment = 'proc means data=sashelp.%s; run;'
for dsn in ['fish','iris']:
    print(sas.submit(sas_code_fragment%dsn, results='TEXT')['LST'])
```

Notes:

1. A string object named `sas_code_fragment` is created with templating placeholder `%s` , which will be filled using other strings in subsequent uses of `sas_code_fragment` .
2. The output of PROC MEANS applied to SAS datasets `sashelp.fish` and `sashelp.iris` is then displayed.
3. The `sas` object represents a connection to a SAS session and was created when a previous cell was run. Here, `sas` calls its `submit` method for each value of the for-loop indexing variable `dsn` , and the `%s` portion of `sas_code_fragment` is replaced by the value of `dsn` . In other words, the following SAS code is submitted to the SAS kernel:

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

4. The same outcome could also be achieved with the following SAS macro code:

```
%macro loop();
     %let dsn_list = fish iris;
     %do i = 1 %to 2;
         %let dsn = %scan(&dsn_list.,&i.);
         proc means data=sashelp.&dsn.;
         run;
     %end;
%mend;
%loop()
```

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','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 variable `dsn_list` above) need to be used together with functions like `%scan` to extract a sequence of values.

In [14]:

```
sas_code_fragment = 'proc means data=sashelp.%s; run;'
for dsn in ['fish','iris']:
    print(sas.submit(sas_code_fragment%dsn, results='TEXT')['LST'])
```

```
                                                           The SAS System                   Saturd
ay, August 24, 2019 06:15:00 AM   1

                                                        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
                          ----------------------------------------------------------------------
--------


                                                           The SAS System                   Saturd
ay, August 24, 2019 06:15:00 AM   2

                                                        The MEANS Procedure

              Variable       Label                   N            Mean         Std Dev         Min
imum         Maximum
              ----------------------------------------------------------------------------------
--------------------
              SepalLength    Sepal Length (mm)     150      58.4333333       8.2806613       43.000
0000      79.0000000
              SepalWidth     Sepal Width (mm)      150      30.5733333       4.3586628       20.000
0000      44.0000000
```

```
              PetalLength    Petal Length (mm)    150      37.5800000      17.6529823      10.000
0000      69.0000000
              PetalWidth     Petal Width (mm)     150      11.9933333       7.6223767       1.000
0000      25.0000000
              ------------------------------------------------------------------------------------
--------------------
```

Extra Credit Example 4. [Python w/ `saspy`] Get information about a SAS session

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
sas_submit_return_value = sas.submit('PROC PRODUCT_STATUS; RUN;')
sas_submit_log = sas_submit_return_value['LOG']
print(sas_submit_log)
```

Notes:

1. The SAS PRODUCT_STATUS procedure is called, and the following is printed:
   - the log returned by PROC PRODUCT_STATUS
2. The `sas` object, which was created in a cell above, is a persistent connection to a SAS session, and its `submit` method is used to submit the PROC PRODUCT_STATUS step to the SAS kernel. A dictionary is returned with the following two key-value pairs:

   - `sas_submit_return_value['LST']` is a string comprising the results from executing PROC PRODUCT_STATUS, which is empty because no output is produced by this procedure
   - `sas_submit_return_value['LOG']` is a string comprising the plain-text log resulting from executing PROC PRODUCT_STATUS
3. Since a plain-text value is being printed, Python's `print` function is used to render the result.
4. Like the Python command `help('modules')` gives us information about the Python modules available to our Python session, the PRODUCT_STATUS procedure gives us information about the products available in the SAS environment we're connected to.

In [15]:

```
sas_submit_return_value = sas.submit('PROC PRODUCT_STATUS; RUN;')
sas_submit_log = sas_submit_return_value['LOG']
print(sas_submit_log)
```

```
239  ods listing close;ods html5 (id=saspy_internal) file=stdout options(bitmap_mode='inline') de
vice=svg style=HTMLBlue; ods
239! graphics on / outputfmt=png;
NOTE: Writing HTML5(SASPY_INTERNAL) Body file: STDOUT
240
241  PROC PRODUCT_STATUS; RUN;
For Base SAS Software ...
   Custom version information: 9.4_M6
   Image version information: 9.04.01M6P110718
For SAS/STAT ...
   Custom version information: 15.1
For SAS/ETS ...
   Custom version information: 15.1
For SAS/IML ...
   Custom version information: 15.1
For High Performance Suite ...
   Custom version information: 2.2_M7
For SAS/ACCESS Interface to PC Files ...
   Custom version information: 9.4_M6
NOTE: PROCEDURE PRODUCT_STATUS used (Total process time):
      real time           0.13 seconds
      cpu time            0.01 seconds

242
243  ods html5 (id=saspy_internal) close;ods listing;

244
```

Extra Credit Example 5. [Python w/ `saspy`] Adding and dropping columns from a `DataFrame`

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
class_df = sas.sasdata2dataframe(table='class', libref='sashelp')
print(class_df.columns)
print()
class_df['BMI'] = (class_df['Weight']/class_df['Height']**2)*703
print(class_df.head())
print()
class_df.drop(columns=['Height','Weight'], inplace=True)
print(class_df.head())
```

Notes:

1. A DataFrame object named `class_df` with dimensions 19x5 (19 rows and 5 columns) is created from the SAS dataset `class` in the `sashelp` library, and the following are printed with blank lines between them:
   - the names of the columns in `class_df`
   - the first five rows of `class_df` after a new column named `BMI` has been added, using the formula (https://www.cdc.gov/nccdphp/dnpao/growthcharts/training/bmiage/page5_2.html) provided by the CDC
   - the first five rows of `class_df` after the columns `Height` and `Weight` have been dropped, with the `inplace=True` option used to change `class_df` itself rather than create a copy with the columns removed
2. The `sas` object represents a connection to a SAS session and was created when a previous cell was run. Here, `sas` calls its `sasdata2dataframe` method to create `class_df` .
3. The same outcome could also be achieved with the following SAS code:

   ```
   data class(drop = Height Weight);
    set sashelp.class;
    BMI = (Weight/Height**2)*703;
   run;
   ```

   However, note the following differences: Python allows us to concisely create a new column by manipulating the entire DataFrame class_df in memory, whereas the SAS DATA step requires rows to be loaded from disk and manipulated individually.

In [16]:
```
class_df = sas.sasdata2dataframe(table='class', libref='sashelp')
print(class_df.columns)
print()
class_df['BMI'] = (class_df['Weight']/class_df['Height']**2)*703
print(class_df.head())
print()
class_df.drop(columns=['Height','Weight'], inplace=True)
print(class_df.head())
```

```
Index(['Name', 'Sex', 'Age', 'Height', 'Weight'], dtype='object')

      Name Sex  Age  Height  Weight        BMI
0   Alfred   M   14    69.0   112.5  16.611531
1    Alice   F   13    56.5    84.0  18.498551
2  Barbara   F   13    65.3    98.0  16.156788
3    Carol   F   14    62.8   102.5  18.270898
4    Henry   M   14    63.5   102.5  17.870296

      Name Sex  Age        BMI
0   Alfred   M   14  16.611531
1    Alice   F   13  18.498551
2  Barbara   F   13  16.156788
3    Carol   F   14  18.270898
4    Henry   M   14  17.870296
```

Extra Credit Example 6. [Python w/ `saspy`] Merging `DataFrame` objects

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
steel_df = sas.sasdata2dataframe(table='steel', libref='sashelp')
tourism_df = sas.sasdata2dataframe(table='tourism', libref='sashelp')
merged_df = steel_df.merge(tourism_df,left_on='DATE', right_on='year')
print(steel_df)
print()
print(tourism_df)
print()
print(merged_df)
```

Notes:

1. Two DataFrame objects named `steel_df` (44 rows by 2 columns) and `tourism_df` (29 rows by 8 columns) are created from the SAS datasets `steel` and `tourism` in the `sashelp` library, respectively, and the following are printed with blank lines between them:

   - all rows of `steel_df`
   - all rows of `tourism_df`
   - all rows of `merged_df` (15 rows by 10 columns), which was created by merging `steel_df` with `tourism_df` based on matching values in the columns `DATE` and `year`, respectively
2. The `sas` object represents a connection to a SAS session and was created when a previous cell was run. Here, `sas` calls its `sasdata2dataframe` method to create `steel_df` and `tourism_df`.
3. The same outcome could also be achieved with the following SAS code:

   ```
   proc sql;
    create table merged as
        select
             A.*
            ,B.*
        from
            sashelp.steel as A
            inner join
            sashelp.tourism as B
            on A.DATE = B.year
     ;
   quit;
   ```

   However, note the following differences:

   - The PROC SQL version is more flexible since the join condition `A.DATE = B.year` can be changed arbitrarily (not necessarily involving equality), whereas the Python can only merge based on the equality of values in one or more columns.
   - PROC SQL version can be extended to arbitrarily many tables, whereas the Python version can only operate on two DataFrame objects at a time.
4. If you see a message about datasets not existing, a SAS installation without the product SAS/ETS has been chosen.

```
In [17]: steel_df = sas.sasdata2dataframe(table='steel', libref='sashelp')
```

```
In [17]: steel_df = sas.sasdata2dataframe(table='steel', libref='sashelp')
tourism_df = sas.sasdata2dataframe(table='tourism', libref='sashelp')
merged_df = steel_df.merge(tourism_df,left_on='DATE', right_on='year')
print(steel_df)
print()
print(tourism_df)
print()
print(merged_df)
```

```
         DATE  STEEL
0  1937-01-01   3.89
1  1938-01-01   2.41
2  1939-01-01   2.80
3  1940-01-01   8.72
4  1941-01-01   7.12
5  1942-01-01   7.24
6  1943-01-01   7.15
7  1944-01-01   6.05
8  1945-01-01   5.21
9  1946-01-01   5.03
10 1947-01-01   6.88
11 1948-01-01   4.70
12 1949-01-01   5.06
13 1950-01-01   3.16
14 1951-01-01   3.62
15 1952-01-01   4.55
16 1953-01-01   2.43
17 1954-01-01   3.16
18 1955-01-01   4.55
19 1956-01-01   5.17
20 1957-01-01   6.95
21 1958-01-01   3.46
22 1959-01-01   2.13
23 1960-01-01   3.47
24 1961-01-01   2.79
25 1962-01-01   2.52
26 1963-01-01   2.80
27 1964-01-01   4.04
28 1965-01-01   3.08
29 1966-01-01   2.28
30 1967-01-01   2.17
```

31	1968-01-01	2.78
32	1969-01-01	5.94
33	1970-01-01	8.14
34	1971-01-01	3.55
35	1972-01-01	3.61
36	1973-01-01	5.06
37	1974-01-01	7.13
38	1975-01-01	4.15
39	1976-01-01	3.86
40	1977-01-01	3.22
41	1978-01-01	3.50
42	1979-01-01	3.76
43	1980-01-01	5.11

	year	vsp	pdi	puk	exuk	pop	cpisp	exsp
0	1966-01-01	1.2823	201207	0.134250	0.485709	54.643	0.096155	0.435193
1	1967-01-01	1.2718	204171	0.137742	0.563200	54.959	0.102310	0.505549
2	1968-01-01	1.5370	207772	0.144298	0.568364	55.216	0.107425	0.506419
3	1969-01-01	1.9501	209684	0.152272	0.564515	55.461	0.109823	0.508160
4	1970-01-01	1.8300	217675	0.161250	0.566155	55.632	0.113580	0.505694
5	1971-01-01	2.6126	220344	0.175189	0.576441	55.928	0.122852	0.519910
6	1972-01-01	3.1535	238744	0.186568	0.626625	56.079	0.133083	0.500617
7	1973-01-01	3.0601	254329	0.202143	0.703709	56.223	0.148190	0.498295
8	1974-01-01	2.5966	252360	0.236360	0.706515	56.236	0.170490	0.498295
9	1975-01-01	2.8815	253814	0.292056	0.784033	56.226	0.200623	0.507580
10	1976-01-01	2.1514	253012	0.337778	0.924894	56.216	0.235952	0.575470
11	1977-01-01	2.5440	247695	0.387586	0.863692	56.190	0.293742	0.712845
12	1978-01-01	2.8602	265925	0.424315	0.867812	56.178	0.351930	0.662508
13	1979-01-01	2.8615	281084	0.482226	0.802721	56.240	0.407082	0.632045
14	1980-01-01	2.9249	285411	0.560568	0.724715	56.330	0.470626	0.733154
15	1981-01-01	3.5820	283176	0.623576	0.826736	56.352	0.539365	0.822732
16	1982-01-01	4.4511	281722	0.677889	0.925951	56.318	0.538406	1.004932
17	1983-01-01	5.4049	289204	0.710628	0.978113	56.377	0.604108	1.189962
18	1984-01-01	6.1042	299756	0.712308	1.148640	56.506	0.672368	1.232828
19	1985-01-01	5.4272	309821	0.785876	1.030533	56.685	0.731596	1.228113
20	1986-01-01	6.3056	323622	0.817104	1.124233	56.852	0.795940	1.174585
21	1987-01-01	6.4000	334702	0.852381	1.027294	57.009	0.837343	1.121564
22	1988-01-01	6.7200	354627	0.894970	1.007860	57.158	0.877548	1.107348
23	1989-01-01	5.8800	371676	0.947763	1.109298	57.358	0.937175	1.045840
24	1990-01-01	4.6370	378325	1.000000	1.000000	57.561	1.000000	1.000000

```
25 1991-01-01  4.4283  377969  1.073715  1.036266  57.808  1.058988  1.003191
26 1992-01-01  4.6480  386804  1.124222  1.232419  58.006  1.121493  1.143178
27 1993-01-01  5.8668  393125  1.163626  1.256732  58.191  1.173048  1.416842
28 1994-01-01  7.2167  396181  1.192823  1.266178  58.395  1.228119  1.394937

         DATE  STEEL        year     vsp     pdi       puk      exuk     pop  \
0  1966-01-01   2.28 1966-01-01  1.2823  201207  0.134250  0.485709  54.643
1  1967-01-01   2.17 1967-01-01  1.2718  204171  0.137742  0.563200  54.959
2  1968-01-01   2.78 1968-01-01  1.5370  207772  0.144298  0.568364  55.216
3  1969-01-01   5.94 1969-01-01  1.9501  209684  0.152272  0.564515  55.461
4  1970-01-01   8.14 1970-01-01  1.8300  217675  0.161250  0.566155  55.632
5  1971-01-01   3.55 1971-01-01  2.6126  220344  0.175189  0.576441  55.928
6  1972-01-01   3.61 1972-01-01  3.1535  238744  0.186568  0.626625  56.079
7  1973-01-01   5.06 1973-01-01  3.0601  254329  0.202143  0.703709  56.223
8  1974-01-01   7.13 1974-01-01  2.5966  252360  0.236360  0.706515  56.236
9  1975-01-01   4.15 1975-01-01  2.8815  253814  0.292056  0.784033  56.226
10 1976-01-01   3.86 1976-01-01  2.1514  253012  0.337778  0.924894  56.216
11 1977-01-01   3.22 1977-01-01  2.5440  247695  0.387586  0.863692  56.190
12 1978-01-01   3.50 1978-01-01  2.8602  265925  0.424315  0.867812  56.178
13 1979-01-01   3.76 1979-01-01  2.8615  281084  0.482226  0.802721  56.240
14 1980-01-01   5.11 1980-01-01  2.9249  285411  0.560568  0.724715  56.330

       cpisp      exsp
0   0.096155  0.435193
1   0.102310  0.505549
2   0.107425  0.506419
3   0.109823  0.508160
4   0.113580  0.505694
5   0.122852  0.519910
6   0.133083  0.500617
7   0.148190  0.498295
8   0.170490  0.498295
9   0.200623  0.507580
10  0.235952  0.575470
11  0.293742  0.712845
12  0.351930  0.662508
13  0.407082  0.632045
14  0.470626  0.733154
```

Extra Credit Example 7. [Python w/ `saspy`] Appending `DataFrame` objects

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
countseries_df = sas.sasdata2dataframe(table='countseries', libref='sashelp')
print(countseries_df.head())
print()
countseries_df.columns = ['Date', 'Amount']
print(countseries_df)
print()

rockpit_df = sas.sasdata2dataframe(table='rockpit', libref='sashelp')
print(rockpit_df)
print()
rockpit_df.columns = [column.title() for column in rockpit_df.columns]
print(rockpit_df)
print()

appended_df = countseries_df.append(rockpit_df)
print(appended_df)
```

Notes:

1. Two DataFrame objects named `countseries_df` (108 rows by 2 columns) and `rockpit_df` (6 rows by 8 columns) are created from the SAS datasets `countseries` and `rockpit` in the `sashelp` library, respectively, and the following are printed with blank lines between them:
   - the first five rows of `countseries_df` before its columns are renamed
   - all rows of `countseries_df` after its columns are renamed by providing a new list of column names
   - all rows of `rockpit_df` before its columns are renamed
   - all rows of `rockpit_df` after its columns are renamed using a list comprehension in order to have the column `'DATE'` match `'Date'` in `countseries_df` (where, e.g., `'DATE'.title()` results in `'Date'` since `title` is the Python equivalent of the SAS DATA step function `propcase`)
   - all rows of `appended_df` (114 rows by 3 columns), which was created by appending `countseries_df` and `rockpit_df`
2. The `sas` object represents a connection to a SAS session and was created when a previous cell was run. Here, `sas` calls its

`sasdata2dataframe` method to create `countseries_df` and `rockpit_df`.

3. The same outcome could also be achieved with the following SAS code:

```
proc sql;
 create table appended as
     select Date as Date, Units as Amount from sashelp.countseries
     union all corr
     select DATE as Date, AMOUNT as Amount from sashelp.rockpit
 ;
quit;
```

   However, note the following differences:

   - The PROC SQL version is more flexible since the set operation `union` could be replaced by other operations (e.g., `intersect` to get just rows in column), whereas more work would be needed to achieve the same result in Python.
   - The PROC SQL version can be extended to arbitrarily many tables, whereas the Python version can only operate on two DataFrame objects at a time.
   - The PROC SQL version doesn't require the use of column aliases to change case (e.g., `DATE as Date`) since the SAS implementation of SQL is not case sensitive. However, it's been included above to exactly mirror the Python version.

4. As an alternative to carefully renaming columns, we could have also begun this example with `sas.submit('OPTIONS VALIDVARNAME=UPCASE;')`, which would have converted all SAS dataset column names to uppercase before import.
5. If you see a message about datasets not existing, a SAS installation without the product SAS/ETS has been chosen.

In [18]:
```python
countseries_df = sas.sasdata2dataframe(table='countseries', libref='sashelp')
print(countseries_df.head())
print()
countseries_df.columns = ['Date', 'Amount']
print(countseries_df)
print()

rockpit_df = sas.sasdata2dataframe(table='rockpit', libref='sashelp')
print(rockpit_df)
print()
rockpit_df.columns = [column.title() for column in rockpit_df.columns]
print(rockpit_df)
print()
```

```
appended_df = countseries_df.append(rockpit_df)
print(appended_df)
```

```
         Date  Units
0 2004-01-01      0
1 2004-02-01      0
2 2004-03-01      4
3 2004-04-01      0
4 2004-05-01      4

          Date  Amount
0   2004-01-01       0
1   2004-02-01       0
2   2004-03-01       4
3   2004-04-01       0
4   2004-05-01       4
5   2004-06-01       0
6   2004-07-01      10
7   2004-08-01       4
8   2004-09-01       0
9   2004-10-01       9
10  2004-11-01       5
11  2004-12-01       0
12  2005-01-01       0
13  2005-02-01       2
14  2005-03-01       3
15  2005-04-01       4
16  2005-05-01       9
17  2005-06-01       0
18  2005-07-01       0
19  2005-08-01       5
20  2005-09-01       5
21  2005-10-01       3
22  2005-11-01       5
23  2005-12-01       7
24  2006-01-01       0
25  2006-02-01       3
26  2006-03-01       5
27  2006-04-01       5
28  2006-05-01      10
```

```
29  2006-06-01       5
..         ...     ...
78  2010-07-01       1
79  2010-08-01       5
80  2010-09-01       6
81  2010-10-01       3
82  2010-11-01       5
83  2010-12-01       8
84  2011-01-01       0
85  2011-02-01       1
86  2011-03-01       4
87  2011-04-01       3
88  2011-05-01       5
89  2011-06-01       6
90  2011-07-01       0
91  2011-08-01       1
92  2011-09-01       7
93  2011-10-01       4
94  2011-11-01       3
95  2011-12-01       5
96  2012-01-01       0
97  2012-02-01       2
98  2012-03-01       4
99  2012-04-01       0
100 2012-05-01       3
101 2012-06-01       2
102 2012-07-01       0
103 2012-08-01       9
104 2012-09-01       4
105 2012-10-01       5
106 2012-11-01       0
107 2012-12-01       2

[108 rows x 2 columns]

        DATE  AMOUNT
0 1998-01-01  -84000
1 1999-01-01  -36000
2 2000-01-01  -36000
3 2001-01-01 -120000
```

```
4 2002-01-01  -36000
5 2003-01-01  -26000

        Date  Amount
0 1998-01-01  -84000
1 1999-01-01  -36000
2 2000-01-01  -36000
3 2001-01-01 -120000
4 2002-01-01  -36000
5 2003-01-01  -26000

          Date  Amount
0   2004-01-01       0
1   2004-02-01       0
2   2004-03-01       4
3   2004-04-01       0
4   2004-05-01       4
5   2004-06-01       0
6   2004-07-01      10
7   2004-08-01       4
8   2004-09-01       0
9   2004-10-01       9
10  2004-11-01       5
11  2004-12-01       0
12  2005-01-01       0
13  2005-02-01       2
14  2005-03-01       3
15  2005-04-01       4
16  2005-05-01       9
17  2005-06-01       0
18  2005-07-01       0
19  2005-08-01       5
20  2005-09-01       5
21  2005-10-01       3
22  2005-11-01       5
23  2005-12-01       7
24  2006-01-01       0
25  2006-02-01       3
26  2006-03-01       5
27  2006-04-01       5
```

```
28  2006-05-01      10
29  2006-06-01       5
..         ...     ...
84  2011-01-01       0
85  2011-02-01       1
86  2011-03-01       4
87  2011-04-01       3
88  2011-05-01       5
89  2011-06-01       6
90  2011-07-01       0
91  2011-08-01       1
92  2011-09-01       7
93  2011-10-01       4
94  2011-11-01       3
95  2011-12-01       5
96  2012-01-01       0
97  2012-02-01       2
98  2012-03-01       4
99  2012-04-01       0
100 2012-05-01       3
101 2012-06-01       2
102 2012-07-01       0
103 2012-08-01       9
104 2012-09-01       4
105 2012-10-01       5
106 2012-11-01       0
107 2012-12-01       2
0   1998-01-01  -84000
1   1999-01-01  -36000
2   2000-01-01  -36000
3   2001-01-01 -120000
4   2002-01-01  -36000
5   2003-01-01  -26000

[114 rows x 2 columns]
```

Extra Credit Example 8. [Python w/ `saspy`] Indexing a column in a `DataFrame`

Type the following code into the cell labelled `[ ]:` immediately below, and then run that cell using Shift-Enter:

```
class_df = sas.sasdata2dataframe(table='class', libref='sashelp')
print(class_df.head())
print()

class_df.set_index('Name', inplace=True)
print(class_df.head())
print()

alfreds_row = class_df.loc['Alfred',:]
print(alfreds_row)
print()
```

Notes:

1. A DataFrame object named `class_df` with dimensions 19x5 (19 rows and 5 columns) is created from the SAS dataset `class` in the `sashelp` library, and the following are printed with blank lines between them:
   - the first five rows of `class_df`
   - the first five rows of `class_df` after the column `'Name'` has been set as its index, which eliminates the previously used default numerical index column and makes querying by student more streamlined
   - the row in `class_df` corresponding to `'Name'='Alfred'`, which would have required a more complex operation to first look up the row corresponding to `Alfred` if an index hadn't been created
2. The `sas` object represents a connection to a SAS session and was created when a previous cell was run. Here, `sas` calls its `sasdata2dataframe` method to create `class_df`.
3. The same outcome could also be achieved with the following SAS code:

```
proc sql;
 create table class(index=(names)) as
     select * from sashelp.class
 ;
quit;

data alfreds_row;
    set class(idxwhere=yes);
    where name='Alfred';
run;
```

However, note the following differences: Python allows us to set one (or more) columns as indexes for a DataFrame, allowing rows to be selected by implicitly querying the values in the index column(s). Since a DataFrame is stored entirely in memory, this allows specific rows to be retrieved much more efficiently than the SAS DATA step, which requires rows to be loaded from disk and inspected individually.

In [19]:
```
class_df = sas.sasdata2dataframe(table='class', libref='sashelp')
print(class_df.head())
print()

class_df.set_index('Name', inplace=True)
print(class_df.head())
print()

alfreds_row = class_df.loc['Alfred',:]
print(alfreds_row)
print()
```

```
      Name Sex  Age  Height  Weight
0   Alfred   M   14    69.0   112.5
1    Alice   F   13    56.5    84.0
2  Barbara   F   13    65.3    98.0
3    Carol   F   14    62.8   102.5
4    Henry   M   14    63.5   102.5

        Sex  Age  Height  Weight
Name
Alfred    M   14    69.0   112.5
Alice     F   13    56.5    84.0
Barbara   F   13    65.3    98.0
Carol     F   14    62.8   102.5
Henry     M   14    63.5   102.5

Sex              M
Age             14
Height          69
Weight       112.5
Name: Alfred, dtype: object
```