# I Don't Use Python or R
# - Why Should I Use Jupyter Notebooks?

**Jack Hamilton**

**Division of Research**

**Kaiser Permanente**

**Oakland, California**

## Abstract

Jupyter Notebooks is a web-based tool for editing and running programs in various languages, and for displaying and distributing the results.

Most Jupyter "advertising" to SAS users has focused on how it works with R and Python. But what if you don't know/care about those languages? Can Jupyter Notebooks still be useful?

This presentation starts with an all-SAS demonstration of Jupyter Notebooks, and shows how they can be useful to old-school programmers. A few of the ways: Iterative development, Integrated output, and Stylish documentation.

**A Note About This Paper**

This paper was written using the markdown document facility in Jupyter; markdown is one of the topics of this paper. As a consequence of using pure markdown instead of Word, it will not be possible to make this paper look like all the other papers.

One thing I didn't completely realize about Jupyter until I wrote this presentation is what a tremendous time-saver it is to use markdown instead of Microsoft Word. It saved at least a day of work. Besides the time savings in the initial creation, using markdown will make updates much easier

- no need to copy and paste code and results from SAS to Word, no worries about whether I've accidentally updated a piece of code without updating all the results... It happens automatically!

This presentation, updates, and source materials can be found at https://github.com/jack-hamilton-kpdor/why-jupyter-notebooks

**Some History of Notebooks**

Jupyter Notebooks have their roots in the paper notebooks you may have encountered in chemistry class, if you're old enough to remember paper. Wikipedia says:

> A notebook (notepad, writing pad, drawing pad, legal pad) is a book or binder of paper pages, often ruled, used for purposes such as recording notes or memoranda, writing, drawing or scrapbooking. Scientists and other researchers use lab notebooks to document their experiments. The pages in lab notebooks are sometimes graph paper to plot data. During the fourteenth and fifteenth centuries notebooks were often made by hand at home by folding pieces of paper in half into gatherings that were then bound at a later date. The pages were blank and every notekeeper had to make ruled lines across the paper. Making and keeping notebooks was such an important information management technique that children learned how to do it in school. https://en.wikipedia.org/wiki/Notebook - edited and re-ordered from the original

> Notebooks are traditionally used in the sciences as electronic lab notebooks to document research procedures, data, calculations, and findings. Notebooks track methodology to make it easier to reproduce results and calculations with different data sets. The notebook interface was first introduced in 1988 with the release of Wolfram Mathematica 1.0 on the Macintosh. It was followed by Maple in 1989 when their first notebook-style graphical user interface was released with version 4.3 for the Macintosh. https://en.wikipedia.org/wiki/Notebook_interface - edited and re-ordered from the original

Notebooks have a long pedigree, both on paper and in electrons.

**What's so good about notebooks?**

- You can break a program up into small pieces and rerun those pieces in any order, without losing the history of what you've done.

- You can easily include documentation in the notebook using a simple but powerful markup language named *markdown*.

- Not the topic of this paper, but you can switch between different programming languages in the same notebook. Many of the SAS Institute examples use SAS and Python, or SAS and R; other languages are also available.

**Why should you care about Notebooks?**

Why should you care about notebooks? Maybe you shouldn't. You might not need or want anything Jupyter does, or you might find it slow and cumbersome (the way I feel about Enterprise Guide, which many people claim to like), or you may have existing processes that are time-tested and comfortable. But I've found it very useful, especially for small-ish projects that need to be presented to others. Having the programs and documentation is one place is a big time-saver.

**A PR Problem**

Jupyter has a bit of a PR problem for traditional SAS programmers. It's often presented as a way to do things that plain old fashioned coders like me don't need or want to do, such as passing data to Python to do in 12 steps something I could do in SAS in 1 step. And the "advertising" uses a lot of buzzwords (italics mine):

> With the recent introduction of the official SASPy package, it is now trivial to incorporate SAS® into new *workflows leveraging* the simple yet *presentationally elegant* Jupyter Notebook *coding and publication environment*, along with the broader Python *data science ecosystem* that comes with it. https://support.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2822-2018.pdf

(I don't mean to pick on the author of that paper - it's actually a good paper, and I encourage you to read it. But the introduction is Buzzword City.)

Also, the examples tend to emphasize Pythonic aspects of Notebooks:

```
In [7]: import saspy
        import pandas as pd
        import ssl
        import urllib

        context = ssl._create_unverified_context()
        link_info = urllib.request.urlopen('https://raw.githubusercontent.com/zonination/perceptions/master/probly.csv',
          context=context)

        df = pd.read_csv(link_info)
        df.describe()
```

Out[7]:

| | Almost Certainly | Highly Likely | Very Good Chance | Probable | Likely | Probably | We Believe | Better Than Even | About Even | We Doubt | Improbable | Unlik |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 46.000000 | 46.000000 | 46.000000 | 46.000000 | 46.000000 | 46.000000 | 46.000000 | 46.000000 | 46.000000 | 46.000000 | 46.000000 | 46.0000 |
| mean | 92.645652 | 86.217391 | 79.760870 | 71.456522 | 72.000000 | 71.521739 | 68.521739 | 58.393478 | 49.565217 | 27.869565 | 18.021761 | 19.9347 |
| std | 7.103542 | 12.878773 | 6.467645 | 10.243061 | 9.874771 | 10.974190 | 16.734182 | 11.788213 | 1.857743 | 20.175462 | 14.384361 | 10.0474 |
| min | 60.000000 | 15.000000 | 65.000000 | 50.000000 | 40.000000 | 45.000000 | 5.000000 | 5.000000 | 40.000000 | 1.000000 | 0.000000 | 2.0000 |
| 25% | 90.000000 | 84.250000 | 75.000000 | 65.000000 | 65.000000 | 65.000000 | 60.000000 | 55.000000 | 50.000000 | 15.000000 | 7.000000 | 10.0000 |
| 50% | 95.000000 | 90.000000 | 80.000000 | 70.000000 | 70.000000 | 75.000000 | 70.000000 | 60.000000 | 50.000000 | 25.000000 | 15.000000 | 20.0000 |
| 75% | 97.750000 | 95.000000 | 85.000000 | 78.750000 | 75.000000 | 80.000000 | 80.000000 | 60.000000 | 50.000000 | 33.000000 | 28.000000 | 28.7500 |
| max | 99.000000 | 99.000000 | 91.000000 | 90.000000 | 90.000000 | 90.000000 | 100.000000 | 98.000000 | 52.000000 | 100.000000 | 50.000000 | 36.0000 |

https://blogs.sas.com/content/sasdummy/2018/02/05/python-sas-university-edition/

Nice, but a bit scary as the first example in an introduction.

Note, of course, that the SAS equivalent would be equally scary to a Python programmer, and might even have more lines of code.

## Looking at a Notebook

Let's forget the scary stuff for a minute and concentrate on what a simple interface looks like. Here's what a brand new notebook created in Jupyter Lab looks like:
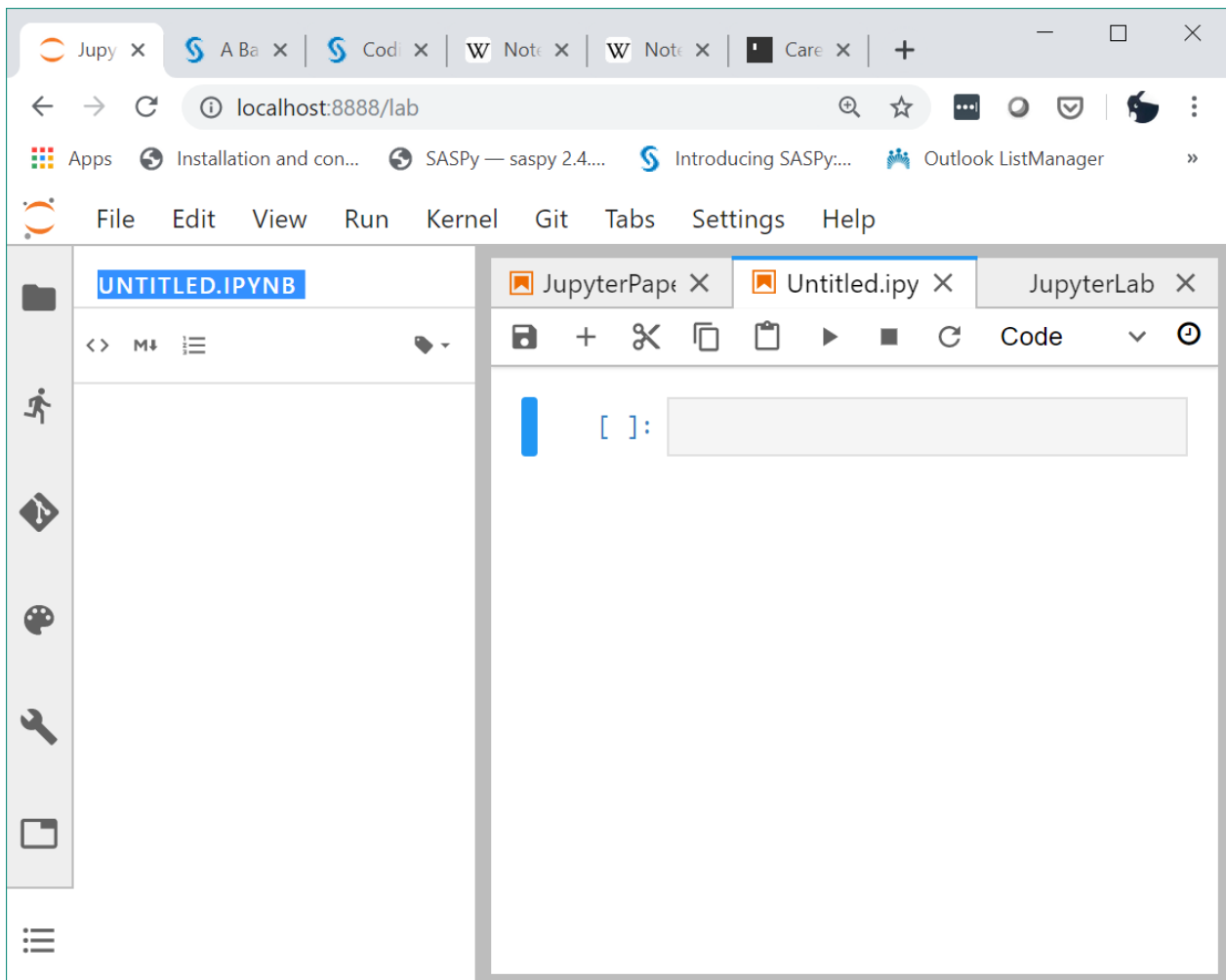
This is just an ordinary browser window, Chrome in this case but other browsers also work. You can see browser tabs along the top, the address bar, extensions, bookmarks, and so forth. Everything below that is an extremely fancy web page created by Jupyter.

**It's a lot like other SAS interfaces**

I'll skip ahead a bit and put in some SAS code and run it, and I will also click on the icon on the left that looks like a file folder. Here's what I get:
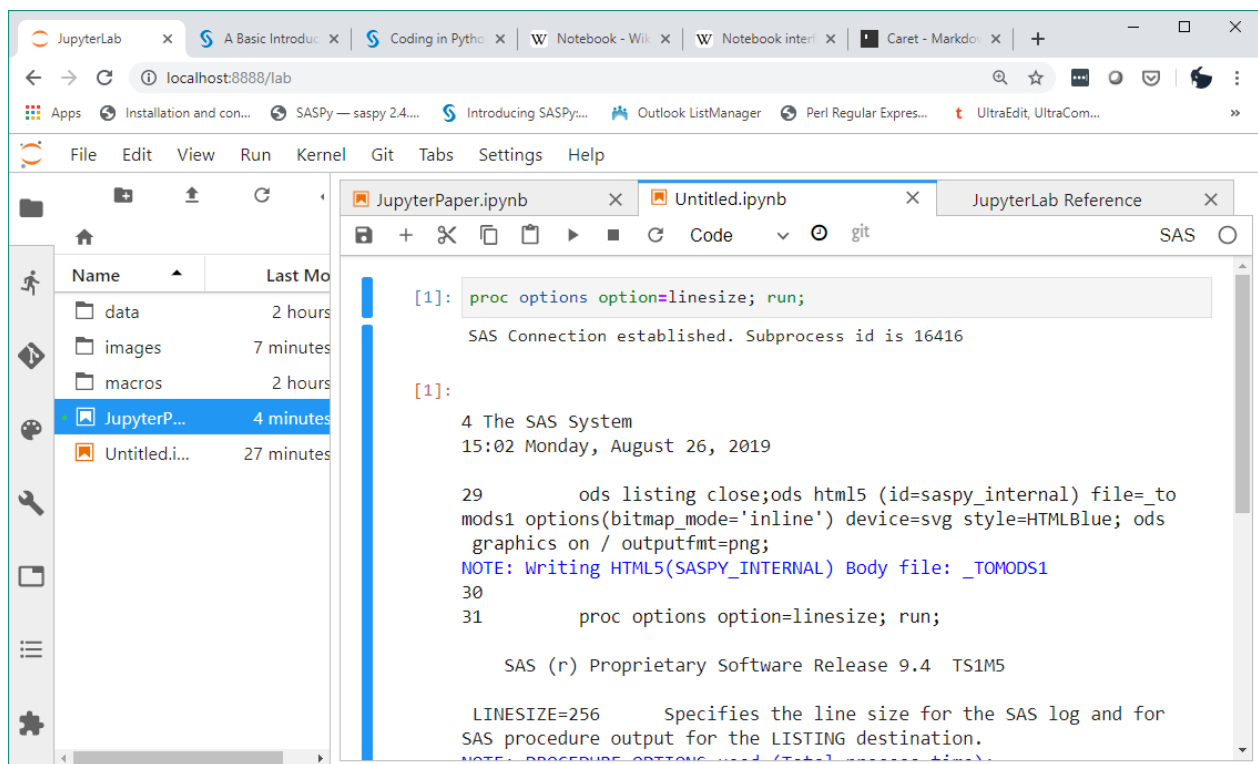
This looks quite a bit like a more traditional interface, just arranged differently. I can see my code and the SAS log on the right, and a file explorer on the left.

**Code and Output Cells**

This also illustrates one of the most important concepts in Jupyter - **cells**. A cell in Python is a box that might contain SAS code, SAS log output, SAS ODS output, or documentation.

To run the code in a cell, I position the cursor in the cell and click on the right-pointing triangle in the command bar.

After the code has completed running, you will see two types of cells. The cell with text in a grey box (labeled with a blue 1 at the left) is a code cell, and the cell underneath (with a red 1) is an output cell that shows the log resulting from running the code in the code cell. The code cell is editable, and as you can see, it has some syntax coloring.
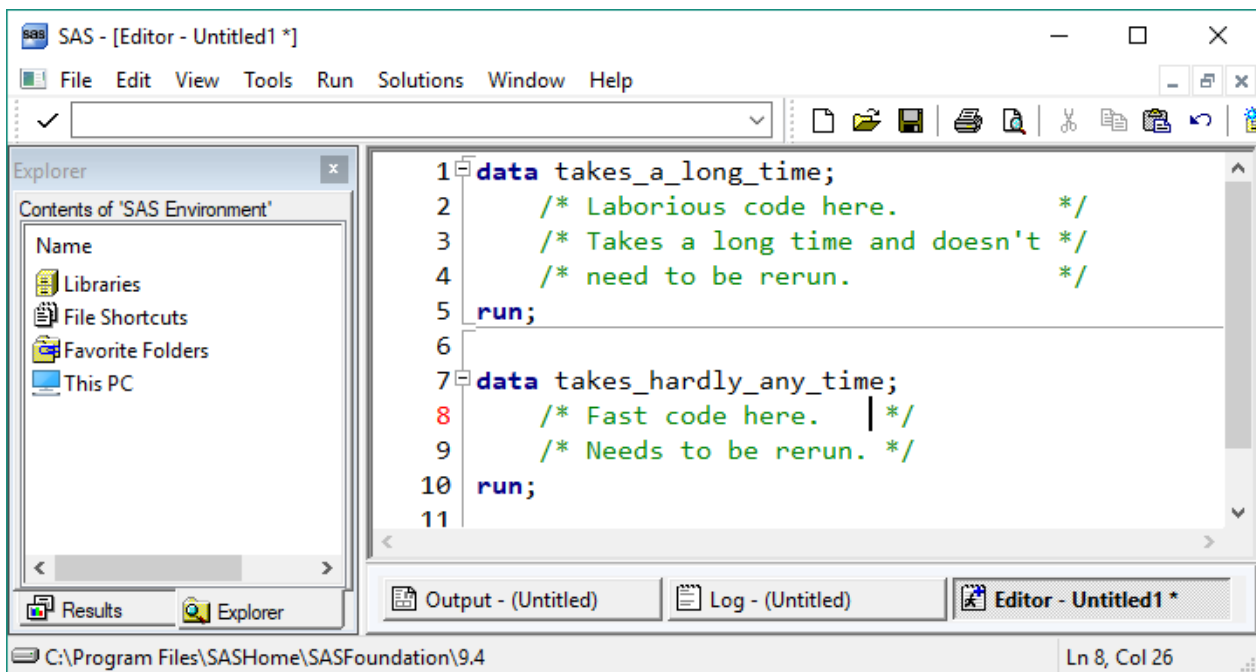
Like SAS Studio and Enterprise Guide, Jupyter sends code off into the ether to run. Generally will go to a server somewhere. The first thing to happen is establishing a connection, as shown in `SAS Connection established. Subprocess id is 16416` . There's usually a 10-20 second pause while communications are being established, but that happens only once at the beginning. Jupyter will keep re-using the same connection until you close it.

**Multiple cells, and Joining and Splitting cells**

Usually, you will have multiple code cells in a notebook. There is often a way to divide your code into logical units - for example, you might have a section where you define all your libnames and global macro variables, another where you define all your macros, another for importing data, another for massaging the data, another for creating a graph or report, and so on. Depending on the size and complexity of your program, you may have dozens of cells.

If you divide your program properly, this can be very helpful. If, for example, you change a macro at the beginning of the program that isn't used until the end, you can run only the first and last cells without having to rerun the unchanged part in the middle. Then when you think everything is in its final perfected state you can run everything straight through from the beginning.

But there's an additional benefit for those of us with poor motor skills. Suppose you have something like this:



I would not care to count the number of times I have accidentally run the whole set of code and had to wait while the unnecessary code runs again. Maybe you don't have that problem, but I do.

I might have the same problem with Notebook code:

```
[1]: data takes_a_long_time;
         /* Laborious code here.          */
         /* Takes a long time and doesn't */
         /* need to be rerun.             */
     run;

     data takes_hardly_any_time;
         /* Fast code here.    */
         /* Needs to be rerun. */
     run;
```

But I can position the cursor where I want the code to be split, select a menu item or hotkey, and:

```
[1]: data takes_a_long_time;
         /* Laborious code here.          */
         /* Takes a long time and doesn't */
         /* need to be rerun.             */
     run;
```

```
[1]: data takes_hardly_any_time;
         /* Fast code here.    */
         /* Needs to be rerun. */
     run;
```

It's harder to accidentally run more than one cell at a time, because there's no command bar icon for that action.

After you have finished debugging or testing, you can join the cells together again if you want. How large the code sections are is a matter of style.

## Documentation

There are lots of good things about Jupyter, but one of the best is that it includes support for "markdown", which is a text-based format for rich text display. This presentation was written entirely in markdown.

What I like about markdown:

- It's easy to write, because ease of writing was one of the design goals.

- It looks like plain text, because it is plain text. Markdown elements that tell Jupyter how to display the documentation are just text, and are not hard to understand even if you don't have a markdown viewer.

> The overriding design goal for Markdown's formatting syntax is to make it as readable as possible. The idea is that a Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions. http://daringfireball.net/projects/markdown/

- It's not Powerpoint or Word. There are no invisible settings that do weird things to your document and are impossible to find.

- There are both free and commercial programs to edit and display markdown outside of Jupyter. UltraEdit and Sublime Edit are commercial editors that can display markdown in additon to being general purpose editors. Caret is an example of a commercial editor that specializes in markdown; it's what I used to put the finishing touches on this document (which was otherwise written entirely in Jupyter)

What I don't like about markdown:

- I don't have the complete control over every aspect of appearance that I can get in Word or Powerpoint if I spend hours slaving over a hot keyboard. You can use raw HTML and CSS, but ugh. Still, basic HTML code is useful for highlighting text and creating page breaks in output. Also, you can use HTML comments to exclude text from display while still keeping it available.

- There are several different versions of markdown, slightly incompatible with each other, and it's hard even to know which version a particular program is using.

If you want to try markdown without installing Jupyter, you can use this online version, which includes Python but not SAS:

https://jupyter.org/try

**What does markdown look like?**

It looks like text when you type it, and like a formatted document when you display it. An earlier version of the first part of this presentation looked like this:

### Some History

Jupyter Notebooks have their roots in the paper notebooks you may have encountered in chemistry class, if you're

> A notebook (notepad, writing pad, drawing pad, legal pad) is a book or binder of paper pages, often ruled,
> Scientists and other researchers use lab notebooks to document their experiments. The pages in lab notebo
> notebooks were often made by hand at home by folding pieces of paper in half into gatherings that were t
> across the paper. Making and keeping notebooks was such an important information management techniq
> and re-ordered from the original

> Notebooks are traditionally used in the sciences as electronic lab notebooks to document research procedu
> results and calculations with different data sets. The notebook interface was first introduced in 1988 with th
> their first notebook-style graphical user interface was released with version 4.3 for the Macintosh. https://er

Notebooks have a long pedigree, both on paper and in electrons.

### What's so good about notebooks?

- You can lets you break a program up into small pieces and rerun those pieces in any order, without losing the

- You can easily include documentation in the notebook.

Different text sizes, indented text, bullet points - all things you might want in a document. The source looks like this:

```
### Some History

Jupyter Notebooks have their roots in the paper notebooks you may have

>A notebook (notepad, writing pad, drawing pad, legal pad) is a book o
scrapbooking.  Scientists and other researchers use lab notebooks to d
fifteenth centuries notebooks were often made by hand at home by foldi
notekeeper had to make ruled lines across the paper. Making and keepin
[https://en.wikipedia.org/wiki/Notebook - edited and re-ordered from t

>Notebooks are traditionally used in the sciences as electronic lab no
to reproduce results and calculations with different data sets. The no
followed by Maple in 1989 when their first notebook-style graphical us
[https://en.wikipedia.org/wiki/Notebook_interface - edited and re-orde

Notebooks have a long pedigree, both on paper and in electrons.
```

```
### What's so good about notebooks?

- You can lets you break a program up into small pieces and rerun thos

- You can easily include documentation in the notebook.
```

And it does tables, which can be a chore to set up in Word or Powerpoint. In markdown, you just type in the data with some straightforward markup:

```
| Title                   | Author            | Price  |
| :---------------------- | :---------------: | -----: |
| Meditations             | Marcus Aurelius   | 10.00  |
| Rational Optimist       | Matt Ridley       | 12.00  |
| Poor Charlie's Almanack | Charles T. Munger | 16.50  |
```

and you get this:

| Title | Author | Price |
| :--- | :---: | ---: |
| Meditations | Marcus Aurelius | 10.00 |
| Rational Optimist | Matt Ridley | 12.00 |
| Poor Charlie's Almanack | Charles T. Munger | 16.50 |

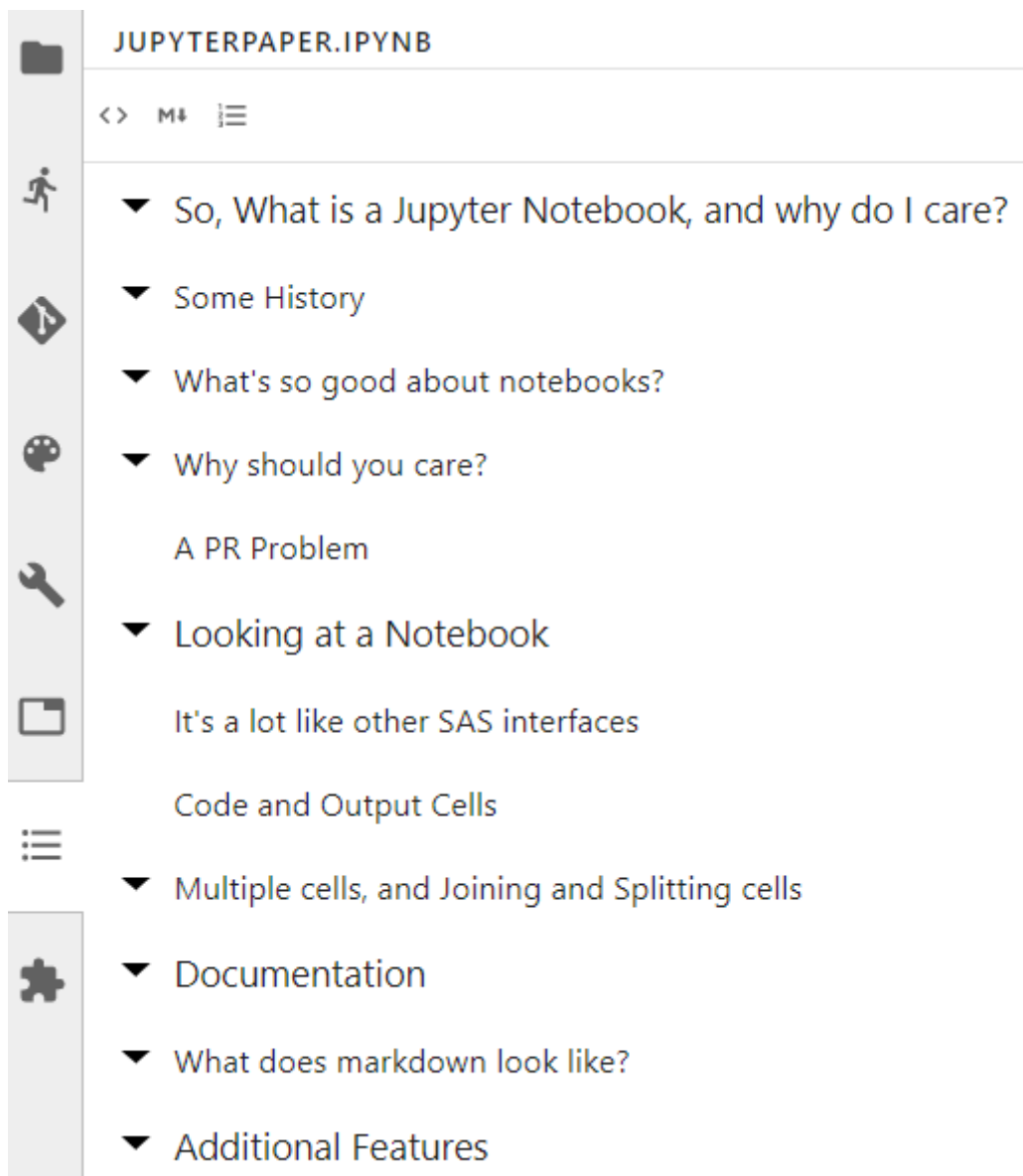You don't even need to make the input columns line up if you don't want to.

You can also enter math symbols and equations in some versions of markdown.

## Additional Features

Jupyter is extensible, and there are numerous extensions out there, of varying quality. Here are some I have found helpful:

**Table of Contents**

The table of contents extension is really handy. If you have a large document, you can use the table of contents to move between distant sections without hunting. This may come already installed in your copy of Jupyter.

JUPYTERPAPER.IPYNB

`<>` M↓ ≣

▼ So, What is a Jupyter Notebook, and why do I care?

  ▼ Some History

  ▼ What's so good about notebooks?

  ▼ Why should you care?

    A PR Problem

  ▼ Looking at a Notebook

    It's a lot like other SAS interfaces

    Code and Output Cells

  ▼ Multiple cells, and Joining and Splitting cells

  ▼ Documentation

  ▼ What does markdown look like?

  ▼ Additional Features

**imgclip for images from the clipboard**

This extension lets you copy and paste an image into a markdown cell. Can't easily show this on paper, but if there's time, I'll do a demo.

**Git version control support**

I don't have Git on this machine, but this extension works well on a machine that does.

**nbdime to show differences between two notebooks**

Jupyter keeps a recent checkpoint copy of your notebooks. nbdime will do a comparison between the current copy and the most recent checkpoint, so you can see what you have changed. In this case, I added a word to the cell above, and added the current cell. (this is out of date, because I've made additional changes, but it gives the idea.)

| Checkpoint | JupyterPaper.ipynb |
|---|---|

<div align="center">16 unchanged cell(s) hidden</div>

```
1  ### Git version control support
2
3  I don't have Git on this machine, but this
   extension works fairly well on a machine that
   does.
```

```
1  ### Git version control support
2
3  I don't have Git on this machine, but this
   extension works well on a machine that does.
```

```
1  ### nbdime for showing differences between two
   notebooks or between the current notebook and
   the checkpoint/backup copy
2
```

```
1  ### nbdime for showing differences between two
   notebooks or between the current notebook and
   the checkpoint/backup copy
2
3  Jupyter keeps a recent checkpoint copy of your
   notebooks.  nbdime will do a comparison between
   the current copy and the most recent
   checkpoint, so you can see what you have
   changed.  In this case, I added a word to the
   cell above, and added this cell
4
```

You can also run nbdime against any two notebooks using a command at the Anaconda prompt:

```
nbdiff-web JupyterPaper.ipynb backup\JupyterPaper-backup.ipynb
```

**Jupyter Magics**

If a code cell starts with a % sign, Jupyter interprets it as a command (called a "magic"). You can get a list of magics in the Help menu, or with the `%lsmagic` magic. Here are a few

```
%%time

%put Shows cell execution time
```

```
Time: 0.7189953327178955 seconds.
```

```
374
375         %put Shows cell execution time
376
```

```
%ls -r ./images
```

./images/

[Capture.JPG](Capture.JPG)

[extensions.png](extensions.png)

[markdown1.png](markdown1.png)

```
%lsmagic
```

```
Available line magics:
%activity  %cd  %connect_info  %conversation  %dot  %download  %edit  %get  %help
%html  %include  %install  %install_magic  %javascript  %jigsaw  %kernel  %kx
%latex  %load  %ls  %lsmagic  %macro  %magic  %matplotlib  %parallel  %plot  %pmap
%prompt4var  %px  %python  %reload_magics  %restart  %run  %scheme  %set  %shell
%showFullLog  %showLog

Available cell magics:
%%activity  %%brain  %%conversation  %%debug  %%dot  %%file  %%help  %%html
%%javascript  %%kx  %%latex  %%macro  %%pipe  %%processing  %%prompt4var  %%px
%%python  %%scheme  %%shell  %%show  %%time  %%tutor
```

# Problems with Jupyter

OK, it's not perfect.

- Probably the biggest is lack of the ability to browse SAS data sets. You can run a PROC PRINT, but there's no equivalent of viewtable (that I know of - there might well be one I don't know of, and there's no reason it couldn't be done).

- You can't interrupt it in the middle of a step if it seems to be hung up. All you can do is kill it. You won't get the current log if you kill a session, so problems can be hard to debug.

- % signs have a special meaning. If you want to define or call a macro as the first action, you will need to put in a comment first.

- There are no wizards to generate code for you, and there's no autocomplete in the editor. I don't think those things are impossible, they're just not there yet. It might be possible to

prompt for input using ipwidgets, but I have not looked into that. You'd have to learn Python, but you already need to learn additional languages to create tasks in SAS Studio or EG.

- Keyboard assignments are different from those in SAS for Windows or SAS Studio, and you will have to learn them, or reassign them, or use the Menu or Command Palette.

- Jupyter supports only one kind of output per code cell. SAS has two different kinds of output - log and list/results, but you can get only one of them to display automatically. See Appendix 1 for an example.

# Appendix 1 : Need to save and restore output

If there are no errors in the log, and ODS output was created, you won't see the log in Jupyter:

```
proc sgplot data=SASHELP.CLASS;
    heatmap x=Height y=Weight;
run;
```
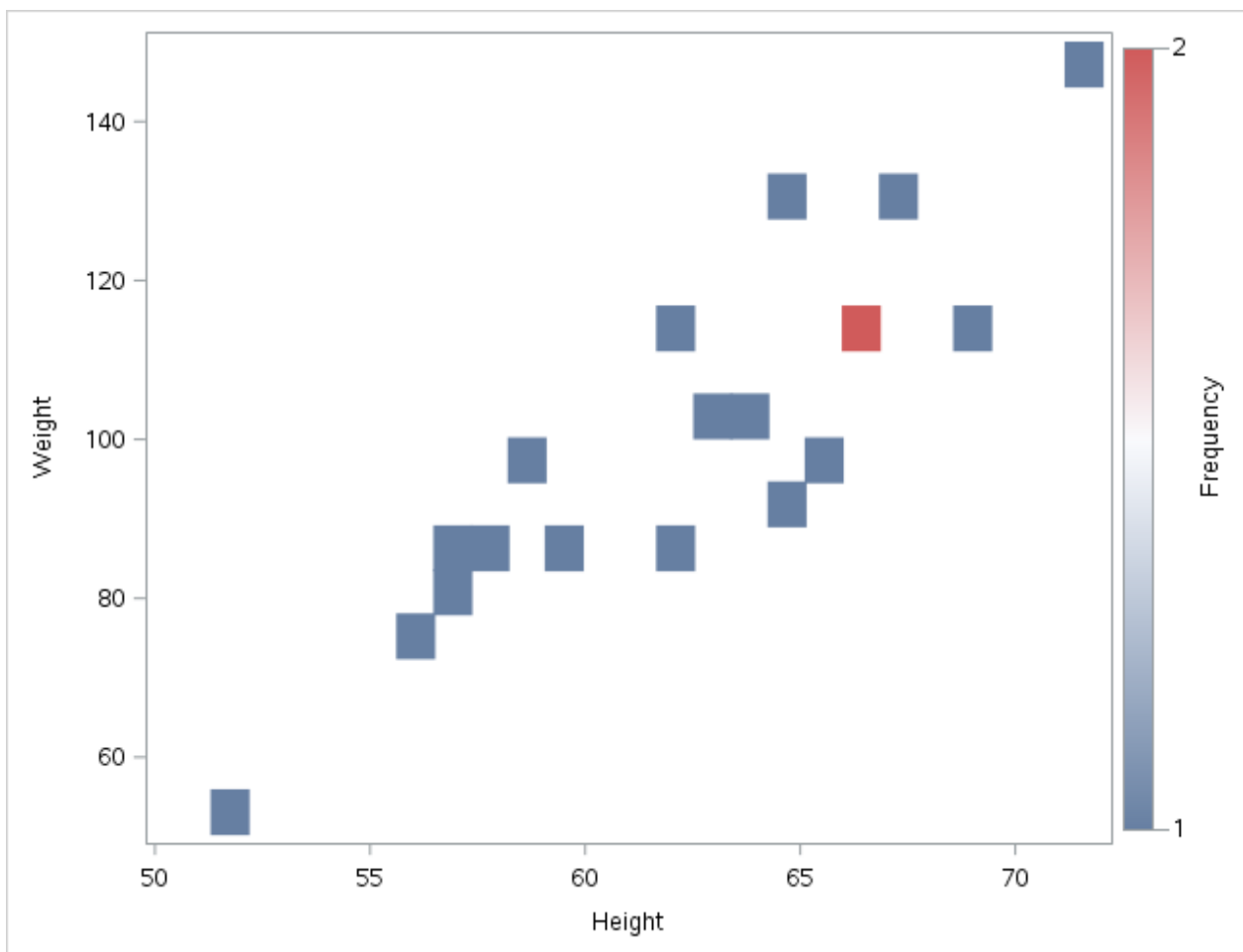


You won't see the log from that code cell, because it created output and there were no errors.. The designers of the SAS interface made the reasonable assumption that you don't need to see the log unless there was an error, so that's the behavior we get. If you want to see both the log and the output, you'll have to reroute the output and display it separately, as shown in Appendix 2.

Another way to get around this problem is to do all the data processing that doesn't produce output in one step, and isolate the code that produces output in a cell of its own.

# Appendix 2 : Show both log and output

You can turn ODS output on and off.

**Macros to control ODS output**

```
/* Macro to get list of open output destinations. */
%macro util_return_open_ods_dests(delim= |, showdests=1, returndests=1);

    %let data = %UPCASE(sashelp.vdest);

    %local dsid ;

    /* Open the data set.  */
    %let dsid = %SYSFUNC(open(&DATA., I));

    %if &DSID.=0 %then
        %do;
            %put ERR%str()OR: Error opening dataset &DATA..;
            %put %sysfunc(sysmsg());
            %return;
        %end;

    /* Get variable numbers */
    %local dest_var_num style_var_num ;

    %let dest_var_num = %sysfunc(varnum(&DSID. , DESTINATION));
    %if &DEST_VAR_NUM.= 0 %then
        %do;
            %put ERR%str()OR: Variable not in data set.;
            %put %sysfunc(sysmsg());
            %goto exit;
        %end;

    %let style_var_num = %sysfunc(varnum(&DSID. , STYLE));
    %if &STYLE_VAR_NUM.= 0 %then
        %do;
            %put ERR%str()OR: Variable not in data set.;
            %put %sysfunc(sysmsg());
            %goto exit;
        %end;

    /* Get the values. */

    %local fetch_rc name_value style_value ;

    %let fetch_rc = 0;

    %do %until (&FETCH_RC. ne 0);
        %let fetch_rc = %sysfunc(fetch(&DSID., noset));
```

```sas
        %if &FETCH_RC. ne 0 %then
            %goto exit;
        %let name_value = %sysfunc(getvarc(&DSID., &DEST_VAR_NUM.));
        %let style_value = %sysfunc(getvarc(&DSID., &STYLE_VAR_NUM.));
        %if &SHOWDESTS. %then
            %put NOTE: ODS Destination &NAME_VALUE. is open with style
&STYLE_VALUE.;
        %if &RETURNDESTS. %then
            %do;
            &DELIM.&NAME_VALUE.
        %end;
    %end;

%EXIT:

    %local close_rc ;

    %let close_rc = %SYSFUNC(close(&DSID.));

    %if &CLOSE_RC. ne 0 %then
        %do;
            %put ERR%str()OR: Error closing dataset &DATA..;
            %put %sysfunc(sysmsg());
        %end;

    %return;

%mend util_return_open_ods_dests;


/* Macro to send default output to ODS document */
%macro util_reroute_ods_to_document(name=work.__thisdoc, delim=|);

    %local i outcount outdest outlist;

    %let name = %upcase(&NAME.);

    %put NOTE: ODS Output will be rerouted to ODS Document &NAME. ;

    %let outlist = %util_return_open_ods_dests(delim=&DELIM.);

    %let outcount = %sysfunc(countw(%superq(outlist), %superq(delim)));

    %if &OUTCOUNT. = 0 %then
        %do;
        %put NOTE: No ODS Destinations active.;
        %return;
        %end;

    %do i = 1 %to &OUTCOUNT.;
        %let outdest = %scan(%superq(OUTLIST), &I., %superq(DELIM));
        %if "&OUTDEST." ne "DOCUMENT" %then
            %do;
                %put NOTE: ods &OUTDEST. exclude all;
                ods &OUTDEST. exclude all;
            %end;
    %end;
```

```
    %put NOTE: ods document name=&NAME. (write);
    ods document name=&NAME. (write);

%mend util_reroute_ods_to_document;

/* Macro to replay ODS document. */
%macro util_replay_ods_document(name=work.__thisdoc, delim=|);

    %local i outcount outdest outlist;

    %let name = %upcase(&NAME.);

    %put NOTE: ODS Output will be reprinted from ODS Document &NAME. ;

    %let outlist = %util_return_open_ods_dests(delim=&DELIM.);

    %let outcount = %sysfunc(countw(%superq(outlist), %superq(delim)));

    %if &OUTCOUNT. = 0 %then
        %do;
        %put NOTE: No ODS Destinations active.;
        %return;
        %end;

    %do i = 1 %to &OUTCOUNT.;
        %let outdest = %scan(%superq(OUTLIST), &I., %superq(DELIM));
        %if "&OUTDEST." ne "DOCUMENT" %then
            %do;
                %put NOTE: ods &OUTDEST. select all;
                ods &OUTDEST. select all;
            %end;
    %end;

    ods document close;

    proc document name=&NAME.;
        replay;
        run;
    quit;

%mend util_replay_ods_document;
```

**Use %util_reroute_ods_to_document() macro to reroute output**

```
/* No log was produced above.  If you want both log and output, reroute the output
and play it back separately. */

%util_reroute_ods_to_document();

proc sgplot data=SASHELP.CLASS;
    heatmap x=Height y=Weight / name='HeatMap';
    gradlegend 'HeatMap';
run;
```

```
   /* Log will appear below */
```

```
542         /* No log was produced above.  If you want both log and output,
reroute the output and play it back separately. */
543
544         %util_reroute_ods_to_document();
NOTE: ODS Output will be rerouted to ODS Document WORK.__THISDOC
NOTE: ODS Destination HTML5(SASPY_INTERNAL) is open with style HTMLBlue
NOTE: ods HTML5(SASPY_INTERNAL) exclude all
NOTE: ods document name=WORK.__THISDOC (write)
545
546         proc sgplot data=SASHELP.CLASS;
547             heatmap x=Height y=Weight / name='HeatMap';
548             gradlegend 'HeatMap';
549         run;

NOTE: PROCEDURE SGPLOT used (Total process time):
      real time           0.02 seconds
      cpu time            0.02 seconds
      memory              787.40k
      OS Memory           25,776.00k
      Timestamp           2019-08-29 14:07:16

NOTE: There were 19 observations read from the data set SASHELP.CLASS.

550
551         /* Log will appear below */
552
```
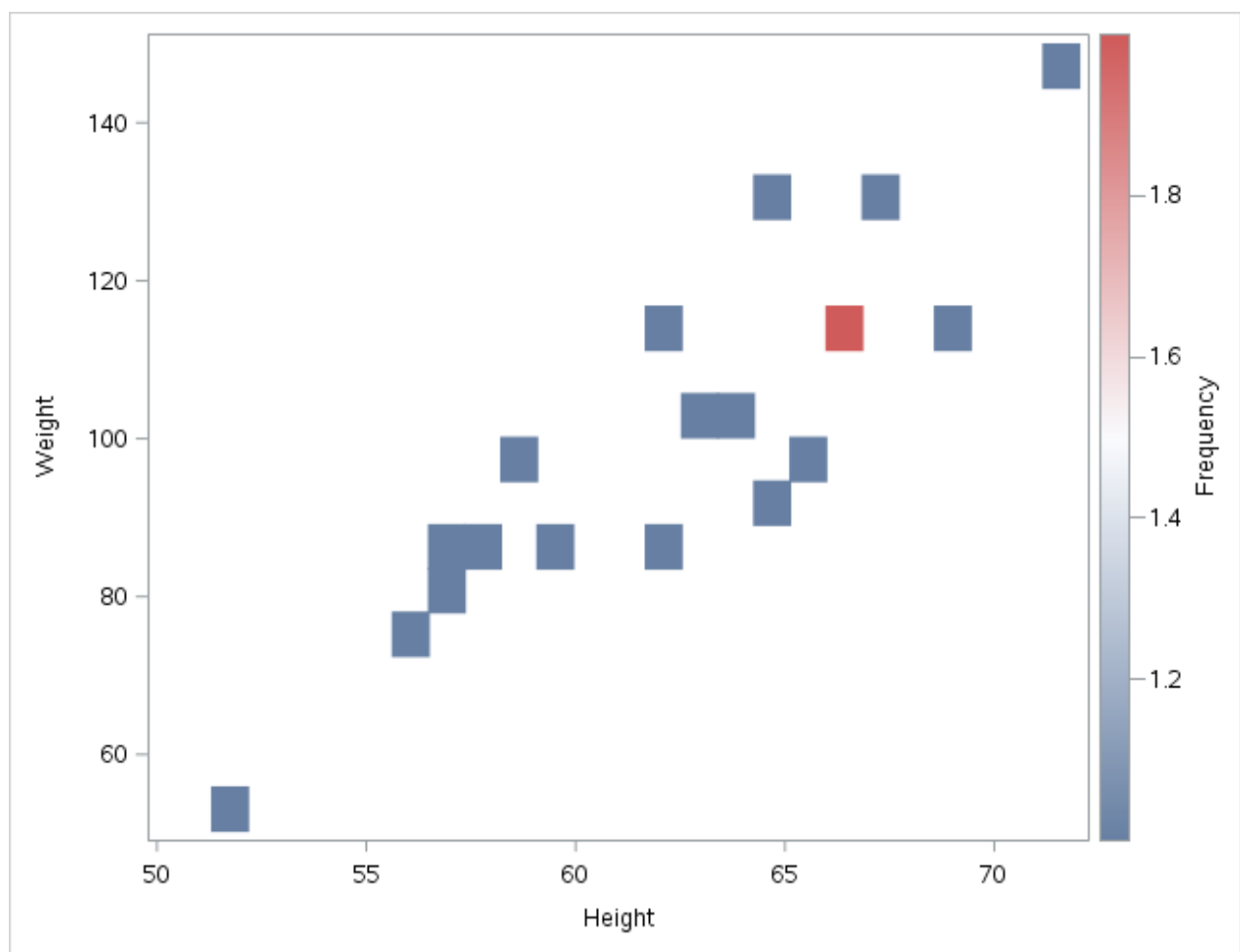
**Use %util_replay_ods_document() to display the output**

```
  /* Replay the output */
  %util_replay_ods_document();
```

# Appendix 3: Installation

Jupyter is basically a big Python application. I run it on Windows in Anaconda, which is a pre-packaged installation of Python and various tools. Jupyter is one of those tools.

Anaconda has an interactive prompt, which you can use to start Jupyter, Python, or other programs. After the initial installation, you won't need the use the prompt except to run nbdime.

If you're lucky, someone in your IT department will set this up for you. But if not, you can set it up yourself. All programs can be installed in single-user mode, and so do not need admin rights.

I didn't mention this earlier, but there are actually 3 different versions of Jupyter Notebooks: the tradition notebook, which has the most features but is deprecated, and I have found it to be unstable; Jupyter Lab, supposedly the way of the future, with many fewer extensions but a cleaner interface, and it's what comes with SAS University Edition; and Jupyter Hub, a multi-user system which I have not used. There's also another notebook implementation named Zeppelin that supports SAS.

The instructions below are not guaranteed to work. You may have to experiment.

**Anaconda installation**

- Install anaconda (not miniconda) from https://www.anaconda.com/distribution/

Anaconda includes Jupyter Notebook and Jupyter Lab. The commands below are to be typed at the Anaconda Prompt, which will appear as an option in your Start Menu.

- Update one of the packages to speed things up a bit

```
conda update conda-build
```

-From the Anaconda prompt, open a Jupyter notebook to make sure it was installed correctly. It will have only a Python kernel at this point. After it opens you can close it again.

```
jupyter notebook
```

- Do the same for Jupyter lab

```
jupyter lab
```

**SAS Kernel Installation**

Jupyter uses software components called *kernels* to run programs. Python comes built in, but you (or your administrator) will have to add the SAS kernel yourself.

https://communities.sas.com/t5/SAS-Communities-Library/Installing-SASpy-Kernel-on-Jupyter-Notebooks/ta-p/464873

https://sassoftware.github.io/sas_kernel/install.html

To install the SAS kernel for Jupyter, type these commands at the Anaconda prompt:

```
pip install sas_kernel
conda install -c conda-forge saspy
jupyter kernelspec list
```

The last command should show sas as a kernel.

**Extensions**

- Install Jupyter Notebook extensions and see what you've got:

```
conda install -c conda-forge jupyter_contrib_nbextensions
jupyter nbextension install --py sas_kernel.showSASLog
jupyter nbextension enable sas_kernel.showSASLog --py
jupyter nbextension install --py sas_kernel.theme
jupyter nbextension enable sas_kernel.theme --py
jupyter nbextension list
```

I can't recommend using the traditional Jupyter notebooks, but you may have a need to, and it doesn't hurt to have the capability.

- Follow these instructions to enable Lab extensions. They might already be enabled:

https://jupyterlab.readthedocs.io/en/stable/user/extensions.html

- Install prerequisite for Jupyter Lab extensions and the table of contents extension, then see what's there. The table of contents extension might already be installed.

```
conda install -c conda-forge nodejs
jupyter labextension install @jupyterlab/toc
jupyter labextension list
```

You might need to enable extensions before you can use them. Open the command palette (it's the icon that looks like a palette) and type *extension*; if extensions are turned off you will be able to turn them on here.

- Install nbdime / nbdiff for difference listings

https://nbdime.readthedocs.io/en/latest/ https://github.com/jupyter/nbdime
https://nbdime.readthedocs.io/en/latest/cli.html

```
pip install nbdime
jupyter serverextension enable --py nbdime --user

jupyter nbextension install --py nbdime --user
jupyter nbextension enable --py nbdime --user


jupyter labextension install nbdime-jupyterlab
```

The Jupyter interface is designed to compare two versions of the same file, either the current and a checkpoint, or the current and a Git version. To compare two notebooks from the Anaconda command line, run:

```
nbdiff-web notebook1.ipynb notebook2.ipynb
```

https://nbdime.readthedocs.io/en/latest/

- You can also use the extension manager to install extensions.

Here's what I currently have installed:

SEARCH

INSTALLED

**@jupyterlab/celltags**
Extension for adding descriptive tags to notebook cells (Beta)

Uninstall
Disable

**@jupyterlab/git**
A JupyterLab extension for version control using git

Uninstall
Disable

**@jupyterlab/toc**
Table of Contents extension for JupyterLab

Uninstall
Disable

**@agoose77/jupyterlab-attachments**
A JupyterLab extension to add support for attachments in the JupyterLab editor.

Uninstall
Disable

**@haidark/experiments_ext**
Define cell execution orders for different experiments in a single notebook

Uninstall
Enable

**jupyterlab-imgclip**
Paste images from clipboard

Uninstall
Disable

**nbdime-jupyterlab**
A JupyterLab extension for showing Notebook diffs.

Uninstall
Disable

DISCOVER

I disabled the experiments_ext extension for this presentation, but you might want to look at the documentation to see if it's something you want.

**LaTex**

If you aren't using a separate markdown editor, you might need to install LaTex to print PDFs of your notebooks.

Issue the following commands:

```
conda install -c conda-forge miktex
```

The first time you try to use LaTeX to print to PDF, you will be prompted to install additional packages.

**Passwords**

For information about Jupyter passwords, see:

https://jupyter-notebook.readthedocs.io/en/stable/security.html

I hard-code a userid and encoded password in my configuration file, but you might not want to do that.

**Configuration file**

All of the instructions here will **not** work for you, but they may be of help.

- Open Jupyter and create a new Python notebook containing the statements

```
import saspy
saspy
```

Run the code. You will get a message like

```
<module 'saspy' from
'C:\\Users\\c449630\\AppData\\Local\\Continuum\\anaconda3\\lib\\site-
packages\\saspy\\__init__.py'>
```

(with double slashes reduced, that's
C:\Users\c449630\AppData\Local\Continuum\anaconda3\lib\site-packages\saspy\ )

"c449630" is my Windows userid. Substitute yours wherever it occurs.

Copy the file sascfg.py in that directory into sascfg_personal.cfg, and make the necessary changes for your system. I can't help you with that in any way, shape, or form.

Where I work, running Jupyter under Windows 10 with a Unix grid server, this sascfg_personal.py file does the needful. If you have a standard SAS install, the directory names will be similar, but the SAS version number is built into some of the directory names.

```python
SAS_config_names = ['winiomsolaris']


SAS_config_options = {'lock_down': False,
                      'verbose'  : True
                     }

SAS_output_options = {'output' : 'html5'}

# build out a local classpath variable to use below for Windows clients
cpW  =  "C:\\Program
Files\\SASHome\\SASDeploymentManager\\9.4\\products\\deploywiz__94494__prt__xx__sp
0__1\\deploywiz\\sas.svc.connection.jar"
cpW += ";C:\\Program
Files\\SASHome\\SASDeploymentManager\\9.4\\products\\deploywiz__94494__prt__xx__sp
0__1\\deploywiz\\log4j.jar"
cpW += ";C:\\Program
Files\\SASHome\\SASDeploymentManager\\9.4\\products\\deploywiz__94494__prt__xx__sp
0__1\\deploywiz\\sas.security.sspi.jar"
cpW += ";C:\\Program
Files\\SASHome\\SASDeploymentManager\\9.4\\products\\deploywiz__94494__prt__xx__sp
0__1\\deploywiz\\sas.core.jar"
cpW += ";C:\\Users\\c449630\\AppData\\Local\\Continuum\\anaconda3\\lib\\site-
packages\\saspy\\java\\saspyiom.jar"

# And, if you've configured IOM to use Encryption, you need these client side
jars.
cpW += ";C:\\Program
Files\\SASHome\\SASVersionedJarRepository\\eclipse\\plugins\\sas.rutil_904500.0.0.
20170816190000_v940m5\\sas.rutil.jar"
cpW += ";C:\\Program
Files\\SASHome\\SASVersionedJarRepository\\eclipse\\plugins\\sas.rutil.nls_904500.
0.0.20170816190000_v940m5\\sas.rutil.nls.jar"
cpW += ";C:\\Program
Files\\SASHome\\SASVersionedJarRepository\\eclipse\\plugins\\sastpj.rutil_6.1.0.0_
SAS_20121211183517\\sastpj.rutil.jar"

winiomsolaris = {'java'    : 'java',
                 'iomhost'   :
['pdorsasgrid01.kaiser.org','pdorsasgrid02.kaiser.org','pdorsasgrid03.kaiser.org',
 'pdorsasgrid031x.kaiser.org'],
                 'iomport'   : 8591,
                 'classpath' : cpW ,
                 'encoding'  : 'latin1'
                 }

import os
os.environ["PATH"] += ";C:\\Program
Files\\SASHome\\SASFoundation\\9.4\\core\\sasext\\sspiauth.dll"
```

```
os.environ["PATH"] += ";C:\\Program
Files\\SASHome\\SASFoundation\\9.4\\core\\sasext\\"
```

- Close and reopen Jupyter

-- Create a new notebook. Change the kernel type to sas. In the notebook cell, enter:

```
%put &=SYSPROCESSID. ;
```

-- Run the cell. You should be prompted for your userid and password, and after a pause you should see the SAS log for the code you just ran.

# Appendix 4 : Elementary markdown

Here's a pretty good basic reference, which also contains links to more advanced markdown.

Remember that markdown comes in various flavors, so you will have to experiment to discover which work in your environment.

Basic Syntax | Markdown Guide: https://www.markdownguide.org/basic-syntax/

Another reference, in case you don't like the style of the first one:

https://github.com/adam-p/markdown-here/wiki/Markdown-Here-Cheatsheet