presentation

December 3, 2021

1 Dynamic Documents with Jupyter Notebooks

1.1 Before We Begin

- Special thanks to Oscar Barriga Cabanillas for helping out today
- Thank you for BITSS for organizing
- How has it been so far?
- A Question:
  - How familiar are you with python? Jupyter?
  - What are you thinking of getting out of this talk?

1.2 How to watch this presentation:

- Either:
  - `git clone` the repository
  - Start up jupyter lab
  - open `presentation.ipynb`
  - follow along
- Or:
  - Go to the repository and press the `Launch Binder` Button
  - To be explained later.

1.3 Why Dynamic Documents?

Dynamic Documents are a part of the bigger picture of Reproducible Science. Sure, there is a fixed cost; **BUT**, they make my life easier in these ways:

- Short term: Easier to document fresh out of the oven results
- Medium term: Fast, reliable and tractable new results
- Long term: You can see how everything was created

1.4 What are Dynamic Documents?

Based on principles of literate programming, we aim at combining code and paper in one single document

- Best framework to achieve the holy grail of one-click reproducible workflow
- Best two current implementations: RMarkdown (R) & Jupyter (Python).
- Stata is catching up: We will come back to this in a second

1.5 The State of Things Now

Currently, the code and the narrative components live in separate universes

1.6 Part of Larger Workflow

- Dynamic documents are best used as part of a larger organized workflow
  - Structuring folders: Data, analysis, output
  - Documenting code
  - Combining both into a final document: Pre analysis or final paper

1.7 Markdown's Entrance

- In terms of writing the "paper"/documentation part of dynamic documents, there are many solutions
  - Latex, HTML, RST (ReStructured Text)
- But most have honed in on using Markdown
  - Markdown is an easy way to write formatted text in a plain text format
  - But without as verbose and difficult of a syntax like latex/HTML
- Although basic markdown has the basics for formatting, creating tables, adding figures
- We will use Pandoc, which is used in both the Stata and R sessions

1.8 Markdown Cheatsheet

There are loads of markdown cheatsheats on the web. One can be found here

1.9 What is Pandoc?

- Pandoc is sort of what it says: pan (all), doc (document)
- It's a way to convert between and across different file formats
  - Word -> HTML
  - Latex -> Markdown
  - HTML -> XML
  - Anything to anything
- See Pandoc's website for all input and output filetypes

1.10 The Magic of Pandoc

- Pandoc and Markdown allows you to create one file that can then be used in many different places
- Example:
  - You're writing your CV and want to put it up in various places.
  - Your website needs HTML
  - One job posting allows PDF
  - One job posting only allows Word
- Ordinarily, you would need to have three versions, Word, HTML, PDF
  - This might get unruly as you change one but forget to change the other
  - What if there's another file format you might need?
- With Pandoc and markdown, you would:
  - write your CV in markdown

  - convert to PDF, Word and PDF with pandoc

1.11 What are Jupyter Notebooks?

- A way to do literate programming and dynamic documents
- Provide code and writing/analysis, on a language agnostic platform
  - Meaning that it is not restricted to just one language
  - Currently there are so-called kernels for many languages
  - Including Stata, Python, R, C, Golang, C++, Fortran and more coming!
- Uses the power of Markdown/Latex Math and Code to tell a story and provide an efficient workflow
- Convert into several different formats including Latex, HTML, Presentations etc…
- The Jupyter server is also available in other text editors such as Atom and VS Code.
- And now available in STATA!

1.12 Under the Hood

- You can think of Jupyter as broadly being made up of two parts:
  - A JSON document that organizes text between markdown, code, figures, widgets, etc…
  - A server that loads a "kernel" with a particular language and knows how to translate the markdown to formatted text and the code to execution
  - A web interface (although not required)

1.13 Why Jupyter Notebooks?

- Jupyter is ubiquitous
- Jupyter is used by basically all of the data science community
- Jupyter is used by other software (VS Code, Atom/Hydrogen)
- Since Jupyter is a JSON document and built using web tools, anything that uses webtools can use it
- Science and publishing is changing (PDFs are becoming old, open access and web journals are becoming more popular)
  - Present results in a dynamic way
  - Interactive
  - More efficient to show quick interactive widget to experiment with colleagues/advisors than 50 figures in a static PDF

1.14 Extensions

JupyterLab (the web interface) comes with many extensions for anything you might want:

- A language server
- multicursor support
- git integration
- and more…

1.15 Running Code

[ ]:
```
import pandas as pd

df = pd.DataFrame({'hhid' : range(5), 'income' : [2,6,2,7,8]})

df.groupby('hhid').mean()
```

[ ]:
```
      income
hhid
0          2
1          6
2          2
3          7
4          8
```

1.16 Kernel Magics

- Many Jupyter kernels have something called magics
  - A way to make certain actions easy without having to write too much code
  - Often language specific
    - A few examples from python:
      - support for command line with `!`
      - `conda` installing with `%conda`
      - Calling other languages, like Stata and R
      - A full list can be found here

1.17 An Example with `%conda`

- Suppose you want to install a package into python
- you can use the `%conda` for a conda environment or `%pip` for a regular Python installation

[ ]:
```
%pip install seaborn
```

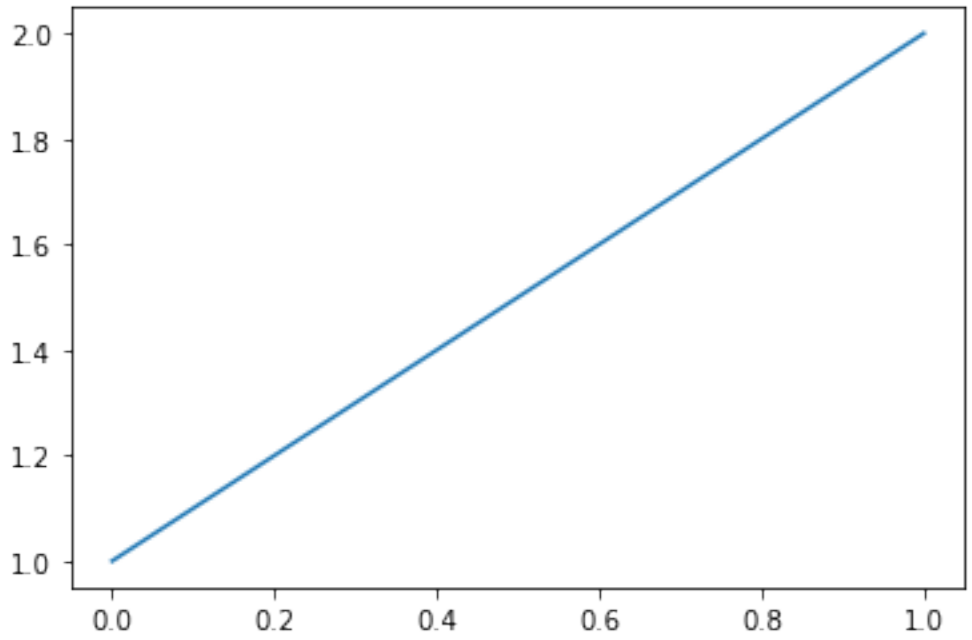
```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: seaborn in /usr/lib/python3.8/site-packages
(0.10.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/lib/python3.8/site-packages
(from seaborn) (1.19.1)
Requirement already satisfied: scipy>=1.0.1 in /usr/lib/python3.8/site-packages
(from seaborn) (1.5.2)
Requirement already satisfied: pandas>=0.22.0 in /usr/lib/python3.8/site-
packages (from seaborn) (1.1.1)
Requirement already satisfied: matplotlib>=2.1.2 in /usr/lib/python3.8/site-
packages (from seaborn) (3.3.1)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/lib/python3.8/site-packages (from pandas>=0.22.0->seaborn) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /usr/lib/python3.8/site-packages
(from pandas>=0.22.0->seaborn) (2020.1)
```

```
Collecting certifi>=2020.06.20
  Downloading certifi-2020.6.20-py2.py3-none-any.whl (156 kB)
     |                | 156 kB 2.1 MB/s eta 0:00:01
Requirement already satisfied: cycler>=0.10 in /usr/lib/python3.8/site-
packages (from matplotlib>=2.1.2->seaborn) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/lib/python3.8/site-
packages (from matplotlib>=2.1.2->seaborn) (1.2.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/lib/python3.8/site-packages
(from matplotlib>=2.1.2->seaborn) (7.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/usr/lib/python3.8/site-packages (from matplotlib>=2.1.2->seaborn) (2.4.7)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.8/site-packages
(from python-dateutil>=2.7.3->pandas>=0.22.0->seaborn) (1.15.0)
Installing collected packages: certifi
Successfully installed certifi-2020.6.20
Note: you may need to restart the kernel to use updated packages.
```

1.18 Figures

[ ]:
```
import matplotlib.pyplot as plt

plt.plot([1,2]);
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f32ea17c6d0>]
```



1.19 Exporting

- Exporting to HTML, PDF and slides is possible through the menu: File -> Export Notebook as -> PDF

1.20 Extras

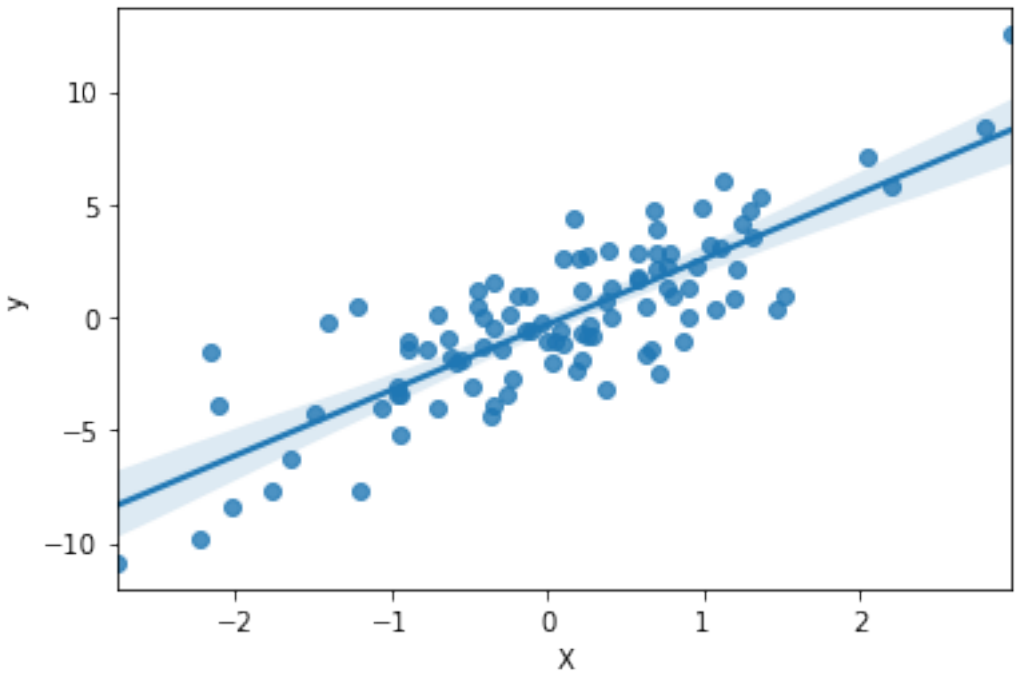
- Now that we have the basics down, let's take a few minutes for a break, as well as to ask questions
- When we come back, we'll talk about some advanced techniques:
  - Interactive widgets using `ipywidgets`
    - One step closer to dashboarding: `plotly`
    - Super dashboarding: `dash`
  - Using Jupyter notebooks in Binder and Github
  - Port-forwarding and setting up Jupyter remotely
  -

1.21 `ipywidgets` in Jupyter

- `ipywidgets` provides an easy way to create interactive widgets (python only)
- Example: Let's say you want to see how more noise in a linear model changes your regression line
  - This involves creating some fake data with an increasing standard deviation, running a regression, plotting the results
  - Ordinarily, in a static file, you have a PDF and maybe you have to get creative with plots you make, or show many plots to convey the information
    - Or perhaps the plot has to be very "busy" and complex

[ ]:
```
from ipywidgets import interact
import pandas as pd
import seaborn as sns
import numpy as np

N=100
sigma=2

df = pd.DataFrame({'eps' : np.random.normal(0, sigma,N),
                   'X' : np.random.normal(0,1,N)})

df_ready = df.assign(y = lambda df: 3*df['X'] + df['eps'])

# Get some data
sns.regplot(x = 'X', y = 'y', data = df_ready);
```

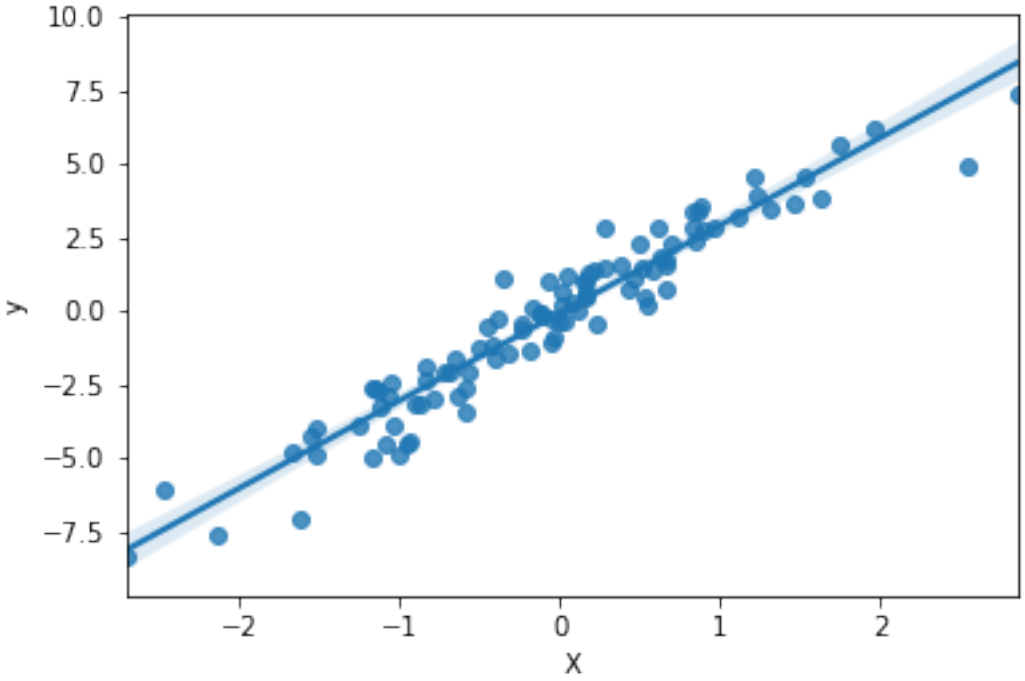
[ ]:
```
<AxesSubplot:xlabel='X', ylabel='y'>
```

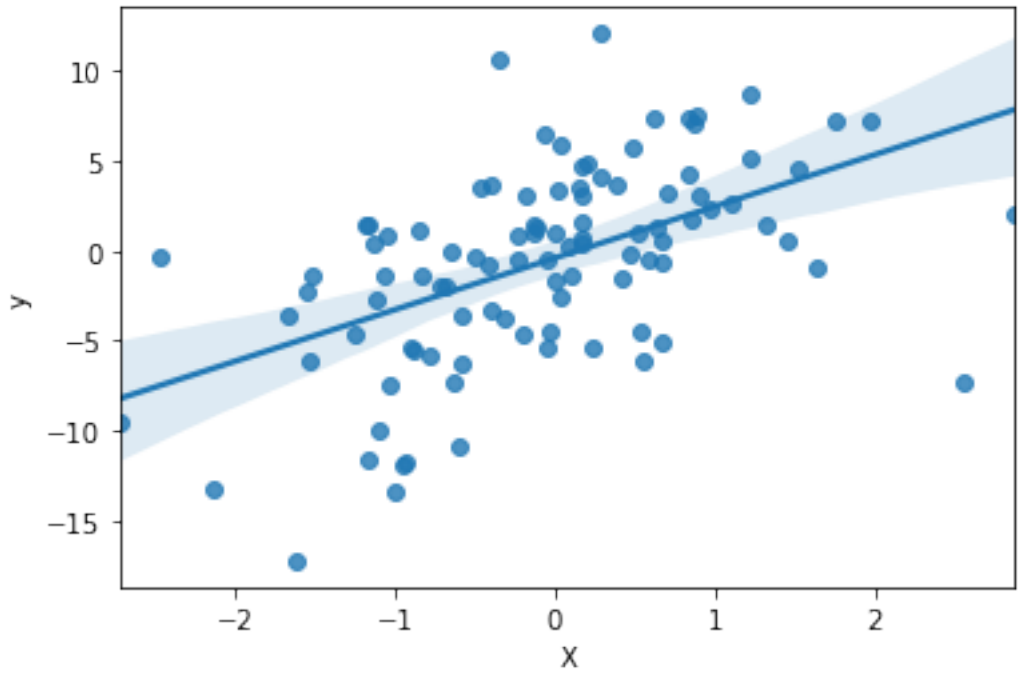


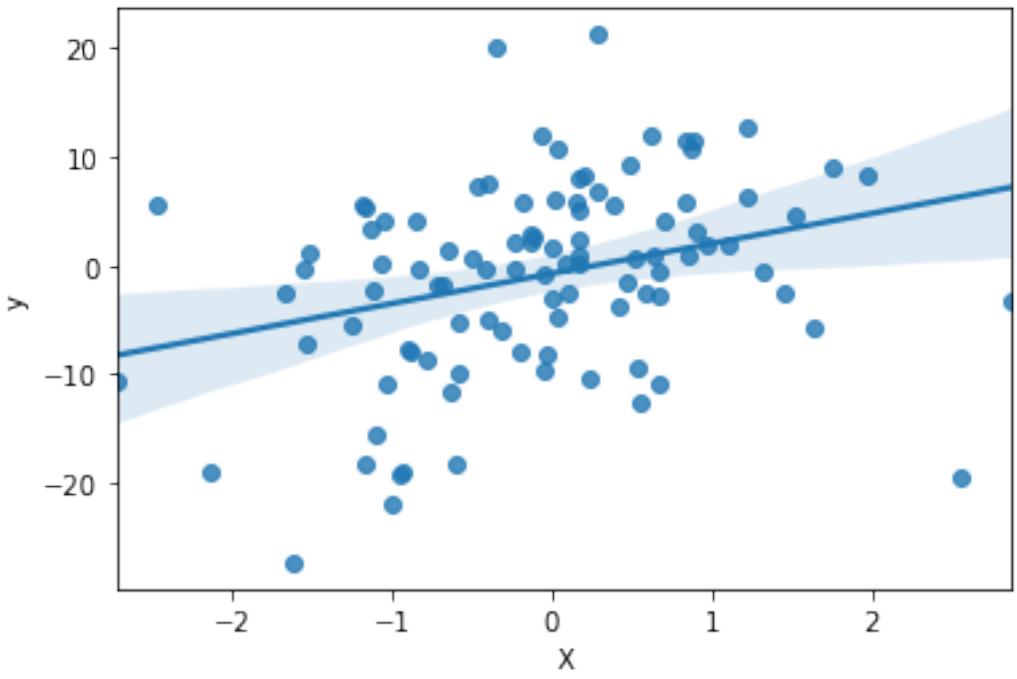
1.22 GREAT!

- But what if we wanted to see how things changed if we increased σ? or if we increased N?
- We can try making a plot for every possibility:

[ ]:
```
import matplotlib.pyplot as plt

for i in np.linspace(1,10,3):
    plt.figure()
    rng = np.random.default_rng(1)
    df = pd.DataFrame({'eps' : rng.normal(0, i,N),
                       'X' : rng.normal(0,1,N)})

    df_ready = df.assign(y = lambda df: 3*df['X'] + df['eps'])

    sns.regplot(x = 'X', y = 'y', data = df_ready)
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
y
−2
−1
0
1
2
X


10
5
0
−5
−10
−15
y
−2
−1
0
1
2
X



1.23 Or...

- We can make it interact...

[ ]:
```
@interact(sigma = (1,20,1), N=(1,1000,100))
def r_plot(sigma, N):

    rng = np.random.default_rng(1)
    plt.figure()
    rng = np.random.default_rng(1)
    df = pd.DataFrame({'eps' : rng.normal(0, sigma,N),
                  'X' : rng.normal(0,1,N)})

    df_ready = df.assign(y = lambda df: 3*df['X'] + df['eps'])

    sns.regplot(x = 'X', y = 'y', data = df_ready)
```

```
interactive(children=(IntSlider(value=10, description='sigma', max=20, min=1),␣
↪IntSlider(value=401, descriptio…
```

1.24 Adding More Interaction

- We can also interactivity to our plots, themselves with `plotly`

[ ]:
```
# Make interactive Plot
@interact(sigma = (1,20,1), N=(1,1000,100))
def r_plot(sigma, N):

    rng = np.random.default_rng(1)
    plt.figure()
    rng = np.random.default_rng(1)
    df = pd.DataFrame({'eps' : rng.normal(0, sigma,N),
                  'X' : rng.normal(0,1,N)})

    df_ready = df.assign(y = lambda df: 3*df['X'] + df['eps'])

    fig= px.scatter(df_ready, x="X", y="y", trendline="ols")
    fig.show()
```

```
interactive(children=(IntSlider(value=10, description='sigma', max=20, min=1),␣
↪IntSlider(value=401, descriptio…
```

1.25 Creating a Dashboard

- It doesn't end there, we can also develop whole dashboard for interactivity:

https://mybinder.org/v2/gh/plotly/jupyter-dash/master?urlpath=tree/notebooks/getting_started.ipynb

1.26 Binder and Hosting Notebooks

- Jupyter is great and all, but what if you advisor/boss doesn't have Jupyter installed?
- What if they do, but they don't have all the dependencies needed for your cool dashboard?
- That's where binder comes in
- Binder uses `docker` to create a containerized version of your notebook with all dependencies installed and anyone can access it even if they don't have jupyter installed at all.
- They just need a web browser
- We've already seen how this works, either with the dashboard or with this very presentation!
- All you need is a public github repository and notebook in that repository and that's it!
- Waiting time for spinning up the notebook will vary
- While it's spinning it up, any questions so far?

1.27 Port-forwarding and setting up Jupyter to work on a server

- Many people might have servers in their universities/organizations that are more powerful than a laptop.
- Jupyter allows the ability to run a notebook locally (on your laptop screen), but using the power of the server.
  - This requires jupyter being installed on the server
  - This isn't a difficult thing to do for a sysadmin, so it's worth finding out whether that's possible

1.27.1 Setting up jupyter on a server

- The first thing you need to do is log on to the server and start a jupyter instance:

`jupyter notebook --no-browser --port=8888`

- This tells the server to start an instance of jupyter, without a browser (we won't need it, nor can a server open up a browser window), in port 8888 (this will be important later)
- For Mac users, you can use `ssh` to finish the process. Just type: `ssh username@host -L 8888:localhost:8888`
- Which will forward your computer 8888 port, to the server's 8888 port.
- For Windows, ssh also exists, but you will need to enable it.
  - head to Settings > Apps and click "Manage optional features" under Apps & features.
  - Click Add a Feature, and find OpenSSH
- Then use the same command as for Macs: `ssh username@host -L 8888:localhost:8888`
- Then go to your browser:
  - `localhost:8888` and you should be taken to a Jupyter page and prompted for a token.
  - You can find this token in the window where you started Jupyter on the server
    - Copy and paste this token into the prompt, and VOILA!
- Now you have Jupyter running on your computer's browser window, but with the power of the server!

1.28 Taking all this to Pure Text

- One drawback of jupyter (besides dependencies) is the fact that you need to install all of this and use a browser
- Not very good for version control
- `Rmd` files (used with RMarkdown) are just markdown files with code cells.
  - Great for version control
  - Readble
  - Easy to share and read in its raw form
- Can we do this with Jupyter?
  - Yes! with `jupytext`

1.29 Other Extras not Covered

- `jupyter-cache`
- `codebraid`