

Winter Institute in Data Science and Big Data

Comparative Computing

Le Bao

Massive Data Institute, Georgetown University

January 7, 2022

Plan


- Python
 - Basics
 - Data structures
 - Application: working with web data
- Comparative computing
 - Python, R, Shell
 - Polyglot programming and computing tools
 - Computing environment
- Containers and cloud computing
 - Operating systems and system dependencies
 - Docker
 - Cloud computing with Code Ocean

Data Science Toolbox

- Some most used DS tools:

- Apache Spark
- BigML
- D3.js
- MATLAB
- Excel
- tidyverse
- Tableau
- Jupyter
- ggplot2
- Matplotlib
- NLTK
- Scikit-learn
- TensorFlow
- Weka
- ...

Calculating Path to Jupyter Using Excel




Search qu

MARKETS BUSINESS INVESTING TECH POLITICS CNBC TV WATCHLIST CRAMER PRO





INVESTING IN SPACE

NASA spacecraft launches toward Jupiter asteroids on an intricate path charted by Excel

PUBLISHED MON, OCT 18 2021-12:40 PM EDT | UPDATED MON, OCT 18 2021-8:12 PM EDT

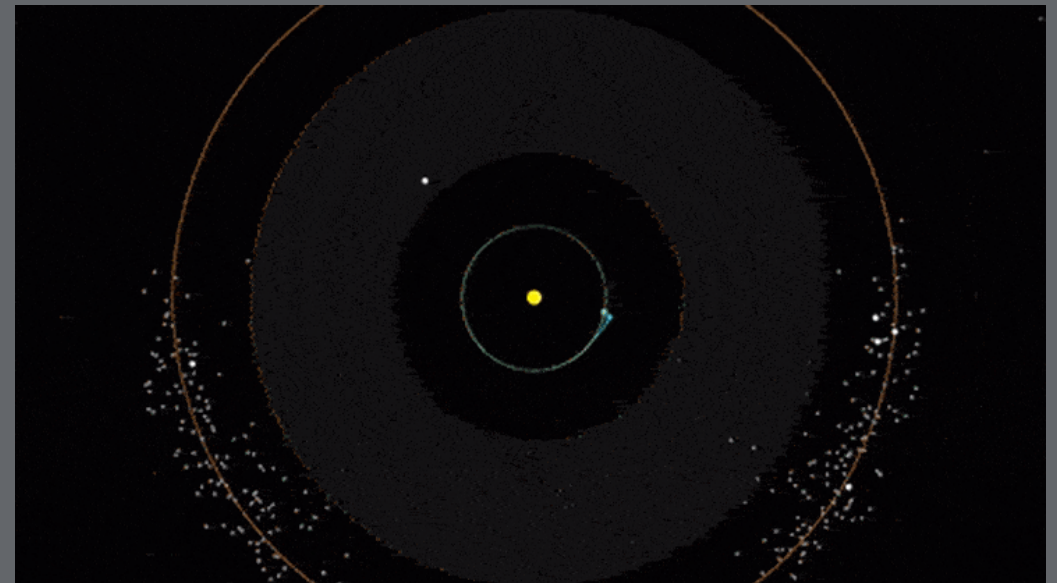


Michael Sheetz
@THESHEETZTWEETZ

SHARE    

KEY POINTS

- NASA's latest exploration spacecraft launched over the weekend, beginning a 12-year journey to visit Jupiter's Trojan asteroids.
- The mission's intricate path was charted using a tool familiar to Wall Street – Microsoft Excel.
- Lockheed Martin mission architect Brian Sutter explained to CNBC how he used Excel to find which asteroids would be worth visiting, saying he aimed to try “to try to look at as many of these Trojans as we can in a single mission.”



Knowing Your Tools

- Recall that data science is an intersection of
 - statistical practice
 - computational tools
 - substantive knowledge
- The **interdisciplinary nature** of data science
 - Different kinds of statistical practices, computational tools, and substantive focuses
- Goal:
 - Know what you are doing: statistical and substantive knowledge
 - Know how to use your tools
 - Pick up new tools fairly quickly

R vs Python

- Let's forget about Stata.
- **Open-source** programming language
- High-level, expressive, front-end
 - Both were (partially) written in C
 - Both were Turing complete
 - Fewer lines of code to achieve complex functions
 - Closer to human languages
 - Slow (relatively)
- Vibrant online communities
 - Libraries, support, new things, etc.

R vs Python

- **R**

- Statistical analysis
- Primarily procedural and functional programming, but can also do OOP
- Academics, researchers, data scientists
- Easier to get started but some rough curves on the way
- Good at data wrangling, exploration, modeling, and visualization
- Can do data scraping, machine learning (deep learning)
 - `caret`, `tidymodels`, `keras`
- Fewer packages, slow adaptation beyond statistical methods

- **Python**

- General purpose
- Procedural, object-oriented, and functional programming
- Programmers, developers, engineers
- Deeper curve for beginners but smooth, linear later
- Good at data collection, interface with different types of data, machine learning (deep learning)
- Can do data wrangling, modeling, and visualization
 - `NumPy`, `Pandas`, `Seaborn` / `Matplotlib` / `Plotly`
- Fewer package on modeling (esp. specific models)

R vs Python

- There is *no* programming language wars
- Both can achieve what you want: “Turing complete”
 - Sometimes, one is easier than the other.
- Learn both!
 - Use one when appropriate and to your advantage!

Using Both: Polyglot Programming

- Integrating different languages: wrapper (interface)
- Tools and facilities in programming
 - IDE, code/text editor, interpreter/console
 - Command line and shell
 - Interactive vs non-interactive programming
- Quarto

Wrapper

- **Wrapper:** interface with other languages
- R interface for python: `reticulate`
- Python interface for R: `rpy2`
- R interface for C++: `Rcpp`
- Python interface for C/C++: Python C-API

Wrapper: Example

- Running Python code in R

```
#install.packages("reticulate")
library(reticulate)
# Specify python location and version
use_python("/usr/bin/python3")
# Run one line of python code
py_run_string("print 'Hello World'")
# Install python library
py_install("pandas")
# Use python in R
np <- import("numpy", convert = TRUE)
np1 <- np$array(c(1:4))
np1
# Source python script
source_python("somepythoncode.py")
```

Tools

- **Integrated Development Environment (IDE)**
 - R: RStudio (dominant), but there are others (VSCode)
 - Python: Spyder, PyCharm, VSCode, and lots of others
- What does it mean by *integrated*?
 - Text/code editor
 - Interpreter/console
 - Other tools: Git, debugging tool, file manager, viewer, etc.

RStudio as an example

RStudio interface showing a script, console output, and a boxplot.

Script: flights-example.R

```
1 library(nycflights13) ## package containing flights dataset
2 library(lubridate)
3 library(dplyr)
4 library(ggplot2)
5
6 head(flights, n = 3)
7 daily <- flights %>%
8   mutate(date = make_date(year, month, day)) %>%
9   count(date) %>%
10  mutate(wday = wday(date, label = TRUE))
11 head(daily, n = 3)
12 ggplot(daily, aes(wday, n)) +
13   geom_boxplot(outlier.colour = "hotpink") +
14   labs(x = "Weekday", y = "Flights",
15        subtitle = "Number of 2013 New York Flights Each Weekday")
```

Environment

Global Environment

daily: 365 obs. of 3 variables

- \$ date: Date[1:365], format: "2013-01-01" "2013-01-02" ...
- \$ n : int [1:365] 842 943 914 915 720 832 933 899 902...
- \$ wday: Ord.factor w/ 7 levels "Sun"<"Mon"<"Tue"<...: 3 ...

Console

```
~/Documents/Flights/
# A tibble: 3 x 19
  year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
<int> <int> <int> <int> <int> <dbl> <int> <int> <dbl> <chr>
1 2013 1 1 517 515 2 830 819 11 UA
2 2013 1 1 533 529 4 850 830 20 UA
3 2013 1 1 542 540 2 923 850 33 AA
# ... with 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
# distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
> daily <- flights %>%
+   mutate(date = make_date(year, month, day)) %>%
+   count(date) %>%
+   mutate(wday = wday(date, label = TRUE))
> head(daily, n = 3)
# A tibble: 3 x 3
  date           n wday
<date> <int> <ord>
1 2013-01-01 842 Tue
2 2013-01-02 943 Wed
3 2013-01-03 914 Thu
> ggplot(daily, aes(wday, n)) +
+   geom_boxplot(outlier.colour = "hotpink") +
+   labs(x = "Weekday", y = "Flights",
+        subtitle = "Number of 2013 New York Flights Each Weekday")
>
```

Plots

Number of 2013 New York Flights Each Weekday

Weekday	Min	Q1	Median	Q3	Max	Outliers
Sun	880	900	910	920	930	940, 950, 960, 970, 980, 990
Mon	910	920	930	940	950	960, 970, 980, 990
Tue	900	910	920	930	940	950, 960, 970, 980, 990
Wed	910	920	930	940	950	960, 970, 980, 990
Thu	900	910	920	930	940	950, 960, 970, 980, 990
Fri	910	920	930	940	950	960, 970, 980, 990
Sat	680	690	700	710	720	730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990

Command Line Interface (CLI)

- Command line tools
 - Mac/Linux: Terminal
 - Windows: Command Prompt, PowerShell
- Command-line interface (CLI).
 - CLI vs GUI (Graphical User Interface)

R and Python with CLI

- Mac/Linux:

```
R  
python  
python3
```

- Windows

```
# Command Prompt  
"C:\Program Files\R\R-4.1.2\bin\R.exe"  
"C:\Users\[Your User Name]\anaconda3\python.exe"  
# PowerShell  
& "C:\Program Files\R\R-4.1.2\bin\R.exe"  
# For python, Anaconda Prompt  
python
```

Non-Interactive Mode

- Interactive vs non-interactive mode
 - Batch execution
- R: source methods

```
source("script.R")
```

- Python

```
exec(open("script.py").read())
```


Non-Interactive Mode using Command Line

- Mac/Linux:

```
R CMD BATCH script.R
Rscript script.R
Rscript -e "rnorm(100)"
python script.py
python -c "print('hello world')"
```

- Windows

```
# Command Prompt
"C:\Program Files\R\R-4.1.2\bin\R.exe" CMD BATCH script.R
"C:\Program Files\R\R-4.1.2\bin\Rscript.exe" script.R
"C:\Program Files\R\R-4.1.2\bin\Rscript.exe" -e "rnorm(100)"
#### For PowerShell, remember to add &
"C:\Users\[Your User Name]\anaconda3\python.exe" script.py
"C:\Users\[Your User Name]\anaconda3\python.exe" -c "print('hello world')"
```

Shell

- The shell is a user interface for access to an operating system's services.
 - Most often we interact with the shell using a command-line interface (CLI).
- Usage:
 - Batch run
 - Multiple scripts, long running time, etc.
 - Remote machines, cloud computing

Some Commonly Used Commands

- Unique to OS
 - Mac OS (Unix) and Linux (Unix-like) commands are 90% similar
 - Linux is the most common OS for super computer and cloud computing

```
cd [foldername] # change directory
ls # list contents
pwd # print working directory
rm [filename] # remove files
mv [filename] [destination] # move files
cp [filename] [destination] # copy files
mkdir [foldername] # create folder
touch script.py # create file
cat script.py # print file
echo "hello world" # output text
grep [option] [pattern] [files] # search text
man mkdir # getting help
Rscript --help
```

Some Commands for Cloud Computing

- Log on and file transfer

```
ssh [remote user]@[remote machine address]  
ssh baole@zorro.american.edu  
scp -r ~/Desktop/myfolder baole@zorro.american.edu:~/Documents/
```

- Run scripts without hangup

```
nohup R CMD BATCH script.R &  
screen
```

Shell Script

```
#!/bin/sh
# This is a comment!
echo 'Starting ...'
mkdir
python python-script1.py
python python-script2.py
python python-script3.py
R CMD BATCH rscript1.R
nohup R CMD BATCH rscript2.R &
R CMD BATCH rscript3.R
echo 'Finished!'
```

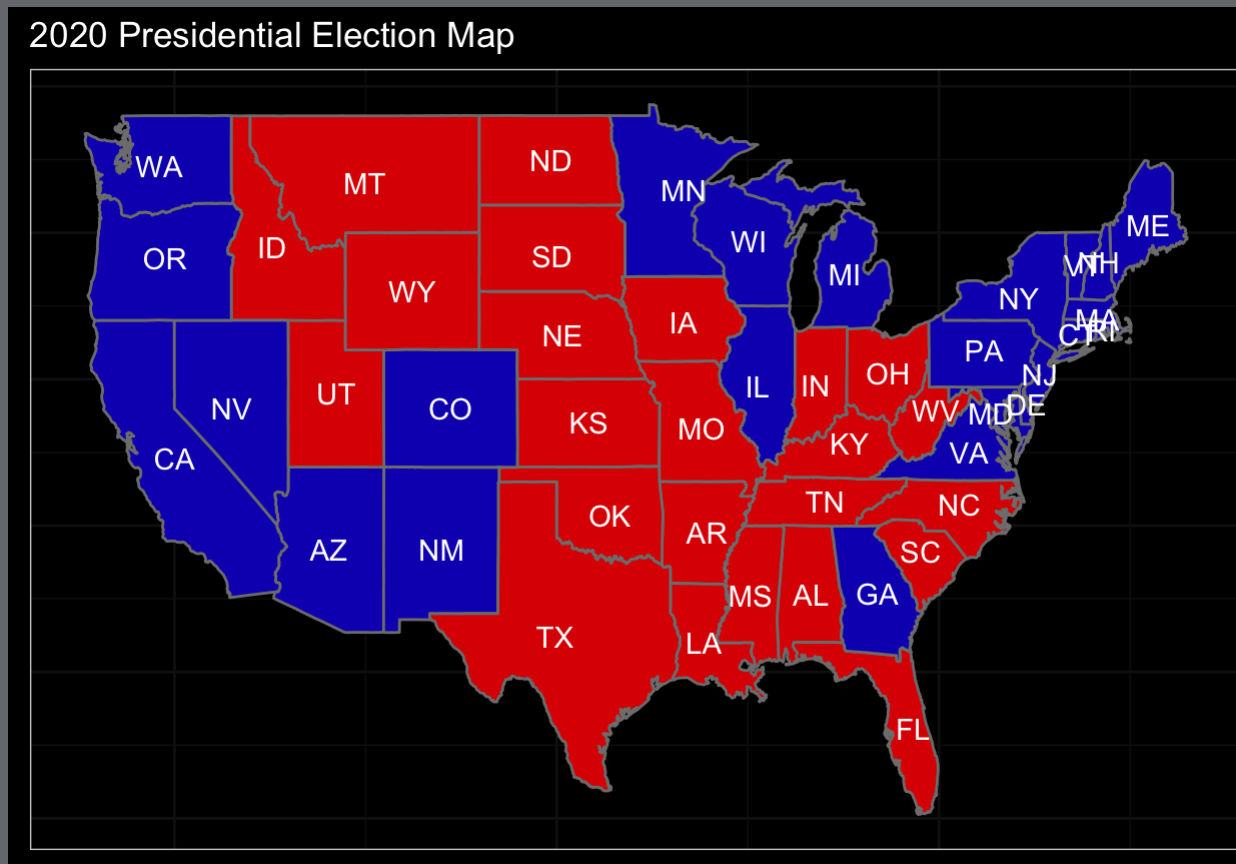
Exercise

- Try out one aforementioned way to run the following Python and R scripts
 - `demo/election-2020.py`
 - `demo/election-map-2020.R`

State	Candidate	Party	Votes	Percent	EC_vote
Alabama	Trump	Republican	1441170	62.0	9
Alabama	Biden	Democrat	849624	36.6	0
Alabama	Jorgensen	Libertarian	25176	1.1	0
Alabama	Write-ins		7312	0.3	0
Alaska	Trump	Republican	189951	52.8	3
Alaska	Biden	Democrat	153778	42.8	0

Exercise

- Try out one aforementioned way to run the following Python and R scripts
 - `demo/election-2020.py`
 - `demo/election-map-2020.R`



Exercise

- Use Wrapper

```
library(reticulate)
use_python("/usr/bin/python3")
source_python("election-2020.py")
source("election-map-2020.R")
```

- Use command line

```
python election-2020.py
Rscript election-map-2020.R
```


Computing Environments

- Hardware
- Software
 - Operating systems
 - System dependencies
 - R/python: versions, packages/libraries

Environment Management in R

- `renv`: project dependency management
 - Works with R Project

```
install.packages("renv")
```

- Functions:
 - `renv::init()`: initiate to include all R packages currently used by the project.
 - `renv::snapshot()`: save the state of project dependency.
 - `renv::restore()`: restore the state of project dependency when it's under a new environment.

Environment Management in R

- Workflow:
 - Call `renv::init()` to initialize a new project-local environment with a private R library,
 - Work in the project as normal, installing and removing new R packages as they are needed in the project,
 - Call `renv::snapshot()` to save the state of the project library to the lockfile (called `renv.lock`),
 - Continue working on your project, installing and updating R packages as needed.
 - Call `renv::snapshot()` again to save the state of your project library if your attempts to update R packages were successful, or call `renv::restore()` to revert to the previous state as encoded in the lockfile if your attempts to update packages introduced some new problems.

Environment Management in Python

- `venv`: virtual environments

```
pip install virtualenv
```

- Workflow:
 - Call `python3 -m venv myproj` in shell to create a virtual environment
 - Call `source myproject/bin/activate` to activate the environment
 - Work in the environment, installing and removing libraries as normal
 - Deactivate the environment by calling `deactivate`

Environment Management in Python

- conda has powerful, built-in environment management
 - Downside: platform specific
- Workflow:
 - Use `conda create -n myproj python` to create a Python virtual environment (click “y” to proceed).
 - Call `conda activate myproj` to activate the environment.
 - Work in the environment, installing and removing libraries using `conda install`.
 - Call `conda env export > environment.yml` to save the version info to `environment.yml`.
 - Call `conda deactivate` to stop the environment.

Exercise

1. Create a new folder and R project.
 2. Open R studio and initiate an R environment.
 - `renv::init()`: initiate to include all R packages currently used by the project.
 - `renv::snapshot()`: save the state of project dependency.
 - * `renv::restore()`: restore the state of project dependency when it's under a new environment.
 3. Install a new R package: e.g. `fortunes`, `cowsay`, etc.
 4. Use the library in an R script: e.g. `library(fortunes); fortune()`
 5. Install and use a older version of an R package: e.g. `devtools::install_version("ggplot2", version = "3.3.6", repos = "http://cran.us.r-project.org")`.
 6. Create a new R environment snapshop.
 7. Check out `renv.lock` file to make sure the versions are correct.
- Note: `renv` require that you have Rscripts that uses the packages in order to make them recorded in `renv.lock`

Multilingual Publishing with Quarto

- A open-source scientific and technical publishing system
- Dynamic document with `reveal.js`
- Next generation of Rmarkdown/Jupyter notebook: Python, R, Julia, Observable, or even language that doesn't exist yet.
- IDE: VS Code, RStudio, Jupyter Lab, etc.

Quarto Demo

