watch by Tuesday, October 6, 2020 | **Lesson #1**

Why code?

An introduction to scientific data analysis, principles of programming, and Python

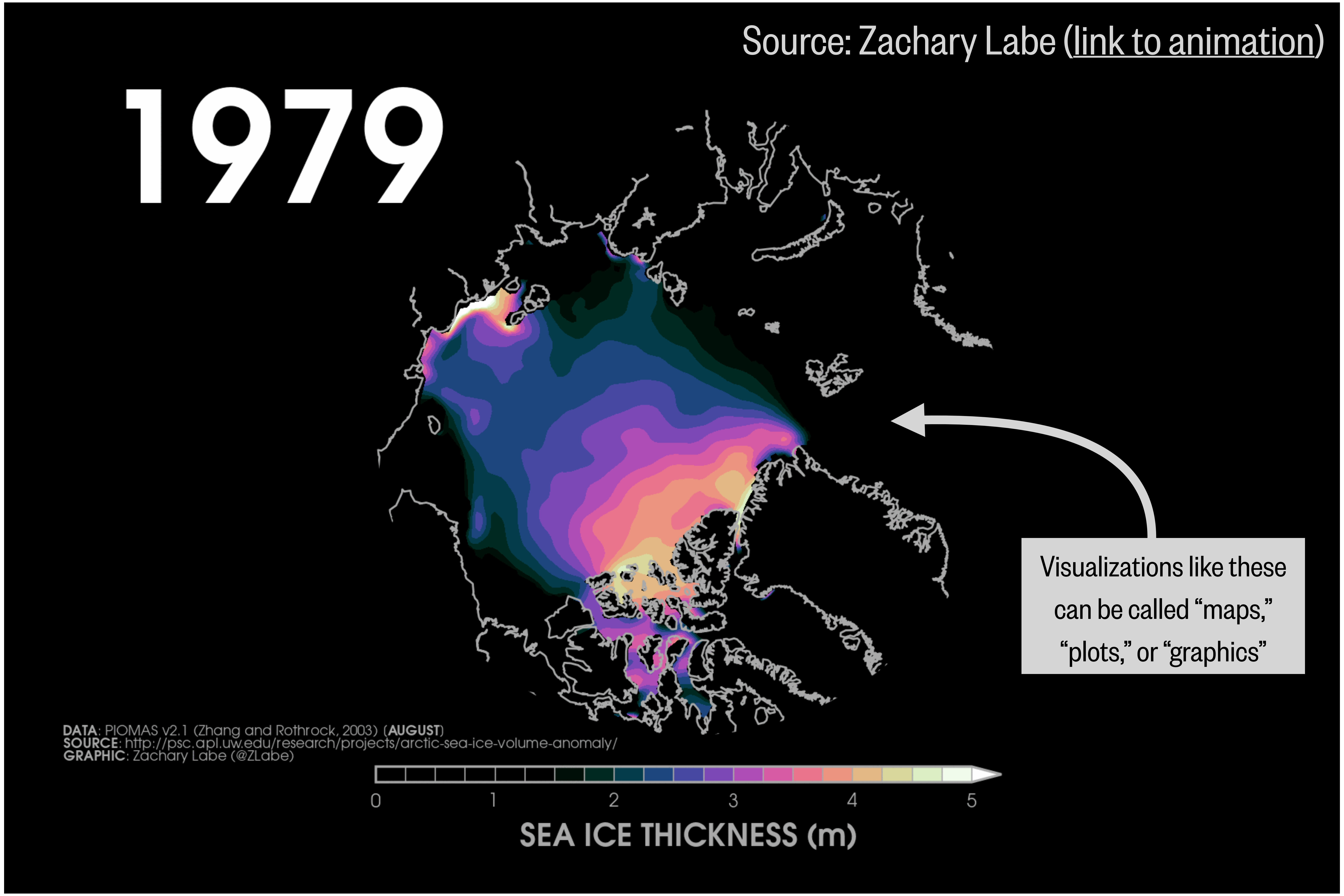
OCEAN 215 | Autumn 2020

Ethan Campbell and Katy Christensen

What we'll cover in this lesson

1. **The power of scientific data analysis**
2. Fundamental principles of programming languages
3. Why do we use Python in this course?
4. Different ways of using Python

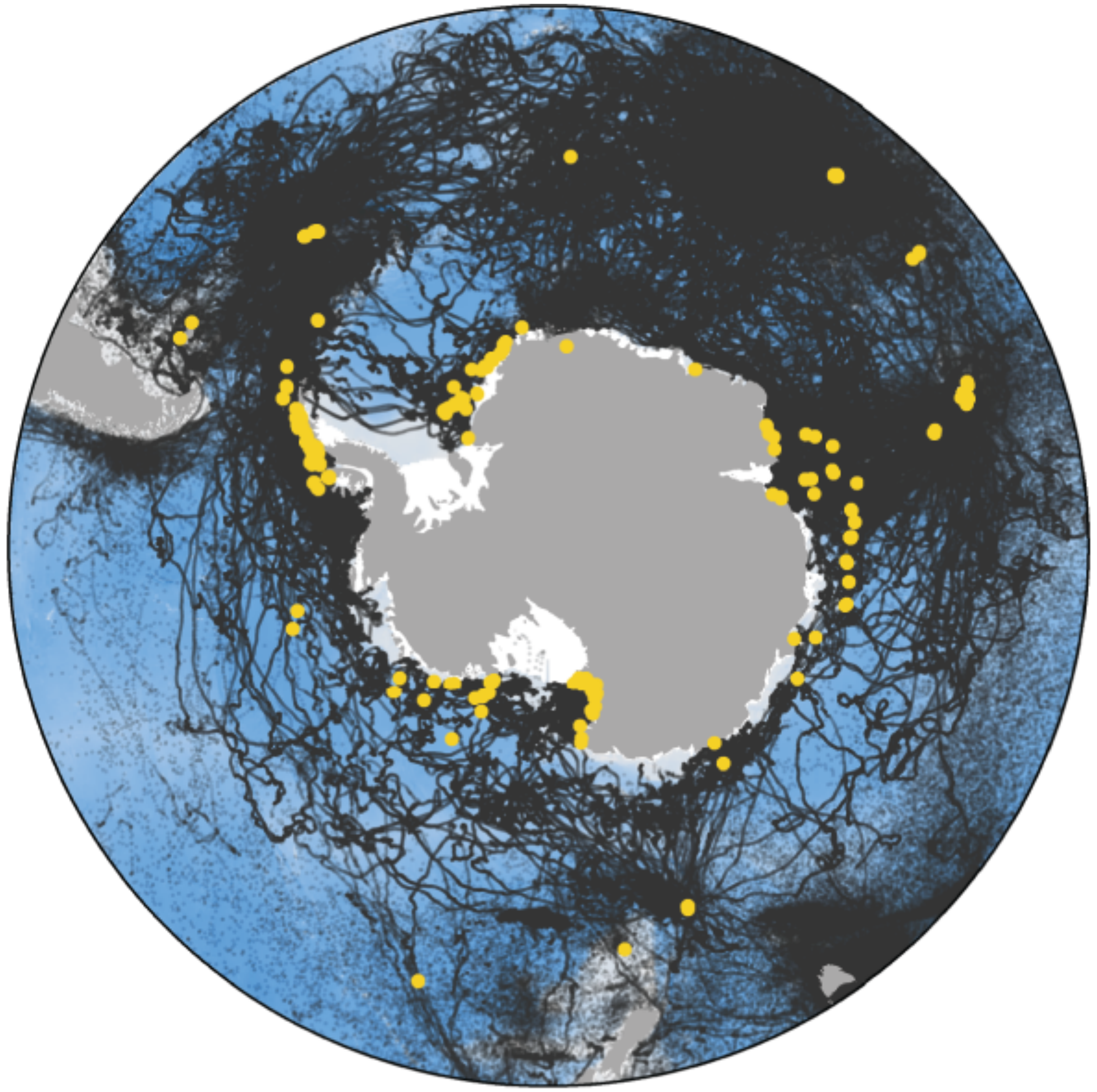
What can data tell us?



- **Example:** Arctic Ocean sea ice thickness in August from 1979 to 2020
- What can we learn from this data?
  - Ice has become thinner
  - Ice is usually thickest near Greenland
  - Ice thickness changes a lot year-to-year, but the patterns are usually similar
- **You can't do this in Excel!**

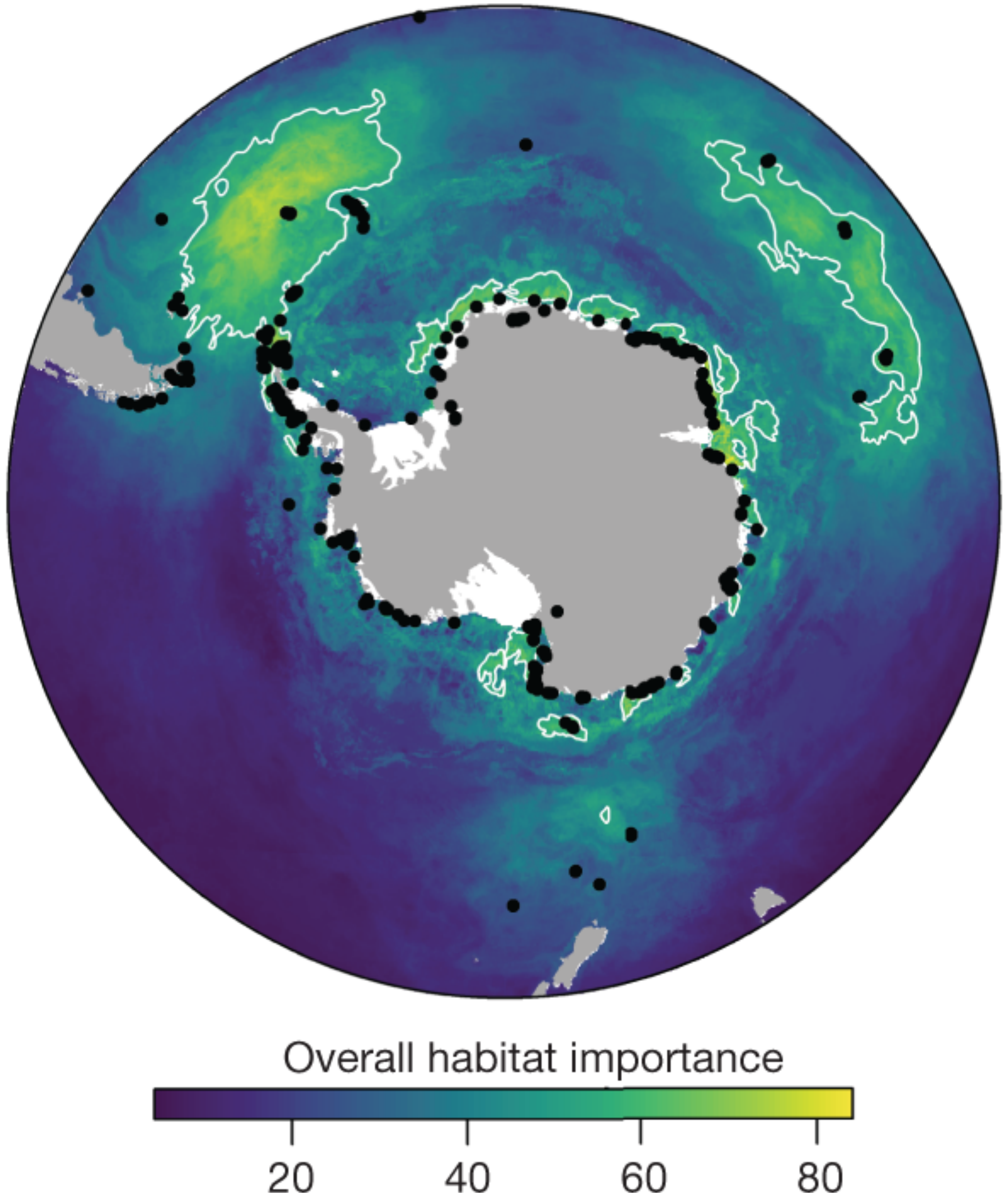
What can data tell us?

More examples from oceanography and marine science



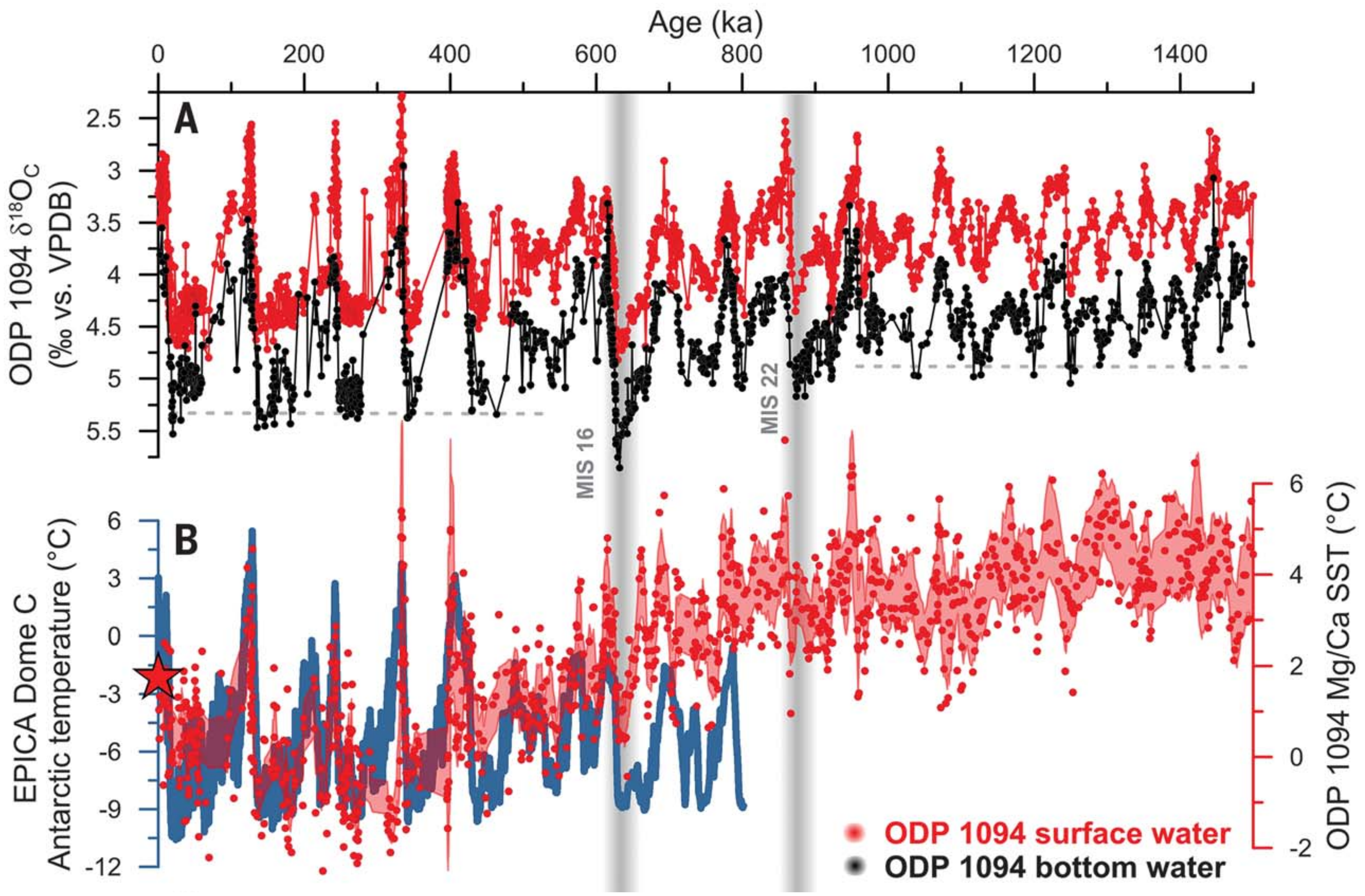
GPS tracks of marine mammals and seabirds show habitat importance around Antarctica

(Hindell et al. 2020)



What can data tell us?

More examples from oceanography and marine science

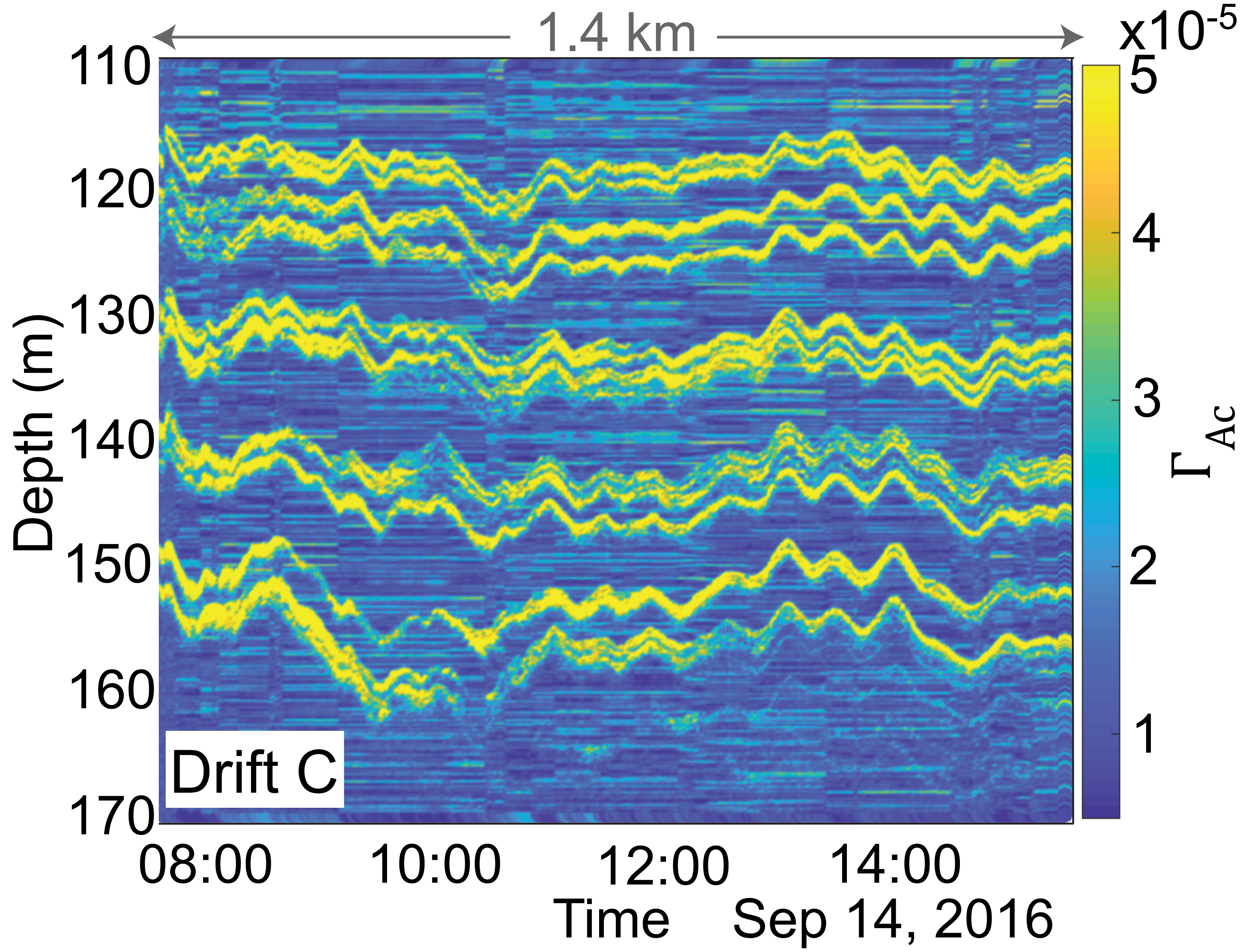


Oxygen isotopes measured in forams show how slower ocean mixing prolonged ice ages

(Hasenfrantz et al. 2019)

Echosounder profiles underneath Arctic sea ice reveal fluctuations in temperature and salinity layers

(Shibley et al. 2020)



Who uses data?

- It's not just oceanography! Every area of **earth science research** uses data...
    - *Satellite data* to estimate increases in **harvested forest area** and its impact on soil erosion
    - *Seismograms* to predict large **earthquakes** in real-time by identifying foreshocks
    - *Hydrometric gauge measurements* to show how **river flooding** is affected by climate change
    - *Emissions, air quality, and population data* to relate **air pollution** to human mortality
    - *Aircraft remote sensing* to identify super-emitting **methane sources** from landfills, agriculture, and the oil/gas sector
    - *Spacecraft measurements* to understand how **solar plasma ejections** impact Earth's atmosphere
- And it's not just earth science research! **Data science** touches every aspect of everyday life, such as:
    - Public health (tracking new **COVID-19 outbreaks** by looking for keywords in millions of *internet searches*)
    - Transportation (updating **driving directions** on-the-fly based on traffic jams inferred from *phone data*)
    - Sports (predicting successful **basketball shots** based on *a player's performance metrics*)
    - Social justice* (assessing **racial disparities in police stops** of cyclists in Seattle from *court infraction records***)

* If you're curious about how data can be used in service of racial justice, check out <u>this list of resources</u> and this <u>article in PNAS</u>. ** This is a project that I'm currently working on!

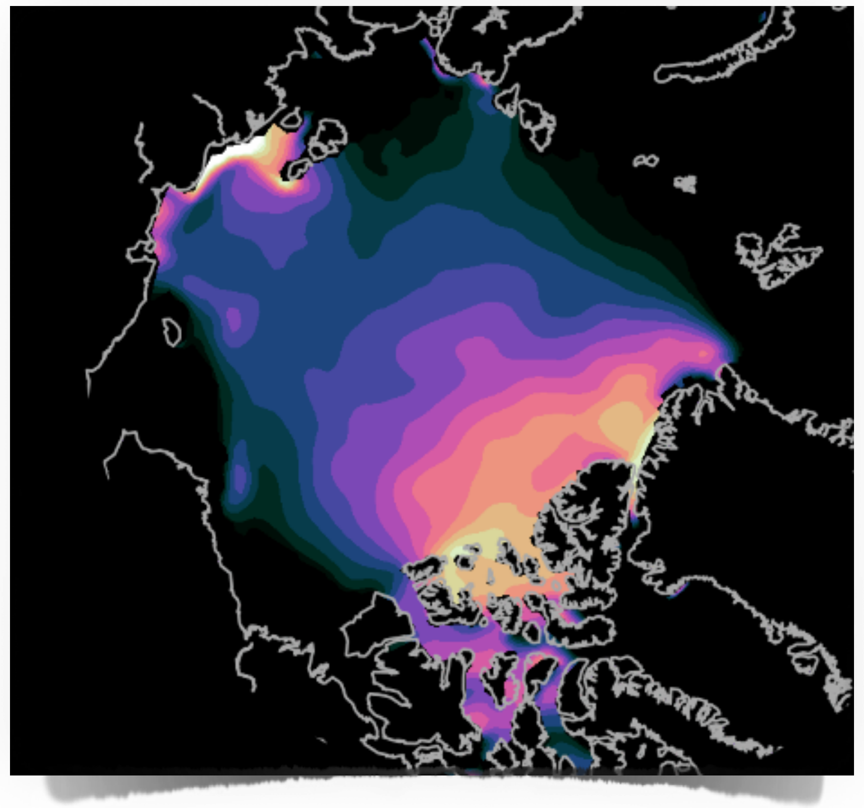
The modern scientific method

Frame the question

"I want to understand how sea ice thickness changes year-to-year"

Collect or find data

Existing data: PIOMAS sea ice thickness (1979-2020)



Write code

About 4 pages of Python code (available here)

Run code, debug, edit, repeat

*Code **never** works right the first time*

Analyze result; discover something!

Eureka! Arctic sea ice is getting thinner. But why? It looks like we'll need to look at air temperature...

When you're left with more questions than answers...

When the original data just isn't enough...

When there's more to discover in the original data...

When you made an oops...

The power of programming is its versatility

A common – but unnecessarily complicated – workflow using specialized software:

Download data → Clean up data → Conduct statistical manipulations of data → Visualize data

Programming enables you to make a computer do anything you want, in a unified workflow:

Start → → Finish

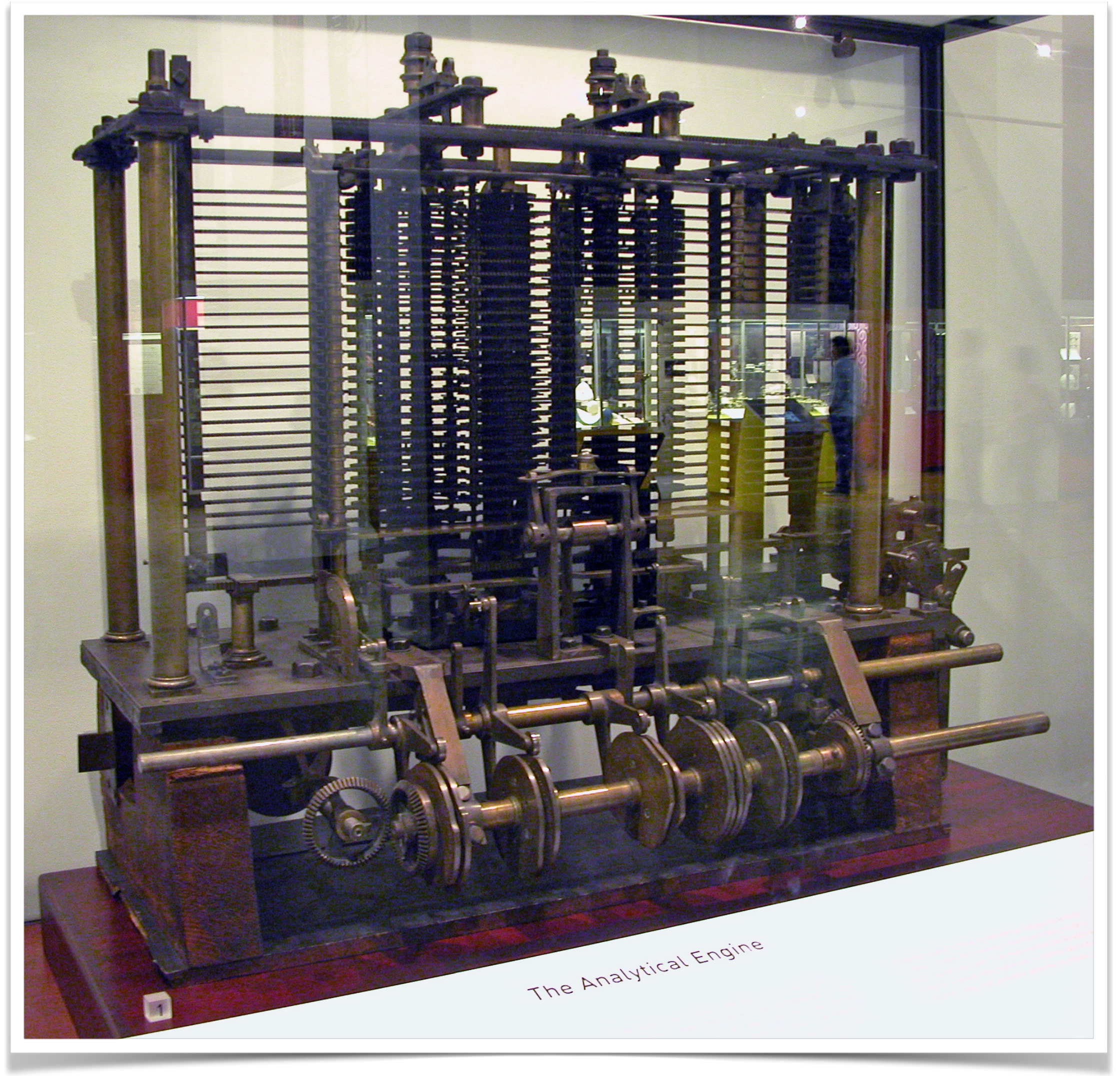
What we'll cover in this lesson

1. The power of scientific data analysis
2. **Fundamental principles of programming languages**
3. Why do we use Python in this course?
4. Different ways of using Python

Programming requires a human and a computer

First programmer →

Ada Lovelace (1815–1852)



The Analytical Machine

← First computer

("Turing complete," i.e. computationally universal)

Modern computers have:

- **Processors** that can perform a billion calculations per second
- **Memory** that can store hundreds to thousands of gigabytes (GB) of results
- **Input/output devices** to take human instructions and display results

Aspects of languages

Syntax describes valid combinations of symbols, words, and phrases:

- English: `Cat dog boy` → not syntactically valid

  `Cat hugs boy` → syntactically valid
- programming language: `“hi”!“hello”` → not syntactically valid

  `3*5` → syntactically valid

Python says:
“SyntaxError: invalid syntax”

Semantics gives meaning to a syntactically valid phrase:

- English: `Cat hugs boy` → syntactically valid but semantic error
- programming language: `3 + ”hi”` → syntactically valid but semantic error

Python says:
“TypeError: unsupported operand type for +”

Where things go wrong

Syntax errors:

- Common and easily caught; program won't run

Semantic errors:

- Some languages check for these before running a program, but some (including Python) don't
- Can cause unpredictable behavior

No semantic errors but **different meaning than what the programmer intended:**

1. Program crashes and stops running, or...
2. Program runs forever, or...
3. Program gives an answer, but different than expected

Python code:

```
"hi"!"hello"
```

← program won't run

Python code:

```
a = 3
print(a)
a + "hi"
```

← program will run

← this line will print "3"

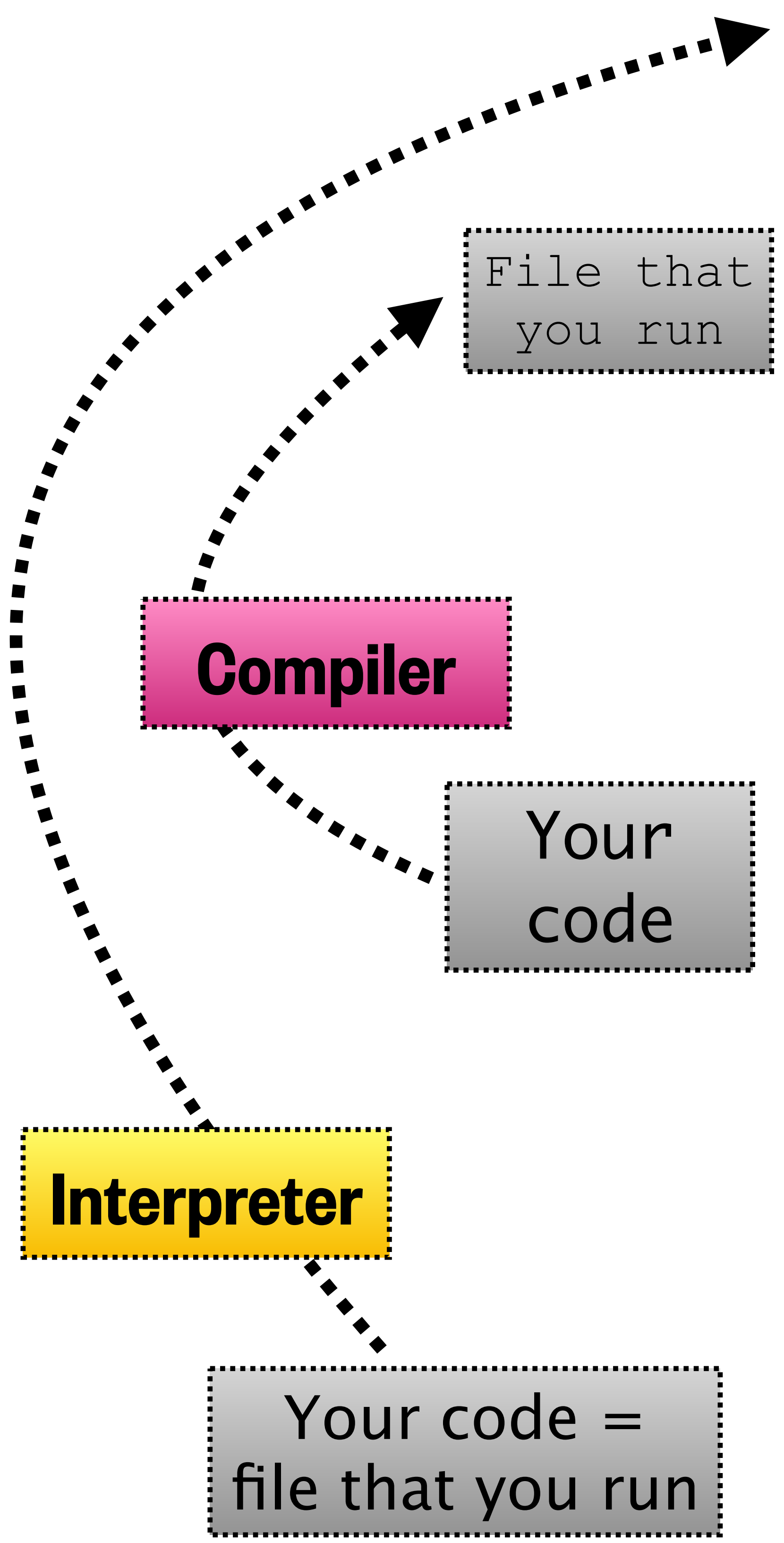
← program will crash on this line

Python code:

```
print("jello")
```

↑ program will run despite programmer intending for it to print "hello"

Types of programming languages



Machine language:

- This is what the computer understands
- Very difficult and error-prone to write

Assembly language:

```
LEA     MESSAGE,A1
MOVE.B #14,DO
TRAP    #15
MOVE.B #9,DO
TRAP    #15
MESSAGE DC.B 'HELLO!',0
END     START
```

Compiled languages (e.g. C, Fortran, Java):

- Full control over computer, runs very fast
- Programmer sometimes must manage memory manually
- Not easy to read or write

C:

```
#include <stdio.h>
int main()
{
    printf("Hello!");
    return 0;
}
```

Interpreted languages (e.g. Python, R, Matlab):

- Less control over computer, runs slightly slower
- Memory management is usually handled automatically
- "Expressive" (syntax closer to human language)
  → easy to read and write

Python:

```
print("Hello!")
```

What we'll cover in this lesson

1. The power of scientific data analysis
2. Fundamental principles of programming languages
3. **Why do we use Python in this course?**
4. Different ways of using Python

No language is perfect

> *"There are only two kinds of programming languages: the ones people complain about and the ones nobody uses."*

– **Bjarne Stroustrup** (the inventor of C++)

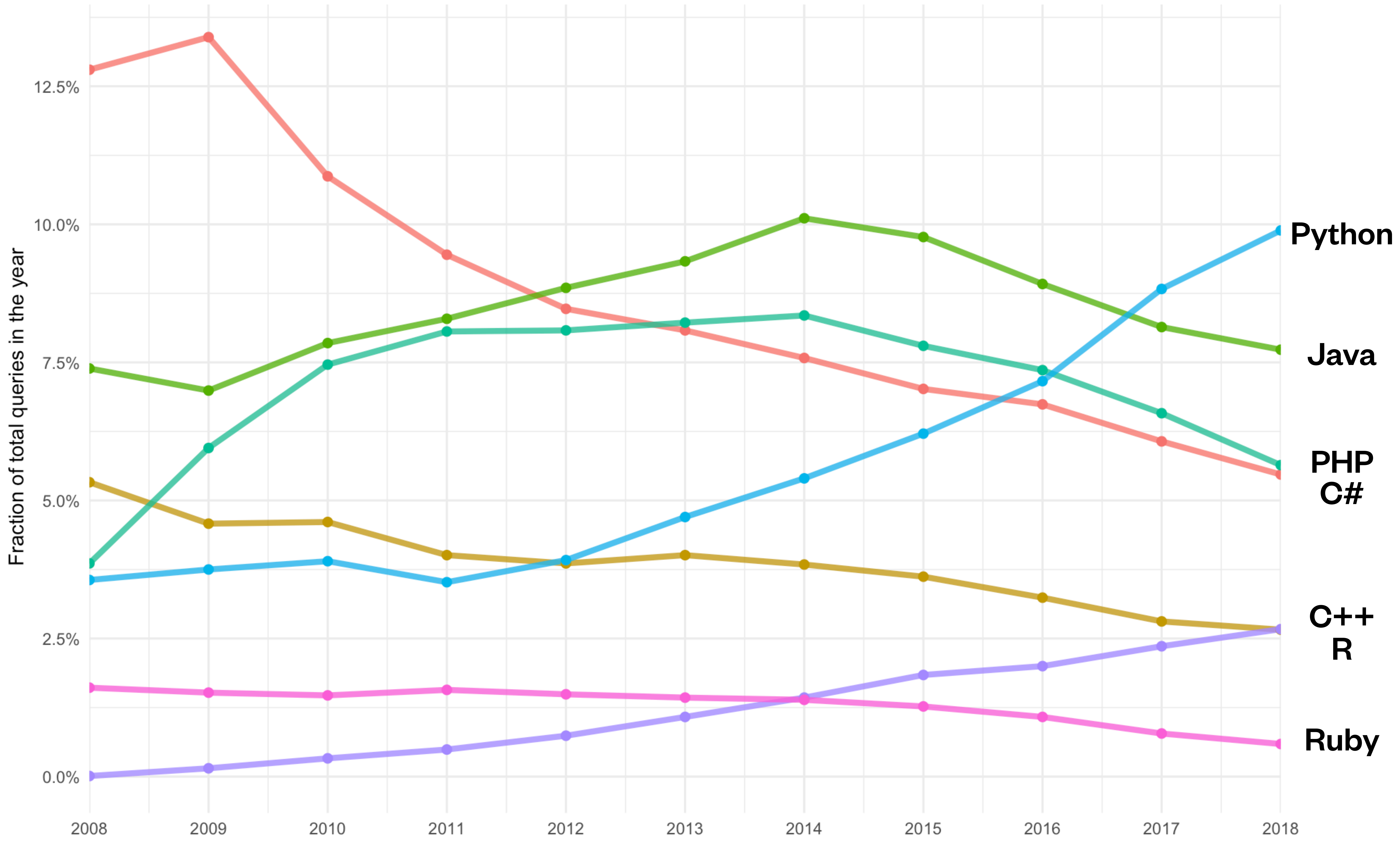
Three different programming languages:

But why Python? It is...

- **Free!** (unlike Matlab)
- Open source (unlike Matlab)
  - That means you're not dependent on a company to fix bugs
  - Large user community constantly working to improve language
- Old, so it's very stable (Python was created in 1991)
- General-purpose
- Incredibly popular in all areas of science
- Incredibly popular outside of science, too
- Easy to teach and learn

Python's rising popularity

Fraction of total questions per year on StackOverflow (a Q&A website for programmers)



What we'll cover in this lesson

1. The power of scientific data analysis
2. Fundamental principles of programming languages
3. Why do we use Python in this course?
4. **Different ways of using Python**

Python versions

- **Python 2**
  - Released in 2000
  - Very few good reasons to use it anymore
  - Print statements look like this: `print “Hello”`
  - Some people still use it, but we won’t
- **Python 3**
  - Released in 2009; latest version is 3.7
  - Incredibly similar syntax to Python 2, but different enough to not be backwards compatible
  - Print statements look like this: `print(“Hello”)`
  - This is what we will use

Different ways to code in Python

Type of Python code:

Interactive Python (**IPython**) shell

```
>>> print("Hello")
Hello
>>> print(3)
3
```

Python script

.py file

```
print("Hello")
print(3)
```

run

output

```
Hello
3
```

Jupyter notebook

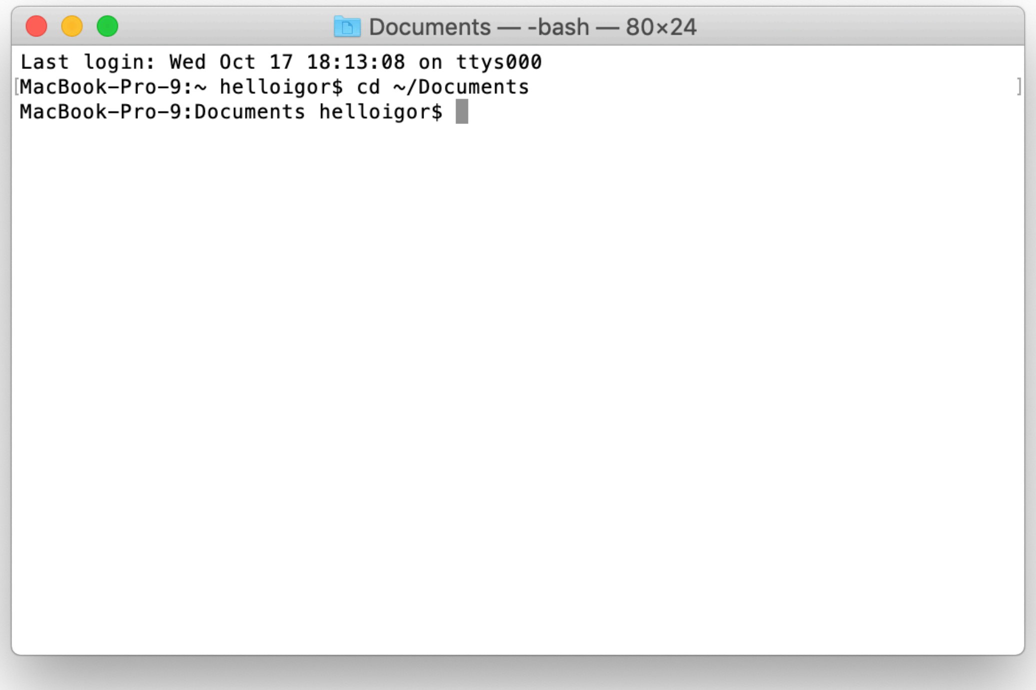
.ipynb file

```
[1]:  a = "Hello"
      b = 3
...
[26]: print(a,b)
      Hello 3
```

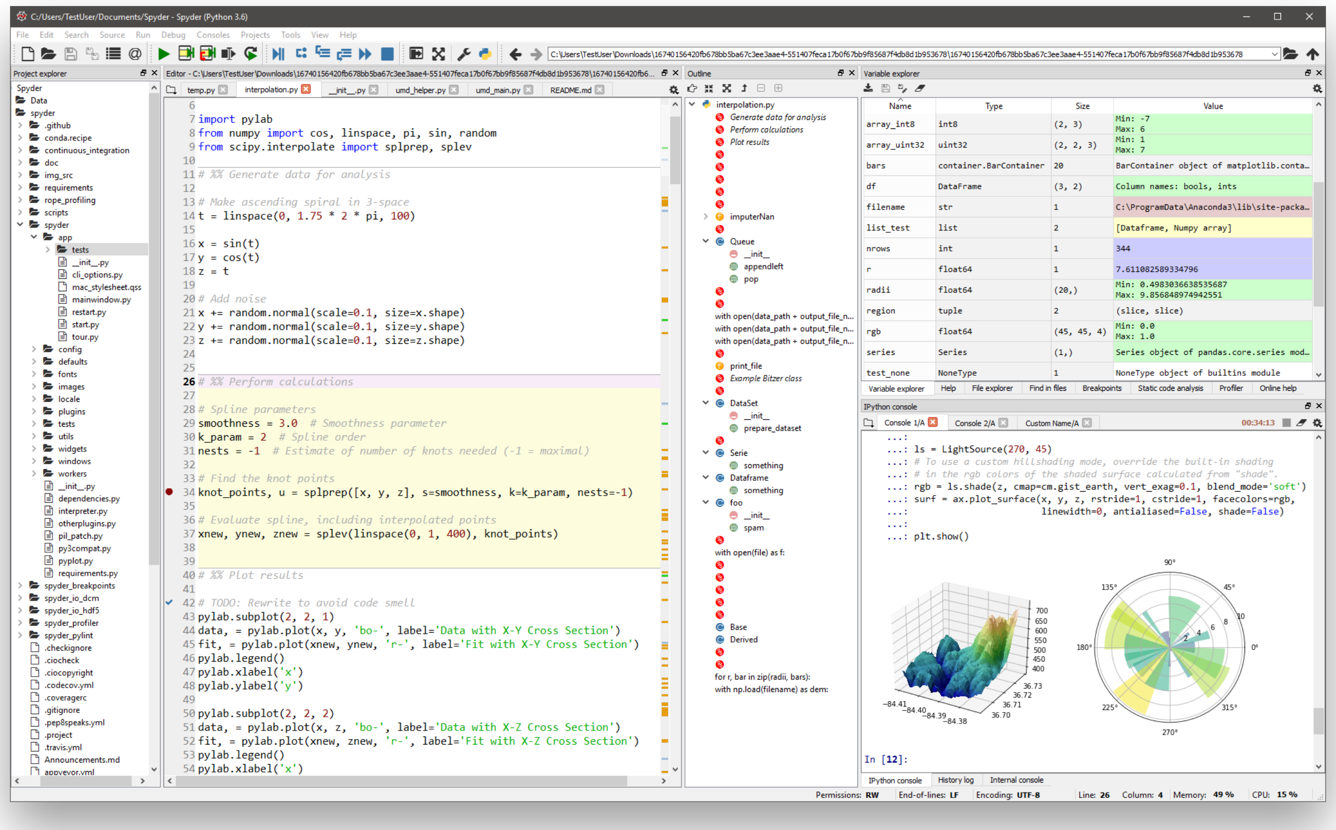
Mac/Windows application:

Command line

(MacOS Terminal or Windows Command Prompt)

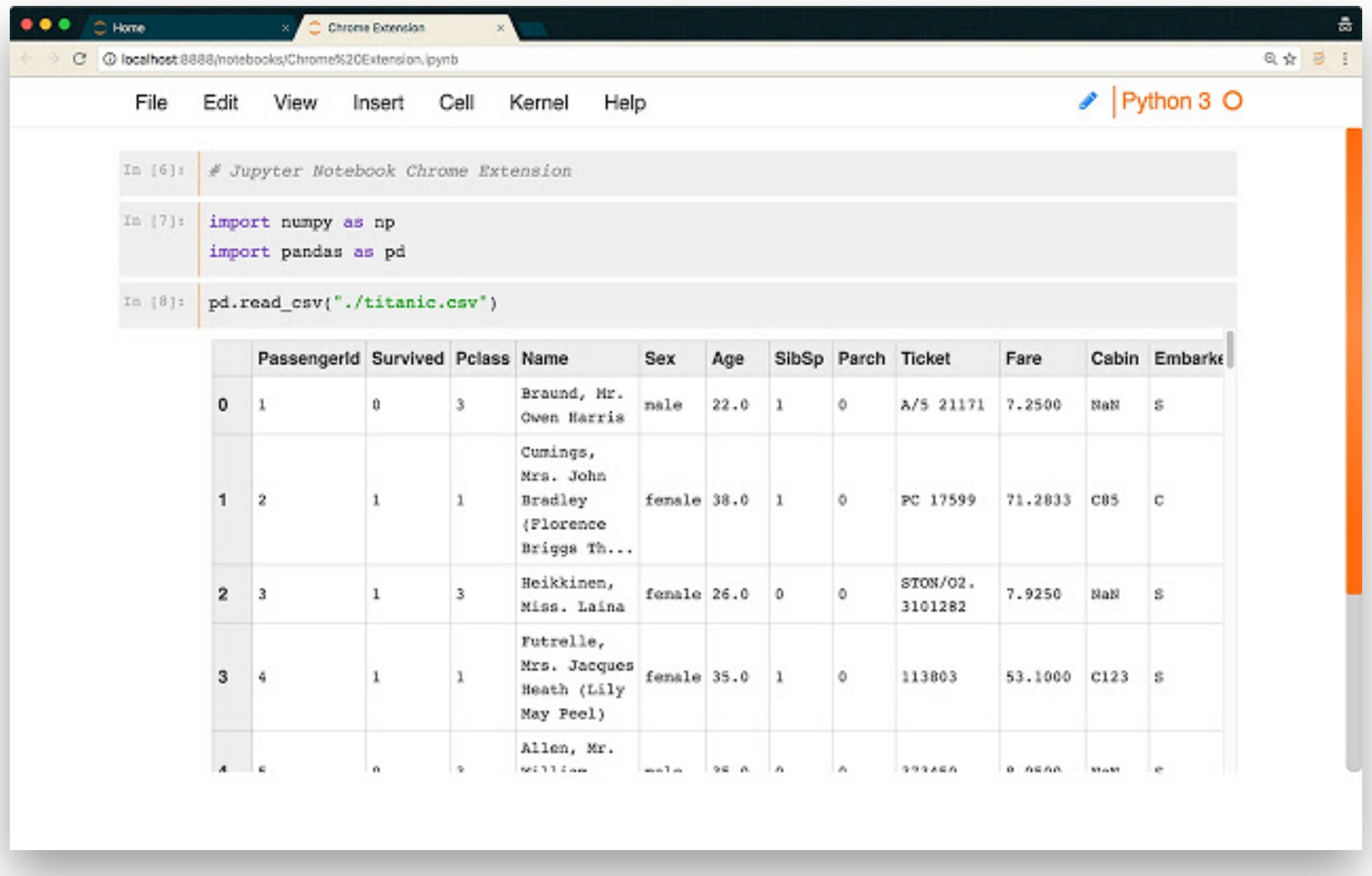


Integrated development environment (**IDE**)

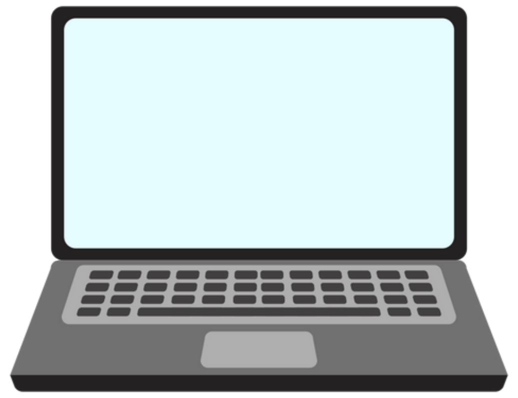


Internet browser

(Chrome, Safari, Firefox, etc.)



Jupyter vs. Google Colab notebooks

	Where is the code run?	How to access them?	Advantages (+) and disadvantages (-)
Jupyter notebooks	Your computer ("the local machine") 	1. Install Jupyter 2. Open command line app (Terminal on Macs, Command Prompt on PCs) 3. Type "jupyter notebook," which will start a local server 4. Open internet browser 5. Navigate to server address	• **(-) Some setup required** • (+) No internet connection required • (+) Code runs fast if your computer is fast • (-) Code runs slow if your computer is slow • (+) Bonus features, customizability, ability to install any package, etc. • (+) Free
Google Colab notebooks We'll be using these!	Google's servers ("the cloud")	1. Open internet browser 2. Navigate to: colab.research.google.com	• **(+) No setup required** • (-) Requires internet connection • (+/-) Code runs decently fast but not blazingly fast • (-) Less customizability, more difficult package management • (+/-) Free, as long as Google says it's free • (+) Google Drive integration; easy to share

Texts used to create this lesson + useful resources

- Princeton University – Computer Science: An Interdisciplinary Approach, Lecture 1
- MIT OpenCourseware – Introduction to Computer Science and Programming in Python, Lecture 1
- Rutgers University – Introduction to Computer Science, Lectures 1 and 2
- CU Boulder – Introduction to Earth Data Science, Lecture 1
- Johnny Wei-Bing Lin – A Hands-On Introduction to Using Python in the Atmospheric and Oceanic Sciences
- Monash University – Introduction to Data Science
- Ryan Abernathey and Kerry Key – An Introduction to Earth and Environmental Data Science
- Jake VanderPlas – Python Data Science Handbook
- Johnny Wei-Bing Lin – Bulletin of the American Meteorological Society – "Why Python Is the Next Wave in Earth Sciences Computing"
- Tom Waterman – Medium – "Why Python is better than R for Data Science careers"
- Ada Lovelace image: https://www.computerhistory.org/babbage/adalovelace/
- Analytical Machine image: https://en.wikipedia.org/wiki/File:AnalyticalMachine_Babbage_London.jpg