Python for Geospatial Big Data and Data Science Using the FASRC

Robert Spang
Visiting Scholar, Centre for Geographic Analysis, Harvard University
Quality and Usability Lab, Technical University of Berlin, Germany

Devika Kakkar and Xiaokang Fu
Centre for Geographic Analysis, Harvard University

Center for Geographic Analysis
Harvard University

Agenda

- Five chapters
  - Introduction
  - Case Study & Data Sets

  —— Lunch Break ——

  - Moving to the FASRC
  - Performance Optimization
  - Outro

Input

Hands on

Center for Geographic Analysis
Harvard University

Chapter 1
Introduction

Python for Geospatial Big Data and Data Science Using the FASRC

Center for
Geographic Analysis
Harvard University

Who we are

- Robert Spang
  - TU Berlin, visiting scholar with the CGA
- Devika Kakkar
  - Data Science Project Manager
- Xiaokang Fu
  - Postdoctoral Fellow

Center for
Geographic Analysis
Harvard University

Who are you?

- Short round of introduction
  - Name & field of research
  - Why are you here? What do you hope to take away from this workshop? What do you want to use HPC for?

- Warm up
  - Have you worked with the FASRC before? The command line?
  - How do you rate your own programming experience level?

Files & Data for the Workshop

- Files
  - Slides, exercises, command cheat sheet, and code files
  - GitHub: https://github.com/RGreinacher/python-workshop-gis-big-data/tree/main

- Datasets
  - All data we'll work with
  - Publicly accessible on the cluster: `/n/holyscratch01/cga/rspang/workshop_data/`
  - Harvard's Dataverse: [TODO]

Introduction to Big Data Processing & HPC

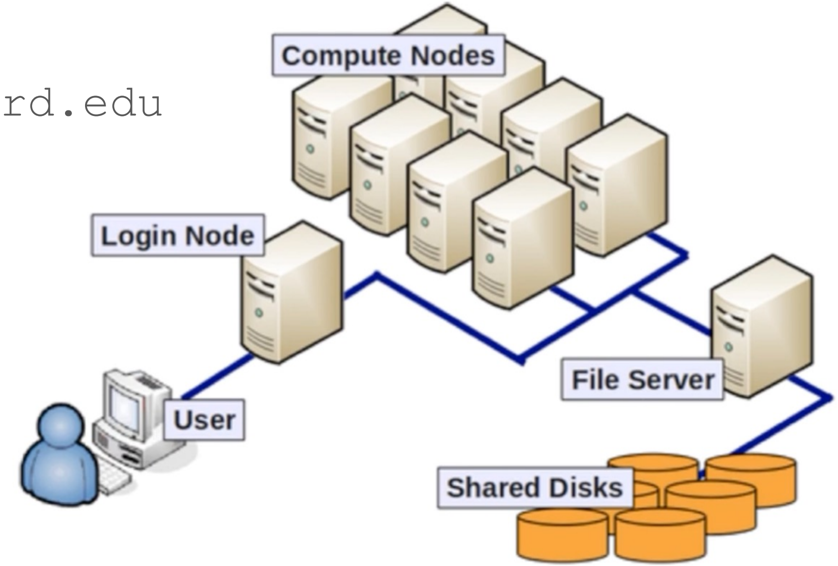
- Big Data
  - Volume, Velocity, and Variety (3Vs)
  - Also: Veracity and Value
- Processing
  - Large dataset → aggregated results → smaller datasets
- Faculty of Arts and Sciences Research Cluster
  - Harvard's High-Performance Cluster (HPC)

Differences between a HPC & the Cloud

- Purpose
- Architecture
- Resource Allocation
- Usage and Applications
- Flexibility
- Economic Model
- Location

Structure of the FASRC – Structure

- Access the cluster through a “login node”
    - `ssh user@login.rc.fas.harvard.edu`
- From there, connect to compute nodes to run apps
    - Login nodes are not designed for anything compute/memory intensive



Center for Geographic Analysis
Harvard University

Source: https://www.youtube.com/watch?v=Ay8oR5n-yyQ

Structure of the FASRC – SLURM

- SLURM is a job scheduler
- Specify the resources needed, get a node allocated
  - Specify node-type, CPU cores, memory, and runtime
  - A job terminates after the designated time is up
  - SLURM ensures users do not exceed resource request
- Mostly used to queue jobs, but can also run an interactive session
  - `srun --pty -p test -mem 100 -t 0-01:00 /bin/bash`

Structure of the FASRC – Partitions

- Partitions are the different node-classes
  - Each class comes with different attributes for different use-cases

Partitions:	shared	gpu	test	gpu_test	serial_requeue	gpu_requeue	bigmem	unrestricted	**pi_lab**
Time Limit	7 days	7 days	8 hrs	1 hrs	7 days	7 days	no limit	no limit	**varies**
# Nodes	530	15	16	1	1930	155	6	8	**varies**
# Cores / Node	48	32 + 4 V100	48	32 + 4 V100	varies	varies	64	64	**varies**
Memory / Node (GB)	196	375	196	375	varies	varies	512	256	**varies**

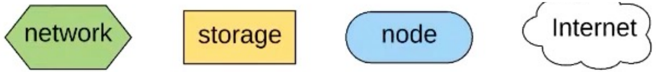
Center for Geographic Analysis
Harvard University

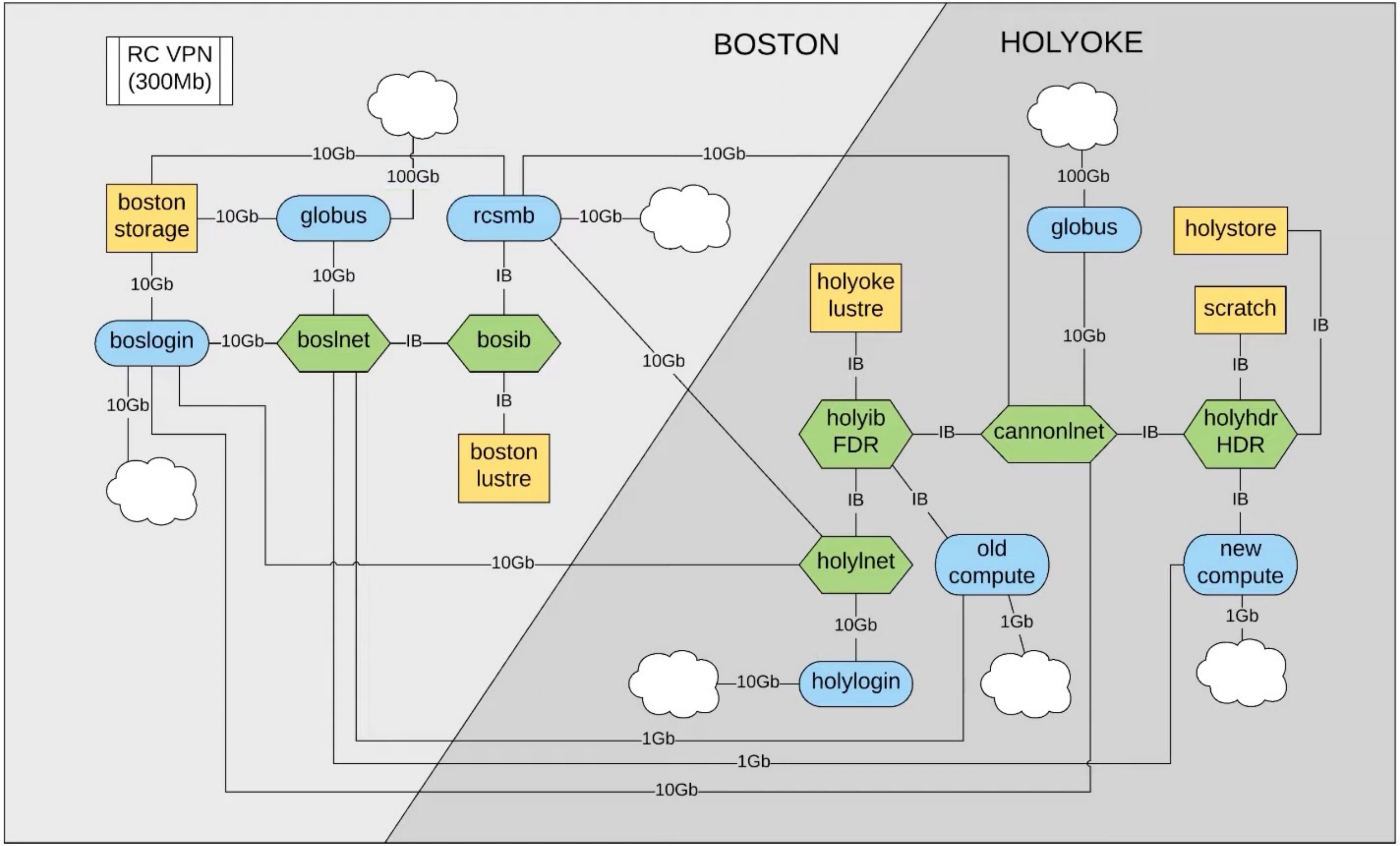
Source: https://www.youtube.com/watch?v=Ay8oR5n-yyQ

Structure of the FASRC – Storage

- Two sites with different networks
- Consider where to up-/download data
  - Tools e.g. rsync, scp, …
- Login-nodes are are suitable for most application (10Gb/s)
  - For large data sets, make sure to use a cable connection (instead of e.g. EDUROAM WiFi)

FAS RC Network Diagram





Source: https://www.youtube.com/watch?v=n1I4-4TGVP8

Structure of the FASRC – Storage

- Home directory is your primary, private space
  - 100GB
  - Moderate performance, not suitable for heavy I/O

- Local (node) scratch
  - /scratch
  - 200-300GB/node (shared with all users on the node!)
  - **Lives for job duration**

- Global scratch
  - /n/$SCRATCH
  - 2.4PB (shared with everyone on the cluser)
  - **Lives for 90 days**

Center for Geographic Analysis
Harvard University

	Home Directories	Lab Storage	Local Scratch	Global Scratch	Persistent Research Data
Mount Point	/n/home#/ $USER	/n/pi_lab	/scratch	/n/$SCRATCH	/n/$REPOS
Size Limit	100GB	4TB+	200-300 GB/node	2.4PB total	3PB
Availability	All cluster nodes + Desktop/laptop	All cluster nodes + Desktop/laptop	Local compute node only.	All cluster nodes	All cluster nodes
Retention Policy	Indefinite	Indefinite	Job duration	90 days	3-9 mo
Backup	Hourly snapshot + Daily Offsite	Daily Offsite	No backup	No backup	External Repos No backup
Performance	Moderate. Not suitable for high I/O	Moderate. Not suitable for high I/O	Suited for small file I/O intensive jobs	Appropriate for large file I/O intensive jobs	Appropriate for large I/O intensive jobs
Cost	Free	4TB Free + Expansion at a cost	Free	Free	Free

Center for Geographic Analysis
Harvard University

Source: https://www.youtube.com/watch?v=rGH96xVQNyM

Code of Conduct

- Shared resources
  - Expect > 800 users of the RC
  - Your actions might influence others' experiences
  - Request only the resources you need
- Login nodes only to login, to submit jobs and to receive data
- Consider the right storage space for each task

Exercise I - Web

- Login to the FASRC web interface
  - `https://rcood.rc.fas.harvard.edu/`
  - Start an interactive Jupyter notebook session
    - Default settings are ok
    - In Jupyter, create a new Python Notebook, name it `test.ipynb`
    - Run a simple demo code (e.g. `print("Hello World")`)

Exercise II - CLI

- Login to the cluster via SSH
  - `ssh user@login.rc.fas.harvard.edu`

- Setup a python environment
  - Load the Python / mamba module
    - `module load Mambaforge/23.3.1-fasrc01`
  - Create a new *mamba* environment
    - `mamba create -n workshop python=3.9 --file requirements.txt`

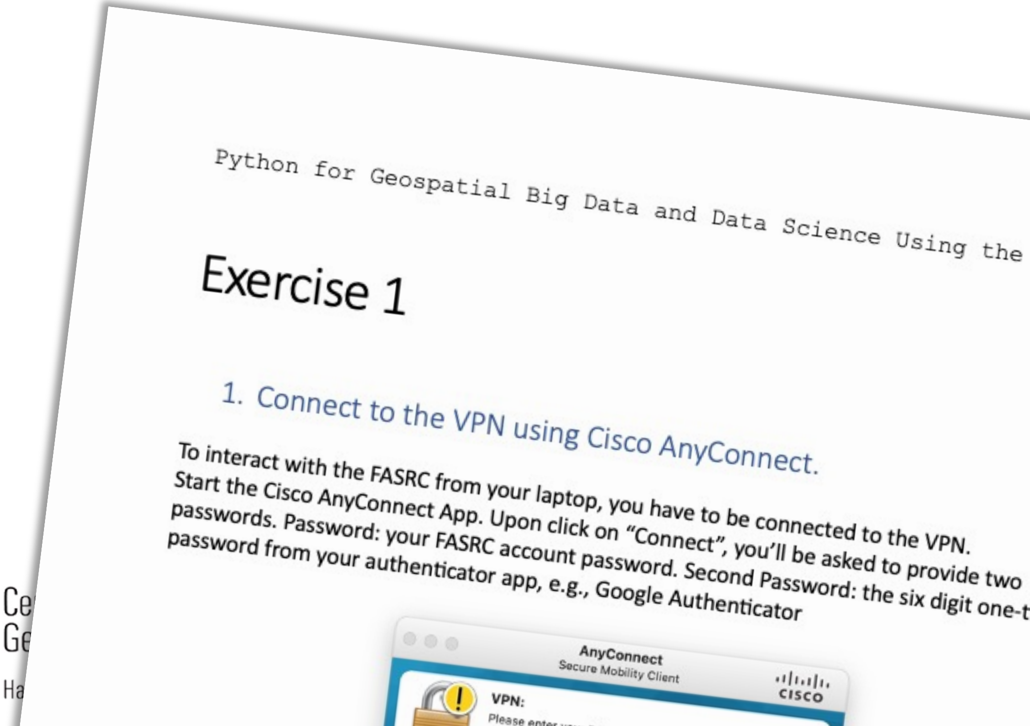
Center for Geographic Analysis
Harvard University

Exercise II - CLI

- Start an interactive shell on a compute node
    - `srun --pty -p test --mem 1000 -c 2 -t 0-01:00 /bin/bash`
- Load Python module & start environment
    - `module load Mambaforge/23.3.1-fasrc01`
    - `mamba activate workshop`
- Run Python code
    - `jupytext --to py test.ipynb # converts .ipynb to .py`
    - `python test.py`

Exercise

Find the exercise document in the GitHub repo


Python for Geospatial Big Data and Data Science Using the

Exercise 1

1. Connect to the VPN using Cisco AnyConnect.

To interact with the FASRC from your laptop, you have to be connected to the VPN. Start the Cisco AnyConnect App. Upon click on "Connect", you'll be asked to provide two passwords. Password: your FASRC account password. Second Password: the six digit one-t password from your authenticator app, e.g., Google Authenticator

Summary of Chapter 1

- Web and CLI
    - The FASRC is accessible through the web and via the CLI
    - The command line is the main access channel

- SLURM
    - SLURM manages resource requests
    - Resources must be known beforehand

Center for
Geographic Analysis
Harvard University

Chapter 1
Questions & Comments?

5min coffee break

Center for
Geographic Analysis
Harvard University

Chapter 2
Case Study & Data Sets

Python for Geospatial Big Data and Data Science Using the FASRC

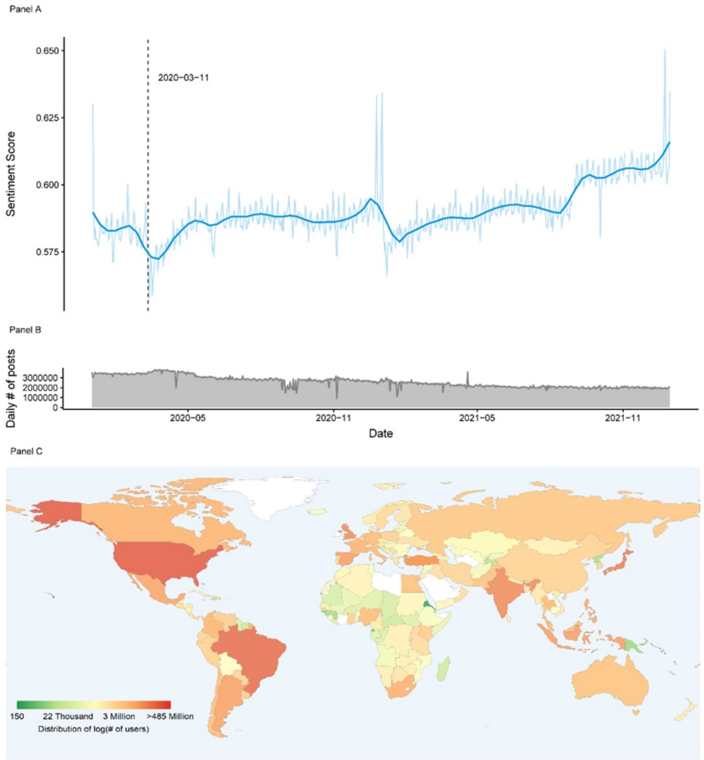
Center for
Geographic Analysis
Harvard University

The Data Set

- Twitter Sentiment Geographical Index (TSGI)
  - Tweets with coordinates → sentiment analysis
  - Average per day, aggregated per county, state, or country
  - CGA, Harvard University & Sustainable Urbanization Lab, MIT
- Compare sentiments in regions across time
  - 164 countries, > 10 years coverage, ~ 83% accuracy
  - CSV files, one per year
- Links
  - https://www.globalsentiment.mit.edu/dataset
  - https://sdgstoday.org/dataset/twitter-sentiment-geographical-index

Center for
Geographic Analysis
Harvard University

TSGI: Statistics



7.3 Billion enriched tweets

High Performance system running on NERC

Real Time enrichment (**150,000** tweets/hours)

Global coverage **(164** countries**)**

83% computation accuracy

10 years of data coverage (2012-Present)

Multiple administrative level **daily** indices

To the best of our knowledge, it is the first SWB dataset of this scale and granularity!
In contrast to the previous works focusing on SWB, TSGI is not limited to a specific topic, period, or location.

The Data Set

```
tweets_df.sample(10) ...
```

	date	country	state	county	sentiment_score	tweets
6720731	2022-12-17	United Kingdom	England	Northumberland	0.606733	1549
2404880	2022-05-09	Japan	Shiga	Taga	0.720183	15
2279265	2022-05-03	Philippines	Bohol	Tagbilaran City	0.616437	72
231971	2022-01-12	Japan	Aomori	Noheji	0.772601	4
4786320	2022-09-09	Cambodia	Kaôh Kong	Botum Sakor	0.800125	1
2458829	2022-05-12	United States	Oregon	Clatsop	0.671406	38
1121073	2022-03-02	China	Liaoning	Benxi	0.625170	1
5749926	2022-10-28	United States	North Carolina	Cabarrus	0.594906	481
6713367	2022-12-16	Colombia	Santander	Ocamonte	0.868015	28
4305041	2022-08-16	Brazil	Piauí	Bom Princípio do Piauí	0.444439	100

The Case Study

Current research:

- Wang, J., Guetta-Jeanrenaud, N., Palacios, J., Fan, Y., Kakkar, D., Obradovich, N., & Zheng, S. (2022). **A global nonlinear effect of temperature on human sentiment.** Nature Human Behavior (Under Review).

Center for Geographic Analysis
Harvard University

The Case Study

- Research question:
  Does rainfall influence public sentiment?
  - Hypothesis: Public sentiment is, on average, lower on rainy days compared to non-rainy days.

- Available data
  - Sentiment per region, per day (TSGI)

- Data needed
  - Precipitation for the same time span

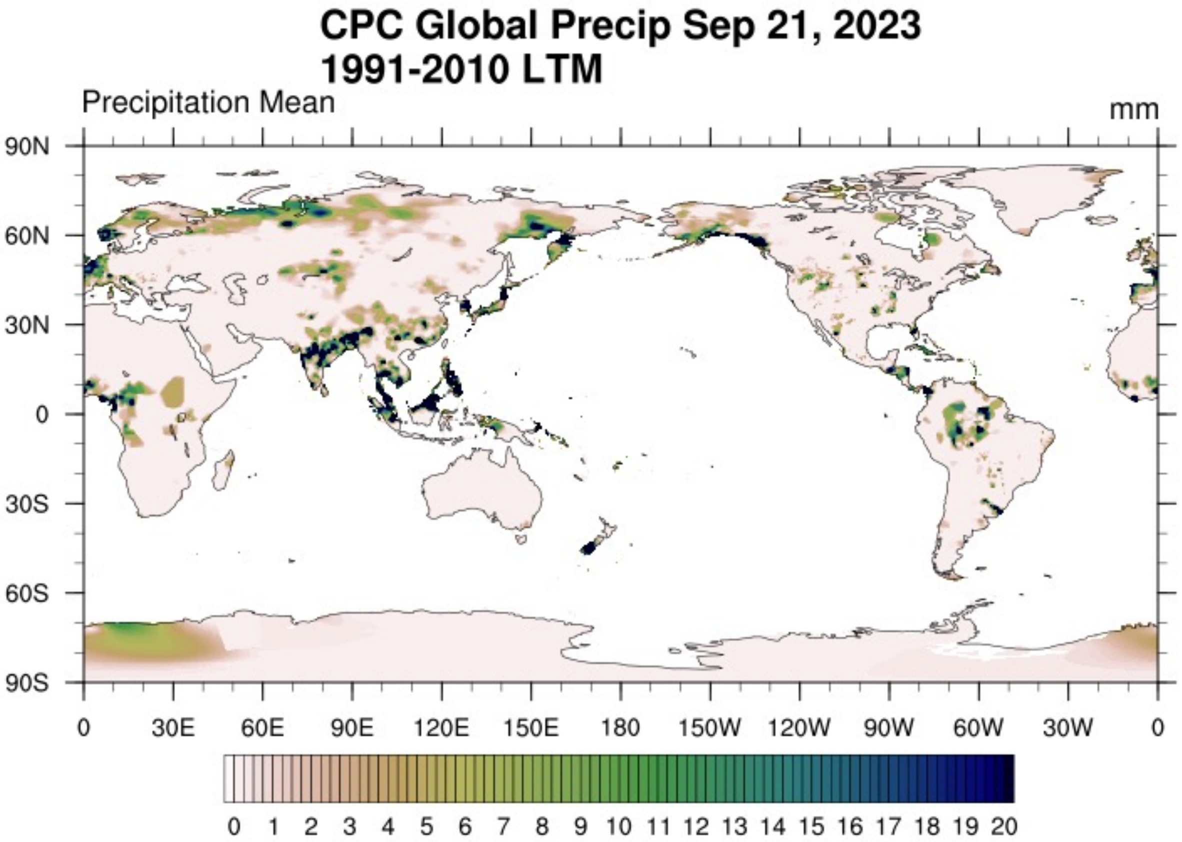
Center for
Geographic Analysis
Harvard University

Precipitation Data

- National Oceanic and Atmospheric Administration
  - Climate Prediction Center (CPC)
  - Global Unified Gauge-Based Analysis of Daily Precipitation
    - Temporal Coverage: Daily 1979/01/01 to 2023/09/21
    - Spatial Coverage: 0.5 degree lat x 0.5 degree lon (720x360)
    - NetCDF files, one per year
  - https://psl.noaa.gov/data/gridded/data.cpc.globalprecip.html

Center for Geographic Analysis
Harvard University

Precipitation Data: CPC 0.5x0.5 Global Daily



Precipitation Data: CPC 0.5x0.5 Global Daily

✓ `dataset` ···

xarray.Dataset

▶ Dimensions: (**lat**: 360, **lon**: 720, **time**: 365)

▼ Coordinates:

lat	(lat)	float32	89.75 89.25 88.75 ... -89.25 -89.75
lon	(lon)	float32	0.25 0.75 1.25 ... 359.2 359.8
time	(time)	datetime64[ns]	2022-01-01 ... 2022-12-31

▼ **Data variables:**

precip	(time, lat, lon)	float32	...

▶ Indexes: (3)

▼ Attributes:

Conventions :	CF-1.0
version :	V1.0
title :	CPC GLOBAL PRCP V1.0 RT
References :	https://www.psl.noaa.gov/data/gridded/data.cpc.globalprecip.html
dataset_title :	CPC GLOBAL PRCP V1.0
Source :	ftp://ftp.cpc.ncep.noaa.gov/precip/CPC_UNI_PRCP/
history :	Updated 2023-01-02 23:31:13

Precipitation Data: CPC 0.5x0.5 Global Daily

```
dataset.precip.values ...
```

```
array([[[       nan,        nan,        nan, ...,        nan,
                nan,        nan],
        [       nan,        nan,        nan, ...,        nan,
                nan,        nan],
        [       nan,        nan,        nan, ...,        nan,
                nan,        nan],
        ...,
        [0.        , 0.        , 0.        , ..., 0.        ,
         0.        , 0.        ],
        [0.        , 0.        , 0.        , ..., 0.        ,
         0.        , 0.        ],
        [0.05244859, 0.05249586, 0.05254338, ..., 0.04819115,
         0.05235494, 0.05240161]],
```

Precipitation Data: CPC 0.5x0.5 Global Daily

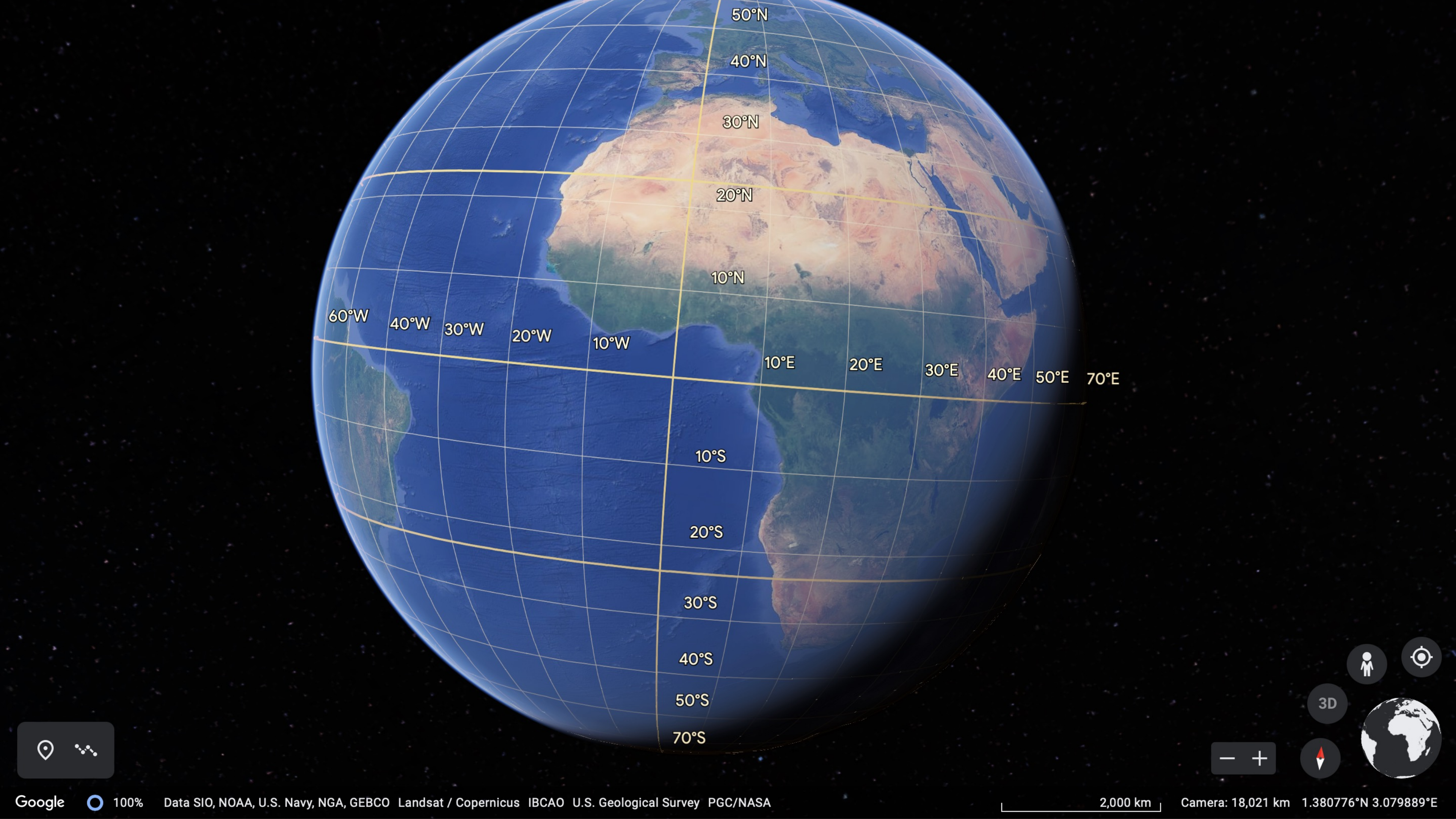
```
precip_values.shape
(365, 360, 720)
```

Days
1 - 365

Latitude
89.5N - 89.5S

Longitude
0.25E - 359.75E

Center for Geographic Analysis
Harvard University


50°N
40°N
30°N
20°N
10°N
60°W
40°W
30°W
20°W
10°W
10°E
20°E
30°E
40°E
50°E
70°E
10°S
20°S
30°S
40°S
50°S
70°S
3D
Google
100%
Data SIO, NOAA, U.S. Navy, NGA, GEBCO Landsat / Copernicus IBCAO U.S. Geological Survey PGC/NASA
2,000 km
Camera: 18,021 km 1.380776°N 3.079889°E

Case Study Data

- TSGI
  - `date, country, state, county, sentiment_score, tweets`
- Precipitation
  - `day of year, latitude, longitude`
- How to connect the two?

Center for Geographic Analysis
Harvard University

Join Table Needed

- Using a list of counties with coordinates
  - Initial idea: web search
  - https://en.wikipedia.org/wiki/User:Michael_J/County_table

User:Michael J/County table

From Wikipedia, the free encyclopedia

< User:Michael J

You have **a new message (last change).**

Table of United States counties [edit]

This is a table adapted from the U.S. Census Bureau's gazetteer of county populations and areas, with April 1, 2010 population counts added from elsewhere on the Census Bureau site. It has been augmented with county seats and a few explanatory footnotes.

Sort [1]	State	FIPS	County [2]	County Seat(s) [3]	Population (2010)	Land Area km²	Land Area mi²	Water Area km²	Water Area mi²	Total Area km²	Total Area mi²	Latitude	Longitude
1	AL	01001	Autauga	Prattville	54,571	1,539.582	594.436	25.776	9.952	1,565.358	604.388	+32.536382°	–86.644490°
2	AL	01003	Baldwin	Bay Minette	182,265	4,117.522	1,589.784	1,133.190	437.527	5,250.712	2,027.311	+30.659218°	–87.746067°
3	AL	01005	Barbour	Clayton	27,457	2,291.819	884.876	50.865	19.639	2,342.684	904.515	+31.870670°	–85.405456°
4	AL	01007	Bibb	Centreville	22,915	1,612.481	622.582	9.289	3.587	1,621.770	626.169	+33.015893°	–87.127148°
5	AL	01009	Blount	Oneonta	57,322	1,669.962	644.776	15.157	5.852	1,685.119	650.628	+33.977448°	–86.567246°
6	AL	01011	Bullock	Union Springs	10,914	1,613.057	622.805	6.057	2.338	1,619.113	625.143	+32.101759°	–85.717261°
7	AL	01013	Butler	Greenville	20,947	2,011.977	776.829	2.727	1.053	2,014.704	777.882	+31.751667°	–86.681969°
8	AL	01015	Calhoun	Anniston	118,572	1,569.190	605.868	16.624	6.419	1,585.814	612.287	+33.771706°	–85.822513°
9	AL	01017	Chambers	Lafayette	34,215	1,545.009	596.531	17.048	6.582	1,562.057	603.113	+32.917943°	–85.391812°
10	AL	01019	Cherokee	Centre	25,989	1,434.076	553.700	119.859	46.278	1,553.935	599.978	+34.069515°	–85.654242°

Join Table Needed

- Using a list of counties with coordinates
  - ~~Initial idea: web search~~

- Geocoding
  - “Geocoding refers to the assignment of geocodes or coordinates to geographically reference data provided in a textual format.” – https://en.wikipedia.org/wiki/Geocode

Join Table

- 41,765 pairs
  https://www.here.com/platform/geocoding
- Provides coordinates for each country-state-country pair
- Different providers, “Here Maps” in this case
  - Demo code in repo `geocoding example.py`

Country	State	County	lat	lon
Afghanistan	Badakhshan	Baharak	36.9624	70.86874
Afghanistan	Badakhshan	Ishkashim	36.97669	71.45003
Afghanistan	Badakhshan	Kishim	36.82093	70.09848
Afghanistan	Badakhshan	Shighnan	37.5589	71.48942
Afghanistan	Badakhshan	Zebak	36.70442	71.57036
Afghanistan	Badghis	Ghormach	35.76058	63.593
Afghanistan	Badghis	Jawand	35.07173	64.12571
Afghanistan	Baghlan	Andarab	35.62599	69.18434
Afghanistan	Baghlan	Baghlan City	36.14867	68.72341
Afghanistan	Baghlan	Baghlani Jadid	36.14867	68.72341
Afghanistan	Baghlan	Burka	36.14867	68.72341
Afghanistan	Baghlan	Doshi	36.14867	68.72341
Afghanistan	Baghlan	Khinjan	36.14867	68.72341
Afghanistan	Baghlan	Nahrin	36.14867	68.72341
Afghanistan	Baghlan	Puli Khumri	35.96299	68.70387
Afghanistan	Balkh	Balkh	34.53313	69.10221
Afghanistan	Balkh	Kaldar	34.53313	69.10221
Afghanistan	Balkh	Khulm	34.53313	69.10221
Afghanistan	Balkh	Mazar-i-Sharif	36.70745	67.10885

Case Study Data

- TSGI
  - `date, country, state, county, sentiment_score, tweets`
- Precipitation
  - `day of year, latitude, longitude`
- How to connect the two?

Center for
Geographic Analysis
Harvard University

Case Study Data

- TSGI
  - `date, country, state, county, sentiment_score, tweets`
- Join table
  - `country, state, county, latitude, longitude`
- Precipitation
  - `day of year, latitude, longitude`

Center for Geographic Analysis
Harvard University

How to work with big data sets?

1. Develop toy example locally
2. Move data & code to the Cluster
3. Scale example to use entire data set

Center for
Geographic Analysis
Harvard University

Toy Example & Scaling

- Understand the problem & the how to use the data

- Develop a toy example
  - Vertical prototype
  - “Divide and conquer”
    - Solve the problem using a manageable subset of the data
  - Allows you to work on your local machine
    - Faster development iterations
    - Your favorite development environment

Toy Example & Scaling

- Transfer toy example to HPC environment
  - Test if everything works as expected
    - “Fail fast, fail often”
  - Note differences: stdout, error messages, results, …

- Scale to the entire dataset
  - Allocate HPC resources

Exercise - Develop a Toy Example I

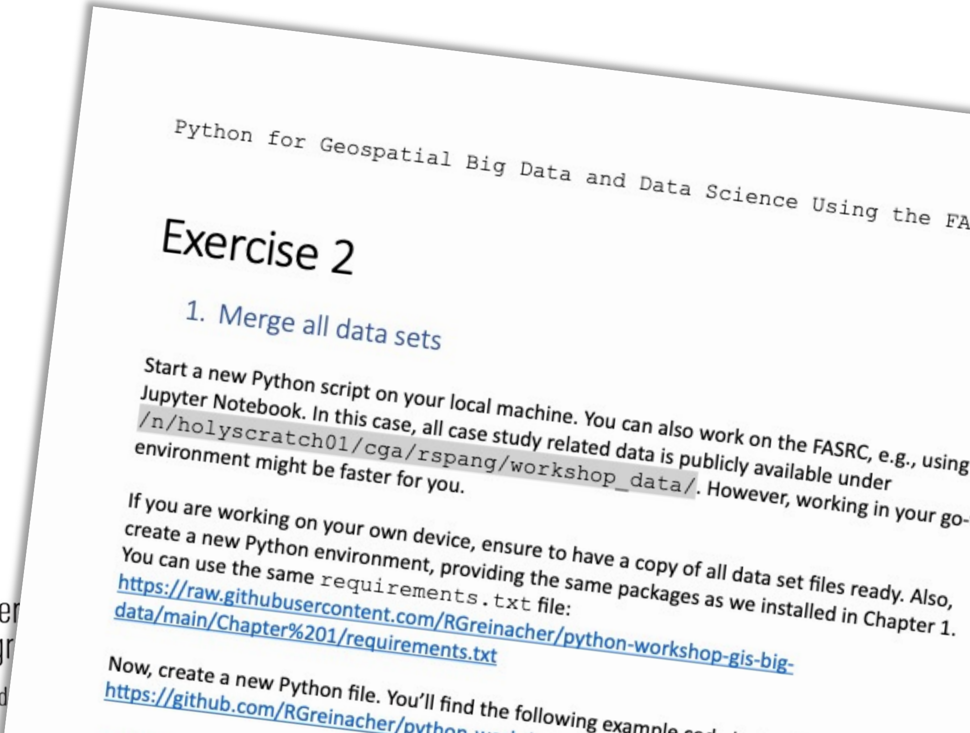
- Merge all datasets
  - Develop this locally on your device
  - Load one TSGI file
    - E.g. year 2022
    - E.g. only Massachusetts counties, only east coast counties, …
  - Load one NOAA CPC file (same year)
  - Merge both using the join table

Exercise - Develop a Toy Example II

- Analyze the dataset
  - For each county
    - Collect all rainy days → compute average sentiment score
    - Collect all non-rainy days → compute average sentiment score
    - Return difference
  - Aggregate differences per country

Exercise

Find the exercise document in the GitHub repo



Summary of Chapter 2

- Openly available data sets
  - TSGI & NOAA CPC precipitation
  - Join both using a geocoding look-up table
  - Translate county coordinates to best-matching precipitation data
- Toy examples locally, full scale version remotely
- We now have a functional prototype for our research question

Chapter 2
Questions & Comments?

5min coffee break

Center for
Geographic Analysis
Harvard University

Chapter 3
Moving to the FASRC

Python for Geospatial Big Data and Data Science Using the FASRC

Center for
Geographic Analysis
Harvard University

Environment Differences: Local vs. HPC

- Parallelism
- Memory
- Storage
- **Environment and Dependencies**
- **Batch Systems**
- **Data Transfer**
- **Error Handling and Debugging**
- Optimizations
- Networking
- Code Scalability

Center for Geographic Analysis
Harvard University

Copy your Script to the FASRC

- Using the command line or the web interface
  - `scp -r ./* USER@login.rc.fas.harvard.edu:/DEST/`
  - Replace `USER` and `/DEST/`
  - Alternatively: Use the Jupyter web interface to create a new Python file

- Test in an interactive session
  - Make sure you run code only on compute nodes
  - Use the workshop environment we created earlier

Center for
Geographic Analysis
Harvard University

Feedback & Output

- Feedback is not as easy to obtain, especially when running asynchronously
  - Add print-statements to your code to log important events
    - Loaded dataset A, B, C
    - Finished augmenting tweets with coordinates
    - Finished analysis
    - Finished running
  - Write your results to a file
    - Can’t be the dataset-source directory
    - Home folder is usable for results files

SLURM's sbatch Jobs

- Interactive shells good for testing, but submittings jobs is main way to run code
  - Write a script that defines what should be done
  - The script also defines the requirements (CPU, mem, …)
  - Submit job via sbatch myscript.sh

https://docs.rc.fas.harvard.edu/kb/running-jobs/#articleTOC_8

SLURM’s sbatch Jobs

```
#!/bin/bash
# https://docs.rc.fas.harvard.edu/kb/running-jobs/#articleTOC_8

#SBATCH -c 1                    # Number of cores (-c)
#SBATCH -t 0-00:10              # Runtime in D-HH:MM, minimum of 10 minutes
#SBATCH -p test                 # Partition to submit to
#SBATCH --mem=4000             # Memory pool for all cores (see also --mem-per-cpu)
#SBATCH -o /n/home01/rspang/results/job_stdout_%j.out  # File to which STDOUT will be written, %j inserts jobid
#SBATCH -e /n/home01/rspang/results/job_errout_%j.err  # File to which STDERR will be written, %j inserts jobid

# load modules
module load Mambaforge/23.3.1-fasrc01

# set python environmant
mamba activate workshop

# run code
python precipitation_sentiment_toy_example_chap3.py
```

Monitoring

- Monitor the load on a system
  - Command: `htop`, or command: `ps -U username`
  - E.g. run a job in an interactive session;
    have a second SSH connection to monitor the load

- List running SLURM jobs
  - Command: `squeue`
  - Monitor if a job is (still) running from a login node

Center for Geographic Analysis

Harvard University

Exercise I

- Transfer your Python script to the FASRC
  - Copy your code to the FASRC
  - Adjust paths (to dataset files) to the FASRC locations
- Run your script in an interactive session
  - Write results to a file
  - Time the execution
  - Compare speed with local machine

Exercise II

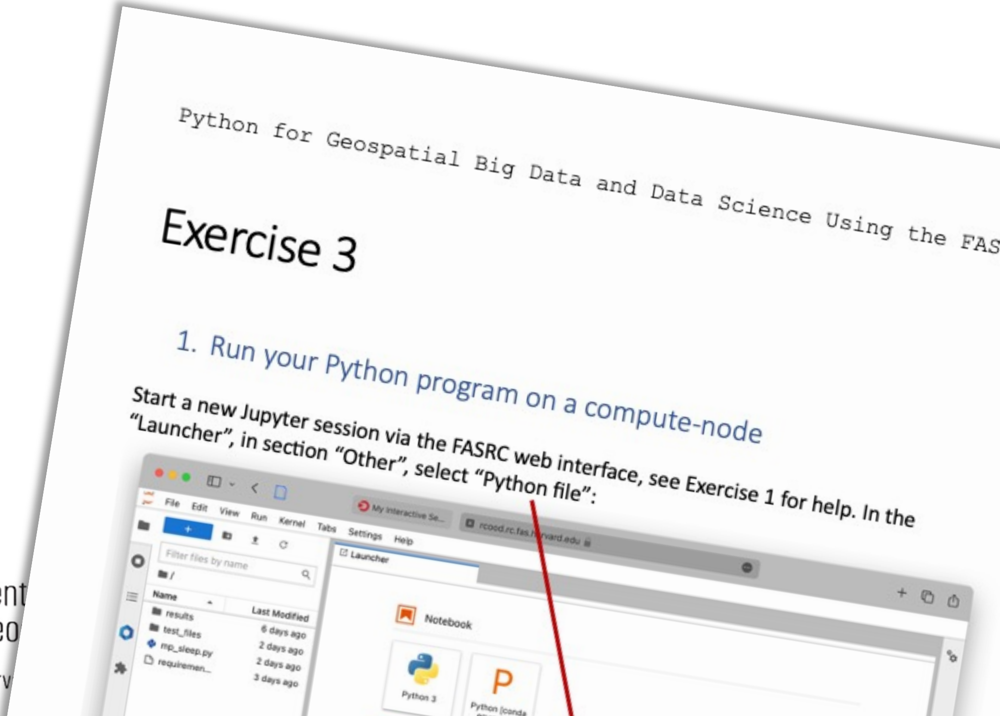
- Run your script as a sbatch job
  - Submit your job for asynchronous execution

- Monitor multiple processes using htop
  - While the program is running on a compute-node, open `https://rcood.rc.fas.harvard.edu/pun/sys/shell/ssh/COMPUTE_NODE.rc.fas.harvard.edu` in a browser; replace “COMPUTE_NODE” with the node-ID you are connected to

Center for Geographic Analysis
Harvard University

Exercise

Find the exercise document in the GitHub repo


Python for Geospatial Big Data and Data Science Using the FAS

Exercise 3

1. Run your Python program on a compute-node

Start a new Jupyter session via the FASRC web interface, see Exercise 1 for help. In the "Launcher", in section "Other", select "Python file":

Summary of Chapter 3

- Transferring scripts to remote environment requires small code changes
  - Paths to files
  - Logs
  - Results
- Running scripts interactively or asynchronously

Chapter 3
Questions & Comments?

1h lunch break

Center for
Geographic Analysis
Harvard University

Chapter 4
Performance Optimization

Python for Geospatial Big Data and Data Science Using the FASRC

Center for Geographic Analysis
Harvard University

Compare Runtime: Local vs. FASRC

- Which system was faster?
  - My laptop: `this script took 0:00:10.92sec to execute`
  - FASRC: `this script took 0:00:18.83sec to execute`

- Why?

Environment Differences: Local vs. HPC

- **Parallelism**
- **Memory**
- **Storage**
- Environment and Dependencies
- Batch Systems
- Data Transfer
- Error Handling and Debugging
- **Optimizations**
- Networking
- **Code Scalability**

Center for Geographic Analysis
Harvard University

Code Optimization

- Optimize your code
  - I/O
  - Memory
  - Storage
  - Use parallelization if possible

- Test locally first (toy example approach)

Code Efficiency: Dynamic Programming

Example: computing the Fibonacci sequence

```python
from datetime import datetime

# Naive, recursive implementation
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)

start = datetime.now()
print(fib(41))
end = datetime.now()
print(end - start)
```

Fib(41) ~ 31sec

Fib(112) ~ age of the universe

```python
from datetime import datetime

# Dynamic Programming: Memoization (still recursive!)
def fib(n, memo={}):
    if n <= 1:
        return n
    elif n not in memo:
        memo[n] = fib(n-1, memo) + fib(n-2, memo)
    return memo[n]

start = datetime.now()
print(fib(41))
end = datetime.now()
print(end - start)
```

Fib(41) ~ 0.000094sec

Fib(120) ~ 0.000112sec

Code Efficiency: Parallelization

- Use multi-core CPUs
  - By default, a Python script runs on a single core
  - Modern CPUs have multiple cores → distribute load

Simple example: summing a large list of numbers

```
[1] -> [2] -> [3] -> [4] -> [5] -> [6] -> ... -> [N]
 \___ Worker (CPU Core 1) moving sequentially, summing up.
```

```
Segment 1          Segment 2          Segment 3              Segment k
[1] -> [2]         [3] -> [4]         [5] -> [6]    ...      [N-1] -> [N]
 \___ Worker 1      \___ Worker 2      \___ Worker 3          \___ Worker k
```

Code Efficiency: Parallelization

Example: summing a large list of numbers

```
numbers = range(10000)
total = sum(numbers)
```

Serial computing

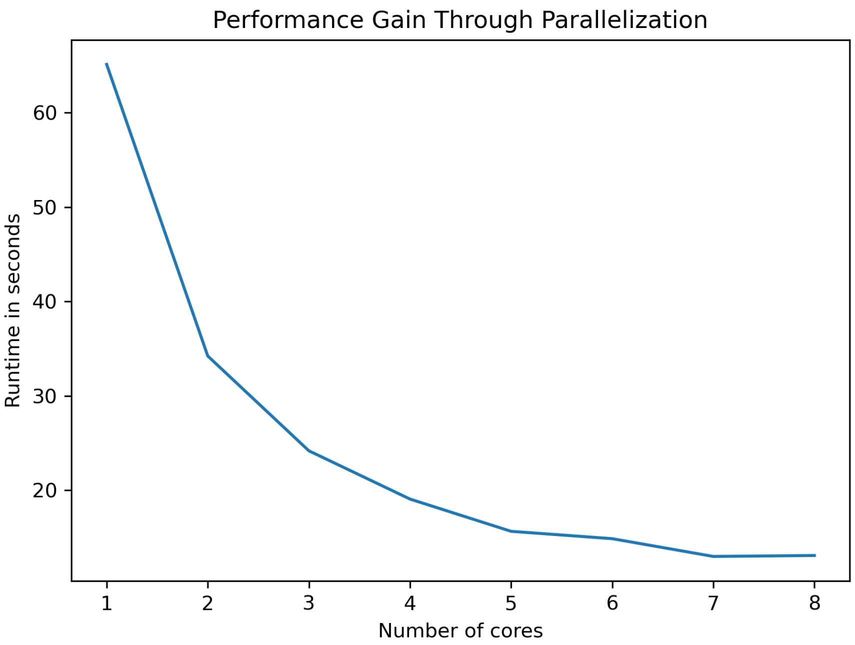
```
from concurrent.futures import ProcessPoolExecutor

numbers = range(10000)
segment_size = len(numbers) // 4 # assuming 4 CPU cores

with ProcessPoolExecutor() as executor:
    segments = [numbers[i:i + segment_size] for i in
    range(0, len(numbers), segment_size)]
    partial_sums = list(executor.map(sum, segments))
    total = sum(partial_sums)
```

Parallel computing

Code Efficiency: Parallelization



# Cores	Runtime in sec	Speedup
1	65.10	1.00
2	34.21	1.90
3	24.17	2.69
4	19.06	3.42
5	15.64	4.16
6	14.86	4.38
7	12.98	5.01
8	13.08	4.98

Center for Geographic Analysis
Harvard University

Scaling on a HPC

- Vertical scaling
  - Using more resources on a node
  - “scaling up”

- Horizontal scaling
  - Growing the system by connecting multiple hardware entities to distribute the load across nodes
  - “scaling out”

Exercise I

- Screen your code for optimization potential
  - Test which parts take the most time
  - Check if you run the same code repeatedly
  - Think about parallelization: which parts could profit from parallel execution?
  - What improvement might have the biggest impact on the performance of your script?

Exercise II

- Implement parallelization using `joblib`
  - When using more resources, adjust your toy example
    - E.g., run the analysis on data of the entire U.S.
  - https://joblib.readthedocs.io/en/stable/

- Run the precipitation analysis for the entire world
  - Use all the data of one year, or even all the data from 2012-2023
  - Which country is the most “weather-sensitive”?

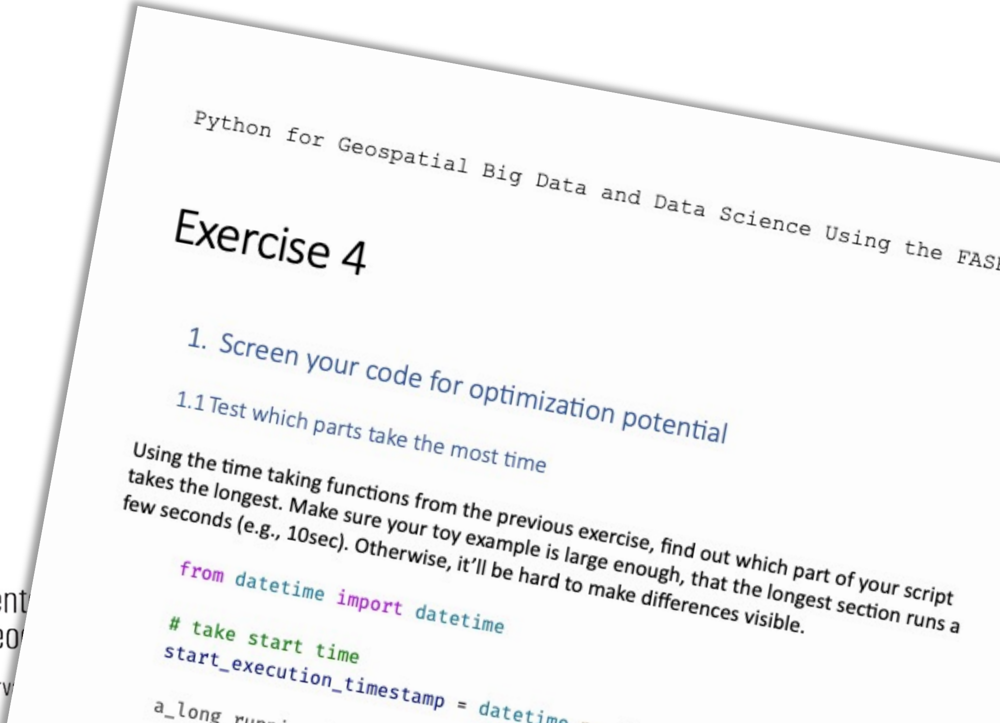
Center for Geographic Analysis
Harvard University

Bonus Exercise

- Improve the data analysis
    - Drop days with only very few tweets (e.g., having at least 10)
    - Include only such counties, that have at least 20 rainy days
    - Add a statistical analyses
        - effect size of difference (e.g., Cohen's *d*)
        - significance testing (e.g., Welch's *p*-test)
    - Only compare days with at least three days rain / no rain in a row

Exercise

Find the exercise document in the GitHub repo


Python for Geospatial Big Data and Data Science Using the FAS

Exercise 4

1. Screen your code for optimization potential

1.1 Test which parts take the most time

Using the time taking functions from the previous exercise, find out which part of your script takes the longest. Make sure your toy example is large enough, that the longest section runs a few seconds (e.g., 10sec). Otherwise, it'll be hard to make differences visible.

```
from datetime import datetime

# take start time
start_execution_timestamp = datetime
```

Summary of Chapter 4

- Optimization is complex
  - Trade-off between code engineering and runtime
  - **But: necessary to leverage a HPC!**

- No off-the-shelf solution
  - Reducing I/O (loading data) & inspecting redundancies is always a good idea
  - Understanding the concept of **parallelization** (and looking for opportunities to use it) is worthwhile

Chapter 4
Questions & Comments?

5min coffee break

Center for
Geographic Analysis
Harvard University

Chapter 5
Recap, Wrap-up & Outro

Python for Geospatial Big Data and Data Science Using the FASRC

Center for
Geographic Analysis
Harvard University

Status Update

- What were you working on, how far did you get?
- What’s your most important take away?

Center for
Geographic Analysis
Harvard University

Workshop Ad

- Topics Covered:
  - Fundamentals of High-Performance Computing, with a focus on FASRC
  - Foundations of Data Science
  - Big Data Concepts using Python
  - Practical application: large social media data set

- Learning Objectives:
  - Learn how to analyze large data sets using Python and FASRC
  - Various tools and techniques of Data Science and Big Data computation
  - Prepare to work with your own data using the FASRC

Center for
Geographic Analysis
Harvard University

Wishes and Interests

- TODO

Open Questions?

- Get in touch
  - robertspang@fas.harvard.edu
  - CGA, K00A, 1737 Cambridge Street
  - Until 13th of October

Center for
Geographic Analysis
Harvard University

Inspirations: CGA's Big Data Projects

- Complex geospatial analysis
- Scale geospatial applications on cluster and cloud computing environments
- Geospatial databases (PostGIS, OmniSci)
- Visualization of large geospatial data using GPU databases

→ https://gis.harvard.edu/gis-data-science-big-data-workstream-cga

Center for Geographic Analysis
Harvard University

Feedback

Please take a moment to let us know how we could do better and what you liked about today

→ https://gis.harvard.edu/workshop-evaluation

Center for Geographic Analysis
Harvard University

Chapter 5
Questions & Comments?

Thank you for joining us!

Center for
Geographic Analysis
Harvard University