Scalable and Computationally Reproducible Approaches to Arctic Research

Matt Jones, Bryce Mecum, Jeanette Clark, Sam Csik

September 19, 2022

Table of contents

Preface

About

This 5-day in-person workshop will provide researchers with an introduction to advanced topics in computationally reproducible research in python and R, including software and techniques for working with very large datasets. This includes working in cloud computing environments, docker containers, and parallel processing using tools like parsl and dask. The workshop will also cover concrete methods for documenting and uploading data to the Arctic Data Center, advanced approaches to tracking data provenance, responsible research and data management practices including data sovereignty and the CARE principles, and ethical concerns with data-intensive modeling and analysis.

Schedule

Code of Conduct

Please note that by participating in this activity you agree to abide by the NCEAS Code of Conduct.

	Monday	Tuesday	Wednesday	Thursday	Friday
08:00-08:30	Coffee (optional)	Coffee (optional)	Coffee (optional)	Coffee (optional)	Coffee (optional)
08:30-09:00 09:00-09:30	**1. Welcome and Course Overview** (Jeanette)	**6. Data structures and formats for large data** (Bryce)	**10. Spatial and Image Data using GeoPandas** (Jeanette)	**15. Google Earth Engine** (Ingmar, Sam)	**19. What is cloud computing anyways?** (Matt)
09:30-10:00 10:00-10:30	**2. Remote computing** (Sam)		**11. Data futures: Parquet and Arrow** (Jeanette)		
10:30-11:00	BREAK	BREAK	BREAK	BREAK	BREAK
11:00-11:30	**3. Python programming on clusters** (Jeanette)	**7. Parallelization with Dask** (Bryce)	**12. Software Design II** (Bryce)	**16. Billions of Ice Wedge Polygons** (Chandi)	**20. Reproducibility redux via containers** (Bryce)
11:30-12:00					**Survey Feedback** **Q & A**
12:00-12:30 12:30-13:00	Lunch	Lunch	Lunch	Lunch	Adjourn
13:00-13:30 13:30-14:00 14:00-14:30 14:30-15:00	**4. Pleasingly Parallel Programming** (Matt)	**8. Group project I** **Data staging and pre-processing** (Jeanette)	**13. Group project II** **Parallel data processing** (Jeanette)	**17. Group project III** **Visualizing big geospatial data** (Jeanette)	
15:00-15:30	Break	Break	Break	Break	
15:30-16:00	**5. Documenting and Publishing Data** (Daphne)	**9. Software design I** (Bryce)	**14. Data Ethics** (Matt)	**18. Workflows for data staging and publishing** (Jeanette)	
16:00-16:30			**Breather** Catch-up		
16:30-17:00	Q&A	Q&A	Q&A	Q&A	

Setting Up

In this course, we will be using Python (> 3.0) as our primary language, and VS Code as our IDE. Below are instructions on how to get VS Code set up to work for the course. If you are already a regular Python user, you may already have another IDE set up. We strongly encourage you to set up VS Code with us, because we will use your local VS Code instance to write and execute code on one of the NCEAS servers.

Download VS Code and Extensions

First, download VS Code if you do not already have it installed.

Check to make sure you have Python installed if you aren't sure you do. To do this, from the terminal run:

```
python3 --version
```

If you get an error, it means you need to install Python. Here are instructions for getting installed, depending on your operating system. Note: There are many ways to install and manage your Python installations, and advantages and drawbacks to each. If you are unsure about how to proceed, feel free to reach out to the instructor team for guidance.

- Windows: Download and run an installer from Python.org.
- Mac: Install using homebrew. If you don't have homebrew installed, follow the instructions from their webpage.
  - `brew install python3`

After you run your install, make sure you check that the install is on your system PATH by running `python3 --version` again.

Set up VS Code

This section summarizes the official VS Code tutorial. For more detailed instructions and screenshots, see the source material

First, install the Python extension for VS Code.

Open a terminal window in VS Code from the Terminal drop down in the main window. Run the following commands to initialize a project workspace in a directory called `training`. This example will show you how to do this locally. Later, we will show you how to set it up on the remote server with only one additional step.

```
mkdir training
cd training
code .
```

Next, we will select the Python interpreter for the project. Open the **Command Palette** using Command + Shift + P (Control + Shift + P for windows). The Command Palette is a handy tool in VS Code that allows you to quickly find commands to VS Code, like editor commands, file edit and open commands, settings, etc. In the Command Palette, type "Python: Select Interpreter." Push return to select the command, and then select the interpreter you want to use (your Python 3.X installation).

Finally, download the Jupyter extension. You can create a test Jupyter Notebook document from the command pallete by typing "Create: New Jupyter Notebook" and selecting the command. This will open up a code editor pane with a notebook that you can test.

Test your local setup (Optional)

To make sure you can write and execute code in your project, create a Hello World test file.

- From the File Explorer toolbar, or using the terminal, create a file called `hello.py`

- Add some test code to the file, and save

```
msg = "Hello World"
print(msg)
```

- Execute the script using either the Play button in the upper-right hand side of your window, or by running `python3 hello.py` in the terminal.
  - For more ways to run code in VS Code, see the tutorial

About this book

These written materials reflect the continuous development of learning materials at the Arctic Data Center and NCEAS to support individuals to understand, adopt, and apply ethical open science practices. In bringing these materials together we recognize that many individuals have contributed to their development. The primary authors are listed alphabetically in the citation below, with additional contributors recognized for their role in developing previous iterations of these or similar materials.

Citation: Matthew B. Jones, Bryce Mecum, S. Jeanette Clark, Samantha Csik. 2022. Scalable and Computationally Reproducible Approaches to Arctic Research.

Additional contributors: Amber E. Budden, Natasha Haycock-Chavez, Noor Johnson, Stephanie Hampton, Jim Regetz, Bryce Mecum, Julien Brun, Julie Lowndes, Erin McLean, Andrew Barrett, David LeBauer, Jessica Guo.

This is a Quarto book. To learn more about Quarto books visit https://quarto.org/docs/books.

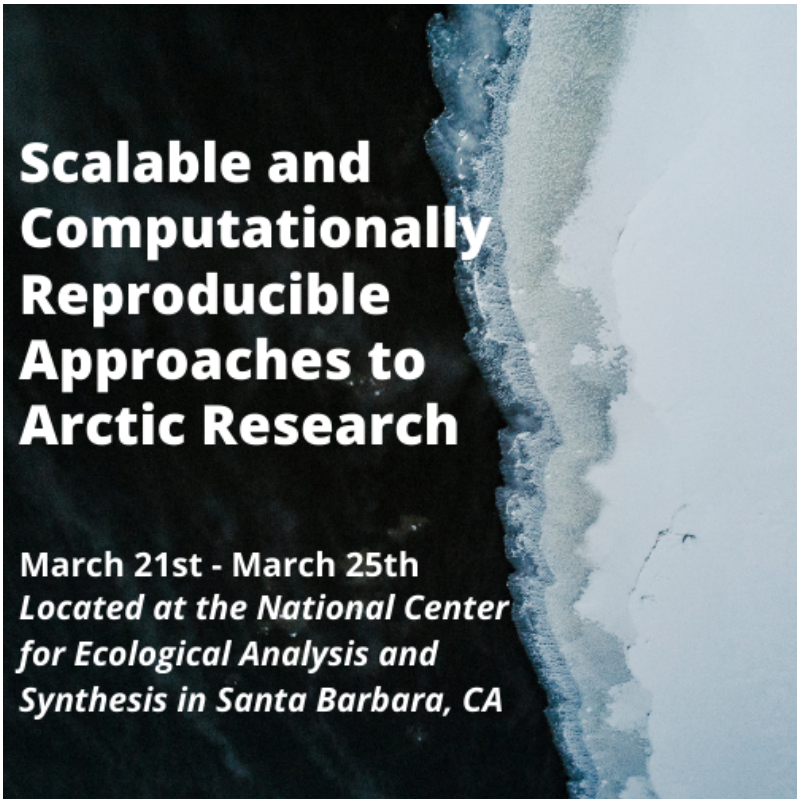
1 Welcome and Introductions

Jeanette Clark

This course is one of three that we are currently offering, covering fundamentals of open data sharing, reproducible research, ethical data use and reuse, and scalable computing for reusing large data sets.


Fundamentals in
Data Management
for Qualitative
and Quantitative
Arctic Research
April 18th - April 22nd
Located at the National Center for Ecological
Analysis and Synthesis in Santa Barbara, CA


Reproducible Practices for Arctic Research Using R
February 14th - February 18th, 2022
This course will be taught virtually


Scalable and Computationally Reproducible Approaches to Arctic Research
March 21st - March 25th
Located at the National Center for Ecological Analysis and Synthesis in Santa Barbara, CA

2 Remote Computing

Sam Csik

Notes from Google Sheet (DELETE LATER)
- Servers & Networking
- IP addressing
- Bash shell programming
- SSH
- Remote session in VS Code

- Understand the basic architecture of computer networks
- Become familiarized with Bash Shell programming to navigate your computer's file system **(??)**
- Learn how to connect to a remote computer via a shell

2.1 Introduction

- Scientific synthesis and our ability to effectively and efficiently work with big data depends on the use of computers & the internet
- VS Code + remote development on a cluster is easy and way faster than your local machine

2.2 Servers & Networking

- Host computers connect via networking equipment and can send messages to each other over communication protocols (aka internet protocols)

- **Client:** the host *initiating* the request
- **Server:** the host *responding* to a request

2.3 IP addressing

- Hosts are assigned a **unique numerical address** used for all communication and routing called an Internet Protocol Address (IP Address). They look something like this: 128.111.220.7
- Each IP Address can be used to communicate over various "ports", which allows multiple applications to communicate with a host without mixing up traffic
- IP addresses can be difficult to remember, so they are also assigned **hostnames**
  - Hostnames are handled through the global Domain Name System (DNS)
  - Clients first look up a hostname in DNS to find the IP address, then they open a connection the the IP address
    - aurora.nceas.ucsb.edu == 128.111.220.46 **(UPDATE THIS WITH SERVER USED FOR COURSE?)**

2.4 Bash Shell Programming

- *What is a shell?* From Wikipedia

  "a computer program which exposes an operating system's services to a human user or other programs. In general, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI), depending on a computer's role and particular operation."

- *What is Bash Shell?* A command line tool (language) commonly used to manipulate files and directories
  - **Mac:** bash via the Terminal **(QUESTION: Mac users may have to switch from zsh to bash? `exec bash`? or `exec zsh` to switch back)**
  - **Windows:** bash via Git Bash

2.5 Some group exercise:

- Navigate file system (show that this is equivalent to using Finder/Windows version), create a file, edit file, etc.
  - `pwd`
  - `cd`
  - `ls`
  - `touch`
  - `mkdir`
  - (**Queston:** Do we want/need to show all of these? Missing any important ones?)

2.6 Connecting to a remote computer via a shell

- You can use a shell to gain accesss to and remotely control (manage/transfer files/etc) other computers. To do so, you'll need the following:
  - remote computer (e.g. server) turned on
  - IP address or name of remote computer
  - necessary permissions to access the remote computer
- Secure Shell, or SSH, is often used for securely connecting to and running shell commands on a remote host
  - Tremendously simplifies remote computing
  - Supported out-of-the-box on Linux and Macs

2.7 Exercise:

1. Launch your Terimal program:
    - **MacOS:** navigate to `Applications | Utilities` and open `Terminal`
    - **Windows:** Navigate to `Windows Start | Git` and open `Git Bash`
    - **ALTERNATIVELY, from VS Code:** Two options to open a terminal program
        a) Click on `Terminal | New Terminal` in top menu bar
        b) Click on the `+ (dropdown menu) | bash` in the bottom right corner **(QUESTION: Not sure that is always open/available depending on user configurations??)**
2. Connect to a remote server **(UPDATE THIS SECTION)**

```
jones@powder:~$ ssh jones@aurora.nceas.ucsb.edu
jones@aurora.nceas.ucsb.edu's password:
jones@aurora:~$
```

3. Change your password **(UPDATE THIS SECTION)**

```
jones@aurora:~$ passwd
Changing password for jones.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
```

4. **create python script on server | write/execute some code | etc**

3 Python Programming on Clusters

Jeanette Clark

- Basic Python review
- Using virtual environments
- Writing in Jupyter notebooks
- Writing functions in Python

3.1 Introduction

- VS Code + remote development on a cluster is easy and way faster than your local machine
- Jupyter is a great way to do literate analysis
- Functions provide ways to reuse your code across notebooks/projects

3.2 Python on the cluster

- Connect to the server
- Start a `training` project and pick interpreter (this could also go in Sam's session)
- Create and execute `hello.py`
  - from the IDE as a whole
  - from IDE line by line
  - from the terminal

3.3 Virtual Environments

Why virtual environments? We'll answer this.

First install `virtualenvwrapper`

```
pip3 install virtualenvwrapper
```

Now we will create the `.bash_profile` file to create variables that point to the install locations of python and `virtualenvwrapper`.

In VS Code, select "File > New Text File" then paste this into the file:

```
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
export VIRTUALENVWRAPPER_VIRTUAWORKON_HOME=$HOME/.virtualenvs
source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
```

Save the file in the top of your user directory as `.bash_profile`.

Restart your terminal, then check to make sure it was installed and configured correctly

```
mkvirtualenv --version
```

Now we can create the virtual environment we will use for the course

```
mkvirtualenv scomp
```

You'll see the name of the env you are working in on the left side of your terminal prompt in parentheses.

Now lets install the dependencies for this course into that environment. (Note: need to figure out how to get them this file)

```
python3 -m pip install -r requirements.txt
```

To deactivate your environment (like if you want to work on a different project), just run `deactivate`. To activate it again, run

```
workon scomp
```

3.4 Brief overview of python syntax

- lists, arrays, dictionaries, associative arrays

3.5 Jupyter notebooks

- Create a notebook
- Load in some libraries (pandas, numpy, scipy, matplotlib)
- Read in a csv
- group and summarize by a variable
- create a simple plot

3.6 Functions

- create `myplot.py`
- write `myplot()` function to create the same plot we did in section above
- load myplot into jupyter notebook (`from myplot.py import myplot`)
- replace old plot method with new function
- more to come in Bryce's section

3.7 Resources

4 Pleasingly Parallel Programming

Matt Jones

4.1 Introduction

5 Documenting and Publishing Data

Daphne Virlar-Knight, Natasha Haycock-Chavez, Amber Budden, Matt Jones

5.1 Introduction

6 Spatial and Image Data Using GeoPandas

Jeanette Clark

- Reading raster data with rasterasterio
- Using geopandas and rasterasterio to process raster data
- Working with raster and vector data together

6.1 Introduction

- Raster vs vector data
- What is a projection
- Processing overview
  - goal is to calculate vessel distance per commercial fishing area

6.2 Pre-processing raster data

This is a test to make sure we can run some code in this notebook.

```
import geopandas as gpd
import rasterio
import rasterio.mask
import rasterio.warp
import rasterio.plot
from rasterio import features
```

```
from shapely.geometry import box
from shapely.geometry import Polygon
import requests
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import style
import pandas as pd
import numpy as np
```

Download the ship traffic raster from Kapsar et al.. We grab a one month slice from December, 2020 of a coastal subset of data with 1km resolution.

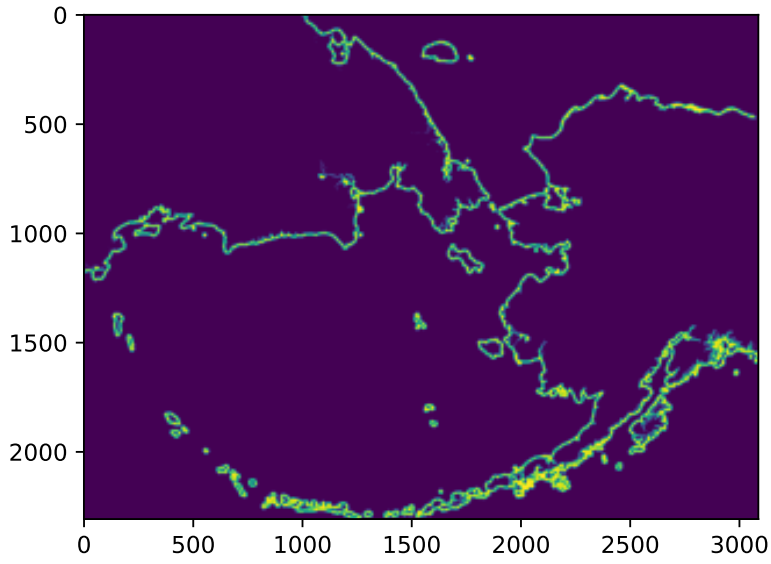
```
url_sf = 'https://cn.dataone.org/cn/v2/resolve/urn:uuid:dd61089d-f50e-4d87-9b75-6b4e2bd24776

response_sf = requests.get(url_sf)
open("Coastal_2020_12.tif", "wb").write(response_sf.content)
```

```
1132748
```

Open the raster file, plot it, and look at the metadata.

```
with rasterio.open("Coastal_2020_12.tif") as dem_src:
    ships = dem_src.read(1)
    ships_meta = dem_src.profile

plt.imshow(ships)
print(ships_meta)
```

```
{'driver': 'GTiff', 'dtype': 'float32', 'nodata': -3.3999999521443642e+38, 'width': 3087, 'heig
       0.0, -999.9687691991521, 2711703.104608573), 'tiled': False, 'compress': 'lzw', 'interle
```



6.3 Distance per commercial area

Now download a vector shapefile of commercial fishing districts in Alaska.

```
url = 'https://knb.ecoinformatics.org/knb/d1/mn/v2/object/urn%3Auuid%3A7c942c45-1539-4d47-b4

response = requests.get(url)
open("Alaska_Commercial_Salmon_Boundaries.gpkg", "wb").write(response.content)
```

```
36544512
```

Read in the data

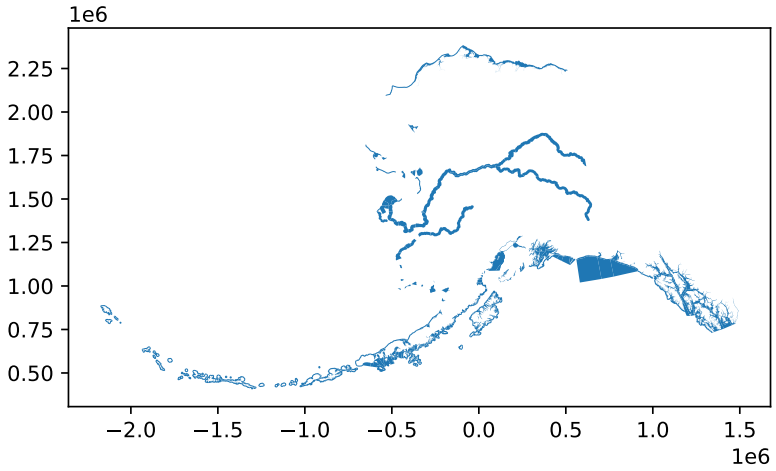
```
comm = gpd.read_file("Alaska_Commercial_Salmon_Boundaries.gpkg")
```

The raster data is in 3338, so we need to reproject this.

```
comm.crs
comm_3338 = comm.to_crs("EPSG:3338")
```

```
comm_3338.plot()
```

```
<AxesSubplot:>
```



We can extract the bounding box for the area of interest, and use that to clip the original raster data to just the extent we need. We use the `box` function from `shapely` to create the bounding box, then create a `geoDataFrame` from them and convert the WGS84 coordinates to the Alaska Albers projection.

todo: explain the warp transform thing here

```
coords = rasterio.warp.transform_bounds('EPSG:4326',
                                        'EPSG:3338',
                                         -153.5,
                                         56,
                                         -142.5,
                                         62)
coord_list = list(coords)

coord_box = box(coord_list[0],coord_list[1], coord_list[2], coord_list[3])

bbox_crop = gpd.GeoDataFrame(
    crs = 'EPSG:3338',
```

```
    geometry = [coord_box])
```

Read in raster again cropped to bounding box.

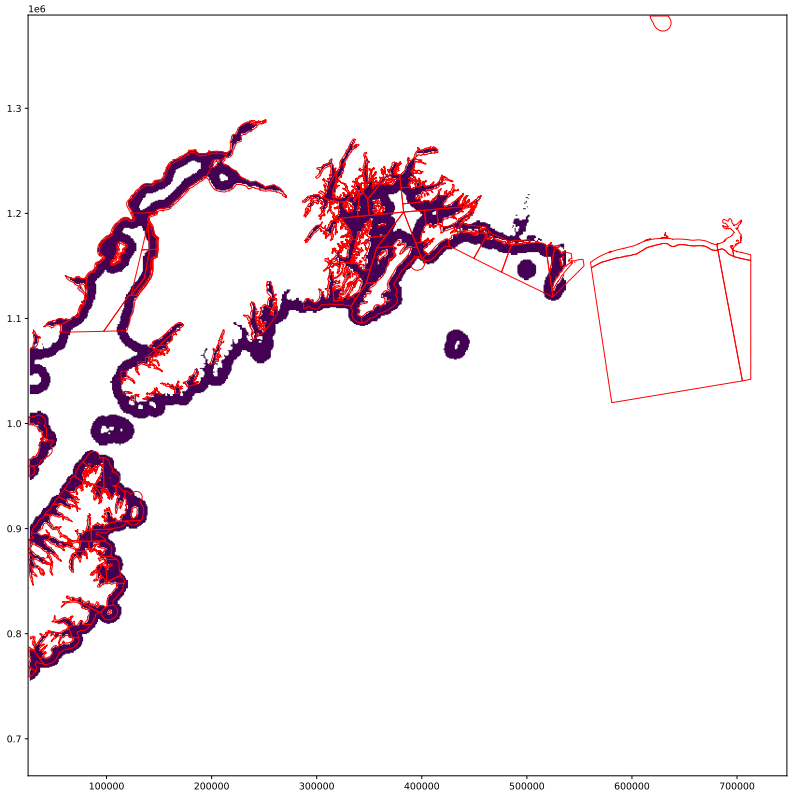
```
with rasterio.open("Coastal_2020_12.tif") as src:
    out_image, out_transform = rasterio.mask.mask(src, bbox_crop["geometry"], crop=True)
    out_meta = src.meta

out_meta.update({"driver": "GTiff",
                 "height": out_image.shape[1],
                 "width": out_image.shape[2],
                 "transform": out_transform,
                 "compress": "lzw"})

with rasterio.open("Coastal_2020_12_masked.tif", "w", **out_meta) as dest:
    dest.write(out_image)
```

We can also clip the shapefile data to the same bounding box

```
comm_clip = comm_3338.clip(bbox_crop['geometry'])
```

Quick plot to ensure they are in the same extent, and look as expected.

```
r = rasterio.open('Coastal_2020_12_masked.tif')
fig, ax = plt.subplots(figsize=(15, 15))
rasterio.plot.show(r, ax=ax)
comm_clip.plot(ax=ax, facecolor='none', edgecolor='red')
src.close()
```



6.4 Calculate total distance per fishing area

Rasterize each polygon in the shapefile that falls within the bounds of the raster data we are calculating statistics for.

We return a dictionary of indexed arrays, where each item corresponds to one polygon (fishing area). The array contains the indices of the original raster that fall within that fishing area.

```
with rasterio.open('Coastal_2020_12_masked.tif') as src:
    shape = src.shape
    transform = src.transform
    # read in the cropped raster
    r_array = src.read(1)
    # turn no data values into actual NaNs
    r_array[r_array == src.nodata] = np.nan
```

```
comm_3338['id'] = range(0,len(comm_3338))

crosswalk_dict = {}
for geom, idx in zip(comm_3338.geometry, comm_3338['id']):
    rasterized = features.rasterize(geom,
                                    out_shape=shape,
                                    transform=transform,
                                    all_touched=True,
                                    fill=0,
                                    dtype='uint8')
    # only save polygons that have a non-zero value
    if any(np.unique(rasterized)) == 1:
        crosswalk_dict[idx] = np.where(rasterized == 1)
```

```
/usr/local/lib/python3.9/site-packages/rasterio/features.py:284: ShapelyDeprecationWarning: It
  for index, item in enumerate(shapes):
```

Now we use the dictionary to calculate the sum of all of the pixels in the original raster that fall within each fishing area.

```
mean_dict = {}
# for each item in the dictionary
for key, value in crosswalk_dict.items():
    # save the sum of the indices of the raster to a new dictionary
    mean_dict[key] = np.nansum(r_array[value])
# create a data frame from the result
df = pd.DataFrame.from_dict(mean_dict,
    orient='index',
    columns=['distance'])
# extract the index of the data frame as a column to use in a join
df['id'] = df.index
```

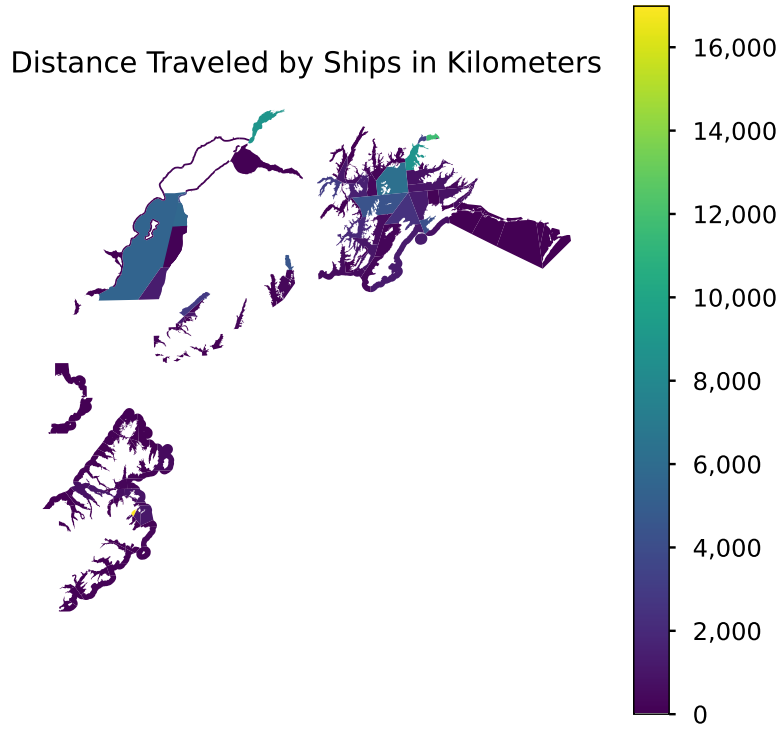
Now we join the result to the original geodataframe.

```
# join the sums to the original data frame
res_full = comm_3338.merge(df, on = "id", how = 'inner')
```

todo: Group by/summarize across another variable

```
fig, ax = plt.subplots(figsize=(7, 7))
plt.style.use("seaborn-talk")
ax = res_full.plot(column = "distance", legend = True, ax = ax)
fig = ax.figure
cb_ax = fig.axes[1]
cb_ax.set_yticklabels(["0", "2,000", "4,000", "6,000", "8,000", "10,000", "12,000", "14,000"
ax.set_axis_off()
ax.set_title("Distance Traveled by Ships in Kilometers")
plt.show()
```

```
/var/folders/24/8k48jl6d249_n_qfxwsl6xvm0000gn/T/ipykernel_15708/3303410610.py:6: UserWarning:
  cb_ax.set_yticklabels(["0", "2,000", "4,000", "6,000", "8,000", "10,000", "12,000", "14,000"
```



References