David Leifer

2010 - 2013
High School and UW-Eau Claire

I played sports and worked at a grocery store.

2014 - 2015
UW- Eau Claire
Junior and Senior Year

Worked as a traveling salesperson, in a restaurant, as a sports official, and a security guard.

I interned for a company digitizing hiking trails for course credit.

2016 January - August
UW-Eau Claire Final Semester

All of our coursework was made available online. I spoke frequently with people in class and made three posters for presentation. They were on web development, open source python, and proprietary python. I had to work tremendously hard to pass my final statistics course, leading me to study more stats after graduation.

2016 September - December
Jupyter Notebook

This was my first few attempts at using Jupyter Notebooks to learn more about statistics. I had found a graduate school that I was entirely unqualified for but hoped to gain entrance to anyway. A Professor was already unselfishly posting his class' lectures on GitHub and I attempted to stumble along.

```
import IPython
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns  # Makes for nicer plots
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 8, 4
%matplotlib inline

import pysal as ps
import pysal.spatial_dynamics.interaction as interaction
```

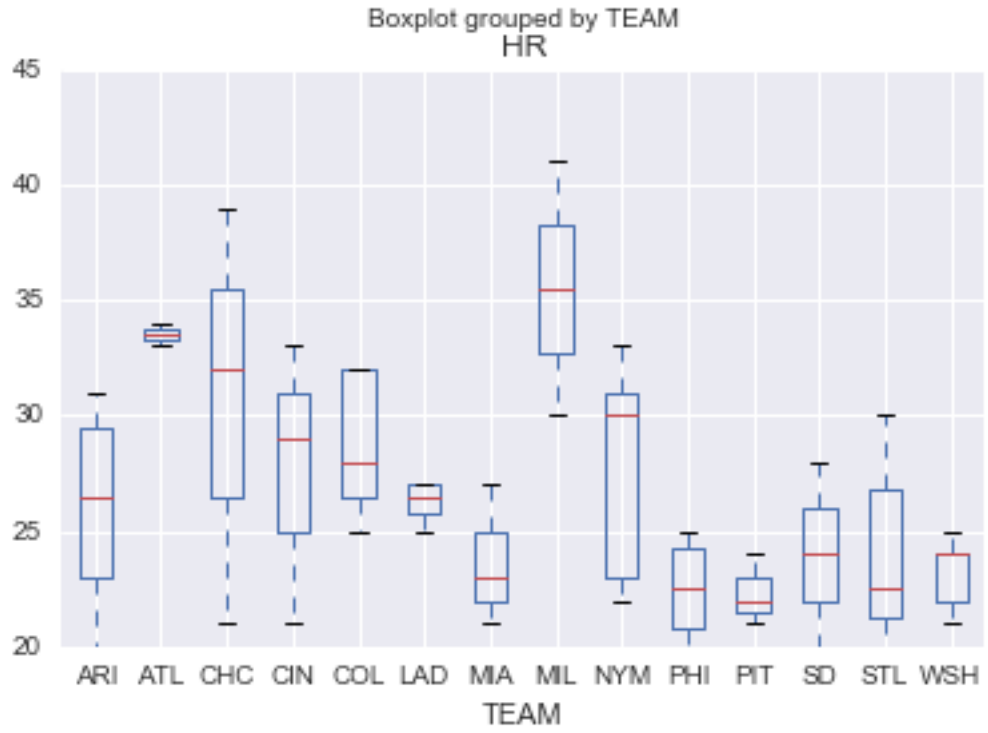
```
from matplotlib.patches import Ellipse, Circle
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import mpl_toolkits.mplot3d.art3d as art3d
from mpl_toolkits.mplot3d import proj3d
from ipywidgets import interact, interactive, fixed

from shapely.geometry import Point
import geopandas
```

National League Regular Season Home Runs

Here's a list of what I want to do:

- Tidy some Home Run data
- Visualize this Home Run data
- It might be interesting to visualize how many NL leaders are from each team as well as distance and velocity
- It might be even more interesting to figure out the x,y,z of the home run path through calculating the angle off the bat/distance. The z value is from the apex or height of the ball.

```
data = pd.read_html(data,header=1)
data[0].head()

data[0].drop(['RK','PROJ'], axis=1, inplace=True)
data[0].head()

df = data[0].iloc[::2, :]
df.head()

df.boxplot(column = ‘HOME_RUN’, by = 'TEAM')
```



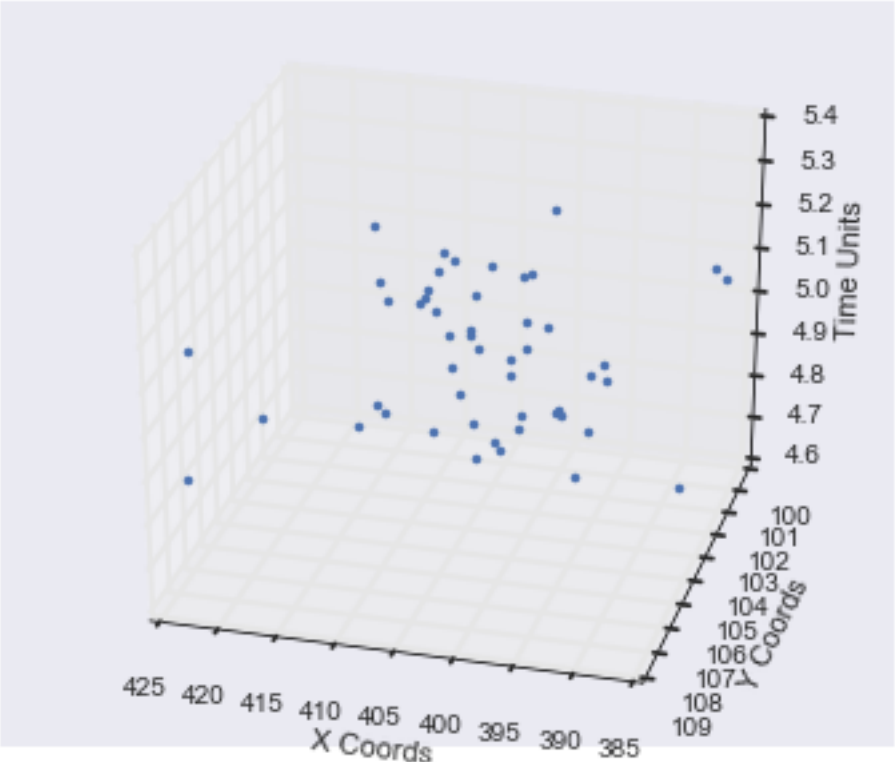
The box plot gives a nice visual of the average number of home runs hit by each team based on the top sluggers in the NL. A few teams have more observations. This doesn't really prove anything but looks cool.

Let's try to compute a Knox test to find out the space-time relationship between speed, distance, and duration spent in air. The speed and distance will substitute for the spatial component and duration in the air will act as the time component.

To test this using pysal, we need to convert the dataframe to a shapefile, which is easy to do with shapely and geopandas.

```
df['geometry'] = df.apply(lambda x: Point((float(x.DIST),
float(x.SPD))), axis=1)
df = geopandas.GeoDataFrame(df, geometry='geometry')
df.to_file('hr.shp', driver='ESRI Shapefile')

path = "hr.shp"
burkitt = ps.pdio.read_files(path)
events = interaction.SpaceTimeEvents(path,'DUR')

@interact(angle=(0, 240, 5))
def make_plot(angle):
    fig = plt.figure(figsize=(6, 5))
    ax = fig.gca(projection='3d')
    ax.view_init(30, angle)  # Use 'angle' parameter
```

```
    ax.plot(events.space[:,0], events.space[:,1], events.t[:,0])
    plt.show()
```




```
result = interaction.knox(events.space, events.t, delta=1,
tau=.25, permutations=99)
result
```

```
{'pvalue': array([ 0.11]), 'stat': 31}
```

Let's run some more stats.

Modified Knox Test

Just because it was close in distance doesn't mean it was close in time. The Modified Knox Test kind of takes care of that.

$$T = \frac{1}{2}(\Sigma_{i=1}^{n}\Sigma_{j=1}^{n} f_{ij}g_{ij} - \frac{1}{n-1}\Sigma_{k=1}^{n}\Sigma_{l=1}^{n}\Sigma_{j=1}^{n} f_{kj}g_{lj})$$

Where
n = number of events,

f = adjacency in space,

g = adjacency in time (calculated in a manner equivalent to *as* and *at* in the Knox test).

The first part of this statistic is equivalent to the original Knox test, while the second part is the expected value under spatial and temporal randomness.

```
result = interaction.modified_knox(events.space, events.t,
delta=1, tau=.25, permutations=99)
result
```

```
{'pvalue': 0.35999999999999999, 'stat': 1.6428571428571459}
```

Mantel Test

The Mantel test keeps distance information discarded by the Knox tests. Unstandardized Mantel stat is calculated by summing the product of spatial and temporal distances between pairs.

Unstandardized Mantel Test

$$Z = \Sigma_i^n \Sigma_j^n (d_{ij}^s + c)^p (d_{ij}^t + c)^p$$

Where:
ds and *dt* denote distance in space and time.
The constant, c, and the power, p, usually default to 0 and 1.

Standardized Mantel Test

$$r = \frac{1}{n^2 - n - 1} \Sigma_i^n \Sigma_j^n \left[\frac{d_{ij}^s - \bar{d}^s}{\sigma_{d^s}} \right] \left[\frac{d_{ij}^t - \bar{d}^t}{\sigma_{d^t}} \right]$$

Where *ds (hat)* refers to the average distance in space, and *dt (hat)* the average distance in time. For notational convenience σds and σdt refer to the sample (not population) standard deviations, for distance in space and time, respectively.

```
{'pvalue': 0.70999999999999996, 'stat': -0.036547539535649536}
```

This type of stat is a Pearson correlation coefficient. Extreme values could distort the relationship and large samples are required to produce an accurate estimate. Correlations tend to be small since it compares all pairs of observations.

Jacquez Test

This is a fix if we don't know what distances to use and instead substitute a nearest neighbor distance approach:

$$a_{ijk}^{t} = \begin{cases} 1, & \text{if } event\,j\,is\,k\,nearest\,neighbor\,of\,event\,i\,in\,space \\ 0, & \text{otherwise} \end{cases}$$

Where n = number of cases; as = adjacency in space; at = adjacency in time.

Finding Actual x,y,z Data

This isn't real x,y,z data of the ball's flight path like I initially set out to find. It's more of a spatial cluster examination between speed, distance, and duration of the ball's flight when these attributes are plotted.

A good project would be to sift through the data to get the batted ball distance, exit velocity, distance, etc. and combine this information with weather.

An article did by using launch angle, batted ball speed, overall distance of the hit, and the location in x,y to feed a modified random forest algorithm, which is similar to a collection of decision trees. The Hardball website also looked at the impact of atmospheric conditions on flight trajectory. Another article looks at the spatial component of ballparks in how weather and pressure influence the ball's flight path.

The Continuous Forest Fire Model

Implementing the ContinuousSpace class

This is an implementation of the continuous surface argument to model forest fires over an arbitrary position. As can be found in the documentation of the space.py file in the directory, this class draws each agent as a point storing the position as a 2D tuple and is based internally on the prior MultiGrid class. The MultiGrid class itself is a Grid that can hold multiple objects. The addition of the ContinuousSpace class speeds up neighborhood lookups, however the addition of more cells slows down movement. This is similar to a Digital Elevation Model (DEM).

Hindsight: I also included a “height” parameter with the intention of showing how fire burns from lower elevation to higher elevation as hot air rises but it was beyond my ability in 2016.

Calling it a DEM model is confusing because I was inaccurately swapping variables around. Even more egregious is the redundancy in naming convention.

Creating a data structure

The original model.py file was deleted from the newly created dem-forest-fire dir and a new file called demmodel.py was created. The only import that was changed was the space.py import, taking all of the classes included in that file instead of just the Grid class as was done in the original forest_fire example:

```
import random

from library import Model, Agent
from library.datacollection import DataCollector

from library.space import Grid, MultiGrid, ContinuousSpace

from library.time import RandomActivation
from library.batchrunner import BatchRunner

from agent import TreeCell
#from fake_surface import fake_surface
```

Initialize fire and include trees

The first section of the self arguments initializes the model parameters as maximum x and y coordinates for space, while torus is set as a boolean for whether the edges loop around.

Hindsight: This is actually incorrect, below I set torus to a range of densities to see results.

The minimum x and y coordinates are set to default 0 and the grid width and height are set to default 100.

Below these are more model parameters that create the cell width and height based on the subtraction of x,y max and min values divided by the grid height and grid width respectively. The schedule is created to add the time component, the grid uses MultiGrid and the previously created grid_width and grid_height parameters. The DataCollector provides a simple way to store information created by the model in three types: model, agent, and table data. Some dictionary things happen and you can eventually put these in pandas DataFrames.

The last code part creates a tree in each cell and sets it on fire:

```
def __init__(self, x_max, y_max, density, x_min=0, y_min=0,
             grid_width=100, grid_height=100):
    self.x_min = x_min
```

```
        self.x_max = x_max
        self.density = density
        self.width = x_max - x_min
        self.y_min = y_min
        self.y_max = y_max
        self.height = y_max - y_min
        self.torus = torus

        self.cell_width = (self.x_max - self.x_min) / grid_width
        self.cell_height = (self.y_max - self.y_min) / grid_height

        self.schedule = RandomActivation(self)
        self.grid = MultiGrid(grid_width, grid_height, density)

        self.datacollector = DataCollector({"Fine": lambda m:
        self.count_type(m, "Fine"),"On Fire": lambda m:
        self.count_type(m, "On Fire"),"Burned Out": lambda m:
        self.count_type(m, "Burned Out")})

        # Place a tree in each cell with Prob = density
        for (contents, x, y) in self.grid.coord_iter():
            if random.random() < 100:
                # Create a tree
                new_tree = TreeCell((x, y), self)
                # Set all trees in the first column on fire.
                if x == 0:
                    new_tree.condition = "On Fire"
                self.grid._place_agent((x, y), new_tree)
                self.schedule.add(new_tree)
        self.running = True
```

The next def is a straightforward stepping argument:

```
def step(self):
    """
    Advance the model by one step.
    """
    self.schedule.step()
    self.datacollector.collect(self)

    # Halt if no more fire
    if self.count_type(self, "On Fire") == 0:
        self.running = False
```

The last part of demmodel.py is a static method to keep track of the burn status of each type of tree in the forest:

```
@staticmethod
def count_type(model, tree_condition):
    """
    Helper method to count tree condition.
    """
    count = 0
    for tree in model.schedule.agents:
        if tree.condition == tree_condition:
            count += 1
    return count
```

To put it all together:

```
import random

from library import Model
from library.datacollection import DataCollector
from library.space import Grid, MultiGrid, ContinuousSpace
from library.time import RandomActivation
from library.batchrunner import BatchRunner

from agent import TreeCell

class demmodel(Model):

    _grid = None
    def __init__(self, x_max, y_max, torus, x_min=0, y_min=0,
                 grid_width=100, grid_height=100):

        self.x_min = x_min
        self.x_max = x_max
        self.torus = torus
        self.width = x_max - x_min
        self.y_min = y_min
        self.y_max = y_max
        self.height = y_max - y_min

        self.cell_width = (self.x_max - self.x_min) / grid_width
        self.cell_height = (self.y_max - self.y_min) /
        grid_height

        self.schedule = RandomActivation(self)
        self.grid = MultiGrid(grid_width, grid_height, torus)

        self.datacollector = DataCollector({"Fine": lambda m:
```

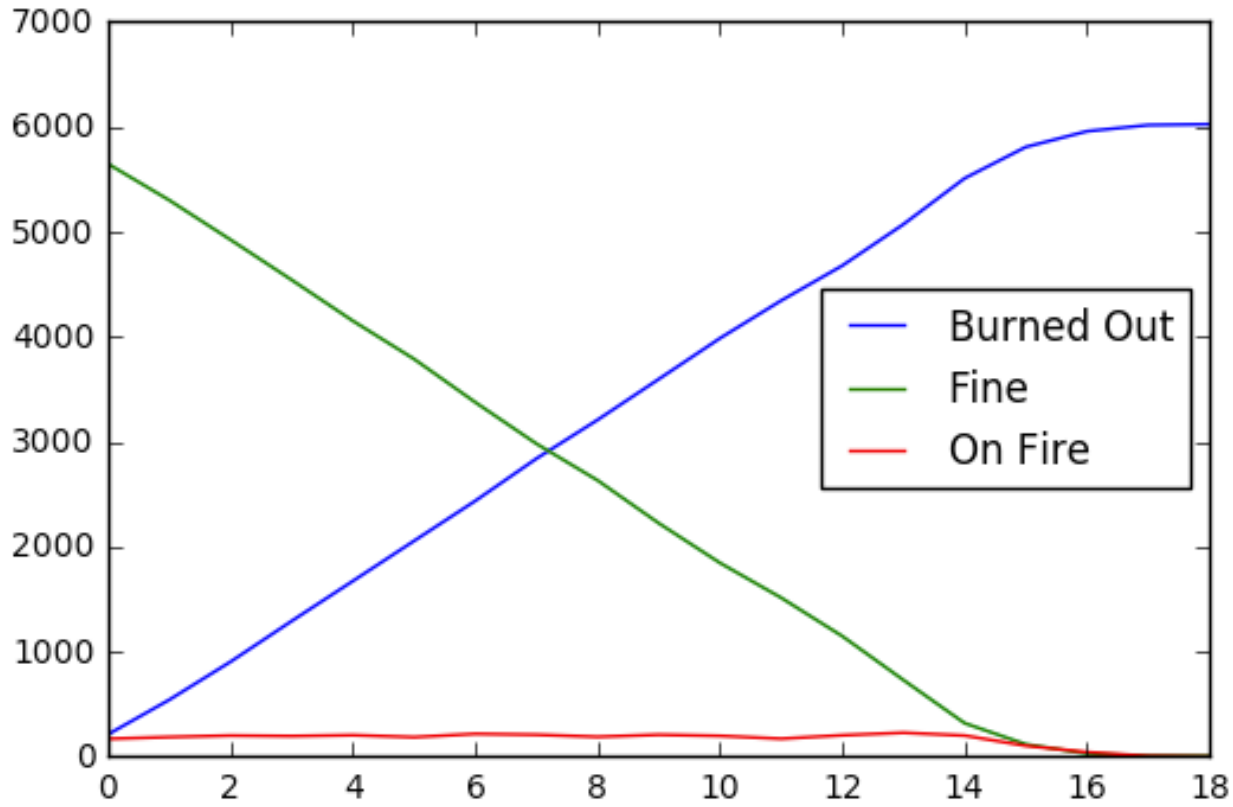
```
            self.count_type(m, "Fine"),"On Fire": lambda m:
            self.count_type(m, "On Fire"),"Burned Out": lambda m:
            self.count_type(m, "Burned Out")})

        # Place a tree in each cell with Prob = density
        for (contents, x, y) in self.grid.coord_iter():
            if random.random() < torus:
                # Create a tree
                new_tree = TreeCell((x, y), self)
                # Set all trees in the first column on fire.
                if x == 0:
                    new_tree.condition = "On Fire"
                self.grid._place_agent((x, y), new_tree)
                self.schedule.add(new_tree)
        self.running = True

    def step(self):
        """
        Advance the model by one step.
        """
        self.schedule.step()
        self.datacollector.collect(self)

        # Halt if no more fire
        if self.count_type(self, "On Fire") == 0:
            self.running = False

    @staticmethod
    def count_type(model, tree_condition):
        """
        Helper method to count trees.
        """
        count = 0
        for tree in model.schedule.agents:
            if tree.condition == tree_condition:
                count += 1
        return count
```

Analyzing the model

In a manner similar to the example, the model will be initialized and results will be drawn. We need to fill out the x_max, y_max, torus, x_min, y_min, grid_width, and grid_height and then run demmodel. A visual is created with matplot lib.

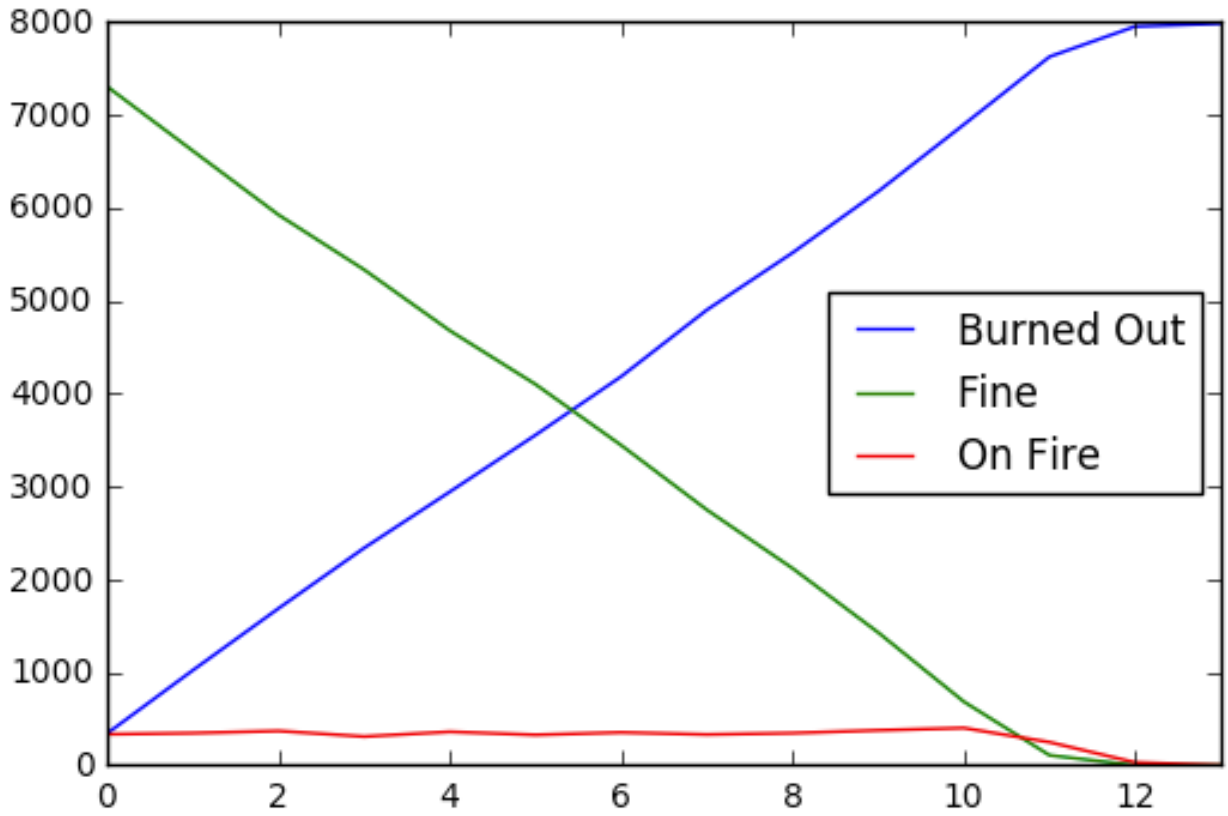
```
import numpy as np
```

```
from numpy.random import uniform
from numpy import multiply
from scipy.ndimage.filters import gaussian_filter
import matplotlib.pyplot as plt
%matplotlib inline
fire = demmodel(100, 100, .6, 0, 0, 100, 100)
fire.run_model()

results = fire.datacollector.get_model_vars_dataframe()
results.plot()
```



The fire burned out after 17 steps, incinerating all trees prior to that point.
Let's change the density to see an alternative situation.

```
fire = demmodel(100, 100, .8, 0, 0, 100, 100)
fire.run_model()

results = fire.datacollector.get_model_vars_dataframe()
results.plot()
```

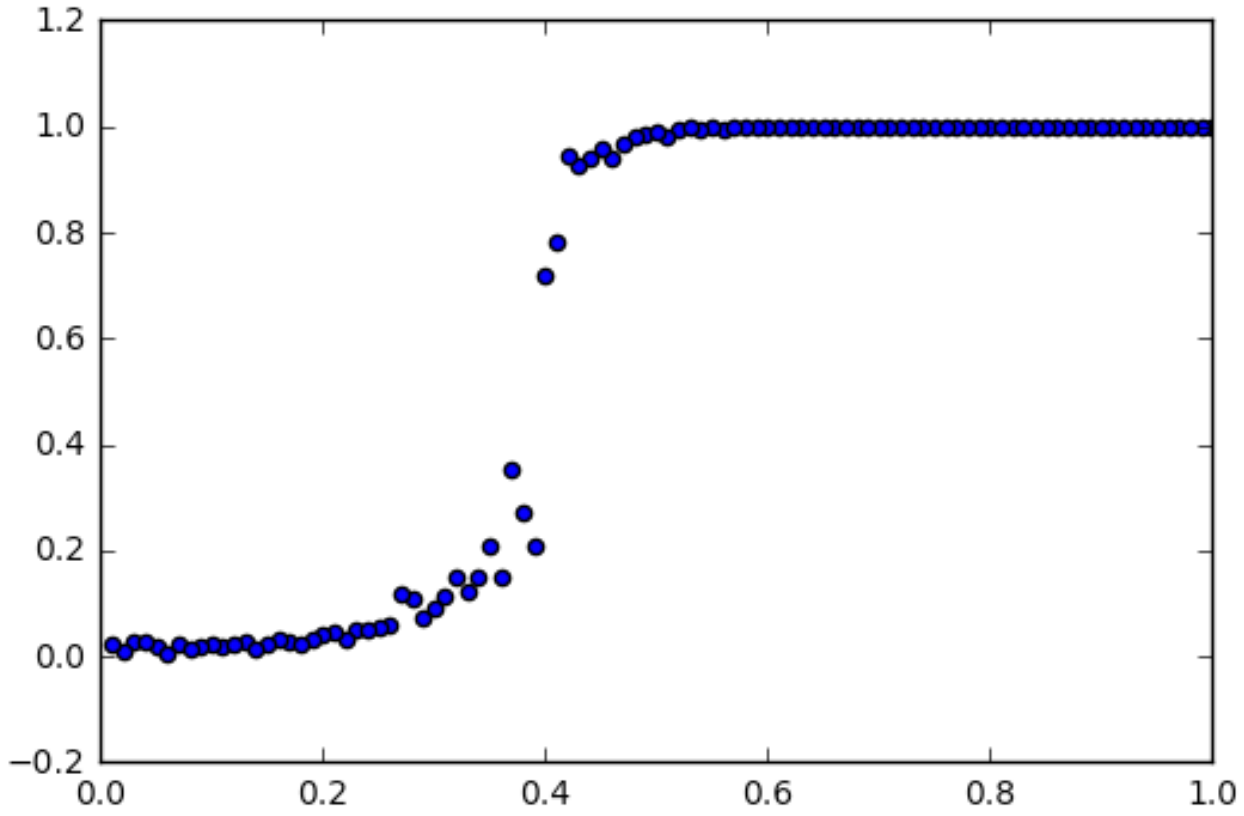


Batchruns

This python library provides a built-in object to fulfill all of the batch data testing and exploratory needs.

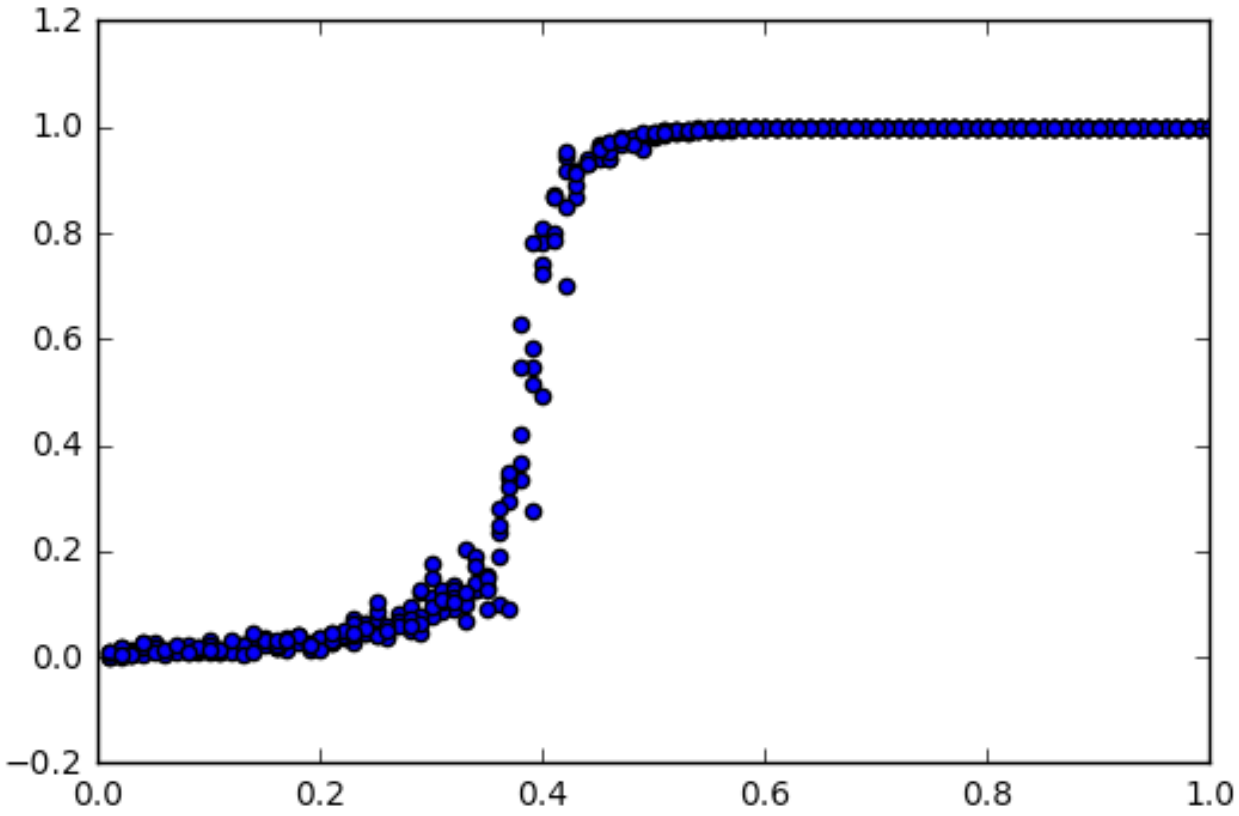
```
param_set = dict(x_max=1, # Height and width are constant
                 y_max=1,
                 torus=np.linspace(0,1,101)[1:],
                 x_min=0,
                 y_min=0,
                 grid_width=100,
                 grid_height=100)

# At the end of each model run, calculate the fraction of trees
which are Burned Out
model_reporter = {"BurnedOut": lambda m: (demmodel.count_type(m,
     "Burned Out") / (m.schedule.get_agent_count()))}

# Create the batch runner
param_run = BatchRunner(demmodel, param_set,
model_reporters=model_reporter)
```

```
param_run.run_all()

100%|██████████| 100/100 [00:44<00:00,  1.23it/s]

df = param_run.get_model_vars_dataframe()
df.head()

plt.scatter(df.torus, df.BurnedOut)
plt.xlim(0,1)
```



The critical value occurs around .3 where the trees begin to be burned, which is sooner than the Forest Fire Model included in the examples. Since this batch run only runs once per value, it will need to increase iterations for each value to 5 to provide additional information about the model.

```
param_run = BatchRunner(demmodel, param_set, iterations=5,
model_reporters=model_reporter)
param_run.run_all()
df = param_run.get_model_vars_dataframe()
plt.scatter(df.torus, df.BurnedOut)

plt.xlim(0,1)
```



Closing remarks

The critical factor of using the ContinuousSpace class in the demmodel.py occurred sooner than using the standalone Grid class in the original forest_fire example by about .2 steps in time. This conserves time in this continuous model and still serves to effectively burn the forest.

2017 - 2019
SIU Carbondale Graduate School

I moved to Madison for a few months and worked digitizing 3D LiDAR roads before applying for graduate school in Illinois.

Although I went to a different school, my thesis built off the work of the previously mentioned professors. It examined sentiment analysis on social media data and was published in a peer-reviewed journal.

2019 - 2020
Jeffersonville, Indiana

I was under a year contract for the US Census Bureau sorting mail, calling local municipalities, and digitizing/processing land parcels.

2020 - 2021
St. Louis, Missouri

I worked as a federal contractor geo-referencing charts primarily for the USGS and FAA. This job did not require a security clearance.

I wrote a series of web development and statistics tutorials because I was bored and curious. This was done on my own and I didn't get paid for it.

Setting up Geoserver on Google Cloud Platform for Timeseries Raster Display

Hindsight: The Tomcat server stopped being updated with security patches around 2015 according to Geoserver's website accessed in early 2023. I didn't use Geoserver in my website but I did use it in a container in GeoNode on a different project for about a month in 2022. Hopefully the container was configured properly (security is a nightmare).

Today we will walk through the steps necessary to set up our own Linux Ubuntu server on Google Cloud Platform (GCP). After that we will install Java (OpenJDK version 1.8.0_252). Then Tomcat8 for our servlet container. Within Tomcat8 we will install our Geoserver.war file. We will configure our Geoserver installation to allow the upload of a timeseries of precipitation for the state of Oregon using the Image Mosaic plugin. Finally, this WMS will be added to an HTML/CSS/JS file and hosted in Geoserver's www folder.

Step 1: Sign up for 300 Credits with GCP

Select "GO TO COMPUTE ENGINE".

On "VM Instances" click "Create". Select your desired Linux distribution.
Now we have an instance! Click the SSH button on the right side of the page to bring up the terminal.

Step 2: Installing OpenJDK

Run the following command and type yes to install Java:

```
sudo apt update
sudo apt install openjdk-jdk
```

Now check to see if it was installed with this command:

```
java -version
```

Step 3: Tomcat8 Installation

To install tomcat8:

sudo apt-get install tomcat8
sudo apt-get install tomcat8-docs tomcat8-examples tomcat8-admin

If you want to start, stop, or restart the servlet container, use:

sudo systemctl start tomcat8
sudo systemctl stop tomcat8
sudo systemctl restart tomcat8

To use the GUI, open up “tomcat-users.xml” with this command:

sudo nano /var/lib/tomcat8/conf/tomcat-users.xml

Add the following lines replacing “password” with your password and “username” with your username:

<role rolename="manager-gui"/>
<role rolename="admin-gui"/>
<user username="username" password="password" roles="manager-gui,admin-gui"**/>**

Save the file and restart tomcat8:

sudo systemctl restart tomcat8

Now we can point our browser to the external ip of our GCP Linux VM at port 8080:
http://XX.xxx.XXX.xxx:8080/

Part 4: Installing Geoserver

To install the Geoserver.war package, navigate to the “webapps” folder within your “tomcat8” installation with this command:

cd /var/lib/tomcat8/webapps

Next, download the latest Geoserver.war, which is a zipped file:

sudo wget *the_file*

Now we need to install the “unzip” utility:

sudo apt-get install unzip

Then to unzip the package:

sudo unzip download

Then make the .war executable by entering:

sudo chmod +x /var/lib/tomcat8/webapps/geoserver.war
(laugh in hindsight at this chmod)

Finally, restart “tomcat8”:

sudo service tomcat8 restart

Point your browser at the following address:
http://XX.xxx.XXX.xxx:8080/geoserver/web/

The default behavior of Geoserver is to set user to “admin” and password to “geoserver” but you can change it.

Step 5: Configuring tomcat8 for the ImageMosaic Plugin

Now we need to configure the servlet container (tomcat8) to allow timeseries data to be uploaded onto Geoserver. To do this, navigate to the following directory:

cd /usr/share/tomcat8/bin/

We need to make a script file:

sudo nano setenv.sh

Add the following Java methods to this file:

export JAVA_OPTS="$JAVA_OPTS -Duser.timezone=GMT"
export JAVA_OPTS="$JAVA_OPTS -Dorg.geotools.shapefile.datetime=true"

Exit the file and make the script executable, then run the script:

sudo chmod +x setenv.sh
./setenv.sh
(laugh in hindsight again)

Now restart tomcat8:

sudo service tomcat8 restart

Check to see if the changes took place by entering the following:

ps aux | grep java

Step 6: Configuring Geoserver for Timeseries

Now we need to get data onto the server and change permissions on that folder to upload onto Geoserver. Navigate to the "data" folder and make a directory to host our files:

cd /var/lib/tomcat8/webapps/geoserver/data/data
sudo mkdir timeseries
cd timeseries

SSH into the GCP and transfer over some data.

The two .properties files look like this:

timeregex.properties

regex=[0-9]{8}

indexer.properties

TimeAttribute=ingestion
ElevationAttribute=elevation
Schema=*the_geom:Polygon,location:String,ingestion:java.util.Date,elevation:Integer
PropertyCollectors=TimestampFileNameExtractorSPI[timeregex](ingestion)

Change permissions on the data folder:

sudo chmod -R ugo+rw /var/lib/tomcat8/webapps/geoserver/data/data/timeseries
(This would given everyone permissions to that folder. Good thing I didn't use this in production.)

Step 7: Upload Timeseries Package to Geoserver

Under "Data" on the left hand side of the screen go to "Stores". Select "Add new stores".

Select "ImageMosaic" and "Browse". Navigate to the folder you put the data in (data/timeseries) and select "OK".

Fill out "Data Source Name" and "Description" then save.

Click “Publish” on the next screen. Click “Dimensions” and enable “Time”. “Presentation” should be set to “List”. Then select “Save”.

If it worked, you can go to “Layer Preview” and click “timeseries”.

Step 8: Modifying HTML/CSS/JS to make a Time Slider

Navigate to the www folder with this command:

cd /var/lib/tomcat8/webapps/geoserver/data/www

Create a file name oregon.html

sudo nano oregon.html

Copy this code in there! You need to modify the ip address with your website address.

```
<!DOCTYPE html>
<html>
  <head>
    <title>WMS Time</title>
    <link rel="stylesheet" href="https://openlayers.org/en/v4.6.5/css/ol.css" type="text/css">
    <script src="https://openlayers.org/en/v4.6.5/build/ol.js"></script>
    <style>
    /* Setting up the slider styling */
        .slidecontainer {
            width: 120px;
            margin-top: -30px;
            margin-left: 10px;
            z-index: 100px;
        }
    .slider {
            -webkit-appearance: none;
            width: 100%;
            height: 15px;
            border-radius: 5px;
            background: #d3d3d3;
            outline: none;
            opacity: 0.7;
            -webkit-transition: .2s;
            transition: opacity .2s;
        }
    .slider::-webkit-slider-thumb {
            -webkit-appearance: none;
            appearance: none;
```

```
            width: 25px;
            height: 25px;
            border-radius: 50%;
            background:   #00008B;
            cursor: pointer;
        }
    .slider::-moz-range-thumb {
            width: 25px;
            height: 25px;
            border-radius: 50%;
            background: #4CAF50;
            cursor: pointer;
        }
    </style>
  </head>
<body>
    <!-- Create map container -->
    <div id="map" class="map"></div>
    <div role="group" aria-label="Animation controls">
    <!-- Create slider container -->
     <div class="slidecontainer">
          <input type="range" min="0" max="2" value="0" class="slider" id="myRange">
          <p>Date: <span id="date_value"></span></p>
      </div>
<script>

// Set up the layers

var layers = [
        new ol.layer.Tile({
            source: new ol.source.OSM()
          }),
        new ol.layer.Tile({
          source: new ol.source.TileWMS({
            url: 'http://xx.XXX.XXX.XXX:8080/geoserver/cite/wms',
            params: {'LAYERS': 'cite:timeseries'}
          })
        })
      ];

var map = new ol.Map({
        layers: layers,
        target: 'map',
        view: new ol.View({
          // Defining the location in Lat Lon.
          center:  ol.proj.transform([-120.5542,43.8041], 'EPSG:4326', 'EPSG:3857'),
```

```
        zoom: 5
      })
    });

// Define the available dates

var dates = ['1981-01-01T00:00:00.000Z', '1991-01-01T00:00:00.000Z',
'2001-01-01T00:00:00.000Z', '2011-01-01T00:00:00.000Z']
var sliderRange = document.getElementById("myRange");
sliderRange.max = dates.length-1;

var dateValue = document.getElementById("date_value");
dateValue.innerHTML = dates[sliderRange.value].slice(0,10);
layers[1].getSource().updateParams({'TIME': dates[sliderRange.value]});

// Update the current slider value (each time you drag the slider handle)

sliderRange.oninput = function() {
    dateValue.innerHTML = dates[this.value].slice(0,10);
    layers[1].getSource().updateParams({'TIME': dates[this.value]});
}

</script>
</body>
</html>
```

Climate Change Script

The following is a description of a way to build a climate change pipeline. It installs libraries, downloads data with R, runs statistics with python, and builds applications with Node Package Manager.

Hindsight: The packages change frequently and it is probably better to enter the commands manually.

Part 1: Installation of the libraries

This first part outlines what libraries we are installing. First we install WGET and GIT. Then we install R for Debian. Next we install some python dependencies including libcurl4-openssl-dev and libssl-dev. Python3-pip is then installed. Then we need some libraries to compile GDAL version 3.X.0 from source. We install ubuntugis-unstable, followed by sqlite3 and libsqlite3-dev. We also need libtiff5-dev. A library for the Geopandas python package is installed called libspatialindex-dev. To build the web applications, we install nodejs and npm. We then use npm to install ol, parcel-bundler, and ol-ext.

```
#!/bin/bash
echo begin

# Install the correct libraries
sudo apt-get update
# sudo apt-get upgrade
sudo apt-get install wget
sudo apt install git
# Install R
sudo apt install dirmngr apt-transport-https ca-certificates software-properties-common
gnupg<VERSION>
sudo apt-key adv --keyserver keys.gnupg.net --recv-key ' '
sudo add-apt-repository 'deb https://cloud.r-project.org/bin/linux/debian buster-
cran<VERSION>/‘
sudo apt install r-base
sudo apt install build-essential
# Install python
sudo apt-get install libcurl<VERSION>-openssl-dev libssl-dev
sudo apt install python3-pip
sudo add-apt-repository -y ppa:ubuntugis/ubuntugis-unstable
sudo apt install sqlite3
sudo apt install libsqlite3-dev
sudo apt-get install -y libtiff<VERSION>-dev
# Install GDAL python
# sudo apt-get remove python-gdal
# Install geopandas
sudo apt install libspatialindex-dev
# Install node and npm
sudo apt install nodejs npm
# Install ol with npm
sudo npm install ol
sudo npm install -g parcel-bundler
sudo npm install ol-ext
```

Part 2: Making some folders, setting our path, and running the R scripts

Next we use mkdir to create some folders in the data folder. Then we create a VARIABLENAME exported variable. Then we create two variables concatenated onto VARIABLENAME to run the R scripts. The Rscript is run with the two concatenated variables.

```
# Make folders to store the data
cd data
sudo mkdir ppt
sudo mkdir tmean
```

```
sudo mkdir pptJanELNin
sudo mkdir pptJanLANin
sudo mkdir pptJanNeutral
sudo mkdir tmeanJanELNin
sudo mkdir tmeanJanLANin
sudo mkdir tmeanJanNeutral
sudo mkdir tmean_pearson_final
sudo mkdir ppt_pearson_final
sudo mkdir ppt_elnino_minus_neutral
sudo mkdir ppt_lanina_minus_neutral
sudo mkdir tmean_elnino_minus_neutral
sudo mkdir tmean_lanina_minus_neutral
cd ..

# Create the paths
DATA_PATH=$(pwd)
export VARIABLENAME=$DATA_PATH

# Create tmean R script paths
SCRIPT_PATH_PLUS_TMEAN="$VARIABLENAME/scripts/rscripts_tmean/
download_prism_data.R"
# Create ppt R script paths
SCRIPT_PATH_PLUS_PPT="$VARIABLENAME/scripts/rscripts_ppt/
download_prism_data.R"

# Run the download scripts
sudo Rscript $SCRIPT_PATH_PLUS_TMEAN
echo downloaded tmean data
sudo Rscript $SCRIPT_PATH_PLUS_PPT
echo downloaded ppt data
```

Part 3: Installing python libraries and compiling GDAL 3.X.X from source

We use pip3 to upgrade pip. Then we install rasterio, seaborn, guppy3, pyproj, fiona, geopandas, rtree, pandas, streamhist, and gdal for python.
Then we have to use wget to download proj-X so we can compile that from source. We unzip with tar, cd into the directory, configure the package without curl and make/make install. Then we exit that directory and use wget and unzip to download GDAL and unzip it. We cd into the directory and configure the project with proj specified as the folder path /usr/local.
We also configure with python3 and make/make install.

You need to export the LD_LIBRARY_PATH and enter the command ldconfig after GDAL compiles.

```
# Install python libraries with pip3
sudo pip3 install rasterio
sudo pip3 install seaborn
sudo pip3 install guppy3
sudo pip3 install --upgrade pip
sudo pip3 install pyproj
sudo pip3 install fiona
sudo pip3 install geopandas
sudo pip3 install rtree
sudo pip3 install pandas== X.X
sudo pip3 install streamhist
sudo pip3 install gdal==3.X.X

# Install proj-<version>
cd ..
wget https://download.osgeo.org/proj/proj-X.X.0.tar.gz
tar xvzf proj-X.X.X.tar.gz
cd proj-X.X.X
sudo ./configure --without-curl
sudo make && sudo make install

# Download GDAL v3.X.X from source
cd ..
sudo wget download.osgeo.org/gdal/3.X.X/gdal3XX.zip
sudo unzip gdal3XX.zip
cd gdal-3.X.X
sudo ./configure --with-proj=/usr/local --with-python3
sudo make clean && sudo make && sudo make install
# Set LD_LIBRARY_PATH so that recompiled GDAL is used
export LD_LIBRARY_PATH=/usr/local/lib
sudo ldconfig
```

Part 4: Run the pythons scripts

```
# Make path variables to python scripts
PATH_PLUS_PPT_LaElNeu="$VARIABLENAME/scripts/python/
ppt_bil2tif_LaElNeu_analysis.py"
PATH_PLUS_TMEAN_LaElNeu="$VARIABLENAME/scripts/python/
tmean_bil2tif_LaElNeu_analysis.py"
PATH_PLUS_PPT_ANOVA="$VARIABLENAME/scripts/python/ppt_ANOVA_analysis.py"
PATH_PLUS_TMEAN_ANOVA="$VARIABLENAME/scripts/python/
tmean_ANOVA_analysis.py"
PPT_COR="$VARIABLENAME/scripts/python/ppt_cor.py"
TMEAN_COR="$VARIABLENAME/scripts/python/tmean_cor.py"
TMEAN_COR_ACTUALLY="$VARIABLENAME/scripts/python/tmean_cor_actually.py"
```

```
GEOSENT_2ROUNDED="$VARIABLENAME/scripts/python_weather/geosent_2rounded.py"
GEO_STREAMHIST="$VARIABLENAME/scripts/python_weather/Geo_streamhist.py"
PPT_COR_ACTUALLY="$VARIABLENAME/scripts/python/ppt_cor_actually.py"
PPT_QUANTILE_RECLASSIFY="$VARIABLENAME/scripts/python/quantile_reclassify.py"
TMEAN_COR_ACTUALLY="$VARIABLENAME/scripts/python/tmean_cor_actually.py"
TMEAN_QUANTILE_RECLASSIFY="$VARIABLENAME/scripts/python/
tmean_quantile_reclassify.py"
TMEAN_LAELNEU_DIFF="$VARIABLENAME/scripts/python/tmean_LaElNeu_diff.py"
PPT_LAELNEU_DIFF="$VARIABLENAME/scripts/python/ppt_LaElNeu_diff.py"
PPT_OG_QUANTILE_RECLASSIFY="$VARIABLENAME/scripts/python/
ppt_og_quantile_reclassify.py"
TMEAN_OG_QUANTILE_RECLASSIFY="$VARIABLENAME/scripts/python/
tmean_og_quantile_reclassify.py"

# Run the python scripts
sudo python3 $PATH_PLUS_PPT_LaElNeu
sudo python3 $PATH_PLUS_TMEAN_LaElNeu
sudo python3 $PATH_PLUS_PPT_ANOVA
sudo python3 $PATH_PLUS_TMEAN_ANOVA
sudo python3 $PPT_COR
sudo python3 $TMEAN_COR
sudo python3 $GEOSENT_2ROUNDED
sudo python3 $GEO_STREAMHIST
sudo python3 $PPT_COR_ACTUALLY
sudo python3 $PPT_QUANTILE_RECLASSIFY
sudo python3 $TMEAN_COR_ACTUALLY
sudo python3 $TMEAN_QUANTILE_RECLASSIFY
sudo python3 $TMEAN_LAELNEU_DIFF
sudo python3 $PPT_LAELNEU_DIFF
sudo python3 $PPT_OG_QUANTILE_RECLASSIFY
sudo python3 $TMEAN_OG_QUANTILE_RECLASSIFY
```

Part 5: Create an apache web server, make some directories, and copy over some files

First we move some data around. Then install a web server using the instructions provided by Google Cloud Platform.

We then use mkdir to create a bunch of folders on the web server to hold our output XYZ files from our GDAL commands.

We also copy over the ANOVA txt output for consumption in the web application along with the output Histogram image and Breaks image from the Twitter scripts.

```
# Move the geojson to /data/
```

```
sudo mv $VARIABLENAME/*geojson $VARIABLENAME/data

# Open the gates!
sudo apt-get install apache2 -y
sudo a2ensite default-ssl
sudo a2enmod ssl
sudo vm_hostname="$(curl -H "Metadata-Flavor:Google" \
http://ip.address/computeMetadata/v1/instance/name)"
sudo echo "Page served from: $vm_hostname" | \
sudo tee /var/www/html/index.html

# Make some dirs to hold the xyz tiles
sudo mkdir /var/www/html/ppt_bil2tif_LaElNeu_analysis_xyz/
sudo mkdir /var/www/html/tmean_bil2tif_LaElNeu_analysis_xyz/
sudo mkdir /var/www/html/tmean_cor_xyz/
sudo mkdir /var/www/html/ppt_cor_xyz/
sudo mkdir /var/www/html/ppt_bil2tif_LaElNeu_analysis_xyz/xyz_pptJanELNin
sudo mkdir /var/www/html/ppt_bil2tif_LaElNeu_analysis_xyz/xyz_pptJanLANin
sudo mkdir /var/www/html/ppt_bil2tif_LaElNeu_analysis_xyz/xyz_pptJanNeutral
sudo mkdir /var/www/html/tmean_bil2tif_LaElNeu_analysis_xyz/xyz_tmeanJanELNin
sudo mkdir /var/www/html/tmean_bil2tif_LaElNeu_analysis_xyz/xyz_tmeanJanLANin
sudo mkdir /var/www/html/tmean_bil2tif_LaElNeu_analysis_xyz/xyz_tmeanJanNeutral
sudo mkdir /var/www/html/tmean_elnino_minus_neutral_xyz
sudo mkdir /var/www/html/tmean_lanina_minus_neutral_xyz
sudo mkdir /var/www/html/ppt_elnino_minus_neutral_xyz
sudo mkdir /var/www/html/ppt_lanina_minus_neutral_xyz
sudo mkdir /var/www/html/tmean_bil2tif_resize_xyz
sudo mkdir /var/www/html/ppt_bil2tif_resize_xyz

# Copy over the ANOVA files onto web server
sudo cp ~/polarbearGIS/data/tmean_ANOVA_output.txt /var/www/html
sudo cp ~/polarbearGIS/data/ppt_ANOVA_output.txt /var/www/html

# Copy over the sentiment histogram html files output from streamhist
sudo cp ~/polarbearGIS/data/Histogram.png /var/www/html
sudo cp ~/polarbearGIS/data/Breaks.png /var/www/html
```

Part 6: Generating XYZ tiles with GDAL command line

The second to last step is generating XYZ tiles for the outputs of our python scripts so our web applications can use them. There are a few variations of this that are used, however only the complicated one is described here. The more basic versions use similar commands.

For the correlation tif files, we first need to create a variable named ppt_pearson_final that concatenates VARIABLENAME onto the path /data/ppt_pearson_final/. We then do the same for the location of the PPT_PEARSON_FINAL_COLOR, which is a txt file that looks like this:

```
0       0 0 0 0
1       200 221 240
2       115 179 216
3       40 121 185
4       8 48 107
```

The first column is the breaks while the other numbers are the color in RGBA. Then we create a COLOR variable to hold the color ending. We also make a variable named PPT_COR_XYZ_BASE, which will be used in the for loop. Next we make an increment variable starting at 1981. The for lop creates an index variable named ppt_index from the ppt_pearson_final directory by using a wild card to grab all the tifs named _reclassed.tif. We then name a variable ppt_index_color_output and concatenate the ppt_index onto the COLOR variable to make an output color file location. This is then used in gdaldem color-relief GDAL command. The first argument accepts our ppt_index, which is a single tif file location. Then we command it to use the PPT_PEARSON_FINAL_COLOR txt file for the color argument. The output gets sent to ppt_index_color_output and a flag called -alpha is attached.

We then use gdal2tiles.py to build the XYZ tiles and specify the --zoom flag to be between 2 and 8 and the –tilesize flag to be 128. We use the output from the previous command called ppt_index_color_output as an input while declaring the output as PPT_COR_XYZ_OUTPUT, which is the PPT_COR_XYZ_BASE plus the increment. We next add a value of 1 to the increment variable. Finally, an if statement is used to tell the program if the increment variable equals 2015, then we stop the for loop and move on.

```
# Make ppt_bil2tif_resize variable
ppt_bil2tif_resize="$VARIABLENAME/data/ppt_bil2tif_resize"
# ppt_pearson_final_color
PPT_BIL2TIF_RESIZE_FINAL_COLOR=$VARIABLENAME/data/
ppt_pearson_final_color.txt
COLOR="_color.tif"
# PPT_COR_XYZ_BASE folders for xyz tiles
PPT_BIL2TIF_XYZ_BASE="/var/www/html/ppt_bil2tif_resize_xyz/
ppt_bil2tif_resize_xyz_ppt_bil2tif_resize_"

increment=1981
# Loop over ppt_bil2tif_resize, concat _color.tif to the output, build xyz tiles
for ppt_resize in "$ppt_bil2tif_resize"/*_reclassed.tif
do
        ppt_resize_color_output=${ppt_resize%%.*}$COLOR
        PPT_BIL2TIF_XYZ_OUTPUT=$PPT_BIL2TIF_XYZ_BASE$increment
        sudo gdaldem color-relief $ppt_resize $PPT_BIL2TIF_RESIZE_FINAL_COLOR
        $ppt_resize_color_output -alpha
```

```
        echo "Color tif for year "$increment
        sudo gdal2tiles.py --zoom=2-8 --tilesize=128 $ppt_resize_color_output
        $PPT_BIL2TIF_XYZ_OUTPUT
        echo "xyz tiles for year"$increment
  increment=$((increment+1))
  if [[ $increment -eq 2015 ]];
  then
    break
  fi
done
```

Part 7: Build the web applications using NPM

The final step is to build the web applications. We first move all the txt quantile files into the web application locations with mv and a wildcard. Then we cd into each of the applications and use the command run-script build to create the files. Finally, we move the build dist folders onto the web server.

```
# Move the quantile txt files from /data/ to server for legend
sudo mv $VARIABLENAME/data/tmean_pearson_final/*txt $VARIABLENAME/
pandamoniumGIS20210110_tmeanbuild/data/reclassed_txt
sudo mv $VARIABLENAME/data/ppt_pearson_final/*txt $VARIABLENAME/polarbear/data/
reclassed_txt

sudo mv $VARIABLENAME/data/tmean_pearson_final/*txt $VARIABLENAME/
pandamoniumGIS20210110_tmeanbuild/data/reclassed_txt_tmean_og
sudo mv $VARIABLENAME/data/ppt_pearson_final/*txt $VARIABLENAME/polarbear/data/
reclassed_txt_ppt_og

# Build node applications with npm
cd pandamoniumGIS20210110_tmeanbuild
sudo npm run-script build
cd ..
cd pandamoniumGIS_part2Section2_build
sudo npm run-script build
cd ..
cd polarbear
sudo npm run-script build
cd ..
cd tha_difference
sudo npm run-script build
cd ..
cd polar_radar
sudo npm run-script build
cd ..
```

```
# Move the newly created files to their web home
sudo mv ~/polarbearGIS/pandamoniumGIS20210110_tmeanbuild/dist /var/www/html/
pandamoniumGIS20210110_tmeanbuild
sudo mv ~/polarbearGIS/pandamoniumGIS_part2Section2_build/dist /var/www/html/
pandamoniumGIS_part2Section2_build-dist
sudo mv ~/polarbearGIS/polarbear/dist /var/www/html/polarbearGIS
sudo mv ~/polarbearGIS/tha_difference/dist /var/www/html/tha_difference-dist
sudo mv ~/polarbearGIS/polar_radar/dist /var/www/html/polar_radar
```

And we're done!

Generating Quantile Breaks and a Time-slider with AngularJS

Part 1: Calculating quantile breaks with python

The first step is to import some libraries, including numpy, gdal, subprocess, os, and glob. We create a variable named cwd which gets the current working directory. Then we create a variable that concatenates the cwd to the /data/ppt_pearson_final/ directory. Then we use glob and a wildcard to grab all the the tifs in the folder. This list is then sorted.

```
import numpy as np
from osgeo import gdal, gdal_array
import subprocess
import os
from glob import glob

cwd = os.getcwd()
input_raster_folder = cwd + '/data/ppt_pearson_final/'
input_raster_folder_list = glob(os.path.join(input_raster_folder, '*.tif'))
input_raster_folder_list.sort()
```

Part 2: Creating a for loop and reading in the data

Next we loop over the list of file paths contained in input_raster_folder_list with an index variable called input_Raster_folder_list. We read that in with gdal and call it dataset. The GetRasterBand method is applied to dataset and set to band. Band is then read as an array into the variable array. To avoid no data values, we create a variable named nodata_val and get the no data value from the variable band. We use an if statement to create a variable named array as a np.ma.masked_equal, based on the array variable and nodata_val. We then take all the values greater than or equal to the array's minimum value and set it to the variable array_ignored_nan. We also call an output array variable as a numpy array of zeros in place of the array values.

```
for input_raster in input_raster_folder_list:
```

```
# Open the dataset and retrieve raster data as an array
dataset = gdal.Open(input_raster)
band = dataset.GetRasterBand(1)
array = band.ReadAsArray()

nodata_val = band.GetNoDataValue()
if nodata_val is not None:
    array = np.ma.masked_equal(array, nodata_val)

array_ignored_nan = array[array >= array.min()]

# Create an array of zeros the same shape as the input array
output = np.zeros_like(array).astype(np.uint8)
```

Part 3: Calculating percentiles, writing to txt files, and saving as raster tif

Now we set five variables to be percentiles based on 80, 60, 40, 20, and 0. These are also rounded to five decimal places. We then save these values into a txt file for later use in our web applications by taking the path and using splitext to take the basic tif file without the extension and concatenating _reclassed.txt string onto the end of it. This string is used in a write statement for our five percentile variables. Next we classify the array variable where the array is greater then percentile 0 (which is all the data) by using the output variable of numpy zeros, replacing the data with ones. We set this array to the variable output. Then we update the output variable by replacing all the array values greater than percentile 20 (80% of the data) with a value of 2. And so on until the 80th percentile. Then we use gdal_array SaveArray method to save the variable output to the outname variable, which is the input_raster path concatenated with the string _reclassed.tif. Finally, we build the XYZ files as described in the BASH script from before.

```
# Use the numpy percentile function to calculate percentile thresholds, gotta round for
scientific notation
percentile_80 = round(np.percentile(array_ignored_nan, 80), 5)
percentile_60 = round(np.percentile(array_ignored_nan, 60), 5)
percentile_40 = round(np.percentile(array_ignored_nan, 40), 5)
percentile_20 = round(np.percentile(array_ignored_nan, 20), 5)
percentile_0 = round(np.percentile(array_ignored_nan, 0), 5)

print(percentile_0, percentile_20, percentile_40, percentile_60, percentile_80)

txt_outname = os.path.splitext(input_raster)[0] + "_reclassed.txt"
print(txt_outname)

with open(txt_outname, "w") as text_file:
    text_file.write(str(percentile_0) + " " + str(percentile_20) + " " + str(percentile_40) + " " +
    str(percentile_60) + " " + str(percentile_80))
```

```
    output = np.where((array > percentile_0), 1, output)
    output = np.where((array > percentile_20), 2, output)
    output = np.where((array > percentile_40), 3, output)
    output = np.where((array > percentile_60), 4, output)
    output = np.where((array > percentile_80), 5, output)

    outname = os.path.splitext(input_raster)[0] + "_reclassed.tif"
    gdal_array.SaveArray(output, outname, "gtiff", prototype=dataset)

    print(outname)
print("quantile_reclassify.py complete")
```

Part 4: Updating the layers.js in the web app to include every year's precipitation raster

It's ugly but it does work:

```
const ppt_bil2tif_resize_url = "https://www.<website_name>.com/ppt_bil2tif_resize_xyz/
ppt_bil2tif_resize_xyz_ppt_bil2tif_resize_"

// Create ppt_bil2tif_resize_year to hold array of url strings
const ppt_bil2tif_resize_year = [];
for (var ii in year){
  var staged = ppt_bil2tif_resize_url + year[ii] + xyz_ending;
  ppt_bil2tif_resize_year.push(staged);
};
// Create array ppt_bil2tif_resize_all to hold all the tile layers
// Loop over ppt_bil2tif_resize_year and year arrays
const ppt_bil2tif_resize_all = []
ppt_bil2tif_resize_year.forEach((iiii, index) => {
  const ppt_num2 = year[index];
  const ppt_staged_tile_layer = new TileLayer({
    title: ppt_num2 + ' Jan ppt',
    source: new XYZ({
    url: iiii,
    }),
  });
  ppt_bil2tif_resize_all.push(ppt_staged_tile_layer);
});
```

Part 5: Using a promise/await function to add the first layer

Now we make a function called sleep which takes the argument ms. It returns a new Promise which resolves after a set time. The async function is called demo and uses await to sleep for a

brief period, then add the precipitation layer on top of the precipitation correlation layer. This is done to show the viewer that there are two layers to explore. A button is included in the top right corner to either slide transparency or turn off/on the two layers.

```
// Promise and await for the ppt layer
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
async function demo() {
  await sleep(5000);
  map.addLayer(ppt_bil2tif_resize_all[0])
}
demo();
```

Part 6: Writing more ugly code that works

Now we write some horrendously repeating code that uses ‘fs’ to read in the txt files containing our correlation quantile breaks. It’s basically 33 variables for 33 years, which we then put in a variable array called list_of_txt_var.

Hindsight: A better way to do this would be to use a database.

```
const fs = require('fs')

const ppt_pearson_final_1981_reclassed = fs.readFileSync('./data/reclassed_txt/
ppt_pearson_final_1981_reclassed.txt', 'utf8')
const ppt_pearson_final_1982_reclassed = fs.readFileSync('./data/reclassed_txt/
ppt_pearson_final_1982_reclassed.txt', 'utf8')
...etc
const list_of_txt_var = [ppt_pearson_final_1981_reclassed,
                         ppt_pearson_final_1982_reclassed,...etc]
```

Part 7: Sending the split txt files to innerHTML

Then we take the first value of the array list_of_txt_var, split it based on a space, and set it to a variable named words. Then each value of the words array (i.e. words[0], words[1], etc.) has the substr taken of it and set to a variable named zero_value, twenty_value, etc. These values are sent to the document by the method getElementById, an example html id being percentile_0. This is also repeated for the precipitation quantiles.

```
// Add in txt files for initial year for cor quantiles
var words = list_of_txt_var[0].split(" ");
var zero_value = words[0].substr(0, 22);
document.getElementById('percentile_0').innerHTML = zero_value;
var twenty_value = words[1].substr(0, 22);
```

```
document.getElementById('percentile_20').innerHTML = twenty_value;
var forty_value = words[2].substr(0, 22);
document.getElementById('percentile_40').innerHTML = forty_value;
var sixty_value = words[3].substr(0, 22);
document.getElementById('percentile_60').innerHTML = sixty_value;
var eighty_value = words[4].substr(0, 22);
document.getElementById('percentile_80').innerHTML = eighty_value;

// Add in txt files for initial year for ppt_og quantiles
var words_og = list_of_ppt_var[0].split(" ");
var zero_value_og = words_og[0].substr(0, 22);
document.getElementById('percentile_0_ppt_og').innerHTML = zero_value_og;
var twenty_value_og = words_og[1].substr(0, 22);
document.getElementById('percentile_20_ppt_og').innerHTML = twenty_value_og;
var forty_value_og = words_og[2].substr(0, 22);
document.getElementById('percentile_40_ppt_og').innerHTML = forty_value_og;
var sixty_value_og = words_og[3].substr(0, 22);
document.getElementById('percentile_60_ppt_og').innerHTML = sixty_value_og;
var eighty_value_og = words_og[4].substr(0, 22);
document.getElementById('percentile_80_ppt_og').innerHTML = eighty_value_og;
```

Part 8: Reading in the El Nino 3.4 index and displaying it on the map

We use fs to readFileSync the nino34.csv that is contained in our data folder and take the sub string, starting with the first index value, then the year, then index value, and so on. We replace all the line breaks with commas, then split the variable by commas into an array of each value. A function is further used to make an array called elnino_index_value of every other value in the previously defined array. This works because it takes the first index value, then skips the year, then takes the second index value, then skips the year, etc. Finally, the first index value is added to the map via getElementById. We also set the first year type as neutral, since it is below .5 and above -.5.

```
// Read in nino34 index
const elnino_index_sub = fs.readFileSync('../data/nino34.csv', 'utf8').substr(16, 352);
// Replace the strings and split by comma
const elnino_index = elnino_index_sub.replace(/\n/g, ",").split(",")
// Get every other index value and make an array
const elnino_index_value = elnino_index.filter((element, index) => {
  return index % 2 === 0;
})
document.getElementById('nino').innerHTML = elnino_index_value[0];
// Make initial year type
document.getElementById('year_type').innerHTML = "Neutral"
```

Part 9: Using AngularJS to make a timeslider, legend values, and El Nino 3.4 Index

First we make a variable named myApp with the angular module myApp and rzSlider. This myApp has a reference in the html index file, which defines where the slider is to appear on the map. We use a scope function for this, which contains an empty array called dates, where we append the years between 1981 and 2014. For the scope slider, our value is initialized as the first year of our dates array, 1981. Then in an options bracket, we define the stepsArray option as the dates array and the id as 'slider-id'. Then the option onChange is defined as a very long function.

The variable v is set to id and an Object.keys is set for the array containing the all_tile_layers array (which contains all the XYZ tiles loaded into open layers). The forEach loop is used with a nested function, which takes the argument key. Then in this function, we ask if the v variable (which is our id) is equal to the year at a specified key, we addLayer to the map the all_tile_layers at that particular key. The first value of the array corresponds to 1981 (although first value is also known as zero). Next we use the async function again to await 5000 milliseconds to add the ppt_bil2tif_resize_all layer at that particular key. Then we use the same code as before to split the list_of_txt_var and list_of_ppt_var and send it to the legend html, for our quantile break values. This is based on accessing the array value based on the key value. We also send the elnino_index_value to the html document based on the key. A simple if, else if, else statement sends the appropriate string if the year type is El Nino, La Nina, or Neutral.

Finally, we finish the original function's if statement with an else statement and removeLayer from the map the all_tile_layers based on the key. We also use another delayed async function to remove the ppt_bil2tif_resize_all based on the key.

```
var myApp = angular.module('myApp', ['rzSlider'])
myApp.controller('GreetingController', ['$scope', function($scope) {
   var dates = [];
   for (var i = 1981; i <= 2014; i++) {
    dates.push(i);
   }
   $scope.slider = {
    value: dates[0],
    options: {
     stepsArray: dates,
     id: 'slider-id',
     onChange: function(event, id) {
      var v = id;
      Object.keys(all_tile_layers).forEach(function(key){
       if (v == year[key]){
        map.addLayer(all_tile_layers[key]);
        // Wait 5 seconds to add the ppt layer
        async function demo1() {
         await sleep(5000);
```

```
    map.addLayer(ppt_bil2tif_resize_all[key])
  }
  demo1();
  // Split the txt file by space and send it to the p id in html doc
  var words = list_of_txt_var[key].split(" ");
  var zero_value = words[0].substr(0, 22);
  document.getElementById('percentile_0').innerHTML = zero_value;
  var twenty_value = words[1].substr(0, 22);
  document.getElementById('percentile_20').innerHTML = twenty_value;
  var forty_value = words[2].substr(0, 22);
  document.getElementById('percentile_40').innerHTML = forty_value;
  var sixty_value = words[3].substr(0, 22);
  document.getElementById('percentile_60').innerHTML = sixty_value;
  var eighty_value = words[4].substr(0, 22);
  document.getElementById('percentile_80').innerHTML = eighty_value;
  // Add in nino index
  var nino_index_key = elnino_index_value[key]
  // Logic to decide what year type it is
  document.getElementById('nino').innerHTML = nino_index_key;
  if (nino_index_key > .5){
    document.getElementById('year_type').innerHTML = "El Nino"
  } else if (nino_index_key < -.5){
    document.getElementById('year_type').innerHTML = "La Nina"
  } else{
    document.getElementById('year_type').innerHTML = "Neutral"
  }
  // Add in txt files for initial year for ppt_og quantiles
  var words_og = list_of_ppt_var[key].split(" ");
  var zero_value_og = words_og[0].substr(0, 22);
  document.getElementById('percentile_0_ppt_og').innerHTML = zero_value_og;
  var twenty_value_og = words_og[1].substr(0, 22);
  document.getElementById('percentile_20_ppt_og').innerHTML = twenty_value_og;
  var forty_value_og = words_og[2].substr(0, 22);
  document.getElementById('percentile_40_ppt_og').innerHTML = forty_value_og;
  var sixty_value_og = words_og[3].substr(0, 22);
  document.getElementById('percentile_60_ppt_og').innerHTML = sixty_value_og;
  var eighty_value_og = words_og[4].substr(0, 22);
  document.getElementById('percentile_80_ppt_og').innerHTML = eighty_value_og;
}
else {
  map.removeLayer(all_tile_layers[key]);
  map.removeLayer(ppt_bil2tif_resize_all[key])
  // Remove ppt layer
  async function demo2() {
    await sleep(5001);
    // Clear the map of ppt layer
```

```
            map.removeLayer(ppt_bil2tif_resize_all[key])
          }
          demo2();
        }
      });
    }
  }
};
}]);
```

That was the interesting JavaScript portion, more HTML that is not included here was added for display.

Other projects included running a correlation window across climate raster datasets and a web application to add sightings of trees to a MariaDB.

Fall 2021 - Present
Chicago and Wisconsin

I moved to Chicago but quit my job after two months and returned to my parents house.

I worked as an Upwork contractor on two paid projects. In the first contract, I wrote an address sequencer using Python and a web visualization with a login and time slider function. In the second I researched how to serve Vector/Raster Tiles for a couple hours.

I worked a few months at a local hospital as a custodian. I mainly worked here because I need health insurance but another benefit was that I helped out a staffing shortage during the Corona Virus spike of early 2022. Outside of work one of my coworkers encouraged me to make a shift countdown timer using JavaScript, which I completed for free because I enjoy programming. The page launches and lands a rocket that I drew at the end of each shift.

I also bought a RaspberryPi and experimented with local area networks but I think my networking is bad. It used Python Flask and Minikube to convert a wifi network signal into distance to router to estimate which room my phone was in and play a song. I also made a mediocre music application in JavaScript without libraries. I have bills to pay so I stopped programming for free in early March of 2022 but still read about networks occasionally.

I briefly worked for a grocery store until I was hired by another company as a part-time merchandiser in grocery stores where I currently work.