Introduction to Spatial Analysis in R

CODE Academy 2023

Megan Malish (megan.malish@ou.edu)

1

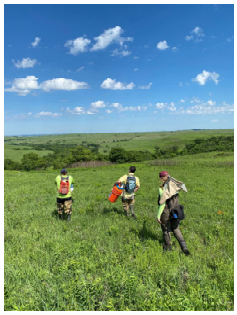
Megan Malish

PhD Candidate
University of Oklahoma

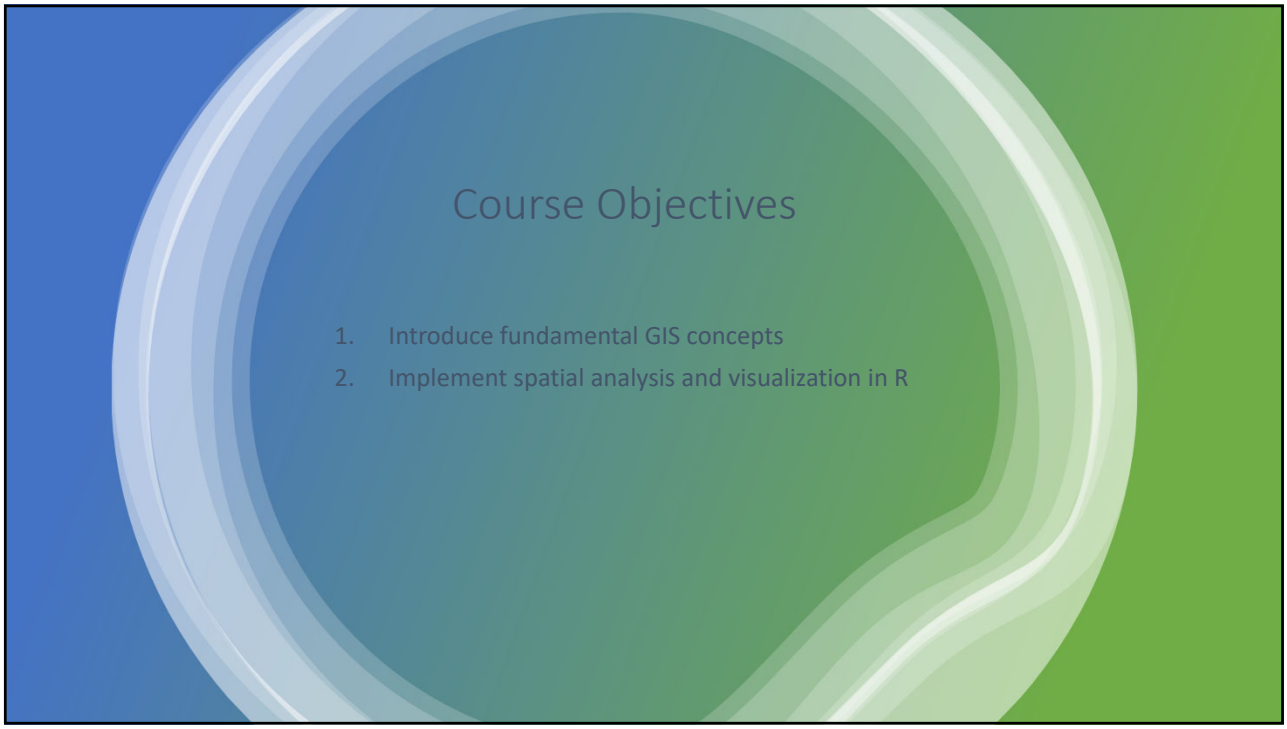
The goal of my research is to understand drying patterns of streams, how those drying patterns might change in the future, and why that matters.

I get to spend some of my time sampling for bugs in rivers.

Most of my time is spent on the computer, coding in R to analyze and understand hydrologic and ecological data.




2


Course Objectives
1. Introduce fundamental GIS concepts
2. Implement spatial analysis and visualization in R

3


Course Overview
Day 1
• Introductions
• Background Information: GIS and R
• R Warm-Up
• Start Tutorial
Day 2
• Spatial R Warm-Up
• Recap Day 1
• Finish Tutorial
• Making maps (if time)

4

Day 1

5

What is a GIS?

A *geographic information system* is a computer-based information system that enables capture, modeling, storage, retrieval, sharing, manipulation, analysis, and presentation of geographically referenced data.

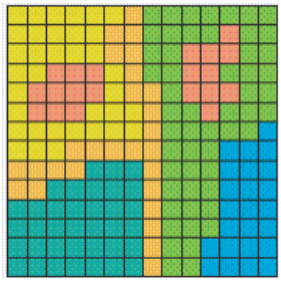
6

What can we do with a GIS?

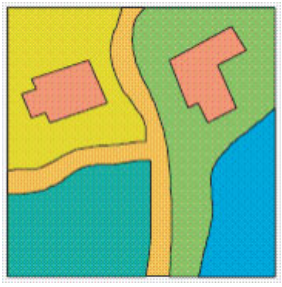
1. Inventory: What is this? Where is this?
2. Network Analysis: How do we get from location to location? What is the fastest way? What is the shortest way?
3. Geospatial Analysis: Where is the closest hospital? Where is the best location to open a new business?
4. Spatiotemporal Analysis: How have property lines changed since 1950?

7

Types of Spatial Data



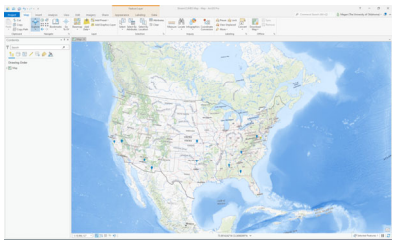
Raster

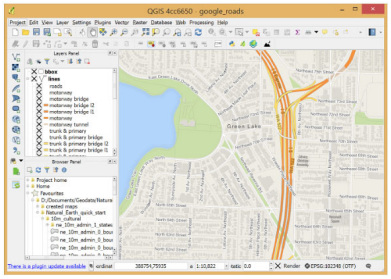


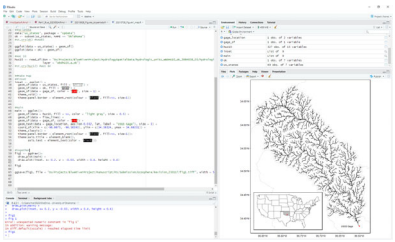
Vector

8

GIS Software







9

Why Use R?

- Open Source vs. Proprietary Software
- Geospatial Analysis vs. Geospatial Visualization
- Coding vs. GUI

10

R Warm-Up 1

Open RStudio. Open a new .R file and complete the following:

1. Find the answer to **3 + 5 * 2**
2. Assign the answer to #1 to a variable called 'out1'
3. Find the answer to **2 * 2 + 6**
4. Assign the answer to #2 to a variable called 'out2'
5. Is *out1* is greater than *out2?*

11

Other Key Reminders

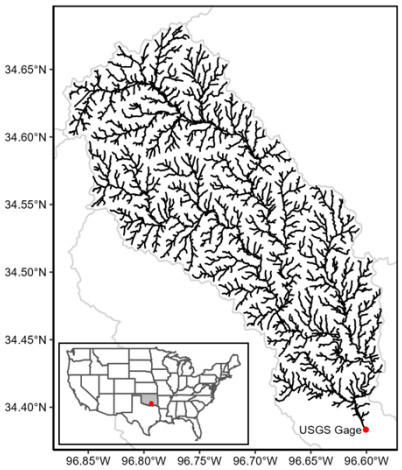
- To install a new package, use:
  `install.packages("package")`
- To load a package, use:
  `library(package)`
- To set your working directory, use:
  `setwd("this/is/the/path")`
- To read data into R, use:
  `read.csv("file.csv")`

12

5 MINUTE BREAK!

13

Blue River, OK



14

Data Overview

- Spatial Data: flowlines of Blue River
- Non-Spatial Data: modeled stream flow for the Blue River
  - Current stream flow
  - Future stream flow

15

Spatial Analysis

- Explore stream drying patterns in the Blue River
- Measure stream drying patterns in the Blue River
- Determine if and how stream drying patterns will change as a result of climate change

16

Day 2

17

Course Overview

Day 1

- Introductions
- Background Information: GIS and R
- R Warm-Up
- Start Tutorial

Day 2

- Spatial R Warm-Up
- Recap Day 1
- Finish Tutorial
- Making maps (if time)

18

R Warm-Up 2

Open RStudio. Open a new .R file and complete the following:

1. Load the file blue_current.csv into R
2. Use one function to look at the first few rows of the dataset
3. Calculate the mean of the second column of the dataset (X20020101)
4. Load the shapefile blue_streams into R
5. Plot (make a map) of blue_streams

19

Spatial Analysis in R

Learning Objectives

1. Introduce fundamental GIS concepts.
2. Implement spatial analysis and visualization in R.

Tools and Resources Required

1. Access to the folder *RSpatial*
2. R & RStudio

Lab Overview

1. Prepare data.
2. Visualize stream drying.
3. Measure stream drying.

Background

About the Blue River Watershed

In this lab, we will examine stream drying patterns in the Blue River watershed. The Blue River is located in the south central part of Oklahoma, USA. This region experiences temperature and precipitation extremes, which result in periods of drought and flood. The average temperature is 62.5°F with an average of 73 days per year above 90°F. Average annual rainfall is 1080 mm, with monthly amounts ranging from 55 mm in January to 141mm in May. Parts of the watershed are connected to a groundwater source called the Arbuckle Simpson aquifer.

Lab Activity

Part A1. Prepare Data

File Organization

Before we begin working with our data in R, we have to make sure our data files are organized and accessible.

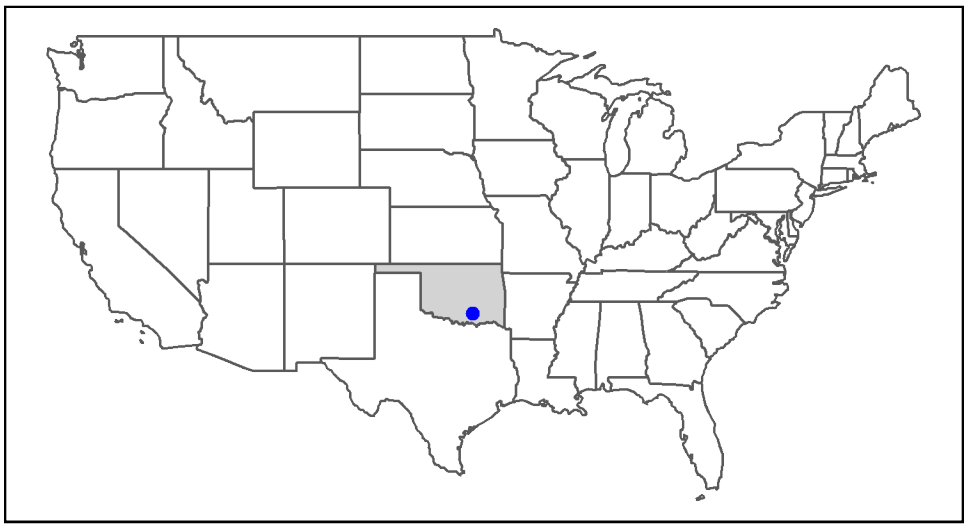


Figure 1: Location of Blue River Watershed

1. Save the zipped folder *RSpatial.zip* to your desktop.
2. Extract the contents of the zipped folder.
    - Windows: Right click on the zipped folder *RSpatial.zip* and select "Extract All". When a new Window pops up, select "Extract".
    - Mac: Double-click on the zipped folder.

Load R Packages

Now we are ready to work in R. Open RStudio.

We first have to load the necessary R packages using the `library()` function. The package `sf` is used for working with spatial data and `ggplot2` helps make figures.

If you have not previously installed these packages, run this line of code:

```
install.packages(c("sf", "ggplot2"))
```

Then, everybody will load the R packages. You will need to load your required packages each time you open R.

```
library(sf)
library(ggplot2)
```

You may get warnings here. Remember that warnings are usually okay, but errors are not.

Load Data

Set the working directory to the folder *LabMaterials* using the function `setwd()`. Setting the working directory lets RStudio know the location of the files we will be working with.

```
setwd("location/of/desktop/RSpatial") #replace with the actual location of the folder

#should look something like: setwd("C:/Users/mali0023/Desktop/RSpatial")
```

Alternatively, you can set the working directory using the drop down menus. Do this by selecting *Session > Set Working Directory > Choose Directory* and manually navigating to the folder *RSpatial*.

Check that your working directory was set correctly using the following code.

```
getwd()
```

The output of this line of code will be different than what's above for you, but should be the path to the folder *LabMaterials*. If you are having trouble setting your working directory using `setwd()`, double check spelling, that you're using the forward slash (/) and not a back slash (\), and that the directory is in quotation marks.

Now that we have set our working directory, we will load three datasets.

The first is a spatial dataset, called a shapefile, that maps the shape of the streams within the Blue River Watershed. We will load the shapefile of the streams in the watershed using the function `read_sf()`. We will name this object 'flow_lines' so we can easily refer to it later.

```
flow_lines <- read_sf(dsn = "data", #name of folder where shapefile is located
                      layer = "blue_streams") #name of shapefile
```

The second and third datasets contain daily stream flow data from hydrological models.

We will load the daily stream flow data using the function `read.csv`.

```
current_flow <- read.csv("data/blue_current.csv")
future_flow <- read.csv("data/blue_future.csv")
```

About This Data Hydrological models are computer simulations of the movement of water through a watershed. They can be used to help us understand stream flow patterns and factors that affect the movement of water.

In this lab, we will work with data from two hydrological models, called *current_flow* and *future_flow*. The data from the *current_flow* model simulates stream flow for current climate conditions. The data from the *future_flow* model simulates stream flow for climate conditions expected in about 50 years from now. These two datasets contain two years of modeled daily stream flow for every stream segment contained in the shapefile that we previously loaded and named 'flow_lines'.

Next, we will look more closely at these datasets.

Part A2. Visualize Stream Drying

Examine Data

Use the function `head()` to look at the first few rows of data from the shapefile.

```
head(flow_lines)
```

```
## Simple feature collection with 6 features and 4 fields
## Geometry type: LINESTRING
## Dimension:     XY
## Bounding box:  xmin: -96.7744 ymin: 34.67505 xmax: -96.76546 ymax: 34.68231
## Geodetic CRS:  NAD83
## # A tibble: 6 x 5
##   ARCID GRID_CODE FROM_NODE TO_NODE                                   geometry
##   <int>     <int>     <int>   <int>                             <LINESTRING [°]>
## 1     1         4         4       6 (-96.77088 34.67991, -96.77088 34.6794, -96~
## 2     2         2         2       6 (-96.76759 34.6813, -96.7681 34.68079, -96.~
## 3     3         1         1       7 (-96.76546 34.68231, -96.7656 34.68218, -96~
## 4     4         5         5       7 (-96.76574 34.67889, -96.76588 34.67875, -9~
## 5     5         6         6       9 (-96.77106 34.67866, -96.77134 34.67838, -9~
## 6     6         7         7       9 (-96.76856 34.67801, -96.76875 34.67782, -9~
```

You first see some information about the spatial attributes of the file. We can see that the geometry type, or type of spatial data, contained in this file is called "LINESTRING". This means that the data represents lines. The data has information about for the X and Y dimensions. The bounding box information describes the spatial extent of the data, meaning how far it extends in the X and Y (or east/west and north/south) dimensions. Geodetic CRS provides information about how the spatial data is projected, or drawn.

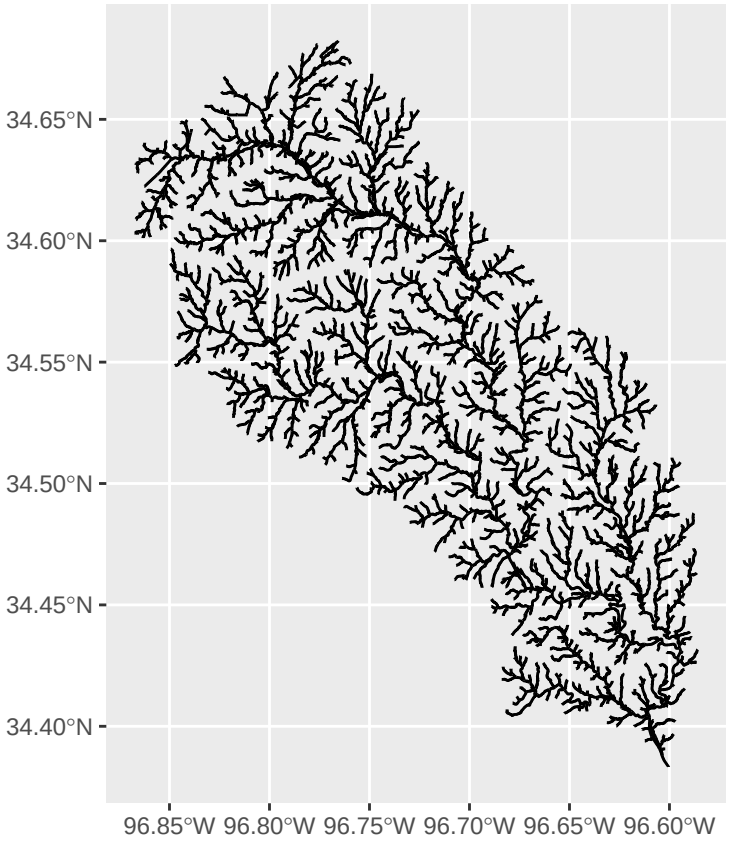
Following that, you see that "flow_lines" is organized into rows and columns.

The entire watershed is divided into smaller stream segments. Each row in the dataset contains the information for a stream segment.

You can see that the dataset contains 5 columns. 'ARCID' and 'GRID_CODE' are used to identify stream segments. 'FROM_NODE' and 'TO_NODE' contain information about which stream segments are connected to each other. The information about how to draw each line is contained in the column 'geometry'.

We can also plot the flow lines. We will do this using `ggplot()`, which is used to create graphics. `geom_sf()` specifies that we are creating a map of our data.

```
ggplot() + geom_sf(data = flow_lines)
```



Now we can explore the stream flow data for the two climate scenarios.

```
head(current_flow)
head(future_flow)
```

In the stream flow datasets, each row contains the mean daily flow (cubic meters per second) for each stream segment, identified by the column called 'ARCID'. Columns are days. For example, 'X20020101' contains data for January 1, 2002. We have data for two years of data, Januay 1, 2002 through Dec 31, 2003.

Notice that all files contain a column 'ARCID'. Later we will use this information to combine the flow line data with the daily flow data.

Now that we understand the structure of our data, we can explore it more.

Explore Data

We can get to know our data better by using some simple analyses.

Starting with the current stream flow data, let's measure the stream flow at the watershed outlet for each day.

We'll start by indexing only the stream segment at the outlet of the watershed and the columns that contain stream flow data (remember that the first column of the dataset is ARCID, which is just a label). All the water in the watershed will eventually leave the watershed through the outlet. In this case, the outlet of the watershed is the stream segment with ARCID 2510, which is located in row 2510. We will use this information to pull out the data we want and save this information with the name 'current_flow_outlet'.

```r
#index row 2510 and all columns except the 1st column
current_flow_outlet <- current_flow[2510,-1]

#convert the data so it contains only numeric information
current_flow_outlet <- as.numeric(current_flow_outlet)
```

Recall that our stream flow data is in units of cubic meters per second. We can convert this to daily stream flow, or cubic meters per day.

```r
current_flow_outlet <- current_flow_outlet *60*60*24 #60 sec/min, 60 min/hr, 24 hr/day
```

Now that we have our data set up, we can calculate some basic metrics, such as mean and standard deviation of daily stream flow at the outlet.

```r
mean(current_flow_outlet) #mean watershed daily stream flow
```

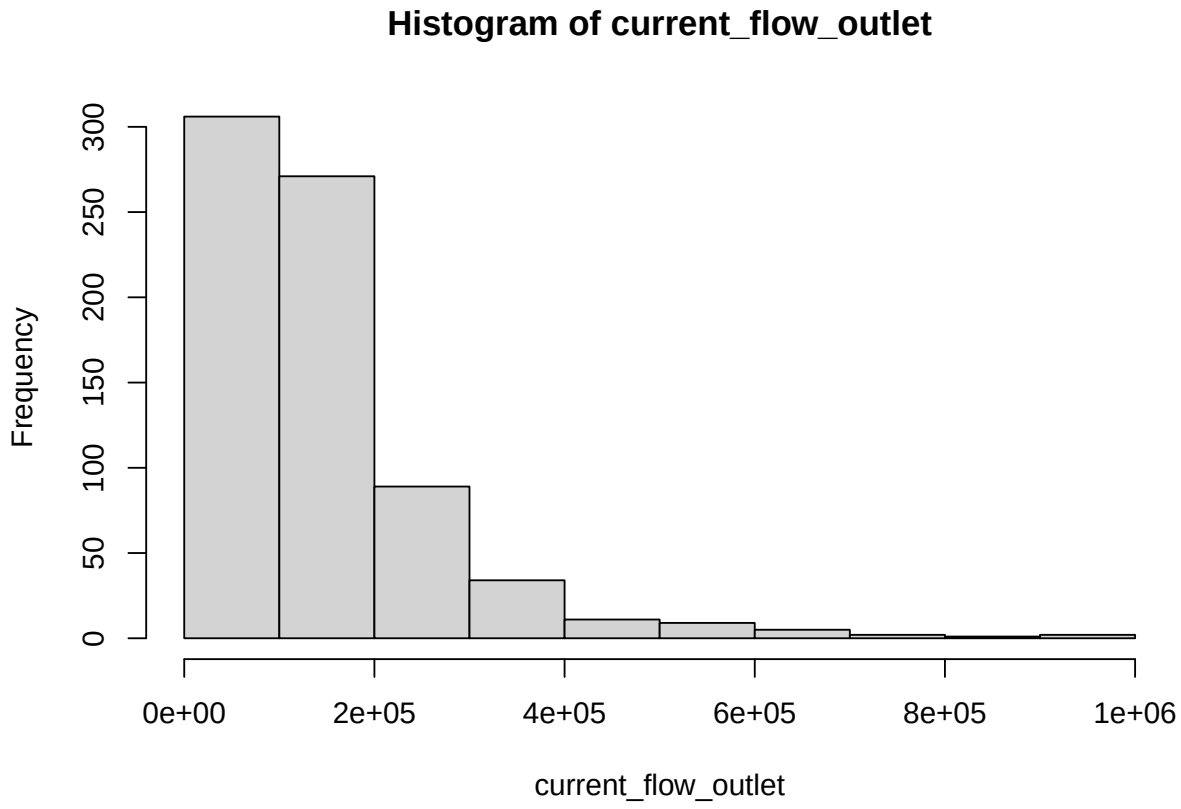
```
## [1] 150629.8
```

```r
sd(current_flow_outlet) #standard deviation of daily stream flow
```

```
## [1] 121588.6
```

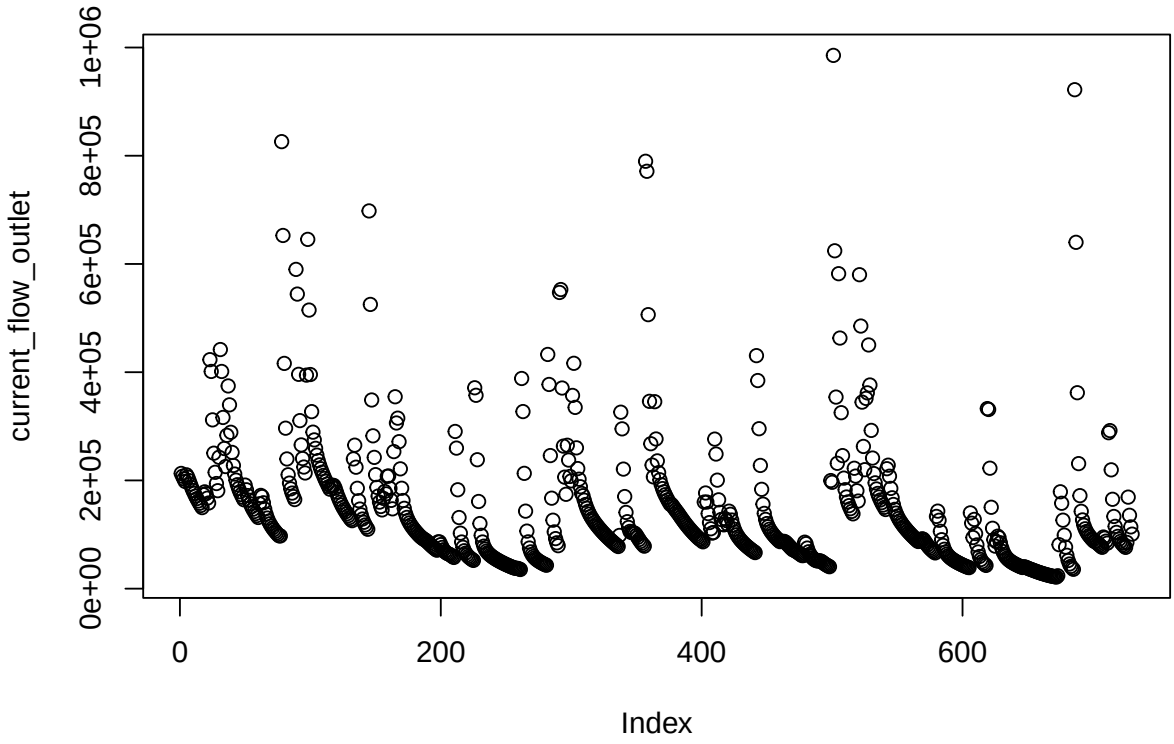
We can use a histogram to explore the distribution of daily stream flow values.

```r
hist(current_flow_outlet, breaks = 10)
```



We can also plot daily stream flow over time using the `plot()` function.

```
plot(current_flow_outlet)
```



Interpreting the plot: In this plot, the x-axis, labeled "Index", represents time at a daily timestep. The point at the index value of 1 is the first day of this dataset, January 1, 2002. The point at the Index value of 366 is the 366th day of this dataset, January 1, 2003. The y-axis is daily stream flow (cubic meters per day). Each point represents the stream flow for a given day.

Exercise 1: Repeat these steps for the *future_flow* dataset.

Use the following code:

```
future_flow_outlet <- future_flow[2510,-1] #extract outlet
futuret_flow_outlet <- as.numeric(future_flow_outlet) #convert to numeric

future_flow_outlet <- future_flow_outlet *60*60*24 #convert to daily stream flow

mean(future_flow_outlet) #mean watershed daily stream flow
sd(future_flow_outlet) #standard deviation of daily stream flow

hist(future_flow_outlet, breaks = 10)
plot(future_flow_outlet)
```

Manipulate data

Now that we have a basic understanding of our data, we can begin to work with our datasets and ask questions.

Because we are interested in understanding stream drying, we can convert our data from mean daily flow to a simpler form. In this case, we will convert the mean daily flow data to wet/dry status. We will use the number 0 to represent dry stream segments and the number 1 to represent wet stream segments. In our daily flow datasets, we will change any flow values greater than 0 cubic meters per second to be equal to 1.

Recall that the first column of our datasets are 'ARCID' and the remaining columns represent days and contain stream flow data. We want to make sure we only change the columns that contain stream flow data.

```
wetdry_current <- current_flow  #create new data frame to manipulate
wetdry_current[,-1][wetdry_current[,-1] > 0] <- 1
```

The above code does the following: In all columns except the first column of the data set 'wetdry_current', if the value is greater than 0, replace it with a 1.

Repeat for the future flow dataset.

```
wetdry_future <- future_flow
wetdry_future[,-1][wetdry_future[,-1] > 0] <- 1
```

Now, in both our datasets, 0 represents a dry stream segment and 1 represents a wet stream segment.

Next, we will create new datasets that contain both the spatial data and the stream flow data. To do this, we merge the flow lines with the daily wet/dry status data. This creates a spatial dataset that contains the daily wet/dry status of each stream segment.

A function called `merge()` exists in several R packages. To make sure we are calling the correct function, from base R, we call the function using `base::merge()`. The function `merge()` requires three arguments: the name of each dataset we want to combine and a specification of how we want to combine the datasets. We will use the 'ARCID' columns in our datasets to merge them.

The new spatial datasets we create will be called 'current_sp' and 'future_sp'.

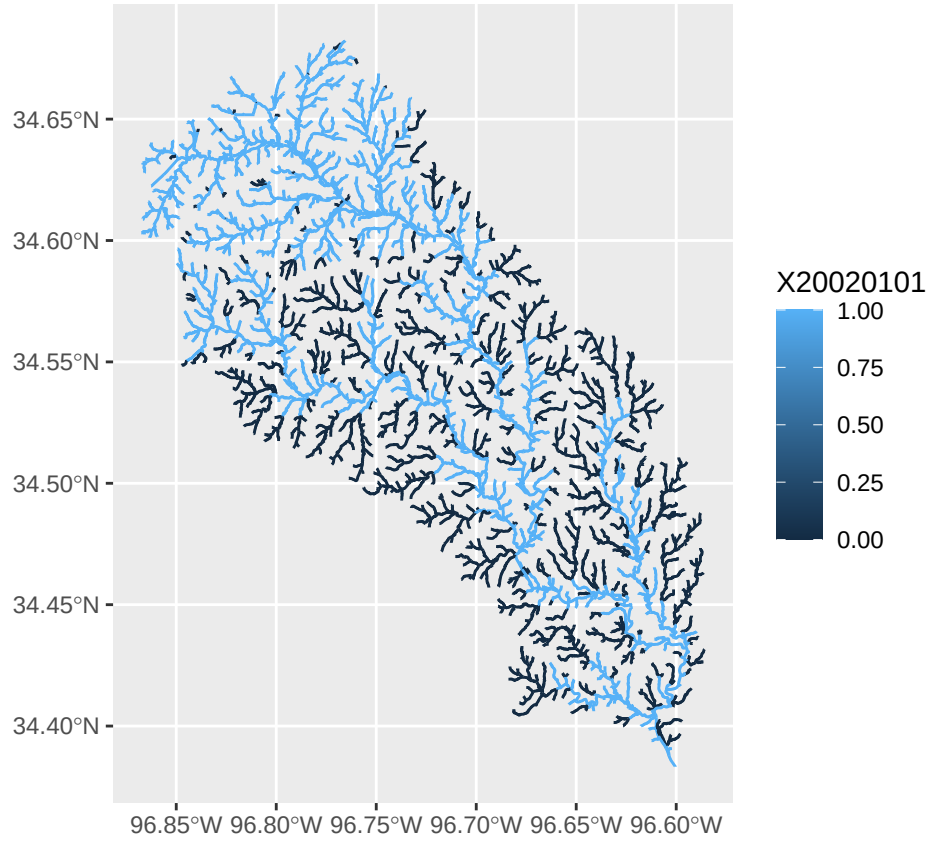
```
#create new dataset that contains spatial flowline data and wet/dry data from the current simulation
current_sp <- base::merge(flow_lines, wetdry_current, by = 'ARCID')

#create new dataset that contains spatial flowline data and wet/dry data from the future simulation
future_sp <- base::merge(flow_lines, wetdry_future, by = 'ARCID')
```

Visualize data

Now that we have combined our flow data with our spatial data, we can map daily wet/dry status of the watershed.

Start with visualizing the first day of the dataset (January 1, 2002) for current and future climates. We will use similar code as when we first mapped the dataset, except we will add more information. Using `aes()` allows us to control the aesthetics of the map. Here, we specify that we want the colors to represent values for January 1, 2002.

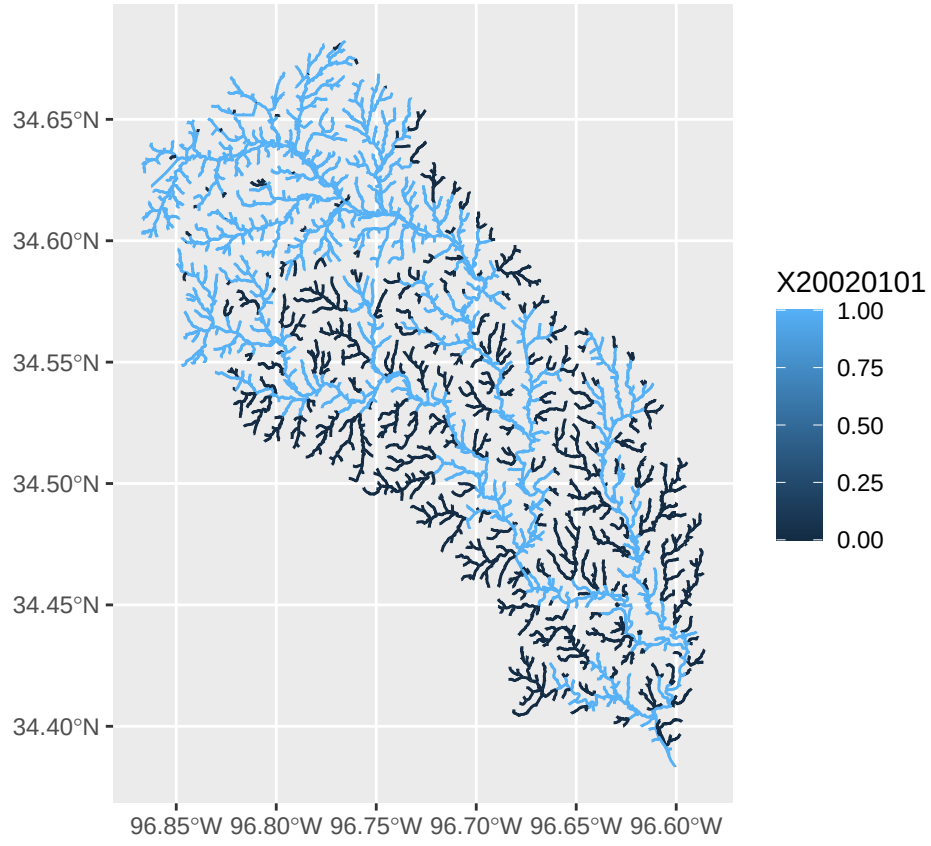
```
ggplot() + geom_sf(data = current_sp, aes(color = X20020101))
```



Note that the legend of this map implies that we have a continuous range of values from 0 to 1. We know this is not true because we converted the dataset so that it contains only 0s and 1s (dry or wet segments). There are ways to change the legend to reflect this. For simplicity, we will leave the legend as is for now.

We can make a similar map, this time using the *future_sp* dataset.

```
ggplot() + geom_sf(data = future_sp, aes(color = X20020101))
```



Exercise 2: On your own, visualize stream drying.

- Make 4 maps for the current climate scenario and 4 maps for the future climate scenario.
  - To accomplish this, replace 'X20020101' in the above code with other column names to visualize other days. Column names are structured as X$YYYYMMDD$ where Y represents year, M represents month, and D represents day.
  - Consider using your daily stream flow plots to select days to visualize. For example, you could visualize the days with high stream flow and low stream flow for the current and future periods. Or, consider visualizing consecutive days and looking for patterns over time.
- What patterns do you observe? Do you notice any similarities or differences between current and future climate scenarios?

Part A3. Measure Stream Drying

While we can visually observe patterns, it can be useful to use quantitative metrics to help us better understand our data. In this section we will calculate the number of dry stream segments and percent wet length of the watershed for each day.

We will start by calculating metrics for one day, January 1, 2002.

One metric we can use is the number of dry segments that occur on this day. To calculate this, first we create a new dataset that contains only the dry stream segments. We can than count how many stream segments are in this new dataset to tell us how many dry stream segments there are on this day.

```
# Current climate, January 1, 2002

#create new dataset containing only dry (value = 0) stream segments from 01/01/02
dry <- subset(current_sp, X20020101 == 0)
dry_count <- nrow(dry) #count number of segments are in this new data frame
dry_count
```

```
## [1] 1186
```

We can see there are 1186 dry segments on this day under current climate conditions.

A second metric we can use is the percent of the watershed that is wet, which we will call 'wet length'. To do this, we start by calculating the total length of all the stream segments in this dataset.

```
total_length <- sum(st_length(current_sp)) #calculate total length of the stream segments
```

Repeat the same process of subsetting the dataset and calculate the total length for wet stream segments.

```
#create new dataset containing only wet (value = 1) stream segments
wet <- subset(current_sp, X20020101 == 1)
wet_length <- sum(st_length(wet)) #calculate total length of the wet segments
```

Finally, calculate the percent wet length of the watershed.

```
pct_wet <- wet_length / total_length *100

pct_wet
```

```
## 53.09771 [1]
```

The wet length of the watershed is 53.1% on this day for the current climate scenario.

Calculate these metrics for the same day of the future climate scenario.

```
# Future climate, January 1, 2002

#number of dry segments
#subset dataset to include only dry segments
dry <- subset(future_sp, X20020101 == 0)
dry_count <- nrow(dry) #count number of dry stream segments

#wet length
#subset dataset to include only wet segments
wet <- subset(future_sp, X20020101 == 1)

#calculate the length of the wet segments
wet_length <- sum(st_length(wet))
```

```r
#calculate percent wet length of watershed
pct_wet <- wet_length / total_length * 100

#note that the total length of the watershed doesn't change,
#so we don't need to recalculate that value
```

Print values:

```r
dry_count
```

```
## [1] 1050
```

```r
pct_wet
```

```
## 58.69479 [1]
```

There are 1050 dry stream segments and the percent wet length of the watershed is 58.7% on this day under future climate conditions.

This would take a long time to repeat for each day in our dataset. We can automate these calculations using a *loop*. A loop is a common programming tool that allows us to apply the same code repeatedly. In this case, we apply the same code to calculate the number of dry stream segments and percent wet length for every day in the dataset.

```r
#current
dry_segments_current <- c() #create a place to store number of dry segments
pct_wet_current <- c() #create a place to store percent wet length values

#This is where the loop starts.
#We start by telling it to run the code inside the loop on columns 5 through 734
#column 5 is where the stream flow data begins. Columns 1-4 contain the shapefile information.
#Recall, you can use the head() function to take a look at the column names.
for (i in 5:734){ #everything between the curly brackets {} is "inside" the loop
  day_i <- subset(current_sp[,i]) #subset a single column, i (a single day)

  dry_i <- subset(day_i, day_i[[1]] == 0) #subset only dry segments of selected column
  dry_i_count <- nrow(dry_i) #count number of dry segments

  wet_i <- subset(day_i, day_i[[1]] == 1) #subset only wet segments of selected column
  wet_i_length <- sum(st_length(wet_i)) #calculate total length of wet segments

  pct_wet_i <- wet_i_length/total_length * 100 #calculate percent wet length

  dry_segments_current <- c(dry_segments_current, dry_i_count) #store number of dry segments values
  pct_wet_current <- c(pct_wet_current, pct_wet_i) #store percent wet length values
} #end of loop
```

In this code, we're applying (or looping) the same code to columns 5 through 734 of our dataset, which means we're calculating these metrics for every day in our dataset.

Repeat for the future climate scenario.

```r
#future
dry_segments_future <- c() #create empty vector which we will use to store number of dry segments
pct_wet_future <- c() #create empty vector which we will use to store percent wet length values

for (i in 5:734){
  day_i <- subset(future_sp[,i]) #subset a single column (a single day)

  dry_i <- subset(day_i, day_i[[1]] == 0) #subset only dry segments of selected column
  dry_i_count <- nrow(dry_i) #count number of dry segments

  wet_i <- subset(day_i, day_i[[1]] == 1) #subset only wet segments of selected column
  wet_i_length <- sum(st_length(wet_i)) #calculate total length of wet segments

  pct_wet_i <- wet_i_length/total_length * 100 #calculate percent wet length

  dry_segments_future <- c(dry_segments_future, dry_i_count) #store number of dry segments values
  pct_wet_future <- c(pct_wet_future, pct_wet_i) #store percent wet length values
}
```

dry_segments_current contains the number of dry stream segments for each day of the current climate scenario dataset.

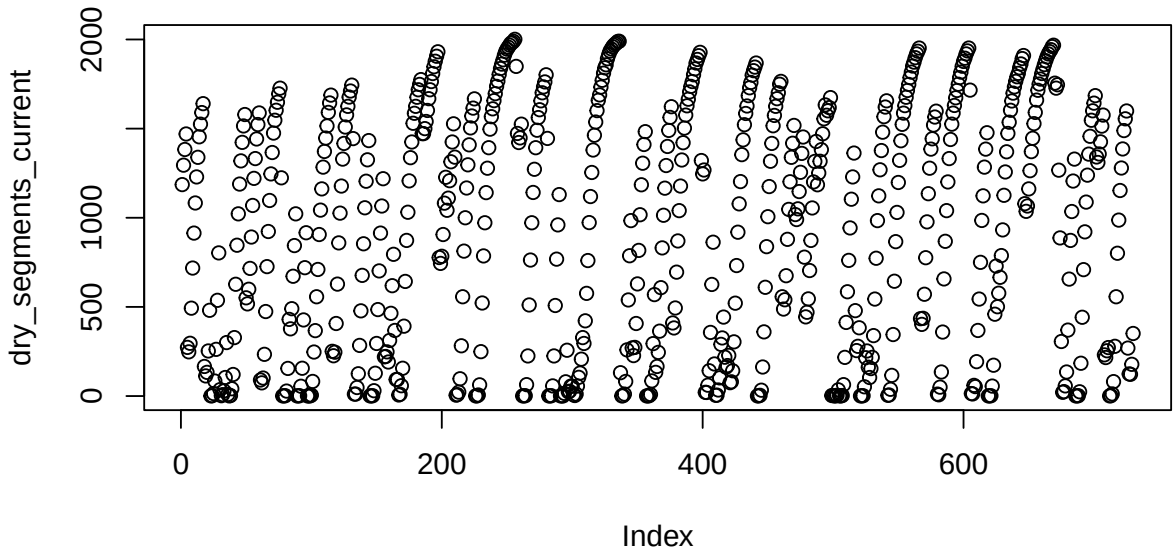
dry_segments_future contains the number of dry stream segments for each day of the future climate scenario dataset.

pct_wet_current contains the percent wet length values for each day of the current climate scenario dataset.

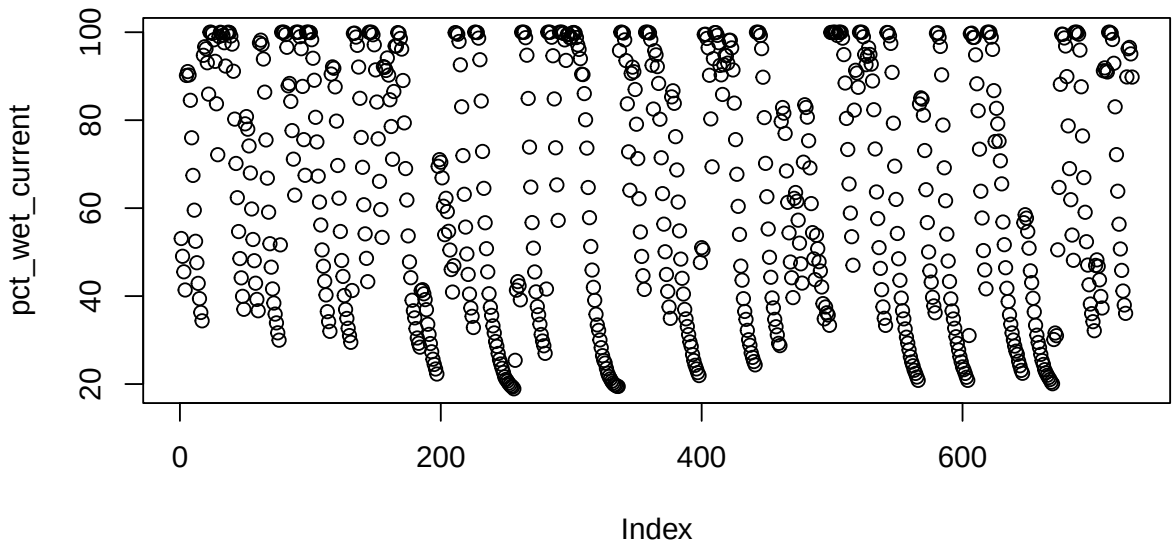
pct_wet_future contains the percent wet length values for each day of the future climate scenario dataset.

We can plot our calculations:

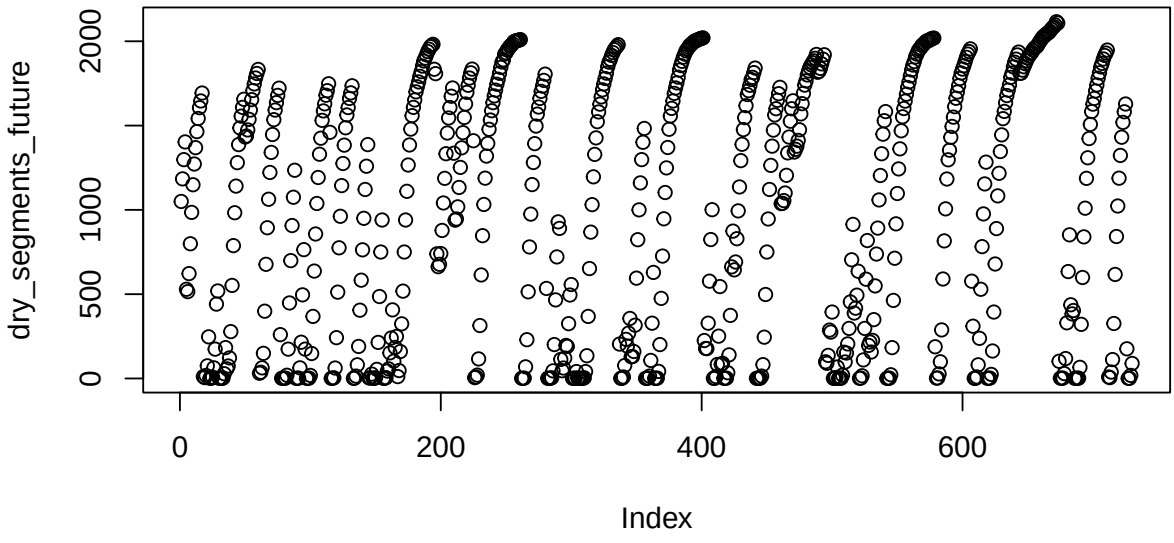
```r
plot(dry_segments_current)
```



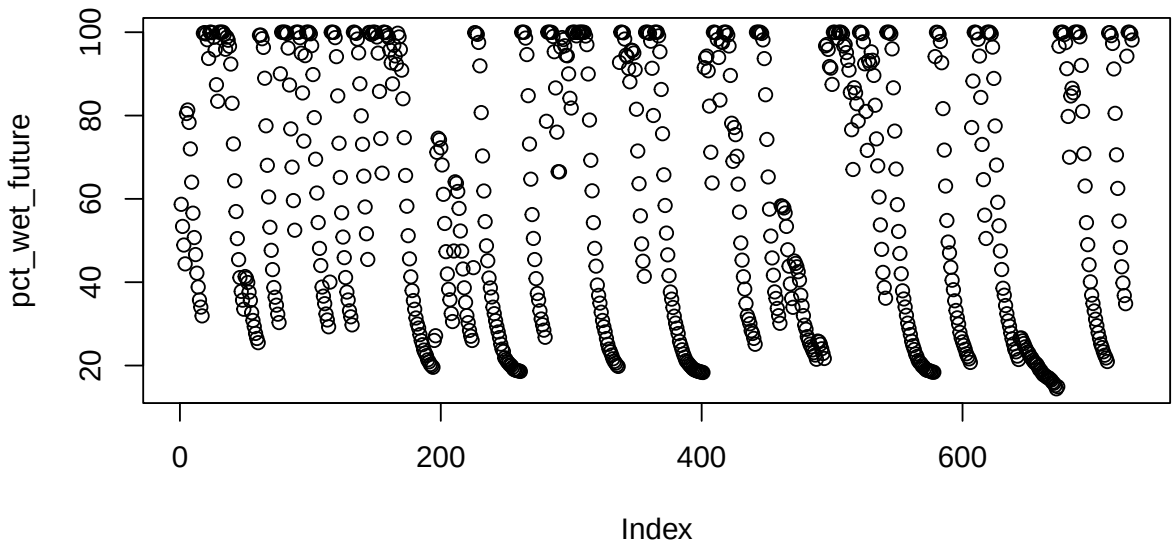
```
plot(pct_wet_current)
```




```
plot(dry_segments_future)
```



```
plot(pct_wet_future)
```



In these plots, the x-axes, labeled "Index", represent time at a daily timestep. The point at the Index value of 1 is the first day of this dataset, January 1, 2002. The point at the Index value of 366 is the 366th day of this dataset, January 1, 2003. The y-axes are the metrics we calculated.

Exercise 3: Compare number of dry segments and percent wet length for the current and future climate scenarios

Use the following code to calculate mean and standard deviation for each metric (number of dry segments and percent wet length) and climate scenario (current and future).

```
mean(dry_segments_current)
sd(dry_segments_current)

mean(pct_wet_current)
sd(pct_wet_current)

mean(dry_segments_future)
sd(dry_segments_future)

mean(pct_wet_future)
sd(pct_wet_future)
```

Which climate scenario tends to have more dry segments? Which tends to have a higher percent wet length?

Follow-up Questions

Answer these questions in your separate document.

1. Based on these findings, does stream drying increase or decrease increase or decrease in the Blue River watershed in the future?
2. Number of dry stream fragments and percent wet length can be useful metrics, but they are only two ways to measure stream drying. Can you think of something we could measure using this dataset in addition to number of dry stream fragments and percent wet length that could be useful?
