Objects, Loops, and Linear Algebra

Day 1 of the RII workshop, summer 2025

Austin Cutler

FSU

What is *this* workshop?

- This workshop is a course on using R to prepare you for MLE, causal inference, and the rest of your lives as political scientists
- By the end of this workshop, you should be at least familiar with:
  - Nested `for` loops
  - Simulating data
  - Sampling
  - Creating functions
  - Optimization
  - And more! (?)

Ground Rules for the Class

- Use Stack Exchange
- Use the help function in Rstudio
- Please don’t use ChatGpt…yet
- Keep this workshop a safe and collaborative environment:
  - Ask questions even if you think they’re bad questions
  - *Do not* skip ahead if you finish before everyone else
    - Help each other if you run into problems
    - Related point, don’t work on other things during the workshop
  - Do not make anybody feel bad if they don’t know as much on a topic as you

Ice breakers

- What's your name?
  - *Austin Cutler*
- What's one fun thing you did over the summer?
  - *I went to New York for a wedding and visited my friend in Albany*
- What's one non-academic thing you're hoping to do in the next year (or so)?
  - *Coach one of my hurdlers to win the Florida state championships (or at least make it to the finals)*

Course website

Intro Quiz

- Below is a qr code for a short google form
- The purpose of the form is for you to let me know what things you want covered in the workshop
- Please note that I have a good bit of content that the methods faculty have specifically asked for, but if there is extra time during teh 4th session, I will try to include the content you all want covered

Brief review of objects

- There are several different types of objects in R.
- Beyond an object's *class* (i.e., `numeric`, `character`, `factor`, etc.), how the data is actually structured is going to determine how we can use the data
- The categories of object are primarily `vectors`, `lists`, `matrices`, and `data frames`

Vectors

- Each of the following is a vector:

```
vec_1 <- 2
vec_1
```

```
[1] 2
```

```
vec_2 <- "beta"
vec_2
```

```
[1] "beta"
```

```
vec_3 <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
vec_3
```

```
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
vec_4 <- c("alpha", "beta", "gamma", "delta", "epsilon")
vec_4
```

```
[1] "alpha"   "beta"    "gamma"   "delta"   "epsilon"
```

- We can use the `as.numeric()`, `as.character()`, `as.double()`, and `as.factor()` functions to force the vector to be a certain type of object

Getting the data out

- There are times where we'll want to get specific data out of a vector, we can use brackets and the data's index position

```
vec_3[11]
```

```
[1] 10
```

```
vec_4[3]
```

```
[1] "gamma"
```

- We can also use this syntax to filter observations based on logical expressions (similar to the `filter()` function)

```
##all values less than 6
vec_3[vec_3 < 6]
```

```
[1] 0 1 2 3 4 5
```

```
##all values less than 9 and greater than 2
vec_3[vec_3 < 9 & vec_3 > 2]
```

```
[1] 3 4 5 6 7 8
```

Practice

Exercise

Individually, construct both a character and numeric vector. Practice extracting data from both types.

Matrices

- Matrices are another way to store data
- Some tasks in R (such as matching and estimating some quantities of interest) need our data to be in a `matrix`
- We can create a matrix using the `matrix()` function in R

```
1  # the identity matrix
2  id_mat_data <- c(1,0,0,0,1,0,0,0,1)
3  mat_1 <- matrix(id_mat_data, nrow = 3, ncol = 3)
4  mat_1
```

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Matrices

- We can also merge several `vectors` manually

```r
# the identity matrix "the old fashioned way"
id_1 <- c(1,0,0)
id_2 <- c(0,1,0)
id_3 <- c(0,0,1)
mat_2 <- cbind(id_1, id_2, id_3)

mat_2
```

```
     id_1 id_2 id_3
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Matrices

- Or by converting a `dataframe` into a `matrix` using `as.matrix`

```
#note that this is how you can make a dataframe
dat_2 <- data.frame(id_1,id_2,id_3)

as.matrix(dat_2)
```

```
     id_1 id_2 id_3
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Getting Data out of Matrices

- Like with `vectors` (and with `lists`) we can use `[]`'s to extract data
  - We can use `,` to distinguish between rows and columns

```
1 #an entire row
2 mat_2[1,]
```

```
id_1 id_2 id_3
   1    0    0
```

```
1 #an entire column
2 mat_2[,1]
```

```
[1] 1 0 0
```

- We can provide additional information to get a specific point

```
1 mat_2[1,1]
```

```
id_1
   1
```

Exercise

Individually, construct a 4x4 matrix (or larger) – that is not the identity matrix. Practice extracting a row, a column, and a single data point from the matrix you created. (You can also practice naming rows and columns if you want to.)

Linear Algebra

- Believe it or not, we're going to need to do some math when doing research
- Sometimes, especially when implementing a new statistical approach that doesn't have software support, it will be useful to be able to do the math "by hand"
- The main functions you will need when manually doing linear algebra are `%*%`, `t()`, and `solve()`
  - `%*%` matrix multiplication
  - `t()` transpose
  - `solve()` take the inverse of any invertible `matrix`

Using Linear Algebra

- In groups, using the `USJudgeRatings` dataset, predict a judge's "worthy of retention" rating (`RTEN`)
- Remember that we can estimate regression coefficients with linear algebra as follows

$$\beta = (X^T X)^{-1} X^T y$$

  - X is a matrix of independent variables and a column of 1s for the intercept
  - T is the transpose of a matrix
  - -1 is taking the inverse
  - Compare the "by hand" result to the coefficients using the `lm()` function

Using Linear Algebra

- In groups, using the `USJudgeRatings` dataset, predict a judge's "worthy of retention" rating (`RTEN`)
- Pseudocode:
  - *Separate out the outcome variable from the independent variables (i.e., make X and y to separate objects in R).*
  - *Transform objects to their correct type (i.e., matrices or vectors)*
  - *BEFORE doing linear algebra.*
  - *Use the rep() function to create a vector of 1s*
  - *Add this vector to the matrix of independent variables using the cbind() function*
  - *Do matrix algebra*

Lists and Dataframes

- `lists` are another way of storing data
- Lists can be used to store multiple types of other object types while retaining their original class
  - i.e., a list of `dataframes`, or `vectors`, or `lists`
- `Dataframes` (and/or `tibbles`) are the most commonly used data structure in r
- Columns in `dataframes` can be selected using the `$` function or with the `select` function
- `select` is slightly easier, but `$` is more flexible because it can be used on lists as well

Lists and Dataframes (continued)

- We can also extract data from dataframes using `[]`, just like with matrices and vectors
- We can *also* using `select()` and `filter()` and `subset()` for this if we have the right packages loaded
- Using `[]` is especially beneficial for instances where we're looping through data
- With lists, extracting data can be done using `[]`, however, these may need to be nested further (`[[]]`) due to how complex lists can get
- In addition to being useful for storing data from loops, objects created from `lm()` or `glm()` are lists as well

Practice with dataframes and lists

- Individually, extract data from the `USJudgeRatings` data using complex logical conditions
  - Make two new dataframes using these complex logical conditions (at least two conditions per statement)
- Store these new dataframes into a list
- Call each of these dataframes individually from the list
- Repeat the above steps, creating two *new* dataframes and a new list, combine each of the lists into a new list
- Call each dataframe individually from the new list
  - Hint: for this step you will need to use double brackets, multiple pairs (i.e., `[[#]][#]`)

For Loops

For Loops Explained

- Now that we're all experts on using lists, it's time to talk through using for loops
- There are plenty of instances where you will want to iterate through a specific task repeatedly, and we can use `for` loops to do so
- The general structure of a `for` loop is as follows:

```
results <- container_for_results

for (variable in vector) {
  function to perform(vector[variable]) -> results[[variable]]
}
```

For Loop Example

- Remember the vector `vec_4` from earlier, if we want to print each observation individually, we can do the following

```
for (j in vec_4) {
print(j)
}
```

```
[1] "alpha"
[1] "beta"
[1] "gamma"
[1] "delta"
[1] "epsilon"
```

- OR

```
for (j in 1:length(vec_4)) {
print(vec_4[j])
}
```

```
[1] "alpha"
[1] "beta"
[1] "gamma"
```

Even More Loops

- While the last example may not be too practical, it shows the logic being how for loops work
- A more practical application might be to repeatedly do some calculation
- We can make an empty vector, and stor our results in it while looping through like so

```
# calculations within a for loop
out <- rep(NA, 5)

out
```

```
[1] NA NA NA NA NA
```

```
for (i in 1:length(out)) {
out[i] <- ((i^2) + i)
}

out
```

```
[1]  2  6 12 20 30
```

Nesting

- For loops can also be nested

```
 1  results <- c()
 2
 3  results_n <- c()
 4
 5  for (i in vec_3) {
 6    2+vec_3[i] -> results[i]
 7
 8    for(j in vec_3){
 9      results[j]/length(vec_3)*17 -> results_n[j]
10    }
11  }
```

Nesting

```
1 data.frame(results, results_n)
```

```
   results results_n
1        2  3.090909
2        3  4.636364
3        4  6.181818
4        5  7.727273
5        6  9.272727
6        7 10.818182
7        8 12.363636
8        9 13.909091
9       10 15.454545
10      11 17.000000
```

And now your loops

- Individually, create a for loop that prints out the letters of the alphabet one at a time
- Once you're all done, in pairs, run a for loop estimating the mean fo each column in the `USJudgeRatings` dataframe, storing that mean in a vector.
  - Store the name of the variables you're taking the mean of in a separate vector as well. After the loop is done, create a new dataframe with one column named `variables` and one named `means` with the variable names and their means
- Repeat the previous two steps, instead taking the maximum of each variable and making a new dataframe with `maximums` instead
- Lastly, *as a whole group*, for all of the quantities of interest (means and maximums) use the bootstrap to create 95% confidence intervals

Bootstrap Refresher

- To bootstrap, you:
  - Sample your data *with replacement*
  - Estimate your quantity of interest
  - Repeat this ~1,000-10,000 times
  - Store your results for each iteration
  - Take the 2.5% and 97.5% quantiles