NC Voter File Intro

DPI 610

Week 1

January 24, 2021

This exercise is designed to give you more experience with some of the basic functionality of **R** while also familiarizing yourself with a data set that we will be using in our upcoming problem set.

Load data

First, we use the function `read.csv()` to load external data onto the **R** environment. For this project, let's load in some (synthetic) NC voter file data from a vendor (named `nc.csv`), along with a dictionary file that describes the variables (named `nc_dictionary.csv`).

In **R**, we create a new object in the environment by using the `<-` symbol followed by a value, vector or data.

```r
# load data using read.csv() function,
#  and assign the data to a new object named nc_data
nc_data <- read.csv("nc.csv")
```

Question 1

Following the above code, load the dataset `nc_disctory.csv` onto the environment, and assign the data to a new object named `dictionary`.

Answer 1

```r
#Insert answer here
```

Access Variables in Data Frames

Typically, a dataset consists of multiple variables, and each column corresponds to a variable. In RStudio, we can view the dataset by using the `View()` function.

```r
# View command allows you to "view" dataset in RStudio
View(nc_data)
```

We can access individual variables by using the `$` symbol. For example, to access a variable named `party` in `nc_data`, we type `nc_data$party`.

```r
# use head() to show just first five elements
head(nc_data$party)
```

Question 2

View dataset `dictionary` using `View()` function, and see how many variables are included in the dataset.

Answer 2

```
# Insert answer here
```

Question 3

Access the variable named `age` in dataset `nc_data` using `$`, and show the first five elements of the variable.

Answer 3

```
# Insert answer here
```

More on Functions

Many commands in R take the form of functions. For example, we just used a function `read.csv()` to load the data, and we used `head()` to print out the first five elements. A function takes an input (or inputs) and produces some output. **R** has many built in functions. Packages will add more functions. And we can also write our own functions. Functions save time since we can write one function and then call it multiple times, rather than writing the same code over and over.

Some uses of functions:

- Create objects
- Complex calculations
- Statistical models
- Graphs & Maps
- Generate output (e.g. tables)

Some useful starter functions:

- `c()`
- `length()` & `dim()`
- `seq()` & `rep()`
- `sum()`, `mean()`
- `min()`, `max()`
- `nchar()`, `length()`

```
# Add numbers
sum(1, 2, 4, 5)
```

```
## [1] 12
```

```
# Create a sequence of numbers from 1 to 10
seq(1, 10)
```

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

```
# Create a sequence of numbers from 0 to 10 by 2
seq(0, 10, 2)
```

```
## [1]  0  2  4  6  8 10
```

```
# Create a vector of characters
c("A", "B", "C") # Create a list or vector
```

```
## [1] "A" "B" "C"
```

Question 4

The data `nc_data` includes a variable `age`. Use `max()` function to determine the oldest person in the data set? What is their age?

Answer 4

```
# Insert answer here
```

Question 5

How many people are over the age of 90 in the data set?

Answer 5

```
# Insert answer here
```

Working with Data Frames by Indexing

In base **R**, there are several keys to working with data frames.

Any cell in a data frame can be accessed by row / column. For example, if we wanted to return the value in the third row and 10th column of the NC data set, we would type: `nc_data[3,10]`.

This is very useful. We can also use logical statements to subset the data frame by row or column. For example, if we wanted to create a new data set that included rows only for those people over 90, we could do the following:

```
old_people <- nc_data[nc_data$age > 90, ]
```

Alternatively, we can use the `subset` function, which does the same thing but doesn't require us to refer to the name of the data set more than once.

```
old_people <- subset(nc_data, age > 90)
```

To focus on a narrower set of columns, we can call the columns by number or by name. Suppose we wanted a data set consisting only of a voter's id number and their age. These are the first and third columns in the data.

```
id_ages <- nc_data[, c(1,3)]
head(id_ages)
```

```
##      id age
## 1 36974  30
## 2 37362  58
## 3 36635  65
## 4 38352  65
## 5 36619  52
## 6 37731  40
```

Or, using a vector of names (written as strings):

```r
id_ages <- nc_data[, c("id","age")]
head(id_ages)
```

```
##      id age
## 1 36974  30
## 2 37362  58
## 3 36635  65
## 4 38352  65
## 5 36619  52
## 6 37731  40
```

We also might want to sort a vector or even reorder a data set. A useful function for this is `sort()`, which takes a vector and reorders it in increasing order.

Question 6

Can you sort the ages of the first 10 people in the data set in order of increasing magnitude?

Answer 6

```r
#Insert answer here
```

To perform a similar operation on a data frame you can use the function `order()`. Say we wanted to reorder the NC data so that it is in order of increasing age.

```r
nc_data <- nc_data[order(nc_data$age),]
```

Modifying Data Frames and Variables

Note that we saved the reordered data frame to an object with the same name. We essentially just overwrote the old version of the data. If we had given it a different name, we would have saved the reordered data frame as a new object.

You can do something similar with variables. For example, we can create a new variable named `age_std` in the data `nc_data`, which subtracts the average age from individual age:

```r
# create a new variable age_std
#  (1) access age vairable by nc_data$age
#  (2) subtract average age by - mean(nc_data$age)
#  (3) assign the result by "<-"
nc_data$age_std <- nc_data$age - mean(nc_data$age)
```

Question 7

Suppose we wanted to create a new variable that counted the number of times people had voted between 2008 and 2014. These variables are captured as `voted2008`, `voted2010`, `voted2012`, and `voted2014`.

Can you create a new variable in the `nc_data` data set called `vote_history` that reflects the number of times a person voted in elections between 2008 and 2014?

Answer 7

```
# Insert answer here
```

Summarizing Data by Groups

There are many different ways to summarize data in R using both base R and external packages. To start with, we will use the `aggregate()` function from base R. To use aggregate we write a variable we want to perform an operation upon, followed by a tilde (`~`) and then the variable or variables we would like to group by. Finally, we write the function that we want to apply to the variable.

An example will make this syntax clearer. Suppose we wanted to see what average turnout was for 2008 by gender. We would write the following:

```
aggregate(voted2008 ~ gender, data = nc_data, FUN = mean)
```

```
##   gender voted2008
## 1      F 0.7862662
## 2      M 0.7487256
```

Question 8

Now determine average turnout in 2008 by gender *and* party. Which gender/party combination has the highest turnout rate?

Answer 8

```
# Insert answer here
```

Introducing user-written functions

You've now used several functions that come as part of R. Now let's write one of our own functions.

Let's suppose we wanted to transform a variable by subtracting its mean and dividing by its standard deviation. Let's write a function to do that.

```
dev_from_mean <- function(x){
  # subtract mean, divide by sd
  y <- (x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)
  # return result
  return(y)
}
```

The argument `na.rm=TRUE` means that, when we are applying the mean and sd functions, we want to remove any missing values.

```r
# mean without na.rm = TRUE
mean(c(1, 2, 5, NA))
```

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

```r
# with na.rm = TRUE
mean(c(1, 2, 5, NA), na.rm = TRUE)
```

```
## [1] 2.666667
```

Question 9

Can you apply this function to the age variable in our data set, and add the result in a new variable called `age_std` in `nc_data`.

Answer 9

```r
# Insert answer here
```