Lab 5: Dplyr, Pipes, and More

Statistical Computing, 36-350

Week of Tuesday September 27, 2022

Name:
Andrew ID:
Collaborated with:

This lab is to be done in class (completed outside of class time if need be). You can collaborate with your classmates, but you must identify their names above, and you must submit **your own** lab as an knitted PDF file on Gradescope, by Friday 9pm, this week.

This week's agenda: mastering the pipe operator `%>%`, practicing `dplyr` verbs, and pivoting using `tidyr`.

Loading the tidyverse

Now we'll load the tidyverse suite of packages. (You should already have `tidyverse` installed from the last lab; but if for some reason you still need to install again, then you can just look back at the last lab's instructions.) This gives us access to the pipe operator `%>%` as well as the `dplyr` and `tidyr` packages needed to complete this lab.

Post edit. Loading the `tidyverse` package in its entirety includes `plyr`, and this can cause namespace issues with the `dplyr` package. Better to just load only what you need.

```
library(dplyr)
library(tidyr)
library(purrr)
```

Q1. Pipes to base R

For each of the following code blocks, which are written with pipes, write equivalent code in base R (to do the same thing).

- **1a.**

```
letters %>%
  toupper %>%
  paste(collapse="+")
```

```
## [1] "A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z"
```

```
# YOUR CODE GOES HERE
```

- **1b.**

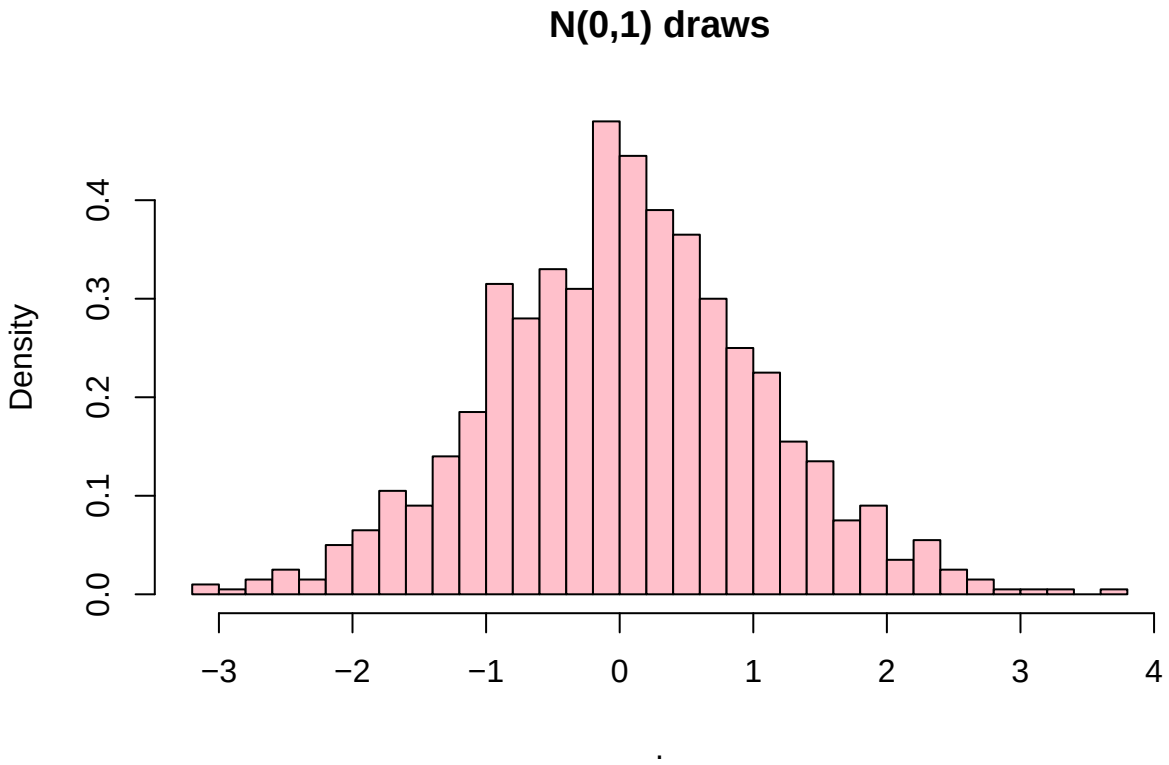
```
"     Ceci n'est pas une pipe     " %>%
  gsub("une", "un", .) %>%
  trimws
```

```
## [1] "Ceci n'est pas un pipe"
```

```
# YOUR CODE GOES HERE
```

- **1c.**

```
rnorm(1000) %>%
  hist(breaks=30, main="N(0,1) draws", col="pink", prob=TRUE)
```



.

```
# YOUR CODE GOES HERE
```

- **1d.**

```
rnorm(1000) %>%
  hist(breaks=30, plot=FALSE) %>%
  `[[`("density") %>%
  max
```

```
## [1] 0.43
```

```
# YOUR CODE GOES HERE
```

Q2. Base R to pipes

For each of the following code blocks, which are written in base R, write equivalent code with pipes (to do the same thing).

- **2a.** Hint: you'll have to use the dot `.`, as seen above in Q1b, or in the lecture notes.

```
paste("Your grade is", sample(c("A","B","C","D","R"), size=1))
```

```
## [1] "Your grade is B"
```

```
# YOUR CODE GOES HERE
```

- **2b.** Hint: you can use the dot `.` again, in order to index `state.name` directly in the last pipe command.

```
state.name[which.max(state.x77[,"Illiteracy"])]
```

```
## [1] "Louisiana"
```

```
# YOUR CODE GOES HERE
```

- **2c.** Note: `str.url` is defined for use in this and the next question; you can simply refer to it in your solution code (it is not part of the code you have to convert to pipes).

```
str.url = "https://www.stat.cmu.edu/~arinaldo/Teaching/36350/F22/data/king.txt"

lines = readLines(str.url)
text = paste(lines, collapse=" ")
words = strsplit(text, split="[[:space:]]|[[:punct:]]")[[1]]
wordtab = table(words)
wordtab = sort(wordtab, decreasing=TRUE)
head(wordtab, 10)
```

```
## words
##        of  the   to  and    a   be will that   is
##  203   98   98   58   40   37   32   25   24   23
```

```
# YOUR CODE GOES HERE
```

- **2d.** Hint: the only difference between this and the last part is the line `words = words[words != ""]`. This is a bit tricky line to do with pipes: use the dot `.`, once more, and manipulate it as if were a variable name.

```
lines = readLines(str.url)
text = paste(lines, collapse=" ")
words = strsplit(text, split="[[:space:]]|[[:punct:]]")[[1]]
words = words[words != ""]
wordtab = table(words)
wordtab = sort(wordtab, decreasing=TRUE)
head(wordtab, 10)
```

```
## words
##   of  the   to  and    a   be will that   is   we
##   98   98   58   40   37   32   25   24   23   21
```

```
# YOUR CODE GOES HERE
```

Prostate cancer data set

Below we read in the prostate cancer data set, as visited in previous labs.

```
pros.df =
  read.table("https://www.stat.cmu.edu/~arinaldo/Teaching/36350/F22/data/pros.dat")
```

Q3. Practice with `dplyr` verbs

In the following, use pipes and `dplyr` verbs to answer questions on `pros.df`.

- **3a.** Among the men whose `lcp` value is equal to the minimum value (across the entire data set), report the range (min and max) of `lpsa`.

```
# YOUR CODE GOES HERE
```

- **3b.** Order the rows by decreasing `age`, then display the rows from men who are older than 70 and without SVI.

```
# YOUR CODE GOES HERE
```

- **3c.** Order the rows by decreasing `age`, then decreasing `lpsa` score, and display the rows from men who are older than 70 and without SVI, but only the `age`, `lpsa`, `lcavol`, and `lweight` columns. Hint: `arrange()` can take two arguments, and the order you pass in them specifies the priority.

```
# YOUR CODE GOES HERE
```

- **3d.** We're going to resolve Q2c from Lab 3 using the tidyverse. Using `purrr` and `dplyr`, perform t-tests for each variable in the data set, between SVI and non-SVI groups. To be precise, you will perform a t-test for each column excluding the SVI variable itself, by running the function `t.test.by.ind()` below (which is just as in Q2c in Lab 3). Print the returned t-test objects out to the console.

```
t.test.by.ind = function(x, ind) {
  stopifnot(all(ind %in% c(0, 1)))
  return(t.test(x[ind == 0], x[ind == 1]))
}
```

```
# YOUR CODE GOES HERE
```

- **3e.** Extend your code from the last part (append just one more line of code, glued together by a pipe) to extract the p-values from each of the returned t-test objects, and print them out to the console.

```
# YOUR CODE GOES HERE
```

Fastest 100m sprint times

Below, we read in two data sets of the 1000 fastest times ever recorded for the 100m sprint, in men's and women's track, as seen in the last lab.

```
sprint.m.df = read.table(
  file="https://www.stat.cmu.edu/~arinaldo/Teaching/36350/F22/data/sprint.m.txt",
  sep="\t", quote="", header=TRUE)
sprint.w.df = read.table(
  file="https://www.stat.cmu.edu/~arinaldo/Teaching/36350/F22/data/sprint.w.txt",
  sep="\t", quote="", header=TRUE)
```

Q4. More practice with `dplyr` verbs

In the following, use pipes and `dplyr` verbs to answer questions on `sprint.w.df`.

- **4a.** Order the rows by increasing `Wind` value, and then display only the women who ran at most 10.7 seconds.

```
# YOUR CODE GOES HERE
```

- **4b.** Order the rows by terms of increasing `Time`, then increasing `Wind`, and again display only the women who ran at most 10.7 seconds, but only the `Time`, `Wind`, `Name`, and `Date` columns.

```
# YOUR CODE GOES HERE
```

- **4c.** Plot the `Time` versus `Wind` columns, but only using data where `Wind` values that are nonpositive. Hint: note that for a data frame, `df` with columns `colX` and `colY`, you can use `plot(colY ~ colX, data=df)`, to plot `df$colY` (y-axis) versus `df$colX` (x-axis).

```
# YOUR CODE GOES HERE
```

- **4d.** Extend your code from the last part (append just two more lines of code, glued together by a pipe) to plot the single fastest `Time` per `Wind` value. (That is, your plot should be as in the last part, but among points that share the same x value, only the point with the lowest y value should be drawn.)

```
# YOUR CODE GOES HERE
```

Q5. Practice pivoting wider and longer

In the following, use pipes and `dplyr` and `tidyr` verbs to answer questions on `sprint.m.df`. In some parts, it might make more sense to use direct indexing, and that's perfectly fine.

- **5a.** Confirm that the `Time` column is stored as character data type. Why do you think this is? Convert the `Time` column to numeric. Hint: after converting to numeric, there will be `NA` values; look at the position of one such `NA` value and revisit the original `Time` column to see why it was stored as character type in the first place.

```
# YOUR CODE GOES HERE
```

- **5b.** Define a reduced data frame `dat.reduced` as follows. For each athlete, and each city, keep the fastest of all times they recorded in this city. Then drop all rows with an `NA` value in the `Time` column Your new data frame `dat.reduced` should have 600 rows and 3 columns (`Name`, `City`, `Time`). Confirm that it has these dimensions, and display its first 10 rows. Hint: `drop_na()` in the `tidyr` package allows you to drop rows based on `NA` values.

```
# YOUR CODE GOES HERE
```

- **5c.** The data frame `dat.reduced` is said to be in "long" format: it has observations on the rows, and variables (`Name`, `City`, `Time`) on the columns. Arrange the rows alphebtically by city; convert this data frame into "wide" format; and then order the rows so that they are alphabetical by sprinter name. Call the result `dat.wide`. To be clear, here the first column should be the athlete names, and the remaining columns should correspond to the cities. Confirm that your data frame has dimension 141 x 152 Do these dimensions make sense to you?

```
# YOUR CODE GOES HERE
```

- **5d.** Not counting the names in the first column, how many non-`NA` values does `dat.wide` have? How could you have guessed this number ahead of time, directly from `dat.reduced` (before any pivoting at all)?

```
# YOUR CODE GOES HERE
```

- **5e.** From `dat.wide`, look at the row for "Usain Bolt", and determine the city names that do not have `NA` values. These should be the cities in which he raced. Determine these cities directly from `dat.reduced`, and confirm that they match.

```
# YOUR CODE GOES HERE
```

- **5f.** Convert `dat.wide` back into "long" format, and call the result `dat.long`. Remove rows that have `NA` values (hint: you can do this by setting `values_drop_na = TRUE` in the call to the pivoting function), and order the rows alphabetically by athlete and city name. Once you've done this, `dat.long` should have matching entries to `dat.reduced`; confirm that this is the case.

```
# YOUR CODE GOES HERE
```