Useful R Packages and Functions

For EEMB 148 assignments

This document contains useful starter code for who would like to code their own analyses. The necessary data for the homework assignments is provided in CSV format on Gauchospace. Feel free to bring coding questions to office hours!

Summary

These packages and functions will be necessary for creating the required statistical output:

- In package **`stats`** (*base R*)
  - `t.test`, `aov`, `summary`, `TukeyHSD`
- In package **`agricolae`**
  - `HSD.test`

These packages and functions within the **`tidyverse`** will be incredibly helpful:

- In package **`readr`**
  - `read_csv`
- In package **`ggplot2`**
  - `ggplot`
- In package **`dplyr`**
  - `filter`, `select`, `group_by`, `summarize`
- In package **`magrittr`**
  - `%>%`

This document is written using the following generic conventions. In your code, you should replace these words with the appropriate object:

- `DATAFRAME` = the data frame from the imported CSV
- `PREDICTOR` = the predictor variable
- `RESPONSE` = the response variable
- `VAR` = some variable
- `VALUE` = some value

Necessary statistical code

t-test

The `t-test` function is included in the base R package `stats`. The function accepts two versions of syntax.

```r
# Version 1
t.test(DATAFRAME$RESPONSE ~ DATAFRAME$PREDICTOR)

# Version 2
t.test(RESPONSE ~ PREDICTOR,
       data = DATAFRAME)
```

ANOVA

Functions for running ANOVA are also included in the `stats` package. In order to view the results, you need to first create an ANOVA object using the `aov` function, then print the results using the `summary` function.

```r
# Create the ANOVA output
anova_output <- aov(RESPONSE ~ PREDICTOR,
                         data = DATAFRAME)

# View the output summary
summary(anova_output)
```

Tukey test

There are several packages you can use to compute Tukey test output. For this class, the most useful is the `HSD.test` function in the package `agricolae`. `TukeyHSD` is a function in `stats` that might also be useful. Both of these functions require that you previously create an ANOVA output using the `aov` function. Notice that the predictor variable in these functions is surrounded by quotes.

```r
# Load necessary package
library(agricolae)

# Create the Tukey output
tukey_output <- HSD.test(anova_output, "PREDICTOR")

# View the groupings from the Tukey output
tukey_output$groups

# Calculate p-values for each pairwise comparison
TukeyHSD(anova_output, "PREDICTOR")
```

Useful `tidyverse` code

If you're not already using `tidyverse`, I would highly recommend it! `tidyverse` is a collection of intuitive and convenient packages that all deal with data wrangling and visualization. You can install/load the entire collection in one go, or just install/load each package individually. For this class, I especially recommend `readr` (for importing data), `dplyr` (for wrangling data), and `ggplot2` (for visualizing data). The pipe function from `magrittr` is also great for keeping your code nice and organized.

```r
# Load all packages in the tidyverse
library(tidyverse)
```

Import data with `readr`

You can either use the tidyverse function `read_csv` or the base R function `read.csv` to import data. `read_csv` is superior in several ways because its defaults are less likely to create mistakes during import. If you are working in an R project (which I highly recommend), then the filepath is just the name of the CSV. Check out a cheatsheet for `readr` here.

```
# Load the package individually (if you don't load the tidyverse)
library(readr)

# Import data
DATAFRAME <- read_csv('filepath.csv')
```

Wrangle data with `dplyr`

Check out a cheat sheet for dplyr here.

```
# Load the package individually
library(dplyr)

# Filter for rows that match a specific value (can also use operators !=, <, >, etc.)
dataframe2 <- filter(DATAFRAME,
                     VAR == 'VALUE')

# Select for columns with useful variables
dataframe3 <- select(dataframe2,
               VAR1, VAR2, VAR3)

# Group by a predictor variable in order to summarize data later on
dataframe4 <- group_by(dataframe3,
                       PREDICTOR)

# After grouping by the predictor variable, summarize the data using functions
# Useful summary functions for this class are mean() and sd()
dataframe5 <- summarize(dataframe4,
                        mean = mean(RESPONSE),
          sd = sd(RESPONSE))
```

Organize code with `magrittr`

The `tidyverse` also contains a handy tool for organizing code in a much more intuitive way: the pipe function `%>%`. The pipe is a little different than other functions in R; it acts as a connector between phrases of code, translating roughly into "and then. . . " The pipe allows you to string together several functions that you would like to perform on the same data frame, without renaming the data frame name at each step (like the messy code in the previous code chunk. The pipe works best with tidyverse functions, especially `dplyr` functions. You can read documentation on the pipe here. (I highly recommend this article; it is informative and also hilarious.)

```
# Load the package individually
library(magrittr)

# Organize your code in a step-wise manner using the pipe
```

```r
# This code does the same thing as the code in the previous chunk!

new_dataframe <- DATAFRAME %>%
  filter(VAR == 'VALUE') %>%
  select(VAR1, VAR2, VAR3) %>%
  group_by(PREDICTOR) %>%
  summarize(mean = sd(RESPONSE),
            sd = sd(RESPONSE))
```

Visualize data with ggplot2

There are several ways to make a bar graph in R; I highly recommend using the `ggplot` function from the `ggplot2` package. For the bar charts that we create for this class, you first need to wrangle the data into a summary table, using `dplyr` functions. Check out a cheatsheet for `ggplot2` here.

```r
# Load the package individually
library(ggplot2)

# Create a summary table for plotting
summary_table <- DATAFRAME %>%
  group_by(PREDICTOR) %>%
  summarise(
    mean = mean(RESPONSE),
    sd = sd(RESPONSE)
  )

# Create a graph using the summary table
ggplot(summary_table, aes(x = PREDICTOR, y = RESPONSE)) +
  geom_col() +
  geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd),
                width = 0.2, position = position_dodge(.9)) +
  scale_y_continuous(expand = c(0,0)) +
  theme_classic() +
  labs(title = "Title",
       x = "Predictor Variable", y = "Response Variable (units)")
```