MATH 3190 Homework 2

Advanced R (due 2/9/22)

2/2/22

Now its time to practice what we have learned in class and learn even more! Note that from now on your homework should be written in R Markdown, and 'turned in' by uploading to your GitHub repository.

1. (10 points) Learn more about the **scan**, **readLines**, **read_html**, **readr**, and **readxl** functions for getting data into R. Also, download the http://kenpom.com/cbbga22.txt data into a tibble using R. How are these different from the functions we learned in class? Report what you find and give some examples!

    - `scan`: The scan function in R is used to read and scan data from a file. It is usually used to read data into a vector. Example: scan("ex.data", skip = 1)
    - `readLines`: The readLines function in R is used to read text lines from an input file. Example: readLines(file)
    - `read_html:` This function is used to read in the content from a .html file. Example: read_html(file, skip = 0, remove.empty = TRUE, trim = TRUE, ...)
    - `readr:` This package is used to read in rectangular data like data from .csv files. Example: heights ¡- read_csv("data/heights.csv")
    - `readxl:` This package is used to read in data from xls or xlsx. Example: path <- readxl_example("geometry.xls"), read_excel(path)

2. (10 points) Learn about the **S3**, **S4**, and **R6** classes in R. When do you think you would use these? Describe what you learned and give some examples!

    - All the classes we are learning about in this question are object oriented programming classes in R. The way I like to think about objects is the idea of a big box, with smaller boxes inside that can be defined as the characteristic of the box. The human body can be used as an explanation, the organs, features, and functions of the body are characteristics of the body as a whole. Hopefully that makes sense! One place to use these object classes would be in designing applications or anything where we need to keep a track of the state of the machine or project we are designing. One would use R object classes if we were to take

a structured approach programming. A developer will easily be able to modify existing objects and create similar objects within a program if a dataset is defined as a custom object. All these classes allow for generic pre-built functions and also inheritance across classes.

- **S3**: S3 refers to a class built into R. It is more related to objected oriented programming. An S3 class object can be created in two ways, with a list or with attributes. Example: x <- structure(1, class = "foo").
- **S4**: The S3 and S4 classes are two generations of implementing functional objected-oriented programming. The S4 formal methods and classes provide all the methods and classes in S3 but require more programming.
- **R6**: R6 classes are similar to the other reference classes like the S4 and S3, but are lighter weight, and avoid some issues that come along with using S4. R6 is more closely related to Java's object-oriented programming classes with reference semantics which basically means it can be modified in place. Here is an example I found in the documentation:

```
Person <- R6Class("Person",
    public = list(
        name = NULL,
        hair = NULL,
        initialize = function(name = NA, hair = NA) {
          self$name <- name
          self$hair <- hair
          self$greet()
    },
    set_hair = function(val) {
      self$hair <- val
    },
    greet = function() {
      cat(paste0("Hello, my name is ", self$name, ".\n"))
    }
  )
)
```

3. (10 points) Now lets practice using our tidy data/tidyverse tools! Using your `cbbga22` tibble, try doing the following:

   (a) Use `mutate()` to create a new column that designates were each game was played. Also create a column that gives the score differences (team1−team2).

   (b) Use `arrange()` to sort the dataset by the location where the game was played.

(c) Use `select()` to remove columns 6 and 7. (name your columns first!)

(d) Use `filter()` to reduce the data down to only games played in 2022 (hint: use the `lubridate` package), save this in a new tibble. Write a function that will filter the tibble to only games played by a given team. Demonstrate your function by displaying games played by SUU.

(e) Use `summarize()` to extact SUU's win/loss recored and winning percentage. Generalize this by writing a function that will do this for a given team, and create a tibble with this information for all teams.

(f) Extra Credit (6 points): Find and download conference information. Use the `inner_join` function to connect the `cbbga22` to the conference, and then `group_by` to group the tibble by conference.

4. (30 points) Create an R package that contains your functions from the previous problem. Also add a function that generates an appropriate graph for the data. Make sure to properly document your code and add some unit tests. Now create a shiny app that relies on your R package, that displays the outputs of your functions (e.g., table displaying your data) and your graphs. Add the shiny app to your `inst/` directory in your package. Post your R package as a repository on GitHub.

   - To find the shiny app please look for the basketball package in my github repos
   - For the extra credit and the summarize function I did not complete it in the shiny app but it is in the homework2.Rmd file

5. (40 points) Walk step by step through the K means shiny app tutorial. Run the code to launch the app at each step. Document what changes are made in each step (which functions are used, what they do in the app). Now, make a K means R package with at least two functions (e.g., one that calculates the clusters, one that plots the result). Make sure to document your functions and create unit tests. Change the shiny app to rely on your functions, and add the shiny app to your `inst/` directory in your package. Post your R package as a repository on GitHub.

   - `Step 0:` This step adds a few things to the shiny app like a sidebar and a main panel but these are empty as of now.
   - `Step 1:` This step adds a title panel to the app and adds a name to it.
   - `Step 2:` This adds a few drop down menus to the sidebar panel in the app. Adding x, y, and a cluster count here.
   - `Step 3:` This adds a plot to the main panel in the application.
   - `Step 4:` This step actually renders the plot in the app.
   - `Step 5:` Addition of reactive axes. Adds scaled axes to the plot in the application.

- `Step 6:` k-means clustering model added to the application
- `Step 7:` The next step adds the cluster centers to the plot
- `Step 8:` This step saves the clusters into a variable. This can be used to adjust the input variables from the drop down menus added earlier.
- `Step 9:` This last step consolidates the final application, also beautifying it. Now the plot has become interactive as well as dynamically adjustable! This was an awesome walk-through.
- To find the R-package that I created, please look up the "kmeanspackage" in my repositories on github.