Homework 2

Basic data frame manipulations

Recall that we have learned the data `state.x77` in the lecture note. Below we create a dataframe `state.df`:

~~~
```r
state.df = data.frame(state.x77, Region=state.region, Division=state.division)
```
~~~

- **1a.** Add a column to `state.df`, containing the state abbreviations that are stored in the built-in vector `state.abb`. Name this column `Abbr`. You can do this in (at least) two ways: by using a call to `data.frame()`, or by directly defining `state.df$Abbr`. Display the first 3 rows and all 11 columns of the new state.df.
- **1b.** Remove the `Region` column from `state.df`. You can do this in (at least) two ways: by using negative indexing, or by directly setting `state.df$Region` to be `NULL`. Display the first 3 rows and all 10 columns of `state.df`.
- **1c.** Add two columns to `state.df`, containing the x and y coordinates (longitude and latitude, respectively) of the center of the states, that are stored in the (existing) list `state.center`. Hint: take a look at this list in the console, to see what its elements are named. Name these two columns `Center.x` and `Center.y`. Display the first 3 rows and all 12 columns of `state.df`.
- **1d.** Make a new data frame which contains only those states whose longitude is less than -100. Do this in two different ways: using manual indexing, and `subset()`. Check that they are equal to each other, using an appropriate function call.
- **1e.** Make a new data frame which contains only the states whose longitude is less than -100, and whose murder rate is above 9%. Print this new data frame to the console. Among the states in this new data frame, which has the highest average life expectancy?

Practice with the apply family

Below we read in a data frame `pros.dat` containing measurements on men with prostate cancer, which has 97 men x 9 variables.

- **2a.** Using `sapply()`, calculate the mean of each variable. Also, calculate the standard deviation of each variable. Each should require just one line of code. Display your results.
- **2b.** Let's plot each variable against SVI. Using `lapply()`, plot each column, excluding SVI, on the y-axis with SVI on the x-axis. This should require just one line of code. Label the y-axes in your plots appropriately. Your solution should still consist of just one line of code and use an apply function. Hint: for this part, consider using `mapply()`.
- **2c.** Now, use `lapply()` to 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 variable excluding the SVI variable itself. For convenience, we've defined a function `t.test.by.ind()` below, which takes a numeric variable `x`, and then an indicator variable `ind` (of 0s and 1s) that defines the groups. Run this function on the columns of `pros.dat`, excluding the SVI column itself, and save the result as `tests`. What kind of data structure is `tests`? Print it to the console.

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

```
    return(t.test(x[ind == 0], x[ind == 1]))
  }
```

- **2d.** Using `lapply()` again, extract the p-values from the `tests` object you created in the last question, with just a single line of code. Hint: first, take a look at the first element of `tests`, what kind of object is it, and how is the p-value stored? Second, run the command `"[["(pros.dat, "lcavol")` in your console—what does this do? Now use what you've learned to extract p-values from the `tests` object.

Data frame and apply practice

Now we're going to examine data from the 2016 Summer Olympics in Rio de Janeiro, taken from https://github.com/flother/rio2016 (itself put together by scraping the official Summer Olympics website for information about the athletes). Below we read in the data and store it as `rio`.

- **3a.** What kind of object is rio? What are its dimensions and columns names of `rio`? What does each row represent? Is there any missing data?
- **3b.** Use `rio` to answer the following questions. How many athletes competed in the 2016 Summer Olympics? How many countries were represented? What were these countries, and how many athletes competed for each one? Which country brought the most athletes, and how many was this? Hint: for a factor variable `f`, you can use `table(f)` see how many elements in `f` are in each level of the factor.
- **3c.** How many medals of each type—gold, silver, bronze—were awarded at this Olympics? Are they equal? Is this result surprising, and can you explain what you are seeing?
- **3d.** Create a column called `total` which adds the number of gold, silver, and bronze medals for each athlete, and add this column to `rio`. Which athlete had the most number of medals and how many was this? Gold medals? Silver medals? In the case of ties, here, display all the relevant athletes.
- **3e.** Using `tapply()`, calculate the total medal count for each country. Save the result as `total.by.nat`, and print it to the console. Which country had the most number of medals, and how many was this? How many countries had zero medals?
- **3f.** Among the countries that had zero medals, which had the most athletes, and how many athletes was this?

Young and old folks

- **4a.** The variable `date_of_birth` contains strings of the date of birth of each athlete. Use the `substr()` function to extract the year of birth for each athlete, and then create a new numeric variable called `age`, equal to 2016 - (the year of birth). (Here we're ignoring days and months for simplicity.) Hint: to extract the first 4 characters of a string `str`, you can use `substr(str, 1, 4)`. As always, you can also look at the help file for `substr()` for more details. Add the `age` variable to the `rio` data frame. Who is the oldest athlete, and how old is he/she? Youngest athlete, and how old is he/she? In the case of ties, here, display all the relevant athletes.
- **4b.** Answer the same questions as in the last part, but now only among athletes who won a medal.
- **4c.** Using a single call to `tapply()`, answer: how old are the youngest and oldest athletes, for each sport?
- **4d.** You should see that your output from `tapply()` in the last part is a list, which is not particularly convenient. Convert this list into a matrix that has one row for each sport, and two columns that display the ages of the youngest and oldest athletes in that sport. The first 3 rows should look like this:

```
              Youngest Oldest
aquatics            14     41
archery             17     44
```

```
athletics                16      47
```

You'll notice that we set the row names according to the sports, and we also set appropriate column names. Hint: `unlist()` will unravel all the values in a list; and `matrix()`, as you've seen before, can be used to create a matrix from a vector of values. After you've converted the results to a matrix, print it to the console (and make sure its first 3 rows match those displayed above).

Transformation on data

- **5a.** Create a new data frame called `sports`, which we'll populate with information about each sporting event at the Summer Olympics. Initially, define `sports` to contain a single variable called `sport` which contains the names of the sporting events in alphabetical order. Then, add a column called `n_participants` which contains the number of participants in each sport. Use one of the apply functions to determine the number of gold medals given out for each sport, and add this as a column called `n_gold`. Using your newly created `sports` data frame, calculate the ratio of the number of gold medals to participants for each sport. Which sport has the highest ratio? Which has the lowest?
- **5b.** Use one of the apply functions to compute the average weight of the participants in each sport, and add this as a column to `sports` called `ave_weight`. Important: there are missing weights in the data set coded as `NA`, but your column `ave_weight` should ignore these, i.e., it should be itself free of `NA` values. You will have to pass an additional argument to your apply call in order to achieve this. Hint: look at the help file for the `mean()` function; what argument can you set to ignore `NA` values? Once computed, display the average weights along with corresponding sport names, in decreasing order of average weight.
- **5c.** As in the last part, compute the average weight of atheletes in each sport, but now separately for men and women. You should therefore add two new columns, called `ave_weight_men` and `ave_weight_women`, to `sports`. Once computed, display the average weights along with corresponding sports, for men and women, each list sorted in decreasing order of average weight. Are the orderings roughly similar?
- **5d.** Repeat the calculation as in the last part, but with BMI (body mass index) replacing weight. Note that $BMI = weight/height^2$.

Merging Dataframes

Below we read in two data frames `dat.m` and `dat.w` in the `sprint.Rda` file, which contains the fastest times in the 100m sprint for men and women. Merge these twoe data frames using `merge()`.

- **6a.** Perform an inner join, using `all=FALSE`, of `dat.m` and `dat.w`, with the join done by the Country column. Call the resulting data frame `dat.ij`, and display its first 10 rows. How many rows does it have in total? Show how could you have arrived at this number ahead of time, from `dat.m$Country` and `dat.w$Country` (hint: `intersect()`). Count the number of `NA` values in `dat.ij`: this should be zero.
- **6b.** Perform a left join, using `all.x=TRUE`, of `dat.m` and `dat.w`, with the join again done by the Country column. Call the resulting data frame `dat.lj`, and display its first 10 rows. How many rows does it have in total? Explain why this makes sense. Count the number of rows with at least one `NA` value in `dat.lj`: this should be 25. Show how you could have arrived at this number from `dat.m$Country` and `dat.w$Country` (hint: `setdiff()`).
- **6c.** Finally, perform an full join, using `all=TRUE`, of `dat.m` and `dat.w`, with the join again done by the Country column. Call the resulting data frame `dat.fj`. How many rows does it have in total? Show how you could have arrived at this number from `dat.m$Country` and `dat.w$Country` (hint: `union()`). Count the number of rows with at least one `NA` value in `dat.fj`: this should be 40. Show how you could have arrived at this number from `dat.m$Country` and `dat.w$Country`.