Intro to Data Handling in R

Here we introduce a few key basics of handling data in R, using RStudio. In the process, you will also get experience with installing and loading packages. Specifically, we will:

a) import data from an excel file as both csv and txt files
b) install and use parts of the *dplyr* package in R – very helpful for managing data

We work here with baby names data from the Social Security Administration. Here each state's records (1880-2021) for babies born in each year are listed, where the minimum number for each name = 5. The data are available as zip file at https://www.ssa.gov/oact/babynames/names.zip

1. Make a folder on your network drive and download and unzip the file into that folder.
2. Set this folder as your working directory in R. Use the menu at the top: Session... Set Working Directory... Choose Directory...

How much data do we have? How many MB in the unzipped folder? _____
Trust me – you don't want to handle all these data sets in Excel.
Instead, let's get them into R.

3. Make an object called file_names that is a list of the .txt files only

```
file_names <- list.files(getwd(),pattern = "txt")
```

4. Now we: use `lapply` to repeatedly read files; add file names as a column; extract just the date from those file names; edit column names, and convert year to numbers

```
files <- lapply(file_names, read.csv, header=F, stringsAsFactors = F)
for (i in 1:length(files)){files[[i]] <-cbind(files[[i]],file_names[i])}
   babynames <- do.call(rbind, files)
colnames(babynames)[c(1,2,3,4)] <- c("name", "sex", "count","year")
babynames$year <- regmatches(babynames$year, gregexpr("[[:digit:]]+",
   babynames$year))
babynames$year <- as.numeric(babynames$year)
```

Wait for the Red Stop Sign to go away (it means the computer is working – if you click on that icon it will stop the process). We now have one dataframe called babynames, that combined all those txt files (!)

How many rows of data do we have? _____ How many columns? _____
How did I know to make those commands? Why, by googling around, of course!

5. Now we play with the data using `dplyr`. Installing can be done in 2 ways: using RStudio menus or commands. In RStudio, click on Packages (top of bottom-right window).
   - If it's not in the list, click Install at the top of the Packages window and type `dplyr` in the pop-up window. That implements the `install.packages(dplyr)` command you see running in the Console.

- If dplyr is in the list, click on the box (or type `library(dplyr)` in the Console. This turns on that package.

6. If you just installed, click on the blue word "`dplyr`" in Packages to see R documentation. A very good strategy is to first read vignettes – for an English translation of what packages do. Lots of other info is here too. Plan to spend time perusing Instructions when you first learn a package. And when learning details or troubleshooting, browse the web – many web sites are out there for help (some are listed in this course's R Resources web site). And at the bottom of Help screens are often helpful examples. And plan to run thru commands using example data sets provided.

   Now let's plunge into some `dplyr` and related commands.

Unique: How many different spellings of a basal name are in there? (try it first with "Dave")

7. First grab names that start with "Dav" :

   ```
   Daveish <-  subset(babynames, grepl("^Dav", babynames$name))
   ```

   and list the unique names in there:

   ```
   Davs <- unique(Daveish$name)
   Davs  # to see what's in Davs: there's a buncha versions of Dav...
   ```

8. ***Edit steps 7 & 8 for your name!*** And keep doing that for steps below...

Filter: Selects **rows** in a dataframe. For example, you can filter for names in a list (like those in Davs above).

9. Make a data frame based on babynames but filtered for only Davs names:

   ```
   Daves <- filter(babynames, name %in% Davs)
   ```

Arrange: Re-sorts columns for your specified variable, much like you may do in a spreadsheet. Our data are already arranged by year.

10. But we can alter that order:

    ```
    Davesreverse <- arrange(Daves, -year) # years are in reverse order
    ```

    or maybe by name

    ```
    Davesalpha <- arrange(Daves, name) # alphabetical by default
    ```

You can see those data (click on Davesreverse or Davesalpha in the upper right box).

Group by:

11. Organize data into (invisible) groups – great for later calculations.

```
Daveversions <- group_by(Daves, name)
```

Summarise:

12. Calculates column values for a data frame (or part of one), such as total, mean, etc. For example:

```
Daveosity <- summarise(Daveversions, namects = sum(count))
# to count the total number per each version of Dav*
```

    It will work best in a minute if you now arrange Daveosity by count in decreasing order.

Group by AND Summarise (Very Handy): You could do steps 11 & 12 in one command.

13. The %>% thingy below is is called a pipeline to link a series of commands - **very efficient code**! For example,

```
Daveosity <- group_by(Daves, name) %>% summarise(namects = sum(count))

Daveosity <- arrange(Daveosity, -namects) # decreasing order by counts
```

    Which version of Dav* is most common? Which are most rare?

Now let's graph results through time.

14. First we have to lump versions by year:

```
Davbyyear <- group_by(Daves, year) %>% summarise(annualcts = sum(count))
```

    and then we can graph:

```
plot(Davbyyear$annualcts ~ Davbyyear$year)
```

Do the same for Ashleys (or your name): How do they compare?

15. If you want to stack graphs for easy comparison, use this:

```
par(mfrow=c(2,1)) # this arranges output in two rows, one column
```

    To put it back to one graph just use `par(mfrow=c(1,1))`

16. One last questions: How many different versions of “Dav” appear each year?

```
Davsthrutime <- group_by(Daves, year) %>% summarise(namesct =
length(unique(name))) # length (or count) of unique names

plot(Davsthrutime$namesct~Davsthrutime$year)
```

And how does that compare different versions of Ashley? Try using the “magnifying glass” at the to Find-Replace in the code to change names – it’s easy to update code and then re-run!

We have just scratched the surface of dplyr. Good dplyr resources include:

- http://blog.rstudio.org/2014/01/17/introducing-dplyr/
- http://www.r-bloggers.com/using-r-quickly-calculating-summary-statistics-with-dplyr/

Others abound – remember, when using R, the web is your friend... For example, you should explore using **`mutate`** and the **`join`** set of commands on your own. *Hint*, *Hint*.

18. Finally, save your R script and data files because you will use this (and more) for this week's homework!