Data I/O, Part 1

Data Wrangling in R

Explaining output on slides

In slides, a command (we'll also call them code or a code chunk) will look like this

```
print("I'm code")
```

```
[1] "I'm code"
```

And then directly after it, will be the output of the code.
So `print("I'm code")` is the code chunk and [1] "I'm code" is the output.

These slides were made in R using `knitr` and `R Markdown` which is covered in later today when we discuss reproducible research.

Data Input

- 'Reading in' data is the first step of any real project/analysis
- R can read almost any file format, especially via add-on packages
- We are going to focus on simple delimited files first
  - tab delimited (e.g. '.txt')
  - comma separated (e.g. '.csv')
  - Microsoft excel (e.g. '.xlsx')

Data Input

UFO Sightings via Kaggle.com: “Reports of unidentified flying object reports in the last century”.

“There are two versions of this dataset: scrubbed and complete. The complete data includes entries where the location of the sighting was not found or blank (0.8146%) or have an erroneous or blank time (8.0237%). Since the reports date back to the 20th century, some older data might be obscured. Data contains city, state, time, description, and duration of each sighting.”

https://www.kaggle.com/NUFORC/ufo-sightings

Data Input

- Download data from http://sisbid.github.io/Module1/data/ufo/ufo_data_complete.csv.gz
- Upload the data to RStudio Cloud

Data Input

Easy way: R Studio features some nice “drop down” support, where you can run some tasks by selecting them from the toolbar.

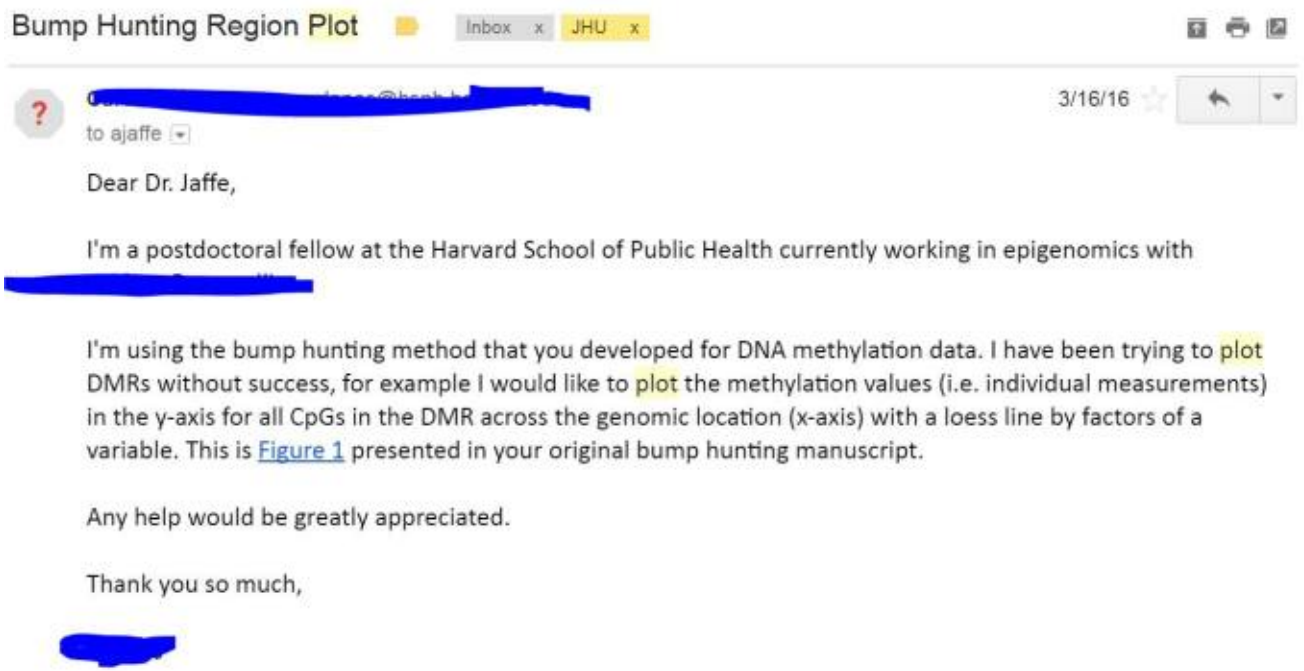
For example, you can easily import text datasets using the “Tools –> Import Dataset –> From Text (readr)” command. Selecting this will bring up a new screen that lets you specify the formatting of your text file.

After importing a dataset, you get the corresponding R commands that you can enter in the console if you want to re-import data.

Commenting in Scripts

Commenting in code is super important. You should be able to go back to your code years after writing it and figure out exactly what the script is doing. Commenting helps you do this. This happens to me often…

Commenting in Scripts



Bump Hunting Region Plot

Inbox x JHU x

[redacted] 3/16/16

to ajaffe

Dear Dr. Jaffe,

I'm a postdoctoral fellow at the Harvard School of Public Health currently working in epigenomics with [redacted]

I'm using the bump hunting method that you developed for DNA methylation data. I have been trying to plot DMRs without success, for example I would like to plot the methylation values (i.e. individual measurements) in the y-axis for all CpGs in the DMR across the genomic location (x-axis) with a loess line by factors of a variable. This is Figure 1 presented in your original bump hunting manuscript.

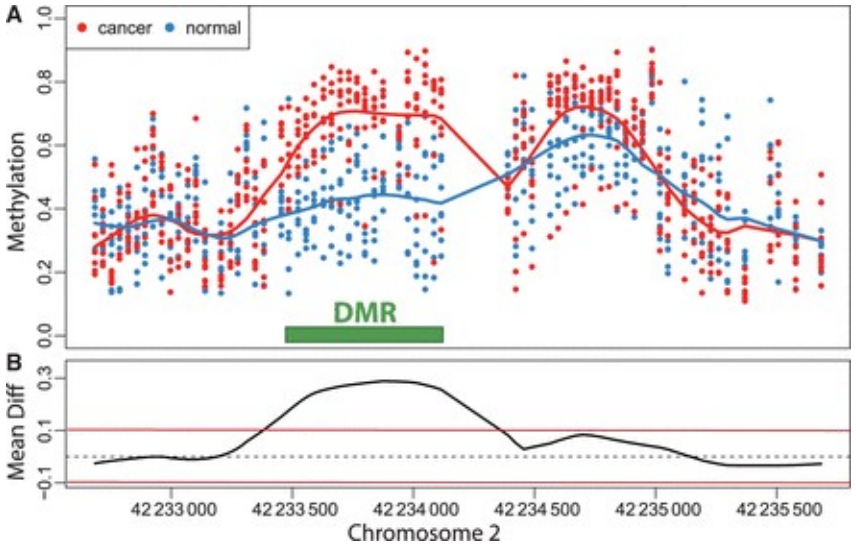
Any help would be greatly appreciated.

Thank you so much,

[redacted]

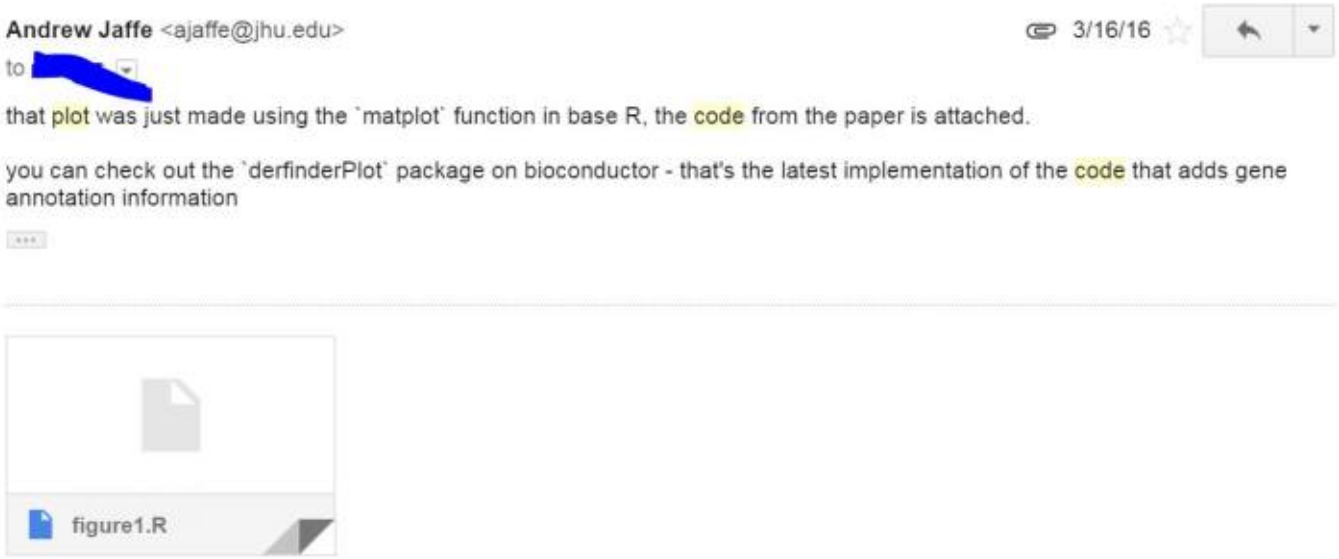
The paper came out January 2012 with code made in 2011

Commenting in Scripts



This was the figure…

Commenting in Scripts



After some digging, I found the code

Commenting in Scripts

Add a comment header to your script from today: `#` is the comment symbol

```
#################
# Title: Demo R Script
# Author: Andrew Jaffe
# Date: 7/13/2020
# Purpose: Demonstrate comments in R
###################

# nothing to its right is evaluated

# this # is still a comment
### you can use many #'s as you want

# sometimes you have a really long comment,
#    like explaining what you are doing
#    for a step in analysis.
# Take it to another line
```

R variables

- You can create variables from within the R environment and from files on your computer
- R uses “=” or “<-” to assign values to a variable name
- Variable names are case-sensitive, i.e. X and x are different

```
x = 2 # Same as: x <- 2
x
```

```
[1] 2
```

```
x * 4
```

```
[1] 8
```

```
x + 2
```

```
[1] 4
```

Help

For any function, you can write `?FUNCTION_NAME`, or `help("FUNCTION_NAME")` to look at the help file:

```
?dir
help("dir")
```

Data Input

Initially-harder-but-gets-way-easier method: Utilizing functions in the `readr` package called `read_delim()` and `read_csv()` with code.

Data Input

So what is going on “behind the scenes”?

`read_delim()`: Read a delimited file into a data frame.

```
function (file, delim, quote = "\"", escape_backslash = FALSE,
    escape_double = TRUE, col_names = TRUE, col_types = NULL,
    locale = default_locale(), na = c("", "NA"), quoted_na = TRUE,
    comment = "", trim_ws = FALSE, skip = 0, n_max = Inf, guess_max = min(1000
        n_max), progress = show_progress(), skip_empty_rows = TRUE)
NULL

# for example: read_delim("file.txt",delim="\t")
```

Data Input

- The filename is the path to your file, in quotes
- The function will look in your “working directory” if no absolute file path is given
- Note that the filename can also be a path to a file on a website (e.g. ‘www.someurl.com/table1.txt’)

Data Input

There is another convenient function for reading in CSV files, where the delimiter is assumed to be a comma:

```
read_csv
```

```
function (file, col_names = TRUE, col_types = NULL, locale = default_locale(),
    na = c("", "NA"), quoted_na = TRUE, quote = "\"", comment = "",
    trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
        n_max), progress = show_progress(), skip_empty_rows = TRUE)
{
    tokenizer <- tokenizer_csv(na = na, quoted_na = quoted_na,
        quote = quote, comment = comment, trim_ws = trim_ws,
        skip_empty_rows = skip_empty_rows)
    read_delimited(file, tokenizer, col_names = col_names, col_types = col_typ
        locale = locale, skip = skip, skip_empty_rows = skip_empty_rows,
        comment = comment, n_max = n_max, guess_max = guess_max,
        progress = progress)
}
<bytecode: 0x7fe72639cbd8>
<environment: namespace:readr>
```

Data Input

- Here would be reading in the data from the command line, specifying the file path:

```
ufo = read_csv("../data/ufo/ufo_data_complete.csv")
```

```
Parsed with column specification:
cols(
  datetime = col_character(),
  city = col_character(),
  state = col_character(),
  country = col_character(),
  shape = col_character(),
  `duration (seconds)` = col_double(),
  `duration (hours/min)` = col_character(),
  comments = col_character(),
  `date posted` = col_character(),
  latitude = col_character(),
  longitude = col_double()
)

Warning: 199 parsing failures.
 row col   expected     actual                                  file
 877  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
1712  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
1814  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
2857  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
3733  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
```

Data Input

The `read_delim()` and related functions returns a "tibble" is a `data.frame` with special printing, which is the primary data format for most data cleaning and analyses.

```
head(ufo)
```

```
# A tibble: 6 x 11
  datetime city  state country shape `duration (seco… `duration (hour… comment
  <chr>    <chr> <chr> <chr>   <chr>             <dbl> <chr>            <chr>  
1 10/10/1… san … tx    us      cyli…              2700 45 minutes       This ev
2 10/10/1… lack… tx    <NA>    light              7200 1-2 hrs          1949 La
3 10/10/1… ches… <NA>  gb      circ…                20 20 seconds       Green/O
4 10/10/1… edna  tx    us      circ…                20 1/2 hour         My olde
5 10/10/1… kane… hi    us      light               900 15 minutes       AS a Ma
6 10/10/1… bris… tn    us      sphe…               300 5 minutes        My fath
# … with 3 more variables: `date posted` <chr>, latitude <chr>, longitude <dbl
```

```
class(ufo)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Data Input

```
ufo
```

```
# A tibble: 88,875 x 11
   datetime city  state country shape `duration (seco… `duration (hour… commer
   <chr>    <chr> <chr> <chr>   <chr>             <dbl> <chr>            <chr>
 1 10/10/1… san … tx    us      cyli…              2700 45 minutes       This e
 2 10/10/1… lack… tx    <NA>    light              7200 1-2 hrs          1949 I
 3 10/10/1… ches… <NA>  gb      circ…                20 20 seconds       Green/
 4 10/10/1… edna  tx    us      circ…                20 1/2 hour         My old
 5 10/10/1… kane… hi    us      light               900 15 minutes       AS a M
 6 10/10/1… bris… tn    us      sphe…               300 5 minutes        My fat
 7 10/10/1… pena… <NA>  gb      circ…               180 about 3 mins     penart
 8 10/10/1… norw… ct    us      disk               1200 20 minutes       A brig
 9 10/10/1… pell… al    us      disk                180 3  minutes       Strobe
10 10/10/1… live… fl    us      disk                120 several minutes  Saucer
# … with 88,865 more rows, and 3 more variables: `date posted` <chr>,
#   latitude <chr>, longitude <dbl>
```

Data Input

There are also data importing functions provided in base R (rather than the `readr` package), like `read.delim` and `read.csv`.

These functions have slightly different syntax for reading in data, like `header` and `as.is`.

However, while many online resources use the base R tools, recent versions of RStudio switched to use these new `readr` data import tools, so we will use them in the class for slides. They are also up to two times faster for reading in large datasets, and have a progress bar which is nice.

Data Input - Excel

Many data analysts collaborate with researchers who use Excel to enter and curate their data. Often times, this is the input data for an analysis. You therefore have two options for getting this data into R:

- Saving the Excel sheet as a .csv file, and using `read.csv()`
- Using an add-on package, like `readxl`

For single worksheet .xlsx files, I often just save the spreadsheet as a .csv file (because I often have to strip off additional summary data from the columns)

For an .xlsx file with multiple well-formated worksheets, I use the `readxl`package for reading in the data.

Data Input - Other Software

- **haven** package (https://cran.r-project.org/web/packages/haven/index.html) reads in SAS, SPSS, Stata formats
- **sas7bdat** reads .sas7bdat files
- **foreign** package - can read all the formats as **haven**. Around longer (aka more testing), but not as maintained (bad for future).

Common new user mistakes we have seen

1. **Working directory problems: trying to read files that R "can't find"**
   - RStudio can help, and so do RStudio Projects
   - discuss in this Data Input/Output lecture
2. Lack of comments in code
3. Typos (R is **case sensitive**, `x` and `X` are different)
   - RStudio helps with "tab completion"
   - discussed throughout
4. Data type problems (is that a string or a number?)
5. Open ended quotes, parentheses, and brackets
6. Different versions of software

Working Directories

- R “looks” for files on your computer (or cloud) relative to the “working” directory
- Many people recommend not setting a directory in the scripts
  - assume you’re in the directory the script is in
  - If you open an R file with a new RStudio session, it does this for you.
- If you do set a working directory, do it at the beginning of your script.
- Example of getting and setting the working directory:

```
## get the working directory
getwd()
setwd("~/Lectures")
```