Practical R: Introduction

Abhijit Dasgupta

BIOF 339

What does R look like?

- A scripting language
  - Provide instructions to the computer
  - in a structured manner
  - to do statistical analysis

```
# Determining frequencies of breast cancer subtypes

type_frequencies <-
  breast_cancer %>%
    mutate(luminalA = ifelse(ER == '+' & PR == '+' & HER2 == '-', 1, 0),
         luminalB = ifelse(ER == '+' & PR == '-' & HER2 == '+', 1, 0),
         her2 = ifelse(ER == '-' & PR == '-' & HER2 == '+', 1, 0),
         basal = ifelse(ER == '-' & PR == '-' & HER2 == '-', 1, 0)) %>%
    mutate(type = case_when(
      luminalA == 1 ~ "Luminal A",
      luminalB == 1 ~ "Luminal B",
      her2 == 1 ~ "HER2",
      basal == 1 ~ "Basal",
      TRUE ~ NA)) %>%
    count(type)
```

What does R look like?

- Start with a data set

```
# Determining frequencies of breast cancer subtypes

type_frequencies <-
  breast_cancer %>%
    mutate(luminalA = ifelse(ER == '+' & PR == '+' & HER2 == '-', 1, 0),
         luminalB = ifelse(ER == '+' & PR == '-' & HER2 == '+', 1, 0),
         her2 = ifelse(ER == '-' & PR == '-' & HER2 == '+', 1, 0),
         basal = ifelse(ER == '-' & PR == '-' & HER2 == '-', 1, 0)) %>%
    mutate(type = case_when(
      luminalA == 1 ~ "Luminal A",
      luminalB == 1 ~ "Luminal B",
      her2 == 1 ~ "HER2",
      basal == 1 ~ "Basal",
      TRUE ~ NA)) %>%
    count(type)
```

What does R look like?

- Start with a data set
- Create new variables from old variables

```
# Determining frequencies of breas

type_frequencies <-
  breast_cancer %>%
    mutate(luminalA = ifelse(ER ==
           luminalB = ifelse(ER ==
           her2 = ifelse(ER == '-' &
           basal = ifelse(ER == '-'
    mutate(type = case_when(
      luminalA == 1 ~ "Luminal A"
      luminalB == 1 ~ "Luminal B"
      her2 == 1 ~ "HER2",
      basal == 1 ~ "Basal",
      TRUE ~ NA)) %>%
    count(type)
```

What does R look like?

- Start with a data set
- Create new variables from old variables
- Deal with missing values

```
# Determining frequencies of breast cancer subtypes

type_frequencies <-
  breast_cancer %>%
    mutate(luminalA = ifelse(ER == '+' & PR == '+' & HER2 == '-', 1, 0),
         luminalB = ifelse(ER == '+' & PR == '-' & HER2 == '+', 1, 0),
         her2 = ifelse(ER == '-' & PR == '-' & HER2 == '+', 1, 0),
         basal = ifelse(ER == '-' & PR == '-' & HER2 == '-', 1, 0)) %>%
    mutate(type = case_when(
      luminalA == 1 ~ "Luminal A",
      luminalB == 1 ~ "Luminal B",
      her2 == 1 ~ "HER2",
      basal == 1 ~ "Basal",
      TRUE ~ NA)) %>%
    count(type)
```

What does R look like?

- Start with a data set
- Create new variables from old variables
- Deal with missing values
- Find the frequencies

```r
# Determining frequencies of breast cancer subtypes

type_frequencies <-
  breast_cancer %>%
    mutate(luminalA = ifelse(ER == '+' & PR == '+' & HER2 == '-', 1, 0),
         luminalB = ifelse(ER == '+' & PR == '-' & HER2 == '+', 1, 0),
         her2 = ifelse(ER == '-' & PR == '-' & HER2 == '+', 1, 0),
         basal = ifelse(ER == '-' & PR == '-' & HER2 == '-', 1, 0)) %>%
    mutate(type = case_when(
      luminalA == 1 ~ "Luminal A",
      luminalB == 1 ~ "Luminal B",
      her2 == 1 ~ "HER2",
      basal == 1 ~ "Basal",
      TRUE ~ NA)) %>%
    count(type)
```

What does R look like?

- Start with a data set
- Create new variables from old variables
- Deal with missing values
- Find the frequencies
- Comment on what you're doing

```
# Determining frequencies of breast cancer subtypes

type_frequencies <-
  breast_cancer %>%
    mutate(luminalA = ifelse(ER == '+' & PR == '+' & HER2 == '-', 1, 0),
           luminalB = ifelse(ER == '+' & PR == '-' & HER2 == '+', 1, 0),
           her2 = ifelse(ER == '-' & PR == '-' & HER2 == '+', 1, 0),
           basal = ifelse(ER == '-' & PR == '-' & HER2 == '-', 1, 0)) %>%
    mutate(type = case_when(
      luminalA == 1 ~ "Luminal A",
      luminalB == 1 ~ "Luminal B",
      her2 == 1 ~ "HER2",
      basal == 1 ~ "Basal",
      TRUE ~ NA)) %>%
    count(type)
```

This is an example of a pipeline in R. We'll develop different aspects of this progressively throughout the semester

Why use a scripting language for analysis?

Pros:

1. Have to think
2. Reproducible (custom) workflows
3. Much less error-prone
4. Much lower costs to repeat analyses, or as you learn more
5. Easily leverage work of smarter developers
6. Easier to work with larger datasets (more than size of screen)

Cons:

1. Have to type
2. Have to know the language
3. Higher initial startup cost
4. Have to think

Why use a scripting language for analysis?

Pros:

1. Have to think
2. Reproducible (custom) workflows
3. Much less error-prone
4. Much lower costs to repeat analyses, or as you learn more
5. Easily leverage work of smarter developers
6. Easier to work with larger datasets (more than size of screen)

- You're giving instructions to a fast but stupid machine
- This machine will do **exactly** what you tell it
- The machine is capable of amazing things
- Can't just *menu-mine* and try things that **seem** to be what you want

With great power comes great responsibility

But also great benefits

Why use a scripting language for analysis?

Pros:

1. Have to think
2. Reproducible (custom) workflows
3. Much less error-prone
4. Much lower costs to repeat analyses, or as you learn more
5. Easily leverage work of smarter developers
6. Easier to work with larger datasets (more than size of screen)

- If your code is not right, it won't run
  - Can be frustrating
  - But if it runs you're much more confident
- If you screw up in Excel
  - almost impossible to recover
- You have much more control over what you're doing

Why use a scripting language for analysis?

Pros:

1. Have to think
2. Reproducible (custom) workflows
3. Much less error-prone
4. Much lower costs to repeat analyses, or as you learn more
5. Easily leverage work of smarter developers
6. Easier to work with larger datasets (more than size of screen)

- Can use *modules* or *packages* developed by others
  - `tidyverse`, `Seurat`, `ggplot2`
- Can "steal" code from others (provided license allows)

Why use a scripting language for analysis?

Pros:

1. Have to think
2. Reproducible (custom) workflows
3. Much less error-prone
4. Much lower costs to repeat analyses, or as you learn more
5. Easily leverage work of smarter developers
6. Easier to work with larger datasets (more than size of screen)

Good luck working with

- GWAS data
- fMRI data
- Stocks and bonds data
- Sports data
- Many more ...

in a unified environment

Learn once, use everywhere

Why use R?

- Specializes in statistics and data visualization
- Flexible
  - ~~*If you can do it*~~ **How you can do it**
- Large ecosystem
  - Over 16,000 packages, 1500+ dedicated to bioinformatics
  - Can read from most sources of data
  - Generic and specialized analyses
- Fantastic community
  - Twitter, StackOverflow, blogosphere, conferences, online books

Why use R?

R is a very high-quality product that is accepted and used widely in government agencies, corporations and universities worldwide

- Standard data analytic software in bioinformatics, behavioral health and many aspects of quantitative finance
- Increasingly used in pharma, economics, political science and engineering

R is open-source, in that anyone can see the actual code and validate the computations directly

Why use R?

- R has a well-deserved reputation for being a great data visualization tool, with users being able to create complex, customizable graphs with relative ease
- As a scripting language, it allows the same workflows to be coded and re-used.
- You can set up workflows to validate data, in terms of data quality and missingness, which avoids visual inspection which can be time-consuming and mistake-prone.
- R can handle large datasets, and can work with multiple datasets at the same time

Why use R?

Specialized packages available for many domains

- Bioinformatics
- Econometrics
- Maps and spatial analytics
- Text mining and Natural Language Processing

The CRAN Task Views provide curated lists of packages based on different domains

The Bioconductor Project is THE source for bioinformatics packages in R. It is the gold standard for many bioinformatic workflows

Things  is not (in this class)

- The 18th letter of the English alphabet
- A magic incantation that will produce an analysis
- Just something the cool kids are doing
- Point-and-click, automatic, WYSIWYG (What you see is what you get)
    - So it's not Excel, SPSS, Prism, GraphPad
    - It's much more!!!

A note on Excel 

Excel is omnipresent, so it becomes the default data medium

It is great for many things, including *quick-and-dirty* analyses

It can be error-prone

It needs to be backed up

It has size limitations

Takes a lot of effort to do complex analyses

- No way to reproduce analyses without macros
- No way to document what you are doing
- Excel has some nasty default behavior
    - Guess what the `MAR1` gene gets recorded as?
- Very hard to recover from errors
    - Shift of one error (off by one row or column)
    - Google *Duke Potti* or *Reinhart Rogoff Herndon*