Lab 2: Intro to R

Due: Monday, January 22, 11:59 PM

Name: Your name here

Mac ID: The first half of your Mac email address

1 R packages

R is open-source software. Which means users can develop their own applications to expand its functionality. That's why a private company like Posit can make RStudio as its own user interface for R.

Much like a smartphone, we can expand R's functionality by installing apps. We call these applications **packages**.

The following code chunk will install the packages we need for today. You can run a line of code with `cmd + Enter` in Mac or `Ctrl + Enter` in Windows.

You can also run a code chunk by clicking on the play button in the top-right corner of chunk.

```
install.packages(c("tidyverse", "DeclareDesign")
```

Notice that I set up the code chunk option `eval = FALSE`. This will print the code in the PDF but not process it in any way. We do this because we only need to install packages once. No need to process it multiple times. *Please wait until you are done working on this*

lab assignment to make the PDF. Further instructions await.

> **Task 1**
>
> Once packages are installed. Use # to comment out the `install.packages` function above.

There are many other ways to install R packages. For example, in a saved copy of an `.qmd` file, R will pop a message at the top of the Source panel asking to install the packages that have not been installed. In future labs we will skip package installation.

Like phone apps, packages only work when you load them. The following code loads the packages we will need.

```
# I set up the code chunk options to process the code but hide the output.
# This is useful to show the packages you are using
# without the messages that follow.
library(tidyverse)
library(DeclareDesign)
```

The `tidyverse` is a collection of popular packages that make R more coherent. Think of it as if base R was written in 20th century British English, whereas the `tidyverse` packages are all written in 21st century American English.[1] This is no minor feat since some modern packages may be written in South African English, for example.

This is all to say that, for this course, you will learn to code in R in a very specific, opinionated, yet useful way.[2]

The `DeclareDesign` package is part of the set of software tools that accompany our textbook. It has all the relevant functions to imagine, declare, and diagnose research designs.

[1] By which I mean US English. I am too new to Canada to know where to place Canadian English in this analogy. Also I apologi~~z~~sze for writing in US English.

[2] Continuing the analogy, eventually you may learn enough English to understand all the different dialects as they evolve. You may need more than this course to do so.

2 Basic operations

In the most basic way, R is a calculator. You can do things like:

```
1/200*30
```

```
[1] 0.15
```

Notice that R knows the order of operations and parenthesis:

```
1+2*3
```

```
[1] 7
```

```
(1+2)*3
```

```
[1] 9
```

And that some common operations are rewritten as functions:

```
sqrt(10)
```

```
[1] 3.162278
```

R *ignores spaces* but is *case sensitive*. For example:

```
# x is the sequence of numbers from 1 to 10
x = 1:10


mean(x)
```

```
[1] 5.5
```

Is not the same as

```
mean(X)
```

```
Error in mean(X): object 'X' not found
```

That said, it pays off to write code as if you were writing prose. Spaces and line breaks make things more readable. Use whatever formatting style suits you, the `formatR` package will take care of the most annoying things.

Notice that we use the `=` operator to store an object in R's memory (which you can consult in the environment tab in the top right).[3] When we say that R is an object-oriented programming language, we mean that the core of what we do in R is storing objects in its environment (data, functions, variables, etc.) and then do stuff with it.

Task 2

Figure out how to insert a new code chunk right below this prompt. Then create an object called `even` with a sequence of even numbers from 1 to 10.

Tip: *There are many ways to accomplish this. I would start by writing* `?seq` *in the console to learn about one way to do it.*

The `tidyverse` has its own way to write map functions onto objects. You can also write:

```
x %>% mean()
```

```
[1] 5.5
```

And it should give the same answer. This may look trivial now but it becomes convenient when you want to string operations. For example:

```
x %>% mean() %>% sqrt()
```

```
[1] 2.345208
```

The symbol `%>%` here is known as the pipe operator because you use it build a pipeline. I tend to write in pipelines, but you are welcome to write code in any way that feels natural.

[3] Some people `<-` instead of `=` as the assignment operator. Use whichever makes more sense to you. I use `=` because it involves fewer keystrokes.

This was a very brief introduction. Learning the basics of R takes weeks, being comfortable with the basic stuff may also take months. Do not hesitate to ask questions about how to do something (that's why we save some of our meeting time to work on lab assignments). Feel free to work with peers or use any of the online resources available out there.

Remember that this course is not quizzing your knowledge of R. We are merely using it as a language to write down research designs.

You may also want to check the suggested textbook or the Resources page of the course website to learn about R as you go.

3 The MIDA workflow

The advantage of using the MIDA framework to think about research design is that we can translate its steps into code. Here's a basic example:

```
# Model
model = declare_model(
  N = 100,
  U = rnorm(N),
  potential_outcomes(Y ~ 0.25 * Z + U)
  )


# Inquiry
inquiry = declare_inquiry(
  PATE = mean(Y_Z_1 - Y_Z_0)
  )
```

```
# Data strategy
sampling = declare_sampling(
  S = complete_rs(N, n = 50)
  )


assignment = declare_assignment(
  Z = complete_ra(N, prob = 0.5)
  )


measurement = declare_measurement(
  Y = reveal_outcomes(Y ~ Z)
  )


# Answer strategy
answer = declare_estimator(
  Y ~ Z,
  .method = difference_in_means,
  inquiry = "PATE"
  )
```

Task 3

To the best of your ability, explain what `model`, `inquiry`, `sampling`, `assignment`, `measurement`, and `answer` are doing in the previous code chunk.

Hint: *Chapter 4 of the textbook.*

Write your answer here, be as detailed as you can be

The advantage of encoding a research design in this way is that we diagnose it. That implies

simulating many realizations of the same study under the same conditions and calculating some statistics relating to its performance. We call these statistics **diagnosands**.

Here's how we put all the steps of a research design together and diagnose:

```
# Put design elements together in MIDA order
design = model + inquiry + sampling + assignment + measurement + answer


# We use set.seed() to fix the random number generation
# This makes it so that numerical results stay the same every time
# In the next line, replace 1234 with your student number
set.seed(1234)


# diagnose design
diagnosis = diagnose_design(design)


diagnosis$diagnosands_df %>%
  select(bias, rmse, power)
```

```
        bias      rmse power
1 0.02084813 0.2661531 0.148
```

Task 4

What numerical results did you get for the columns named `bias`, `rmse`, and `power`? What do you think they mean? What can you conclude based on your findings?

Hint: *Check chapter 4 of the textbook again. Don't stress about getting the answer exactly right, just give it your best shot. We will learn more about this as the semester goes by.*

Write your answer here