TDU.

INTRODUCTION TO R

Advanced Data Processing

Day 3

Public Health Agency of Canada
Agence de la santé publique du Canada

Canada

DEBRIEF – EXERCISE 2

- Who reproduced their own code from course materials? Who wrote their own? Who approached the provided code more conceptually, as a translation exercise?
- Was there anything that you found surprising or easier than expected?
- Did you learn anything that you found interesting?
- Any further questions or comments?

WHAT WE HEARD

- Comparing vectors: == vs. %in%
  - Logical operator vs. value matching
- RStudio theme
  - tools > global options > appearance > editor theme
- Images and dark mode
  - bg= "white"

COURSE LEARNING OBJECTIVES

- By the end of this course, learners will be able to:

- Carry out data cleaning and processing, and descriptive epidemiological analyses (including commonly used data visualizations);

- Create automated data products (e.g., epidemiologic summaries);

- Design and carry out a data collation plan that is consistent with proposed analysis plan;

- Explain when it is most appropriate to program analyses and automates tasks using R;

- Find and appraise possible solutions to R programming challenges

DAY 3: PRE-COURSE LEARNING

Activity	Description
Context	Canadian Chronic Disease Indicators
Refresher	Pre-Course Self-Study Module

THE MAP

Now, with the treasure in sight, we only have to navigate past

- “The forest of advanced data-processing palms”:
  - Appending and merging
- Test your analysis planning and troubleshooting skills in open-water

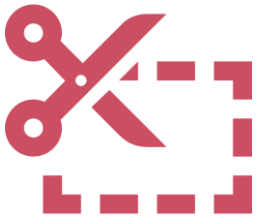
10 Considerations for Data Analysis

1. Read data into R

Consider (1) the type of data, (2) how they are stored, and (3) the size of the dataset.

2. Format the data

Long or wide format? Plan the analysis keeping in mind what your needs are.



3. Select data

Decide what data you'll need in advance from the dataset: subsets vs. filtered dataset. Consider pros and cons of the approach.

4. Modify data

Consider (1) what variables you will need, (2) calculations you will perform, and (3) algorithms needed to create the variables.

5. Link data

Assign a key variable(s) that you will link datasets on. Consider the join type you will need: inner, left, right, full, semi, or anti join.

6. Analyse data

Consider grouping data meaningfully in performing calculations and obtaining descriptive statistics. Evaluate outputs and interpret summaries.

7. Visualise data

Use tables or charts, and consider the best way to display the information.

8. Report findings

Communicate findings to those who need the information created from your work. Consider the audience you are communicating with.

9. Celebrate!

Preferably with cake!

10. Review

Review, revise, and re-do as needed. Often we can go back and simplify/optimise. This is important especially if the code will be used again.

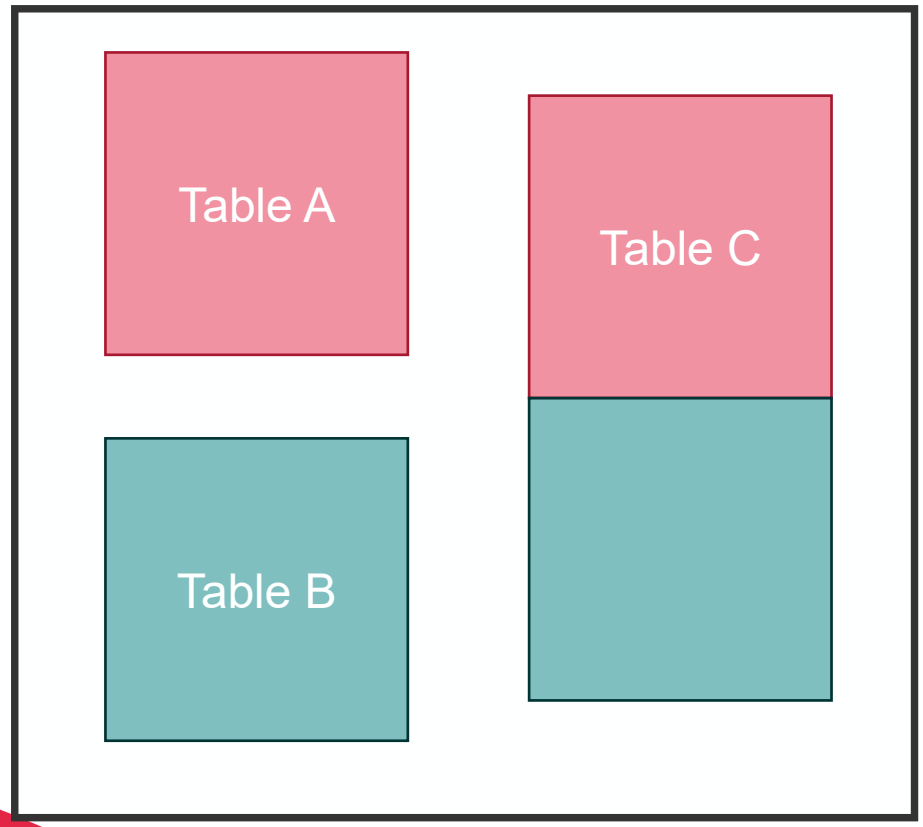
LINKING AND MERGING DATA

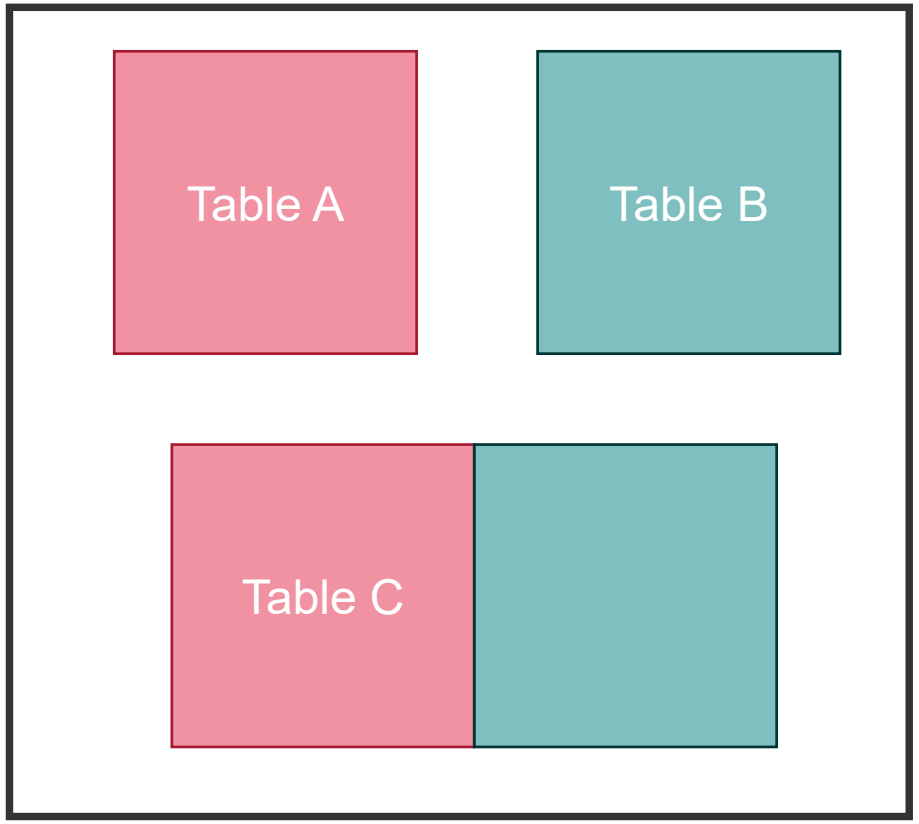
Putting it all together, just like an imaginary jigsaw puzzle

APPENDING DATA

REVIEW

- *Appending* allows us to attach one table to another table:





REVIEW

- Important considerations for appending data:
  1. *The data classes are you appending*
  2. *The column headers*
  3. *The table sizes*
  4. *Ensuring that you have unique key variable(s) between data sets* ★
  5. *Checking to make sure the operation worked as expected*

- Which of the above considerations does **not** belong in the above list?

APPENDING DATA

- Appending data refers simply to adding *some data* to *some other data* to create longer/wider (more) data. (What could be better!?)

APPENDING DATA

- At its simplest, appending data in R can be done using the *combine function:* c()
- For example:

```
> "cat"
[1] "cat"
> "dog"
[1] "dog"
> c("cat", "dog")
[1] "cat" "dog"
```

APPENDING DATA

- You can also use c() to append **objects** together!
- For example:

```
> cats <- c("siamese", "tabby")
> dogs <- c("bulldog", "terrier")
> pets <- c(cats, dogs)
> pets
[1] "siamese" "tabby"   "bulldog" "terrier"
```

- What happens if you use c() to append different data types?
  - E.g., characters and numbers?

APPENDING DATA

- What happens if you use c() to append different data types?
  - E.g., characters and numbers?

```
> dogs <- c("bulldog", "terrier")
> numbers <- c(1, 2, 3, 4)
> assorted <- c(dogs, numbers)
> assorted
[1] "bulldog" "terrier" "1"       "2"       "3"       "4"
```

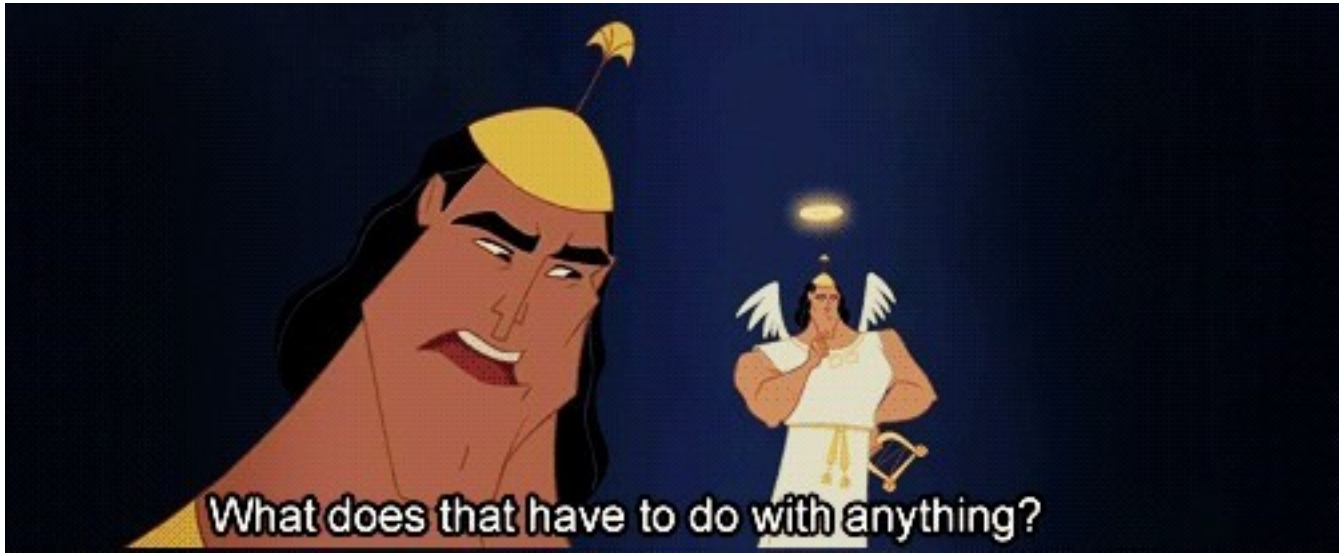
- This works! But what do you notice about the numbers?
  - Are they still numeric?
  - How could we test this?

APPENDING DATA

- Appending objects is useful, but it is important to be aware of what types of data you're left with

```
> is.numeric(assorted)
[1] FALSE
> is.numeric(assorted[3])
[1] FALSE
> is.numeric(assorted[4])
[1] FALSE
> is.numeric(assorted[5])
[1] FALSE
> is.numeric(assorted[6])
[1] FALSE
> class(assorted)
[1] "character"
```

WHY DOES THIS MATTER?



Post in the chat an example of where you think changing data classes could lead to issues!

APPENDING TABLES

- To append one table to the bottom of another in R, we can use the following function:
  - bind_rows()
  - E.g.,

```
> table1
  x1 x2 x3
1  1  4  7
2  2  5  8
3  3  6  9
> table2
  x1 x2 x3
1  3  6  9
2  2  5  8
3  1  4  7
```

```
> bind_rows(table1, table2)
  x1 x2 x3
1  1  4  7
2  2  5  8
3  3  6  9
4  3  6  9
5  2  5  8
6  1  4  7
```

- Column names (e.g., x1, x2, x3) are ***conserved*** in this operation
- What happens if we combine tables with different column names?

APPENDING TABLES

- What happens if we try to bind_rows() two tables with different column names?

```
> table1
  x1 x2 x3
1  1  4  7
2  2  5  8
3  3  6  9
```

```
> table3
  x1 x3 x4
1  1  4  8
2  2  2  9
3  3  5  1
```

APPENDING TABLES

- What happens if we try to bind_rows() two tables with different column names?

```
> table1
  x1 x2 x3
1  1  4  7
2  2  5  8
3  3  6  9
```

```
> table3
  x1 x3 x4
1  1  4  8
2  2  2  9
3  3  5  1
```

```
> bind_rows(table1, table3)
  x1 x2 x3 x4
1  1  4  7 NA
2  2  5  8 NA
3  3  6  9 NA
4  1 NA  4  8
5  2 NA  2  9
6  3 NA  5  1
```

APPENDING TABLES

- The same way we can bind_rows() to make tables *longer*, we can also bind_cols() to make them *wider*

```
> table1
  x1 x2 x3
1  1  4  7
2  2  5  8
3  3  6  9
```

```
> table4
  x4 x5 x6
1  3  2  1
2  1  5  4
3  4  8  9
```

```
> bind_cols(table1, table4)
  x1 x2 x3 x4 x5 x6
1  1  4  7  3  2  1
2  2  5  8  1  5  4
3  3  6  9  4  8  9
```

- Just as column headers were preserved in bind_rows(), row names are preserved in bind_cols()

QUESTION TIME

- What do you think happens if you try to bind_cols() two tables that have the *same* column headers?

QUESTION TIME

- What do you think happens if you try to bind_cols() two tables that have the *same* column headers?

```
> bind_cols(table1, table5)
New names:
* x1 -> x1...1
* x2 -> x2...2
* x3 -> x3...3
* x1 -> x1...4
* x2 -> x2...5
* ...
  x1...1 x2...2 x3...3 x1...4 x2...5    x3...6
1      1      4      7    dog   blue chocolate
2      2      5      8    cat yellow   vanilla
3      3      6      9   fish    red   rainbow
```

QUESTION TIME

- What happens when you try to append tables of different data types (*i.e.*, data *classes*) using bind_rows or bind_cols?

```
> table1
   x1 x2 x3
1   1  4  7
2   2  5  8
3   3  6  9
```

```
> table5
     x1     x2        x3
1   dog   blue chocolate
2   cat yellow   vanilla
3 fish    red   rainbow
```

QUESTION TIME

- What happens when you try to append tables of different data types (*i.e.*, data *classes*) using bind_rows or bind_cols?

```
> bind_rows(table1, table5)
Error: Can't combine `..1$x1` <double> and `..2$x1` <character>.
```

```
> bind_cols(table1, table5)
New names:
* x1 -> x1...1
* x2 -> x2...2
* x3 -> x3...3
* x1 -> x1...4
* x2 -> x2...5
* ...
  x1...1 x2...2 x3...3 x1...4 x2...5    x3...6
1      1      4      7    dog   blue chocolate
2      2      5      8    cat yellow   vanilla
3      3      6      9   fish    red   rainbow
```

TRIVIA: SAILING THE SEVEN SEAS

Feature	Ocean
1. This ocean sits in the oldest existing ocean basin	A – Southern
2. This ocean has the highest salinity	B – Pacific
3. This ocean is the shallowest	C – Atlantic
4. This ocean has the longest continuous ocean current	D – Arctic

Share your answers in the chat box!

NAVIGATING TRICKY WATERS


How people think programming looks


How it really looks 90% of the time

EXERCISE: NAVIGATING TRICKY WATERS

Inspired by course participants in previous run-throughs

- What to do when you're met with an analysis task that you don't know how to complete?

Take 10 minutes to draft an action plan for how you would create or replicate a pirate's sea shanty in R.

Please review Q1, Q2, and Q3. We will work on Q4 as a group.

EXERCISE: NAVIGATING TRICKY WATERS

- **Q1.** What is a sea shanty? What resources will you require to find one and replicate it? How will you transcribe the music? Do you require additional resources to figure out how to do this?

- **Q2.** Can R play music? Are there packages that support this?

- **Q3.** Can you find examples of others' code to help with these tasks? Explain how you would go about this?

EXERCISE: NAVIGATING TRICKY WATERS

- **Q1.** What is a sea shanty? What resources will you require to find one and replicate it? How will you transcribe the music? Do you require additional resources to figure out how to do this?
  - A sea shanty is a simple folk song that was designed to be sung at sea while working without instrumental accompaniment.
  - You will need to find an example of a sea shanty; and search for the melody notes: e.g., “sea shanty sheet music free”
  - If you can’t find free resources: see the pdf sheet music for “Wellerman” on Dropbox
    - Note: may need a resource for how to read sheet music: this can be found here
    - https://www.instructables.com/How-to-Read-Sheet-Music-for-Beginners/

EXERCISE: NAVIGATING TRICKY WATERS

- **Q2.** Can R play music? Are there packages that support this?

- There are several packages available in R to help program and code music; check out the following links:
  - https://cran.r-project.org/web/packages/audio/index.html
  - https://cran.r-project.org/web/packages/sound/index.html
  - https://cran.r-project.org/web/packages/music/index.html
  - https://cran.r-project.org/web/packages/tuneR/tuneR.pdf
  - https://cran.r-project.org/web/packages/gm/index.html

EXERCISE: NAVIGATING TRICKY WATERS

- **Q3.** Can you find examples of others' code to help with these tasks? Explain how you would go about this?

- A great example/tutorial can be found here: https://towardsdatascience.com/compose-and-play-music-in-r-with-the-rmusic-package-b2afa90761ea
- An older stack overflow post with instructions and explanation: https://stackoverflow.com/questions/31782580/how-can-i-play-birthday-music-using-r

EXERCISE: NAVIGATING TRICKY WATERS

- **Q4.** Briefly, using the answers to the prompts in Q1-3, list or sketch out what your analysis plan will look like for completing this task.

  - Step 1:
  - Step 2:
  - Step 3:
  - Etc.

Step 1:
Find an R package that can handle music; preferably one with a tutorial as well (e.g., "Rmusic").

Step 2:
Identify inputs for Rmusic package: E.g., (1) notes; (2) duration.

Step 3:
Investigate Sea Shanties; select one and find sheet music for this.

Step 4:
Use online resource to translate sheet music into something readable for R: E.g., notes and duration.

Step 5:
Install required R packages; translate sheet music into R script.

Step 6:

DATA JOINS

APPENDING TABLES

- Often, you'll want to append one table with another to create a large, single table

x1	x2	x3	x4
1	5	1	4
8	3	6	4

+

x1	x2	x3	x4
5	8	7	9
1	4	8	2

x1	x2	x3	x4
1	5	1	4
8	3	6	4
5	8	7	9
1	4	8	2

TIDY MERGING IN R

- What if you want to *combine* two tables that contain different data for the same individuals?
  - Could you **append** these datasets? Would you try **bind_col()** or **bind_row()**?

Name	# Desserts eaten per week
MAB	4
Ben	8
Joanne	2

Name	# Cavities 2010-2019
Ben	14
Joanne	1
MAB	8

TIDY MERGING IN R

- Yes, you could.. But *should you?*
- When appending tables, *order matters.* Therefore, if your tables aren't sorted exactly the same, or contain different numbers of rows, they may combine *incorrectly.*

Name	# Desserts eaten per week	# Cavities 2010-2019
MAB	4	14
Ben	8	1
Joanne	2	8

TIDY MERGING IN R

- Using a merge (or join) operation ensures that your variables match up with the correct *key* variable
  - In this example, "*Name*" is the key we want to match on

Name	# Desserts eaten per week
MAB	4
Ben	8
Joanne	2

Name	# Cavities 2010-2019
Ben	14
Joanne	1
MAB	8

TIDY MERGING IN R

- Using a merge (or join) operation ensures that your variables match up with the correct *key* variable
  - In this example, "*Name*" is the key we want to match on

```
> desserts %>%  filter(Name %in% c("MAB", "Ben", "Joanne"))
# A tibble: 3 x 2
  Name   Desserts.wk
  <chr>        <dbl>
1 MAB              4
2 Joanne           2
3 Ben              8
```

```
> cavities %>%  filter(Name %in% c("MAB", "Ben", "Joanne"))
# A tibble: 3 x 2
  Name   Cavities.10yr
  <chr>          <dbl>
1 MAB                8
2 Ben               14
3 Joanne             1
```

TIDY MERGING IN R

- Using R (tidyverse) we can call the function full_join() to combine the two tables, matching on a key variable (e.g., “Name”)

```
> full_join(desserts, cavities, by = "Name") %>%
+ filter(Name %in% c("MAB", "Ben", "Joanne"))
# A tibble: 3 x 3
  Name   Desserts.wk Cavities.10yr
  <chr>        <dbl>         <dbl>
1 MAB              4             8
2 Joanne           2             1
3 Ben              8            14
```

JOINING ON KEY VALUES

- Using table joins makes sure that the information you're adding stays with the correct key identifier

Name	# Desserts eaten per week
MAB	4
Ben	8
Joanne	2

Name	# Cavities 2010-2019
Ben	14
Joanne	1
MAB	8

OTHER EXAMPLES OF TABLE JOINS

- Excel?
  - vlookup; hlookup
- Stata?
  - merge(1:m); merge(m:m)
- SAS?
  - Proc SQL
    - Select – From – Where
  - MERGE
    - Data statement option
- Others?

TYPES OF TIDY JOINS IN R

	Join Type	Definition
Mutating joins	**Full**	Returns all data from the left and right side of the statement regardless of whether or not there are matches in the other table.
	Left	A link between two tables where all data from the left side of the statement is returned with only data that matches from the right.
	Right	A link between two tables where all data from the right side of the statement is returned with only data that matches from the left.
	Inner	A link between two tables based on equality between values (e.g., key variables) in a column of tables on the left and right side of the statement.
Filtering joins	**Anti**	Returns rows from the left side of the statement for which there is no match on the right (e.g., **not** in).
	Semi	Returns all rows from the left side of the statement where there are matching values in the right side, keeping only columns from the left

Left table (x)

Name	# Desserts
MAB	4
Ben	8
Joanne	2
Mirna	7

Right table (y)

Name	# Cavities
Ben	14
Joanne	1
MAB	8
Alanah	0

"Mutating joins"

"Filtering joins"

Inner Join

Name	# Desserts	# Cavities
MAB	4	8
Ben	8	14
Joanne	2	1

Left Join

Name	# Desserts	# Cavities
MAB	4	8
Ben	8	14
Joanne	2	1
Mirna	7	NA

Right Join

Name	# Desserts	# Cavities
MAB	4	8
Ben	8	14
Joanne	2	1
Alanah	NA	0

Full Join

Name	# Desserts	# Cavities
MAB	4	8
Ben	8	14
Joanne	2	1
Alanah	NA	0
Mirna	7	NA

Anti Join

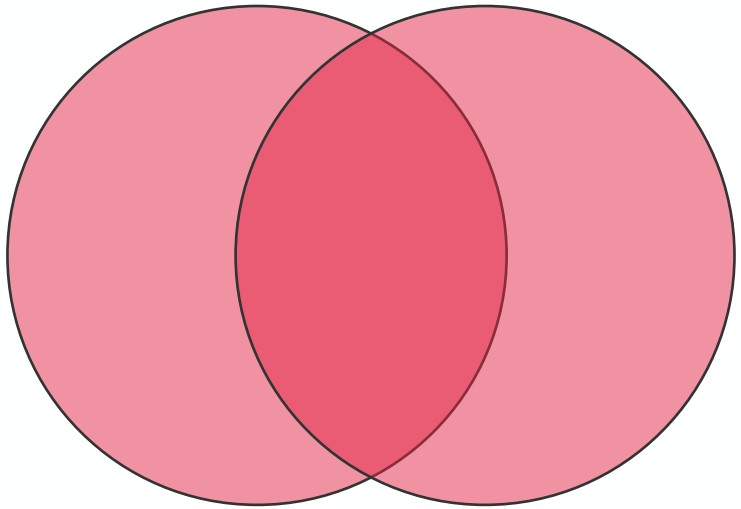
Name	# Desserts
Mirna	7

Semi Join

Name	#Desserts
MAB	4
Ben	8
Joanne	2

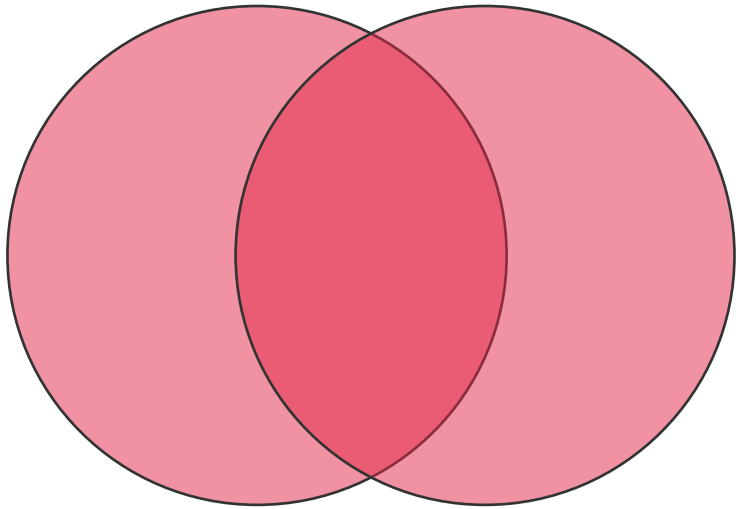
TYPES OF TIDY JOINS IN R:

I am useful when you want to combine everything from both tables



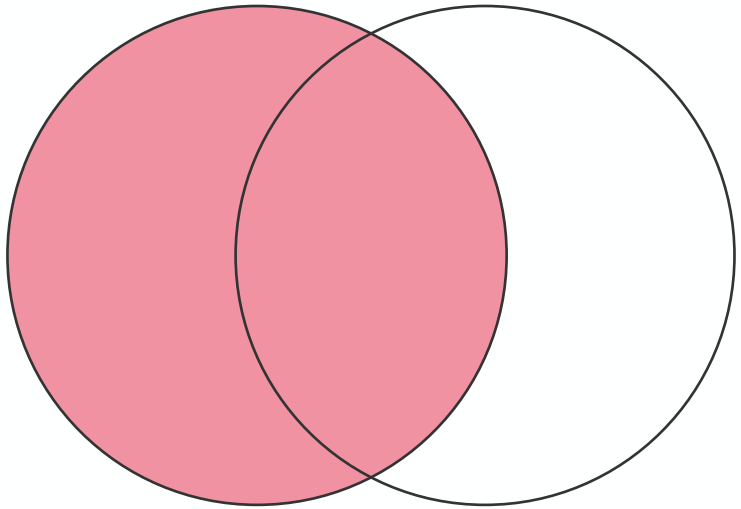
TYPES OF TIDY JOINS IN R:

full_join: useful when you want to combine everything from both tables



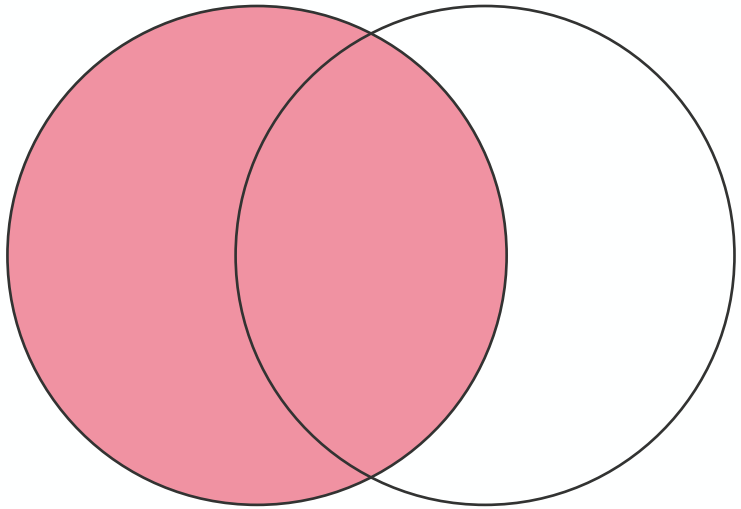
TYPES OF TIDY JOINS IN R:

I join only on key values present in the first table. Duplicates will be appended to the bottom of the resulting table.



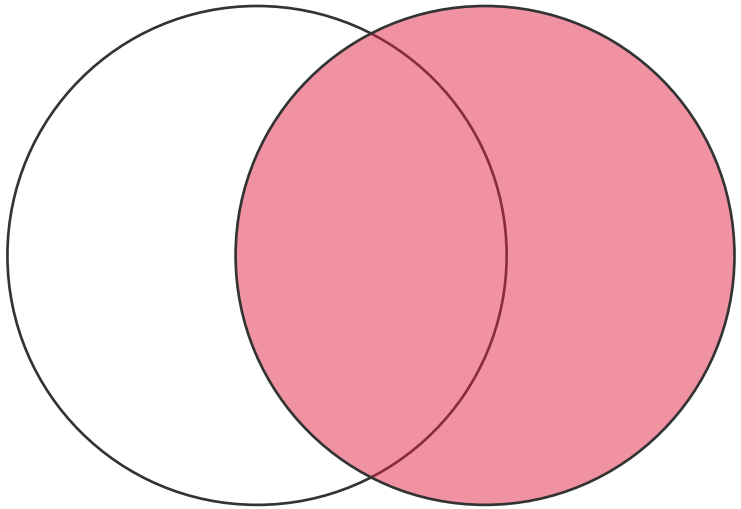
TYPES OF TIDY JOINS IN R:

left_join: only join on key values present in the first table. Duplicates will be appended to the bottom of the resulting table.



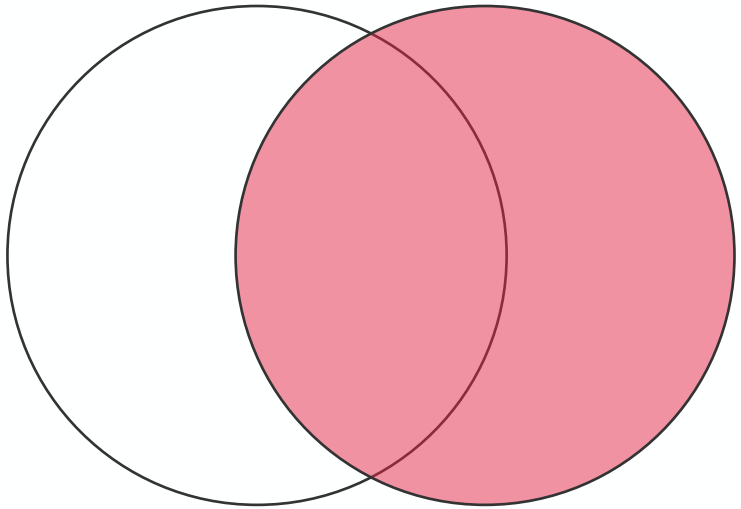
TYPES OF TIDY JOINS IN R:

I join only on key values present in the second table. Duplicates are added to the end of the table.



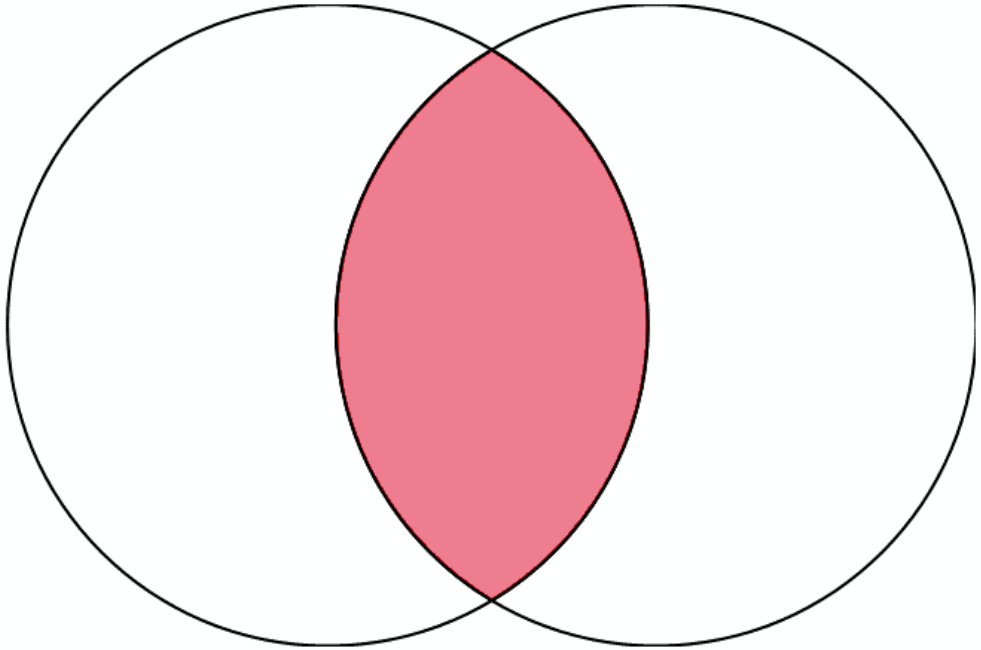
TYPES OF TIDY JOINS IN R:

right_join: Only join on key values present in the second table. Duplicates added to the end of the table.



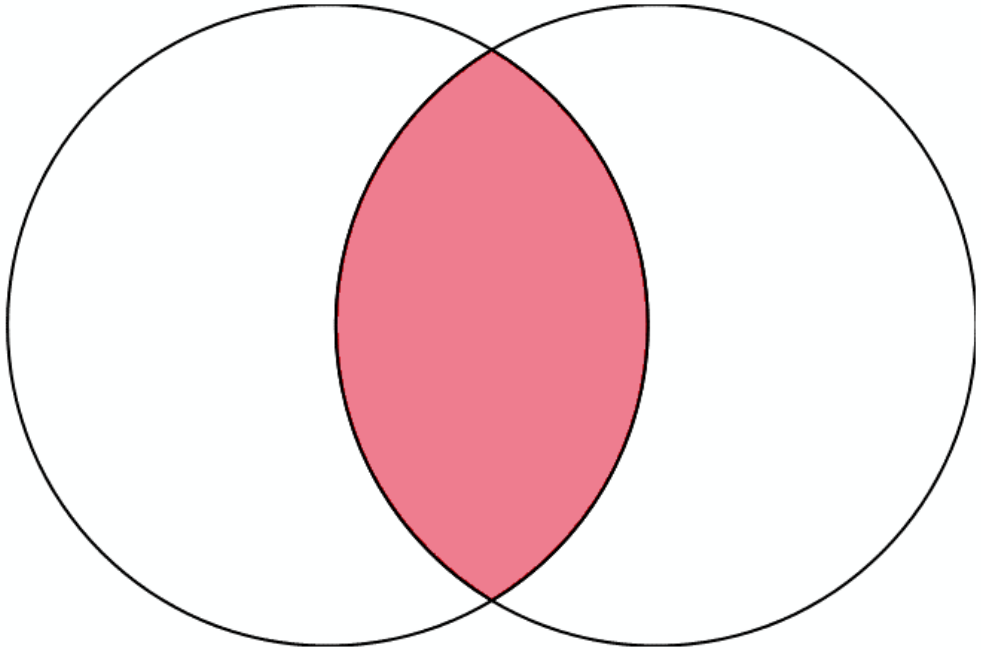
TYPES OF TIDY JOINS IN R:

I produce a table from keys present in *both tables only*.



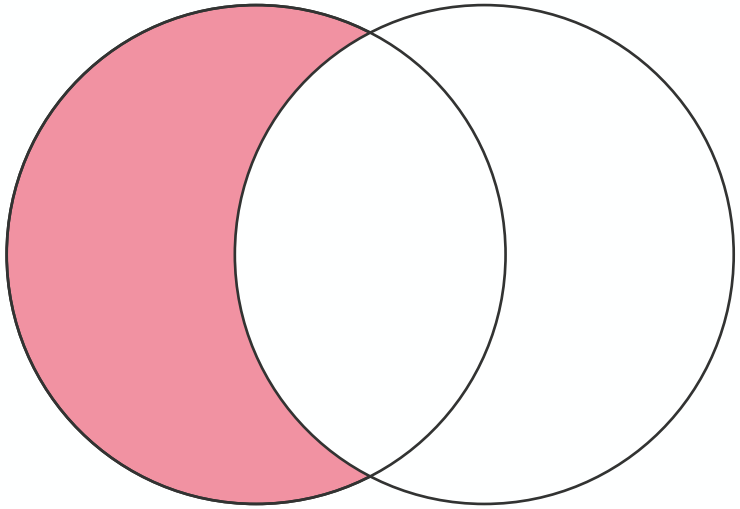
TYPES OF TIDY JOINS IN R:

inner_join: Resulting table from keys present in *both tables only*.



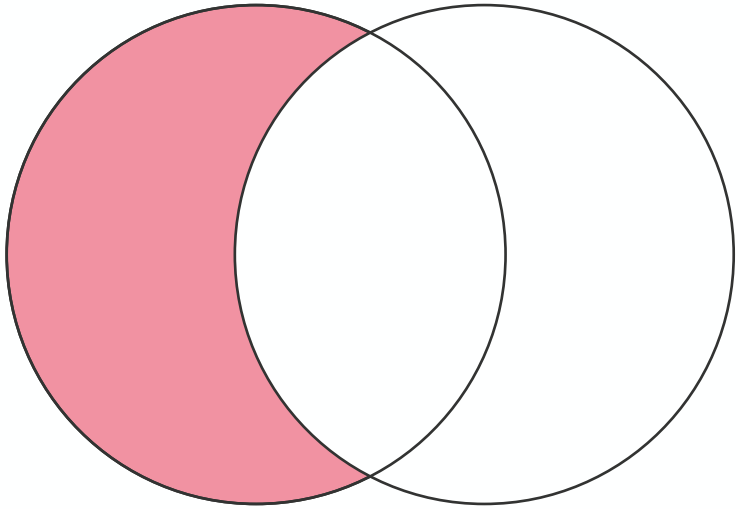
TYPES OF TIDY JOINS IN R:

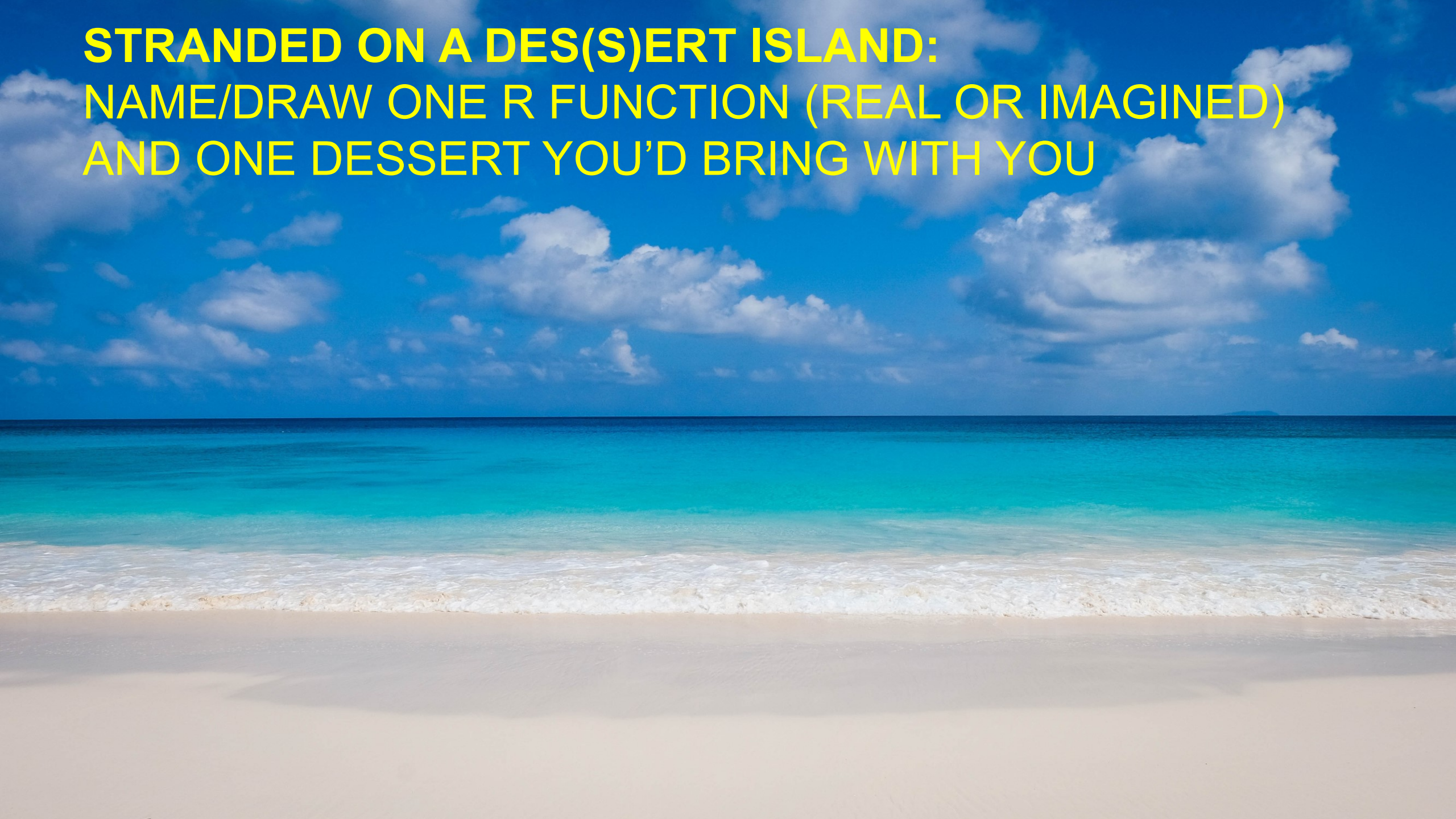
I return a table from key values in the left table *that do not have a match* between the two tables.



TYPES OF TIDY JOINS IN R:

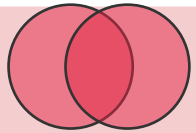
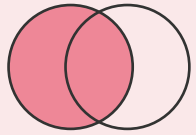
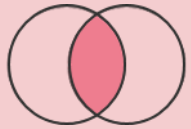
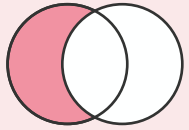
anti_join: Returns values from left table only from key values *that do not have a match* between the two tables.




STRANDED ON A DES(S)ERT ISLAND:
NAME/DRAW ONE R FUNCTION (REAL OR IMAGINED)
AND ONE DESSERT YOU'D BRING WITH YOU

GROUP DISCUSSION

- Take 3 minutes and discuss via private chat with your partner examples of a time when you may have used one of the following join types in your work.
- If you don’t have an example, please brainstorm an application.
- We'll then debrief as a large group

Join type	
Full join	
Left (or right) join	
Inner join	
Anti join	

DEMO

R Markdown and HTML Notebooks

MINI STRETCH BREAK

Take 5 minutes to batten down the hatches and check the rigging!

WRAP UP

DAY 3 EXERCISE

CONCEPTUAL

- Participants will need to combine datasets from hospital admissions records, provincial registry and physician billings in order to apply administrative case definitions and identify incidence of asthma from the datasets
  - Discuss the **source** of each dataset; what variables are contained within?
  - Discuss the case definition: what information do you need to determine whether someone is a case or not

TECHNICAL

- Follow a schematic of how datasets need to be combined in order to apply case definition
- Walk through key-value pairs; how to prepare and append datasets to be ready for merging
- Applying a case definition after merging to create the final dataset of cases

NEED HELP OR CLARIFICATION?

Remember: Slack channel for this training session

- Phacrusers.slack.com -> intro_r_oct2023cohort

- **Use the Slack channel to:**
  - Connect with your peers
  - Find materials shared throughout the course
  - Ask questions about the materials/exercises to your peers, facilitators and subject matter experts
  - Help others who have questions if you know the answer!

REMINDER

- There are no new course materials for you to download or review for tomorrow

- Day 4 focus is on review and supported practical application

Questions?

EXERCISE DEMO

Getting Started

Data Collation