Module 3: R

Wrangling

Instructor: Anjali Silva, PhD

Data Sciences Institute, University of Toronto

2022

Course Documents

- Visit: https://github.com/anjalisilva/IntroductionToR
- All course material will be available via IntroductionToR GitHub repository (https://github.com/anjalisilva/IntroductionToR). Folder structure is as follows:
    - Lessons - All files: This folder contains all files.
    - Lessons - Data only: This folder contains data only.
    - Lessons - Lesson Plans only: This folder contains lesson plans only.
    - Lessons - PDF only: This folder contains slide PDFs only.
    - README - README file
    - .gitignore - Files to ignore specified by instructor

Course Contacts

- Instructor: Anjali Silva Email: a.silva@utoronto.ca (Must use the subject line DSI-IntroR. E.g., DSI-IntroR: Inquiry about Lecture I.)
- TA: see GitHub

Overview

- Importing data (Wickham and Grolemund, 2017, Chapter 11; Timbers et al. 2021, Chapter 2) & Saving data
- Pivot (Wickham and Grolemund, 2017, Chapter 12; Timbers et al. 2021, Chapter 3.4)
- Joining data (Wickham and Grolemund, 2017, Chapter 13)
- data.table (Wiley and Wiley, 2020, Chapter 7; https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html)

Data Import

Packages and functions for reading in different file types

CSV files

```
readr::read_csv()
```

Excel workbooks

```
readxl::read_excel()
```

Packages and functions for reading in different file types

SPSS files

```
haven::read_sav()
```

Stata files

```
haven::read_dta()
```

SAS files

```
haven::read_sas()
```

Specifications for import

When the separator isn't a comma:

```
read_csv("mydata.csv", sep = ";")
```

When the first two lines are metadata:

```
read_csv("mydata.csv", skip = 2)
```

When there are no column names:

```
read_csv("mydata.csv", col_names = FALSE)
```

Import issues

The reader function guesses what data type each column is. This might not always work the way you want. You can specify column types if necessary:

```
read_csv("mydata.csv",
         col_types = cols(.default = col_character())
```

Writing data

```
write_csv(mydata, "mydata.csv")
```

Pivot

Tidy data

Rules for tidy data:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

Pivoting will help you get to tidy data form.

Tidy data

Let's say we have a dataset with the weights of two pet cats:

```
wide_data <- tibble(
  year = c(2017, 2018, 2019, 2020, 2021, 2022),
  milo = c(4, 6.3, 8, 7.9, 8.1, 8.1),
  kitty = c(15.6, 15.9, 14, 12.2, 10.9, 9.9)
)

wide_data
```

```
## # A tibble: 6 × 3
##    year  milo kitty
##   <dbl> <dbl> <dbl>
## 1  2017   4    15.6
## 2  2018   6.3  15.9
## 3  2019   8    14
## 4  2020   7.9  12.2
## 5  2021   8.1  10.9
## 6  2022   8.1   9.9
```

Tidy data

To make the data tidy, we will use a function called pivot_longer:

```
long_data <- wide_data %>%
  pivot_longer(cols = c("milo", "kitty"),
               names_to = "cat",
               values_to = "weight")
long_data
```

```
## # A tibble: 12 × 3
##     year cat   weight
##    <dbl> <chr>  <dbl>
##  1  2017 milo     4
##  2  2017 kitty   15.6
##  3  2018 milo     6.3
##  4  2018 kitty   15.9
##  5  2019 milo     8
##  6  2019 kitty   14
##  7  2020 milo     7.9
##  8  2020 kitty   12.2
##  9  2021 milo     8.1
## 10  2021 kitty   10.9
## 11  2022 milo     8.1
## 12  2022 kitty    9.9
```

Tidy data

You can also reverse:

```
long_data %>%
  pivot_wider(names_from = "cat",
              values_from = "weight")
```

```
## # A tibble: 6 × 3
##    year  milo kitty
##   <dbl> <dbl> <dbl>
## 1  2017   4    15.6
## 2  2018   6.3  15.9
## 3  2019   8    14
## 4  2020   7.9  12.2
## 5  2021   8.1  10.9
## 6  2022   8.1   9.9
```

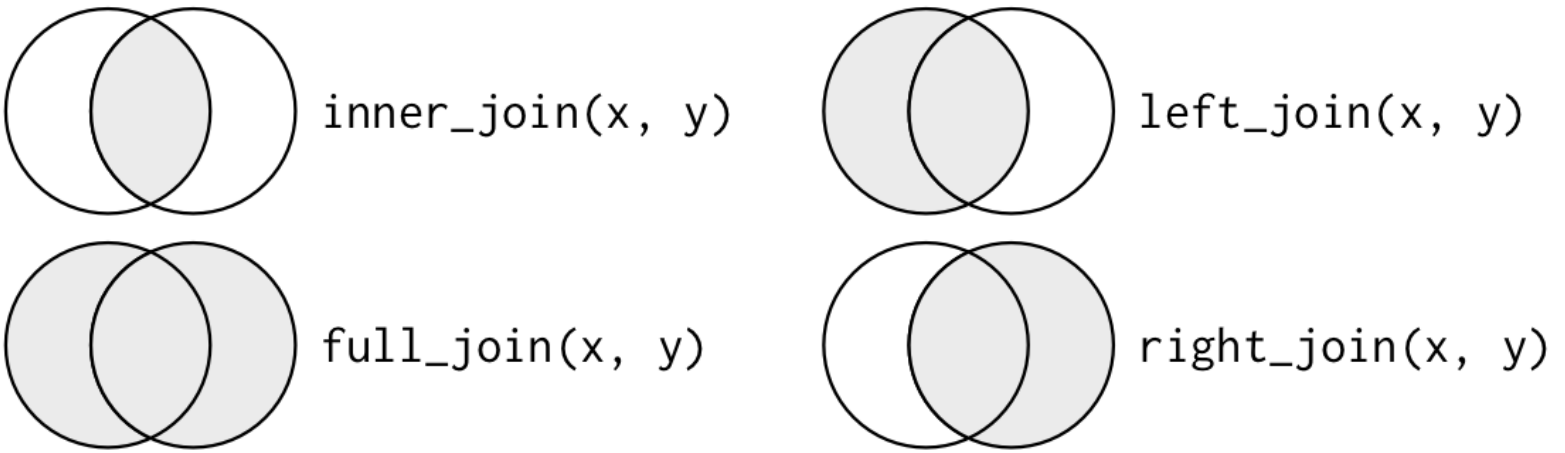
This may be useful when making a summary table to display.

Join

Mutating joins

Joins combine tables based on an identified key, usually a variable that the two tables share in common.

```
left_join(), right_join(), full_join(), inner_join()
```



Keys

1. Primary keys uniquely identify an observation in its table
2. Foreign keys uniquely identify an observation in another table

Primary keys may match foreign keys, if the variable is present in both tables and has the same name.

Primary and foreign keys have a specific relation: it could be one-to-one, or one-to-many.

Example using toy data

We will make two small datasets, each with year as a variable. Year will be the key we use to join.

```
employment <- tibble(year = c(1990, 1991, 1992, 1994),
                     rate = c(.05, .02, .04, .02))
employment
```

```
## # A tibble: 4 × 2
##    year  rate
##   <dbl> <dbl>
## 1  1990  0.05
## 2  1991  0.02
## 3  1992  0.04
## 4  1994  0.02
```

Example using toy data

```
housing <- tibble(date = c(1991, 1992, 1993, 1994, 1995),
                  rate = c(0.89, 0.6, 0.75, 0.88, 0.9))
housing
```

```
## # A tibble: 5 × 2
##    date  rate
##   <dbl> <dbl>
## 1  1991  0.89
## 2  1992  0.6
## 3  1993  0.75
## 4  1994  0.88
## 5  1995  0.9
```

Inner Join

The rows correspond to years that are present in both `employment` and `housing`:

```
employment %>%
  inner_join(housing, by = c("year" = "date"))
```

```
## # A tibble: 3 × 3
##    year rate.x rate.y
##   <dbl>  <dbl>  <dbl>
## 1  1991   0.02   0.89
## 2  1992   0.04   0.6
## 3  1994   0.02   0.88
```

Because both tables have a variable named `rate`, the joined table has columns named `rate.x` from the left table (employment) and `rate.y` from the right table (housing).

Left Join

The rows correspond to years that are present in both `employment` but not necessarily `housing`:

```
employment %>%
  left_join(housing, by = c("year" = "date"))
```

```
## # A tibble: 4 × 3
##    year rate.x rate.y
##   <dbl>  <dbl>  <dbl>
## 1  1990   0.05  NA
## 2  1991   0.02   0.89
## 3  1992   0.04   0.6
## 4  1994   0.02   0.88
```

Missing values are filled with `NA`.

Right Join

The rows correspond to years that are present in `housing` but not necessarily `employment`:

```
employment %>%
  right_join(housing, by = c("year" = "date"))
```

```
## # A tibble: 5 × 3
##    year rate.x rate.y
##   <dbl>  <dbl>  <dbl>
## 1  1991   0.02   0.89
## 2  1992   0.04   0.6
## 3  1994   0.02   0.88
## 4  1993  NA      0.75
## 5  1995  NA      0.9
```

Full Join

The rows correspond to years that are present in `employment`or `housing`:

```
employment %>%
  full_join(housing, by = c("year" = "date"))
```

```
## # A tibble: 6 × 3
##    year rate.x rate.y
##   <dbl>  <dbl>  <dbl>
## 1  1990   0.05  NA
## 2  1991   0.02   0.89
## 3  1992   0.04   0.6
## 4  1994   0.02   0.88
## 5  1993  NA      0.75
## 6  1995  NA      0.9
```

Filtering joins

Semi joins keep all observations in the employment table that have a match in the housing table:

```
employment %>%
  semi_join(housing, by = c("year" = "date"))
```

```
## # A tibble: 3 × 2
##    year  rate
##   <dbl> <dbl>
## 1  1991  0.02
## 2  1992  0.04
## 3  1994  0.02
```

Filtering joins

Anti joins drop all observations in employment that have a match in housing:

```
employment %>%
  anti_join(housing, by = c("year" = "date"))
```

```
## # A tibble: 1 × 2
##    year  rate
##   <dbl> <dbl>
## 1  1990  0.05
```

data.table

Introduction

data.tables are faster and more memory efficient than data.frames, making them better suited for large operations on large data sets.

They are keyed, which makes it possible to use binary search. A key is an index, created from column(s) in the data.table. It may or may not be unique.

data.table

```
data(iris)

dt_iris <- as.data.table(iris)

dt_iris
```

```
##      Sepal.Length Sepal.Width Petal.Length
##   1:          5.1         3.5          1.4
##   2:          4.9         3.0          1.4
##   3:          4.7         3.2          1.3
##   4:          4.6         3.1          1.5
##   5:          5.0         3.6          1.4
##  ---
## 146:          6.7         3.0          5.2
## 147:          6.3         2.5          5.0
## 148:          6.5         3.0          5.2
## 149:          6.2         3.4          5.4
## 150:          5.9         3.0          5.1
##      Petal.Width   Species
##   1:         0.2    setosa
##   2:         0.2    setosa
##   3:         0.2    setosa
##   4:         0.2    setosa
##   5:         0.2    setosa
```

data.table

You can see how many tables are stored and what size they are:

```
tables()
```

```
##       NAME NROW NCOL MB
## 1: dt_iris  150    5  0
##                                                                  COLS
## 1: Sepal.Length,Sepal.Width,Petal.Length,Petal.Width,Species
##    KEY
## 1:
## Total: 0MB
```

data.table

To get information about our data, we can use sapply and class:

```
sapply(dt_iris, class)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    "numeric"    "numeric"    "numeric"    "numeric"
##      Species
##     "factor"
```

data.table

To check if the data.table has a key:

```
haskey(dt_iris)
```

```
## [1] FALSE
```

To set a key:

```
setkey(dt_iris, Species)
haskey(dt_iris)
```

```
## [1] TRUE
```

General form

datatable[i, j, k]

Where:

- from datatable
- i is selected and/or filtered
- j is calculated
- k is grouped by

Subsetting rows

```
dt_iris[Petal.Width > 0.5]
```

Selecting columns

Selecting a column to return as a vector:

```
dt_iris[, Species]
```

Selecting columns

Selecting a column to return as a data.table:

```
dt_iris[, list(Species)]
```

Computations

Counting the number of cases where Petal.Length plus Petal.Width is greater than 2:

```
dt_iris[, sum((Petal.Length + Petal.Width) > 2)]
```

Combining subsetting and computation

Counting the number of cases where Species is "setosa" and Petal.Length plus Petal.Width is greater than 2:

```
dt_iris[Species == "setosa", sum((Petal.Length + Petal.Width) > 2)]
```

Counting observations in current group

```
dt_iris[Species == "setosa", .N]
```

Aggregations

Counting the number in each Species group:

```
dt_iris[, .(.N), by = .(Species)]
```

Sorting

```
dt_iris[Petal.Width > 1, .(mean(Petal.Length)), keyby = .(Species)]
```

Ordering

Ordering by Species and then by reverse Petal.Width:

```
dt_iris[order(Species, -Petal.Width)]
```

Grouping by expression rather than column

Grouping those that have Petal.Length greater/not than 4, and Petal.Width greater/not than 1:

```
dt_iris[, .N, .(Petal.Length > 4, Petal.Width > 1)]
```

Computations for multiple columns

Taking the mean of all columns (except the grouping column, Species) using lapply and .SD:

```
dt_iris[, lapply(.SD, mean), by = Species]
```

Subsetting for each group

Getting the first two rows for all columns, for each Species:

```
dt_iris[, head(.SD, 2), by = Species]
```

Exercises

Exercises

1 - Tidy the data below:

```
data <- tibble(
  group = c("treat", "control"),
  survival = c(17, 11),
  deceased = c(3, 9)
)
```

Exercises

2 - Join the dataset flights to the dataset airlines. What should the key(s) be? What do the different types of joins look like?

```
library(nycflights13)
data("flights")
data("airlines")
```

Exercises

3 - Using flights and data.table, group based on cases that have dep_delay < 0 and those that have arr_delay > 0 and count the number in each group. How many groups/rows are there? How many in each group?

Any questions?