SM-2302 Software for Mathematicians

R3: The `tidyverse`

Dr. Haziq Jamil
Mathematical Sciences, Faculty of Science, UBD
`https://haziqj.ml`

Semester I 2023/24

Preamble

```
# If not installed yet, install them first
library(tidyverse)
library(remotes)

# This may require compilation
remotes::install_github("rstudio/EDAWR")
library(EDAWR)  # to get the data sets: storms, cases, pollution, tb
```

Note that installing EDAWR package may require compilation. For Windows, check out Rtools42. For Macs, I think minimally you will need to install Xcode. Check out this link.

These slides were adapted from the following YouTube playlist.

- Data wrangling with R and the Tidyverse by Garrett Grolemund, RStudio.

I would recommend that you watch them, but note that the video is based on the old version of `tidyverse` packages, so some commands may be deprecated or superseded.

Learning objectives

- Spot the **variables** and **observations** within your data
  - Variables = Columns
  - Observations = Rows
- **Reshape** your data into the layout that works best for R
  - Long vs wide data sets
  - Long → wide using `pivot_wider()`
  - Wide → long using `pivot_longer()`
- *Quickly* derive new variables and observations to explore
  - `mutate()` or `summarise()` to add new variables or summarise data
  - `select()`, `filter()`, `slice()`, `arrange()` to focus and/or reveal information
- Perform **group-wise** summaries to explore hidden levels of information within your data
  - `group_by()`
- Join multiple data sets together
  - `left_join()`, `right_join()`, `inner_join()`, `full_join()`

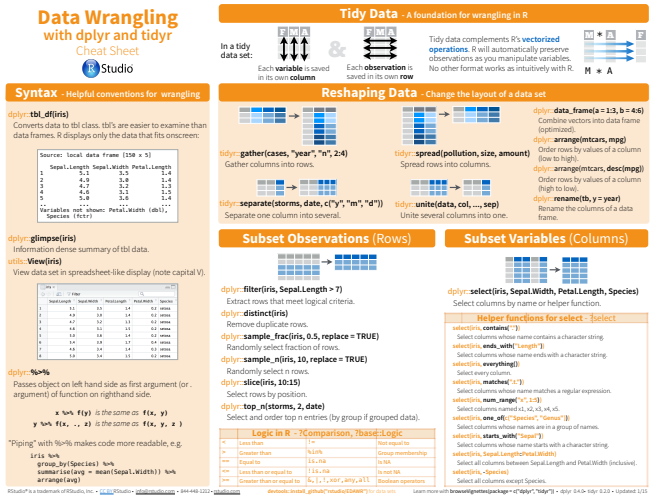
The `tidyverse` package

The `tidyverse` is a collection of R packages designed for data science. We're most interested in the following packages:

1. `tibble`
   A modern re-imagining of `data.frames`.
2. `tidyr`
   Provides a set of functions that help you get to tidy data.
3. `dplyr`
   Provides a set of verbs for data manipulation.

TL;DR–Use the cheat sheet!

Have a look at the Data Wrangling cheat sheet from RStudio.



Universiti Brunei Darussalam

Tidy data

Data comes in various shapes and formats. It's important when you begin your analysis to identify this at the outset.

- What are the variables? How many of them are there?
- What are the observations? How many of them are there?

The goal is to reshape your data into a format that works best in R. The resulting format is called **tidy data**.

Let's have a look at three data sets:

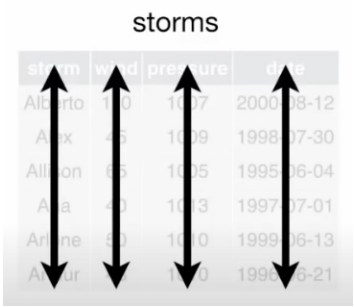
1. `storms` (Wind speed data for six hurricanes)
2. `cases` (Data from the WHO Global Tuberculosis report)
3. `pollution` (Ambient air polution from WHO)

1. `storms` data set

```
EDAWR::storms
```

```
##     storm wind pressure       date
## 1 Alberto  110     1007 2000-08-03
## 2    Alex   45     1009 1998-07-27
## 3 Allison   65     1005 1995-06-03
## 4     Ana   40     1013 1997-06-30
## 5  Arlene   50     1010 1999-06-11
## 6  Arthur   45     1010 1996-06-17
```

- Variables: `storm`, `wind`, `pressure`, `date`
- Observations: 1, 2, 3, 4, 5, 6
- Notice that
  - Each variable is represented by a single column
  - Each row is a single observation



Extracting data by subsetting easily:

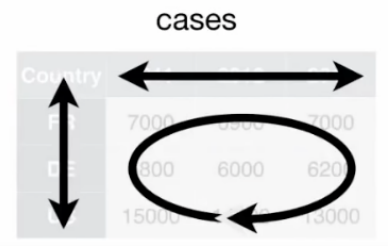
- `storms$storm`
- `storms$wind`
- `storms$pressure`
- `storms$date`

2. cases data set

Wide format

```
EDAWR::cases
```

```
##   country  2011  2012  2013
## 1      FR  7000  6900  7000
## 2      DE  5800  6000  6200
## 3      US 15000 14000 13000
```



- Variables: `country`, `year = c(2011, 2012, 2013)`, `count`
- Observations: $3 \times 3 = 9$
- Notice that
  - Each cell of this data frame corresponds to the `count` for a given `country` and `year`
  - Different from the `storms` data set!

Extracting data by subsetting (not so easy):

- `cases$country`
- `colnames(cases)[-1]`
- `unlist(cases[1:3, 2:4]`

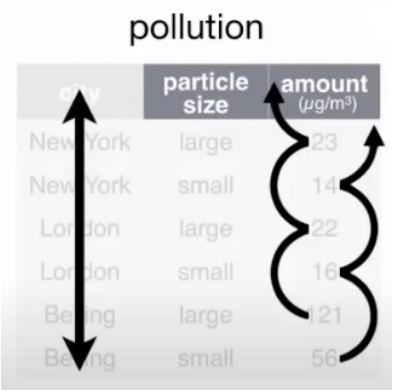
3. `pollution` data set

Long format

```
EDAWR::pollution
```

```
##       city  size amount
## 1 New York large     23
## 2 New York small     14
## 3   London large     22
## 4   London small     16
## 5  Beijing large    121
## 6  Beijing small     56
```

- Variables: `city`, `amount`
- Observations: 1, 2, 3, 4, 5, 6
- Notice that
  - Amount of particles for each city is segregated into two groups: large and small.



Extracting data by subsetting (hard):

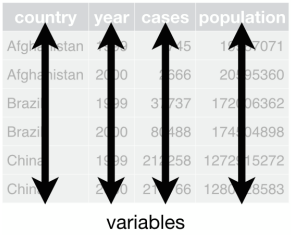
- `pollution$city[c(1, 3, 5)]`
- `pollution$amount[c(1, 3, 5)]`
- `pollution$amount[c(2, 4, 6)]`

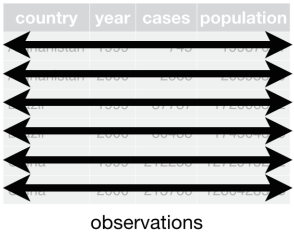
Tidy data

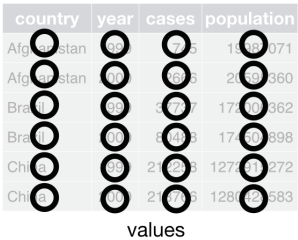
The defining characteristics of tidy data are:

> *Variables in columns, observations in rows, and each type in a table.*

This makes variables easy to access and manipulate (while preserving observations).







Tidy data

Getting tidy data

- `pivot_longer()`
- `pivot_wider()`
- `separate()`
- `unite()`

Data wrangling

Pipeline

Group wrangling

Joining data sets

Getting tidy data

There are two main functions that we will use to reshape the layout of tables:

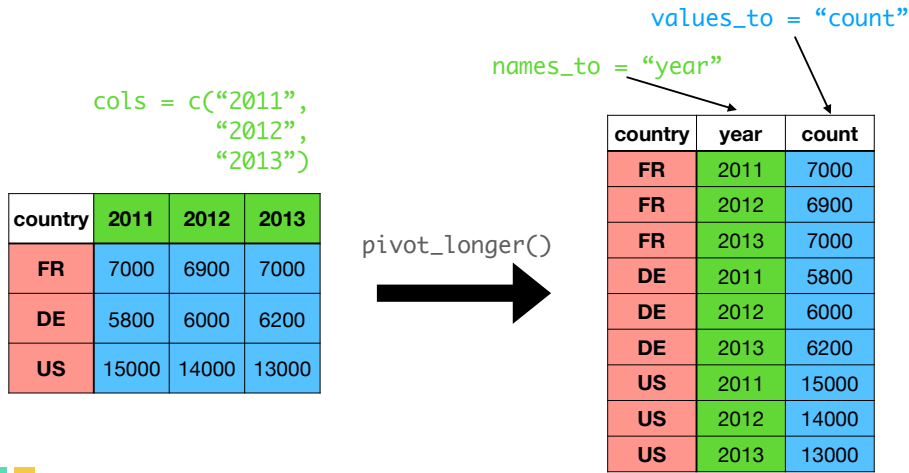
1. `pivot_wider()`
2. `pivot_longer()`

To a lesser degree, these utility functions may be useful too:

1. `separate()`
2. `unite()`

pivot_longer()

This converts data from wide format to long format. In the cases data set, ideally we want to have three columns only: `country`, `year`, and `count`; and each row will be an observation.



pivot_longer() (cont.)

```
cases
```

```
## Error in eval(expr, envir, enclos): obj
```

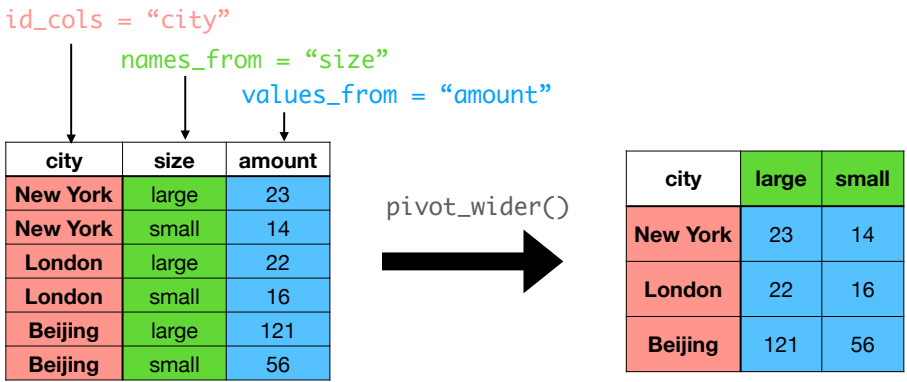
- Select which `cols` to pivot to longer format. Note that these have to be character vectors.
- `names_to` is the name of the new column to store the old columns
- `values_to` is the name of the new column to store the observations

```
pivot_longer(data = cases,
             cols = c("2011", "2012", "2013"),
             names_to = "year",
             values_to = "count")
```

```
## Error in pivot_longer(data = cases, cols = c("20
```

pivot_wider()

This converts data from long format to wide format. In the `pollution` data set, we could instead have a table of city by particle size, and each cell is the amount.



city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14
London	22	16
Beijing	121	56

pivot_longer() (cont.)

```
pollution
```

```
## Error in eval(expr, envir, enclos): object 'pollutio
```

- Select which `id_cols` uniquely identifies each observation
- `names_from` is the name of the column to spread
- `values_from` is the name of column containing the observations

```
pivot_wider(data = pollution,
            id_cols = "city",
            names_from = "size",
            values_from = "amount")
```

```
## Error in pivot_wider(data = pollutio
```

separate()

Turns a single character column into multiple columns.

```
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  lo
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>
##  1 Amy    1975     6    27     0  27.5 -79   tropical depres~
##  2 Amy    1975     6    27     6  28.5 -79   tropical depres~
##  3 Amy    1975     6    27    12  29.5 -79   tropical depres~
##  4 Amy    1975     6    27    18  30.5 -79   tropical depres~
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical depres~
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical depres~
##  7 Amy    1975     6    28    12  33.3 -78   tropical depres~
##  8 Amy    1975     6    28    18  34   -77   tropical depres~
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical storm
## 10 Amy    1975     6    29     6  34   -74.8 tropical storm
## # i 19,056 more rows
## # i 6 more variables: tropicalstorm_force_diameter <int>,
```

```
separate(data = storms,
         col = "date",
         into = c("year", "month", "day"),
         sep = "-")
```

```
## Error in `separate()`:
## ! Can't extract columns that don't exist.
## x Column `date` doesn't exist.
```

unite()

Paste together multiple columns into one.

```
## Error in `separate()`:
## ! Can't extract columns that don't exist.
## x Column `date` doesn't exist.
```

```
storms2
```

```
## Error in eval(expr, envir, enclos): object 'storms2' not found
```

```
unite(data = storms2,
      col = "date",
      "year", "month", "day",
      sep = "-")
```

```
## Error in unite(data = storms2, col = "date", "
```

Tidy data

Getting tidy data

Data wrangling

- `select()`
- `filter()`
- `slice()`
- `mutate()`
- `summarise()`
- `arrange()`

Pipeline

Group wrangling

Joining data sets

Data wrangling

Having obtained a tidy dataset, there are several ways to access information.

1. **Extract** existing variables (columns).
   - `select()`
2. **Extract** existing observations (rows).
   - `filter()` or `slice()`
3. **Derive** new variables (from existing variables).
   - `mutate()`
4. **Change** the unit of analysis.
   - `summarise()`

select()

Select variables in a data frame.

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long status      category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>       <fct>    <dbl>    <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical de~ NA          25     1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical de~ NA          25     1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical de~ NA          25     1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical de~ NA          25     1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical de~ NA          25     1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical de~ NA          25     1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical de~ NA          25     1011
##  8 Amy    1975     6    28    18  34   -77   tropical de~ NA          30     1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical st~ NA          35     1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical st~ NA          40     1002
## # ... with 19,056 more rows
```

```
# After
select(.data = storms, storm, pressure)
```

```
## Error in `select()`:
## ! Can't subset columns that don't exist.
## x Column `storm` doesn't exist.
```

`select()`: Optionally rename

Optionally rename while selecting.

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long status        category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>         <dbl> <int>    <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical depres~       NA    25     1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical depres~       NA    25     1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical depres~       NA    25     1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical depres~       NA    25     1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical depres~       NA    25     1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical depres~       NA    25     1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical depres~       NA    25     1011
##  8 Amy    1975     6    28    18  34   -77   tropical depres~       NA    30     1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical storm        NA    35     1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical storm        NA    40     1002
## # ... with 19,056 more rows
```

```
# After
select(.data = storms, STORM = storm,
       PRESSURE = pressure)
```

```
## Error in `select()`:
## ! Can't subset columns that don't exist.
## ✖ Column `storm` doesn't exist.
```

`select()`: Negative subsetting

Deselecting variables.

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long status      category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>       <ord>    <int>    <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical de~ NA          25     1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical de~ NA          25     1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical de~ NA          25     1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical de~ NA          25     1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical de~ NA          25     1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical de~ NA          25     1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical de~ NA          25     1011
##  8 Amy    1975     6    28    18  34   -77   tropical de~ NA          30     1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical st~ NA          35     1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical st~ NA          40     1002
## # ... with 19,056 more rows
```

```
# After
select(.data = storms, -storm)
```

```
## Error in `select()`:
## ! Can't subset columns that don't exist.
## ✖ Column `storm` doesn't exist.
```

`select()`: Selecting a range of consecutive variables

Use `:` to select a range of consecutive variables

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long status   category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>    <fct>    <dbl>    <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical ~ NA          25     1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical ~ NA          25     1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical ~ NA          25     1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical ~ NA          25     1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical ~ NA          25     1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical ~ NA          25     1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical ~ NA          25     1011
##  8 Amy    1975     6    28    18  34   -77   tropical ~ NA          30     1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical ~ NA          35     1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical ~ NA          40     1002
## # ... with 19,056 more rows
```

```
# After
select(.data = storms, storm:pressure)
```

```
## Error in `select()`:
## ! Can't subset columns that don't exist.
## x Column `storm` doesn't exist.
```

Useful select functions

Call	Use
`-`	Select everything but
`:`	Select range
`contains()`	Select columns whose name contains a character string
`starts_with()`	Select column whose name starts with a character string
`ends_with()`	Select columns whose nmame ends with a string
`everything()`	Select all columns
`matches()`	Select columns whose name matches a regular expression
`num_range()`	Select columns matching a numerical range e.g. `x1`, `x2`, etc.
`one_of()`	Select columns whose names are in a group of names

There are others. See `?select` for further details.

filter()

This is used to subset a data frame, retaining all rows that satisfy your logical tests.

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 Amy    1975     6    27     0  27.5 -79
##  2 Amy    1975     6    27     6  28.5 -79
##  3 Amy    1975     6    27    12  29.5 -79
##  4 Amy    1975     6    27    18  30.5 -79
##  5 Amy    1975     6    28     0  31.5 -78.8
##  6 Amy    1975     6    28     6  32.4 -78.7
##  7 Amy    1975     6    28    12  33.3 -78
##  8 Amy    1975     6    28    18  34   -77
##  9 Amy    1975     6    29     0  34.4 -75.8
## 10 Amy    1975     6    29     6  34   -74.8
## # ... with 19,056 more rows
```

```
# After
filter(.data = storms, wind >= 50)
```

```
## # A tibble: 8,118 x 13
##    name   year month   day  hour   lat  long
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 Amy    1975     6    29    18  36.1 -67.2
##  2 Amy    1975     6    30     0  37.4 -66.7
##  3 Amy    1975     6    30     6  38.6 -65.8
##  4 Amy    1975     6    30    12  39.8 -64.5
##  5 Amy    1975     6    30    18  41.2 -62.8
##  6 Amy    1975     7     1     0  42.5 -60.8
##  7 Amy    1975     7     1     6  43.9 -58.5
##  8 Amy    1975     7     1    12  45.2 -55.8
##  9 Amy    1975     7     1    18  46.5 -52.8
## 10 Amy    1975     7     2     0  47.8 -49.5
## # ... with 8,108 more rows
```

filter() (cont.)

You can combine tests separated by commas.

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  lon
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 Amy    1975     6    27     0  27.5 -79
##  2 Amy    1975     6    27     6  28.5 -79
##  3 Amy    1975     6    27    12  29.5 -79
##  4 Amy    1975     6    27    18  30.5 -79
##  5 Amy    1975     6    28     0  31.5 -78.8
##  6 Amy    1975     6    28     6  32.4 -78.7
##  7 Amy    1975     6    28    12  33.3 -78
##  8 Amy    1975     6    28    18  34   -77
##  9 Amy    1975     6    29     0  34.4 -75.8
## 10 Amy    1975     6    29     6  34   -74.8
## # ... with 19,056 more rows
```

```
# After
filter(.data = storms,
       wind >= 50,
       storm %in% c("Alberto", "Alex",
                    "Allison"))
```

```
##   status                 category  wind pressure tropicalstorm_force_diameter
##   <fct>                     <dbl> <int>    <int>                         <int>
## 1 tropical storm               NA    35     1004                            NA
## 2 tropical storm               NA    40     1002                            NA
## 3 tropical storm               NA    45     1000                            NA
## 4 tropical storm               NA    50      997                            NA
## 5 tropical storm               NA    60      987                            NA
## 6 tropical storm               NA    70      979                            NA
## 7 tropical storm               NA    60      981                            NA
## 8 tropical storm               NA    50      985                            NA
## 9 tropical storm               NA    45      994                            NA
```

```
## Error in filter(): ℹ In argument: `storm %in% c("Alberto", "Ale
## Caused by error in `storm %in% c("Alberto", ...`:
## ! object 'storm' not found
```

Logical tests in R

`?Comparison`

Test	Usage
`<`	Less than
`>`	Greater than
`==`	Equal to
`<=`	Less than or equal to
`>=`	Greater than or equal to
`!=`	Not equal to
`%in%`	Group membership
`is.na()`	Is NA?

`?base::Logic`

Operator	Usage
`&`	Boolean and
`|`	Boolean or
`xor`	Exactly or
`!`	Not
`any()`	Any true
`all()`	All true

slice()

This lets you index rows by their (integer) locations. Thus, it allows you to select, remove, and duplicate rows.

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long status   category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>    <ord>    <int>    <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical~ NA          25     1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical~ NA          25     1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical~ NA          25     1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical~ NA          25     1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical~ NA          25     1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical~ NA          25     1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical~ NA          30     1006
##  8 Amy    1975     6    28    18  34   -77   tropical~ NA          35     1004
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical~ NA          40     1002
## # ... with 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>
```

```
# After
slice(.data = storms, 1:3)
```

```
## # A tibble: 3 x 13
##   name   year month   day  hour   lat  long status   category  wind pressure
##   <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>    <ord>    <int>    <int>
## 1 Amy    1975     6    27     0  27.5 -79   tropical~ NA          25     1013
## 2 Amy    1975     6    27     6  28.5 -79   tropical~ NA          25     1013
## 3 Amy    1975     6    27    12  29.5 -79   tropical~ NA          25     1013
## # ... with 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>
```

```
slice(.data = storms, rep(1, 3))
```

```
## # A tibble: 3 x 13
```

slice() (cont.)

Using which.min() or which.max() (or other functions which return row indices) is quite helpful with slice().

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 Amy    1975     6    27     0  27.5 -79
##  2 Amy    1975     6    27     6  28.5 -79
##  3 Amy    1975     6    27    12  29.5 -79
##  4 Amy    1975     6    27    18  30.5 -79
##  5 Amy    1975     6    28     0  31.5 -78.8
##  6 Amy    1975     6    28     6  32.4 -78.7
##  7 Amy    1975     6    28    12  33.3 -78
##  8 Amy    1975     6    28    18  34   -77
##  9 Amy    1975     6    29     0  34.4 -75.8
## 10 Amy    1975     6    29     6  34   -74.8
```

```
# After
slice(.data = storms, which.min(pressure))
```

```
## # A tibble: 1 x 13
##   name   year month category  day  hour pressure  lat  long
##   <chr> <dbl> <dbl>    <int> <int> <dbl>    <int> <dbl> <dbl>
## 1 Wilma  2005    10       19    25    12      882  17.3 -82.8
## # ... with 2 more variables: tropicalstorm_force_di
## #   hurricane_force_diameter <int>
```

```
slice(.data = storms, which.max(wind))
```

```
## # A tibble: 1 x 13
##   name   year month   day  hour   lat  long
##   <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 Allen  1980     8     7     0  16.8 -93.2
```

mutate()

We may want to **create** new variables from existing variables. Suppose we want to derive a new variable called `ratio` which is defined as

$$\texttt{ratio} = \frac{\texttt{pressure}}{\texttt{wind}}$$

storm	wind	pressure	date
Alberto	110	1007	2000-08-03
Alex	45	1009	1998-07-27
Allison	65	1005	1995-06-03
Ana	40	1013	1997-06-30
Arlene	50	1010	1999-06-11
Arthur	45	1010	1999-06-17

```
storms$pressure / storms$wind → ratio
      1007      /     110    →  9.16
      1009      /      45    → 22.42
      1005      /      65    → 15.46
      1013      /      40    → 25.33
      1010      /      50    → 20.20
      1010      /      45    → 22.44
```

mutate() (cont.)

Using dplyr's mutate() function, we are able to do this easily without having to use $ all the time.

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long status      category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>       <chr>    <dbl>    <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical de… NA          25     1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical de… NA          25     1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical de… NA          25     1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical de… NA          25     1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical de… NA          25     1010
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical de… NA          25     1010
##  7 Amy    1975     6    28    12  33.3 -78   tropical de… NA          25     1011
##  8 Amy    1975     6    28    18  34   -77   tropical de… NA          30     1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical st… NA          35     1004
## 10 Amy    1975     6    29     6  34.8 -74.8 tropical st… NA          40     1002
## # … with 19,056 more rows, and 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>
```

```
# After
mutate(.data = storms,
       ratio = pressure / wind)
```

```
## # A tibble: 19,066 x 14
##    name   year month   day  hour   lat  long status      category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>       <chr>    <dbl>    <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical de… NA          25     1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical de… NA          25     1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical de… NA          25     1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical de… NA          25     1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical de… NA          25     1010
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical de… NA          25     1010
##  7 Amy    1975     6    28    12  33.3 -78   tropical de… NA          25     1011
##  8 Amy    1975     6    28    18  34   -77   tropical de… NA          30     1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical st… NA          35     1004
## 10 Amy    1975     6    29     6  34.8 -74.8 tropical st… NA          40     1002
## # … with 19,056 more rows, and 3 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>, ratio <dbl>
```

mutate() (cont.)

Keep on adding new columns in the same mutate() call. Just separate them by commas.

```
mutate(.data = storms,
       ratio = pressure / wind,
       inverse = ratio ^ (-1))
```

```
## # A tibble: 19,066 x 15
##    name   year month   day  hour   lat  long status           category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>                <dbl> <int>    <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical depres~       NA    25     1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical depres~       NA    25     1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical depres~       NA    25     1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical depres~       NA    25     1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical depres~       NA    25     1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical depres~       NA    25     1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical depres~       NA    25     1011
##  8 Amy    1975     6    28    18  34   -77   tropical depres~       NA    30     1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical storm         NA    35     1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical storm         NA    40     1002
## # ... with 19,056 more rows
```

Useful `mutate()` functions

Operator	Usage
`pmin()`, `pmax()`	Element-wise min and max
`cummin()`, `cummax()`	Cumulative min and max
`cumsum()`, `cumprod()`	Cumulative sum and product
`between()`	Are values between `a` and `b`?
`cumall()`, `cumany()`	Cumulative `all()` and `any()`
`cummean()`	Cumulative mean
`lead()`, `lag()`	Comparing values behind or ahead of current values
`ntile()`	Bin vector into `n` buckets
`row_number()`	Returns row number

Remark

All of these (window) functions take vector values and return vector values of the same length. If using non-window functions, the recycling rule applies.

summarise()

On the other hand, we may want to **condense** the available information. For this, the `summarise()` function returns a new data frame.

```
# Before
pollution
```

```
## Error in eval(expr, envir, enclos): object
```

```
# After
summarise(.data = pollution,
          median = median(amount),
          variance = var(amount))
```

```
## Error in summarise(.data = pollution, median
```

Useful `summarise()` functions

Operator	Usage
`min()`, `max()`	Minimum and maximum values
`mean()`	Mean value
`median()`	Median value
`sum()`	Sum of values
`var()`, `sd()`	Variance and standard deviation of a vector
`first()`, `last()`	First or last value in a vector
`nth()`	Nth value in a vector
`n()`	The number of values in a vector
`n_distinct()`	The number of distinct values in a vector

Remark

All of these functions take vector values and return a single value.

Summary (`summarise`) vs window (`mutate`) functions

- Functions used with `summarise()` should reduce the length of the input vector to a single value.
- Functions used with `mutate()` should keep the vector length.

arrange()

Sorting rows (in ascending order) in a particular column is done using `arrange()`.

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 Amy    1975     6    27     0  27.5 -79
##  2 Amy    1975     6    27     6  28.5 -79
##  3 Amy    1975     6    27    12  29.5 -79
##  4 Amy    1975     6    27    18  30.5 -79
##  5 Amy    1975     6    28     0  31.5 -78.8
##  6 Amy    1975     6    28     6  32.4 -78.7
##  7 Amy    1975     6    28    12  33.3 -78
##  8 Amy    1975     6    28    18  34   -77
##  9 Amy    1975     6    29     0  34.4 -75.8
## 10 Amy    1975     6    29     6  34   -74.8
## # ... with 19,056 more rows
```

```
# After
arrange(.data = storms, wind)
```

```
## # A tibble: 19,066 x 13
##    name       year month   day  hour   lat     l
##    <fct>     <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 Bonnie     1986     8    28     6  36.5 -9
##  2 Bonnie     1986     8    28    12  37.2 -9
##  3 Charley    1986     8    13    12  30.1 -8
##  4 Charley    1986     8    13    18  30.8 -8
##  5 Charley    1986     8    14     0  31.4 -8
##  6 Charley    1986     8    14     6  32   -8
##  7 Charley    1986     8    14    12  32.5 -8
##  8 Charley    1986     8    14    18  32.4 -8
##  9 ABOUT 1987 1987     8    16    18  30.9 -8
## 10 ABOUT 1987 1987     8    17     0  31.4 -8
## # ... with 19,056 more rows
```

arrange() (cont.)

Sorting rows (in descending order) in a particular column is done by applying `desc()` on the variable.

```r
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 Amy    1975     6    27     0  27.5 -79
##  2 Amy    1975     6    27     6  28.5 -79
##  3 Amy    1975     6    27    12  29.5 -79
##  4 Amy    1975     6    27    18  30.5 -79
##  5 Amy    1975     6    28     0  31.5 -78.8
##  6 Amy    1975     6    28     6  32.4 -78.7
##  7 Amy    1975     6    28    12  33.3 -78
##  8 Amy    1975     6    28    18  34   -77
##  9 Amy    1975     6    29     0  34.4 -75.8
## 10 A      1975     6    29     6  34   -74.8
```

```r
# After
arrange(.data = storms, desc(wind))
```

```
## # A tibble: 19,066 x 13
##    name     year month   day  hour   lat  lo
##    <chr>   <dbl> <dbl> <int> <dbl> <dbl> <db
##  1 Allen    1980     8     7    18  21.8 -86
##  2 Gilbert  1988     9    14     0  19.7 -83
##  3 Wilma    2005    10    19    12  17.3 -82
##  4 Dorian   2019     9     1    16  26.5 -77
##  5 Dorian   2019     9     1    18  26.5 -77
##  6 Allen    1980     8     5    12  15.9 -70
##  7 Allen    1980     8     7    12  21   -84
##  8 Allen    1980     8     8     0  22.2 -87
##  9 Allen    1980     8     9     6  25   -94
```

```
## status          category  wind pressure
## <fct>              <dbl> <int>    <int>
## tropical depres-      NA    25     1013
## tropical depres-      NA    25     1013
## tropical depres-      NA    25     1013
## tropical depres-      NA    25     1013
## tropical depres-      NA    25     1012
## tropical depres-      NA    25     1012
## tropical depres-      NA    25     1011
## tropical depres-      NA    30     1006
## tropical storm        NA    35     1004
```

arrange() (cont.)

You may arrange by multiple columns in order.

```
# Before
storms
```

```
## # A tibble: 19,066 x 13
##    name   year month   day  hour   lat  long status   category  wind pressure
##    <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>    <fct>    <dbl>    <int>
##  1 Amy    1975     6    27     0  27.5 -79   tropical ~ NA          25     1013
##  2 Amy    1975     6    27     6  28.5 -79   tropical ~ NA          25     1013
##  3 Amy    1975     6    27    12  29.5 -79   tropical ~ NA          25     1013
##  4 Amy    1975     6    27    18  30.5 -79   tropical ~ NA          25     1013
##  5 Amy    1975     6    28     0  31.5 -78.8 tropical ~ NA          25     1012
##  6 Amy    1975     6    28     6  32.4 -78.7 tropical ~ NA          25     1012
##  7 Amy    1975     6    28    12  33.3 -78   tropical ~ NA          25     1011
##  8 Amy    1975     6    28    18  34   -77   tropical ~ NA          30     1006
##  9 Amy    1975     6    29     0  34.4 -75.8 tropical ~ NA          35     1004
## 10 Amy    1975     6    29     6  34   -74.8 tropical ~ NA          40     1002
## # ... with 19,056 more rows
```

```
# After
arrange(.data = storms, wind, date)
```

```
## Error in `arrange()`:
## ! In argument: `..2 = date`.
## Caused by error:
## ! `..2` must be a vector, not a function.
```

Tidy data

Getting tidy data

Data wrangling

Pipeline

Group wrangling

Joining data sets

Pipeline

Consider the following sequence of actions that describe the process of getting to campus in the morning:

> *I need to find my key, then unlock my car, then start my car, then drive to school, then park.*

Expressed as a set of nested functions in R pseudocode this would look like:

```
park(drive(start_car(find("keys")), to = "campus"))
```

Writing it out using pipes give it a more natural (and easier to read) structure:

```
find("keys") %>%
    start_car() %>%
    drive(to = "campus") %>%
    park()
```

The pipe operator %>%

The pipe operator puts the output of the LHS into the first argument of the function of the RHS.

Therefore, the following code both do the same thing.

```
summarise(storms, mean = mean(wind))
```

```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  50.0
```

```
storms %>% summarise(., mean = mean(wind))
```

```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  50.0
```

In fact, we may drop the '`.`' when the situation is obvious, i.e. `storms %>% summarise(mean = mean(wind))` would give the same thing.

Combining `dplyr` functions

For data wrangling, it seems more natural to progressively write code in a pipeline path. As an example, consider the `nycflights13::flights` data set. How many flights to LAX did each of the legacy carriers (AA, UA, DL or US) have in May from JFK, and what was their average duration?

```
library(nycflights13)
filtered_flights <-
  filter(.data = flights,
         origin == "JFK", dest == "LAX", month == 5,
         carrier %in% c("AA", "UA", "DL", "US"))
res <- summarise(.data = filtered_flights,
                 n = n(), avg_dur = mean(air_time, na.rm = TRUE))
res
```

```
## # A tibble: 1 x 2
##       n avg_dur
##   <int>   <dbl>
## 1   685    320.
```

Combining `dplyr` functions (cont.)

In contrast, we can pipe the entire thing:

```
flights %>%
  # Filter first
  filter(origin == "JFK", dest == "LAX", month == 5,
         carrier %in% c("AA", "UA", "DL", "US")) %>%
  # Then summarise
  summarise(
    n = n(),
    avg_dur = mean(air_time, na.rm = TRUE)
  )
```

```
## # A tibble: 1 x 2
##       n avg_dur
##   <int>   <dbl>
## 1   685    320.
```

Another example

```
flights %>%
  # Select all variables containing "delay" in their name
  select(contains("delay")) %>%
  # Create a new gain variable
  mutate(gain = arr_delay - dep_delay) %>%
  # Drop all rows with NA in them
  drop_na() %>%
  # Summarise
  summarise(
    min = min(gain),
    max = max(gain),
    mean = mean(gain)
  )
```

```
## # A tibble: 1 x 3
##     min   max  mean
##   <dbl> <dbl> <dbl>
## 1  -109   196 -5.66
```

Tidy data

Getting tidy data

Data wrangling

Pipeline

Group wrangling

Joining data sets

Group wrangling

When we used the `summarise()` function, we were in fact using the **entire data set** to arrive at the summaries.

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

`summarise()`

sum	n	mean
252	6	42

e.g. `pollution %>% summarise(sum = sum(amount), n = n(), mean = mean(amount))`.

Group wrangling (cont.)

Grouping the observations by some categorical variable allows us to uncover hidden information lying within the groups. We use `group_by()` to do this.

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

```
group_by(city) %>%
  summarise()
```

sum	n	mean
37	2	18.5
38	2	19
177	2	88.5

Group wrangling (cont.)

The code to obtain the previous grouped summary table is

```
pollution %>%
  group_by(city) %>%
  summarise(
    sum = sum(amount),
    n = n(),
    mean = mean(amount)
  )
```

```
## Error in group_by(., city): object 'pollution' not found
```

ungroup()

Note that when you group data, this will persist throughout the pipeline. This is indicated by the `Groups: city [3]` print out below. If, further down the pipeline, you wish to mutate or summarise based on the entire data set, you must first `ungroup()`.

```
pollution %>%
  group_by(city)
```

```
## Error in group_by(., city): object 'pollution' not found
```

```
pollution %>%
  group_by(city) %>%
  ungroup()
```

```
## Error in group_by(., city): object 'pollutio
```

Multiple groups

Multiple groupings are allowed. Let's find the top 3 routes with the smallest departure delay in the `flights` dataset.

```
flights %>%
  group_by(origin, dest) %>%
  summarise(min_delay = min(dep_delay)) %>%
  arrange(min_delay) %>%
  print(n = 3)
```

```
## # A tibble: 224 x 3
## # Groups:   origin [3]
##   origin dest  min_delay
##   <chr>  <chr>     <dbl>
## 1 LGA    EYW         -18
## 2 JFK    HNL         -16
## 3 JFK    ACK         -13
## # i 221 more rows
```

Tidy data

Getting tidy data

Data wrangling

Pipeline

Group wrangling

Joining data sets

- `bind_cols()`
- `bind_rows()`
- `left_join()`

Joining data sets

Here are the functions useful for combining data sets

1. `bind_cols()`
2. `bind_rows()`
3. `*_join()` commands
4. Other set operation functions such as `union()`, `intersect()`, and `setdiff()`

For more details on these functions, please have a look at `https://github.com/gadenbuie/tidyexplain--This` page contains a helpful information about what the join functions do.

bind_cols()

If you have two or more data frames (or columns) that are meant to go together column-wise, then use `bind_cols()`.

```
x <- storms[, 1:3]  # First 3 columns of storms
y <- storms$date  # The last column of storms
bind_cols(x, y)
```

```
## # A tibble: 19,066 x 3
##    name   year month
##    <chr> <dbl> <dbl>
##  1 Amy    1975     6
##  2 Amy    1975     6
##  3 Amy    1975     6
##  4 Amy    1975     6
##  5 Amy    1975     6
##  6 Amy    1975     6
##  7 Amy    1975     6
##  8 Amy    1975     6
##  9 Amy    1975     6
## 10 Amy    1975     6
```

bind_rows()

Similarly if you wanted to stack two or more data frameson top of each other, then use `bind_rows()`.

```
x <- storms[1:3, ]  # First 3 columns of storms
y <- storms[4:6, ]  # Last 3 columns of storms
bind_rows(x, y)
```

```
## # A tibble: 6 x 13
##   name   year month   day  hour   lat  long status            category  wind pressure
##   <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>                <dbl> <int>    <int>
## 1 Amy    1975     6    27     0  27.5 -79   tropical depress~       NA    25     1013
## 2 Amy    1975     6    27     6  28.5 -79   tropical depress~       NA    25     1013
## 3 Amy    1975     6    27    12  29.5 -79   tropical depress~       NA    25     1013
## 4 Amy    1975     6    27    18  30.5 -79   tropical depress~       NA    25     1013
## 5 Amy    1975     6    28     0  31.5 -78.8 tropical depress~       NA    25     1012
## 6 Amy    1975     6    28     6  32.4 -78.7 tropical depress~       NA    25     1012
## # i 2 more variables: tropicalstorm_force_diameter <int>,
## #   hurricane_force_diameter <int>
```

Remark

`bind_rows()`

`bind_cols()`

Warning

`bind_cols()` and `bind_rows()` have no way of checking whether or not the joining is consistent (e.g. is the row-ordering the same between both data frames in `bind_cols()`?)

left_join()

Out of all the `*_join()` functions, this is probably the most frequently used (at least for me anyway). When calling `left_join(x, y)`, this

- adds columns from `y` to `x`;
- matching rows based on the `by` keys;
- and **includes all rows in `x`** (but possibly not `y`).

`left_join()` (cont.)

An example of `left_join()` on the band members and instruments data sets.

`band_members`

```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

`band_instruments`

```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

```
band_members %>%
  left_join(., band_instruments, by = "name")
```

```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```