Tidy Data and Efficient Data Wrangling

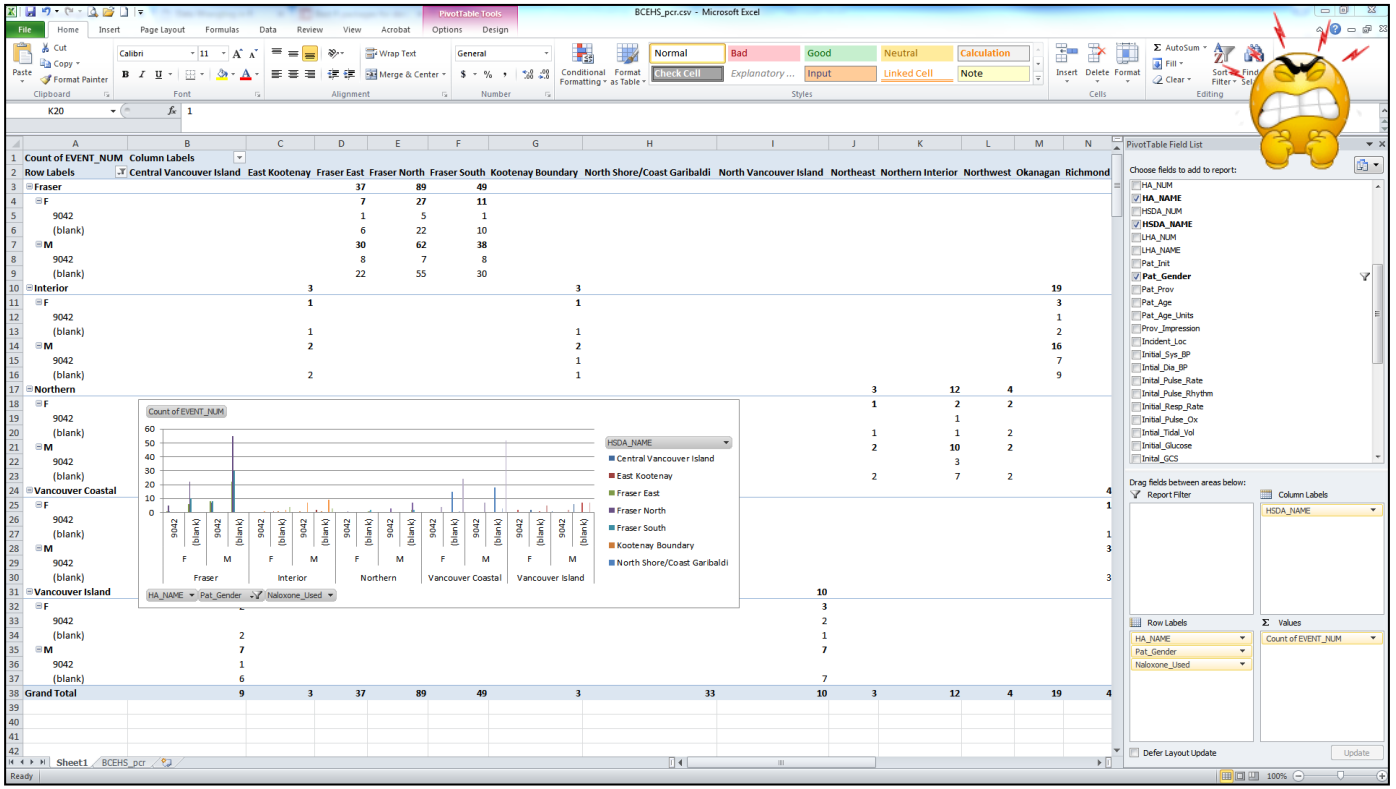
Michael Otterstatter

BCCDC Biostats Session

Jan 18, 2019

Messy data

- Health datasets are typically too large and complex to be efficiently handled in spreadsheets



‘Wrangling’ messy data

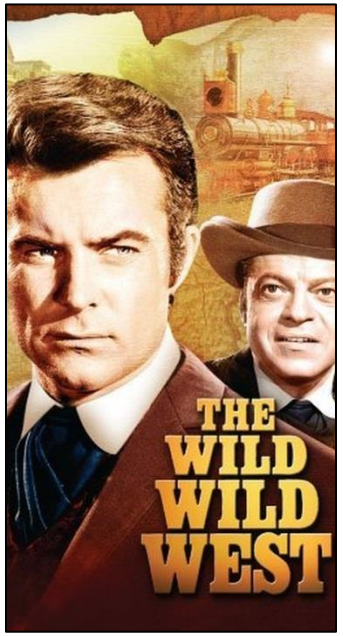
- DATA WRANGLING: process of converting raw data into a format allowing convenient ‘consumption’, including visualization, aggregation, statistical modeling, etc.

- The process typically follows general steps beginning with importing raw data and then "munging" these data using algorithms (e.g., sorting) or parsing the data into ***predefined data structures***.

https://en.wikipedia.org/wiki/Data_wrangling

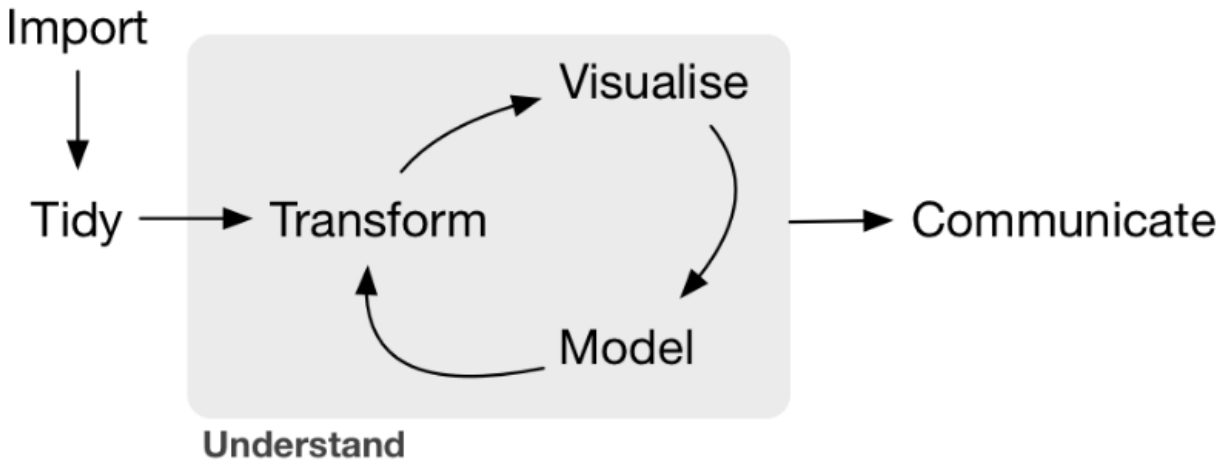
‘Wrangling’ messy data

- Efficient ‘wrangling’ or ‘munging’ of large/messy data requires
- powerful software, e.g.,
  - data management / programming (e.g., SQL)
  - statistical packages (e.g., SAS, R, etc.)
- a framework for data manipulation
  - fly by the seat of your pants?
  - **tidy data**



Tidy data

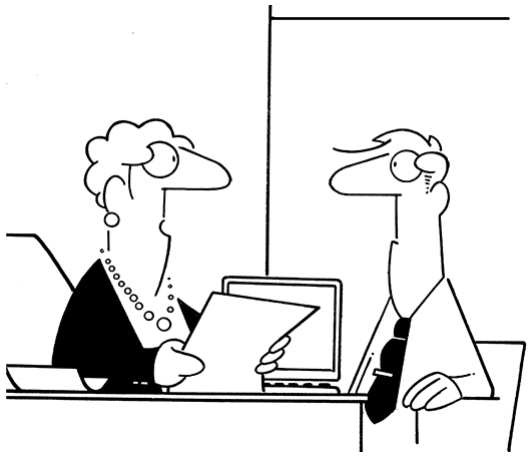
- A foundational activity in the analysis of health data is creating a tidy dataset
- Tidy data are clean, efficient and ready for visualization, linkage and analysis



http://jules32.github.io/2016-07-12-Oxford/dplyr_tidyr/#

Not tidy ☹

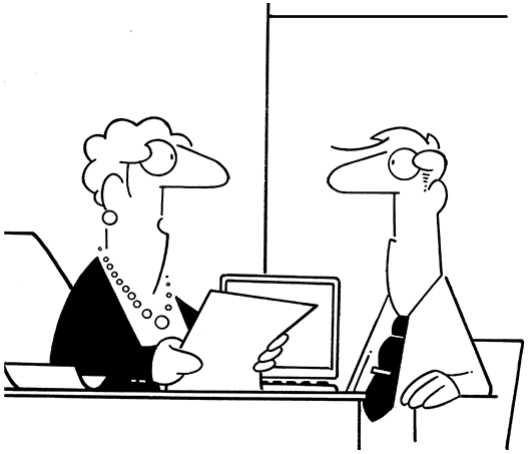
Health Authority	<1 year old	1 to 4 years	5 to 9 years
Fraser	--	4	3
Vancouver Coastal	--	7	5
Vancouver Island	1	9	7
Northern	3	1	9
Interior	2	--	1



"I need these summarised by age group."

Not tidy ☹

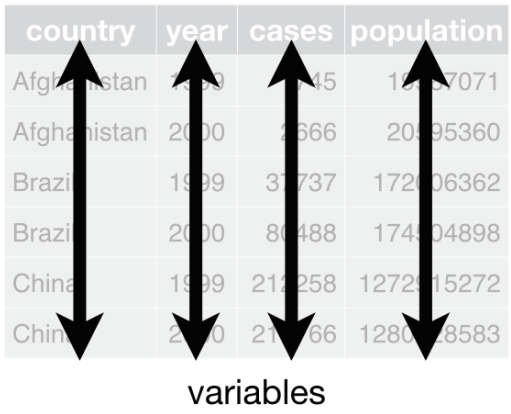
Age	Fraser	Vancouver Coastal	Vancouver Island	Northern	Interior
<1 year old	--	--	1	3	2
1 to 4 years	4	7	9	1	--
5 to 9 years	3	5	7	9	1

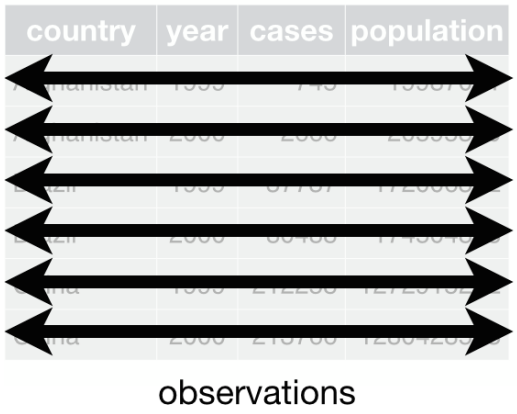


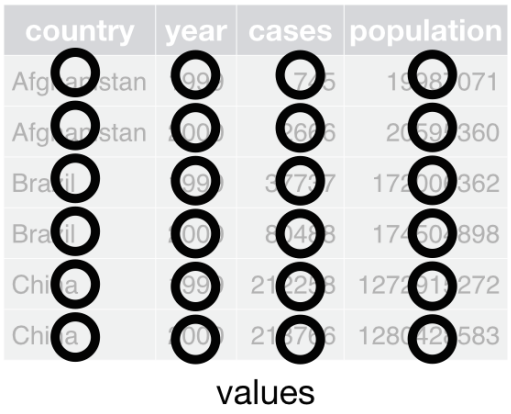
"I changed my mind. I need these summarised by Health Authority."

Tidy data

- Basic structure of tidy data
  - each variable is saved in its own column
  - each observation (individual, or other unit) is saved in its own row
  - data cells contain values of variables







Tidy ☺

Health Authority	Age	Cases
Fraser	<1 year old	--
Fraser	1 to 4 years	4
Fraser	5 to 9 years	3
Vancouver Coastal	<1 year old	--
Vancouver Coastal	1 to 4 years	7
Vancouver Coastal	5 to 9 years	5
Vancouver Island	<1 year old	1
Vancouver Island	1 to 4 years	9
Vancouver Island	5 to 9 years	7
Northern	<1 year old	3
Northern	1 to 4 years	1
Northern	5 to 9 years	9
Interior	<1 year old	2
Interior	1 to 4 years	--
Interior	5 to 9 years	1

Tidy ☺

Health Authority	Age	Cases
Fraser	<1 year old	--
Fraser	1 to 4 years	4
Fraser	5 to 9 years	3
Vancouver Coastal	<1 year old	--
Vancouver Coastal	1 to 4 years	7
Vancouver Coastal	5 to 9 years	5
Vancouver Island	<1 year old	1
Vancouver Island	1 to 4 years	9
Vancouver Island	5 to 9 years	7
Northern	<1 year old	3
Northern	1 to 4 years	1
Northern	5 to 9 years	9
Interior	<1 year old	2
Interior	1 to 4 years	--
Interior	5 to 9 years	1

Variables as columns

Tidy ☺

Health Authority	Age	Cases
Fraser	<1 year old	--
Fraser	1 to 4 years	4
Fraser	5 to 9 years	3
Vancouver Coastal	<1 year old	--
Vancouver Coastal	1 to 4 years	7
Vancouver Coastal	5 to 9 years	5
Vancouver Island	<1 year old	1
Vancouver Island	1 to 4 years	9
Vancouver Island	5 to 9 years	7
Northern	<1 year old	3
Northern	1 to 4 years	1
Northern	5 to 9 years	9
Interior	<1 year old	2
Interior	1 to 4 years	--
Interior	5 to 9 years	1

Observation units as rows

Tidy ☺

Health Authority	Age	Cases
Fraser	<1 year old	--
Fraser	1 to 4 years	4
Fraser	5 to 9 years	3
Vancouver Coastal	<1 year old	--
Vancouver Coastal	1 to 4 years	7
Vancouver Coastal	5 to 9 years	5
Vancouver Island	<1 year old	1
Vancouver Island	1 to 4 years	9
Vancouver Island	5 to 9 years	7
Northern	<1 year old	3
Northern	1 to 4 years	1
Northern	5 to 9 years	9
Interior	<1 year old	2
Interior	1 to 4 years	--
Interior	5 to 9 years	1

Data cells are **values** of variables

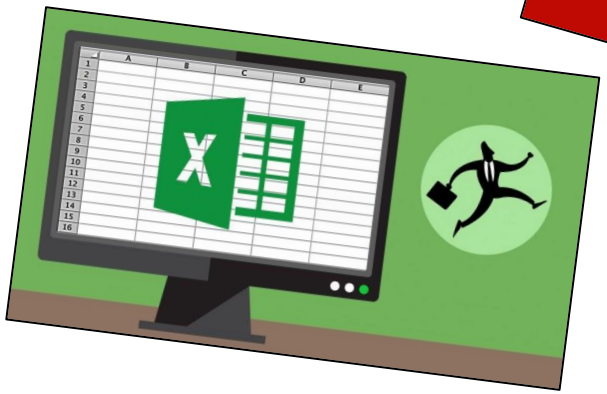
Why Tidy?

- **General benefit:** a consistent way of structuring/storing data, making it easier to learn tidy data tools. These tools work consistently with tidy data because of their standardized structure

- **Specific benefit:** variables as columns allows efficient use of R, which typically functions by using vectors of values

Tidy data

- Principles of tidy data apply regardless of whether you use SAS, R, Stata, SPSS, or Excel (god help you)



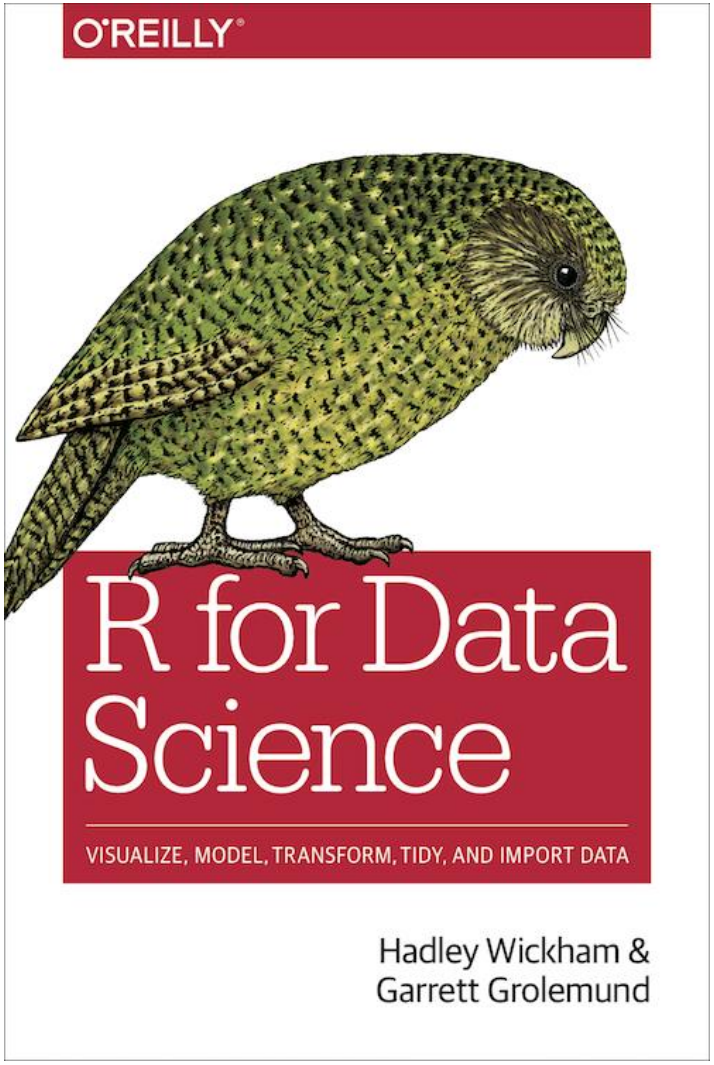
The tidyverse

- In R, tidy data principles are the foundation of a series of packages by Hadley Wickham, built specifically for efficient data wrangling and analysis

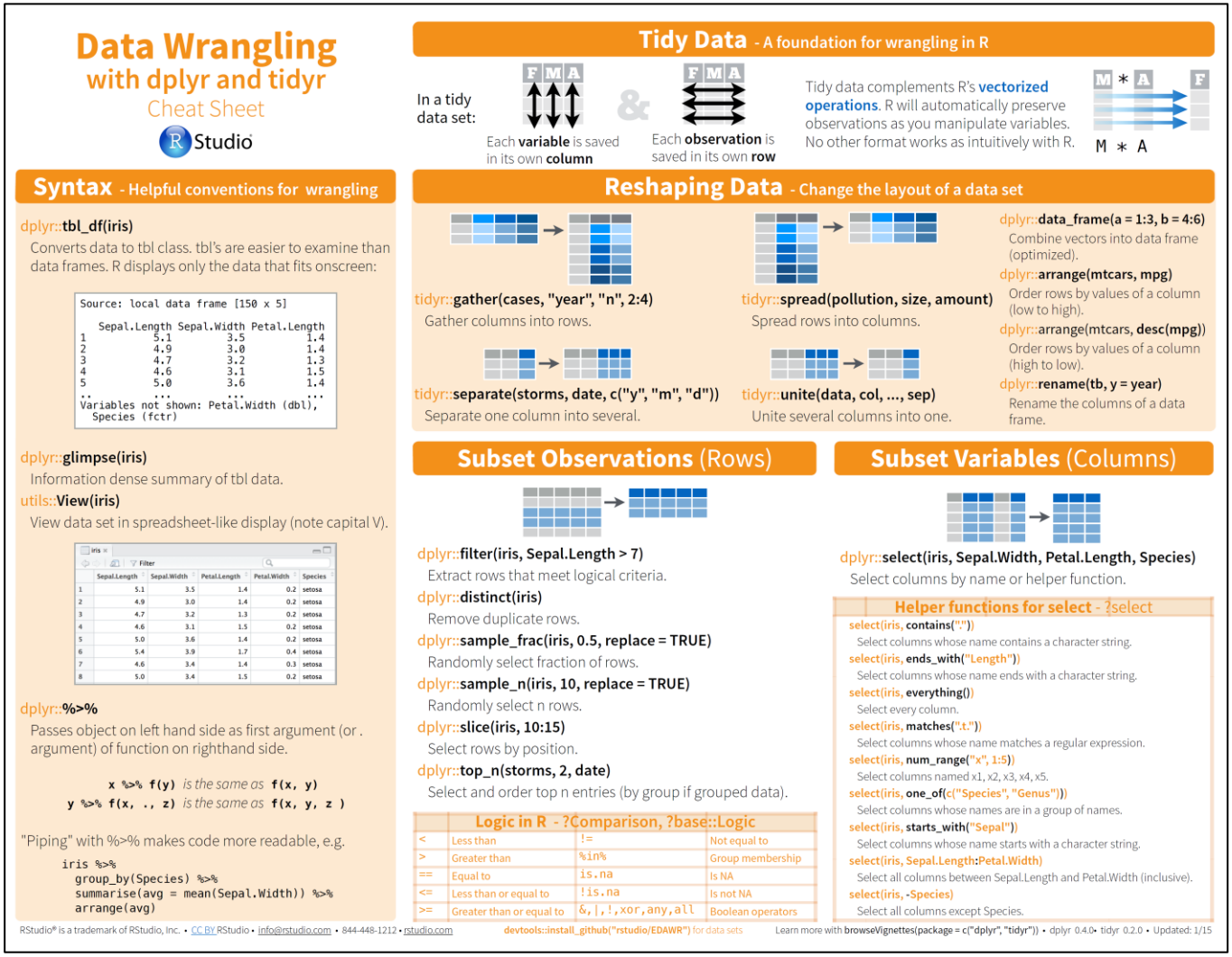
http://hadley.nz/ and http://tidyverse.org/

Introduction to tidy data work

- https://r4ds.had.co.nz/



Tidy data tools in R



Data Wrangling with dplyr and tidyr Cheat Sheet

RStudio

Syntax - Helpful conventions for wrangling

dplyr::tbl_df(iris)

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]

   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
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::glimpse(iris)

Information dense summary of tbl data.

utils::View(iris)

View data set in spreadsheet-like display (note capital V).

dplyr::%>%

Passes object on left hand side as first argument (or . argument) of function on righthand side.

```
x %>% f(y)  is the same as  f(x, y)
y %>% f(x, .., z)  is the same as  f(x, y, z)
```

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Tidy Data - A foundation for wrangling in R

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

Reshaping Data - Change the layout of a data set

tidyr::gather(cases, "year", "n", 2:4)

Gather columns into rows.

tidyr::spread(pollution, size, amount)

Spread rows into columns.

tidyr::separate(storms, date, c("y", "m", "d"))

Separate one column into several.

tidyr::unite(data, col, ..., sep)

Unite several columns into one.

dplyr::data_frame(a = 1:3, b = 4:6)

Combine vectors into data frame (optimized).

dplyr::arrange(mtcars, mpg)

Order rows by values of a column (low to high).

dplyr::arrange(mtcars, desc(mpg))

Order rows by values of a column (high to low).

dplyr::rename(tb, y = year)

Rename the columns of a data frame.

Subset Observations (Rows)

dplyr::filter(iris, Sepal.Length > 7)

Extract rows that meet logical criteria.

dplyr::distinct(iris)

Remove duplicate rows.

dplyr::sample_frac(iris, 0.5, replace = TRUE)

Randomly select fraction of rows.

dplyr::sample_n(iris, 10, replace = TRUE)

Randomly select n rows.

dplyr::slice(iris, 10:15)

Select rows by position.

dplyr::top_n(storms, 2, date)

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, |, !, xor, any, all	Boolean operators

Subset Variables (Columns)

dplyr::select(iris, Sepal.Width, Petal.Length, Species)

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, contains("."))
Select columns whose name contains a character string.

select(iris, ends_with("Length"))
Select columns whose name ends with a character string.

select(iris, everything())
Select every column.

select(iris, matches(".t."))
Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))
Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))
Select columns whose names are in a group of names.

select(iris, starts_with("Sepal"))
Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).

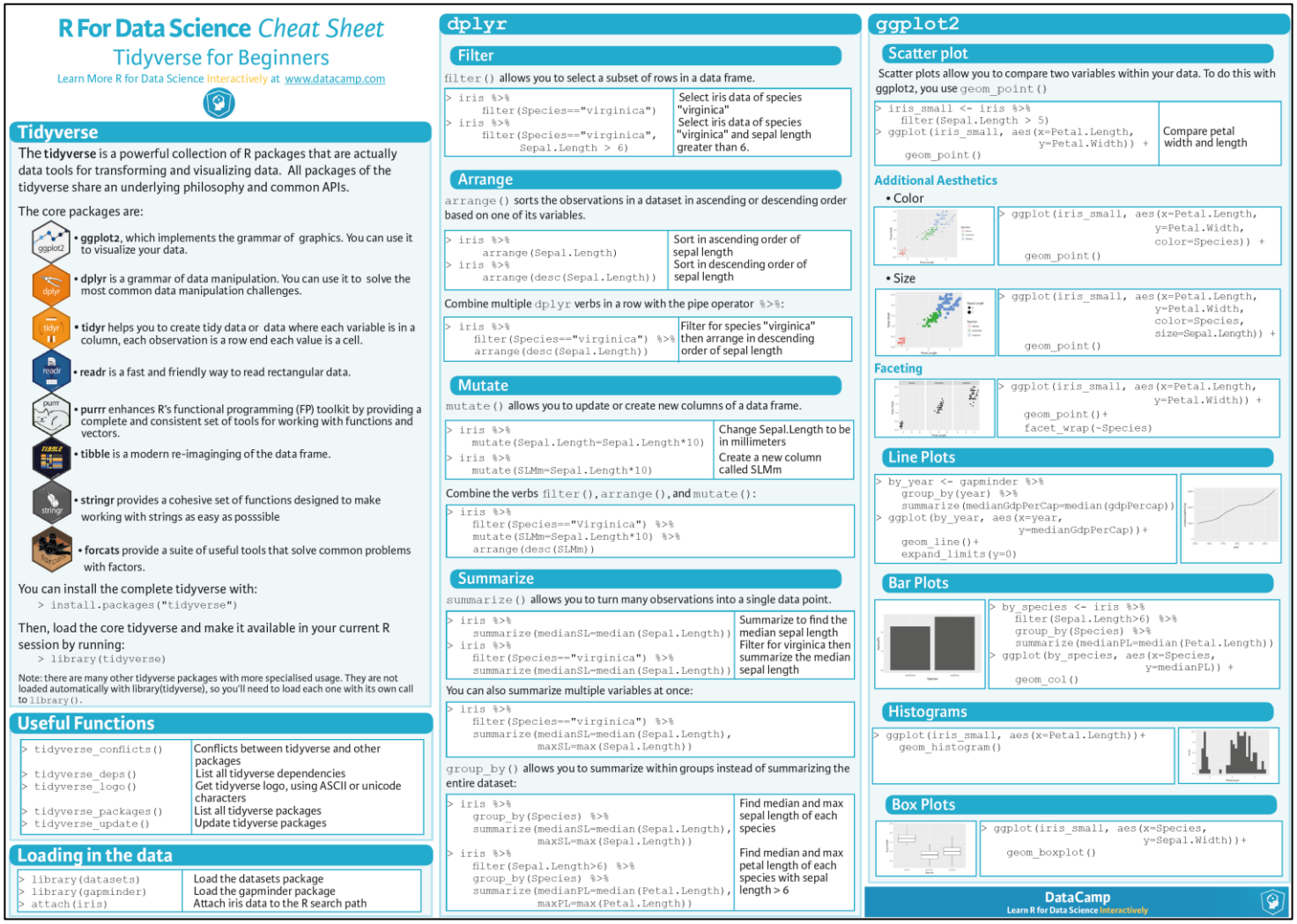
select(iris, -Species)
Select all columns except Species.

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0 • tidyr 0.2.0 • Updated: 1/15

http://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Tidy data tools in R



R For Data Science *Cheat Sheet*

Tidyverse for Beginners

Learn More R for Data Science Interactively at www.datacamp.com

Tidyverse

The **tidyverse** is a powerful collection of R packages that are actually data tools for transforming and visualizing data. All packages of the tidyverse share an underlying philosophy and common APIs.

The core packages are:

- **ggplot2**, which implements the grammar of graphics. You can use it to visualize your data.
- **dplyr** is a grammar of data manipulation. You can use it to solve the most common data manipulation challenges.
- **tidyr** helps you to create tidy data or data where each variable is in a column, each observation is a row end each value is a cell.
- **readr** is a fast and friendly way to read rectangular data.
- **purrr** enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors.
- **tibble** is a modern re-imaginging of the data frame.
- **stringr** provides a cohesive set of functions designed to make working with strings as easy as posssible
- **forcats** provide a suite of useful tools that solve common problems with factors.

You can install the complete tidyverse with:

```
> install.packages("tidyverse")
```

Then, load the core tidyverse and make it available in your current R session by running:

```
> library(tidyverse)
```

Note: there are many other tidyverse packages with more specialised usage. They are not loaded automatically with library(tidyverse), so you'll need to load each one with its own call to `library()`.

Useful Functions

`> tidyverse_conflicts()`	Conflicts between tidyverse and other packages
`> tidyverse_deps()`	List all tidyverse dependencies
`> tidyverse_logo()`	Get tidyverse logo, using ASCII or unicode characters
`> tidyverse_packages()`	List all tidyverse packages
`> tidyverse_update()`	Update tidyverse packages

Loading in the data

`> library(datasets)`	Load the datasets package
`> library(gapminder)`	Load the gapminder package
`> attach(iris)`	Attach iris data to the R search path

dplyr

Filter

`filter()` allows you to select a subset of rows in a data frame.

`> iris %>% filter(Species=="virginica")`	Select iris data of species "virginica"
`> iris %>% filter(Species=="virginica", Sepal.Length > 6)`	Select iris data of species "virginica" and sepal length greater than 6.

Arrange

`arrange()` sorts the observations in a dataset in ascending or descending order based on one of its variables.

`> iris %>% arrange(Sepal.Length)`	Sort in ascending order of sepal length
`> iris %>% arrange(desc(Sepal.Length))`	Sort in descending order of sepal length

Combine multiple `dplyr` verbs in a row with the pipe operator `%>%`:

`> iris %>% filter(Species=="virginica") %>% arrange(desc(Sepal.Length))`	Filter for species "virginica" then arrange in descending order of sepal length

Mutate

`mutate()` allows you to update or create new columns of a data frame.

`> iris %>% mutate(Sepal.Length=Sepal.Length*10)`	Change Sepal.Length to be in millimeters
`> iris %>% mutate(SLMm=Sepal.Length*10)`	Create a new column called SLMm

Combine the verbs `filter()`, `arrange()`, and `mutate()`:

```
> iris %>%
    filter(Species=="Virginica") %>%
    mutate(SLMm=Sepal.Length*10) %>%
    arrange(desc(SLMm))
```

Summarize

`summarize()` allows you to turn many observations into a single data point.

`> iris %>% summarize(medianSL=median(Sepal.Length))`	Summarize to find the median sepal length
`> iris %>% filter(Species=="virginica") %>% summarize(medianSL=median(Sepal.Length))`	Filter for virginica then summarize the median sepal length

You can also summarize multiple variables at once:

```
> iris %>%
    filter(Species=="virginica") %>%
    summarize(medianSL=median(Sepal.Length),
              maxSL=max(Sepal.Length))
```

`group_by()` allows you to summarize within groups instead of summarizing the entire dataset:

`> iris %>% group_by(Species) %>% summarize(medianSL=median(Sepal.Length), maxSL=max(Sepal.Length))`	Find median and max sepal length of each species
`> iris %>% filter(Sepal.Length>6) %>% group_by(Species) %>% summarize(medianPL=median(Petal.Length), maxPL=max(Petal.Length))`	Find median and max petal length of each species with sepal length > 6

ggplot2

Scatter plot

Scatter plots allow you to compare two variables within your data. To do this with ggplot2, you use `geom_point()`

`> iris_small <- iris %>% filter(Sepal.Length > 5)` `> ggplot(iris_small, aes(x=Petal.Length, y=Petal.Width)) + geom_point()`	Compare petal width and length

Additional Aesthetics

- Color

```
> ggplot(iris_small, aes(x=Petal.Length,
                         y=Petal.Width,
                         color=Species)) +
    geom_point()
```

- Size

```
> ggplot(iris_small, aes(x=Petal.Length,
                         y=Petal.Width,
                         color=Species,
                         size=Sepal.Length)) +
    geom_point()
```

Faceting

```
> ggplot(iris_small, aes(x=Petal.Length,
                         y=Petal.Width)) +
    geom_point()+
    facet_wrap(~Species)
```

Line Plots

```
> by_year <- gapminder %>%
    group_by(year) %>%
    summarize(medianGdpPerCap=median(gdpPercap))
> ggplot(by_year, aes(x=year,
                      y=medianGdpPerCap))+
    geom_line()+
    expand_limits(y=0)
```

Bar Plots

```
> by_species <- iris %>%
    filter(Sepal.Length>6) %>%
    group_by(Species) %>%
    summarize(medianPL=median(Petal.Length))
> ggplot(by_species, aes(x=Species,
                         y=medianPL)) +
    geom_col()
```

Histograms

```
> ggplot(iris_small, aes(x=Petal.Length))+
    geom_histogram()
```

Box Plots

```
> ggplot(iris_small, aes(x=Species,
                         y=Sepal.Width))+
    geom_boxplot()
```

DataCamp
Learn R for Data Science Interactively

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Tidyverse+Cheat+Sheet.pdf

Tidy data tools in R

- In the Tidyverse, you'll note that all packages function together seamlessly

- Many of the core commands have counterparts in SQL (e.g., `select`) and so are intuitive and transferable

- Typically the tidyverse commands will have the form: `command(data, optional variables)`

Example dataset `iris.t`

```
> iris.t
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# ... with 140 more rows
```

```
> summary(iris.t)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa     :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Tidy data using `dplyr`

- `filter()`

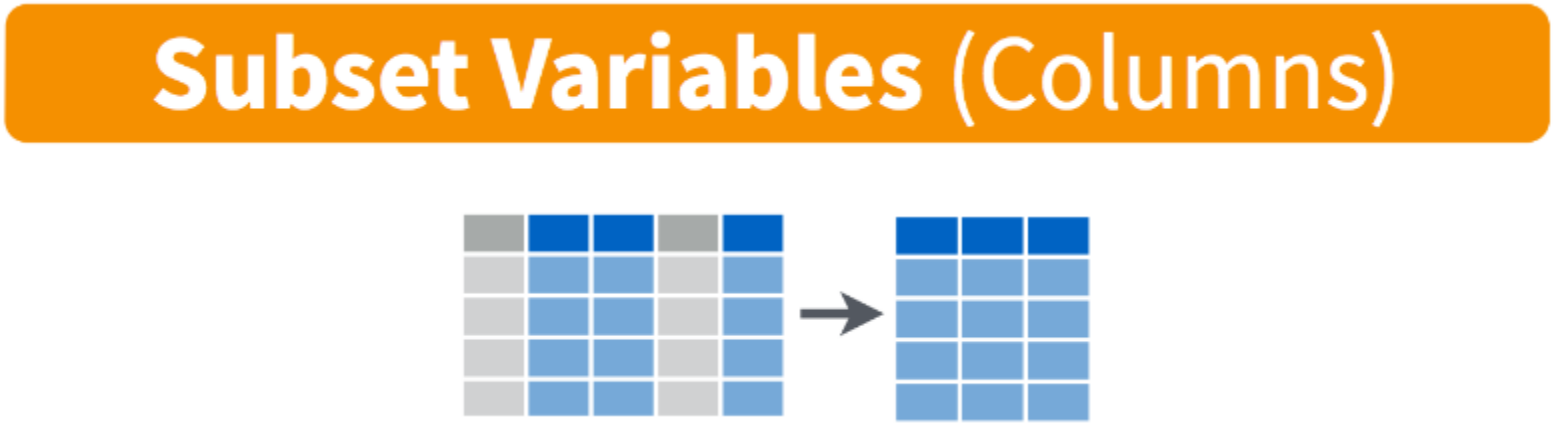

```
# subset rows of data using filter()
filter(iris.t, Species == 'virginica')

filter(iris.t, Species %in% c('setosa', 'virginica'))

filter(iris.t, Petal.Length > 6.0 & Petal.Width < 2)
```

Tidy data using `dplyr`

- `select()`




```
# subset columns of data using select()
select(iris.t, Species, Petal.Width)

select(iris.t, -Petal.Width, -Petal.Length)

select(iris.t, contains("Length"))
```

Tidy data using `dplyr`

Helper functions for select - ?select

```
select(iris, contains("."))
```
Select columns whose name contains a character string.

```
select(iris, ends_with("Length"))
```
Select columns whose name ends with a character string.

```
select(iris, everything())
```
Select every column.

```
select(iris, matches(".t."))
```
Select columns whose name matches a regular expression.

```
select(iris, num_range("x", 1:5))
```
Select columns named x1, x2, x3, x4, x5.

```
select(iris, one_of(c("Species", "Genus")))
```
Select columns whose names are in a group of names.

```
select(iris, starts_with("Sepal"))
```
Select columns whose name starts with a character string.

```
select(iris, Sepal.Length:Petal.Width)
```
Select all columns between Sepal.Length and Petal.Width (inclusive).

```
select(iris, -Species)
```
Select all columns except Species.

Tidy data using `dplyr`

- `mutate()`


```
# add new variables (columns) using mutate()
mutate(iris.t, Sepal.Area = Sepal.Length * Sepal.Width)

mutate(iris.t, Sepal.Petal.Diff = Sepal.Length - Petal.Length)

mutate(iris.t, Avg.Petal.Length = mean(Petal.Length),
       Dev.from.Avg = Petal.Length - Avg.Petal.Length)
```

Tidy data using `dplyr`

- `group_by()` and `summarise()`

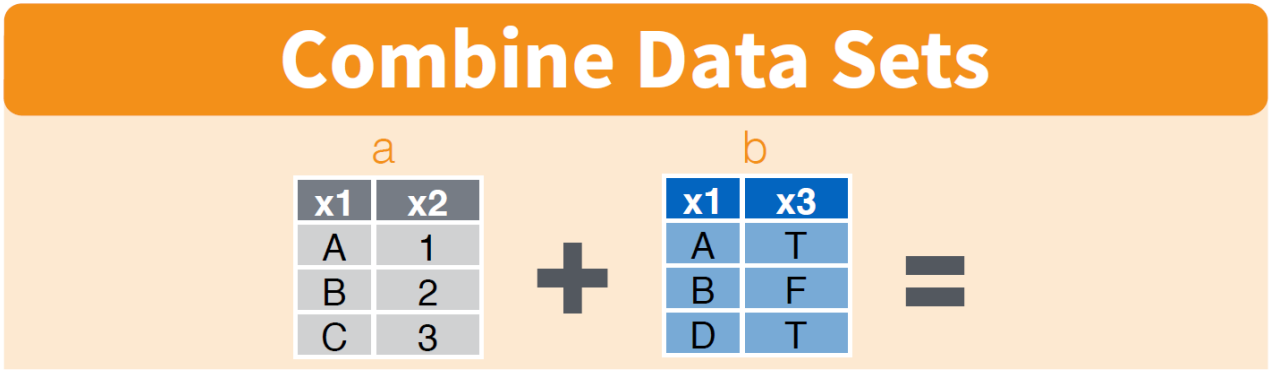

```
# generate overall and group-level summaries with group_by() and summarise()
summarise(iris.t, Avg.Petal.Length = mean(Petal.Length))

iris.grouped <- group_by(iris.t, Species)
summarise(iris.grouped, Avg.Petal.Length = mean(Petal.Length))

iris.grouped <- group_by(iris.t, Species)
filter(iris.grouped, rank(desc(Petal.Length)) < 3)
```

Tidy data using `dplyr`

- `left_join()`, `inner_join()` and `full_join()`

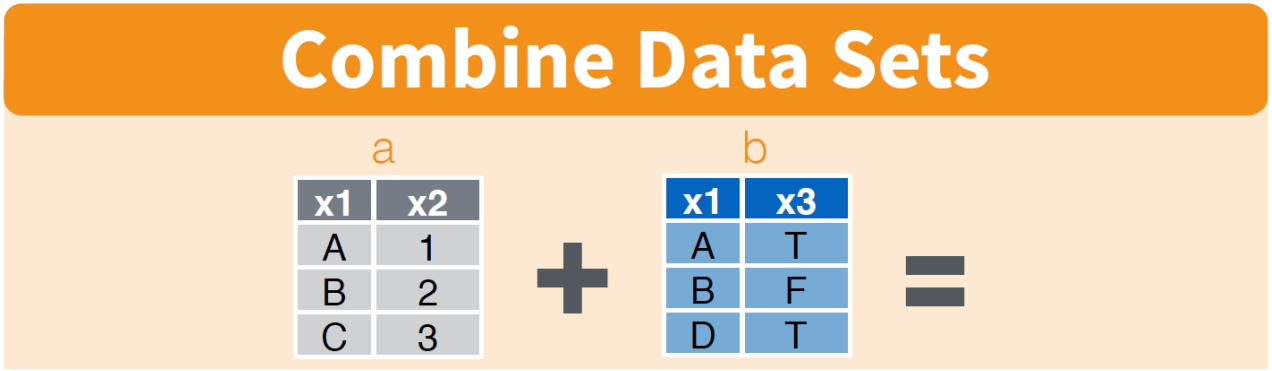



```
> petal.data
# A tibble: 150 x 4
      ID Species Petal.Length Petal.Width
   <int> <fct>           <dbl>       <dbl>
 1    23 setosa            1           0.2
 2    14 setosa            1.1         0.1
 3    15 setosa            1.2         0.2
 4    36 setosa            1.2         0.2
 5     3 setosa            1.3         0.2
 6    17 setosa            1.3         0.4
 7    37 setosa            1.3         0.2
 8    39 setosa            1.3         0.2
 9    41 setosa            1.3         0.3
10    42 setosa            1.3         0.3
# ... with 140 more rows
```

```
> sepal.data
# A tibble: 150 x 4
      ID Species Sepal.Length Sepal.Width
   <int> <fct>           <dbl>       <dbl>
 1    14 setosa            4.3         3
 2     9 setosa            4.4         2.9
 3    39 setosa            4.4         3
 4    43 setosa            4.4         3.2
 5    42 setosa            4.5         2.3
 6     4 setosa            4.6         3.1
 7     7 setosa            4.6         3.4
 8    23 setosa            4.6         3.6
 9    48 setosa            4.6         3.2
10     3 setosa            4.7         3.2
# ... with 140 more rows
```

Tidy data using `dplyr`

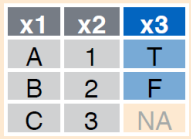
- `left_join()`, `inner_join()` and `full_join()`




```
# join the two datasets back together by species and ID so that the
# original information is recovered
left_join(petal.data, sepal.data, by = c("Species", "ID"))
```

Tidy data using `dplyr`

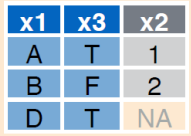
Mutating Joins



x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::**left_join(a, b, by = "x1")**

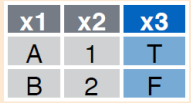
Join matching rows from b to a.



x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::**right_join(a, b, by = "x1")**

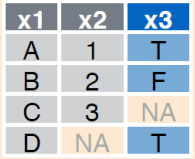
Join matching rows from a to b.



x1	x2	x3
A	1	T
B	2	F

dplyr::**inner_join(a, b, by = "x1")**

Join data. Retain only rows in both sets.

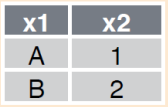


x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::**full_join(a, b, by = "x1")**

Join data. Retain all values, all rows.

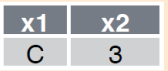
Filtering Joins



x1	x2
A	1
B	2

dplyr::**semi_join(a, b, by = "x1")**

All rows in a that have a match in b.



x1	x2
C	3

dplyr::**anti_join(a, b, by = "x1")**

All rows in a that do not have a match in b.

For more details see: http://stat545.com/bit001_dplyr-cheatsheet.html

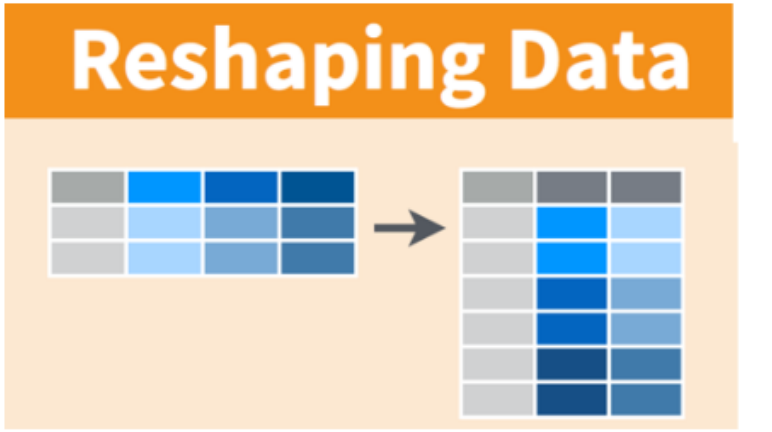
Piping

- In multi-step data manipulation, tidy data tools provide a convenient way to link individual operations together into a single block of code

```
# Note that operations can be linked together using
#  the 'piping' operator '%>%'
iris.t %>%
  mutate(Dev.from.Average = Petal.Length - mean(Petal.Length)) %>%
  group_by(Species) %>%
  summarise(Avg.Dev = mean(Dev.from.Average))
```

Tidy data using `tidyr`

- Wide-to-long format: `gather(key=, value=)`

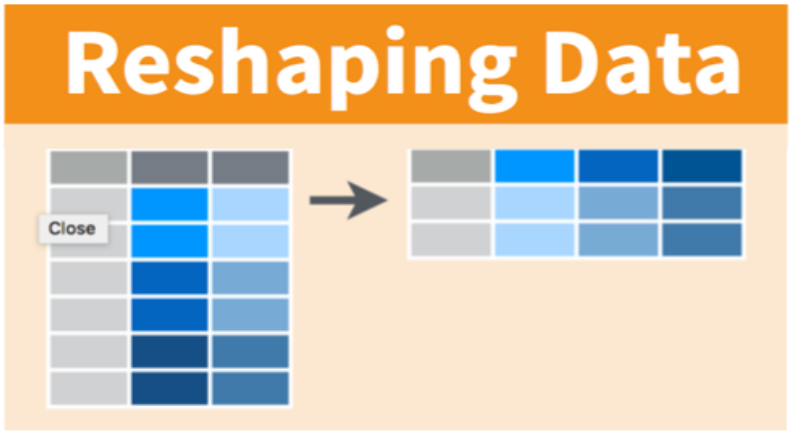



```
# reshaping data: gather columns in key-value pairs (wide-to-long) using gather()
iris.t %>%
  select(Species, Sepal.Length, Sepal.Width) %>%
  gather(key = "Sepal.Attribute", value = "Measurement",
         Sepal.Length, Sepal.Width)

iris.t %>%
  gather(key = "Attribute", value = "Measurement",
         Sepal.Length, Sepal.Width, Petal.Length, Petal.Width)
```

Tidy data using `tidyr`

- **Long-to-wide format:** `spread(key=, value=)`



Getting started – tidy data paper

Hadley Wickham's

Tidy data

Hadley Wickham.
Tidy data.
The Journal of Statistical Software, vol. 59, 2014.
Download: pre-print | from publisher

A huge amount of effort is spent cleaning data to get it ready for analysis, but there has been little research on how to make data cleaning as easy and effective as possible. This paper tackles a small, but important, component of data cleaning: data tidying. Tidy datasets are easy to manipulate, model and visualize, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table. This framework makes it easy to tidy messy datasets because only a small set of tools are needed to deal with a wide range of un-tidy datasets. This structure also makes it easier to develop tidy tools for data analysis, tools that both input and output tidy datasets. The advantages of a consistent data structure and matching tools are demonstrated with a case study free from mundane data manipulation chores.

```
@Article{tidy-data,
  author = {Hadley Wickham},
  issue = {10},
  journal = {The Journal of Statistical Software},
  selected = {TRUE},
  title = {Tidy data},
  url = {http://www.jstatsoft.org/v59/i10/},
  volume = {59},
  year = {2014},
  bdsk-url-1 = {http://www.jstatsoft.org/v59/i10/},
}
```

Built with R, the bibtex package, and brew. Styled with skeleton and subtlepatterns. Hosted on github.

http://vita.had.co.nz/papers/tidy-data.html

Getting started – some resources

- https://r4ds.had.co.nz/
- https://www.r-bloggers.com/hadley-wickhams-dplyr-tutorial-at-user-2014-part-1/
- https://www.r-bloggers.com/hadley-wickhams-dplyr-tutorial-at-user-2014-part-2/
- https://www.rstudio.com/resources/cheatsheets/
  - https://github.com/rstudio/cheatsheets/blob/master/data-transformation.pdf
  - https://github.com/rstudio/cheatsheets/blob/master/data-import.pdf
- http://jules32.github.io/2016-07-12-Oxford/dplyr_tidyr/#
- https://rstudio-pubs-static.s3.amazonaws.com/109948_876c577ae6424d47b529fce960db7bf4.html#1