Political Analysis of Social Media Data

R Basics I

Gregory Eady

Assistant Professor
Department of Political Science
University of Copenhagen

Office: 18-2-21
E-mail: gregory.eady@ifs.ku.dk
Twitter: @GregoryEady

Today

1. R
2. R exercises

Install R:

`https://cran.r-project.org/mirrors.html`

Install RStudio:

`https://rstudio.com/products/rstudio/download`

Errors in R are often cryptic, but there are clever strategies for learning:

VICTORIA FINE • AUG 31, 2015 • 4:59 AM

I Was Intimidated by Coding Until I Learned This Secret Strategy: Googling

The internet will make those bad words go away

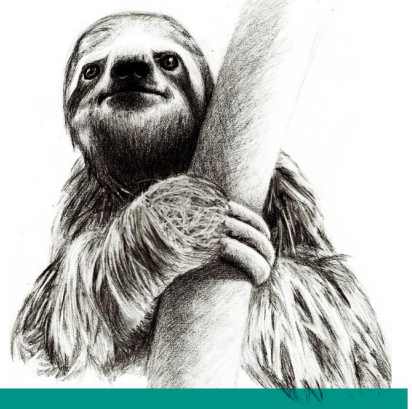
Essential

Googling the Error Message

O RLY?

The Practical Developer
@ThePracticalDev

Cutting corners to meet arbitrary management deadlines



Essential

Copying and Pasting from Stack Overflow

O'REILLY®

The Practical Developer
@ThePracticalDev

Software can be chaotic, but we make it work

Expert

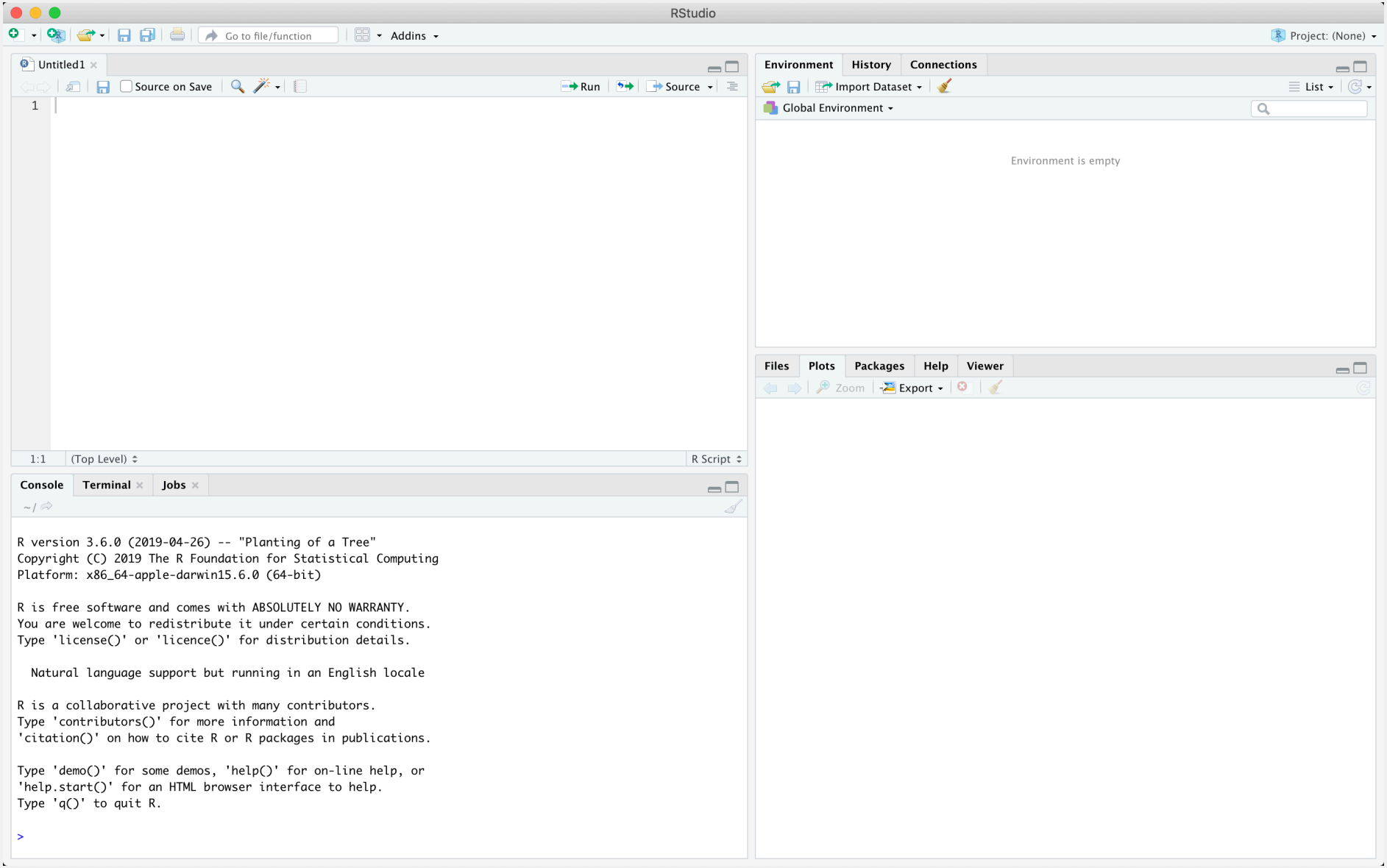
Trying Stuff Until it Works

O RLY?

The Practical Developer
@ThePracticalDev

RStudio

- An Integrated Development Environment (IDE)


RStudio
Go to file/function
Addins
Project: (None)
Untitled1
Source on Save
Run
Source
1
1:1
(Top Level)
R Script
Environment
History
Connections
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Zoom
Export
Console
Terminal
Jobs
~/
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>

Basics:

- Assignment operator: **`<-`**
- Comment and uncomment: **`Ctrl + Shift + c`**
- Run line of code: **`Ctrl + Enter`**
- Run code in console: **`Enter`**
- Clear console: **`Ctrl + l`**
- Indent: **`Tab`**
- Outdent: **`Shift + Tab`**

Basics:

- Calculate: `(2 + 5) / 7`
- Create a variable:
  - `var_1 <- c(1, 4, 5, 7)`
- Create another variable:
  - `var_2 <- c(1, 15, 25, -1)`
- Elements of a vector: `var_1[1:3]`
- Arithmetic with variables:
  - `var_1 * var_2`
  - `var_1 * 2`
- Create a data.frame
  - `D <- data.frame(var_1, x = c(1, 2, 3, 4))`
  - `D <- data.frame(var_1, x = c(1, 2, 3))`

Basics:

- Variables in data: **`names(D)`**
- Index variable within a data.frame:
  - **`D$var1[2:4]`**
- Create new variable within existing data.frame:
  - **`D$new_var <- c(100, 200, 300, 400)`**
- Manipulate variables within existing data.frame:
  - **`D$new_var <- D$new_var * 5`**

Basics:

- Install a package: **`install.packages("ggplot2")`**
- Load a package: **`library(ggplot2)`**
- Load data from a package: **`data(iris)`**
- What variables are in the data? **`names(iris)`**
- Histogram of a variable:
  - **`ggplot(iris, aes(x = Sepal.Length)) + geom_histogram()`**

Installing and loading libraries in R

```
# Install the tidyverse package
install.packages("tidyverse")
library(tidyverse)
```

Creating vectors in R

```
# Create a variable of random words
var_words <- c("Apple", "Orange", "Pear", "Cherry")
var_words
class(var_words)

# Create a variable of random numbers
var_num <- c(123, 91, -50, 15200, -1, 0, 0)
var_num
class(var_num)
```

Creating vectors in R

```
# Create a variable of numbers and words?
var_num_words <- c(123, 91, -50, "Apple", 124)
var_num_words
class(var_num_words)
```

Creating a data.frame in R

```
# Create some variables
var_1 <- c(123, 91, -50, 41, 124)
var_2 <- c("Apple", "Peach", "Pear", "Lemon", "Pineapple")

# Put the variables in a data.frame
D <- data.frame(var_1, var_2)

# Look at the data.frame we created
D
```

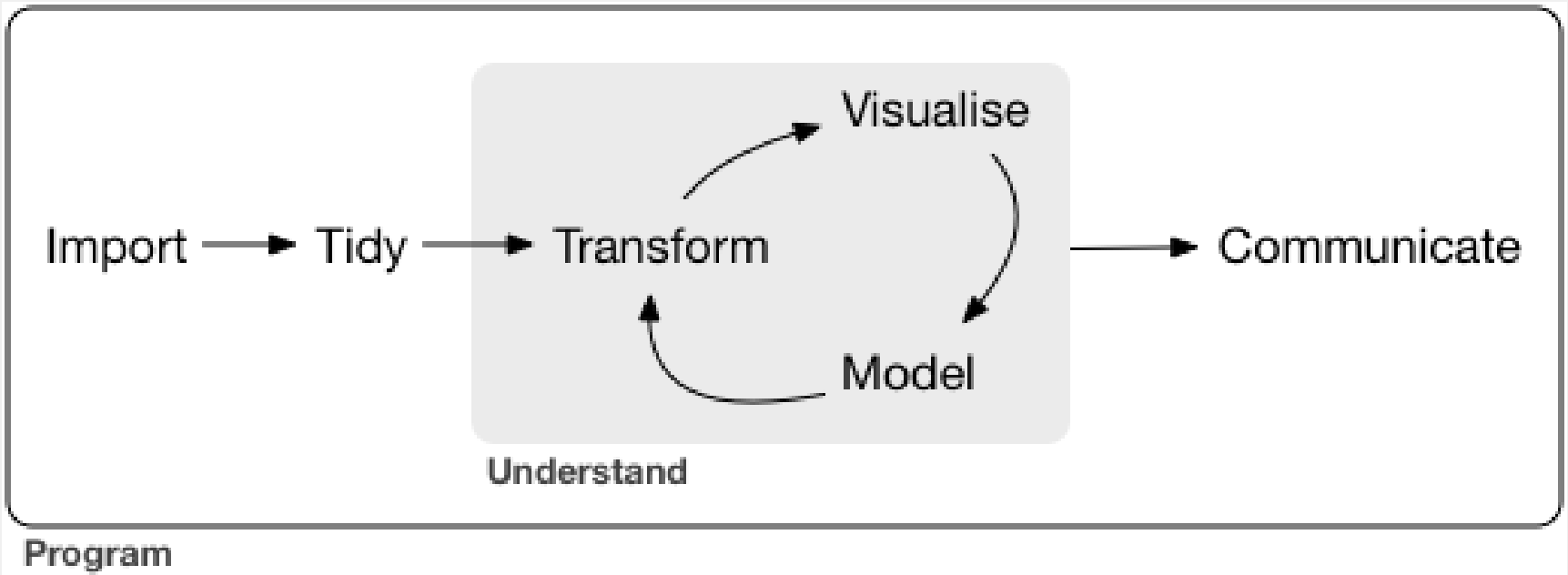
Creating a data.frame in R

```
# Create some variables
var_1 <- c(123, 91, -50, 41, 124)
var_2 <- c("Apple", "Peach", "Pear", "Lemon", "Pineapple")

# Put the variables in a data.frame
D <- data.frame(var_1, var_2,
                var_3 = c("A", "B", "B", "A", "B"))

# Look at the data.frame we created
D
```

However:

- We rarely create data from scratch
- We often merge in data from multiple sources
- These data are often messy
- Social media data in particular frequently need to be cleaned

Workflow


Import
Tidy
Transform
Visualise
Model
Understand
Communicate
Program

Messy data

Name	Publisher	Sales (mil. units)	Critic	User
Wii Sports	Nintendo	82.53	Score = 76, Count = 51	Score = 8, Count = 322
Mario Kart Wii	Nintendo	35.52	Score = 82, Count = 73	Score = 8.3, Count = 709
Wii Sports Resort	Nintendo	32.77	Score = 80, Count = 73	Score = 8, Count = 192
New Super Mario Bros.	Nintendo	29.80	Score = 89, Count = 65	Score = 8.5, Count = 431
Wii Play	Nintendo	28.92	Score = 58, Count = 41	Score = 6.6, Count = 129

Source: https://rss.onlinelibrary.wiley.com/doi/full/10.1111/j.1740-9713.2018.01169.x

Tidy data

Name	Publisher	Sales (mil. units)	Score	
Wii Sports	Nintendo	82.53	Critic	76
Wii Sports	Nintendo	82.53	User	80
Mario Kart Wii	Nintendo	35.52	Critic	82
Mario Kart Wii	Nintendo	35.52	User	83
Wii Sports Resort	Nintendo	32.77	Critic	80
Wii Sports Resort	Nintendo	32.77	User	80
New Super Mario Bros.	Nintendo	29.80	Critic	89
New Super Mario Bros.	Nintendo	29.80	User	85
Wii Play	Nintendo	28.92	Critic	58
Wii Play	Nintendo	28.92	User	66

Source: https://rss.onlinelibrary.wiley.com/doi/full/10.1111/j.1740-9713.2018.01169.x

How do we get from messy to tidy data?

Action	Function in `dplyr`
Filter	**`filter()`**
Aggregate	**`group_by()`** and **`summarize()`**
Sort	**`arrange()`**
Reshape	**`pivot_wider()`** and **`pivot_longer()`**
Recode	**`recode()`**

Data Transformation with dplyr : : CHEAT SHEET

dplyr

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)
ends_with(match)
matches(match)
num_range(prefix, range)
one_of(…)
starts_with(match)
:, e.g. mpg:cyl
-, e.g, -Species

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()

```
iris %>% mutate(Species = case_when(
              Species == "versicolor" ~ "versi",
              Species == "virginica"  ~ "virgi",
                          TRUE ~ Species))
```

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

Download the following file from the course website

`GM.csv`

Gapminder data

Loading data in R

```
# Load the tidyverse library
library(tidyverse)

# Set your working directory (where you saved your file)
setwd("~/Downloads/")

# Load GapMinder data
GM <- read_csv("GM.csv")
```

Note: The path (or file name) will depend on where you save the .csv file.

Gapminder variables

Variable	Description
`country`	Country
`continent`	Continent that country is located on
`year`	Year
`lifeExp`	Life expectancy at birth
`pop`	Total population size
`gdpPercap`	GDP (per capita)

```
> # Look at the loaded data
> GM
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop gdpPercap
   <chr>       <chr>     <dbl>   <dbl>    <dbl>     <dbl>
 1 Afghanistan Asia       1952    28.8  8425333       779.
 2 Afghanistan Asia       1957    30.3  9240934       821.
 3 Afghanistan Asia       1962    32.0 10267083       853.
 4 Afghanistan Asia       1967    34.0 11537966       836.
 5 Afghanistan Asia       1972    36.1 13079460       740.
 6 Afghanistan Asia       1977    38.4 14880372       786.
 7 Afghanistan Asia       1982    39.9 12881816       978.
 8 Afghanistan Asia       1987    40.8 13867957       852.
 9 Afghanistan Asia       1992    41.7 16317921       649.
10 Afghanistan Asia       1997    41.8 22227415       635.
# … with 1,694 more rows
```

```r
# Look at the first (6) rows
head(GM)

# Look at the last (6) rows
tail(GM)

# Show names of variables in the data
names(GM)

# Descriptive statistics of all variables
summary(GM)
```

```
# Look at a single variable
GM$gdpPercap

# Look at a specific values of a single variable
GM$gdpPercap[50:60]
```

Now, let's say we want to do the following:

1. Group the observations by country
2. Calculate average life expectancy and median population over each period for each country
3. Reduce to countries with more than 10M inhabitants
4. Sort reverse alphabetically by country

```
##### Solution 1 (break into pieces)

# Split data by country
By_country <- group_by(GM, country)

# Calculate avg. life expectancy and median pop. per country
Country_sum <- summarize(By_country,
                         avg_life_exp = mean(lifeExp),
                         median_pop = median(pop))

# Subset by countries with higher than 10 million inhabitants
Big_countries <- filter(Country_sum, median_pop > 10000000)

# Sort the data by country and life expectancy
Result <- arrange(Big_countries, desc(country))
```

```
##### Solution 2 (do it all at once)
Result_2 <- GM %>%
            group_by(country) %>%
            summarize(avg_life_exp = mean(lifeExp),
                      median_pop = median(pop)) %>%
            filter(median_pop > 10000000) %>%
            arrange(desc(country))
```

```r
##### Create variable as the log of population
Result_2 <- Result_2 %>%
            mutate(log_median_pop = log(median_pop))

##### Or similarly without mutate()
Result_2$log_median_pop <- log(Result_2$median_pop)
```

To-do now:

- Complete the DataCamp course “Introduction to R” listed on the course website.

To-do for next class:

- DataCamp exercises on R and tidyverse