R for Absolute Beginners and Aspiring Data Scientists

Jen Stirrup

Power BI Gebruikersdag 2018

Power BI Gebruikersgroep

Motion10

VALID
STAY AHEAD

hillstar®

SOGETI

missefficiency
Laat MS Office voor ú werken

QNH
PARTNERSHIP IN IT

avanade

Quanto
collective analytics

macaw
Challenge accepted.

VISSER & VAN BAARS
THE BI & BIG DATA NETWORK

SUMMAVIEW
TEAM BUSINESS INTELLIGENCE

Directing Data
DATADOGS

DataScenarios
Connecting data to insights

ilionx

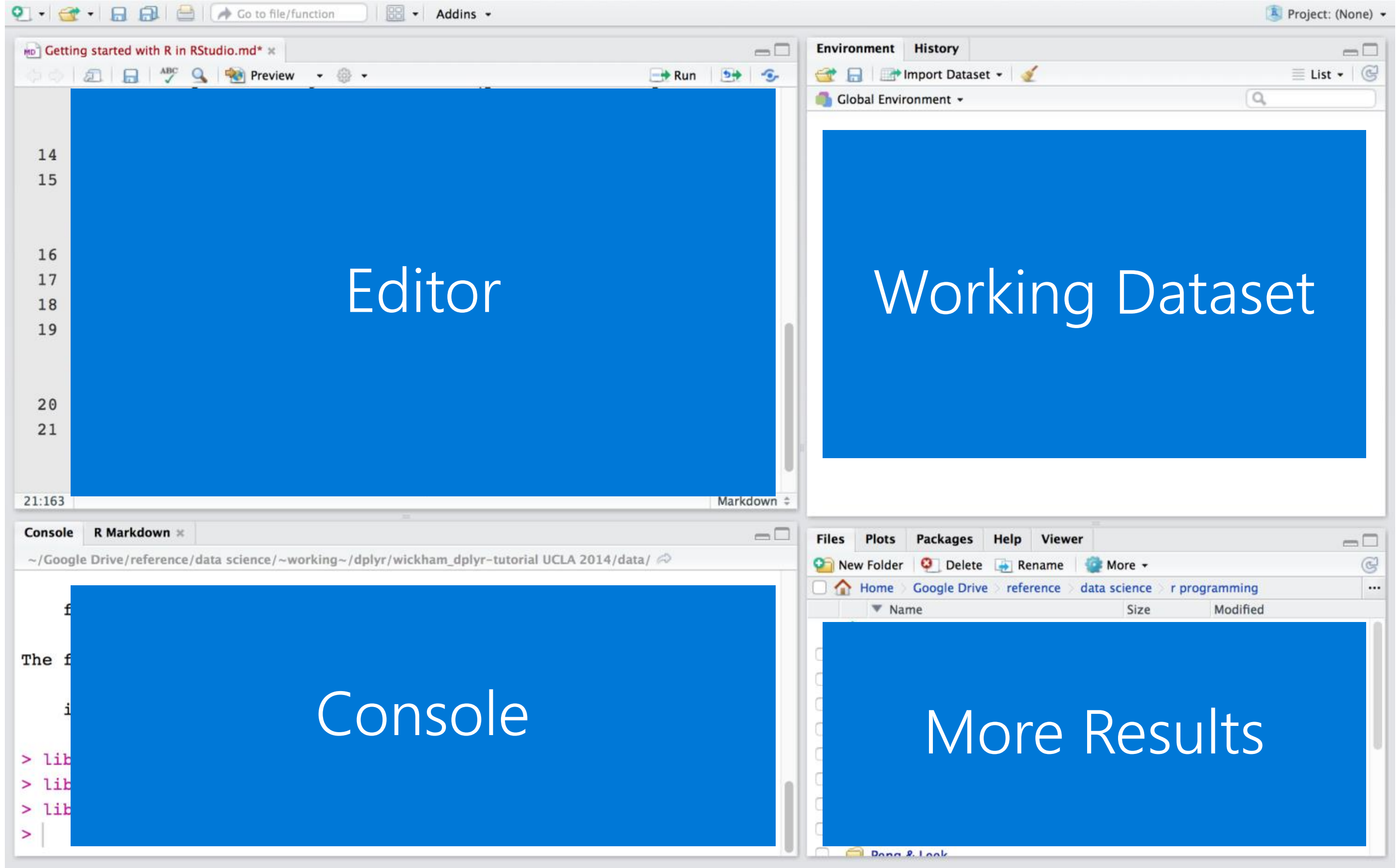
Literate Programming - Notebooks

- Notebooks combine code, results and documentation to identify what the code is doing
- It forms a train of thought
- It also produces reports for your team-mates

Literate Programming - Notebooks

- **Chunks** are executed independently and interactively
- **Output** appears in the gutter below

RStudio


Addins
Project: (None)
Getting started with R in RStudio.md*
Preview
Run
Editor
21:163
Markdown
Environment
History
Import Dataset
List
Global Environment
Working Dataset
Console
R Markdown
~/Google Drive/reference/data science/~working~/dplyr/wickham_dplyr-tutorial UCLA 2014/data/
Console
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
Home
Google Drive
reference
data science
r programming
Name
Size
Modified
More Results

Tidy Data

Tidy code is easier to write, read, maintain and frequently even faster than the base R counterparts.

It is also easier to learn.

So here we are!

Tidy Data

- **Tidy Data** is a standard approach to structure datasets
- Good for **Data Analysis** and **Data Visualization**
- **Variables** make up the **columns**
- **Observations** make up the **rows**
- **Values** go into **cells**

Tidy Data

- A **Variable** is a measurement
- Also known as:
  - Independent or dependent variables
  - Features – this is Microsoft’s terminology
  - Predictors – (machine learning background)
  - Outcomes – (social sciences background)
  - The Response (if you have a statistics background)
  - Attributes (if you have a dimensional modelling background)

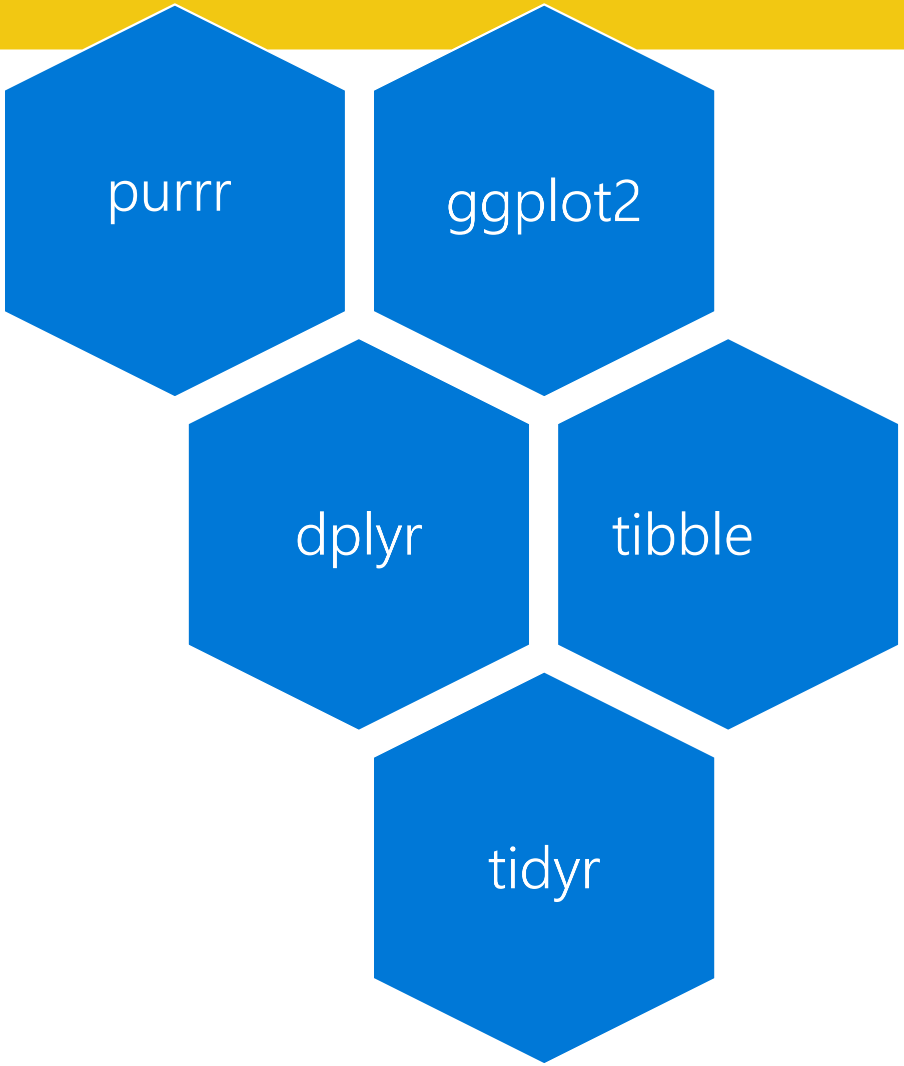
Tidy Data

- A **Variable** can fall into three categories:
- Fixed Variables
  - Known variables prior to the start of the investigation
- Measured Variables
  - Data that’s captured during the investigative process
- Derived Variables
  - Think of a calculated column in DAX or SQL

Tidyverse

Core Packages	Data Manipulation	Data Import	Modelling
ggplot2	hms	DBI	modelr
dplyr	stringer	haven	broom
tidyr	lubridate	httr	
readr	forcats	jsonlite	
purrr		readxl	
tibble		xml2	

Tidyverse Core Packages



purrr

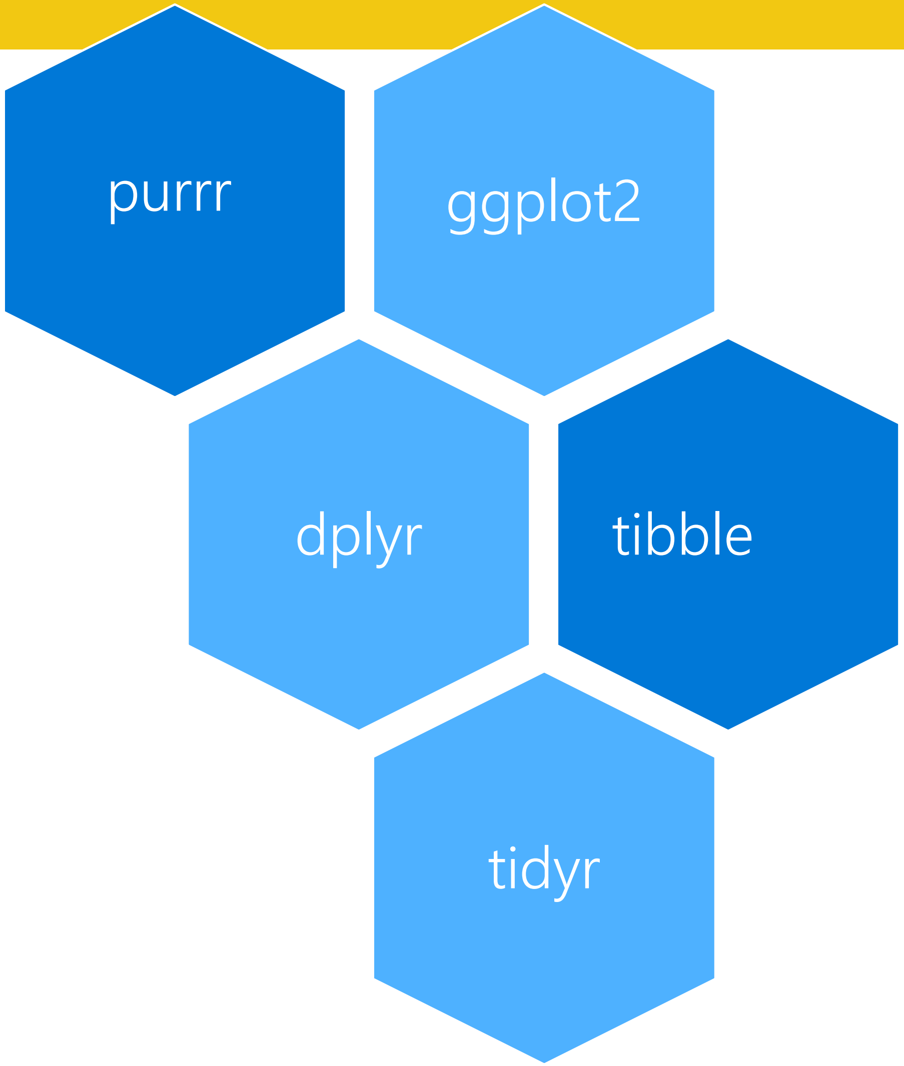
ggplot2

dplyr

tibble

tidyr

What we will cover today...



purrr

ggplot2

dplyr

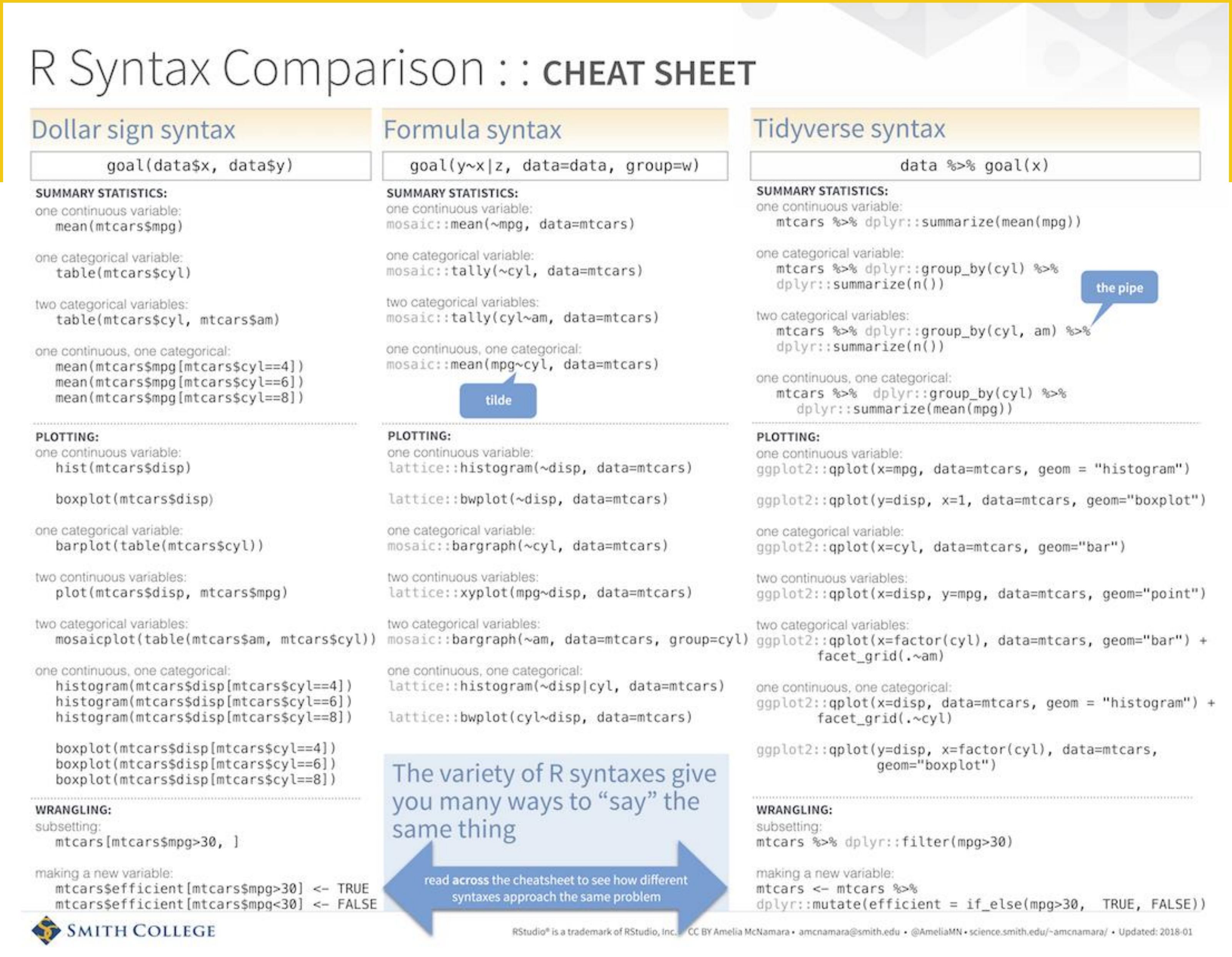
tibble

tidyr

Tidy Data

- A **Variable** can fall into three categories:
- Fixed Variables
  - Known variables prior to the start of the investigation
- Measured Variables
  - Data that's captured during the investigative process
- Derived Variables
  - Think of a calculated column in DAX or SQL

Tidy Data



R Syntax Comparison : : CHEAT SHEET

Dollar sign syntax

```
goal(data$x, data$y)
```

SUMMARY STATISTICS:

one continuous variable:
```
   mean(mtcars$mpg)
```

one categorical variable:
```
   table(mtcars$cyl)
```

two categorical variables:
```
   table(mtcars$cyl, mtcars$am)
```

one continuous, one categorical:
```
   mean(mtcars$mpg[mtcars$cyl==4])
   mean(mtcars$mpg[mtcars$cyl==6])
   mean(mtcars$mpg[mtcars$cyl==8])
```

PLOTTING:

one continuous variable:
```
   hist(mtcars$disp)

   boxplot(mtcars$disp)
```

one categorical variable:
```
   barplot(table(mtcars$cyl))
```

two continuous variables:
```
   plot(mtcars$disp, mtcars$mpg)
```

two categorical variables:
```
   mosaicplot(table(mtcars$am, mtcars$cyl))
```

one continuous, one categorical:
```
   histogram(mtcars$disp[mtcars$cyl==4])
   histogram(mtcars$disp[mtcars$cyl==6])
   histogram(mtcars$disp[mtcars$cyl==8])

   boxplot(mtcars$disp[mtcars$cyl==4])
   boxplot(mtcars$disp[mtcars$cyl==6])
   boxplot(mtcars$disp[mtcars$cyl==8])
```

WRANGLING:

subsetting:
```
   mtcars[mtcars$mpg>30, ]
```

making a new variable:
```
   mtcars$efficient[mtcars$mpg>30] <- TRUE
   mtcars$efficient[mtcars$mpg<30] <- FALSE
```

Formula syntax

```
goal(y~x|z, data=data, group=w)
```

SUMMARY STATISTICS:

one continuous variable:
```
mosaic::mean(~mpg, data=mtcars)
```

one categorical variable:
```
mosaic::tally(~cyl, data=mtcars)
```

two categorical variables:
```
mosaic::tally(cyl~am, data=mtcars)
```

one continuous, one categorical:
```
mosaic::mean(mpg~cyl, data=mtcars)
```

tilde

PLOTTING:

one continuous variable:
```
lattice::histogram(~disp, data=mtcars)

lattice::bwplot(~disp, data=mtcars)
```

one categorical variable:
```
mosaic::bargraph(~cyl, data=mtcars)
```

two continuous variables:
```
lattice::xyplot(mpg~disp, data=mtcars)
```

two categorical variables:
```
mosaic::bargraph(~am, data=mtcars, group=cyl)
```

one continuous, one categorical:
```
lattice::histogram(~disp|cyl, data=mtcars)

lattice::bwplot(cyl~disp, data=mtcars)
```

The variety of R syntaxes give you many ways to "say" the same thing

read across the cheatsheet to see how different syntaxes approach the same problem

Tidyverse syntax

```
data %>% goal(x)
```

SUMMARY STATISTICS:

one continuous variable:
```
   mtcars %>% dplyr::summarize(mean(mpg))
```

one categorical variable:
```
   mtcars %>% dplyr::group_by(cyl) %>%
   dplyr::summarize(n())
```

the pipe

two categorical variables:
```
   mtcars %>% dplyr::group_by(cyl, am) %>%
   dplyr::summarize(n())
```

one continuous, one categorical:
```
   mtcars %>%  dplyr::group_by(cyl) %>%
     dplyr::summarize(mean(mpg))
```

PLOTTING:

one continuous variable:
```
ggplot2::qplot(x=mpg, data=mtcars, geom = "histogram")

ggplot2::qplot(y=disp, x=1, data=mtcars, geom="boxplot")
```

one categorical variable:
```
ggplot2::qplot(x=cyl, data=mtcars, geom="bar")
```

two continuous variables:
```
ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")
```

two categorical variables:
```
ggplot2::qplot(x=factor(cyl), data=mtcars, geom="bar") +
          facet_grid(.~am)
```

one continuous, one categorical:
```
ggplot2::qplot(x=disp, data=mtcars, geom = "histogram") +
          facet_grid(.~cyl)

ggplot2::qplot(y=disp, x=factor(cyl), data=mtcars,
                geom="boxplot")
```

WRANGLING:

subsetting:
```
mtcars %>% dplyr::filter(mpg>30)
```

making a new variable:
```
mtcars <- mtcars %>%
dplyr::mutate(efficient = if_else(mpg>30,  TRUE, FALSE))
```

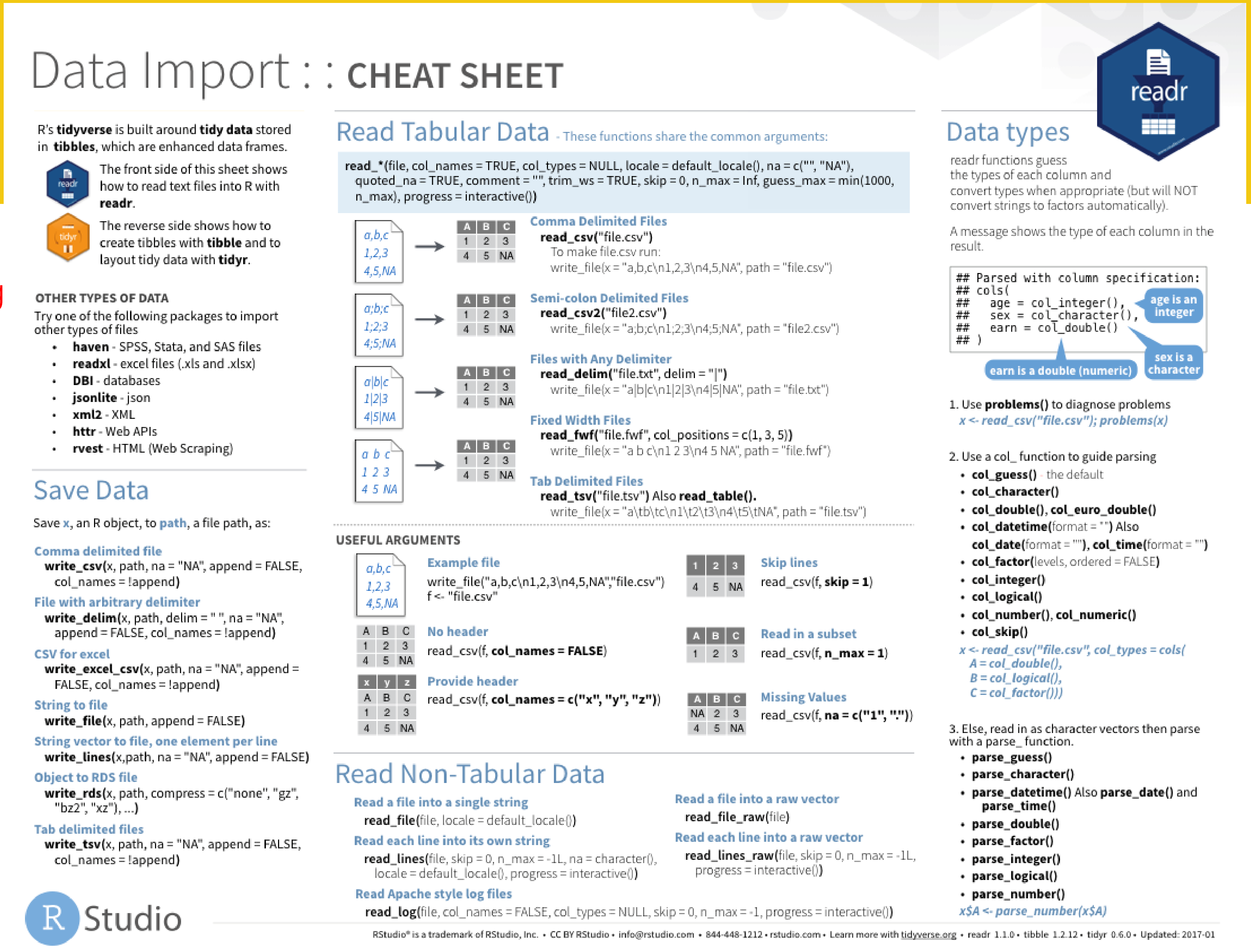
SMITH COLLEGE

RStudio® is a trademark of RStudio, Inc. • CC BY Amelia McNamara • amcnamara@smith.edu • @AmeliaMN • science.smith.edu/~amcnamara/ • Updated: 2018-01

https://www.rstudio.com/resources/cheatsheets/

Tidy Data

http://readr.tidyverse.org



Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save x, an R object, to path, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append)

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append)

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append)

String to file
write_file(x, path, append = FALSE)

String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE)

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...)

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)

RStudio

Read Tabular Data - These functions share the common arguments:

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),
  quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
  n_max), progress = interactive())
```

```
a,b,c
1,2,3
4,5,NA
```
→

A	B	C
1	2	3
4	5	NA

Comma Delimited Files
read_csv("file.csv")
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

```
a;b;c
1;2;3
4;5;NA
```
→

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv")
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

```
a|b|c
1|2|3
4|5|NA
```
→

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
read_delim("file.txt", delim = "|")
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

```
a b c
1 2 3
4 5 NA
```
→

A	B	C
1	2	3
4	5	NA

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5))
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

Tab Delimited Files
read_tsv("file.tsv") Also **read_table()**.
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

```
a,b,c
1,2,3
4,5,NA
```

Example file
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"

A	B	C
1	2	3
4	5	NA

No header
read_csv(f, **col_names = FALSE**)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv(f, **col_names = c("x", "y", "z")**)

1	2	3
4	5	NA

Skip lines
read_csv(f, **skip = 1**)

A	B	C
1	2	3

Read in a subset
read_csv(f, **n_max = 1**)

A	B	C
NA	2	3
4	5	NA

Missing Values
read_csv(f, **na = c("1", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale())

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive())

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())

Read a file into a raw vector
read_file_raw(file)

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive())

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

1. Use **problems()** to diagnose problems
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing
- **col_guess()** - the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime**(format = "") Also **col_date**(format = ""), **col_time**(format = "")
- **col_factor**(levels, ordered = FALSE)
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

*x <- read_csv("file.csv", col_types = cols(
  A = col_double(),
  B = col_logical(),
  C = col_factor()))*

3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

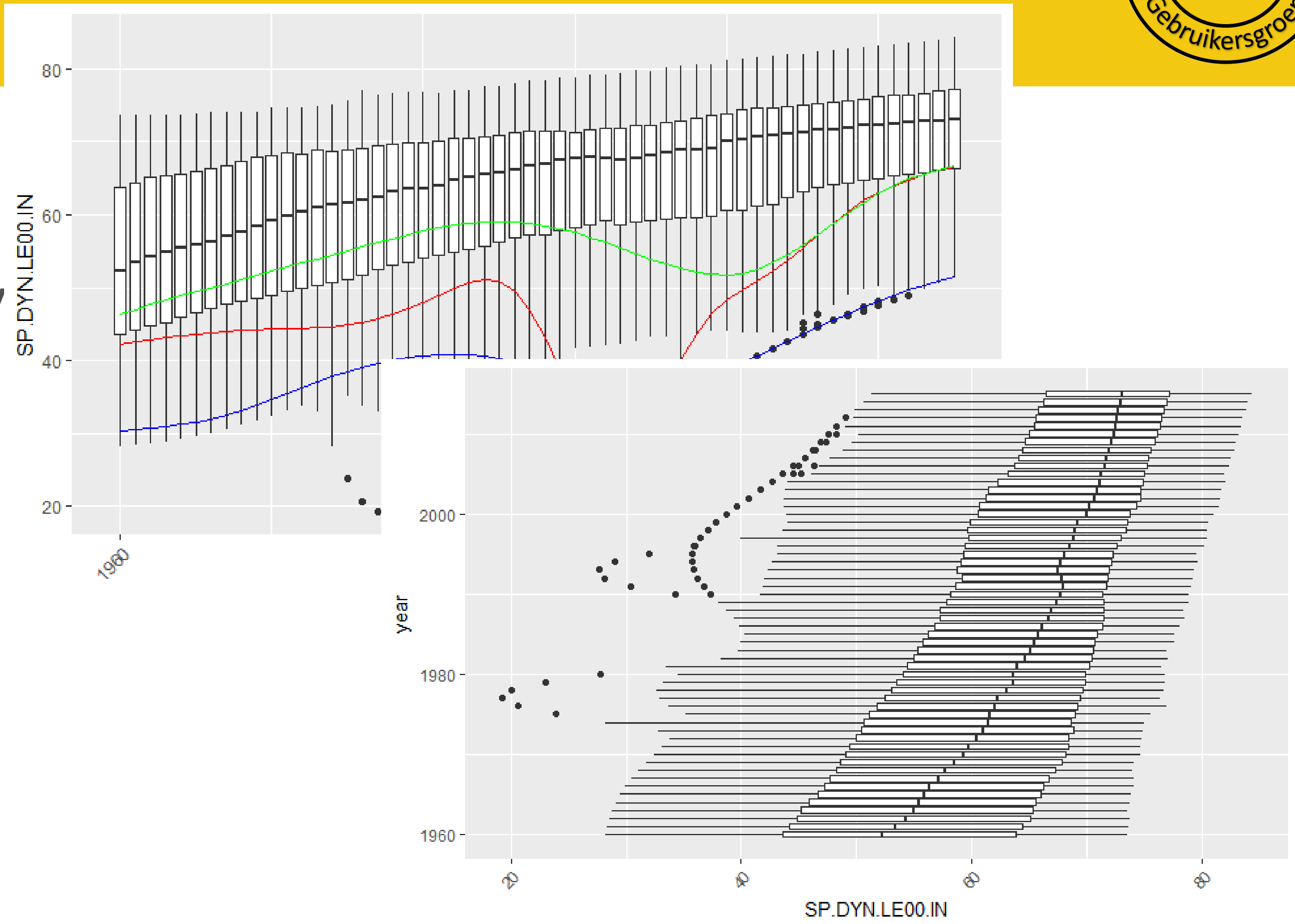
x$A <- parse_number(x$A)

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2017-01

https://www.rstudio.com/resources/cheatsheets/

Tidy Data

We can compare the life expectancy of different countries in order to understand the 'WHY' of the data.



Data Source: World Bank Data

readr

- Ingests data from different sources
- There are lots of options to work with the file
  - Headers
  - Limiters
  - https://cran.r-project.org/web/packages/readr/readr.pdf for more information

dplyr

- Easy data manipulation
- Built for data frames
- There are equivalents in SQL
- Written in C++ so it’s faster

dplyr

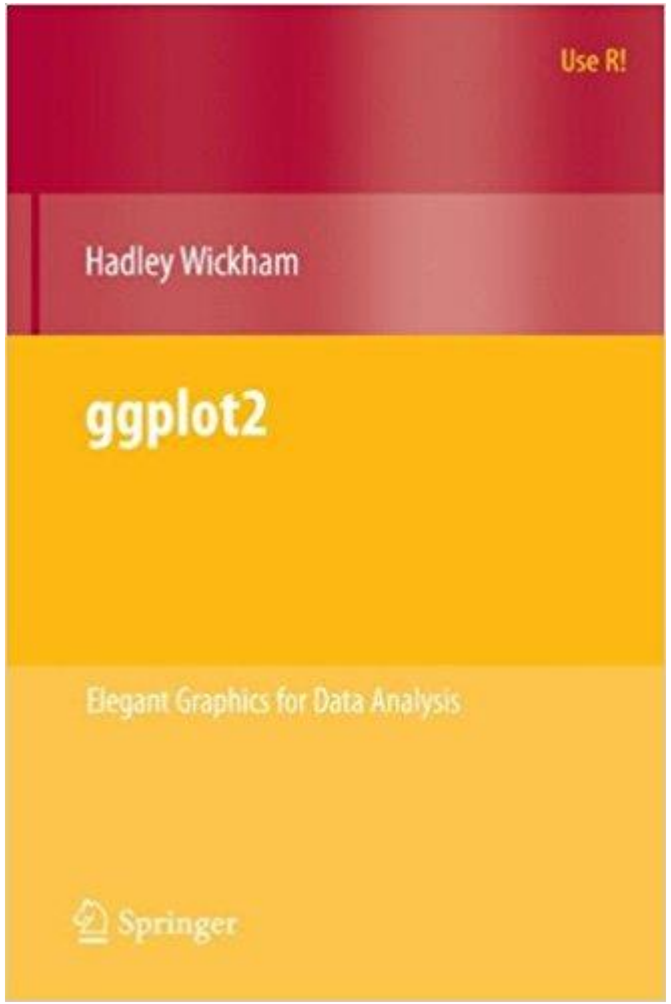
- 6 verbs for data manipulation
  - Select
  - Filter
  - Mutate
  - group_by
  - Summarize
  - Tally
- There are equivalents in SQL

dplyr

dplyr	SQL
SELECT	SELECT
FILTER	WHERE
MUTATE	Creating a calculation in the select statement
GROUP BY	GROUP BY
Summarize	SUM(), AVG() etc with GROUP BY

ggplot2

- ggplot2 is a plotting system for R
- It aims to take the good parts of base R graphics and none of the bad parts



Limitations of Running R in Power BI

- Only data frames are imported
  - In our example, we connect to the World Bank directly
- Complex and Vector columns are not imported
  - They are replaced with errors
- Time Limit
  - 30 minute limit. Why not use **Microsoft ML Server** instead to do the analytics piece, and serve the data to Power BI?
- Identify a full path to the R Working Directory
  - Not a Relative path

Bedankt!

Vergeet niet de evaluatie in te vullen!
Scan de QR code.

Power BI Gebruikersdag 2018

Power BI Gebruikersgroep

Motion10

VALID
STAY AHEAD

hillstar®

SOGETI

miss efficiency
Laat MS Office voor ú werken

QNH
PARTNERSHIP IN IT

avanade

Quanto
collective analytics

macaw
Challenge accepted.

VISSER & VAN BAARS
THE BI & BIG DATA NETWORK

SUMMAVIEW
TEAM BUSINESS INTELLIGENCE

Directing Data
DATADOGS

DataScenarios
Connecting data to insights

ilionx