Introduction to R

Prepared for Queensland Dept. of Treasury

Feb 11th, 2020

Josiah Parry

RStudio

Agenda

- Overview
  - What is R?
  - How are RStudio and R different?
  - Navigating RStudio
- R coding basics
- The tidyverse:
  - Reading, visualizing, and analyzing data.

RStudio®

Learning Objective

- Understand what R and RStudio are
- Differentiate between base R and packages
- A (loose) grasp on the grammar of graphics
- Be exposed to tidyverse code
- Not be *too* overwhelmed

What *is* R?

- The 18th letter in the alphabet
- The fourth letter in ***QWERTY***
- Software for statistical analysis

R is an integrated suite of software facilities for data manipulation, calculation and graphical display.

RStudio®

What is it good for?

- Statistics.
- But other things too.
- Creating graphics
- Machine learning
- Interactive Shiny Applications
- Web APIs
- Books
- Pretty much anything you can dream of

“My computer is just a boot-loader for R”

- *Jared Lander*

RStudio®

What makes it special?

- It is both **free** and **open-source**

“Free software is a matter of liberty, not price. To understand the concept, you should think of ‘free’ as in ‘free speech’, not as in ‘free beer’” - GNU Project

- New statistical techniques are developed in R
- R is a community driven, global project
- Designed to be extended

RStudio IDE

RStudio

R vs RStudio - what's what?

- R is the tool
- RStudio is where you interact with it
- RStudio is an IDE
    - "Integrated development environment"
- R is the car engine
- RStudio is the dashboard

RStudio®

~/Documents/r4ds/data-analysis - RStudio

Go to file/function | Addins | data-analysis

mpg-plot.R

Source on Save | Run | Source

```
1  library(ggplot2)
2
3  ggplot(mpg, aes(x = displ, y = hwy)) +
4    geom_point(aes(colour = class))
5
```

Editor

1:1 (Top Level) R Script

Console ~/Documents/r4ds/data-analysis/

```
> library(ggplot2)
> ggplot(mpg, aes(x = displ, y = hwy)) +
+   geom_point(aes(colour = class))
>
```

Console

Environment | History

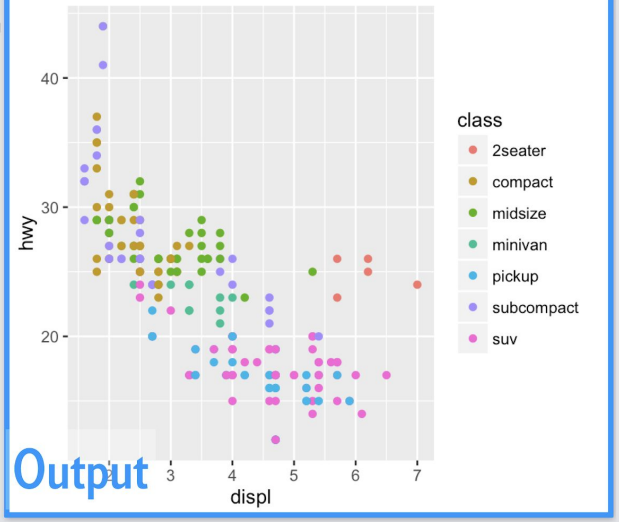
Import Dataset | List

Global Environment

Environment is empty

Files | Plots | Packages | Help | Viewer

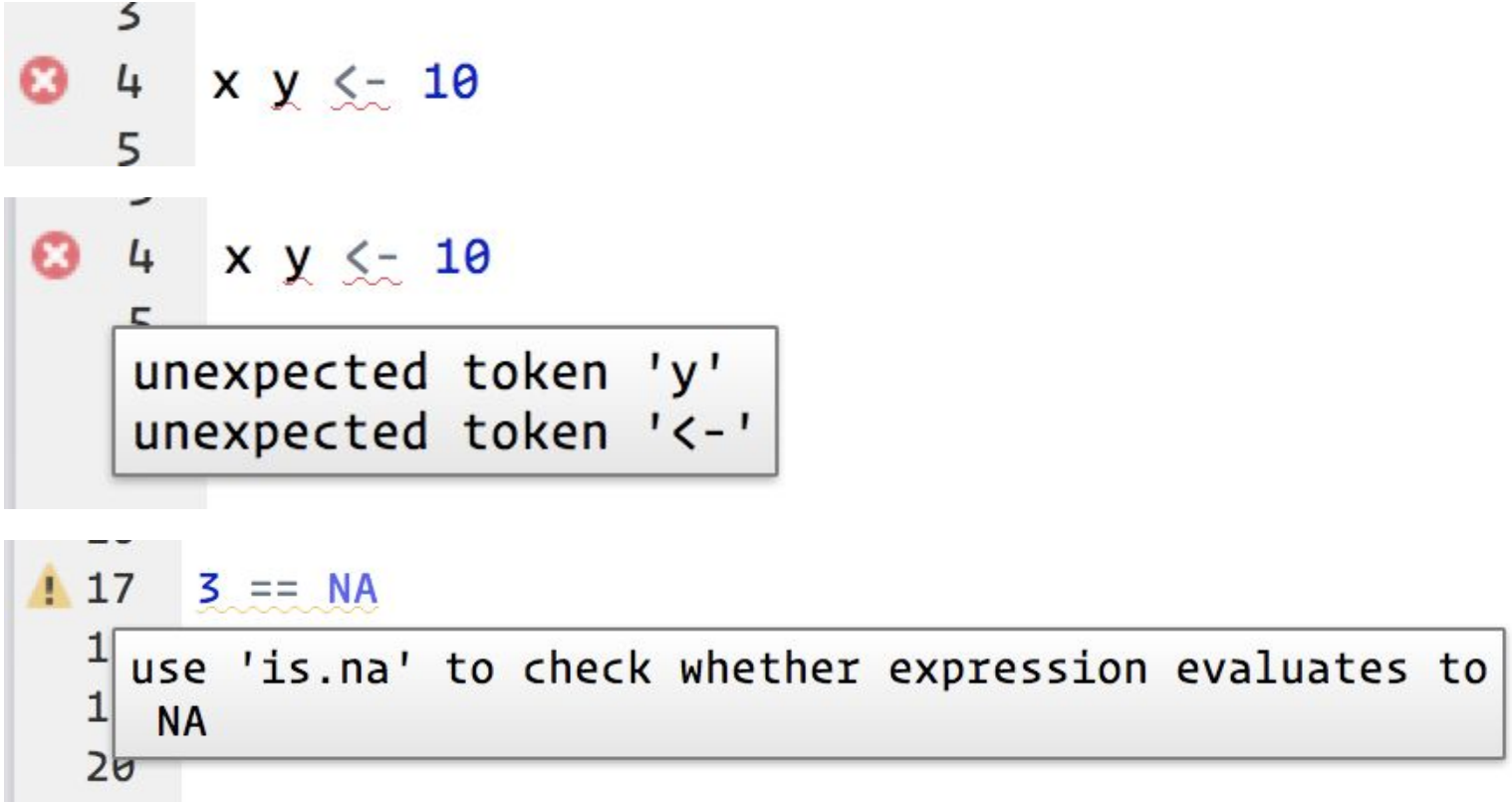
Zoom | Export



Output

RStudio

The world's favorite spell checker



RStudio

Everyone makes mistakes!

Errors are ***okay,*** it happens to everyone!

```
my_variable <- x * 3.14

my_variab1e
```

```
## Warning: # Error: object 'my_variab1e' not found
```

```
my_variable
```

```
## [1] 31.4
```

When you encounter an error:

- Pause
- Breathe
- Read the error message
- Check the help documentation
- Panic a little bit
- Google it
- RStudio Community

RStudio®

R Basics

Order of operations

- The order of operations still apply! Remember **PEMDAS.**
- () : parentheses [P]
- ^ : exponentiation [E]
- / : division [D]
- + : addition [A]
- - : subtraction [S]

$$y = mx + b$$

Creating variables

```
x <- 3 * 4
x
```

```
## [1] 12
```

```
y <- 5
y * x
```

```
## [1] 60
```

Functions

- Functions make difficult things easy
- Special kind of R object
- Recognize them by the parentheses: `function_name()`
- R is extended by packages
- Packages contain functions
- “Out of the box R” is known as base R

`sum()` function basics

- `sum()` takes inputs and calculates their sum
- Inputs are called arguments
- Recognize arguments by commas
- `function_name(argument_1, argument_2)`

```
sum(10, 3, 2)
```

```
## [1] 15
```

A simple example

```
seq(from, to, by,
    length.out, along.with)
```

```
seq(from = 10, to = 100, by = 10)
```

```
##  [1]  10  20  30  40  50  60  70  80  90 100
```

```
seq(10, 100, 10)
```

```
##  [1]  10  20  30  40  50  60  70  80  90 100
```

```
result_out <- seq(10, 100, length.out = 5)

result_out
```

```
## [1]  10.0  32.5  55.0  77.5 100.0
```

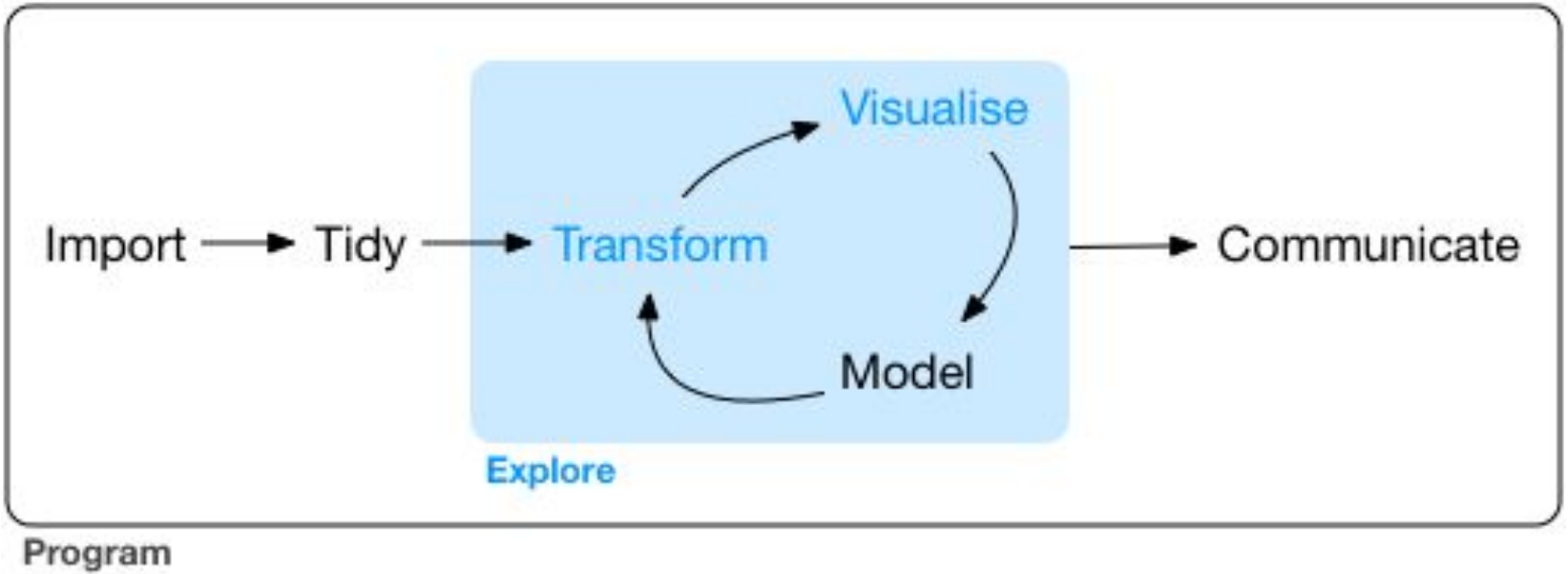
Temp check


tidyverse

"The tidyverse is an **opinionated** collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures"

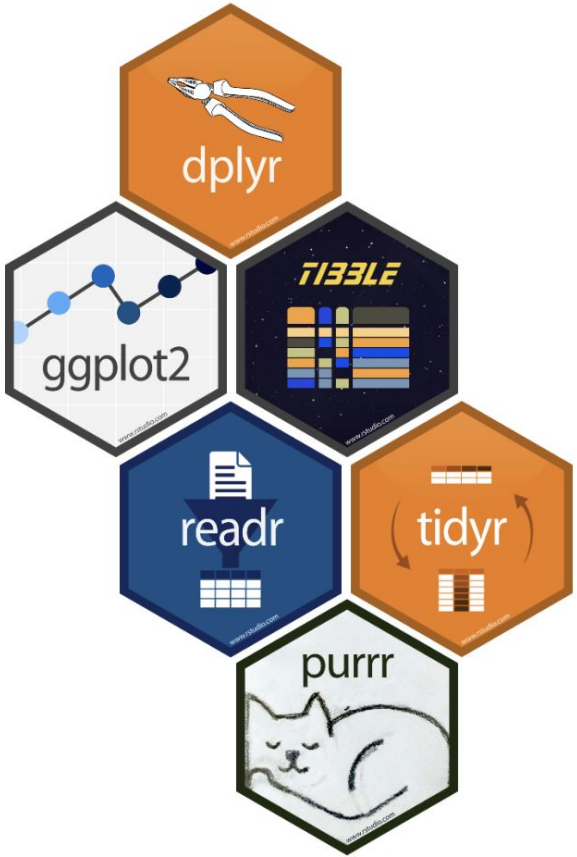
-

Data Science workflow



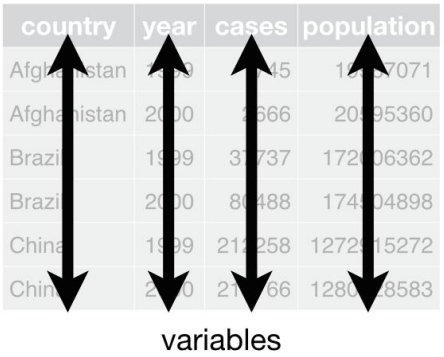
Core Tidyverse Packages

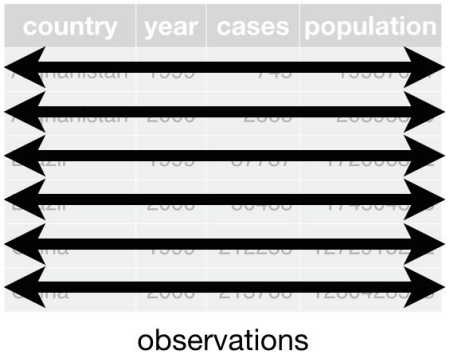
- ggplot2
- dplyr
- tidyr
- readr
- purrr
- tibble
- stringr
- forcats

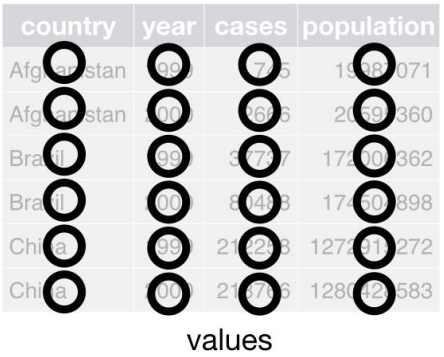


Core principles

- Built around data (the tibble)
- Tidy data, specifically
- Built for humans







RStudio

Untidy Data

```
untidy_df
```

```
## # A tibble: 5 x 7
##   age_group male_2016 female_2016 male_2017 female_2017 male_2018
##   <chr>          <dbl>       <dbl>     <dbl>       <dbl>     <dbl>
## 1 < 18           22000       20000     22000       20000     22000
## 2 18-30          36000       35000     36000       35000     36000
## 3 31-50          50000       40000     50000       40000     50000
## 4 51-60          62000       60000     62000       60000     62000
## 5 > 60           75000       72000     75000       72000     75000
## # … with 1 more variable: female_2018 <dbl>
```

Tidy data

```
tidy_df
```

```
## # A tibble: 30 x 4
##    age_group gender year  income
##    <chr>     <chr>  <chr>  <dbl>
##  1 < 18      male   2016   22000
##  2 18-30     male   2016   36000
##  3 31-50     male   2016   50000
##  4 51-60     male   2016   62000
##  5 > 60      male   2016   75000
##  6 < 18      female 2016   20000
##  7 18-30     female 2016   35000
##  8 31-50     female 2016   40000
##  9 51-60     female 2016   60000
## 10 > 60      female 2016   72000
## # … with 20 more rows
```

RStudio

Importing Data with `readr`

RStudio®

Data Sources

- Databases
- Application programming interfaces (APIs)
- Text files
  - Excel files
  - csv files
  - json

Flat Text Files: csv

- Store rectangular data
- Comma separated values!
- Each column is separated by a comma
- Each row is a new line

```
column_a, column_b, column_c,
10, "these are words", .432,
1, "and more words", 1.11
```

readr::read_csv()

- Functions do things for us!
- `read_csv()` takes a filepath and turns the csv into a tibble
- Example:
- `budget <- read_csv("data/2020-budget.csv")`

```
> budget_clean
# A tibble: 1,109 x 5
```

	portfolio	project_name	estimated_cost	expenditure_june…	budget_2019_20
	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	Aboriginal and Torres Strait …	Other property, plant and eq…	NA	NA	100
2	Aboriginal and Torres Strait …	Cape York splash parks	4000	3880	120
3	Aboriginal and Torres Strait …	Indigenous land and infrastr…	88756	88356	400
4	Aboriginal and Torres Strait …	Kickstart Mossman Gorge infr…	4818	4453	365
5	Aboriginal and Torres Strait …	Kowanyama Men's Shed and Wom…	1249	929	320
6	Aboriginal and Torres Strait …	Palm Island Splash Park	3000	NA	2500
7	Aboriginal and Torres Strait …	Three Rivers Community Centr…	3267	2667	600
8	Aboriginal and Torres Strait …	Thursday Island Splash Park	3000	NA	2500
9	Aboriginal and Torres Strait …	Wathaniin on-country trainin…	500	163	337
10	Agriculture and Fisheries	Computer equipment	NA	NA	4965

```
# … with 1,099 more rows
```

udio

File paths!

- This will get you.
- It gets me allllll the time
- Make sure you work in an R project!
- Paths start from where your project is
- “data/2020-budget.csv”

RStudio

Other data formats

readr

- `read_csv()`, `read_tsv()`, `read_delim()`

arrow

- `read_arrow()`, `read_feather()`, `read_parquet()`, `read_*_arrow()`

haven

- `read_sas()`, `read_spss()`, `read_stata()`, `read_dta()`

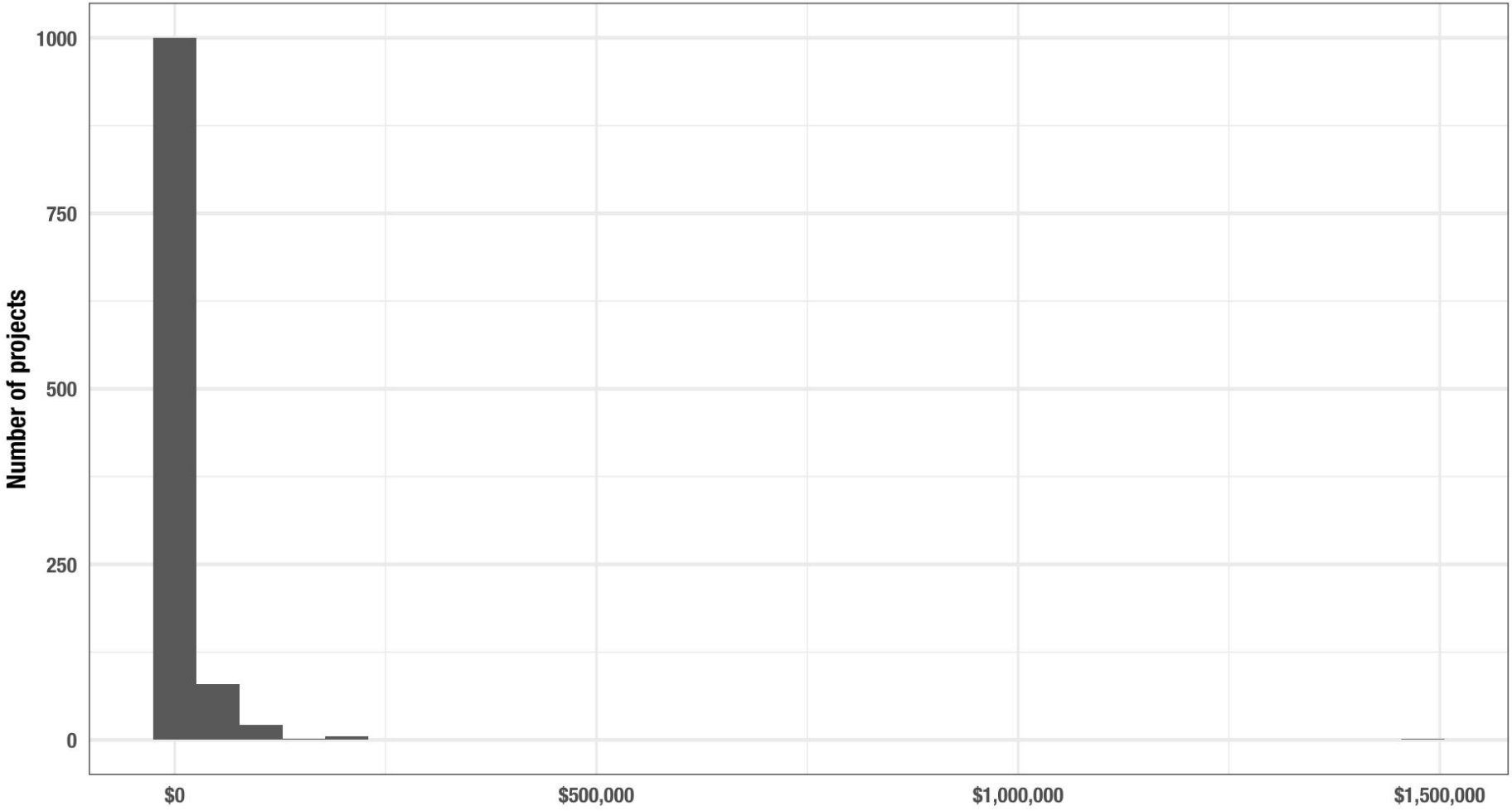
readxl

- `read_xls()`, `read_xlsx()`, `read_excel()`

Visualization & your new friend `ggplot2`

RStudio®

Distribution of 2020 Allocated Budget



A layered grammar of graphics

- A ggplot object is the composed of layers:
- Data
- Aesthetics:
    - Which columns are associated with what graphic element
- Geometry:
    - How to display mapped aesthetics

Budget Example

```
ggplot(data = budget, aes(budget_2019_20)) +
  geom_histogram() +
  labs(title = "Distribution of 2020 Allocated Budget",
       x = "", y = "Number of projects") +
  scale_x_continuous(labels = scales::dollar)
```

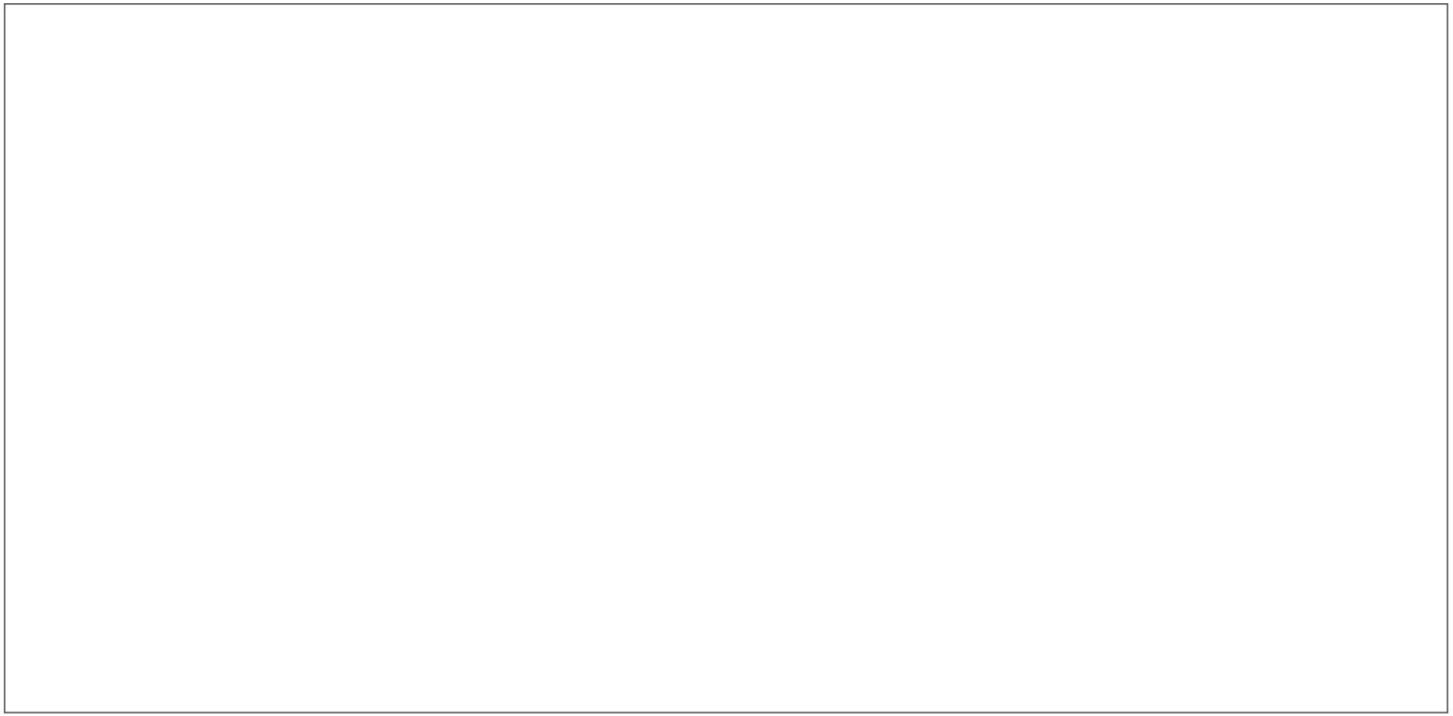
From the ground up

- Specify the data:
- `ggplot(data = budget)`

Aesthetic Mappings

- Specify column mappings
- Use `aes()`
- Arguments: `x`, `y`, `color`, and `fill`, among others
- `ggplot(data = budget, aes(x = budget_2019_20))`
- What does our graphic look like now?

Still nothing...



RStudio®

Adding geometry

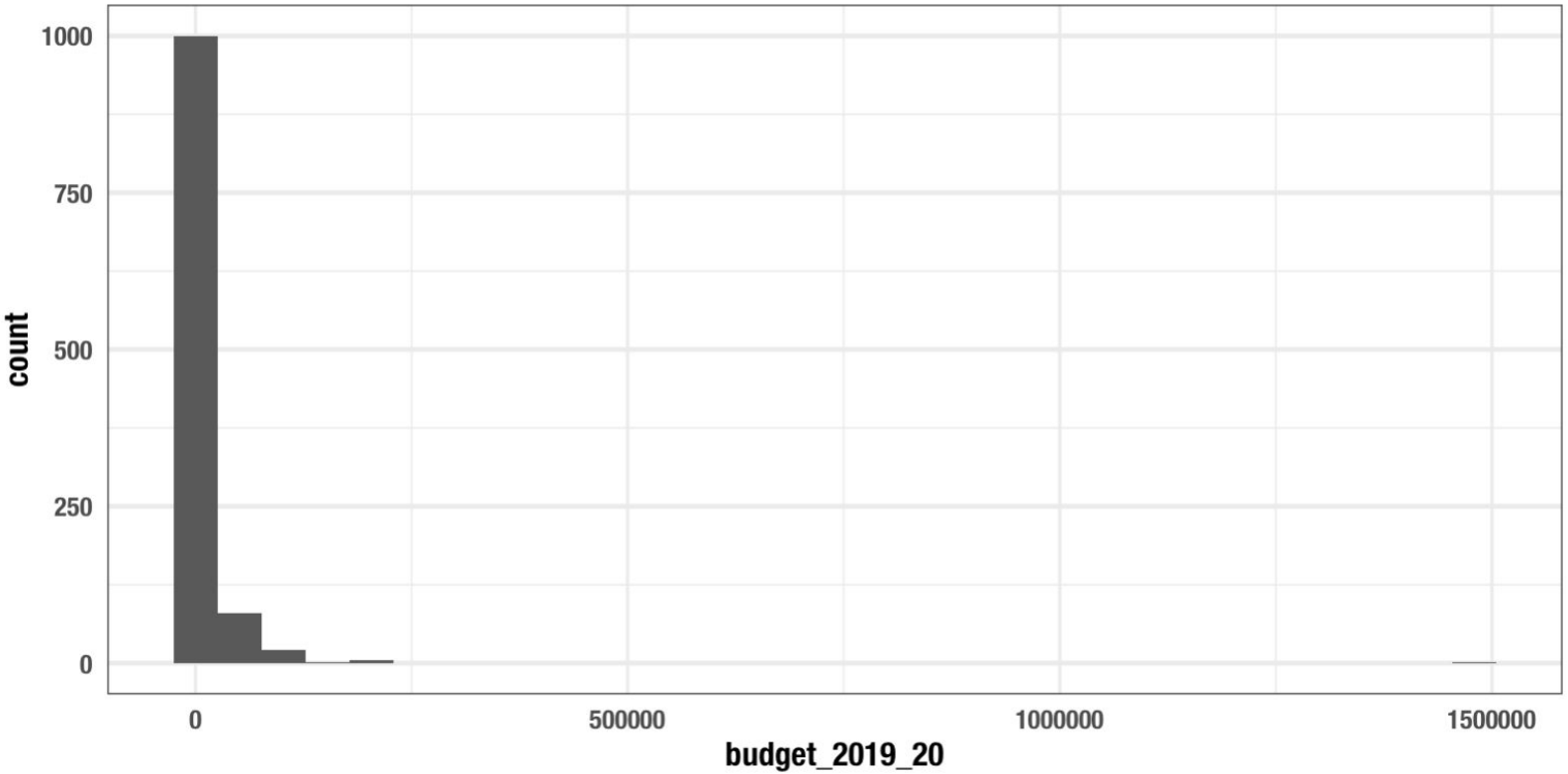
- We **add** layers to `ggplot()` with a **+**
- Geometry layers start with `geom_*()`
- TONS of geoms each with own requirements
- `geom_histogram()` creates a histogram from the mapped **x**

```
ggplot(data = budget, aes(x = budget_2019_20)) +

  geom_histogram()
```

RStudio

Getting there!



RStudio

Additional layers

```
ggplot(data = budget, aes(x = budget_2019_20)) +
  geom_histogram() +
  labs(
    title = "Distribution of 2020 Allocated Budget",
    x = "",
    y = "Number of projects"
  ) +
  scale_x_continuous(labels = scales::dollar)
```

Review

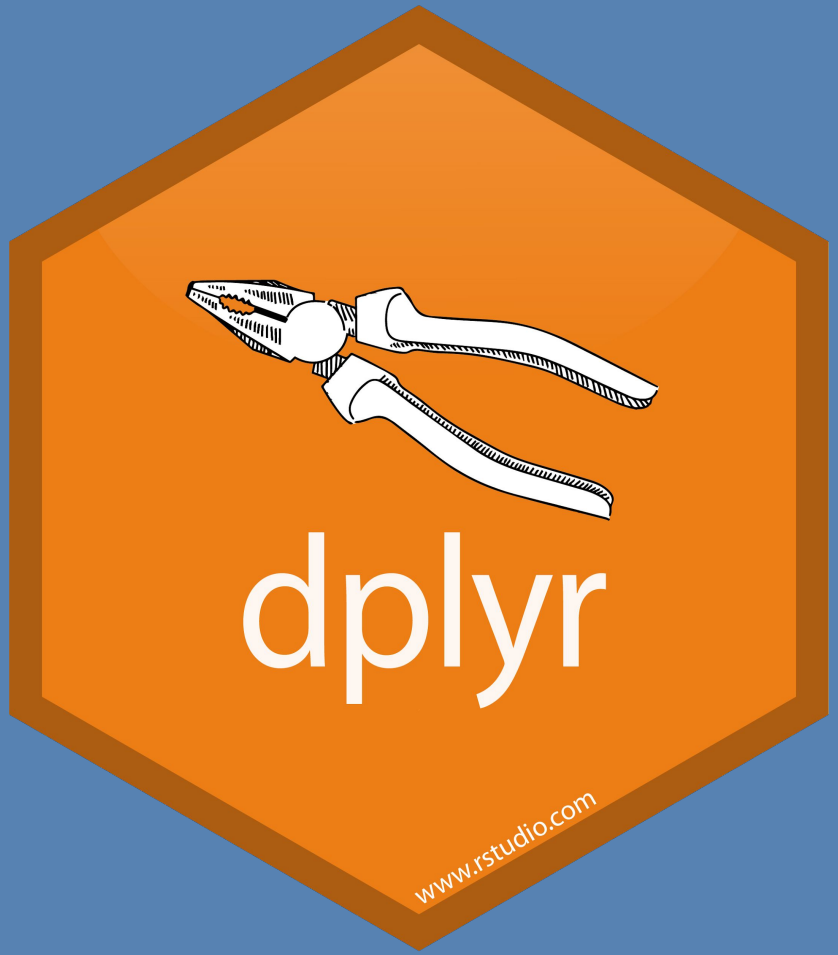
- `ggplot()` builds the base of the graphic
- Specify the aesthetic mappings with `aes()`
- Add geometry layers with `geom_*()`

```
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) +

 <GEOM_FUNCTION>()
```

Some possible geoms

- `geom_point()` - Points
- `geom_dotplot()` - Dot plot
- `geom_hline()` - Horizontal reference line
- `geom_vline()` - Vertical reference line
- `geom_boxplot()` - A box and whisker plot
- `geom_density()` - Smoothed density estimates
- `geom_errorbarh()` - Horizontal error bars
- `geom_hex()` - Hexagonal heatmap of 2d bin counts
- `geom_jitter()` - Jittered points
- `geom_linerange()` - Vertical interval line
- `geom_pointrange()` - Vertical point line
- `geom_line()` - Connect observations line
- `geom_step()` - Connect observations via step lines
- `geom_polygon()` - Polygons
- `geom_segment()` - Line segment
- `geom_ribbon()` - Ribbon plot
- `geom_area()` - Area plot
- `geom_rug()` - Rug plots in the margins
- `geom_smooth()` - Smoothed conditional means
- `geom_label()` - Label points with text
- `geom_text()` - Add text
- `geom_violin()` - Violin plot
- `geom_sf()` - Visual sf objects
- `geom_map()` - Plot map
- `geom_qq_line()` - A quantile-quantile plot
- `geom_histogram()` - Histogram plot



RStudio

6 Main verbs

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

Simple use

- `pull()`
- `n()`/`count()`
- `glimpse()`

Advanced iterations

- `summarize_at()`
- `mutate_at()`

More info

- `dplyr.tidyverse.org`
- `R for Data Science`

RStudio

Getting a feel for the data

```
glimpse(budget)
```

```
## Observations: 5
## Variables: 5
## $ portfolio           <chr> "Aboriginal and Torres Strait Islander
## $ project_name        <chr> "Other property, plant and equipment",
## $ estimated_cost      <dbl> NA, 4000, 88756, 4818, 1249
## $ expenditure_june_19 <dbl> NA, 3880, 88356, 4453, 929
## $ budget_2019_20      <dbl> 100, 120, 400, 365, 320
```

`dplyr::select()`ing columns

- select() selects columns from a tibble and returns another tibble
- select(.data, …)
- … aka “dots” allows us to pass as many arguments as we want
- Example:

```
select(budget, project_name)
```

```
## # A tibble: 1,109 x 1
##    project_name
##    <chr>
##  1 Other property, plant and equipment
##  2 Cape York splash parks
##  3 Indigenous land and infrastructure programs
##  4 Kickstart Mossman Gorge infrastructure
##  5 Kowanyama Men's Shed and Women's Meeting Place
##  6 Palm Island Splash Park
##  7 Three Rivers Community Centre redevelopment
##  8 Thursday Island Splash Park
##  9 Wathaniin on-country training accommodation
## 10 Computer equipment
## # … with 1,099 more rows
```

de-`select()`ing columns

- Deselect with the minus sign

```
select(budget, -portfolio, -project_name)
```

```
## # A tibble: 1,109 x 3
##    estimated_cost expenditure_june_19 budget_2019_20
##             <dbl>               <dbl>          <dbl>
##  1             NA                  NA            100
##  2           4000                3880            120
##  3          88756               88356            400
##  4           4818                4453            365
##  5           1249                 929            320
##  6           3000                  NA           2500
##  7           3267                2667            600
##  8           3000                  NA           2500
##  9            500                 163            337
## 10             NA                  NA           4965
## # … with 1,099 more rows
```

dplyr::filter()

- Pair down your data
- Takes data and logical statement
- `filter(budget, portfolio == "Queensland Treasury")`

portfolio <chr>	project_name <chr>	estimated_cost <dbl>
Queensland Treasury	Office of State Revenue Transformation Program	17739
Queensland Treasury	Queensland First Home Owners' Grant	*NA*
Queensland Treasury	Cross River Rail	6725804
Queensland Treasury	Third party returnable works	162196

4 rows | 1-3 of 5 columns

RStudio

Piping it together!

Forward pipe operator %>%

- The first argument is almost always the data
- The pipe lets us chain functions together
- Makes code readable
- Easier to debug
- In technical speak it makes the output of the left hand side available as the first argument in the RHS function

%>% alternatives

```
did_something <- do_something(data)

did_another_thing <- do_another_thing(did_something)

final_thing <- do_last_thing(did_another_thing)
```

```
final_thing <- do_last_thing(
  do_another_thing(
    do_something(
      data
      )
    )
  )
```

```
final_thing <- data %>%
      do_something() %>%
      do_another_thing() %>%
      do_last_thing()
```

Putting it together

```
budget %>%
  filter(portfolio == "Queensland Treasury") %>%
  select(project_name, budget_2019_20)
```

```
## # A tibble: 4 x 2
##   project_name                                      budget_2019_20
##   <chr>                                                      <dbl>
## 1 Office of State Revenue Transformation Program               561
## 2 Queensland First Home Owners' Grant                        109839
## 3 Cross River Rail                                          1479707
## 4 Third party returnable works                               49658
```

dplyr::mutate()

- Creates new columns from expressions
- Each new column is it's own argument, so we give it a name
- `mutate(data, new_col = col_1 / sum(col_1))`

```
mutate(budget, prop_cost = budget_2019_20 / estimated_cost) %>%
  select(prop_cost)
```

```
## # A tibble: 1,109 x 1
##    prop_cost
##        <dbl>
##  1  NA
##  2   0.03
##  3   0.00451
##  4   0.0758
##  5   0.256
##  6   0.833
##  7   0.184
##  8   0.833
##  9   0.674
## 10  NA
## # … with 1,099 more rows
```

RStudio

How are we feeling?
What makes no sense at all right now?

Aggregating

aka grouping and summarizing

RStudio®

dplyr::count()

```
count(budget, portfolio)
```

```
## # A tibble: 19 x 2
##    portfolio        n
##    <chr>        <int>
##  1 Aborigina...     9
##  2 Agricultu...    15
##  3 Child Saf...     5
##  4 Communiti...    13
##  5 Education      198
##  6 Electoral...     2
##  7 Employmen...    16
##  8 Environme...    39
##  9 Housing a...    90
## 10 Innovatio...    13
## 11 Justice a...    23
## 12 Legislati...     6
## 13 Local Gov...    17
## 14 Natural R...   192
## 15 Public Sa...    58
## 16 Queenslan...   133
## 17 State Dev...    44
## 18 Transport...   231
## 19 Youth Jus...     5
```

`dplyr::count()`

- What was happening there?
- Grouping the tibble by unique portfolio values
- Counting how many times that occurred

dplyr::group_by()

- Explicitly create groups within our tibble
- We can now perform grouped operations

```
budget %>%
  group_by(portfolio)
```

```
## # A tibble: 1,109 x 5
## # Groups:   portfolio [19]
##    portfolio    project_name
##    <chr>        <chr>
##  1 Aborigina... Other property, plant and equipment
##  2 Aborigina... Cape York splash parks
##  3 Aborigina... Indigenous land and infrastructure programs
##  4 Aborigina... Kickstart Mossman Gorge infrastructure
##  5 Aborigina... Kowanyama Men's Shed and Women's Meeting Place
##  6 Aborigina... Palm Island Splash Park
##  7 Aborigina... Three Rivers Community Centre redevelopment
##  8 Aborigina... Thursday Island Splash Park
##  9 Aborigina... Wathaniin on-country training accommodation
## 10 Agricultu... Computer equipment
```

`dplyr::summarise()`

- Creates aggregate measures
- `mutate()` but for groups
- Create summary values
- Only one value per group
- Rule of thumb:
    - If there is more than one value in the output, you probably want `mutate()`

dplyr::summarise()

```
budget %>%
  group_by(portfolio) %>%
  summarise(n = n(),
            avg_budget = mean(budget_2019_20),
            max_budget = max(budget_2019_20))
```

```
## # A tibble: 19 x 4
##    portfolio        n avg_budget max_budget
##    <chr>        <int>      <dbl>      <dbl>
##  1 Aborigina...     9       805.       2500
##  2 Agricultu...    15      1529.       4965
##  3 Child Saf...     5      4952.      14782
##  4 Communiti...    13      1747.       4100
##  5 Education      198      6704.     100098
##  6 Electoral...     2       962.       1846
##  7 Employmen...    16      8059.      25000
##  8 Environme...    39      3067.      10000
##  9 Housing a...    90      9088.     102753
## 10 Innovatio...    13      7185       23055
## 11 Justice a...    23      3371.      14044
## 12 Legislati...     6      1139.       3412
## 13 Local Gov...    17      9550.      50000
## 14 Natural R...   192     13854.     216783
## 15 Public Sa...    58      4545.      45712
## 16 Queenslan...   133     19363.    1479707
## 17 State Dev...    44     10391.     210548
## 18 Transport...   231     17730.     213285
## 19 Youth Jus...     5     10383.      23858
```

Enough for today!

RStudio®

You learned **A LOT!**

RStudio®

You learned:

- What R is and why it's special
- How to navigate RStudio
- The basics of R
- The grammar of graphics
- How to select and filter data, create new columns
- Chain functions together
- Aggregate data

RStudio

How to continue

- R for Data Science r4ds.had.co.nz
- Modern Dive moderndive.com
- RStudio Primers rstudio.cloud/learn/primers
- Create an R user group
- We can schedule another session!

RStudio

Data Transformation with dplyr : : CHEAT SHEET

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

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

&

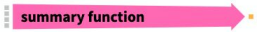
Each **observation**, or **case**, is in its own **row**

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).



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

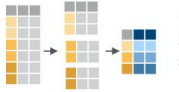
count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

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))
```

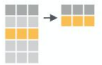
group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

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)

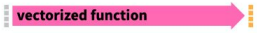
select(.data, …)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

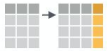
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

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

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).

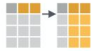




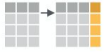
mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

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)

Cheat Sheets

https://rstudio.com/resources/cheatsheets

RStudio