Tidyverse workshop

Attribution:
Materials prepared by Sander Wuyts and Stijn Wittouck **@s_wittouck @s_wuyts**
Updated by Paula Andrea Martinez **@orchid00**

Materials: tiny.cc/tidyverse_intro

Overview

1. Introduction
2. Introduction to visualisation with ggplot2
3. Introduction to data manipulation with dplyr
4. Introduction to tidy data with tidyr

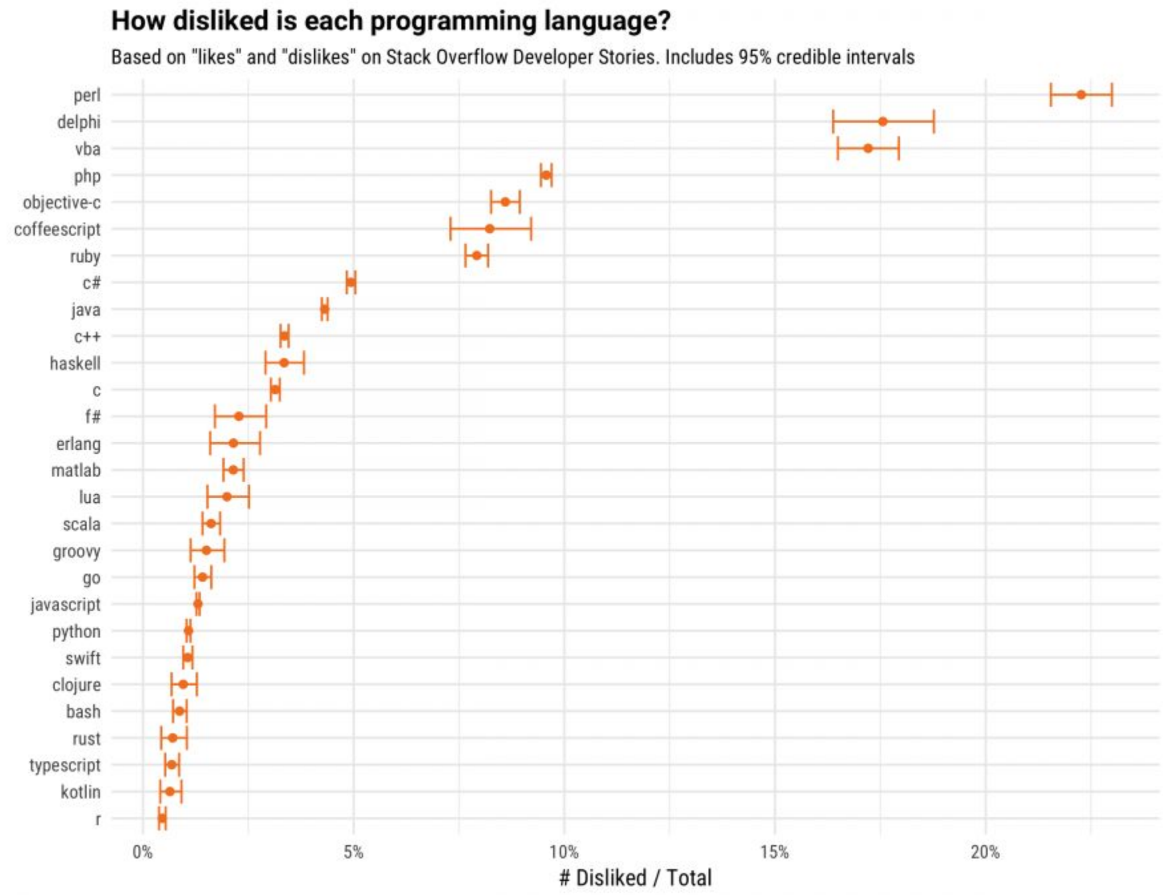
1. Introduction

R

- Open source programming language
- Very approachable and friendly community
- Mostly known as software environment for statistical computing
- Capability is expandable by importing *packages*
  - 14,000+ packages available through CRAN, Bioconductor, Github, …
- Rising popularity in data sciences

Is it a good time to learn R?

Also have a look at the impressive growth or R

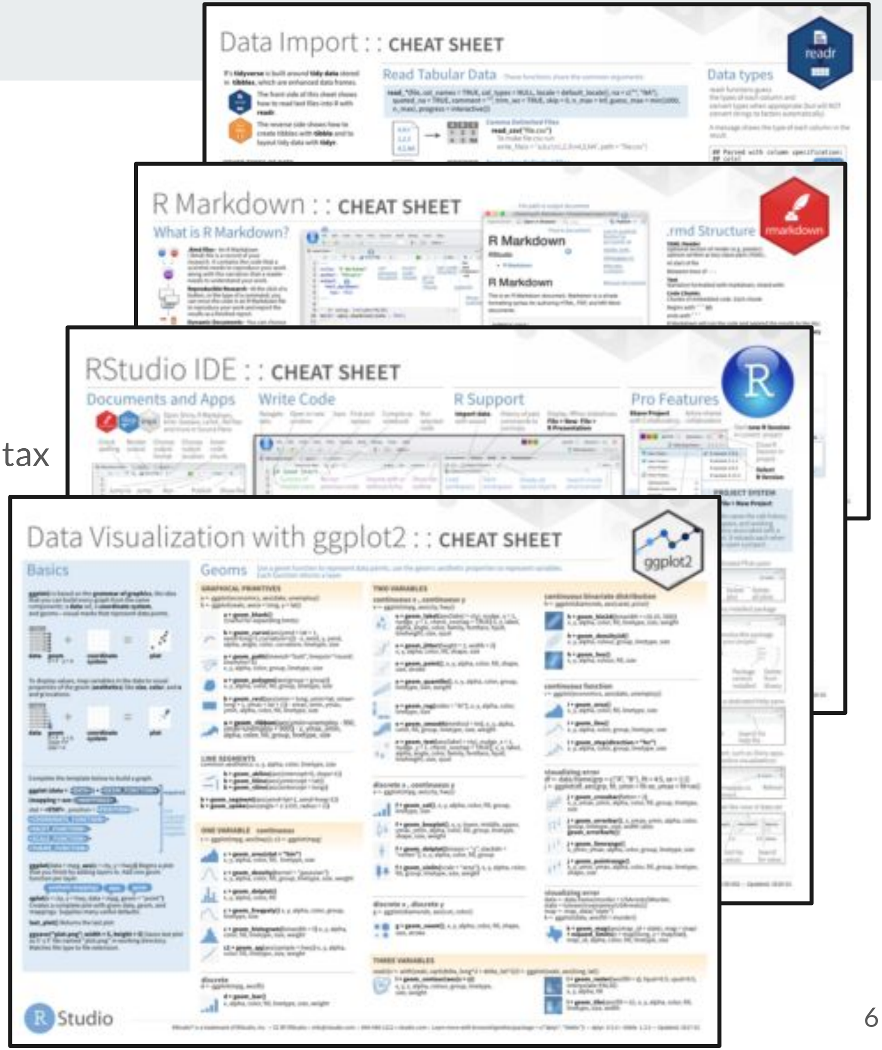


https://stackoverflow.blog/2017/10/31/disliked-programming-languages/
https://stackoverflow.blog/2017/10/10/impressive-growth-r/
by David Robinson Data scientist at stackoverflow now Chief data scientist at Datacamp

RStudio cheat sheets

Very good reference if you can’t remember the right syntax

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



Data Import : : CHEAT SHEET

Read Tabular Data

Data types

R Markdown : : CHEAT SHEET

What is R Markdown?

.rmd Structure

RStudio IDE : : CHEAT SHEET

Documents and Apps

Write Code

R Support

Pro Features

Data Visualization with ggplot2 : : CHEAT SHEET

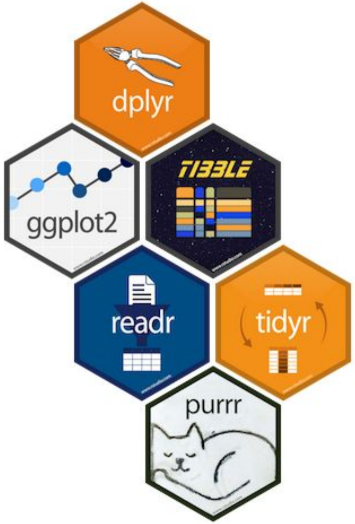
Basics

Geoms

The tidyverse

R packages for data science

- Set of tools to transform and visualise data
- All packages share an underlying philosophy
- Most of them are created by Hadley Wickham
- *packages*:
  - ggplot2
  - dplyr
  - tidyr
  - readr
  - ...



https://www.tidyverse.org/
http://r4ds.had.co.nz/

Datasets

- The plants and animals datasets were gathered from NCBI’s Eukaryote genome data
- Other exercises datasets come from the enterotype dataset (Arumugam, M. *et al. Nature, 2011)* obtained from the R package *Phyloseq*

Our workshop project

Download the slides and the project including the datasets from:
https://github.com/orchid00/tidyverse_intro

2. Introduction to ggplot2

Grammar of Graphics

“An abstraction which makes thinking, reasoning and communicating graphics easier”

- First described by Leland Wilkinson (**G**rammar of **G**raphics, 1999)
- Implemented into **gg**plot2 (Hadley Wickham, 2005)
- Divide your graphics in different layers based on the grammar of graphics
  - Add building blocks to customise your visualisation

A graphing template

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

1. **DATA**
   Your dataset of interest to use in the graph (tip: data.frame)

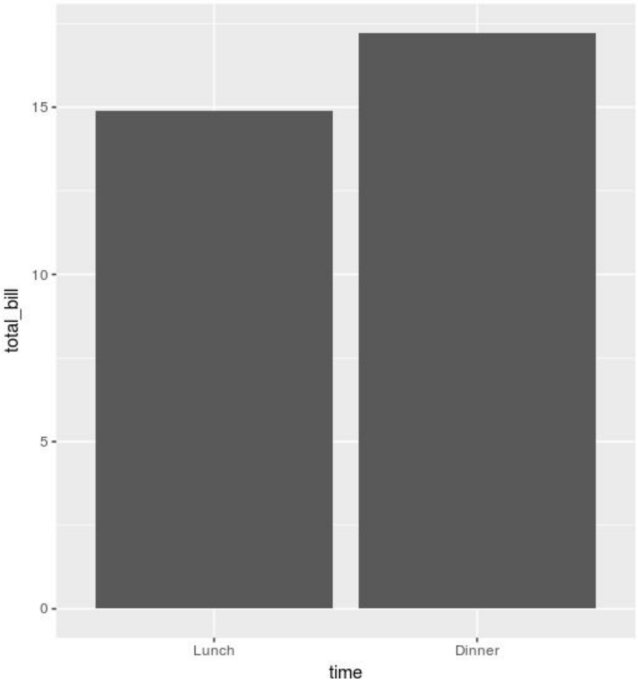
2. **GEOM_FUNCTION**
   Each adds a different type of layer to a plot (*e.g.* point, line, bar, …)

3. **MAPPINGS**
   How **variables** in the **data** are mapped to visual properties (**aesthetics**) that can be used to communicate information (*e.g.* x and y axis)

Build your own ggplot graph

```
                   1
ggplot(data = my_data) +
     geom_bar(mapping = aes(x = time,
         2                       y = total_bill),  3
                  stat = "identity")
```

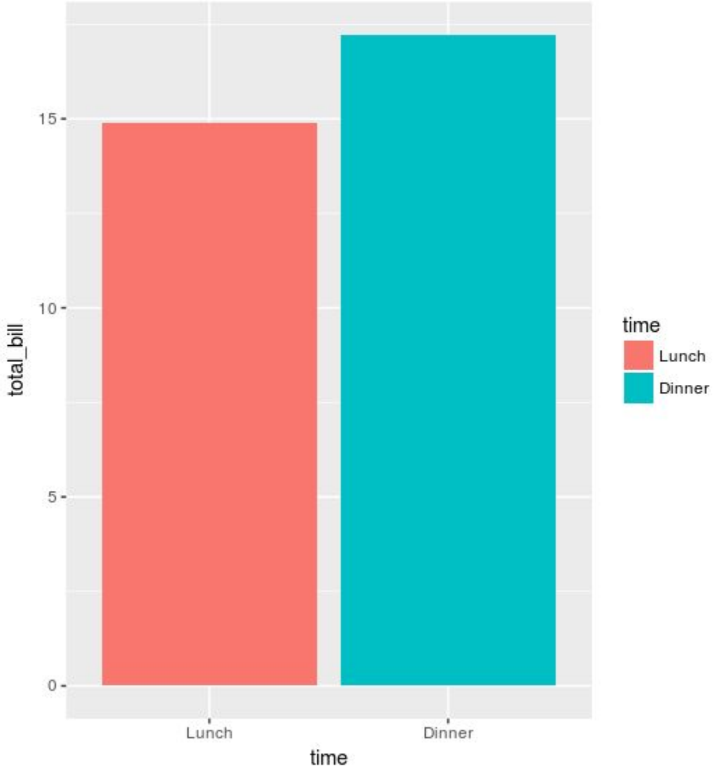
1. **data**
2. **geom_function**
3. **aesthetic mapping**



Build your own ggplot graph

```
                 1
ggplot(data = my_data) +
     geom_bar(mapping = aes(x = time,
           2                y = total_bill,  3
                        fill = time),
                stat = "identity")
```

1. **data**
2. **geom_function**
3. **aesthetic mapping**



Additional layers

4.	stat	Statistical transformations on the data
5.	Position	Resolves overlapping geoms by adjusting
6.	scale_	Tweaks details like limits, colours, axis labels or legend keys
7.	facet_	Displays different subsets of the dataset
8.	coord_	Changes the coordinates shown for *x* and *y* aesthetic s
9.	theme_	Controls the display of all non-data elements of the plot

1. data

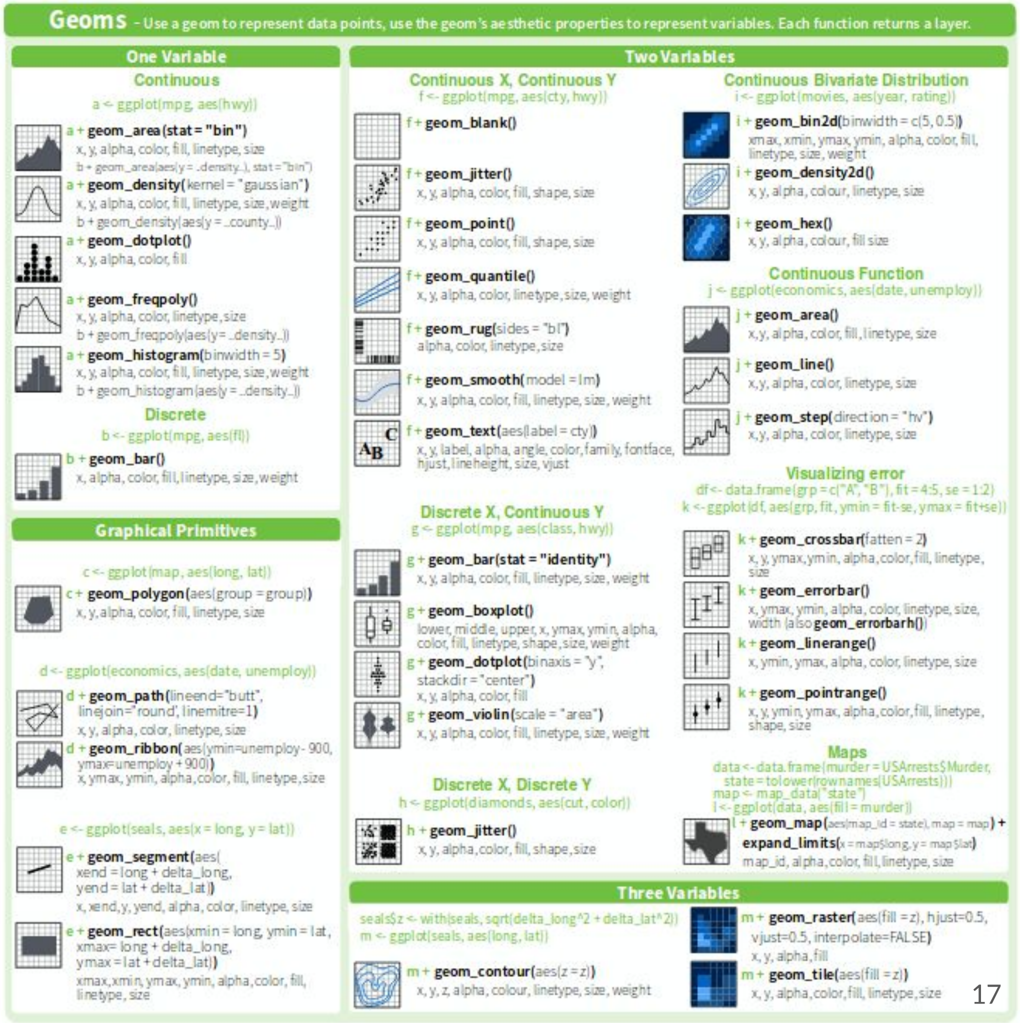
We will read in data using one of these functions from the R package **readr**:

```
read_tsv()

read_csv()

...
```

Returns a **tibble** which is an *updated-improved* version of **data.frame**

2. Geoms

```
geom_point()

geom_bar()

geom_boxplot()

geom_violin()

geom_line()

...
```



Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + geom_dotplot()
x, y, alpha, color, fill

a + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y= ..density..))

a + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + geom_polygon(aes(group = group)**)**
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + geom_path(lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, linetype, size

d + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)**)**
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

e + geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat)**)**
x, xend, y, yend, alpha, color, linetype, size

e + geom_rect(aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + geom_blank()

f + geom_jitter()
x, y, alpha, color, fill, shape, size

f + geom_point()
x, y, alpha, color, fill, shape, size

f + geom_quantile()
x, y, alpha, color, linetype, size, weight

f + geom_rug(sides = "bl"**)**
alpha, color, linetype, size

f + geom_smooth(model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

f + geom_text(aes(label = cty)**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + geom_jitter()
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + geom_bin2d(binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + geom_density2d()
x, y, alpha, colour, linetype, size

i + geom_hex()
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + geom_area()
x, y, alpha, color, fill, linetype, size

j + geom_line()
x, y, alpha, color, linetype, size

j + geom_step(direction = "hv"**)**
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + geom_map(aes(map_id = state), map = map**) +**
expand_limits(x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

m + geom_contour(aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight

m + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

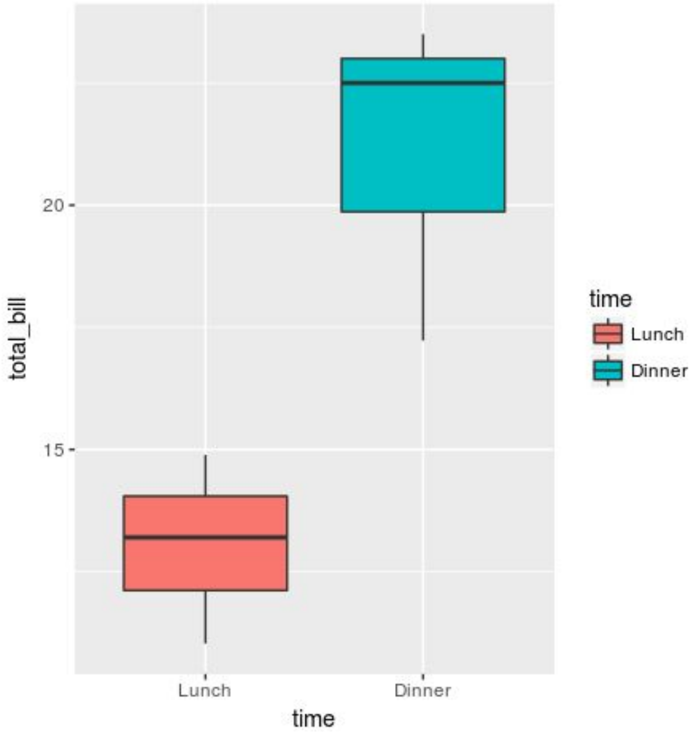
m + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size

3. Aesthetics

You can change the look and feel of your plots using **aes**thetics.

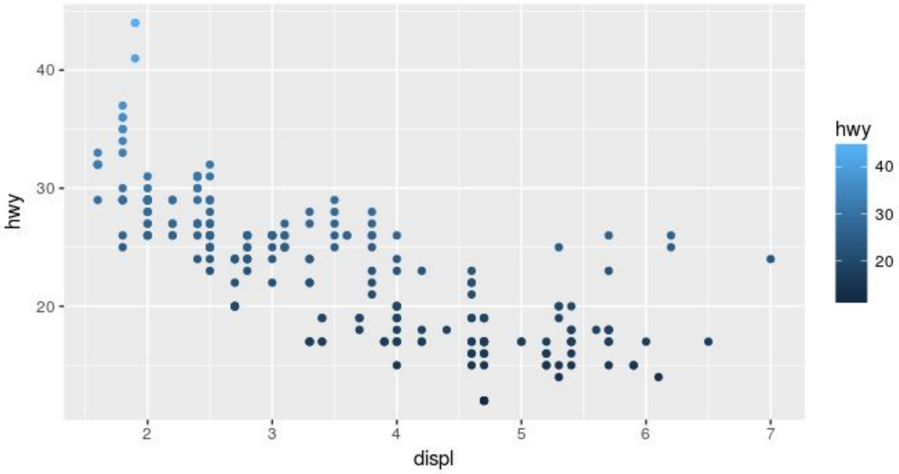
```
ggplot(data = my_data) +
  geom_boxplot(mapping  = aes(x = time,
                              y = total_bill,
                          fill = time))
```

Note: mapping = aes() can be at the top ggplot or for each geom



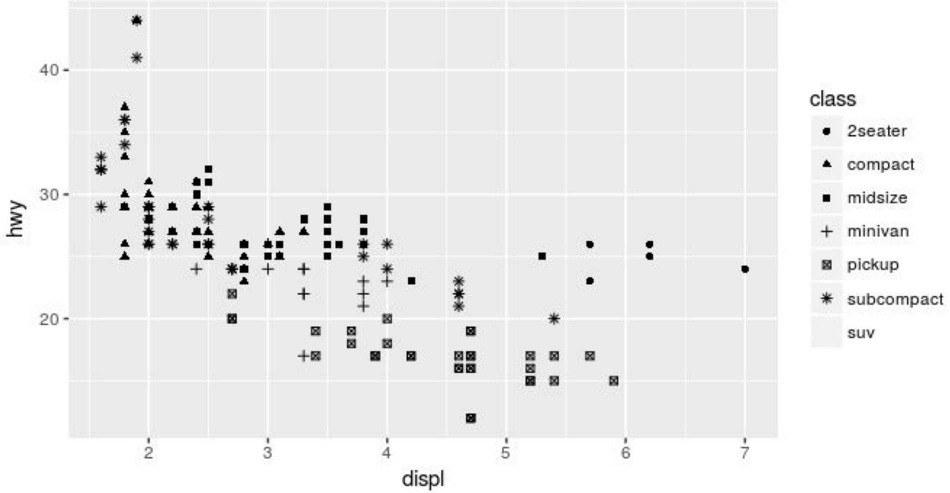
3. Aesthetics

```
ggplot(mpg) +
  geom_point(mapping = aes(x = displ,
                           y = hwy,
                           colour = hwy))
```



3. Aesthetic extras

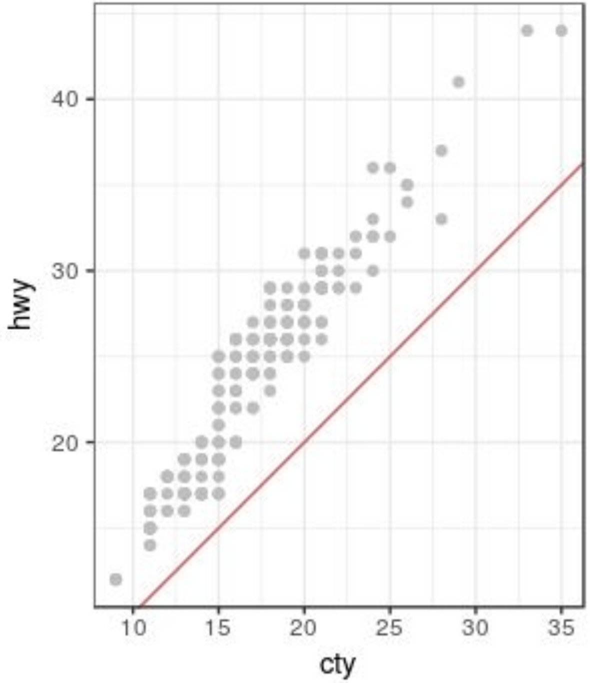
- x- and y-axis
- colour or fill
- alpha (transparency)
- size
- shape



Combining layers

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
          geom_point(colour = "grey") +
          geom_abline(colour = "indianred") +
          coord_fixed() +
          theme_bw()
```

If you want to add multiple geoms you can share the aesthetics for the axis.



Demonstration

We will use the dataset plants

Exercises ggplot2

1. Read in sampledata.tsv
2. Explore the dataset

Save each of plots in the plots folder

3. Plot the amount of males and females in this study using a bar plot
4. Plot the different nationalities using a bar plot. Extra: flip the axes
5. Create a boxplot showing the age distribution for each nationality. Extra: Use the fill or colour aesthetic to make it a little bit more colourful
6. Add an extra layer to 5. with plotting points over the boxplot. Remove that layer again and explore the difference with geom_jitter()
7. Make a density plot of the age distribution coloured by gender, faceted by nationality. Extra: try one plot per row, and use a transparency of 0.5, you can also add geom_rug()

3. Introduction to data manipulation

Data manipulation

6 essential verbs:

- `select()` to select columns
- `mutate()` to create new columns
- `filter()` to filter rows
- `arrange()` to order rows
- `summarise or summarize()` for aggregate functions
- `group_by()` to create subgroups

complex tasks can be expressed as sequences of simple verbs

tibble: Simple data.frames

data structure: “**tibble**” alias “**data_frame**”

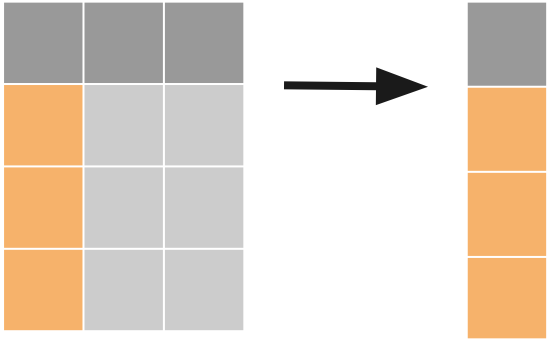
- Efficient and trimmed version of a data.frame
- Gives you glimpse() to explore a tible

Select columns: `select()`

```
tibble <- select(tibble, var_1, var_2, …)
```

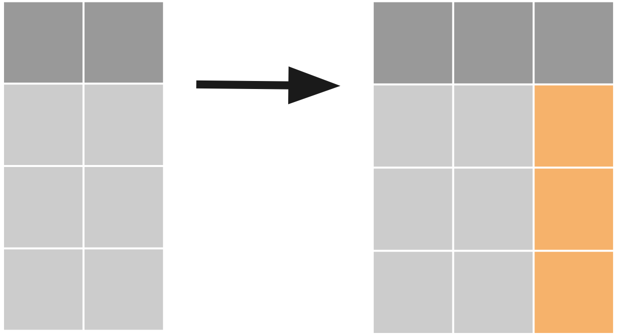
helper verbs for variable selection:

- `contains()`
- `starts_with()`
- `ends_with()`
- `` `-` ``
- `` `:` for a range ``



Make columns: `mutate()`

```
tibble <- mutate(tibble,
  new_var_1 = expression_1,
  new_var_2 = expression_2, …
)
```



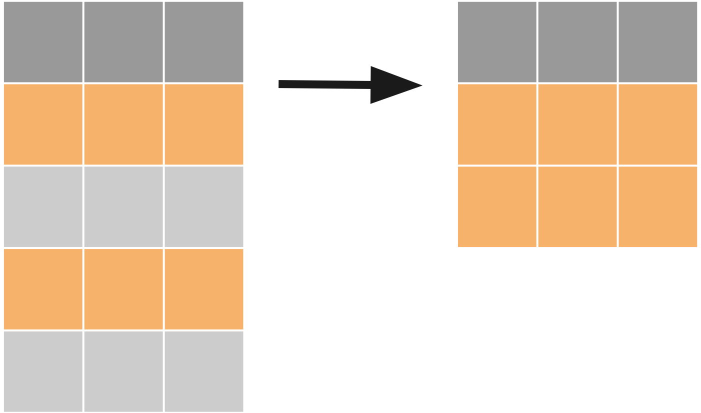
Inside the parenthesis use “=”, not “<-”
The key property is that the expression returns a vector

expressions (too many to list):

- Arithmetic operations: *e.g.* +, -, *, /, ^
- `mean()`, `log()`, `n()` for counting...

Filter rows: `filter()`

```
tibble <- filter(tibble, <expression>)
```



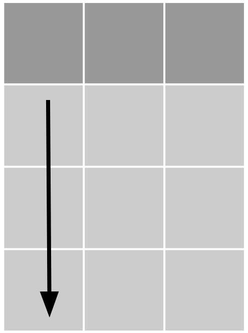
To subset observations based on their value

- Comparison operators:
  - `>, >=, <, <=, != (not equal), and == (equal)`
  - `is.na() or !is.na()`
- Logical operators:
  - `& is “and”, | is “or”, and ! is “not”`

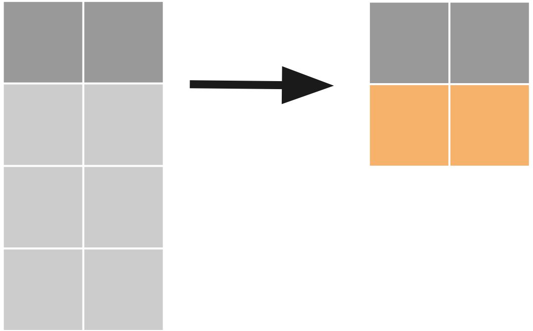
Order rows: `arrange()`

```
tibble <- arrange(tibble, var_1, desc(var_2))
```

Use **desc()** to re-order by a column in descending order



Summarise rows: `summarise()`



```
tibble <- summarise(tibble,
            aggregated_var = expression)
```

Use `summarise()` or `summarize()`

expressions will collapses data to a single row, *e.g.*:

- `mean()`
- `median()`
- `sd()`
- `sum()`
- `n()`

Grouped summaries

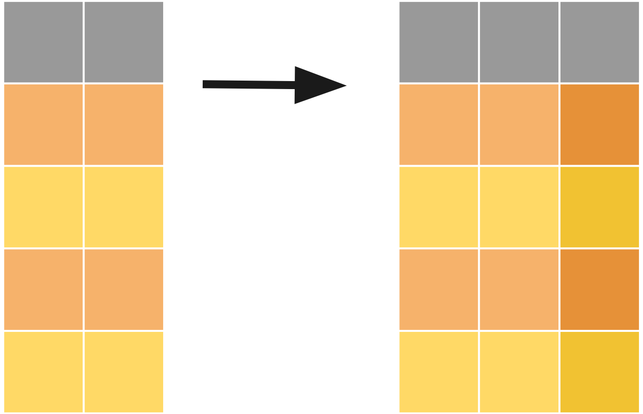
From the complete dataset to individual groups

- computations within other verbs (e.g. `mutate()`, `summarise()`) will happen per group
- verbs:
  - `group_by()` : add grouping
  - `ungroup()` : remove grouping
- you can group by multiple variables simultaneously
  - groups will be combinations of variable values

Group-wise analysis

Workflow for group-wise **mutate**:

1. `select()`
2. `group_by()`
3. `mutate()`
4. `ungroup()`

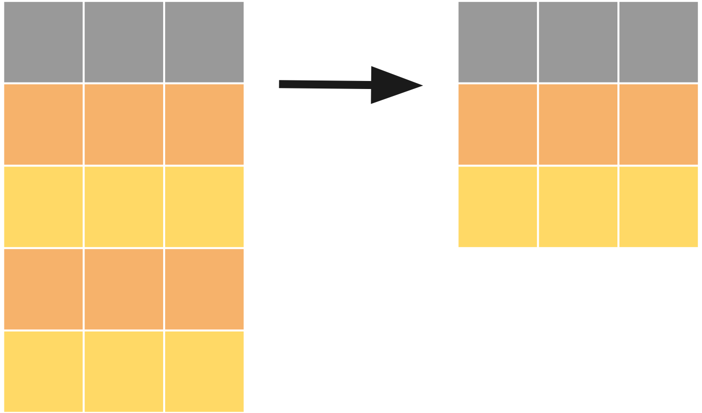


Group-wise analysis

Workflow for group-wise **summarise**:

1. select()
2. group_by()
3. summarise()

No need to ungroup after summarise



The pipe operator (%>%)

Imagine you can combine multiple verbs together! from package **magrittr**

Use the pipe **%>%**
shortcut Ctrl+Shift+m

Advantages:

- less typing
- less redundancy (easier to change object names)
- more readable code

Demonstration

We will use the dataset animals

Exercises dplyr

1. **Import** the file sampledata.tsv.
2. **Select nationality and bmi_group and filter** out all rows where the variables nationality or bmi_group are NA. Store the resulting tibble as “mysamples_filtered”. Dimensions: 1057x2
3. Start from “mysamples_filtered”. Make a tibble "mysamples_summary" with a **count of participants per combination of nationality and bmi_group.** Sort the table by nationality and inversely by count within each nationality. Dimensions: 27x3
4. Make a **bar plot** to inspect whether some nationalities have more overweight participants than others, using mysamples_summary. Extra: Flip the axis and use a palette colour called “Set3”, change the default theme. Save the plot in plots
5. Extra: use mysamples_summary and filter out the nationalities with "Europe" in the name. Hint: resulting dimensions 15 x 3

4. Introduction to tidy data

Untidy data

How would you make the following figure using **ggplot2**:

- day on the x-axis
- count on the y-axis (numbers of turnips)

name	**day_1**	**day_2**	**day_3**
eileen	10	11	11
bart	9	10	17
kim	3	15	16

Untidy data

Plotting without transformation is not possible!

Why?

- Turnip counts should be one variable, but it is spread over multiple columns
- Day should be a variable, but this information is now in the column headers

name	**day_1**	**day_2**	**day_3**
eileen	10	11	11
bart	9	10	17
kim	3	15	16

Tidy data

Getting your data into this format requires some upfront work, but that work pays off in the long term.

1. Each variable forms its own column
2. Each observation forms its own row
3. Each value must have its own cell

name	day	turnips
eileen	day_1	10
bart	day_1	9
kim	day_1	3
eileen	day_2	11
bart	day_2	10
kim	day_2	15
eileen	day_3	11
bart	day_3	17
kim	day_3	16

name	day_1	day_2	day_3
eileen	10	11	11
bart	9	10	17
kim	3	15	16

name	day	turnips
eileen	day_1	10
bart	day_1	9
kim	day_1	3
eileen	day_2	11
bart	day_2	10
kim	day_2	15
eileen	day_3	11
bart	day_3	17
kim	day_3	16

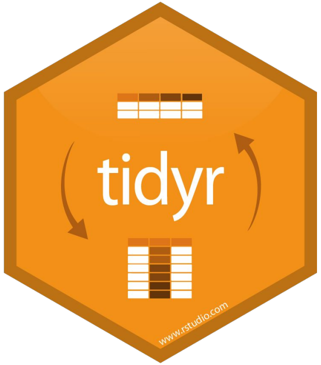
What changed?

1. The variable “turnips” is a column
2. The variable “day” is a separate column
3. Values in all other columns are duplicated

Tidying verbs

Package “**tidyr**”

- `gather()` to make a table tidy (wide to long)
- `spread()` to make a table untidy (long to wide)



Tidy data example

Input:

1. A set of columns that represent values
2. The name for the variable that collects the values (key)
3. The name of the variable whose values are spread over the cells (value)

```
gather(myturnips,
           `day_1`,`day_2`,`day_3`,
            key = "day",
         value = "turnips")
```

name	day	turnips
eileen	day_1	10
bart	day_1	9
kim	day_1	3
eileen	day_2	11
bart	day_2	10
kim	day_2	15
eileen	day_3	11
bart	day_3	17
kim	day_3	16

Why tidy data?

- It provides a standard way of structuring a dataset
- It allows R's vectorised nature to shine
- dplyr, ggplot2, and all the other packages in the tidyverse are designed to work with tidy data

Tidying data: structuring datasets **to facilitate analysis**

A bit of history
http://vita.had.co.nz/papers/tidy-data.html

Demonstration

We will use myturnips

```
myturnips <- tibble(
  name = c("eileen", "bart", "kim"),
  day_1 = c(10, 9, 3),
  day_2 = c(11, 10, 15),
  day_3 = c(11, 17, 16)
)
```

And the animals dataset

Exercises tidyr

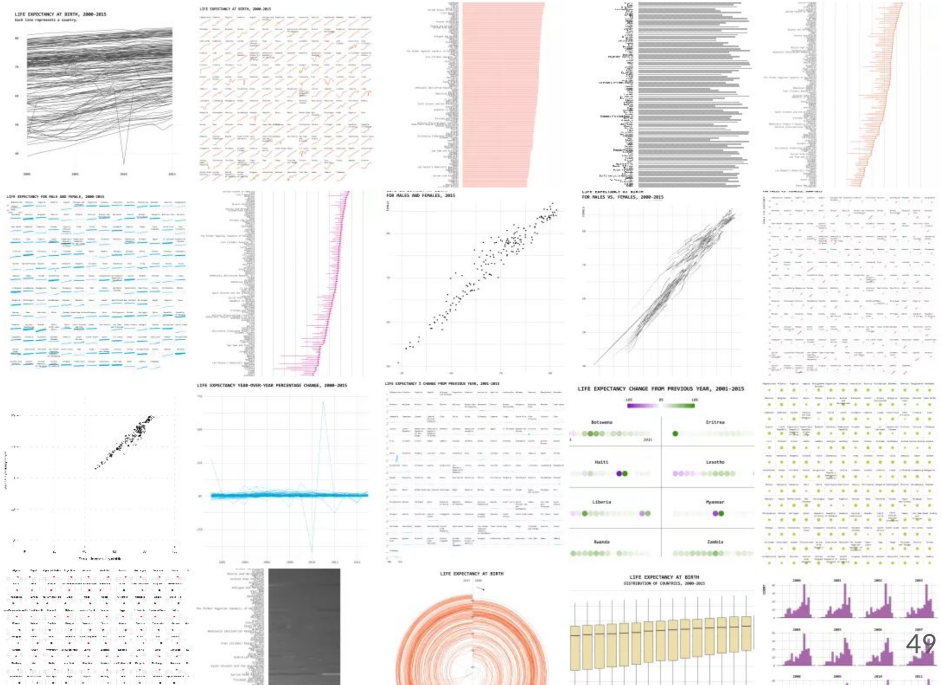
1. **Read in** the file otutable.tsv
2. **Tidy the tibble**. Results: tibble with three columns: taxon, sample, abundance. Dimensions:152360x3
3. Visualise the distribution of abundance using a density plot.
4. Add a fourth column with **relative abundances within a sample**. Call it “rel_ab_sample. Hint: abundance divided by the sum of abundances per sample. Dimensions 152360x4
5. Filter the tibble so that only **taxa with a mean relative abundance of at least 1% are retained**. Important: this is not the same as filtering out rows with rel_ab_sample < 1%. Hint: first try to make a column “mean_taxon_rel_ab”. Resulting dimensions: 30472x5
6. Extra: Visualise the distribution of mean_taxon_rel_ab using a density plot.
7. Extra: **Make a tile plot** to visualise mean_taxon_rel_ab. Put the sample on the x-axis and taxon on the y-axis. Extra: Which taxa is the one with higher value of mean_taxon_rel_ab?

Extra examples and useful functions

Explore the design space

- Once you know the grammar
  you have lots of possibilities!

- Same dataset visualised 25 times



https://flowingdata.com/2017/01/24/one-dataset-visualized-25-ways/

Additional: ggplot2 tweaking

- Transform axes
  `scale_y_log10()`
- Rename titles
  labs() or `xlab(“My x-axis”), ylab(“My y-axis”), ggtitle(“My awesome plot”)`
- More beautiful colours: RColorBrewer
  `scale_colour_brewer()`
- Setting themes
  `theme_bw(), theme_linedraw(), theme_minimal(), ...`
- Customizing themes
  `theme( ... )`

Other useful verbs

```
add_count(tibble, vars) # to add a column with redundant counts

count(tibble, vars) # to summarise  and add a column with counts

str_replace(string, pattern, replacement) # replace a pattern in a vector

str_detect(string, pattern)# to check patterns in a vector
```

Joining tables

Verbs:

- `left_join()`
- `right_join()`
- `inner_join()`
- `full_join()`

Joins by columns with same name

Splitting and merging columns

```
separate(tible, col, into, sep, remove)

unite(data, col, vars, sep)
```

Other tidyverse packages

● forcats	Solve common problems with factors
● *stringr*	Work with strings
● *purr*	Functional programming

https://www.tidyverse.org/packages/

Feedback

Your feedback is important to us

https://tiny.cc/elixir_feedback

Find more workshops
https://tess.elixir-europe.org/