README

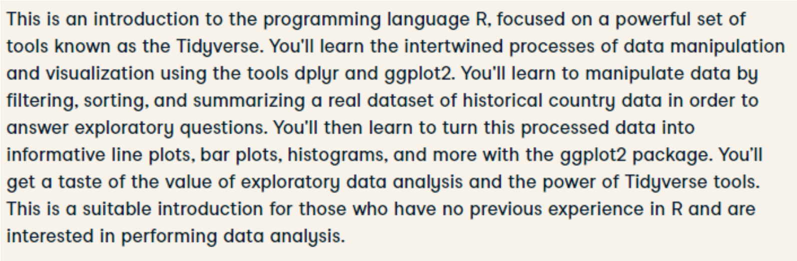
- Review of the 1st chapter of the DataCamp course
- Add your name and `(pledged)` in the `#+AUTHOR:` meta headline
- When you've completed the file, submit it in Canvas
- You'll get solutions after the deadline has passed

DONE Identify yourself

1. Add your name and `(pledged)` at the top
2. Run the `#+PROPERTY` and `#+STARTUP` lines with `C-c C-c`
3. You should open the R session in the same directory as this file
4. To check, run `getwd()` in the R console window

DONE Answer conceptual questions

- **What is the requirement for this course and where can you find this information?** - "Introduction to the Tidyverse" (Bottom of dashboard)

  This is an introduction to the programming language R, focused on a powerful set of tools known as the Tidyverse. You'll learn the intertwined processes of data manipulation and visualization using the tools dplyr and ggplot2. You'll learn to manipulate data by filtering, sorting, and summarizing a real dataset of historical country data in order to answer exploratory questions. You'll then learn to turn this processed data into informative line plots, bar plots, histograms, and more with the ggplot2 package. You'll get a taste of the value of exploratory data analysis and the power of Tidyverse tools. This is a suitable introduction for those who have no previous experience in R and are interested in performing data analysis.

- **What is the "Tidyverse"?** It's a bundle of R packages including `ggplot2` (which predates the "Tidyverse" by several years), `dplyr` for data frame manipulation, and many more. Its functions rely on data being "tidy", which corresponds to Codds 3rd normal form for relational or tabular data.

(Image source: hbctraining.github.io)

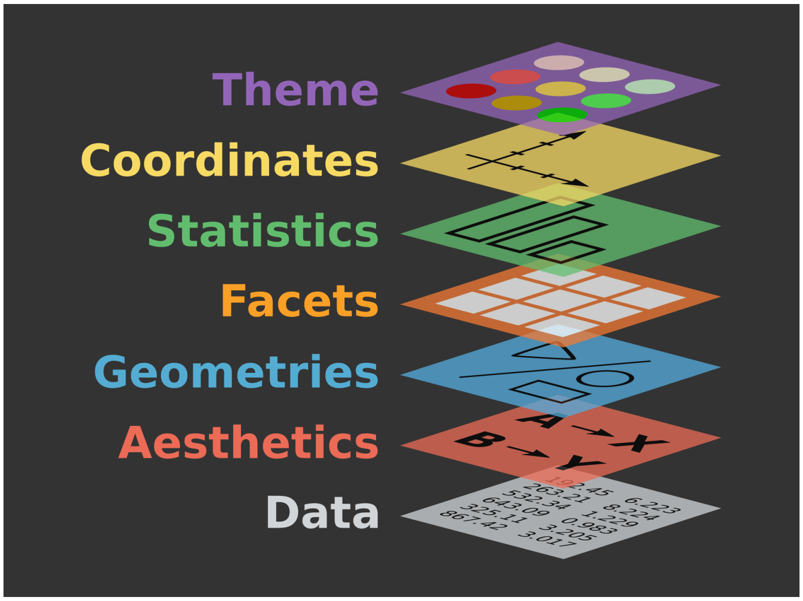
- **Base R is the foundation that every data scientist should know.**

  > "If the user knows base-R (not difficult), she can handle any situation with just a few simple operations. The old adage applies: "Give a man a fish, and he can eat for a day. Teach him how to fish, and he can eat for a lifetime." From: "TidyverseSceptic" (Matloff, 2022)

- **What is the "Grammar of Graphics"?**

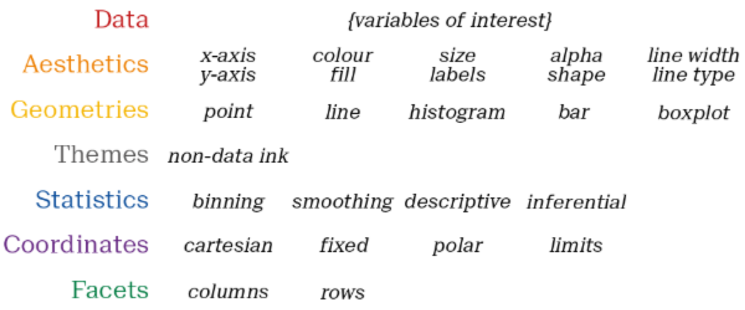
  > The "Grammar of Graphics" (gg) is a plotting framework by Leland Wilkinson (1999) implemented in R's `ggplot2` plotting package. Its core **ideas** are:
  >
  > 1. Graphics are distinct layers of 'grammatical' elements
  > 2. Plots are given meaning through 'aesthetic' mappings

- **What are the layers of the 'Grammar of Graphics'?**



(Image source: r.qcbs.ca)

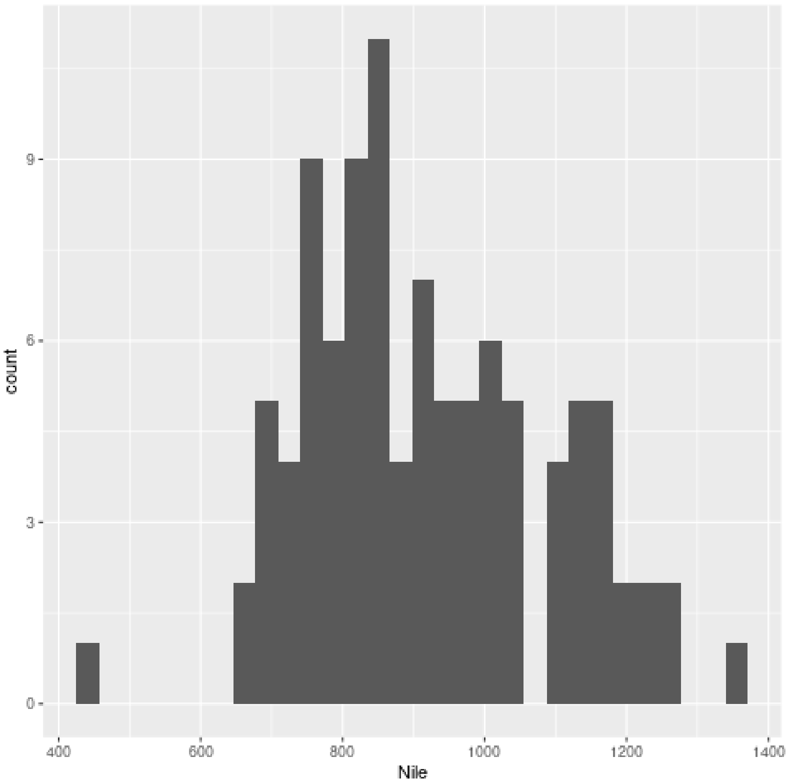
- **What are some examples for these elements in `ggplot2`?**



Data	{variables of interest}				
Aesthetics	x-axis y-axis	colour fill	size labels	alpha shape	line width line type
Geometries	point	line	histogram	bar	boxplot
Themes	non-data ink				
Statistics	binning	smoothing	descriptive	inferential	
Coordinates	cartesian	fixed	polar	limits	
Facets	columns	rows			

- **Can you save a `ggplot2` plot as an R object?**

```
library(ggplot2)
g <- ggplot(
  data = data.frame(Nile),
  aes(Nile)) +
  geom_histogram()
g
```



- Show the structure and attributes of `g`:

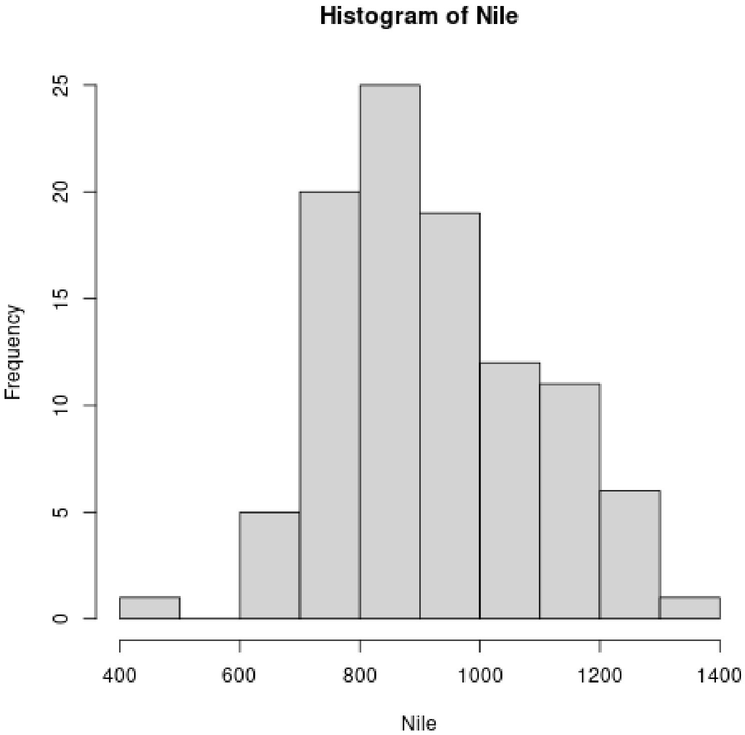
```
attributes(g)
str(g)


$names
 [1] "data"        "layers"      "scales"      "guides"      "mapping"     "theme'
 [7] "coordinates" "facet"       "plot_env"    "layout"      "labels"

$class
[1] "gg"     "ggplot"
List of 11
 $ data       :'data.frame': 100 obs. of  1 variable:
  ..$ Nile: Time-Series [1:100] from 1871 to 1970: 1120 1160 963 1210 1160 1160 81
 $ layers     :List of 1
  ..$ :Classes 'LayerInstance', 'Layer', 'ggproto', 'gg' <ggproto object: Class La
  aes_params: list
  compute_aesthetics: function
  compute_geom_1: function
  compute_geom_2: function
  compute_position: function
  compute_statistic: function
  computed_geom_params: list
  computed_mapping: uneval
  computed_stat_params: list
  constructor: call
  data: waiver
  draw_geom: function
  finish_statistics: function
  geom: <ggproto object: Class GeomBar, GeomRect, Geom, gg>
aesthetics: function
default_aes: uneval
draw_group: function
draw_key: function
draw_layer: function
draw_panel: function
extra_params: just na.rm orientation
handle_na: function
non_missing_aes: xmin xmax ymin ymax
optional_aes:
parameters: function
rename_size: TRUE
required_aes: x y
setup_data: function
setup_params: function
use_defaults: function
super:  <ggproto object: Class GeomRect, Geom, gg>
  geom_params: list
  inherit.aes: TRUE
  layer_data: function
  map_statistic: function
  mapping: NULL
  position: <ggproto object: Class PositionStack, Position, gg>
compute_layer: function
compute_panel: function
fill: FALSE
```

```
 required_aes:
 reverse: FALSE
 setup_data: function
 setup_params: function
 type: NULL
 vjust: 1
 super:  <ggproto object: Class Position, gg>
   print: function
   setup_layer: function
   show.legend: NA
   stat: <ggproto object: Class StatBin, Stat, gg>
 aesthetics: function
 compute_group: function
 compute_layer: function
 compute_panel: function
 default_aes: uneval
 dropped_aes: weight
 extra_params: na.rm orientation
 finish_layer: function
 non_missing_aes:
 optional_aes:
 parameters: function
 required_aes: x|y
 retransform: TRUE
 setup_data: function
 setup_params: function
 super:  <ggproto object: Class Stat, gg>
   stat_params: list
   super:  <ggproto object: Class Layer, gg>
$ scales     :Classes 'ScalesList', 'ggproto', 'gg' <ggproto object: Class Scales
   add: function
   add_defaults: function
   add_missing: function
   backtransform_df: function
   clone: function
   find: function
   get_scales: function
   has_scale: function
   input: function
   map_df: function
   n: function
   non_position_scales: function
   scales: list
   train_df: function
   transform_df: function
   super:  <ggproto object: Class ScalesList, gg>
$ guides     :Classes 'Guides', 'ggproto', 'gg' <ggproto object: Class Guides, gg
   add: function
   assemble: function
   build: function
   draw: function
   get_custom: function
   get_guide: function
   get_params: function
   get_position: function
   guides: NULL
   merge: function
   missing: <ggproto object: Class GuideNone, Guide, gg>
 add_title: function
 arrange_layout: function
 assemble_drawing: function
 available_aes: any
 build_decor: function
 build_labels: function
 build_ticks: function
 build_title: function
 draw: function
 draw_early_exit: function
 elements: list
 extract_decor: function
 extract_key: function
 extract_params: function
 get_layer_key: function
 hashables: list
 measure_grobs: function
 merge: function
 override_elements: function
 params: list
 process_layers: function
 setup_elements: function
 setup_params: function
 train: function
 transform: function
 super:  <ggproto object: Class GuideNone, Guide, gg>
   package_box: function
   print: function
   process_layers: function
   setup: function
   subset_guides: function
   train: function
   update_params: function
   super:  <ggproto object: Class Guides, gg>
$ mapping    :List of 1
 ..$ x: language ~Nile
 .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
 ..- attr(*, "class")= chr "uneval"
$ theme      : list()
$ coordinates:Classes 'CoordCartesian', 'Coord', 'ggproto', 'gg' <ggproto object
   aspect: function
   backtransform_range: function
   clip: on
   default: TRUE
   distance: function
   expand: TRUE
   is_free: function
   is_linear: function
   labels: function
   limits: list
   modify_scales: function
   range: function
   render_axis_h: function
   render_axis_v: function
   render_bg: function
   render_fg: function
   setup_data: function
   setup_layout: function
   setup_panel_guides: function
   setup_panel_params: function
   setup_params: function
   train_panel_guides: function
   transform: function
   super:  <ggproto object: Class CoordCartesian, Coord, gg>
$ facet      :Classes 'FacetNull', 'Facet', 'ggproto', 'gg' <ggproto object: Clas
   compute_layout: function
   draw_back: function
   draw_front: function
   draw_labels: function
   draw_panels: function
   finish_data: function
   init_scales: function
   map_data: function
   params: list
   setup_data: function
   setup_params: function
   shrink: TRUE
   train_scales: function
   vars: function
   super:  <ggproto object: Class FacetNull, Facet, gg>
$ plot_env   :<environment: R_GlobalEnv>
$ layout     :Classes 'Layout', 'ggproto', 'gg' <ggproto object: Class Layout, gg
   coord: NULL
   coord_params: list
   facet: NULL
   facet_params: list
   finish_data: function
   get_scales: function
   layout: NULL
   map_position: function
   panel_params: NULL
   panel_scales_x: NULL
   panel_scales_y: NULL
   render: function
   render_labels: function
   reset_scales: function
   resolve_label: function
   setup: function
   setup_panel_guides: function
   setup_panel_params: function
   train_position: function
   super:  <ggproto object: Class Layout, gg>
```

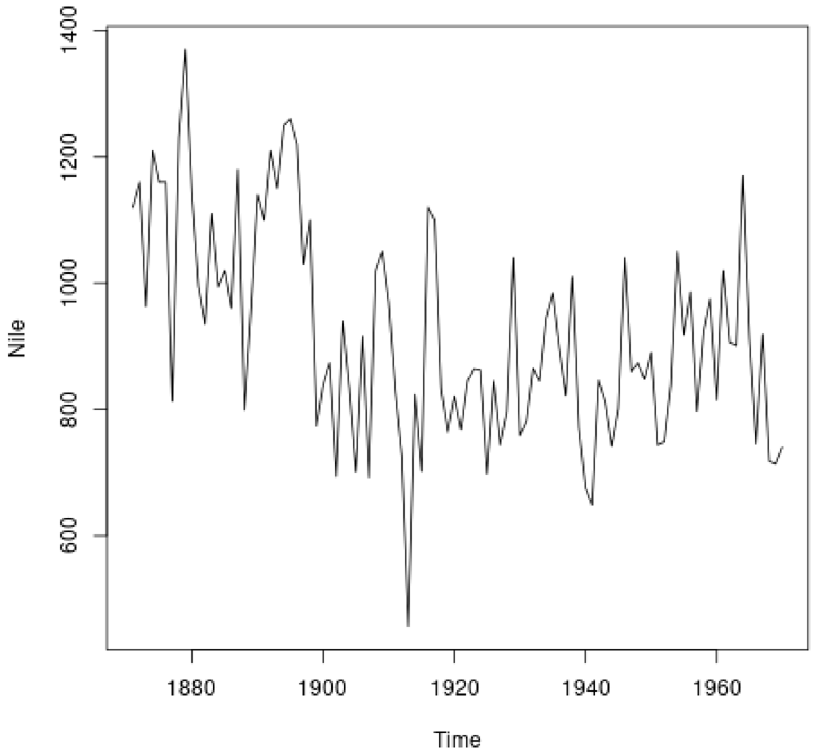
```
 $ labels     :List of 3
  ..$ x     : chr "Nile"
  ..$ y     : chr "count"
  .. ..- attr(*, "fallback")= logi TRUE
  ..$ weight: chr "weight"
  .. ..- attr(*, "fallback")= logi TRUE
 - attr(*, "class")= chr [1:2] "gg" "ggplot"
```

- **Can you save a** `base R` **plot as an R object?**

  ```
  h <- hist(Nile)
  h
  ```



```
p <- plot(Nile)
p
```



```
attributes(h)  # saving histogram to access its attributes and
  # structure
str(h)


$names
[1] "breaks"   "counts"   "density"  "mids"     "xname"    "equidist"

$class
[1] "histogram"
List of 6
 $ breaks  : int [1:11] 400 500 600 700 800 900 1000 1100 1200 1300 ...
 $ counts  : int [1:10] 1 0 5 20 25 19 12 11 6 1
 $ density : num [1:10] 0.0001 0 0.0005 0.002 0.0025 0.0019 0.0012 0.0011 0.0006 0.000
 $ mids    : num [1:10] 450 550 650 750 850 950 1050 1150 1250 1350
 $ xname   : chr "Nile"
 $ equidist: logi TRUE
 - attr(*, "class")= chr "histogram"
```

- **Can you combine `ggplot2` and `base R` graphics in one plot array?** Answer: no. Base R graphics and ggplot2 graphics are completely different and cannot be mixed. In base R, plots are created by opening graphics devices, in ggplot2, plots are layered R objects.

DONE Create simple scatterplots

We're going to work with `MASS::mammals` using `ggplot2` and `base R`.

1. Load the relevant packages.

   ```
   library(MASS)
   library(ggplot2)
   ```

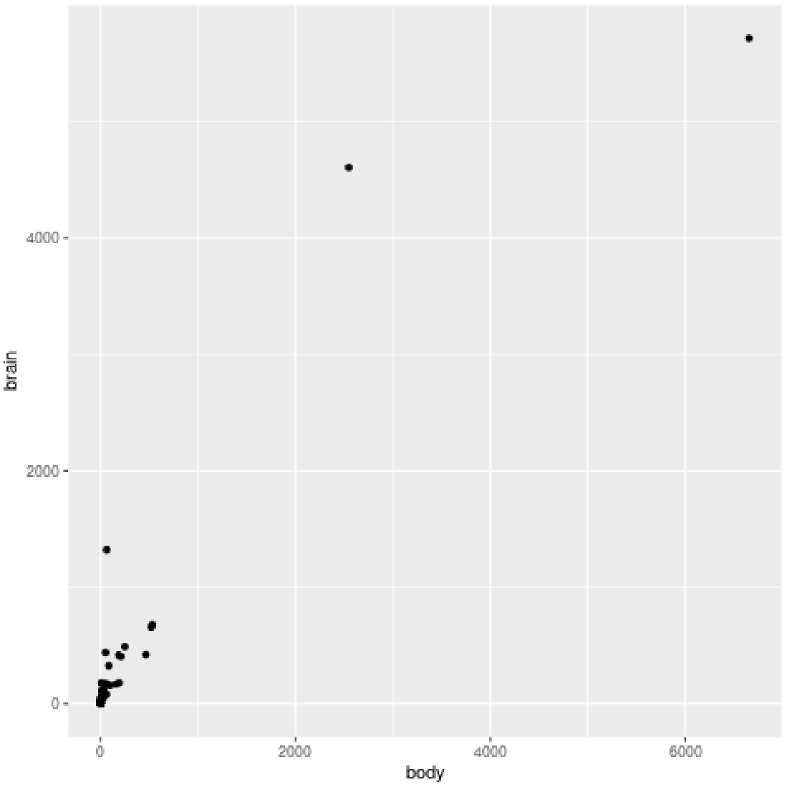
2. Show the data structure of `mammals`.

   ```
   str(mammals)
   ```

   ```
   'data.frame':	62 obs. of  2 variables:
    $ body : num  3.38 0.48 1.35 465 36.33 ...
    $ brain: num  44.5 15.5 8.1 423 119.5 ...
   ```

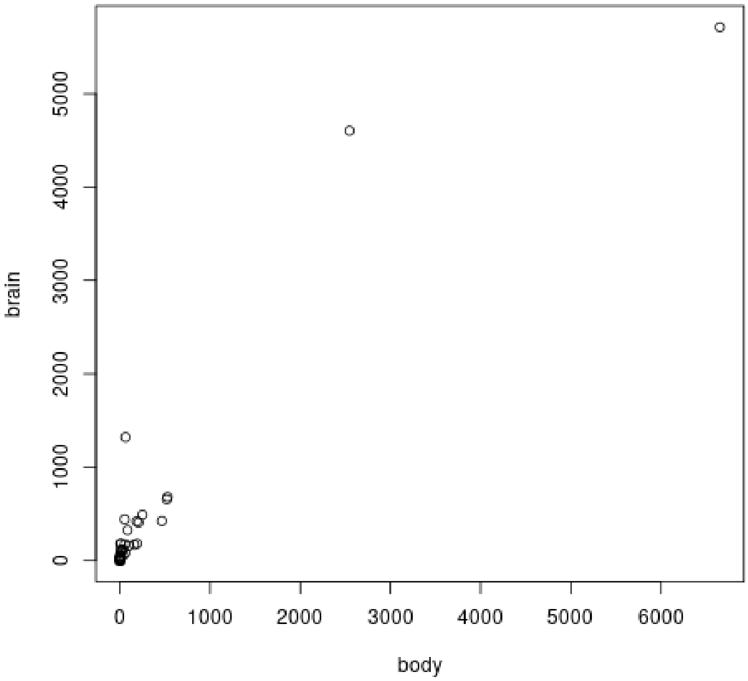
3. Create a scatterplot of `brain` vs `body` of the `mammals` data set in `ggplot2`.

   ```
   ggplot(mammals, aes(x=body,y=brain)) +
     geom_point()
   ```



4. Create a scatterplot of `brain` vs `body` of the `mammals` data set in `base R`.

```
plot(mammals)
```



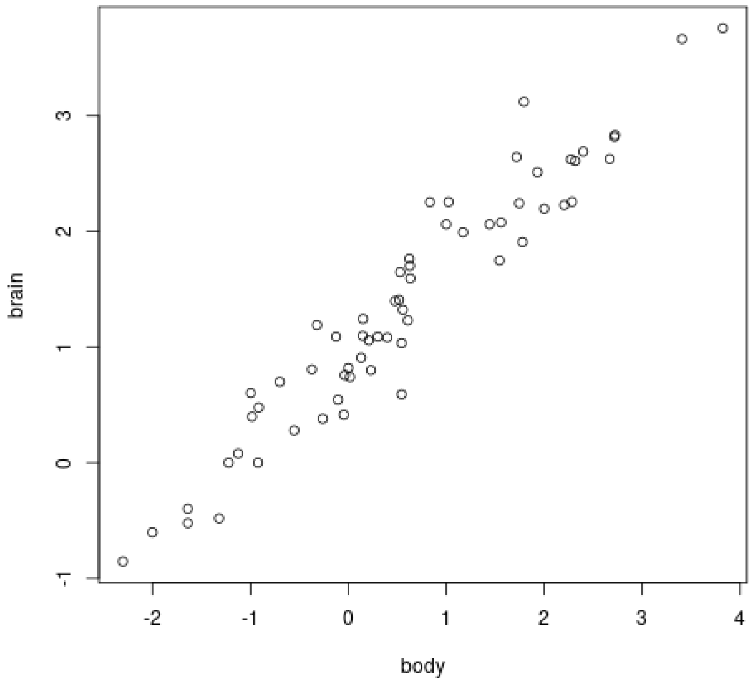
DONE Transform plots

1. What's the problem with these plots and what could you do about it?

   > **The problem:** the data points are too bunched up because mammals have a wide spectrum of body and brain weights (there are very small and very large ones).
   >
   > **The solution:** transform the x- and y-axis logarithmically.

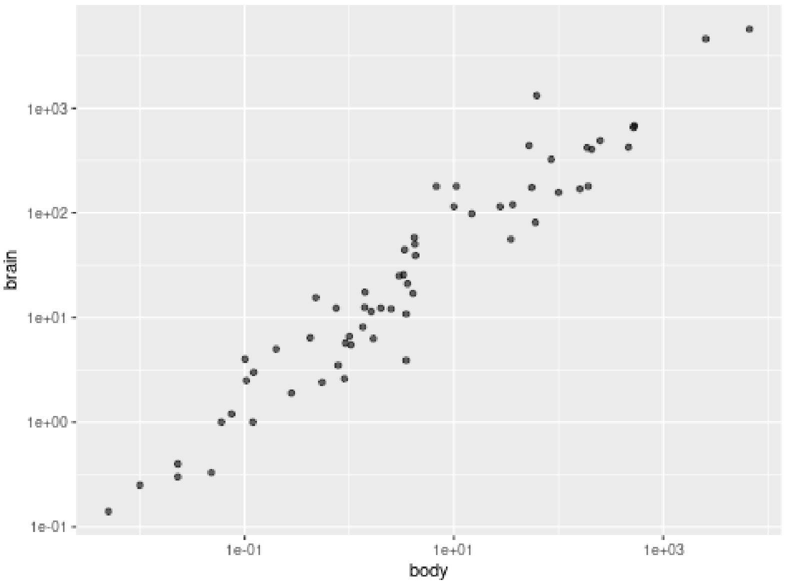
2. Implement the solution with `plot`.

```
plot(log10(mammals))
```



3. Implement the solution with ggplot - save the plot as gg for later, and print it.

```
gg <- ggplot(
  data=mammals,
  aes(x=body,y=brain)) +
  geom_point(alpha=0.6) +
  coord_fixed() +
  scale_x_log10() +
  scale_y_log10()
gg
```



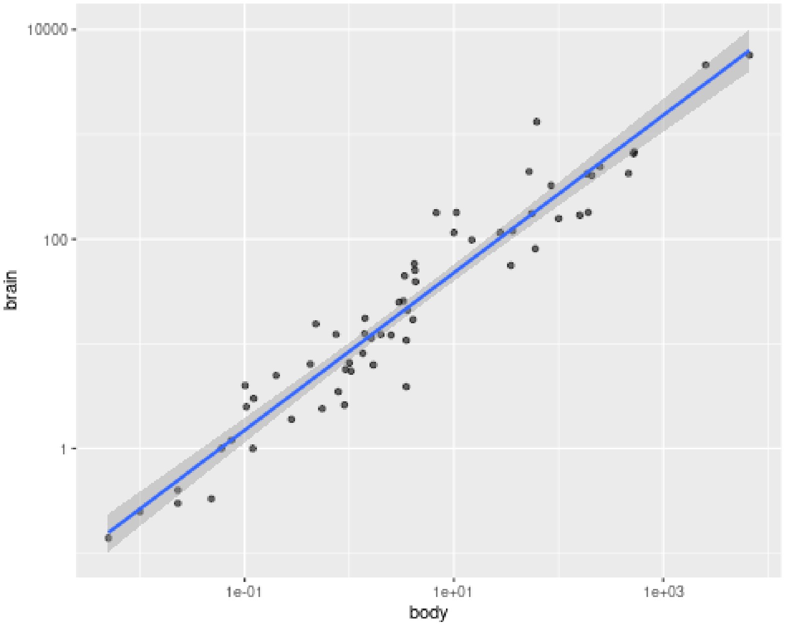
4. What does the `geom_point` argument `alpha` do?

   Answer: it reduces the transparency of the points by 40%.

DONE Create trendlines with `ggplot2` and base R

1. Create a linear trendline for the `ggplot2` plot `gg`. Inside the smoothing geometry, use `method="lm"` to fix the model.

```
gg +
  geom_smooth(method="lm")
```

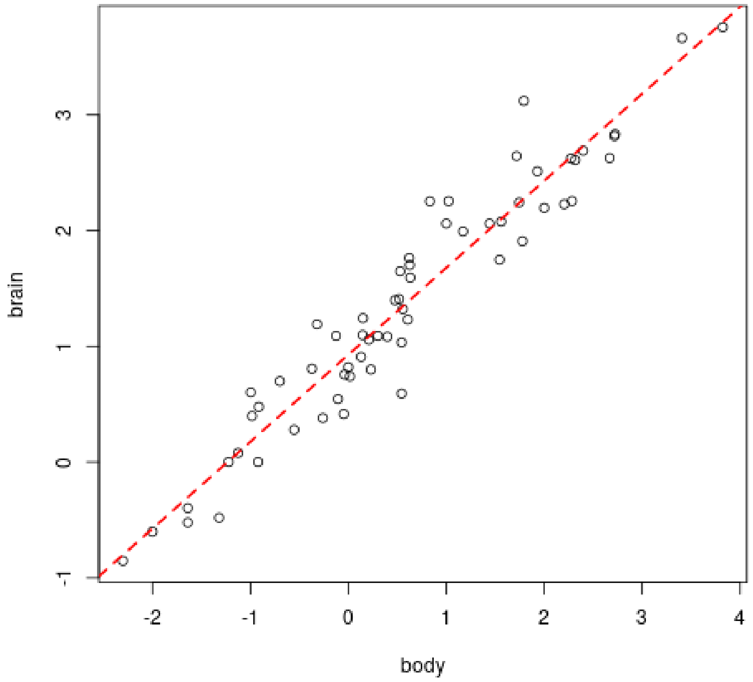


2. Create a linear model in `base R` using `lm` and `data-log10(mammals)`. Save it as `line` and print it.

   ```
   line <- lm(brain ~ body, data=log10(mammals))
   line



   Call:
   lm(formula = brain ~ body, data = log10(mammals))

   Coefficients:
   (Intercept)         body
        0.9271       0.7517
   ```

3. Create a trendline plot in `base R` using the linear model. The line should be red, dashed and double wide.

```
plot(log10(mammals))
abline(line, col="red", lty=2, lwd=2)
```

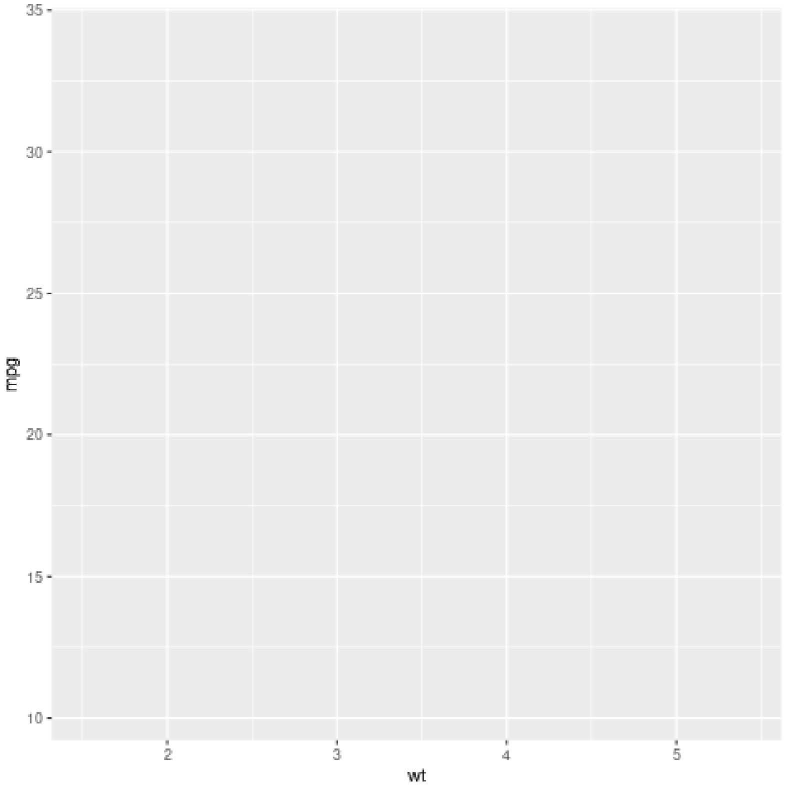


DONE Map 'aesthetics' to variables

Recall that the `mtcars` data frame lists the characteristics mileage (`mpg`), weight (`wt`) and number of cylinders (`cyl`) as `numeric` variables.

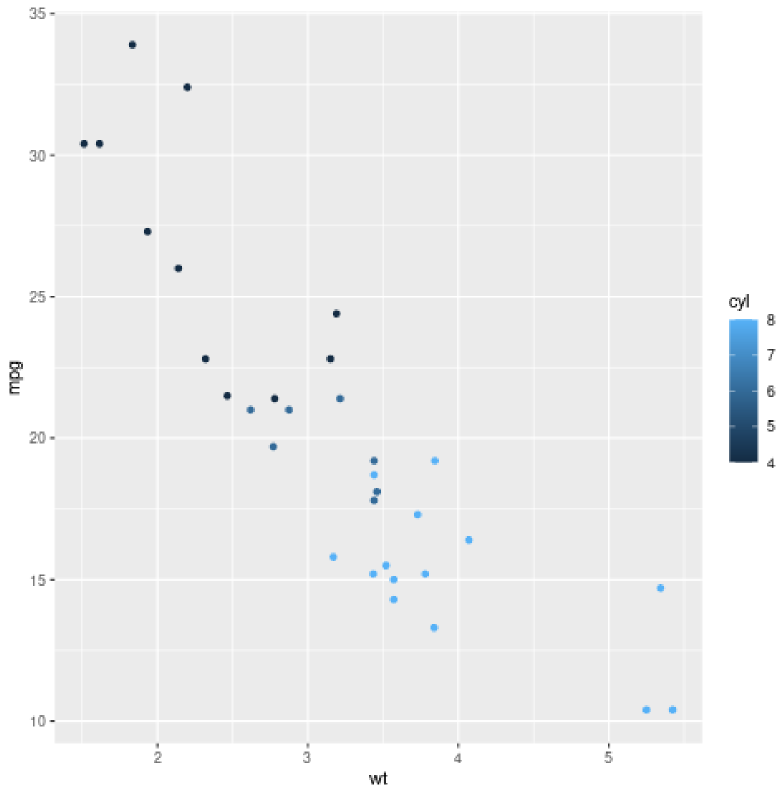
1. Create a `ggplot` of mileage vs. weight using `ggplot2`, save it as `gg` and print it.

   ```
   gg <- ggplot(data=mtcars, aes(wt,mpg))
   gg
   ```



2. Create a scatterplot where the color 'aesthetic' is mapped to the number of cylinders by adding a 'geometry' to gg.

```
ggplot(data=mtcars,
  aes(wt,mpg)) +
  geom_point(
    aes(color=cyl))
```



3. What's the difference between mapping 'aesthetics' inside the 'geometry' or inside the `ggplot` function?

   Answer: the `aes` function knows about the dataset from `data`. You can also pipe the data set into the function using `|>` or `%>%`. Without `aes`, you need to specify `geom_point(mtcars$cyl)`. You can subset data for a specific geometry by putting `aes` into the function.