

A quick tour of R

Abhijit Dasgupta, PhD

Spring 2019

The bare essentials

- Everything in R is an object with a name
- Things are nouns
 - Nouns in R are *data objects*, like scalars, matrices, data.frames/tibbles, strings, vectors
- Nouns are acted upon by verbs
 - Verbs in R are *functions*, like `mean(x)`, `nrow(d)`, `dim(d)`, `ggplot` and so on
- You can modify verbs with adverbs
 - Adverbs in R are *function options*, like `mean(x, na.rm=T)`, `geom_point(color='green')`

The bare essentials

- You have to name things to store them
 - This is done with the `<-` operator, e.g.
 - `mn <- mean(x, na.rm=T)` stores the result of the average
 - `my_theme <- function() theme_bw() + theme(axis.title=element_text(size=14))` stores the function, which you'll call as `my_theme()`

You can see the objects you have created either by typing `ls()` in the console, or looking in the Environment pane

Note, built-in objects don't show up in the Environment pane or using `ls()`

The bare essentials

There are three kinds of brackets in R

`[]` are used for extracting elements from arrays, matrices, data frames.

- `x[3]` is the 3rd element of an array `x`
- `d[1, 3]` is the element in the 1st row and 3rd column of a matrix/data frame `d`
- `d[2,]` is the entire first **row** of a matrix/data frame `d`
- `d[, 4]` is the entire 4th **column** of a matrix/data frame `d`

The bare essentials

There are three kinds of brackets in R

`()` are used for specifying arguments to functions

- `mean(x)` gives the mean of an array of numbers `x`
- `summary(d)` gives a summary representation of a data frame `d`

The bare essentials

There are three kinds of brackets in R

`{ }` are used to contain groups of commands/statements

A conditional statement

```
if (age < 18){  
  person <- 'Minor'  
} else if (age > 65) {  
  person <- 'Senior'  
} else {  
  person <- 'Adult'  
}
```

A function definition

```
my_mean <- function(x, na.rm = T){  
  if(na.rm){  
    x <- x[!is.na(x)]  
  }  
  s <- sum(x)  
  n <- length(x)  
  mn <- s / n # There is a built-in function mean, so  
  return(mn)  
}
```

Data types

A scalar:

- 29
- "cherry"
- TRUE

Data types

A scalar:

- 29 : *numeric*
- "cherry": *character*
- TRUE: *logical*

Data types

Vectors/Arrays

These are constructed using the `c()` function (for *concatenate*).

```
c(1,2,5,6,7,8)
```

```
#> [1] 1 2 5 6 7 8
```

```
c('apple','berry','melon','citrus')
```

```
#> [1] "apple" "berry" "melon" "citrus"
```

▮ Vectors must all contain objects of the same type. Can't mix and match

Data types

Matrices (2-d arrays)

These are typically built from vectors

```
x <- c(1,2,4,5,6,7)
y <- 10:16 # Shortcut for c(10,11,12,13,14,15,16)
```

```
cbind(x, y) # Vectors as columns
```

```
#>      x  y
#> [1,] 1 10
#> [2,] 2 11
#> [3,] 4 12
#> [4,] 5 13
#> [5,] 6 14
#> [6,] 7 15
#> [7,] 1 16
```

```
rbind(x, y) # Vectors as rows
```

```
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> x       1    2    4    5    6    7    1
#> y      10   11   12   13   14   15   16
```

Data types

Lists

Lists are basically buckets or containers. Each element of a list can be anything, even other lists

```
my_list <- list('a', c(2,3,5,6), head(ggplot2::mpg))
my_list
```

```
#> [[1]]
#> [1] "a"
#>
#> [[2]]
#> [1] 2 3 5 6
#>
#> [[3]]
#> # A tibble: 6 x 11
#>   manufacturer model displ  year   cyl trans driv
#>   <chr>         <chr> <dbl> <int> <int> <chr> <chr>
#> 1 audi         a4      1.8  1999     4 auto(...) f
#> 2 audi         a4      1.8  1999     4 manua... f
#> 3 audi         a4      2    2008     4 manua... f
#> 4 audi         a4      2    2008     4 auto(...) f
#> 5 audi         a4      2.8  1999     6 auto(...) f
#> 6 audi         a4      2.8  1999     6 manua... f
```

Data types

data.frame/tibble

This is the typical container for tabular data

- must be rectangular
- each column can be of a different type
- elements within each column have to be of the same type

Data types

data.frame/tibble

```
beaches <- rio::import('data/sydneybeaches3.csv') # use the import function from the package rio
class(beaches)
```

```
#> [1] "data.frame"
```

```
dim(beaches)
```

```
#> [1] 344 12
```

```
head(beaches)
```

```
#>      date year month day season rainfall temperature enterococci
#> 1 2013-01-02 2013     1   2      1      0.0         23.4         6.7
#> 2 2013-01-06 2013     1   6      1      0.0         30.3         2.0
#> 3 2013-01-12 2013     1  12      1      0.0         31.4        69.1
#> 4 2013-01-18 2013     1  18      1      0.0         46.4         9.0
#> 5 2013-01-24 2013     1  24      1      0.0         27.5        33.9
#> 6 2013-01-30 2013     1  30      1      0.6         26.6        26.5
#>      day_num month_num month_name season_name
#> 1          2          1   January      Summer
```

Data types

data.frame/tibble

```
library(tidyverse) # Activate the tidyverse package
beaches_t <- as_tibble(beaches)
class(beaches_t)
```

```
#> [1] "tbl_df"      "tbl"        "data.frame"
```

```
beaches_t
```

```
#> # A tibble: 344 x 12
#>   date   year month   day season rainfall temperature enterococci day_num
#>   <chr> <int> <int> <int> <int>    <dbl>      <dbl>      <dbl>    <int>
#> 1 2013... 2013     1     2     1         0        23.4         6.7         2
#> 2 2013... 2013     1     6     1         0        30.3         2         6
#> 3 2013... 2013     1    12     1         0        31.4        69.1        12
#> 4 2013... 2013     1    18     1         0        46.4         9        18
#> 5 2013... 2013     1    24     1         0        27.5        33.9        24
#> 6 2013... 2013     1    30     1         0.6        26.6        26.5        30
#> 7 2013... 2013     2     5     1         0.1        25.7        66.9        36
#> 8 2013... 2013     2    11     1         8        22.2       118.         42
#> 9 2013... 2013     2    17     1        13.6        26.3         75         48
#> 10 2013... 2013     2    23     1         7.2        24.8       311.         54
#> # ... with 334 more rows, and 3 more variables: month_num <int>,
```

Data types

data.frame/tibble

Extracting columns from a data frame:

1. `beaches$temperature`
2. `beaches[, 'temperature']`
3. `beaches[['temperature']]`
4. `beaches[, 7]`
5. `beaches[[7]]`

Packages in R

R is a modular environment with some base functionality that is augmented by **packages** (think of them as modules)

- Packages can contain *functions* and *data*
- There are over 15K packages on CRAN, the Comprehensive R Archive Network
- There are over 1600 packages on Bioconductor, the main repository for bioinformatics resources
 - Analytic, Annotation, Experimental data and Workflow packages

Finding packages

1. CRAN [Task views](#){target=_blank}
2. Bioconductor [BiocViews](#){target=_blank}
3. GitHub (open source collaboration and version control environment)

Installing packages

From CRAN

```
install.packages("tidyverse")
```

From Bioconductor

```
install.packages("BiocManager") # do once  
BiocManager::install('limma')
```

From GitHub

```
install.packages('remotes') # do once  
remotes::install_github("rstudio/rmarkdown") # usual
```

GitHub often hosts development version of packages published on CRAN or Bioconductor

Both CRAN and Bioconductor have stringent checks to make sure packages can run properly, with no obvious program flaws. There are typically no guarantees about analytic or theoretical correctness, but most packages have been crowd-validated and there are several reliable developer groups including RStudio

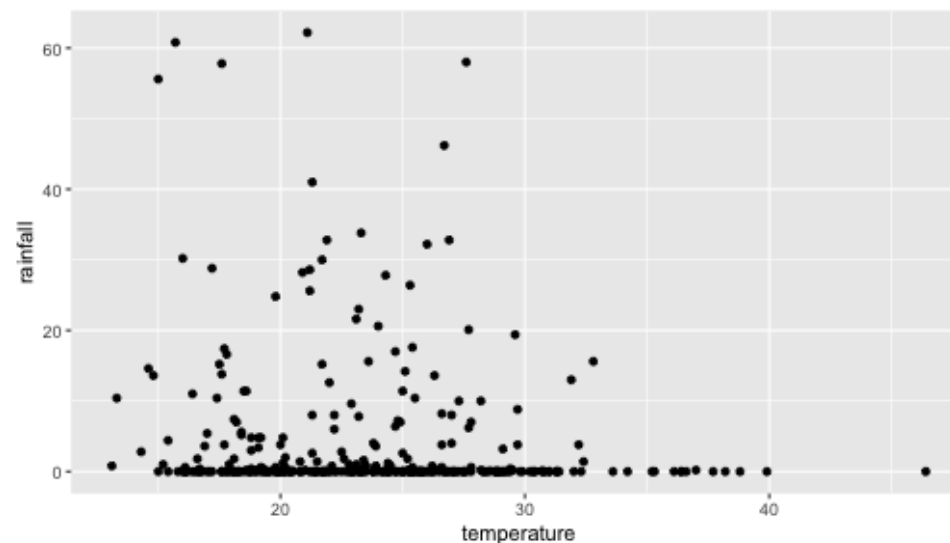
Using packages

You have to first "activate" the package in your current working session using the `library` function.

```
ggplot(beaches, aes(temperature, rainfall)) +  
  geom_point()
```

```
#> Error in ggplot(beaches, aes(temperature, rainfall)) :
```

```
library(ggplot2) # or library(tidyverse)  
ggplot(beaches, aes(temperature, rainfall)) +  
  geom_point()
```



Tidying data using the tidyverse

What is the "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

What is the "Tidyverse"?

A set of R packages that:

- help make data more computer-friendly
- while making your code more human-friendly
- Most of these packages are (co-)written by Dr. Hadley Wickham, who has rockstar status in the R world
- They are supported by the company RStudio

Tidying data

Tidy data

Tidy datasets are all alike,
but every messy data is messy in its own way

Tidy data

Tidy data is a **computer-friendly** format based on the following characteristics:

- Each row is one observation
- Each column is one variable
- Each set of observational unit forms a table

All other forms of data can be considered **messy data**.

Let us count the ways

There are many ways data can be messy. An incomplete list....

- Column headers are values, not variables
- Multiple variables are stored in a single column
- Variables are stored in both rows and columns
- Multiple types of observational units are saved in the same table
- A single observational unit is stored in multiple tables

Ways to have messy (i.e. not tidy) data

1. Column headers contain values

Country	< \$10K	\$10-20K	\$20-50K	\$50-100K	> \$100K
India	40	25	25	9	1
USA	20	20	20	30	10

Ways to have messy (i.e. not tidy) data

Column headers contain values

Country	Income	Percentage
India	< \$10K	40
USA	< \$10K	20

This is a case of reshaping or melting

Ways to have messy (i.e. not tidy) data

Multiple variables in one column

Country	Year	M_0-14	F_0-14	M_15-60	F_15-60	M_60+	F_60+
UK	2010						
UK	2011						

Separating columns into different variables

Country	Year	Gender	Age	Count
---------	------	--------	-----	-------

Tidying data

The typical steps are

- Transforming data from wide to tall (*gather*) and from tall to wide (*spread*)
- Separating columns into different columns
- Putting columns together into new variables

Cleaning data

Some actions on data

- Creating new variables (*mutate*)
- Choose some columns (*select*)
- Selecting rows based on some criteria (*filter*)
- Sort data based on some variables (*arrange*)

Example data

```
head(mtcars, 3)
```

```
#>      mpg cyl  disp  hp drat   wt  qsec vs am gear carb  
#> Mazda RX4      21.0   6  160 110 3.90 2.620 16.46 0  1    4    4  
#> Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02 0  1    4    4  
#> Datsun 710     22.8   4  108  93 3.85 2.320 18.61 1  1    4    1
```

- Car names are in an attribute of the data.frame called `rownames`. So it's not in a column
- We might want to convert fuel economy to metric
- We might just want to look at the relationship between displacement and fuel economy based on number of cylinders

Example data ([link](https://dl.dropboxusercontent.com/s/pqavhcckshqxtjm/brca.csv))

```
link <- 'https://dl.dropboxusercontent.com/s/pqavhcckshqxtjm/brca.csv'
brca_data <- rio::import(link)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave.points_mean	symmetry_mean
1	842302	M	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.300100	0.147100	0.041500
2	842517	M	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.086900	0.070170	0.083600
3	84300903	M	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.197400	0.127900	0.086600
4	84348301	M	11.420	20.38	77.58	386.1	0.14250	0.28390	0.241400	0.105200	0.081600
5	84358402	M	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.198000	0.104300	0.081600
6	843786	M	12.450	15.70	82.57	477.1	0.12780	0.17000	0.157800	0.080890	0.081600
7	844359	M	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.112700	0.074000	0.081600

The tidyverse package

The tidyverse package is a meta-package that installs a set of packages that are useful for data cleaning, data tidying and data munging (manipulating data to get a computationally "attractive" dataset)

The tidyverse package

```
# install.packages('tidyverse')  
library(tidyverse)
```

You can specify a function from a particular package as `dplyr::filter`. Note there are two colons there

Core tidyverse packages

Package	Description
ggplot2	Data visualization
tibble	data.frame on steroids
tidyr	Data tidying (today)
readr	Reading text files (CSV)
purrr	Applying functions to data iteratively
dplyr	Data cleaning and munging (today)
stringr	String (character) manipulation
forcats	Manipulating categorical variables

Additional tidyverse packages

Package	Description
readxl	Read Excel files
haven	Read SAS, SPSS, Stata files
lubridate	Deal with dates and times
magrittr	Provides the pipe operator %>%
glue	Makes pasting text and data easier

Additional useful packages

Package	Description
broom	Turns the results of models or analysis into tidy datasets
fs	Allows directory and file manipulation in OS-agnostic manner
here	Allows robust specification of directory structure in a Project

Pipes

Pipes

Pipes (denoted %>%, spoken as "then") are to analytic pipelines as + is to ggplot layers

```
mpg1 <- mpg %>% mutate(id=1:n()) %>% select(id, year, trans, cty, hwy)
mpg_metric <- mpg1 %>%
  mutate_at(vars(cty, hwy), function(x) {x * 1.6/3.8})
```

Original data

id	year	trans	cty	hwy
1	1999	auto(l5)	18	29
2	1999	manual(m5)	21	29
3	2008	manual(m6)	20	31
4	2008	auto(av)	21	30
5	1999	auto(l5)	16	26

Transformed data

id	year	trans	cty	hwy
1	1999	auto(l5)	7.578947	12.21053
2	1999	manual(m5)	8.842105	12.21053
3	2008	manual(m6)	8.421053	13.05263
4	2008	auto(av)	8.842105	12.63158
5	1999	auto(l5)	6.736842	10.94737

Note I'm assigning a name to the transformed data. Otherwise it'll be lost

Verbs to use in pipes

The verbs in `tidyverse` are specially useful in pipes

Verb	Functionality
<code>mutate</code>	Transform a column with some function
<code>select</code>	Select some columns in the data
<code>arrange</code>	Order the data frame by values of a column(s)
<code>filter</code>	Keep only rows that meet some data criterion
<code>group_by</code>	Group by levels of a variable
<code>gather</code>	Transform a wide dataset to a long dataset
<code>spread</code>	Transform a long dataset to a wide dataset
<code>separate</code>	Separate one column into several columns
<code>unite</code>	Concatenate several columns into 1 column

Pipes almost always start with a `data.frame/tibble` object, and then "pipes" that data through different transformations (functions)

At each `%>%`, the results of the previous step are used as input for the next step.

A complicated example

Grab the raw data

```
url <- "http://varianceexplained.org/files/Brauer2008_DataSet1.tds"
raw_data <- read_delim(url, delim='\t')
head(raw_data)
```

```
#> # A tibble: 6 x 40
#>   GID   YORF  NAME  GWEIGHT G0.05  G0.1  G0.15  G0.2  G0.25  G0.3  N0.05  N0.1
#>   <chr> <chr> <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 GENE... A_06... SFB2...      1 -0.24 -0.13 -0.21 -0.15 -0.05 -0.05  0.2  0.24
#> 2 GENE... A_06... "      1  0.28  0.13 -0.4  -0.48 -0.11  0.17  0.31  0
#> 3 GENE... A_06... QRI7...      1 -0.02 -0.27 -0.27 -0.02  0.24  0.25  0.23  0.06
#> 4 GENE... A_06... CFT2...      1 -0.33 -0.41 -0.24 -0.03 -0.03  0      0.2 -0.25
#> 5 GENE... A_06... SSO2...      1  0.05  0.02  0.4   0.34 -0.13 -0.14 -0.35 -0.09
#> 6 GENE... A_06... PSP2...      1 -0.69 -0.03  0.23  0.2   0    -0.27  0.17 -0.4
#> # ... with 28 more variables: N0.15 <dbl>, N0.2 <dbl>, N0.25 <dbl>,
#> #   N0.3 <dbl>, P0.05 <dbl>, P0.1 <dbl>, P0.15 <dbl>, P0.2 <dbl>,
#> #   P0.25 <dbl>, P0.3 <dbl>, S0.05 <dbl>, S0.1 <dbl>, S0.15 <dbl>,
#> #   S0.2 <dbl>, S0.25 <dbl>, S0.3 <dbl>, L0.05 <dbl>, L0.1 <dbl>,
#> #   L0.15 <dbl>, L0.2 <dbl>, L0.25 <dbl>, L0.3 <dbl>, U0.05 <dbl>,
#> #   U0.1 <dbl>, U0.15 <dbl>, U0.2 <dbl>, U0.25 <dbl>, U0.3 <dbl>
```

Look at the annotation data

```
head(raw_data$NAME)
```

```
#> [1] "SFB2"      || ER to Golgi transport || molecular function unknown || YNL049C || 1082129"  
#> [2] "          || biological process unknown || molecular function unknown || YNL095C || 1086222"  
#> [3] "QRI7"      || proteolysis and peptidolysis || metalloendopeptidase activity || YDL104C || 1085955"  
#> [4] "CFT2"      || mRNA polyadenylation* || RNA binding || YLR115W || 1081958"  
#> [5] "SS02"      || vesicle fusion* || t-SNARE activity || YMR183C || 1081214"  
#> [6] "PSP2"      || biological process unknown || molecular function unknown || YML017W || 1083036"
```

Separate annotation into columns

```
cleaned_data <- raw_data %>%
  separate(NAME, c("name", "BP", "MF", "systematic_name", "number"),
           sep = "\\|\\|\\|")
head(cleaned_data)
```

```
#> # A tibble: 6 x 44
#>   GID  YORF name BP    MF    systematic_name number GWEIGHT G0.05 G0.1
#>   <chr> <chr> <chr> <chr> <chr> <chr>          <chr>   <dbl> <dbl> <dbl>
#> 1 GENE... A_06... "SFB... " ER... " mo... " YNL049C "      " 108...      1 -0.24 -0.13
#> 2 GENE... A_06... "    ... " bi... " mo... " YNL095C "      " 108...      1  0.28  0.13
#> 3 GENE... A_06... "QRI... " pr... " me... " YDL104C "      " 108...      1 -0.02 -0.27
#> 4 GENE... A_06... "CFT... " mR... " RN... " YLR115W "      " 108...      1 -0.33 -0.41
#> 5 GENE... A_06... "SSO... " ve... " t-... " YMR183C "      " 108...      1  0.05  0.02
#> 6 GENE... A_06... "PSP... " bi... " mo... " YML017W "      " 108...      1 -0.69 -0.03
#> # ... with 34 more variables: G0.15 <dbl>, G0.2 <dbl>, G0.25 <dbl>,
#> #   G0.3 <dbl>, N0.05 <dbl>, N0.1 <dbl>, N0.15 <dbl>, N0.2 <dbl>,
#> #   N0.25 <dbl>, N0.3 <dbl>, P0.05 <dbl>, P0.1 <dbl>, P0.15 <dbl>,
#> #   P0.2 <dbl>, P0.25 <dbl>, P0.3 <dbl>, S0.05 <dbl>, S0.1 <dbl>,
#> #   S0.15 <dbl>, S0.2 <dbl>, S0.25 <dbl>, S0.3 <dbl>, L0.05 <dbl>,
#> #   L0.1 <dbl>, L0.15 <dbl>, L0.2 <dbl>, L0.25 <dbl>, L0.3 <dbl>,
#> #   U0.05 <dbl>, U0.1 <dbl>, U0.15 <dbl>, U0.2 <dbl>, U0.25 <dbl>,
#> #   U0.3 <dbl>
```

Get rid of padding in annotation

```
cleaned_data <- raw_data %>%
  separate(NAME, c("name", "BP", "MF", "systematic_name", "number"),
    sep = "\\|\\|\\|") %>%
  mutate_at(vars(name:systematic_name), funs(stringr::str_trim))
head(cleaned_data)
```

```
#> # A tibble: 6 x 44
#>   GID   YORF name BP   MF   systematic_name number GWEIGHT G0.05 G0.1
#>   <chr> <chr> <chr> <chr> <chr> <chr>          <chr>    <dbl> <dbl> <dbl>
#> 1 GENE... A_06... SFB2 ER t... mole... YNL049C      " 108...      1 -0.24 -0.13
#> 2 GENE... A_06... "    biol... mole... YNL095C      " 108...      1  0.28  0.13
#> 3 GENE... A_06... QRI7 prot... meta... YDL104C      " 108...      1 -0.02 -0.27
#> 4 GENE... A_06... CFT2 mRNA... RNA ... YLR115W      " 108...      1 -0.33 -0.41
#> 5 GENE... A_06... SS02 vesi... t-SN... YMR183C      " 108...      1  0.05  0.02
#> 6 GENE... A_06... PSP2 biol... mole... YML017W      " 108...      1 -0.69 -0.03
#> # ... with 34 more variables: G0.15 <dbl>, G0.2 <dbl>, G0.25 <dbl>,
#> #   G0.3 <dbl>, N0.05 <dbl>, N0.1 <dbl>, N0.15 <dbl>, N0.2 <dbl>,
#> #   N0.25 <dbl>, N0.3 <dbl>, P0.05 <dbl>, P0.1 <dbl>, P0.15 <dbl>,
#> #   P0.2 <dbl>, P0.25 <dbl>, P0.3 <dbl>, S0.05 <dbl>, S0.1 <dbl>,
#> #   S0.15 <dbl>, S0.2 <dbl>, S0.25 <dbl>, S0.3 <dbl>, L0.05 <dbl>,
#> #   L0.1 <dbl>, L0.15 <dbl>, L0.2 <dbl>, L0.25 <dbl>, L0.3 <dbl>,
#> #   U0.05 <dbl>, U0.1 <dbl>, U0.15 <dbl>, U0.2 <dbl>, U0.25 <dbl>,
#> #   U0.3 <dbl>
```

Get rid of some columns

```
cleaned_data <- raw_data %>%
  separate(NAME, c("name", "BP", "MF", "systematic_name", "number"),
    sep = "\\|\\|\\|") %>%
  mutate_at(vars(name:systematic_name), funs(stringr::str_trim)) %>%
  select(-number, -GID, -YORF, -GWEIGHT)
head(cleaned_data)
```

```
#> # A tibble: 6 x 40
#>   name BP MF systematic_name G0.05 G0.1 G0.15 G0.2 G0.25 G0.3
#>   <chr> <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 SFB2 ER t... mole... YNL049C -0.24 -0.13 -0.21 -0.15 -0.05 -0.05
#> 2 "" biol... mole... YNL095C 0.28 0.13 -0.4 -0.48 -0.11 0.17
#> 3 QRI7 prot... meta... YDL104C -0.02 -0.27 -0.27 -0.02 0.24 0.25
#> 4 CFT2 mRNA... RNA ... YLR115W -0.33 -0.41 -0.24 -0.03 -0.03 0
#> 5 SS02 vesi... t-SN... YMR183C 0.05 0.02 0.4 0.34 -0.13 -0.14
#> 6 PSP2 biol... mole... YML017W -0.69 -0.03 0.23 0.2 0 -0.27
#> # ... with 30 more variables: N0.05 <dbl>, N0.1 <dbl>, N0.15 <dbl>,
#> # N0.2 <dbl>, N0.25 <dbl>, N0.3 <dbl>, P0.05 <dbl>, P0.1 <dbl>,
#> # P0.15 <dbl>, P0.2 <dbl>, P0.25 <dbl>, P0.3 <dbl>, S0.05 <dbl>,
#> # S0.1 <dbl>, S0.15 <dbl>, S0.2 <dbl>, S0.25 <dbl>, S0.3 <dbl>,
#> # L0.05 <dbl>, L0.1 <dbl>, L0.15 <dbl>, L0.2 <dbl>, L0.25 <dbl>,
#> # L0.3 <dbl>, U0.05 <dbl>, U0.1 <dbl>, U0.15 <dbl>, U0.2 <dbl>,
#> # U0.25 <dbl>, U0.3 <dbl>
```

Make data tidy

```
cleaned_data <- raw_data %>%
  separate(NAME, c("name", "BP", "MF", "systematic_name", "number"),
    sep = "\\|\\|\\|") %>%
  mutate_at(vars(name:systematic_name), funs(stringr::str_trim)) %>%
  select(-number, -GID, -YORF, -GWEIGHT) %>%
  tidyr::gather(sample, expression, G0.05:U0.3)
head(cleaned_data)
```

```
#> # A tibble: 6 x 6
#>   name BP MF systematic_name sample expression
#>   <chr> <chr> <chr> <chr> <chr> <dbl>
#> 1 SFB2 ER to Golgi tra... molecular funct... YNL049C G0.05 -0.24
#> 2 "" biological proc... molecular funct... YNL095C G0.05 0.28
#> 3 QRI7 proteolysis and... metalloendopept... YDL104C G0.05 -0.02
#> 4 CFT2 mRNA polyadenyl... RNA binding YLR115W G0.05 -0.33
#> 5 SS02 vesicle fusion* t-SNARE activity YMR183C G0.05 0.05
#> 6 PSP2 biological proc... molecular funct... YML017W G0.05 -0.69
```

Split columns

```
cleaned_data <- raw_data %>%
  separate(NAME, c("name", "BP", "MF", "systematic_name", "number"),
    sep = "\\|\\|\\|") %>%
  mutate_at(vars(name:systematic_name), funs(stringr::str_trim)) %>%
  select(-number, -GID, -YORF, -GWEIGHT) %>%
  tidyr::gather(sample, expression, G0.05:U0.3) %>%
  separate(sample, c("nutrient", "rate"), sep=1, convert = TRUE)
head(cleaned_data)
```

```
#> # A tibble: 6 x 7
#>   name BP MF systematic_name nutrient rate expression
#>   <chr> <chr> <chr> <chr> <chr> <dbl> <dbl>
#> 1 SFB2 ER to Golgi... molecular f... YNL049C G 0.05 -0.24
#> 2 "" biological ... molecular f... YNL095C G 0.05 0.28
#> 3 QRI7 proteolysis... metalloendo... YDL104C G 0.05 -0.02
#> 4 CFT2 mRNA polyad... RNA binding YLR115W G 0.05 -0.33
#> 5 SS02 vesicle fus... t-SNARE act... YMR183C G 0.05 0.05
#> 6 PSP2 biological ... molecular f... YML017W G 0.05 -0.69
```

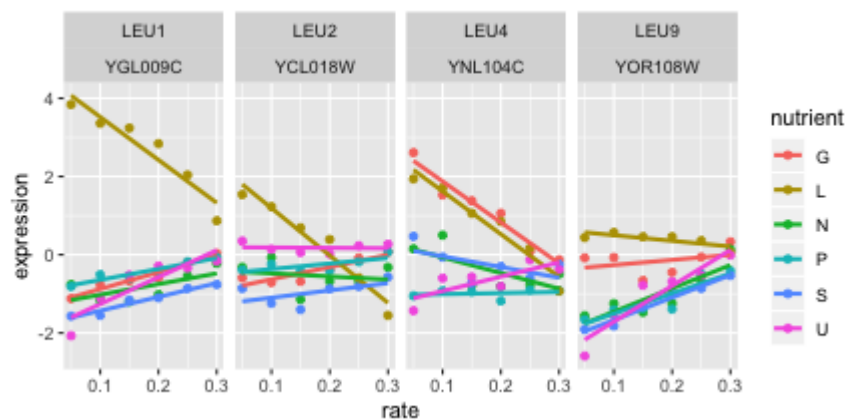

Get rid of rows with missing expression or name

```
cleaned_data <- raw_data %>%
  separate(NAME, c("name", "BP", "MF", "systematic_name", "number"),
            sep = "\\|\\|\\|") %>%
  mutate_at(vars(name:systematic_name), funs(stringr::str_trim)) %>%
  select(-number, -GID, -YORF, -GWEIGHT) %>%
  tidyr::gather(sample, expression, G0.05:U0.3) %>%
  separate(sample, c("nutrient", "rate"), sep=1, convert = TRUE) %>%
  filter(!is.na(expression), systematic_name != '')
head(cleaned_data)
```

```
#> # A tibble: 6 x 7
#>   name BP MF systematic_name nutrient rate expression
#>   <chr> <chr> <chr> <chr> <chr> <dbl> <dbl>
#> 1 SFB2 ER to Golgi... molecular f... YNL049C G 0.05 -0.24
#> 2 "" biological ... molecular f... YNL095C G 0.05 0.28
#> 3 QRI7 proteolysis... metalloendo... YDL104C G 0.05 -0.02
#> 4 CFT2 mRNA polyad... RNA binding YLR115W G 0.05 -0.33
#> 5 SS02 vesicle fus... t-SNARE act... YMR183C G 0.05 0.05
#> 6 PSP2 biological ... molecular f... YML017W G 0.05 -0.69
```

Visualize

```
cleaned_data %>%
  filter(BP == "leucine biosynthesis") %>%
  ggplot(aes(rate, expression, color = nutrient)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~name + systematic_name, nrow=1) +
  theme(legend.position='right')
```



Modeling and the broom package

Representing model relationships

In R, there is a particularly convenient way to express models, where you have

- one dependent variable
- one or more independent variables, with possible transformations and interactions

```
y ~ x1 + x2 + x1:x2 + I(x3^2) + x4*x5
```

y depends on ...

- x1 and x2 linearly
- the interaction of x1 and x2 (represented as x1 : x2)
- the square of x3 (the I() notation ensures that the ^ symbol is interpreted correctly)
- x4, x5 and their interaction (same as x4 + x5 + x4:x5)

Representing model relationships

```
y ~ x1 + x2 + x1:x2 + I(x3^2) + x4*x5
```

This interpretation holds for the vast majority of statistical models in R

- For decision trees and random forests and neural networks, don't add interactions or transformations, since the model will try to figure those out on their own

Our first model

```
library(survival)
data(pbc)
myLinearModel <- lm(chol ~ bili, data = pbc)
```

Note that everything in R is an **object**, so you can store a model in a variable name.

This statement runs the model and stored the fitted model in `myLinearModel`

R does not interpret the model, evaluate the adequacy or appropriateness of the model, or comment on whether looking at the relationship between cholesterol and bilirubin makes any kind of sense.

Our first model

```
myLinearModel
```

```
#>  
#> Call:  
#> lm(formula = chol ~ bili, data = pbc)  
#>  
#> Coefficients:  
#> (Intercept)      bili  
#>    303.20      20.24
```

Not very informative, is it?

Our first model

```
summary(myLinearModel)
```

```
#>
#> Call:
#> lm(formula = chol ~ bili, data = pbc)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -565.39  -89.90  -35.36   44.92 1285.33
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   303.204     15.601   19.435 < 2e-16 ***
#> bili          20.240       2.785    7.267 3.63e-12 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 213.2 on 282 degrees of freedom
#> (134 observations deleted due to missingness)
#> Multiple R-squared:  0.1577,    Adjusted R-squared:  0.1547
#> F-statistic: 52.8 on 1 and 282 DF,  p-value: 3.628e-12
```

A little better

Our first model

```
broom::tidy(myLinearModel)
```

```
#> # A tibble: 2 x 5
#>   term      estimate std.error statistic  p.value
#>   <chr>      <dbl>    <dbl>    <dbl>    <dbl>
#> 1 (Intercept)  303.      15.6      19.4 5.65e-54
#> 2 bili        20.2      2.79      7.27 3.63e-12
```

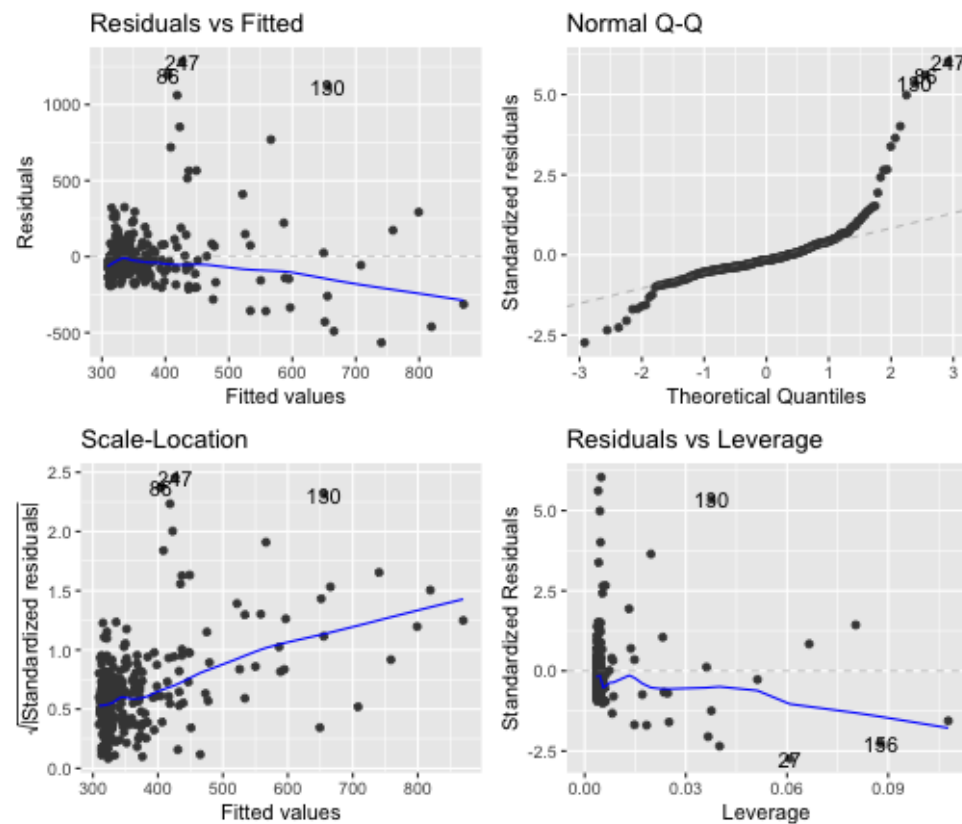
```
broom::glance(myLinearModel)
```

```
#> # A tibble: 1 x 11
#>   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC    BIC
#>   <dbl>      <dbl> <dbl>    <dbl>    <dbl> <int>  <dbl> <dbl> <dbl>
#> 1   0.158      0.155  213.     52.8 3.63e-12     2 -1925. 3856. 3867.
#> # ... with 2 more variables: deviance <dbl>, df.residual <int>
```

Our first model

We do need some sense as to how well this model fit the data

```
# install.packages('ggfortify')  
library(ggfortify)  
autoplot(myLinearModel)
```



Displaying model results

Let's start with this model:

```
myModel <- lm(log10(enterococci) ~ rainfall + temperature + season_name + factor(year), data = beaches)
broom::tidy(myModel)
```

```
#> # A tibble: 11 x 5
#>   term                estimate std.error statistic  p.value
#>   <chr>                <dbl>    <dbl>    <dbl>    <dbl>
#> 1 (Intercept)          1.23      0.208      5.92 8.39e- 9
#> 2 rainfall             0.0284    0.00296     9.58 2.72e-19
#> 3 temperature        -0.00830    0.00769    -1.08 2.81e- 1
#> 4 season_nameSpring  -0.325     0.0840    -3.87 1.31e- 4
#> 5 season_nameSummer   0.0862     0.0909     0.948 3.44e- 1
#> 6 season_nameWinter  -0.332     0.0889    -3.74 2.22e- 4
#> 7 factor(year)2014     0.0498     0.102     0.486 6.27e- 1
#> 8 factor(year)2015     0.0807     0.100     0.804 4.22e- 1
#> 9 factor(year)2016     0.0815     0.0975     0.836 4.04e- 1
#> 10 factor(year)2017   -0.0676     0.0972    -0.696 4.87e- 1
#> 11 factor(year)2018    0.0440     0.107     0.411 6.81e- 1
```

Displaying model results

Let's start with this model:

```
plt_data <- broom::tidy(myModel)
plt_data <- plt_data %>% filter(term != '(Intercept)') %>%
  mutate(term = str_replace(term, 'season_name', ''))
plt_data
```

```
#> # A tibble: 10 x 5
#>   term                estimate std.error statistic  p.value
#>   <chr>              <dbl>      <dbl>      <dbl>    <dbl>
#> 1 rainfall           0.0284    0.00296     9.58  2.72e-19
#> 2 temperature      -0.00830    0.00769    -1.08  2.81e- 1
#> 3 Spring           -0.325     0.0840    -3.87  1.31e- 4
#> 4 Summer           0.0862     0.0909     0.948 3.44e- 1
#> 5 Winter          -0.332     0.0889    -3.74  2.22e- 4
#> 6 factor(year)2014  0.0498     0.102      0.486 6.27e- 1
#> 7 factor(year)2015  0.0807     0.100      0.804 4.22e- 1
#> 8 factor(year)2016  0.0815     0.0975     0.836 4.04e- 1
#> 9 factor(year)2017 -0.0676     0.0972    -0.696 4.87e- 1
#> 10 factor(year)2018 0.0440     0.107      0.411 6.81e- 1
```

Displaying model results

Let's start with this model:

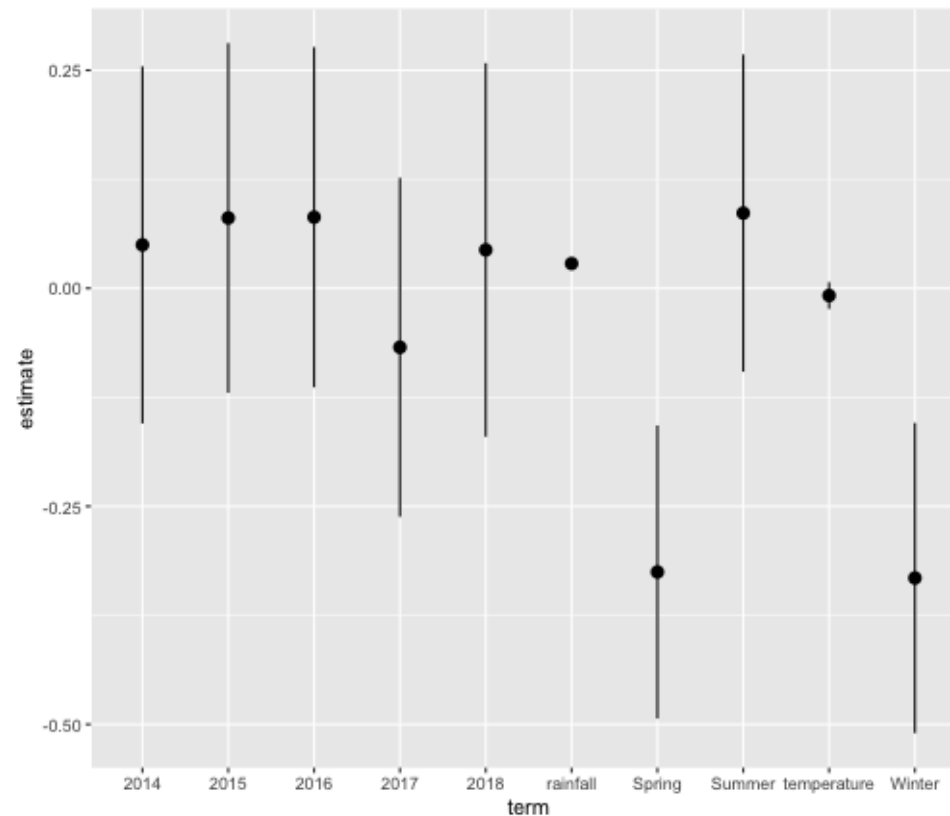
```
plt_data <- broom::tidy(myModel)
plt_data <- plt_data %>% filter(term != '(Intercept)') %>%
  mutate(term = str_replace(term, 'season_name', '')) %>%
  mutate(term = str_replace(term, 'factor\\(year\\)', '')) # Brackets are "escaped" using \\
plt_data
```

```
#> # A tibble: 10 x 5
#>   term      estimate std.error statistic  p.value
#>   <chr>      <dbl>    <dbl>    <dbl>    <dbl>
#> 1 rainfall    0.0284    0.00296     9.58 2.72e-19
#> 2 temperature -0.00830    0.00769    -1.08 2.81e- 1
#> 3 Spring      -0.325     0.0840    -3.87 1.31e- 4
#> 4 Summer       0.0862    0.0909     0.948 3.44e- 1
#> 5 Winter      -0.332     0.0889    -3.74 2.22e- 4
#> 6 2014         0.0498    0.102     0.486 6.27e- 1
#> 7 2015         0.0807    0.100     0.804 4.22e- 1
#> 8 2016         0.0815    0.0975     0.836 4.04e- 1
#> 9 2017        -0.0676    0.0972    -0.696 4.87e- 1
#> 10 2018        0.0440    0.107     0.411 6.81e- 1
```

Displaying model results

Let's start with this model:

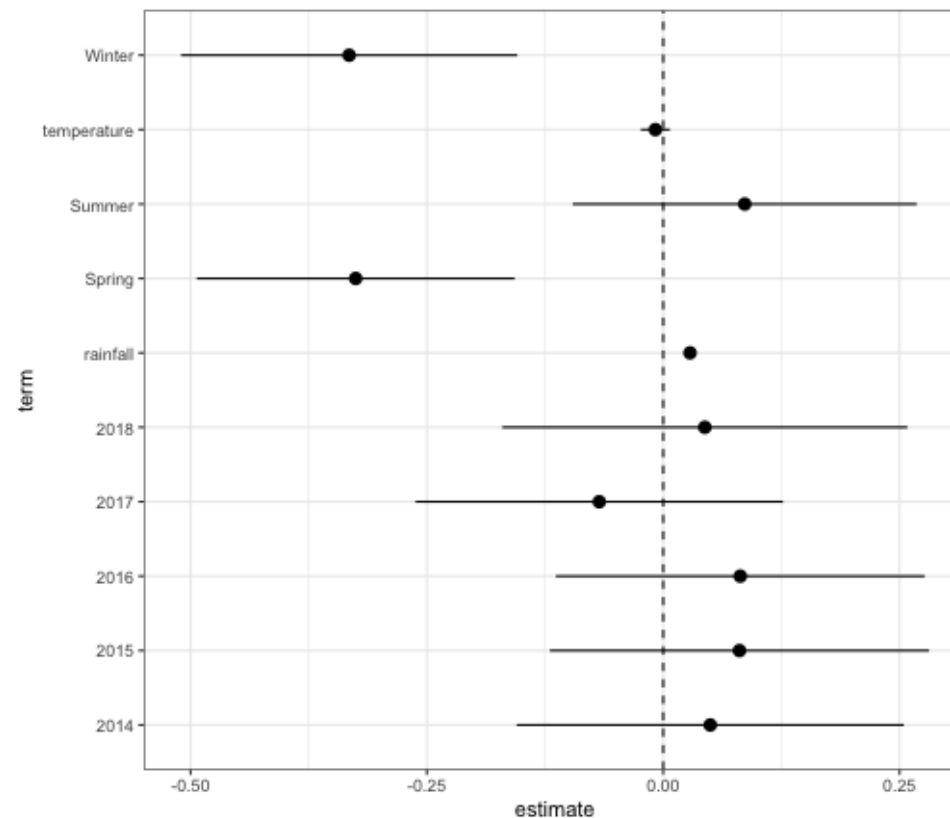
```
plt_data <- broom::tidy(myModel)
plt_data %>% filter(term != '(Intercept)') %>%
  mutate(term = str_replace(term, 'season_name', ''))
  mutate(term = str_replace(term, 'factor\\(year\\)', ''))
  ggplot(aes(x = term, y = estimate,
             ymin = estimate - 2 * std.error,
             ymax = estimate + 2 * std.error)) +
  geom_pointrange()
```



Displaying model results

Let's start with this model:

```
plt_data <- broom::tidy(myModel)
plt_data %>% filter(term != '(Intercept)') %>%
  mutate(term = str_replace(term, 'season_name', ''))
  mutate(term = str_replace(term, 'factor\\(year\\)', ''))
  ggplot(aes(x = term, y = estimate,
             ymin = estimate - 2 * std.error,
             ymax = estimate + 2 * std.error)) +
  geom_pointrange() +
  geom_hline(yintercept = 0, linetype=2) +
  theme_bw() +
  coord_flip()
```



Displaying model results

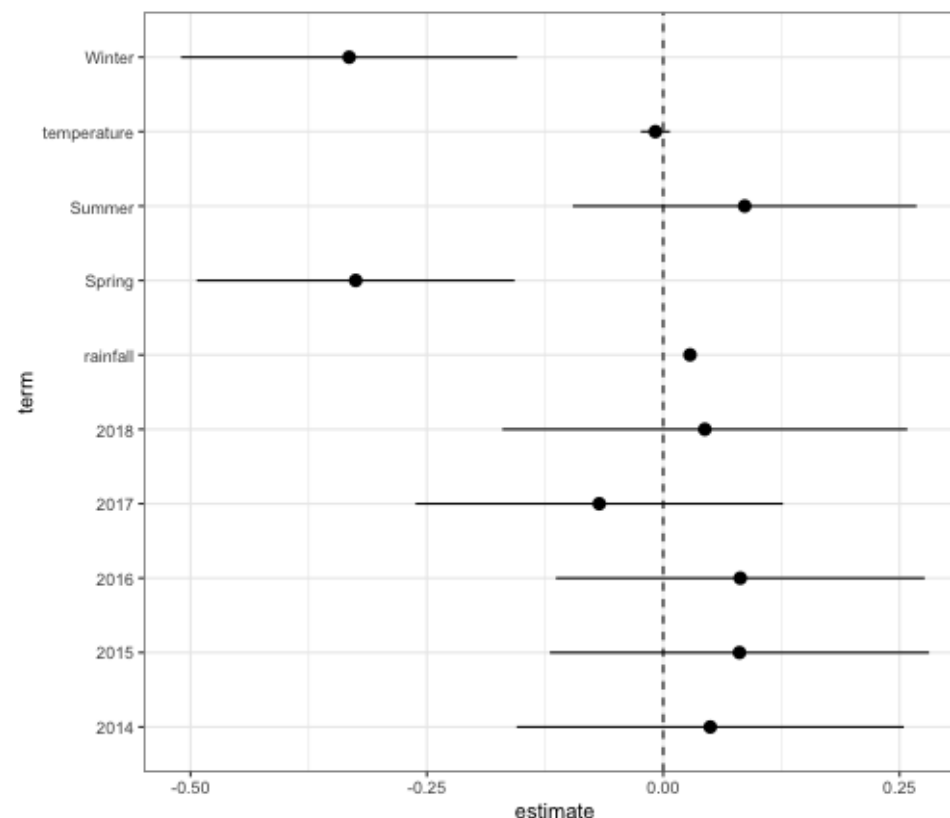
Let's start with this model:

```
plt_data <- broom::tidy(myModel)
plt_data %>% filter(term != '(Intercept)') %>%
  mutate(term = str_replace(term, 'season_name', ''))
  mutate(term = str_replace(term, 'factor\\(year\\)', ''))
  ggplot(aes(x = term, y = estimate,
             ymin = estimate - 2 * std.error,
             ymax = estimate + 2 * std.error)) +
  geom_pointrange() +
  geom_hline(yintercept = 0, linetype=2) +
  theme_bw() +
  coord_flip()

ggsave('results.png') # ggsave knows format from file
```

You can also save the graph from the RStudio Plots pane, but coding it using ggsave is more reproducible

If you need to get a high-definition TIFF, your best bet is to save your graph as a PDF and then convert

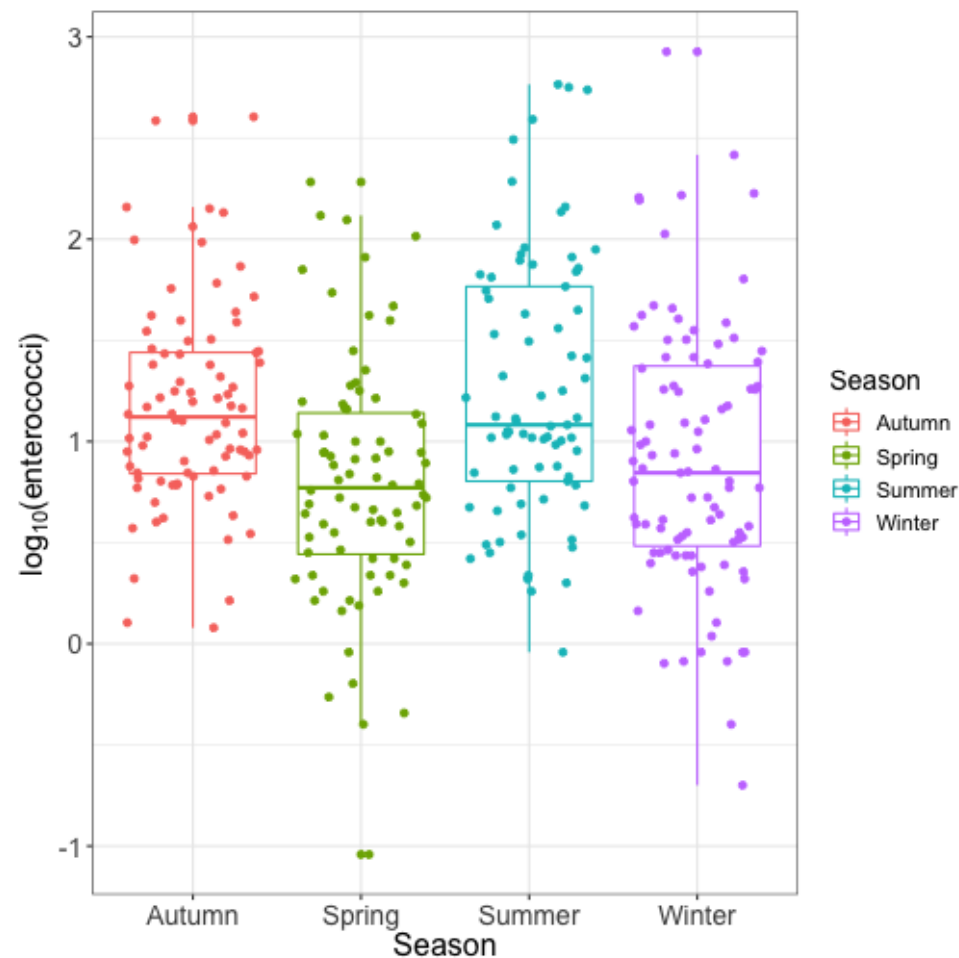


Showing group differences ("Figure 1")

The package `ggpubr`, which extends `ggplot2`, makes this very easy. It provides the function `stat_compare_means`

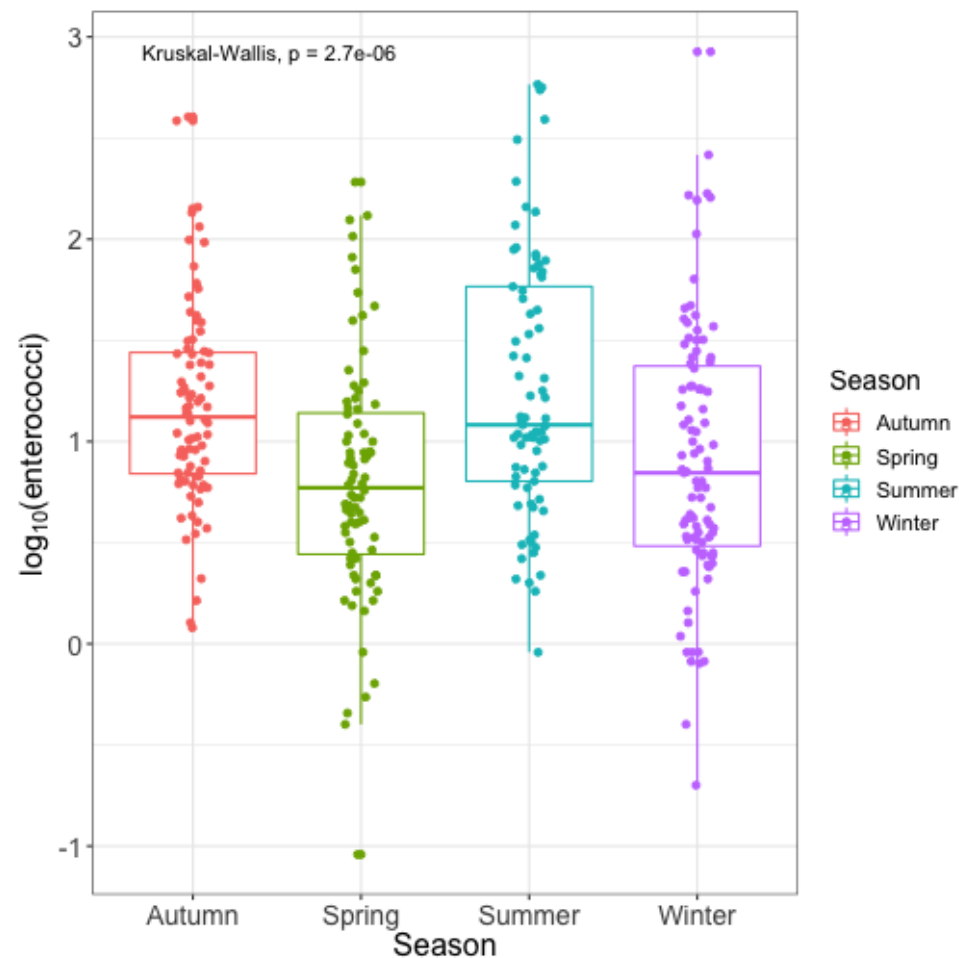
Showing group differences ("Figure 1")

```
library(ggpubr)
theme_viz <- function(){
  theme_bw() +
    theme(axis.title = element_text(size=16),
          axis.text = element_text(size=14),
          text = element_text(size = 14))
}
ggplot(
  data=beaches,
  mapping= aes(x = season_name, y = log10(enterococci)
  geom_boxplot()+geom_jitter()+
  labs(x = 'Season', y = expression(paste('log'['10']
  theme_viz()
```



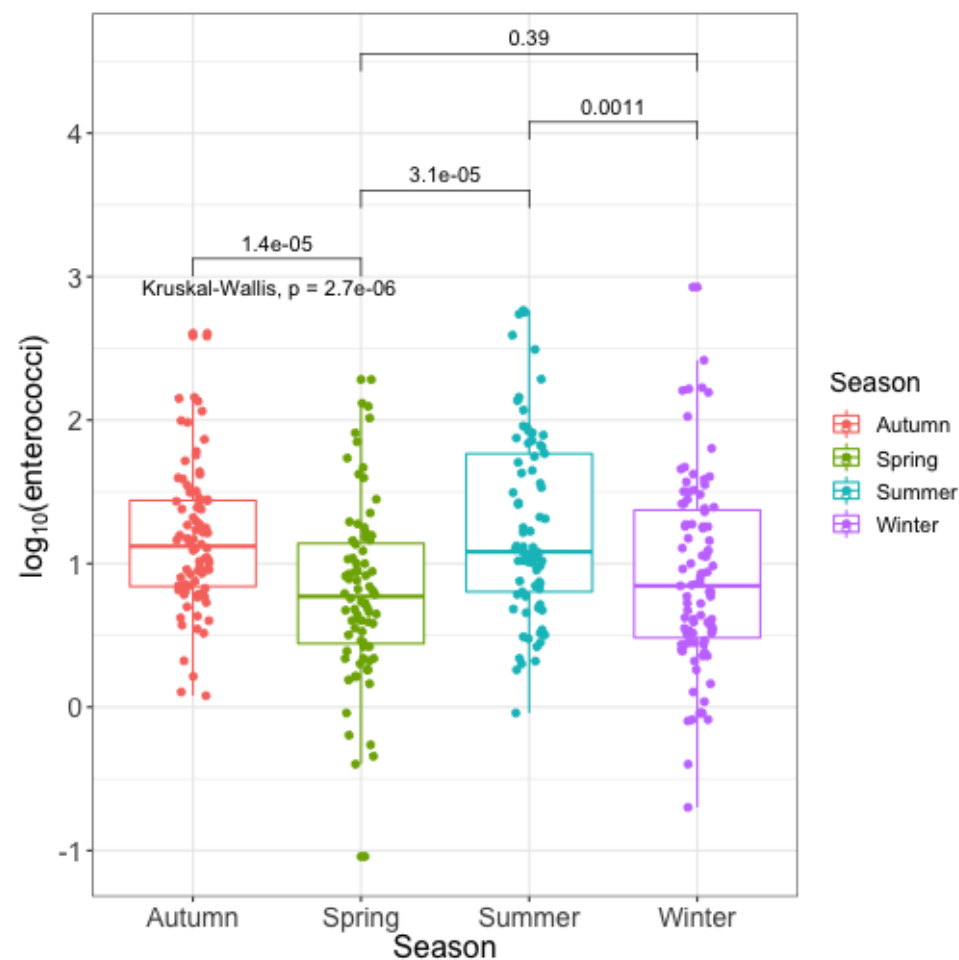
Showing group differences ("Figure 1")

```
library(ggpubr)
plt <- ggplot(
  data=beaches,
  mapping= aes(x = season_name, y = log10(enterococci)
  geom_boxplot() +
  geom_jitter(width=0.1) +
  labs(x = 'Season', y = expression(paste('log'['10']
  theme_viz()
my_comparisons <- list(c('Autumn','Spring'),
                       c('Spring','Summer'),
                       c('Summer','Winter'),
                       c('Spring','Winter'))
plt + stat_compare_means()
```



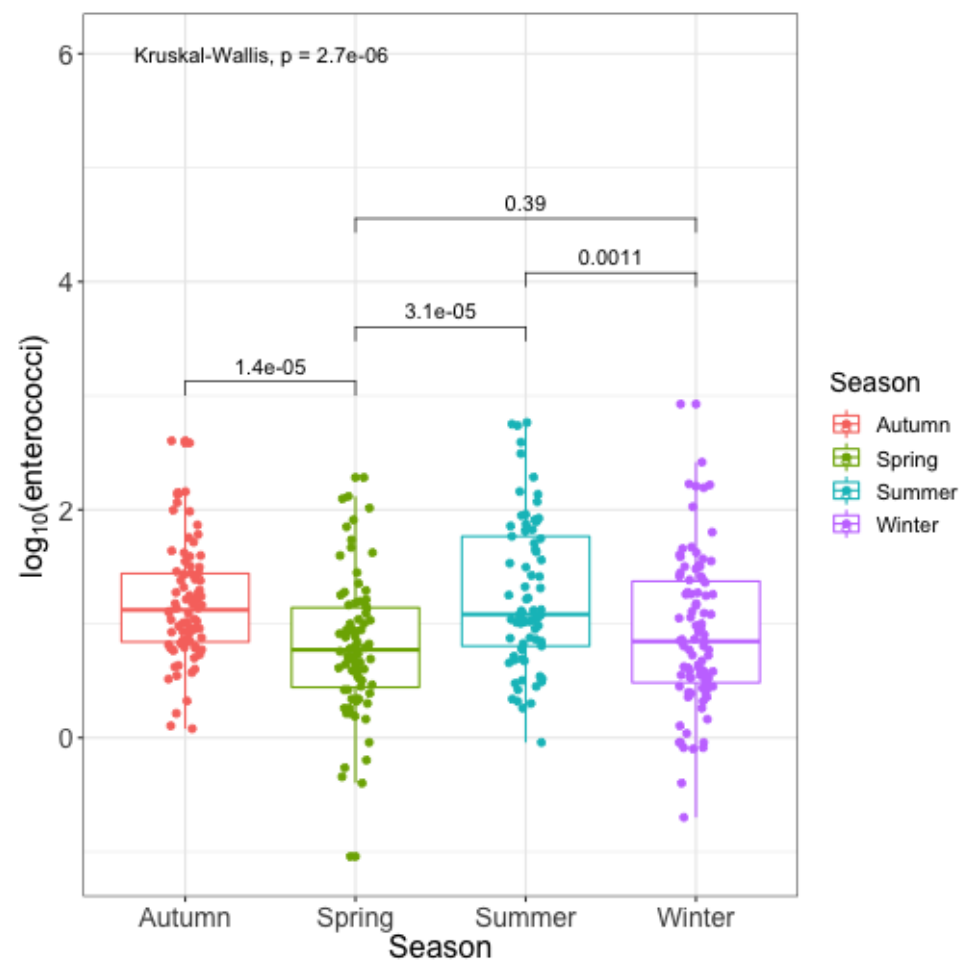
Showing group differences ("Figure 1")

```
library(ggpubr)
plt <- ggplot(
  data=beaches,
  mapping= aes(x = season_name, y = log10(enterococci)
  geom_boxplot() +
  geom_jitter(width=0.1)+
  labs(x = 'Season', y = expression(paste('log'['10
    color='Season')) +
  theme_viz()
my_comparisons <- list(c('Autumn','Spring'),
  c('Spring','Summer'),
  c('Summer','Winter'),
  c('Spring','Winter'))
plt + stat_compare_means() +
  stat_compare_means(comparisons = my_comparisons)
```



Showing group differences ("Figure 1")

```
library(ggpubr)
plt <- ggplot(
  data=beaches,
  mapping= aes(x = season_name, y = log10(enterococci)
  geom_boxplot() +
  geom_jitter(width=0.1)+
  labs(x = 'Season', y = expression(paste('log'['10']
  theme_viz()
my_comparisons <- list(c('Autumn','Spring'),
  c('Spring','Summer'),
  c('Summer','Winter'),
  c('Spring','Winter'))
plt + stat_compare_means(label.y = 6) +
  stat_compare_means(comparisons = my_comparisons)
```



Manipulating data for plotting

We would like to get density plots of all the variables

```
dat_spine <- rio::import('data/Dataset_spine.csv', ch
head(dat_spine)
```

Facets only work by grouping on a variable. Here we have data in several columns

```
#>   Pelvic.incidence Pelvic.tilt Lumbar.lordosis.ar
#> 1      63.02782    22.552586      39.66
#> 2      39.05695    10.060991      25.01
#> 3      68.83202    22.218482      50.09
#> 4      69.29701    24.652878      44.31
#> 5      49.71286     9.652075      28.31
#> 6      40.25020    13.921907      25.12
#>   Pelvic.radius Degree.spondylolisthesis Pelvic.s
#> 1      98.67292             -0.254400    0.744
#> 2     114.40543             4.564259    0.415
#> 3     105.98514            -3.530317    0.474
#> 4     101.86850             11.211523    0.369
#> 5     108.16872             7.918501    0.543
#> 6     130.32787             2.230652    0.789
#>   Thoracic.slope Cervical.tilt Sacrum.angle Scoli
#> 1      14.5386     15.30468   -28.658501
#> 2      17.5323     16.78486   -25.530607
#> 3      17.4861     16.65897   -29.031888
#> 4      12.7074     11.42447   -30.470246
#> 5      15.9546      8.87237   -16.378376
#> 6      12.0036     10.40462    -1.512209
```

Manipulating data for plotting

We would like to get density plots of all the variables.

```
dat_spine %>%  
  tidyr::gather(variable, value, everything())
```

```
#>      variable      value  
#> 1 Pelvic.incidence 63.0278175  
#> 2 Pelvic.incidence 39.05695098  
#> 3 Pelvic.incidence 68.83202098  
#> 4 Pelvic.incidence 69.29700807  
#> 5 Pelvic.incidence 49.71285934  
#> 6 Pelvic.incidence 40.25019968  
#> 7 Pelvic.incidence 53.43292815  
#> 8 Pelvic.incidence 45.36675362  
#> 9 Pelvic.incidence 43.79019026  
#> 10 Pelvic.incidence 36.68635286  
#> 11 Pelvic.incidence 49.70660953  
#> 12 Pelvic.incidence 31.23238734  
#> 13 Pelvic.incidence 48.91555137  
#> 14 Pelvic.incidence 53.5721702  
#> 15 Pelvic.incidence 57.30022656  
#> 16 Pelvic.incidence 44.31890674  
#> 17 Pelvic.incidence 63.83498162  
#> 18 Pelvic.incidence 31.27601184  
#> 19 Pelvic.incidence 38.69791243  
#> 20 Pelvic.incidence 41.72996308
```

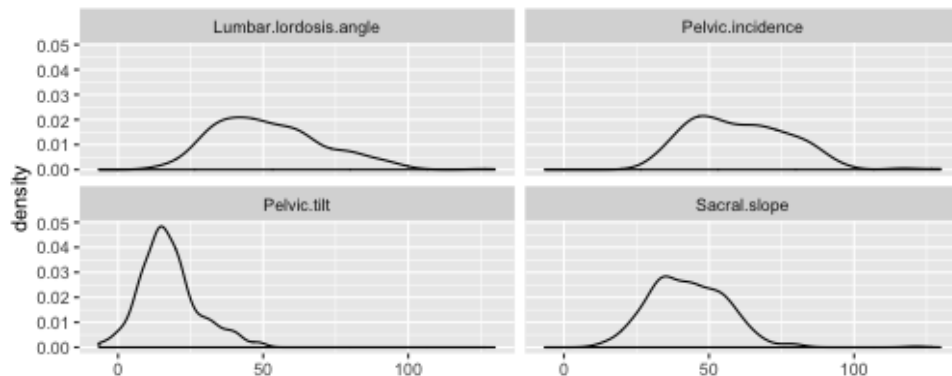
The gather function turns this wide dataset to a long dataset, stacking all the variables on top of each other

Manipulating data for plotting

We would like to get density plots of all the variables.

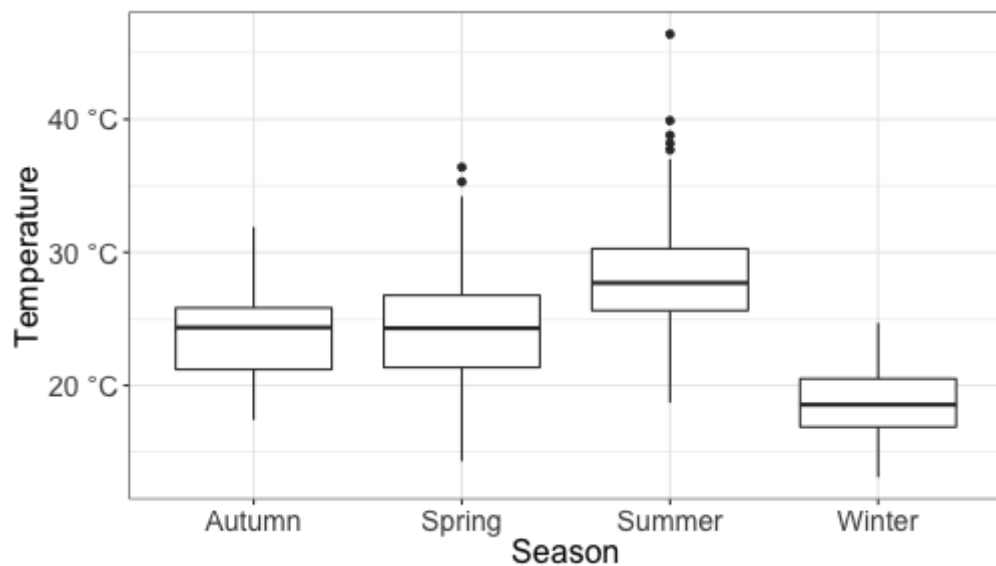
```
dat_spine %>%  
  select(Pelvic.incidence:Sacral.slope) %>%  
  tidyr::gather(variable, value) %>%  
  ggplot(aes(x = value)) +  
  geom_density() +  
  facet_wrap(~variable) +  
  labs(x = '')
```

This is one of my most used tricks for getting faceted plots from wide data



Re-ordering factors

```
beaches %>%  
  ggplot(aes(x = season_name, y = temperature)) +  
  geom_boxplot() +  
  scale_y_continuous(labels = scales::unit_format(unit = "\u00B0C")) +  
  labs(x = 'Season', y = 'Temperature') +  
  theme_bw() +  
  theme(axis.title = element_text(size = 16),  
        axis.text = element_text(size = 14))
```



Re-ordering factors

```
beaches %>%  
  mutate(season_name = fct_relevel(season_name, 'Autumn', 'Winter', 'Spring', 'Summer')) %>%  
  ggplot(aes(x = season_name, y = temperature)) +  
  geom_boxplot() +  
  scale_y_continuous(labels = scales::unit_format(unit = "\u00B0C")) +  
  labs(x = 'Season', y = 'Temperature') +  
  theme_bw() +  
  theme(axis.title = element_text(size = 16),  
        axis.text = element_text(size = 14))
```

