

# Data exploration with **tidyverse**

[Kevin Y. X. Wang](#)

31 July 2017

S0: Prior to lecture

# Preparing for this lecture

- All materials are on Ed and [https://github.com/kevinwang09/2017\\_STAT3914](https://github.com/kevinwang09/2017_STAT3914).
- Please run these codes on your laptop,

```
## Might be a while...  
install.packages(c("ggplot2", "dplyr", "readr", "tidyr", "janitor", "plotly",  
                  "devtools", "learnr", "gapminder"))  
  
library(devtools)  
install_github("kevinwang09/2017_STAT3914", subdir = "learnr3914")
```

- Familiar yourself with the `iris` dataset. Typing `iris` into R console should load this data. Pay attention to its column, row names, summary statistics and structure of each column.

# S1: Necessary of Applied Statistics

# Good statistical discoveries don't fall out from the sky

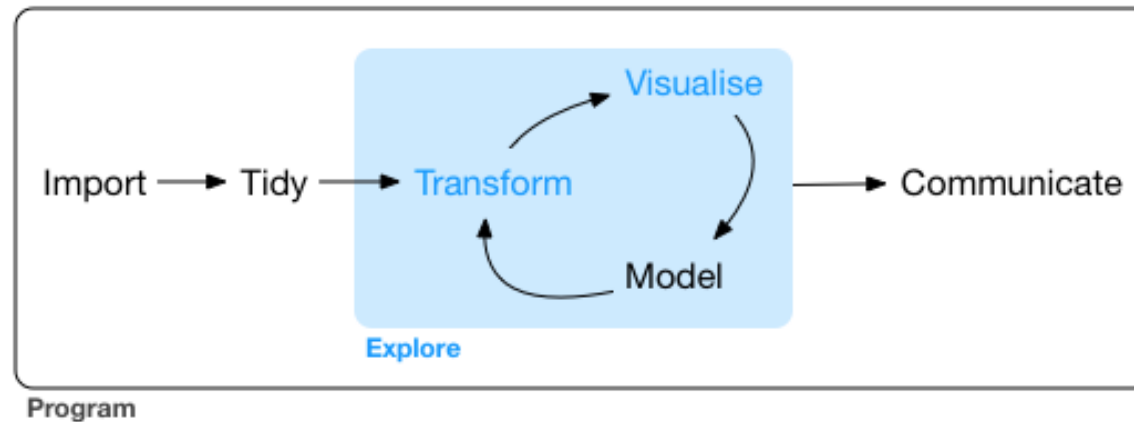
- Statisticians are great at many things:
  1. Understanding data characteristics
  2. Building statistical/mathematical models
  3. Repeat 1 and 2...like...a lot...
  4. Extract insights
- But the mother of all these, i.e. **preparing data** is not trivial. (e.g. STAT2xxx lab exams)

# Let $X$ be the thing I want...

- The real problem is not applying fancy shampoo for your cat. It is getting your cat into the bathtub.



# Hidden side of being a statistician



- Assume we have data
- Assume we have data that can answer our questions
- Assume we have cleaned data
- Assume we interrogated the right aspects of the data using appropriate statistics
- Assume we did everything right, communicate insights with others

# Aim: effectively clean your data (1)

- "Your statistical model is only ever going to be as good as your data quality"  
— Kevin Wang.
- There will be no recipe, there will be a lot of back and forth exploration.
- Computational and visualisation tools.

SepAl....LeNgtH	Sepal.?	Width	petal.Length(*&^	petal.\$%^&Width	species^
6.2		2.8	4.8	1.8	virginica
6.8		2.8	4.8	1.4	versicolor
7.6		3	6.6	2.1	virginica
NA	NA		NA	NA	NA
5.4		3.4	1.7	0.2	setosa

- Corrupted column names, 100% missing column, 100% missing rows, rows with at least 1 missing value.
- Most severe problem: rows with random values.



# Aim: effectively clean your data (2)

- The classical `iris` data is known to be well-separated.
- Running Support Vector Machine (SVM) classification algorithm on the cleaned `iris` data has very low number of misclassifications.
- True `iris` data

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	48	2
virginica	0	2	48

---

## Aim: effectively clean your data (3)

- Not so much when you have corruptions.
- In addition of introduce missing values, I also created non-sense rows in the data, they corrupted classification results.

	setosa	versicolor	virginica
setosa	89	25	27
versicolor	0	47	4
virginica	0	3	46

---

# Summary of this lecture

- Passive learning is not going to work.
- S1: Introduction
- S2: Reading in data using `readr` and `readxl`
- S3: Basic data cleaning using `janitor`
- S4: Clean coding using `magrittr`
- S5: Data filtering using `dplyr`
- S6: Data visualisation using `ggplot2`
- S7: Conclusion

S2: Reading data

# Better read/write data

- `base` R functions are not sufficient for modern uses.
- `readr` functions are superior in data import warnings, column type handling, speed, scalability and consistency.

```
library(readr)
```

# Reading data using (1)

```
dirtyIris = readr::read_csv("dirtyIris.csv")
```

```
## Parsed with column specification:
## cols(
##   Sepal.Length = col_double(),
##   `Sepal.Width` = col_double(),
##   `petal.Length` = col_double(),
##   `petal.Width` = col_double(),
##   `SPECIES` = col_character(),
##   allEmpty = col_character()
## )
```

```
class(dirtyIris) ## `tibble` is a `data.frame` with better formatting.
```

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

- `readxl` and `haven` (for SAS, SPSS etc) packages work similarly.

# Reading data using (2)

dirtyIris

```
## # A tibble: 650 x 6
##   Sepal.Length `Sepal.?` Width `petal.Length(*&` `petal.$#^&Width`
##           <dbl>         <dbl>         <dbl>         <dbl>
## 1      7.7000000         3.8         6.7         2.200000
## 2     -0.1842525         NA         NA         1.099848
## 3      7.2000000         3.6         6.1         2.500000
## 4      6.3000000         2.3         4.4         1.300000
## 5      5.6000000         2.9         3.6         1.300000
## # ... with 645 more rows, and 2 more variables: `SPECIES` <chr>,
## #   allEmpty <chr>
```



- We now proceed to data cleaning on the `dirtyIris` dataset.

# Too trivial? Here is a short homework

Here is a dataset. [Click here.](#)

1. Write 2 sentences about what is a `.gmt` file and who publishes this format?
2. Which packages can read in `.gmt` files?
3. How to download this package?
4. What class is this data once read into R? Is it a `data.frame`?
5. The data contains 50 different gene-sets. What is the size of each gene-set?
6. What is the mostly frequent mentioned 6 genes?



S3: Cleaned data

# What is clean data?

Clean data is a data set that allows you to do statistical modelling without extra processing

1. Good documentation on the entire data.
2. Each column is a **variable**. The name should be informative, and:
  - No bad characters/formatting [@KevinWang009](#)
  - No inconsistent capitalisation or separators (`Cricket_australia` vs `cricket.Australia`)
3. Each row is an **observation**:
  - No bad characters
  - No poorly designed row names (3, 2, 5, ... )
  - No repeated row names (a, a.1, b, b.1, ... )

# Data cleaning in

- Clean data is a well-designed `data.frame`.
- Column type (esp. dates and factors) handling was the primary reason we used `readr` instead of base R when importing data.
- Our goal: clean the `dirtyIris` data to be exactly the same as the original `iris` data.
  - Basic data cleaning using `janitor` package.
  - More advanced data manipulation through `dplyr`.

# : basic data cleaning

- Clean up the bad column names

```
library(janitor)
library(dplyr)
glimpse(dirtyIris)
```

```
## Observations: 650
## Variables: 6
## $ Sepal....LeNgtH <dbl> 7.70000000, -0.18425254, 7.20000000, 6.300000...
## $ Sepal.? Width <dbl> 3.8000000, NA, 3.6000000, 2.3000000, 2.900000...
## $ petal.Length(*&^ <dbl> 6.7000000, NA, 6.1000000, 4.4000000, 3.600000...
## $ petal.$#^&Width <dbl> 2.2000000, 1.0998477, 2.5000000, 1.3000000, 1...
## $ SPECIES^ <chr> "virginica", "setosa", "virginica", "versicol...
## $ allEmpty <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
```

```
## Clean up column names
better = clean_names(dirtyIris)
glimpse(better)
```

```
## Observations: 650
## Variables: 6
## $ sepal_length <dbl> 7.70000000, -0.18425254, 7.20000000, 6.30000000, ...
## $ sepal_width <dbl> 3.8000000, NA, 3.6000000, 2.3000000, 2.9000000, -...
## $ petal_length <dbl> 6.7000000, NA, 6.1000000, 4.4000000, 3.6000000, 0...
## $ petal_width <dbl> 2.2000000, 1.0998477, 2.5000000, 1.3000000, 1.300...
```

# : removal of empty rows and columns

- Purely empty rows/columns are non-informative.

```
## Removing empty rows/columns
```

```
evenBetter = remove_empty_rows(better)
```

```
evenBetter = remove_empty_cols(evenBetter)
```

```
glimpse(evenBetter)
```

```
## Observations: 650
```

```
## Variables: 5
```

```
## $ sepal_length <dbl> 7.70000000, -0.18425254, 7.20000000, 6.30000000, ...
```

```
## $ sepal_width <dbl> 3.8000000, NA, 3.6000000, 2.3000000, 2.9000000, -...
```

```
## $ petal_length <dbl> 6.7000000, NA, 6.1000000, 4.4000000, 3.6000000, 0...
```

```
## $ petal_width <dbl> 2.2000000, 1.0998477, 2.5000000, 1.3000000, 1.300...
```

```
## $ species <chr> "virginica", "setosa", "virginica", "versicolor",...
```

## : removal of rows with NA

- Genuinely missing values should be retained, but in this case, the NA's were added. Only use `na.omit` when you 100% certain of the structure of your data.

```
evenBetterBetter = na.omit(evenBetter)
almostIris = evenBetterBetter
```

```
glimpse(almostIris)
```

```
## Observations: 241
## Variables: 5
## $ sepal_length <dbl> 7.70000000, 7.20000000, 6.30000000, 5.60000000, 6...
## $ sepal_width <dbl> 3.8000000, 3.6000000, 2.3000000, 2.9000000, 2.500...
## $ petal_length <dbl> 6.7000000, 6.1000000, 4.4000000, 3.6000000, 4.900...
## $ petal_width <dbl> 2.2000000, 2.5000000, 1.3000000, 1.3000000, 1.500...
## $ species <chr> "virginica", "virginica", "versicolor", "versicol...
```

```
glimpse(iris)
```

```
## Observations: 150
## Variables: 5
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9,...
## $ Sepal.Width <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1,...
## $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5,...
## $ Petal.Width <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1,...
## $ Species <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa,
```

S4: Clean coding

# Coding complexity increases with the number of brackets

- The "inside out" structure of coding isn't great for human reading.

```
mean(almostIris$sepal_length)
```

```
## [1] 3.602734
```

```
plot(density(almostIris$sepal_length), col = "red", lwd = 2)
```



# Piping: read code from left to right

- We introduce a new notation: " x %>% f " means "f(x)". We call this operation as "x pipe f".
- Compounded operations are possible. Keyboard shortcut is Cmd+shift+M.

```
almostIris$sepal_length %>% mean
```

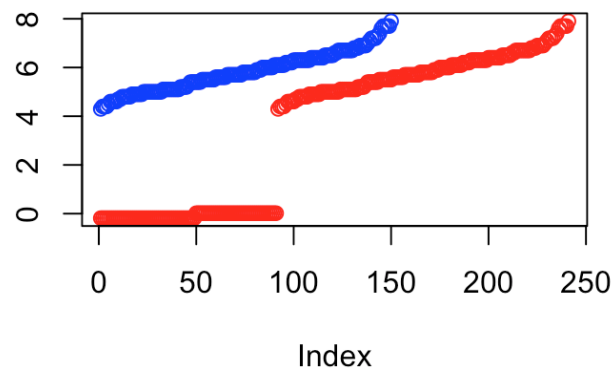
```
## [1] 3.602734
```

```
almostIris$sepal_length %>%  
  density %>%  
  plot(col = "red", lwd = 2)
```

# Using an informative variable (Sepal.Length) in **iris** to guide cleaning

```
almostIris$sepal_length %>%  
  sort %>%  
  plot(col = "red", main = "almostIris is in red, true iris is in blue")  
  
iris$Sepal.Length %>%  
  sort %>%  
  points(col = "blue")
```

**almostIris is in red, true iris is in blue**



S5: **dp1yr**: data subsetting master

# Traditional way of subsetting data in R (1)

- If I want remove all observations with `sepal_length` less than 2:

```
cleanIris = almostIris[almostIris[, "sepal_length"] > 2, ]  
glimpse(cleanIris)
```

```
## Observations: 150  
## Variables: 5  
## $ sepal_length <dbl> 7.7, 7.2, 6.3, 5.6, 6.3, 5.5, 5.0, 6.4, 6.2, 6.7,...  
## $ sepal_width  <dbl> 3.8, 3.6, 2.3, 2.9, 2.5, 2.4, 3.3, 2.7, 3.4, 3.1,...  
## $ petal_length <dbl> 6.7, 6.1, 4.4, 3.6, 4.9, 3.7, 1.4, 5.3, 5.4, 4.4,...  
## $ petal_width  <dbl> 2.2, 2.5, 1.3, 1.3, 1.5, 1.0, 0.2, 1.9, 2.3, 1.4,...  
## $ species      <chr> "virginica", "virginica", "versicolor", "versicol..."
```

- We now have agreement over the size of the two data!
- But this subsetting code is a bit cumbersome!

# Traditional way of subsetting data in R (2)

- Subsetting data in base R might not be the most concise solution.
- Suppose we wish to extract first two rows of column `sepal_length` and `sepal_width` in the `cleanIris` data:

```
## Assuming you know the position of column names.
```

```
## But what if you resample your data?
```

```
cleanIris[1:2, c(1, 2)]
```

```
## Assuming you know the position of column names.
```

```
## Also assuming the first two columns satisfy certain properties.
```

```
cleanIris[1:2, c(T, T, F, F, F)]
```

```
## Much better!
```

```
## What if you can't type out all the column names
```

```
## due to the size of your data?
```

```
cleanIris[1:2, c("sepal_length", "sepal_width")]
```

# Traditional way of subsetting data in R (3)

- Even more complex subsetting: we want to extract rows based on some compounded criteria and select columns based on special keywords.

```
cleanIris[(cleanIris[, "sepal_length"] < 5) &  
          (cleanIris[, "sepal_width"] < 3), c("petal_length", "sepal_length")]
```

```
## # A tibble: 4 x 2  
##   petal_length sepal_length  
##       <dbl>         <dbl>  
## 1         1.3           4.5  
## 2         3.3           4.9  
## 3         1.4           4.4  
## 4         4.5           4.9
```

- (Optional) A pro R user might know about the `subset` function, but it suffers the same problem of not able to have multiple subsetting criteria without predefined variables.

# Subsetting data using

- Think of subsetting rows and columns as two **separate different procedures**:
- **select** columns are operations on variables, and
- **filter** rows are operations on observations
- See [dplyr cheatsheet](#).

```
library(dplyr)

cleanIris %>%
  filter(sepal_length < 5,
         sepal_width < 3) %>%
  select(contains("length"))
```

```
## # A tibble: 4 x 2
##   sepal_length petal_length
##   <dbl>         <dbl>
## 1         4.5         1.3
## 2         4.9         3.3
## 3         4.4         1.4
## 4         4.9         4.5
```

# arrange for ordering rows

```
arrangeCleanIris = cleanIris %>%  
  arrange(sepal_length, sepal_width, petal_length, petal_width)  
  
## The true iris data  
arrangeIris = iris %>%  
  clean_names() %>%  
  arrange(sepal_length, sepal_width, petal_length, petal_width)
```



# Checking if we cleaned the data properly

- We sorted both the processed `dirtyIris` data and the arranged `iris` data.

```
## The `Species` column is character or factor  
all.equal(arrangeCleanIris, arrangeIris)
```

```
## [1] "Incompatible type for column `species`: x character, y factor"
```

```
arrangeIris = arrangeIris %>%  
  mutate(species = as.character(species))
```

```
## Great!  
all.equal(arrangeCleanIris, arrangeIris)
```

```
## [1] TRUE
```

Job done!



# But what about the modelling?

- Cleaned data is one thing, but again, we need to extract insights about the data.
- This can be done via summary statistics, visualisation or running statistical models.
- "Your statistical insights is only going to be as good as the question you ask"  
— Kevin Wang.

## : mutate create new columns

```
iris_mutated = mutate(cleanIris,  
  V1 = sepal_length - sepal_width,  
  V2 = V1 + sepal_width  
)
```

```
iris_mutated
```

```
## # A tibble: 150 x 7
```

```
##   sepal_length sepal_width petal_length petal_width  species    V1    V2  
##         <dbl>      <dbl>      <dbl>      <dbl>      <chr> <dbl> <dbl>  
## 1         7.7         3.8         6.7         2.2  virginica  3.9  7.7  
## 2         7.2         3.6         6.1         2.5  virginica  3.6  7.2  
## 3         6.3         2.3         4.4         1.3  versicolor  4.0  6.3  
## 4         5.6         2.9         3.6         1.3  versicolor  2.7  5.6  
## 5         6.3         2.5         4.9         1.5  versicolor  3.8  6.3
```

```
## # ... with 145 more rows
```

# **group\_by + summarise** will create summary statistics for grouped variables

```
bySpecies = cleanIris %>%  
  group_by(species)
```

```
bySpecies
```

```
## # A tibble: 150 x 5  
## # Groups:   species [3]  
##   sepal_length sepal_width petal_length petal_width species  
##           <dbl>         <dbl>         <dbl>         <dbl>    <chr>  
## 1           7.7           3.8           6.7           2.2  virginica  
## 2           7.2           3.6           6.1           2.5  virginica  
## 3           6.3           2.3           4.4           1.3  versicolor  
## 4           5.6           2.9           3.6           1.3  versicolor  
## 5           6.3           2.5           4.9           1.5  versicolor  
## # ... with 145 more rows
```

# special select functions (advanced)

- `select` only if a column satisfy a certain condition

```
bySpecies %>%  
  summarise_if(is.numeric,  
               funs(m = mean))
```

```
## # A tibble: 3 x 5  
##   species sepal_length_m sepal_width_m petal_length_m petal_width_m  
##   <chr>         <dbl>         <dbl>         <dbl>         <dbl>  
## 1 setosa         5.006         3.428         1.462         0.246  
## 2 versicolor    5.936         2.770         4.260         1.326  
## 3 virginica     6.588         2.974         5.552         2.026
```

```
cleanIris %>%  
  select(starts_with("sepal")) %>%  
  top_n(3, sepal_width)
```

```
## # A tibble: 3 x 2  
##   sepal_length sepal_width  
##   <dbl>         <dbl>  
## 1         5.2         4.1  
## 2         5.5         4.2  
## 3         5.7         4.4
```

# left\_join for merging data

```
flowers = data.frame(species = c("setosa", "versicolor", "virginica"),  
                      comments = c("meh", "kinda_okay", "love_it!"))
```

```
## cleanIris has the priority in this join operation  
iris_comments = left_join(cleanIris, flowers, by = "species")
```

```
## Warning: Column `species` joining character vector and factor, coercing  
## into character vector
```

```
## Randomly sampling 6 rows  
sample_n(iris_comments, 6)
```

```
## # A tibble: 6 x 6  
##   sepal_length sepal_width petal_length petal_width species comments  
##         <dbl>      <dbl>      <dbl>      <dbl>      <chr>    <fctr>  
## 1         6.3         2.9         5.6         1.8  virginica  love_it!  
## 2         6.0         2.9         4.5         1.5 versicolor kinda_okay  
## 3         6.8         3.2         5.9         2.3  virginica  love_it!  
## 4         4.6         3.1         1.5         0.2    setosa      meh
```

S6: **ggplot2**: the best visualisation  
package



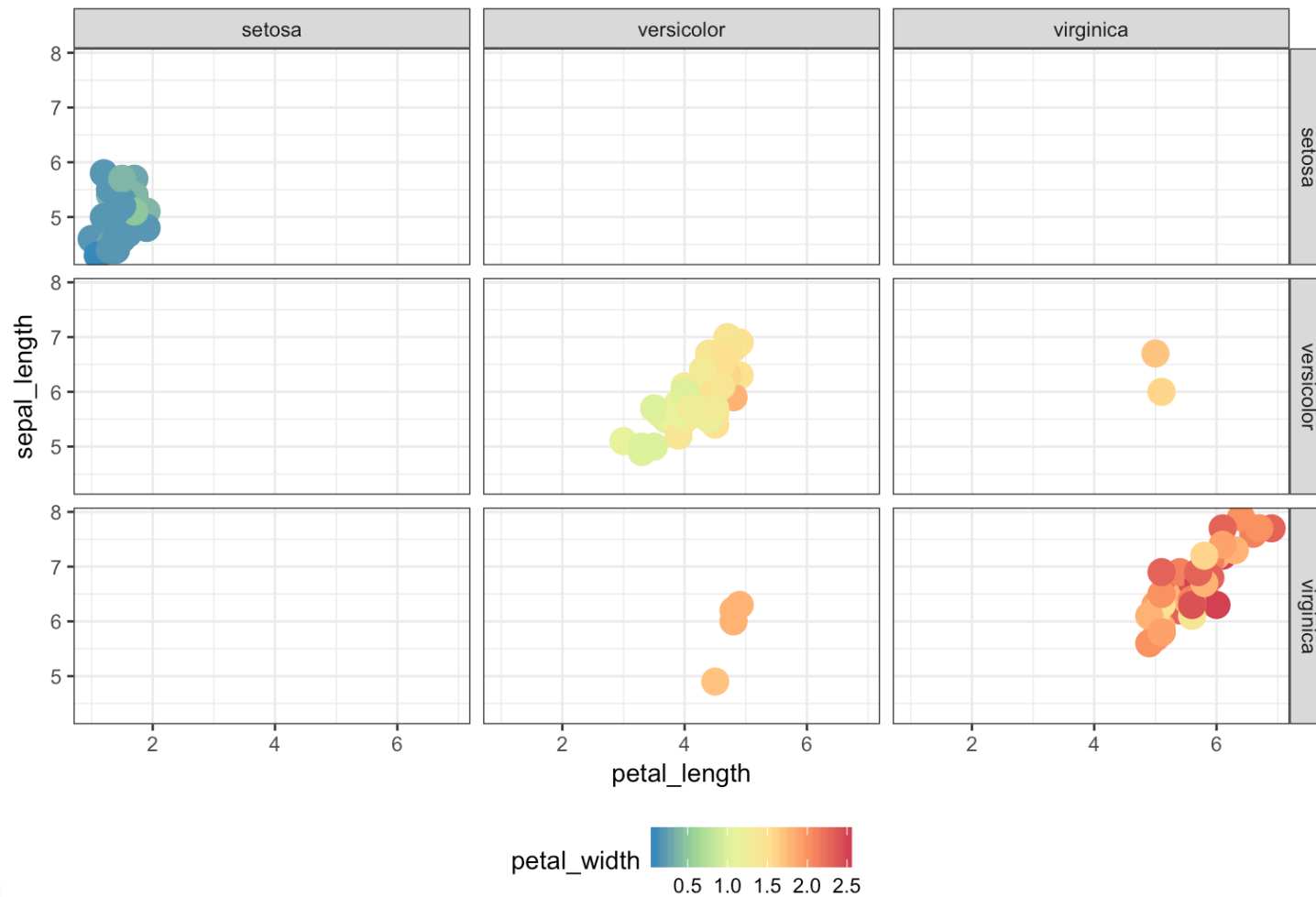
# Why do we visualise? (1)

- `datasaurus`: all statistics describe the data in some limited ways.
- Plots usually give more dimensions to our analysis.
- Suppose in our `cleanIris` data, we will use `sepal_length` and `petal_length` as SVM predictors for the classes of iris flowers.

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	48	2
virginica	0	4	46

---

# Why do we visualise? (2)



## : the philosophy

- Di Cook - the real reason that you should use `ggplot2` is that, its design will force you to use a certain **grammar** when producing a plot.
- $\frac{1}{n} \sum_{i=1}^n X_i$  is a transformation of random variables, i.e., a statistic which provides insights into a data.
- Similarly, `ggplot` is also a statistic, because we take components of the data and presented it in an informative way.
- Publishing quality, rigourous syntax and design, flexible customisations, facetting.

# tutorial sheet

- If you managed to install all packages successfully, you should be able to run the following to get an interactive tutorial sheet.

```
library(learnr3914)  
learnnggplot2()
```

- Otherwise, please download and compile the "ggplot2\_basic\_tutorial.Rmd" from Ed or [here](#)
- If all fails, try [https://gauss17gon.shinyapps.io/ggplot2\\_basic\\_tutorial](https://gauss17gon.shinyapps.io/ggplot2_basic_tutorial) or [https://garhtarr.shinyapps.io/ggplot2\\_basic\\_tutorial](https://garhtarr.shinyapps.io/ggplot2_basic_tutorial)

# S7: Conclusion

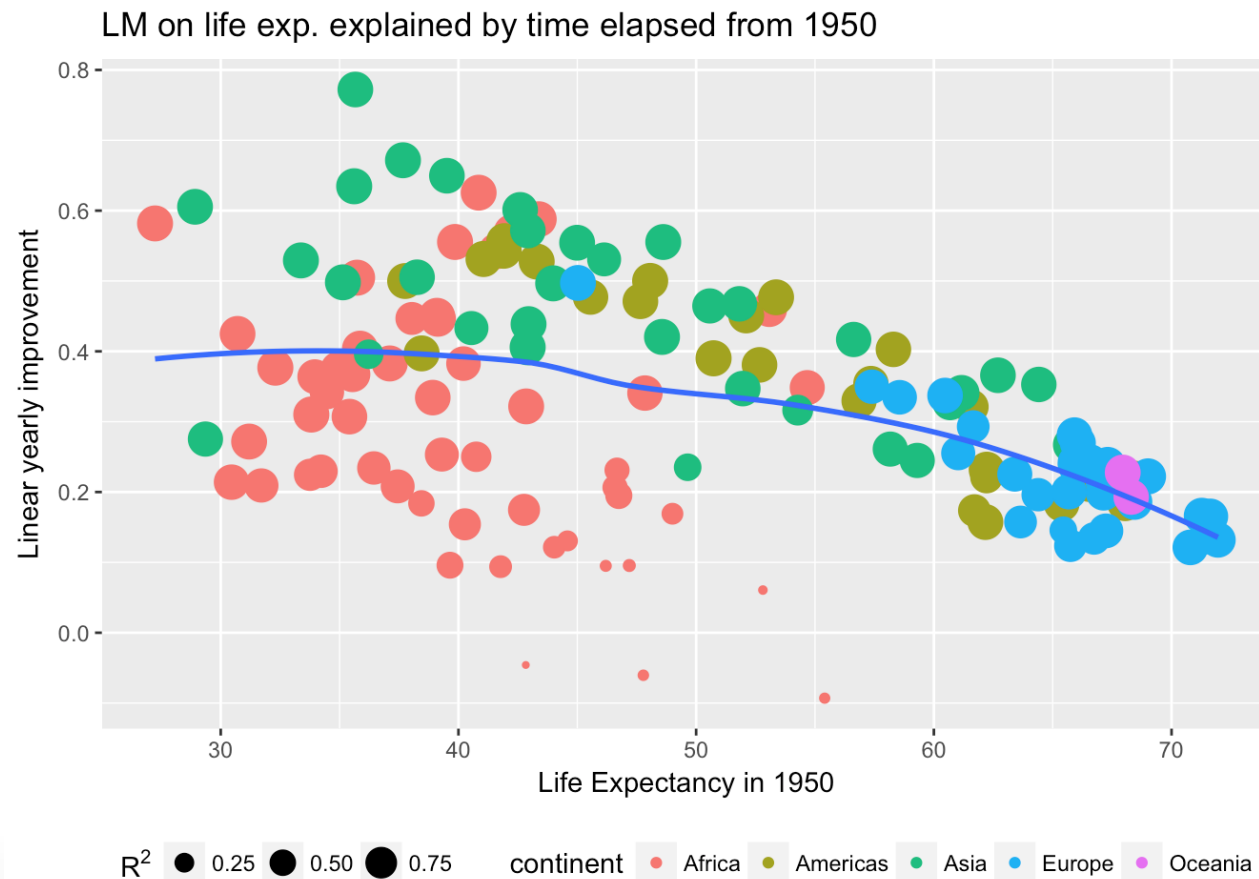
# tidy data, coding, modelling and reporting

- **tidyverse** is a collection of 20+ packages built on the philosophy of being organised for the purpose of collaboration.
- These functions:
  - Well designed programming and data science solutions.
  - They will always throw errors at you if you don't have a thorough understanding of your data.
  - Capable for functional programming.



# Peek at the

<http://edinbr.org/edinbr/2016/05/11/may-Hadley-Update2-PostingTalk.html>



# Interactive plotting from ggplot

```
library(plotly)  
ggplotly(p2)
```



# Advice in the future

- Use RStudio + RMarkdown to document your codes.
- Learn some computational tools. They are not statistics, but not learning them could inhibit your career aspects.
- Find "cool" components and adapt those into your work routine. (Hint: start with [all RStudio cheatsheets](#) and build up gradually.)
- Take time to re-analyse an old dataset.
- Learn core functions and vignette.
- Don't forget the theories and interpretations! This is a course about statistics after all, not Cranking-Out-Numbers-Less-Than-0.05-And-Reject-Null-Hypothesis-101.

# Session Info and References

- Dr. Garth Tarr
- [tidyverse.org](https://tidyverse.org)
- [github.com/sfirke/janitor](https://github.com/sfirke/janitor)
- [gapminder.org](https://gapminder.org)
- [rstudio.com](https://rstudio.com)