Visual exploratory (genomic) data analysis in R

Using ggplot2 and the tidyverse for visual EDA

Kaspar Märtens

05/12/2017

Why you should first visualise your data before launching an ML algorithm

Consider the following data (N=20, p=2)

x	y
1.593	-1.209
-0.195	1.991
-1.388	1.440
-1.794	-0.884
1.676	-1.091
0.598	1.909
1.606	-1.192
1.880	-0.681
-1.063	-1.694
-1.377	-1.450

	x	y
11	1.851	0.757
12	0.546	1.924
13	0.891	1.791
14	-0.771	-1.845
15	-1.493	1.331
16	0.249	-1.984
17	-2.000	0.029
18	-0.404	-1.959
19	1.997	-0.102
20	-1.458	1.369

```
mean(x)
```

```
## [1] 0.04724152
```

```
mean(y)
```

```
## [1] -0.07760144
```

```
cor(x, y)
```

```
## [1] -0.0304444
```

```
mean(x)
```

```
## [1] 0.04724152
```

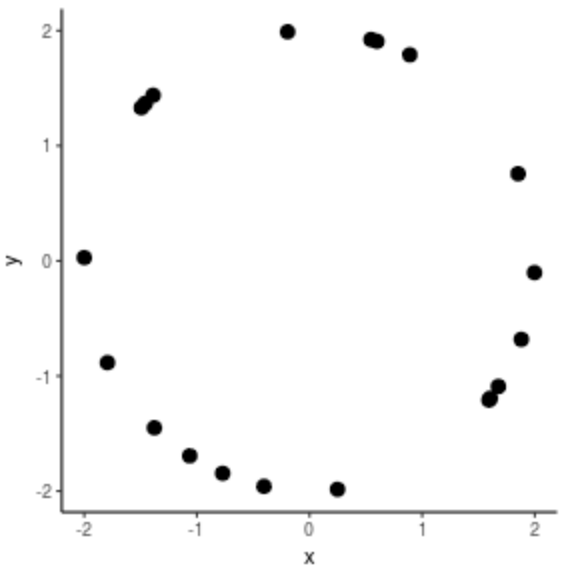
```
mean(y)
```

```
## [1] -0.07760144
```

```
cor(x, y)
```

```
## [1] -0.0304444
```

Scatterplot immediately reveals the structure



Why you should first visualise your data before launching an ML algorithm

Before launching your ML model, you should have a rough idea about:

Why you should first visualise your data before launching an ML algorithm

Before launching your ML model, you should have a rough idea about:

- Data quality
  - Are there any batch effects?
  - Are there possible sample mislabellings?
  - Outliers?
  - Distributions - should you apply a transformation?

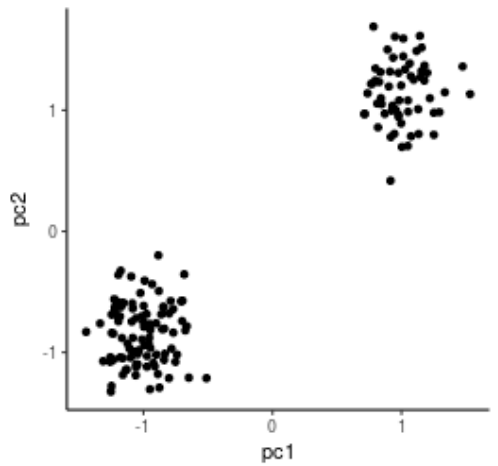
Why you should first visualise your data before launching an ML algorithm

Before launching your ML model, you should have a rough idea about:

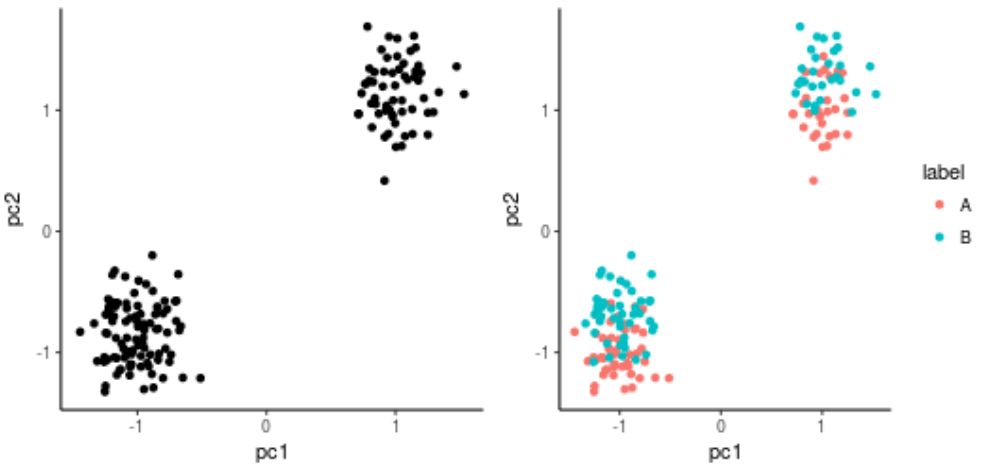
- Data quality
  - Are there any batch effects?
  - Are there possible sample mislabellings?
  - Outliers?
  - Distributions - should you apply a transformation?

- Expected model fit:
  - Is there strong (linear) signal?
  - Your expected prediction accuracy (will it be close to random or close to 100%)

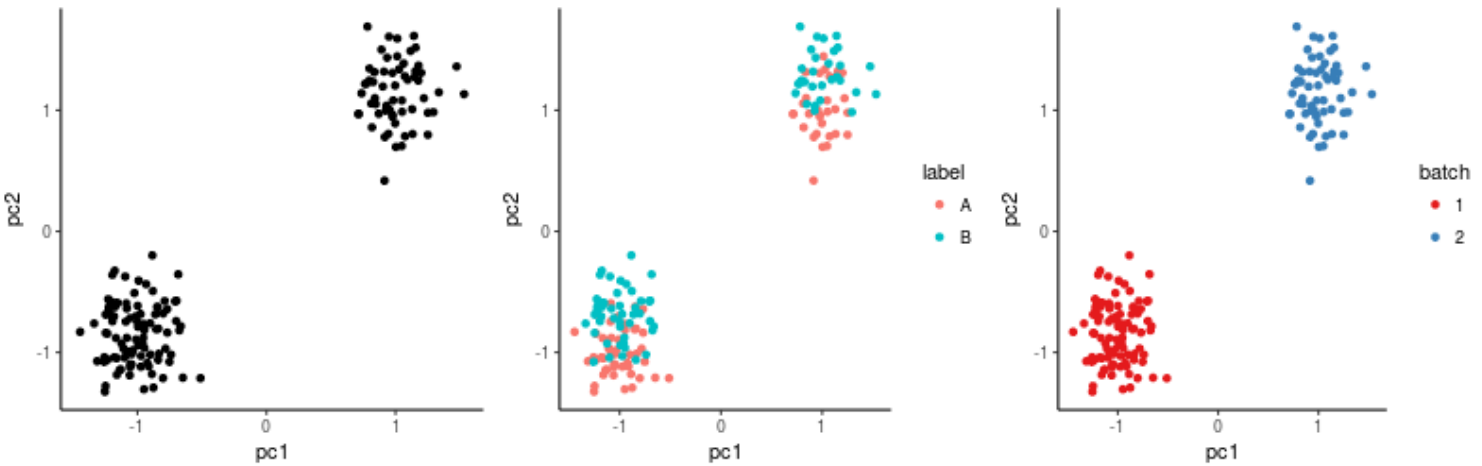
Why you should first visualise your data before launching an ML algorithm



Why you should first visualise your data before launching an ML algorithm



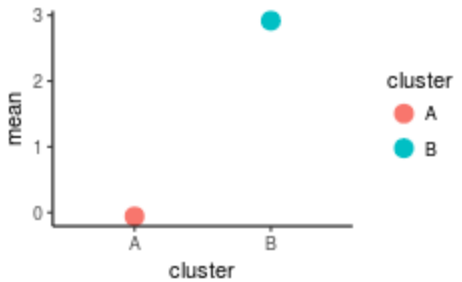
Why you should first visualise your data before launching an ML algorithm



Are summary statistics good enough?

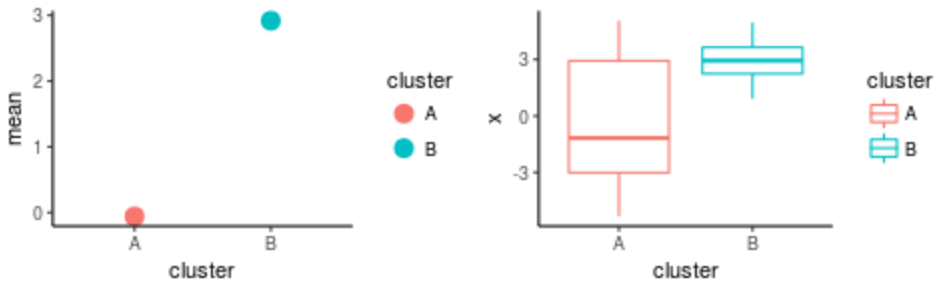
Are summary statistics good enough?

Do groups A and B have a significantly different measurements x?



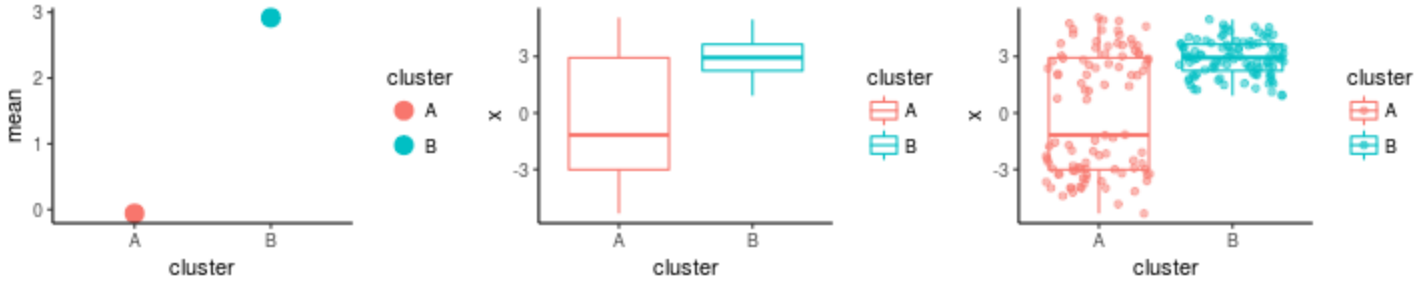
Are summary statistics good enough?

Do groups A and B have a significantly different measurements x?



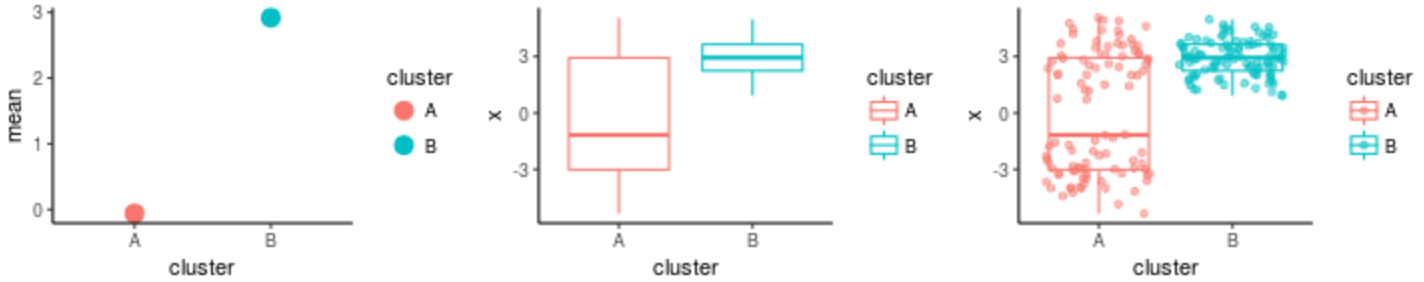
Are summary statistics good enough?

Do groups A and B have a significantly different measurements x?



Are summary statistics good enough?

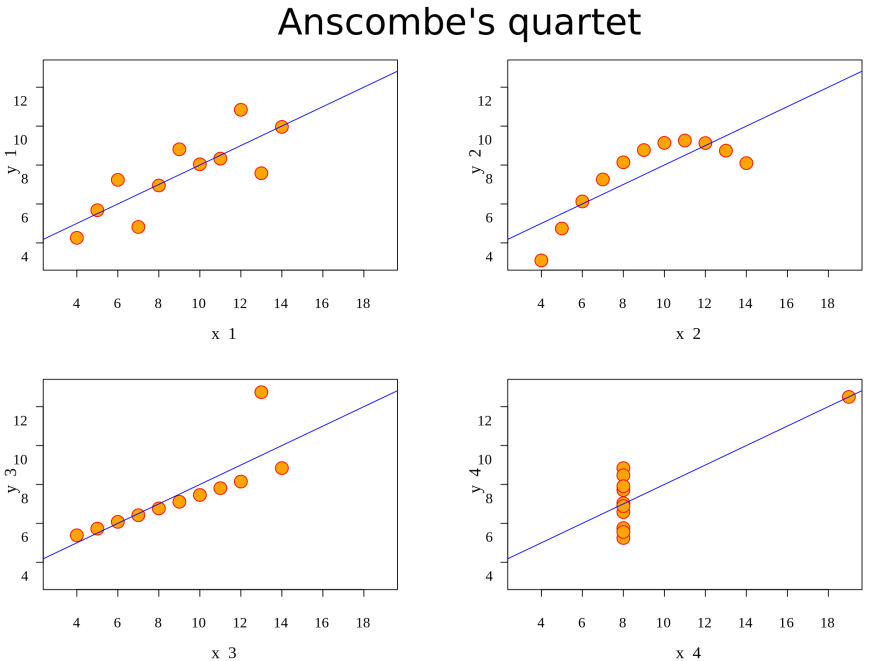
Do groups A and B have a significantly different measurements x?



group means < boxplot < data

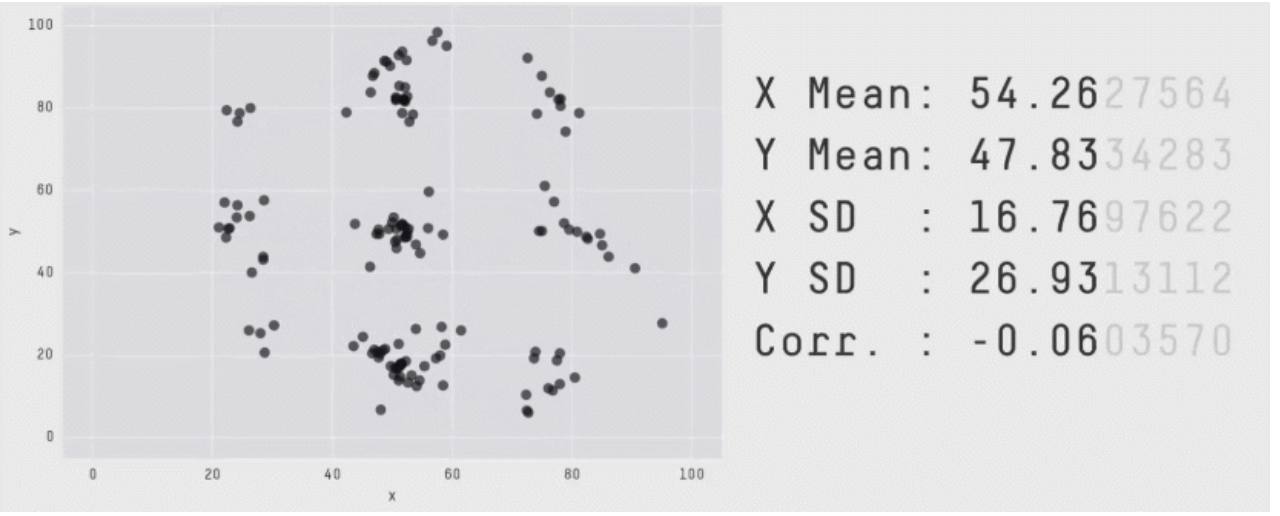
Are summary statistics good enough?

Is there a significant association between two continuous variables x and y?



Source: https://en.wikipedia.org/wiki/Anscombe%27s_quartet

Are summary statistics good enough?



Source: https://www.autodeskresearch.com/publications/samestats

Why visualise?

> "The simple graph has brought more information to the data analyst's mind than any other device." --- John Tukey

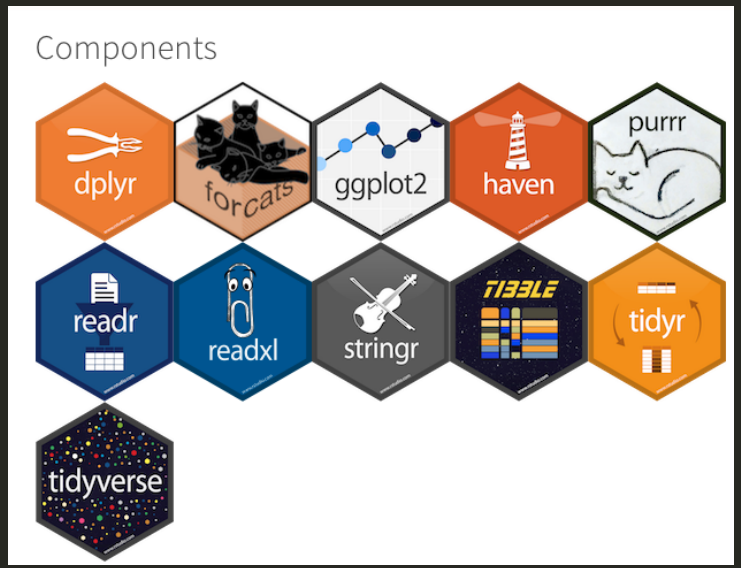
Why R?

Thanks to the tidyverse (https://www.tidyverse.org/) probably the best framework for doing data science.

Tidyverse

A popular collection of R packages for data import, manipulation, exploration and visualization.

- Unified and consistent design and API (unlike base R)
- Intuitive underlying philosophy
- Easy-to-learn
- Easy to get help: large community on stackoverflow, twitter #rstats etc



Outline:

Using TCGA breast cancer data as a case study,

1. Data visualisation using ggplot2
2. Tools for data manipulation and transformation
3. Visual exploratory analysis (combining 1. and 2.)

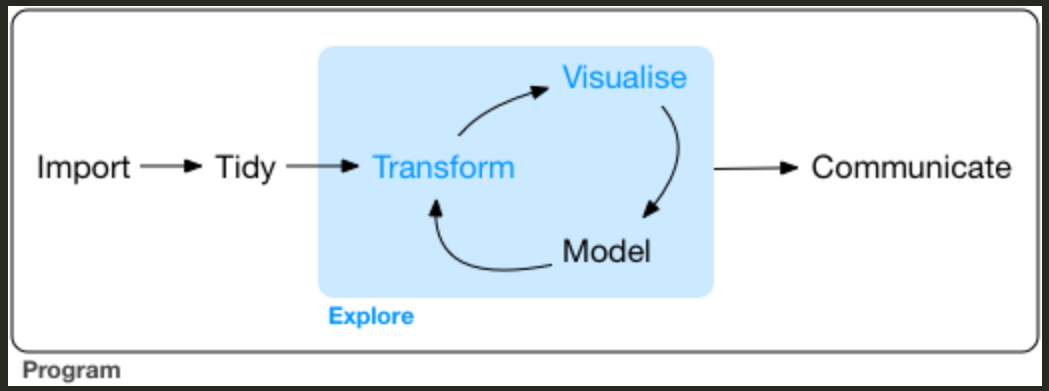
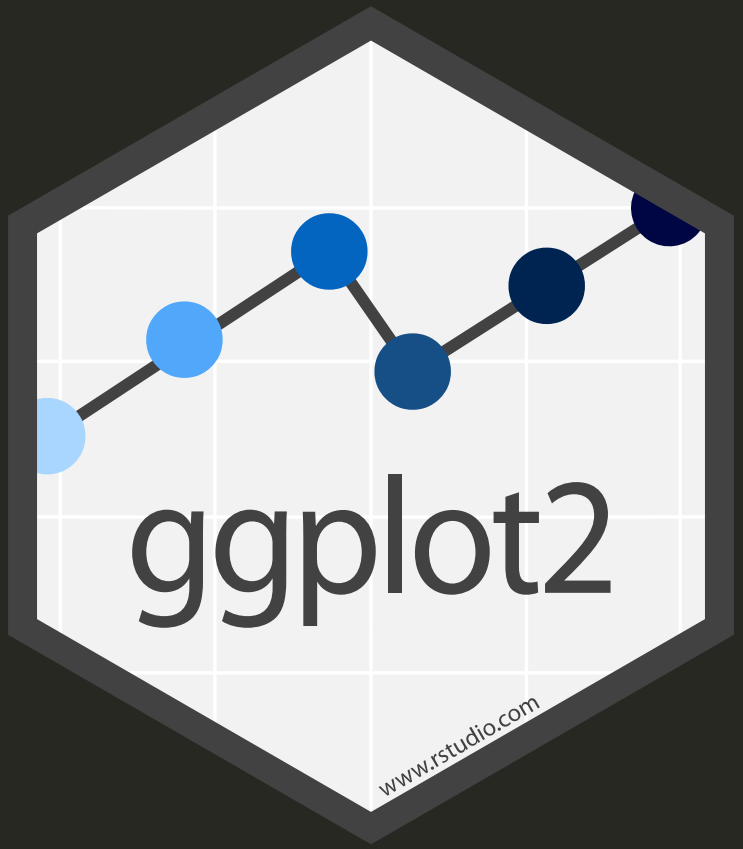


Fig source: http://r4ds.had.co.nz

ggplot2

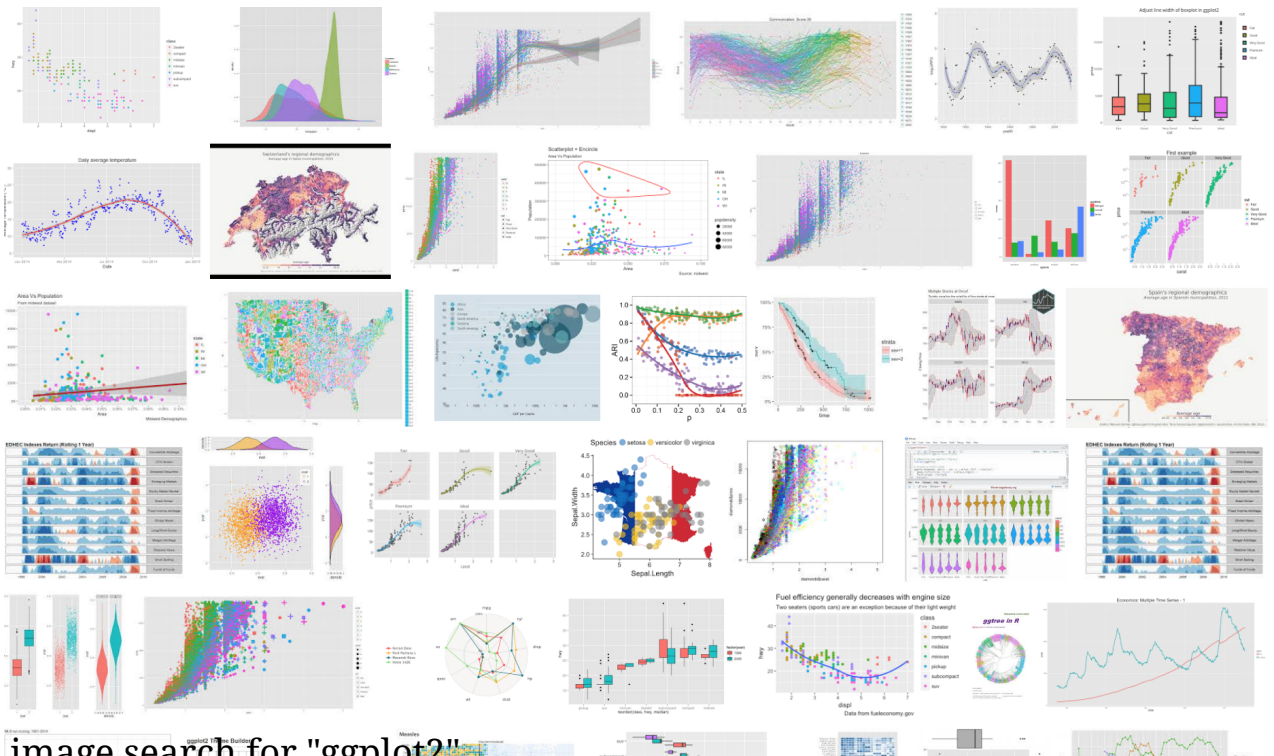


Hadley Wickham (http://hadley.nz/):

- author of many widely used R packages, including ggplot2
- introduced the concept of tidy data and key contributor to the tidyverse
- author of many excellent books, including
  - R for Data Science http://r4ds.had.co.nz/
  - Advanced R http://adv-r.had.co.nz/
  - etc

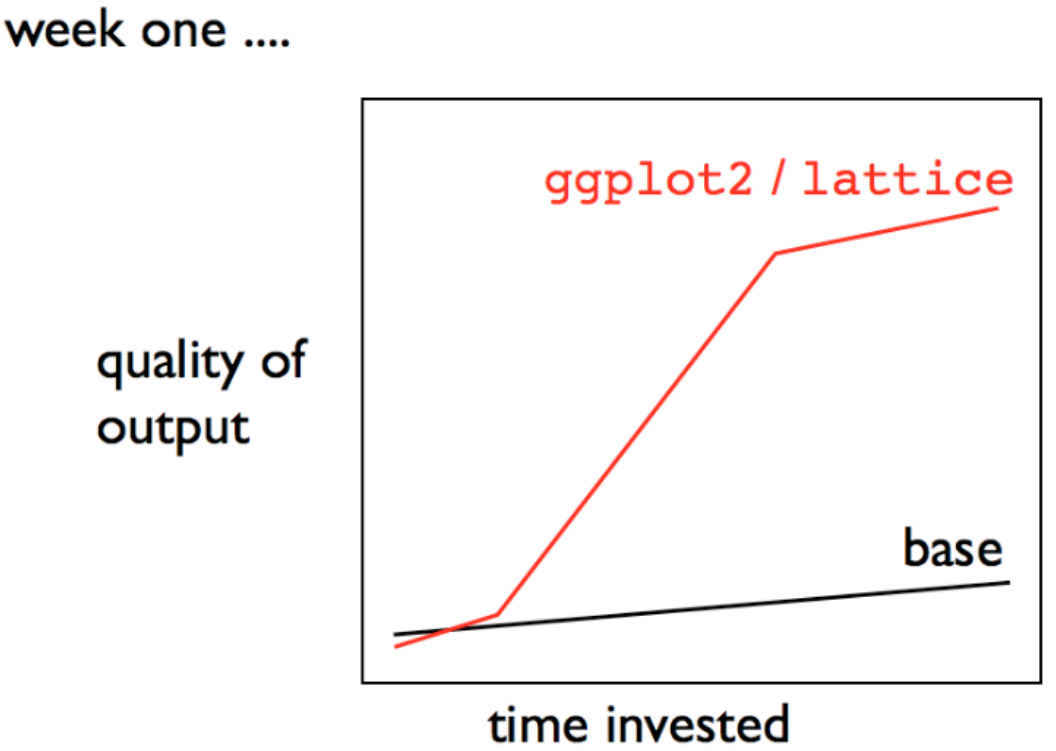
ggplot2

- Quickly iterate over a variety of plots (only making minimal changes to the code).
- Create publication-quality plots with minimal tweaking.



google image search for "ggplot2"

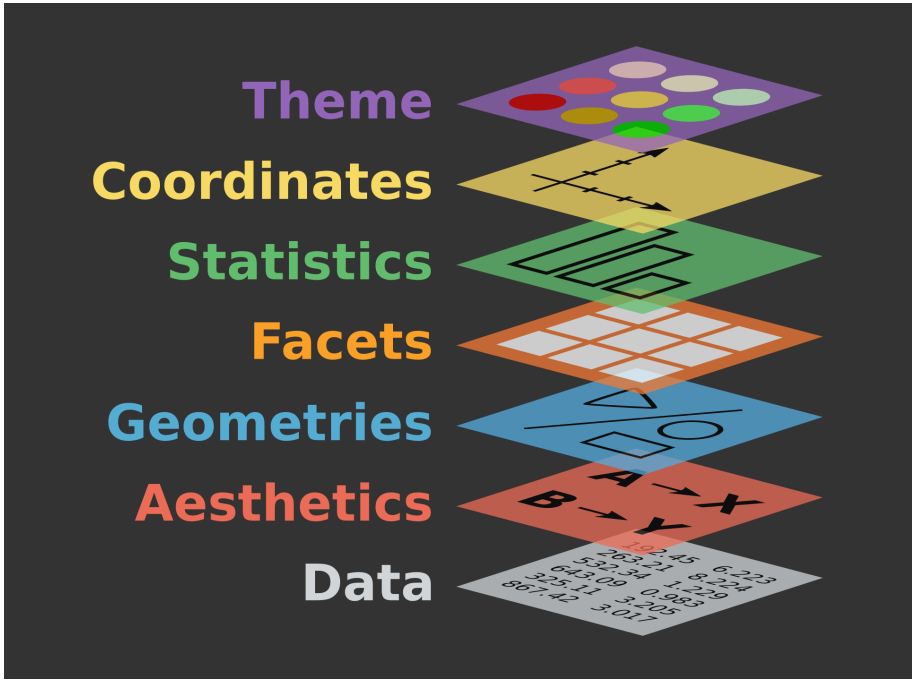
ggplot2 learning curve



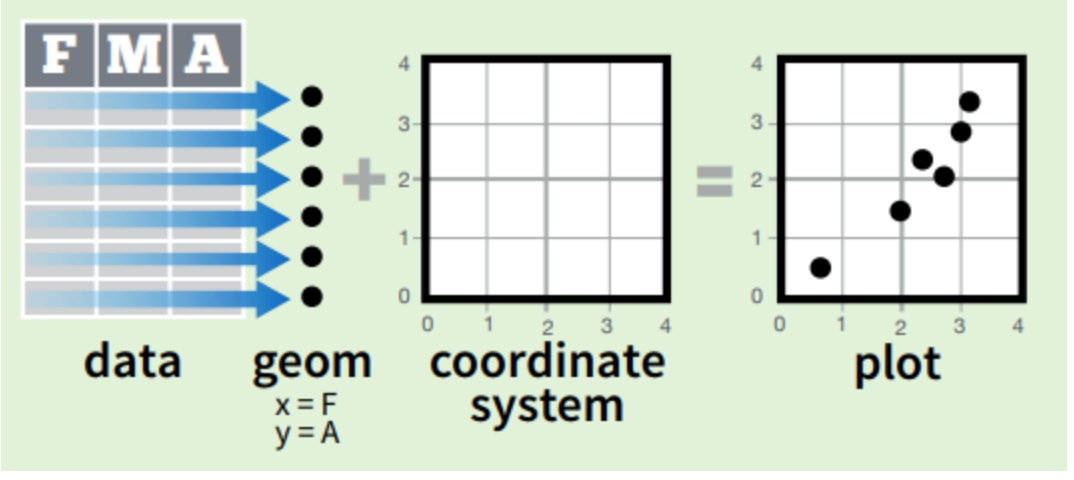
Source: https://github.com/jennybc/ggplot2-tutorial * figure is totally fabricated but, I claim, still true

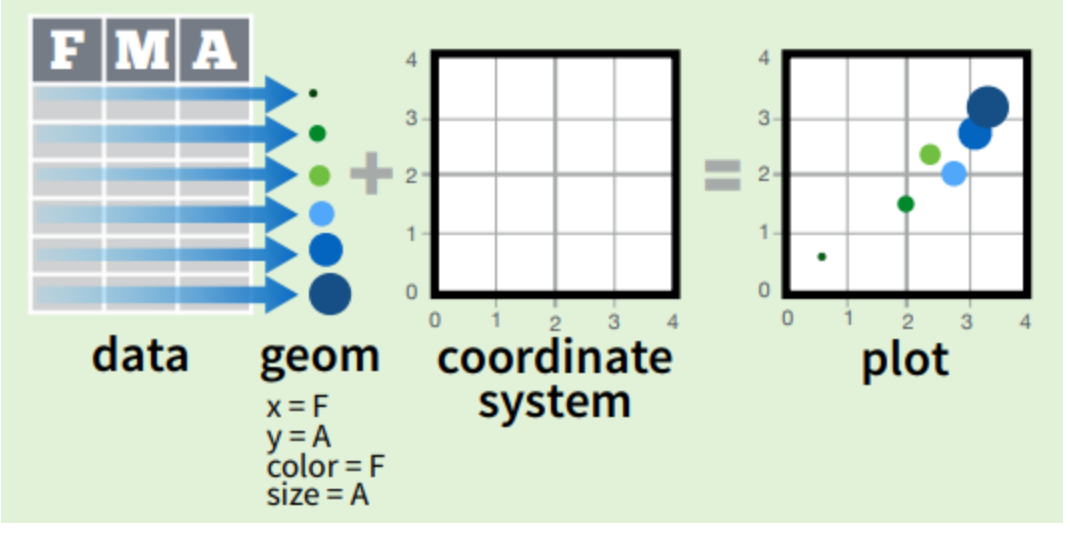
ggplot2

Based on the Grammar of Graphics (book by Leland Wilkinson, 1999/2005) -- idea that every graph can be built from the same components. Results in a very flexible framework for plotting.



ggplot2


F M A
data
geom
x = F
y = A
coordinate
system
plot


F M A
data
geom
x = F
y = A
color = F
size = A
coordinate
system
plot

TCGA breast cancer data set

- Phenotypes:
  - age
  - ER status (estrogen-receptor-positive or negative)
  - PAM50 cancer subtype
- Gene expression data:
  - `log(x+1)` transformed expression for all genes
  - PCA has been applied to expression data => PC1 and PC2 provided

Prepared data frame:

	id	age	ER_status	PAM50	pc1	pc2
800	TCGA-E2-A150	48	Negative	Basal	-1.8488457	-0.0390611
180	TCGA-A8-A07E	81	Positive	LumA	0.3132872	-0.7657378
509	TCGA-B6-A40B	76	Positive	NA	0.7771889	-0.2705657
761	TCGA-D8-A27I	58	Positive	LumA	0.4582089	-1.6383442

TCGA breast cancer data set

- Phenotypes:
  - age
  - ER status (estrogen-receptor-positive or negative)
  - PAM50 cancer subtype
- Gene expression data:
  - `log(x+1)` transformed expression for all genes
  - PCA has been applied to expression data => PC1 and PC2 provided

Prepared data frame:

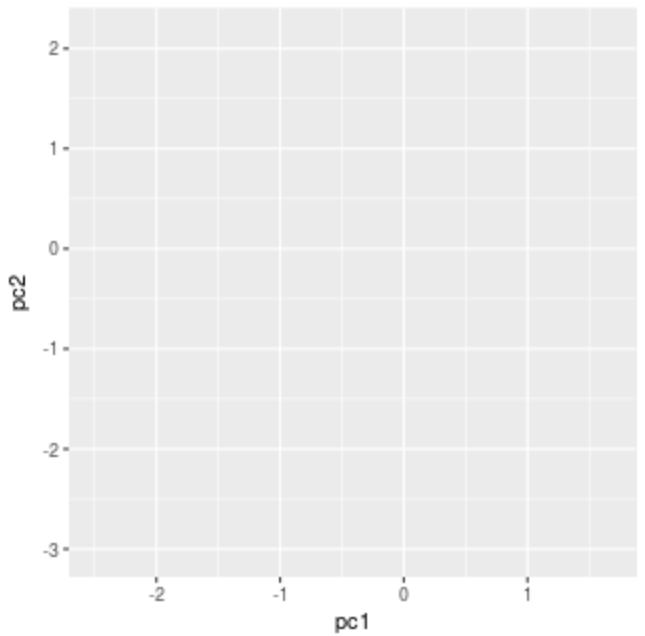
	id	age	ER_status	PAM50	pc1	pc2
800	TCGA-E2-A150	48	Negative	Basal	-1.8488457	-0.0390611
180	TCGA-A8-A07E	81	Positive	LumA	0.3132872	-0.7657378
509	TCGA-B6-A40B	76	Positive	NA	0.7771889	-0.2705657
761	TCGA-D8-A27I	58	Positive	LumA	0.4582089	-1.6383442

Question: Do different subtypes have different expression profiles?

ggplot2 call

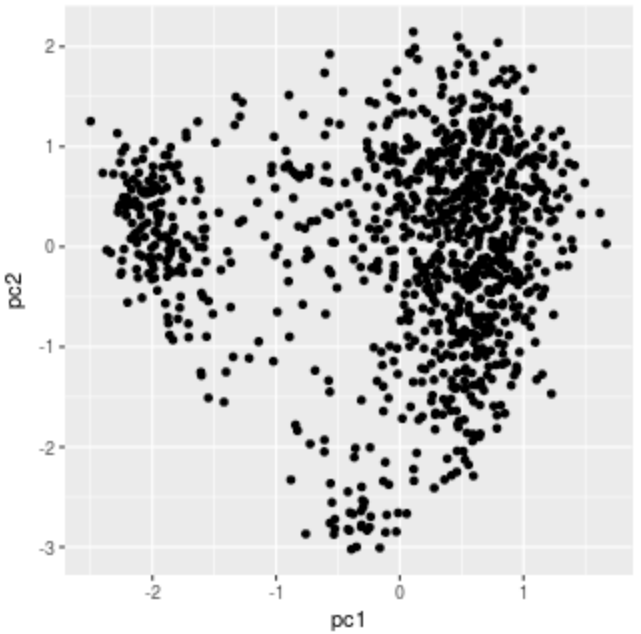
```
library(ggplot2)

ggplot(df, aes(x = pc1, y = pc2))
```



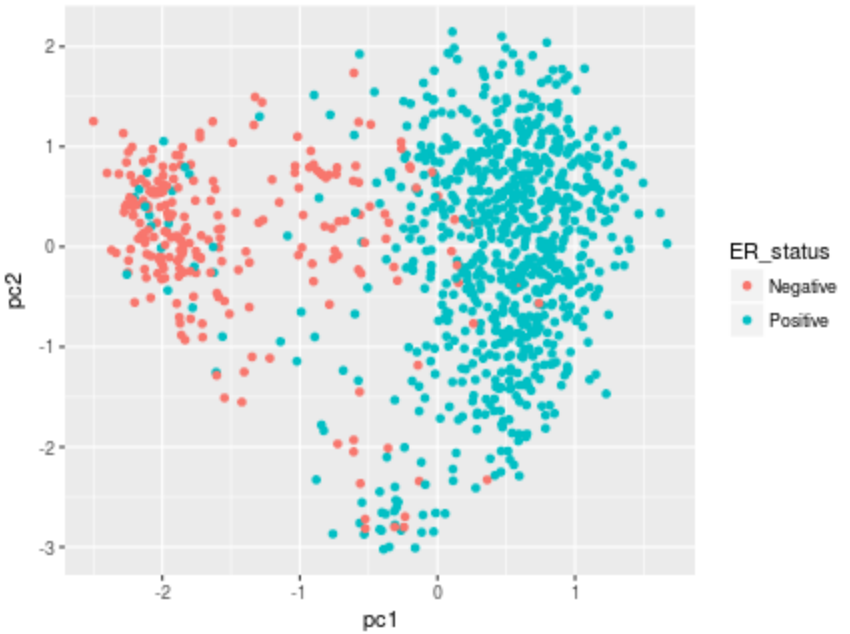
ggplot2 call: scatterplot

```
ggplot(df, aes(x = pc1, y = pc2)) +
  geom_point()
```



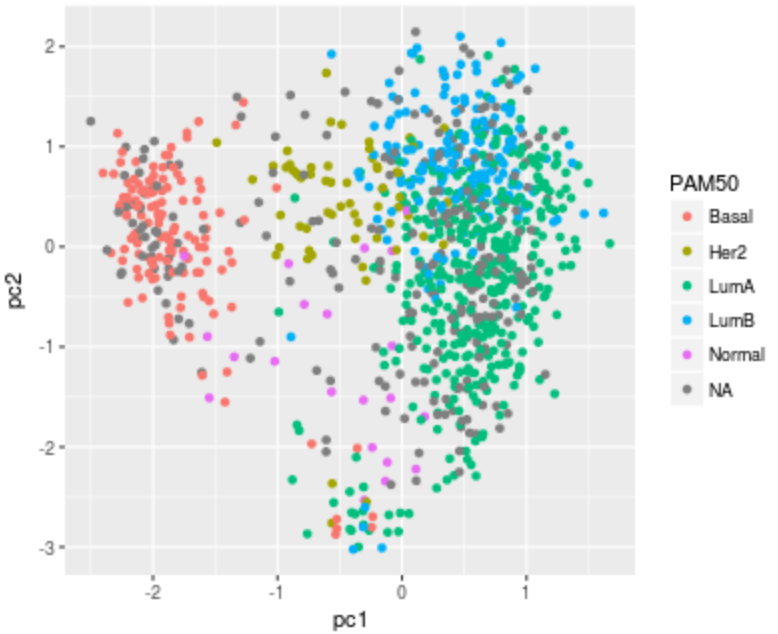
scatterplot, colour by ER_status

```
ggplot(df, aes(x = pc1, y = pc2, col = ER_status)) +
  geom_point()
```



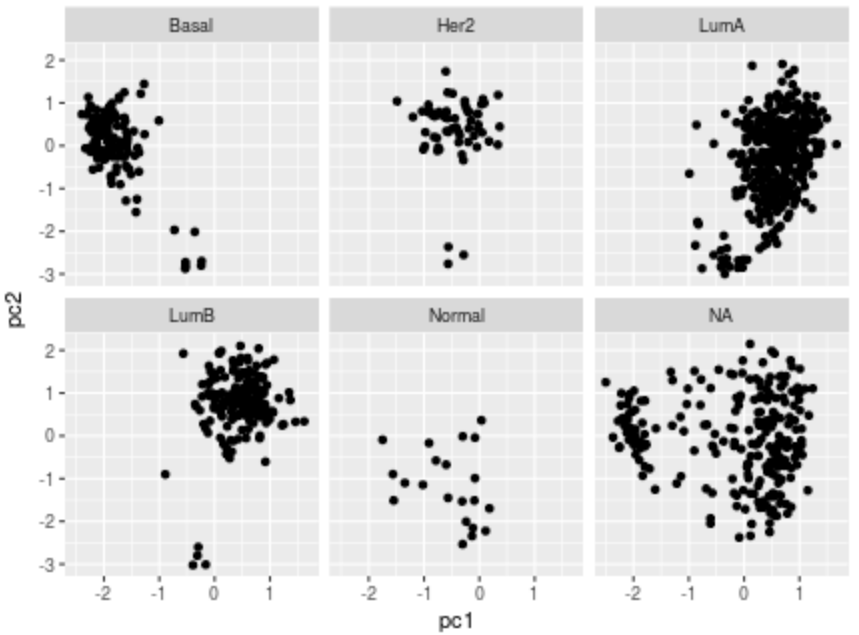
scatterplot, colour by PAM50 subtype

```
ggplot(df, aes(x = pc1, y = pc2, col = PAM50)) +
  geom_point()
```



facet by subtype

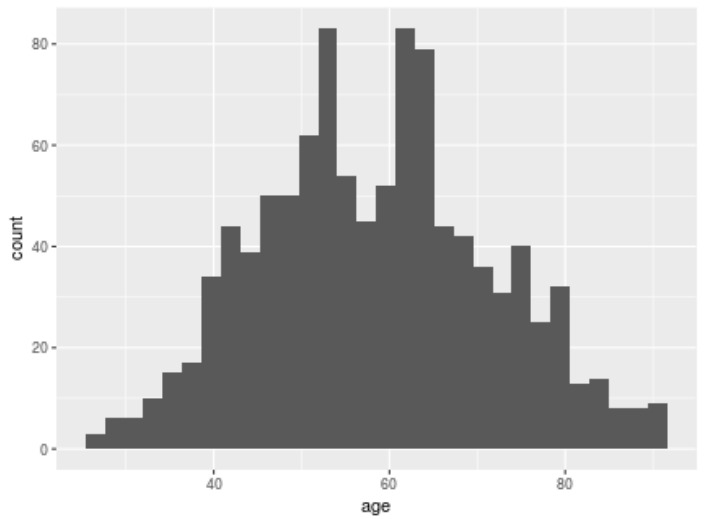
```
ggplot(df, aes(x = pc1, y = pc2)) +
  geom_point() +
  facet_wrap(~ PAM50)
```



Lets explore the age distribution

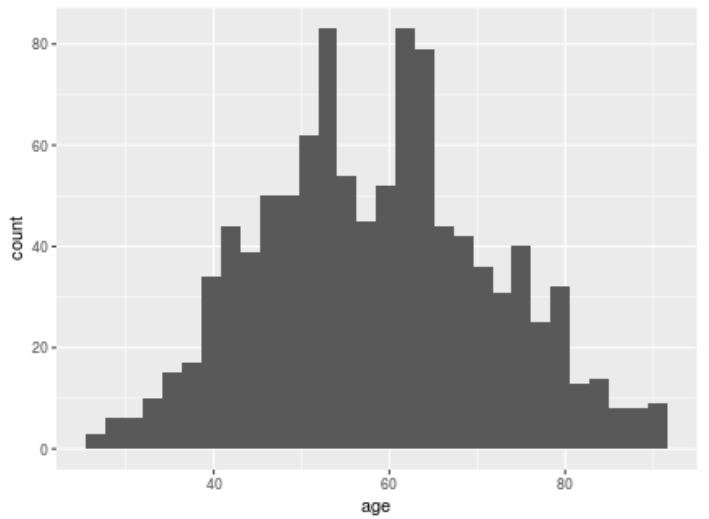
Lets explore the age distribution

```
ggplot(df, aes(x = age)) +
  geom_histogram()
```

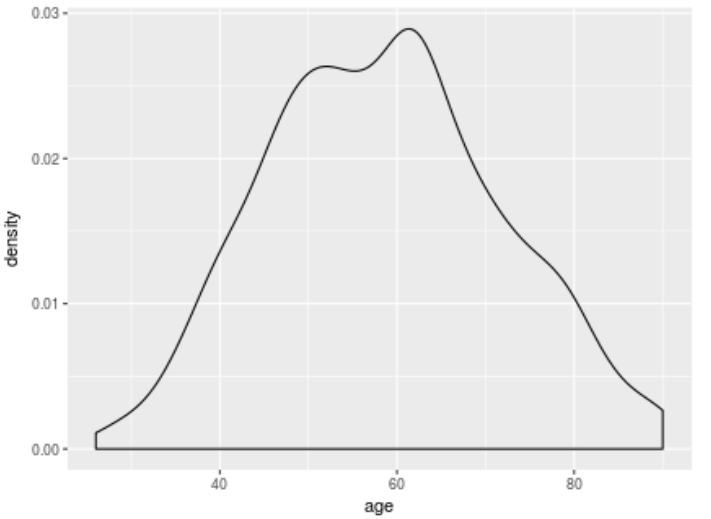


Lets explore the age distribution

```
ggplot(df, aes(x = age)) +
  geom_histogram()
```



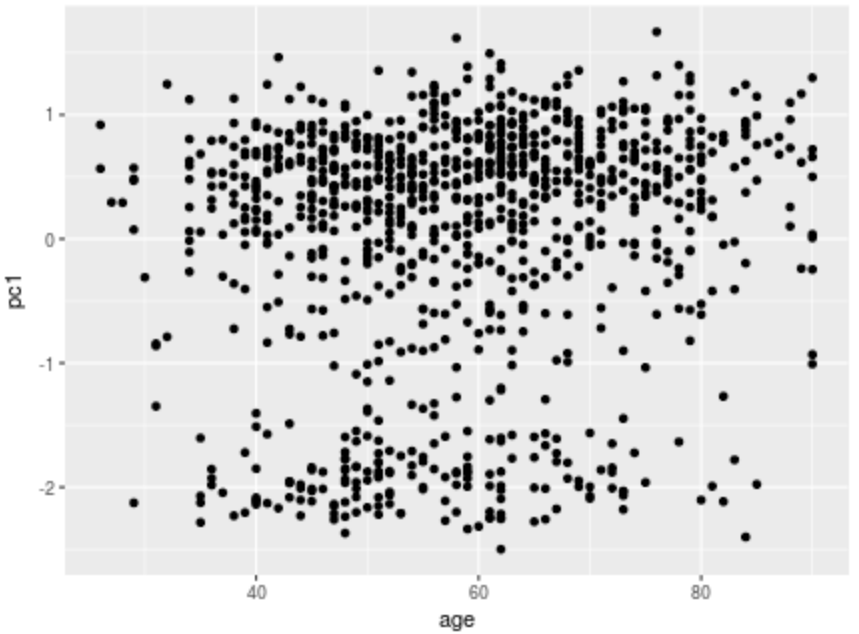

```
ggplot(df, aes(x = age)) +
  geom_density()
```



Is age correlated with PC1?

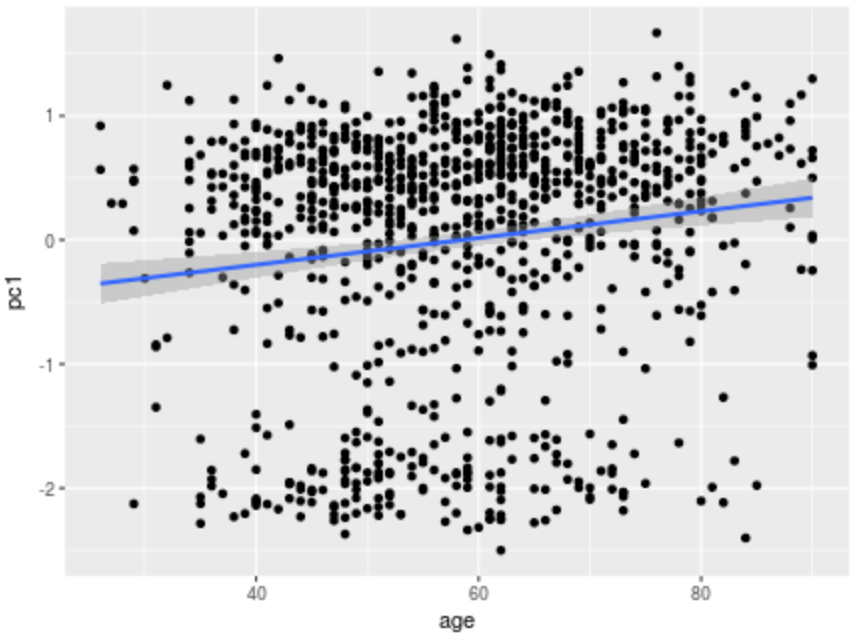
Is age correlated with PC1?

```
ggplot(df, aes(x = age, y = pc1)) +
  geom_point()
```



Is age correlated with PC1?

```
ggplot(df, aes(x = age, y = pc1)) +
  geom_point() +
  geom_smooth(method="lm")
```



aes() for different layers

Every layer can have its own aesthetics.

Here, we have only one layer `geom_point()`, so all of the following are equivalent

```
ggplot(df, aes(x = age, y = pc1, col = ER_status)) +
  geom_point()
```

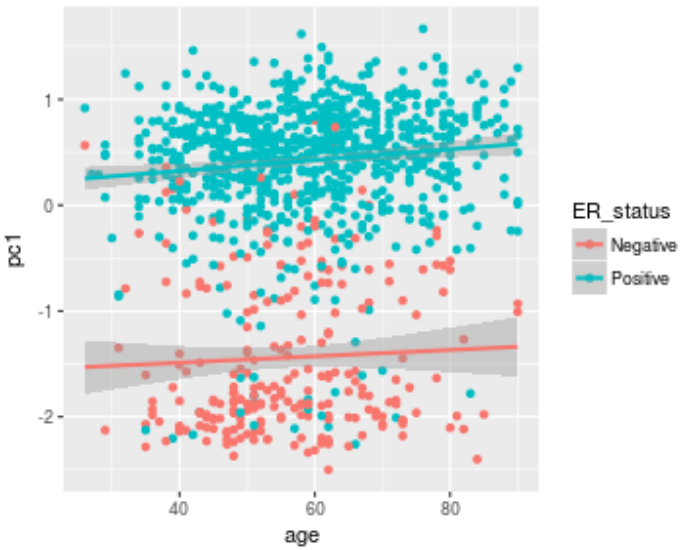
```
ggplot(df, aes(x = age, y = pc1)) +
  geom_point(aes(col = ER_status))
```

```
ggplot(df) +
  geom_point(aes(x = age, y = pc1, col = ER_status))
```

aes() for different layers

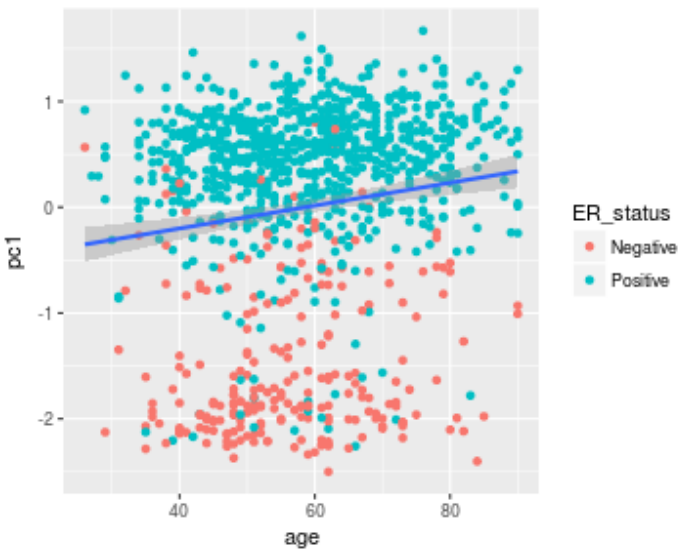
Here `aes(col = ER_status)` is shared for both layers.

```
ggplot(df, aes(x = age, y = pc1,
  geom_point() +
  geom_smooth(method="lm")
```



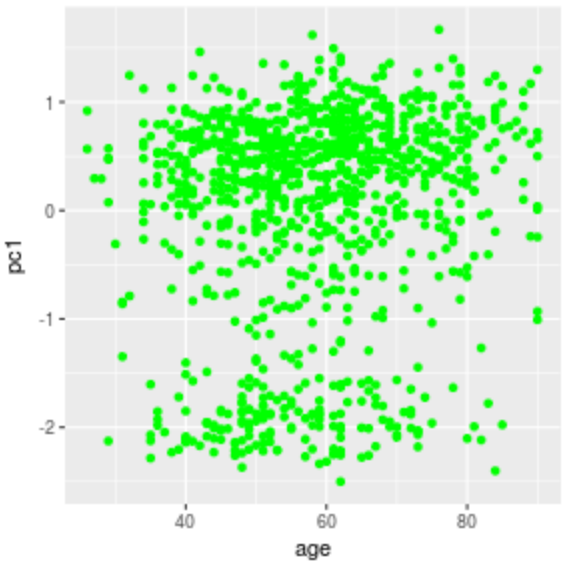
Here `aes(col = ER_status)` is specified for `geom_point()` only.

```
ggplot(df, aes(x = age, y = pc1)
  geom_point(aes(col = ER_status
  geom_smooth(method="lm")
```



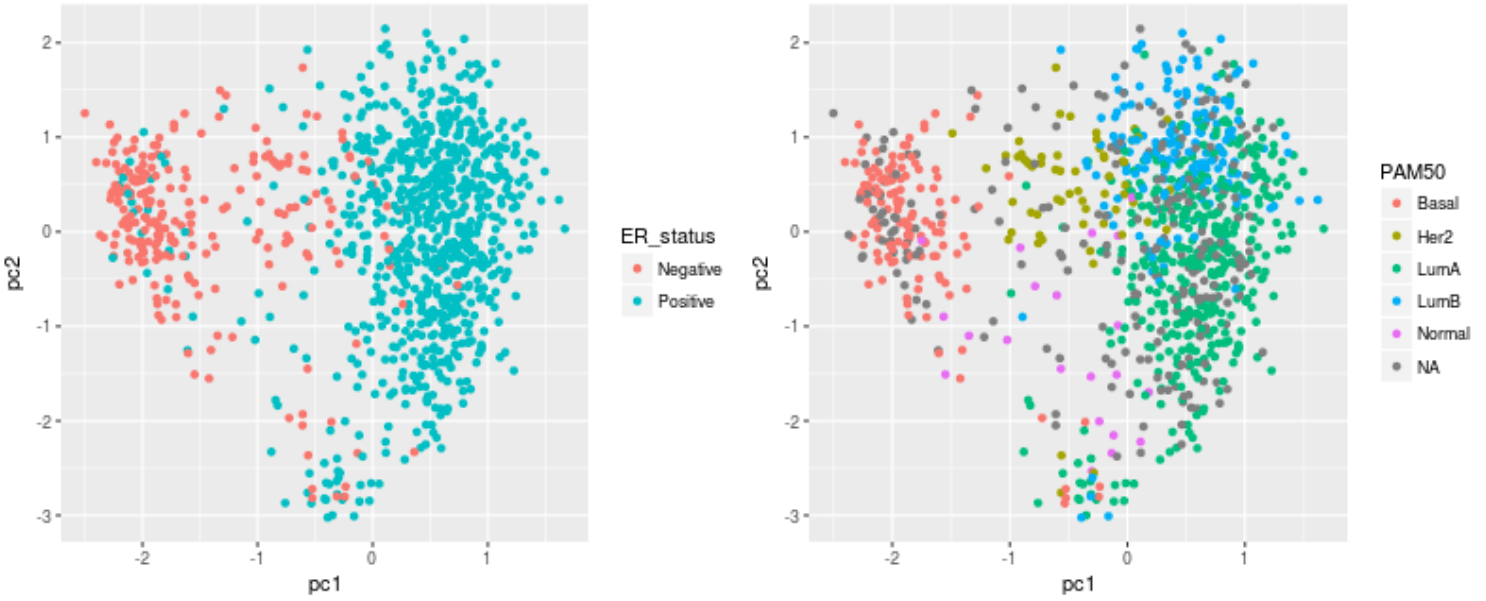
aes(): specifying colour manually

```
ggplot(df, aes(x = age, y = pc1)
  geom_point(col = "green")
```




```
ggplot(df, aes(x = age, y = pc1)
  geom_point(aes(col = "green"))
```

Lets come back to

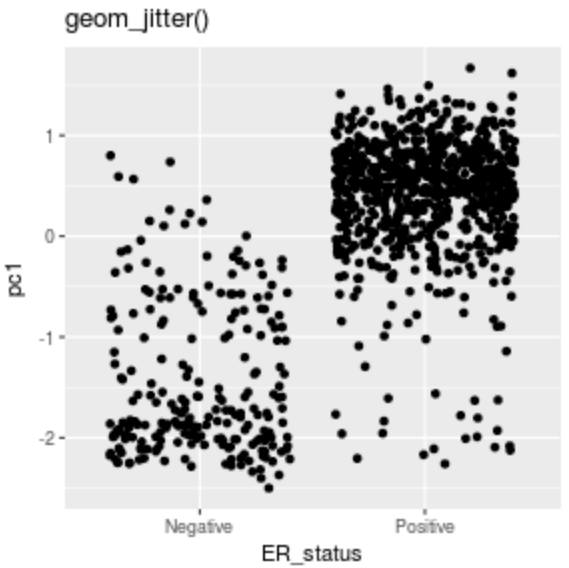


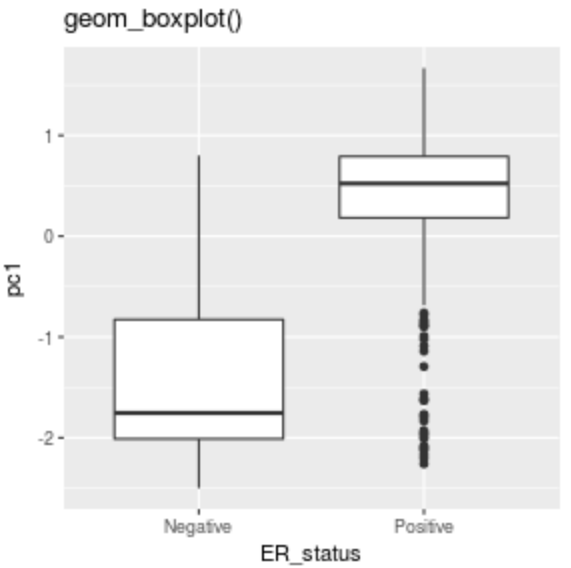
PC1 seems to distinguish well between

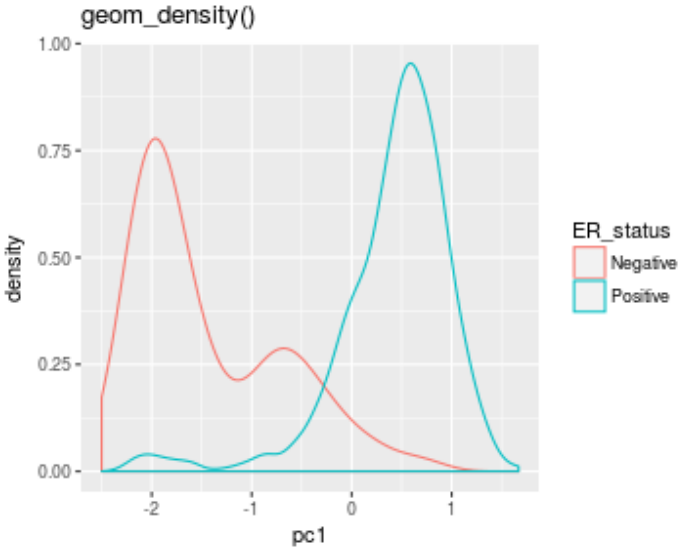
1. ER-positive and ER-negative cancers
2. between PAM50 subtypes

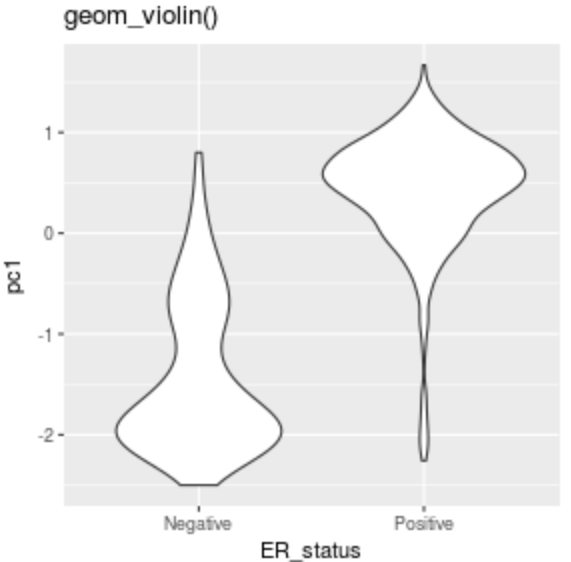
What is a good way to visualise this?

For example, you could consider



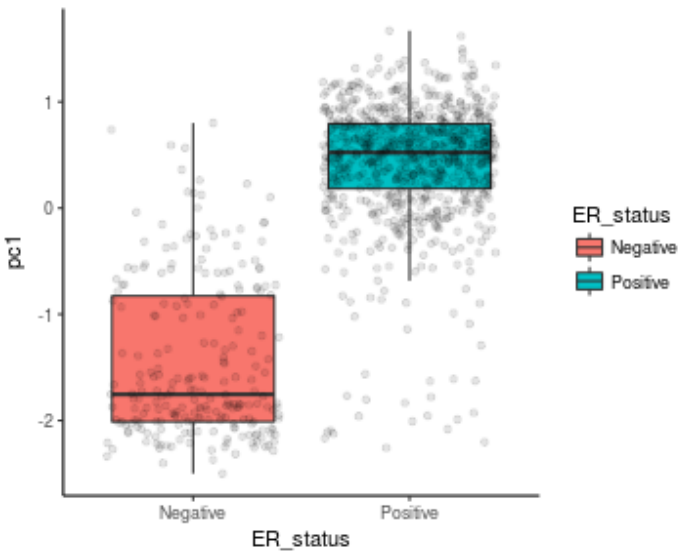


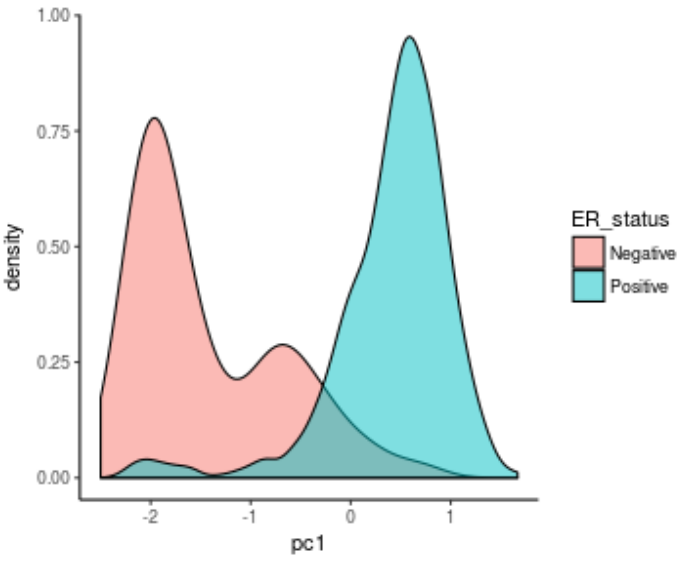




Tweaking

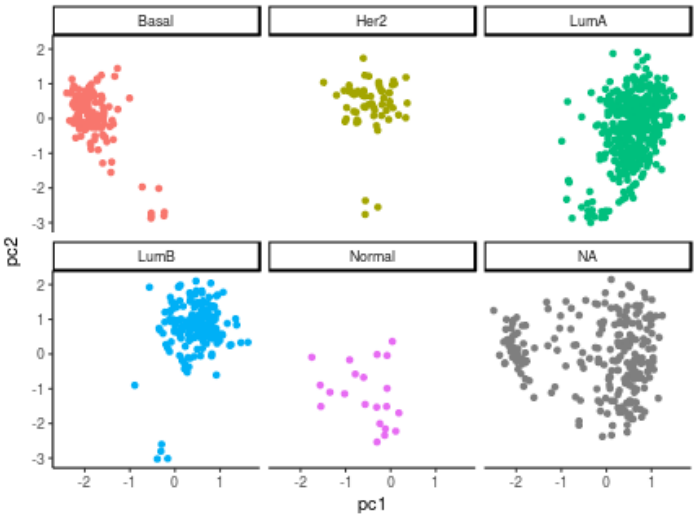
- Combining layers
- Add transparency using `alpha = 0.5`
- Changing the default theme:
  - `+ theme_bw()`
  - `+ theme_classic()`
  - etc, see also the ggthemes package
- Check out the ggplot2 cheat sheet by RStudio
- Various ggplot2 extensions http://www.ggplot2-exts.org/

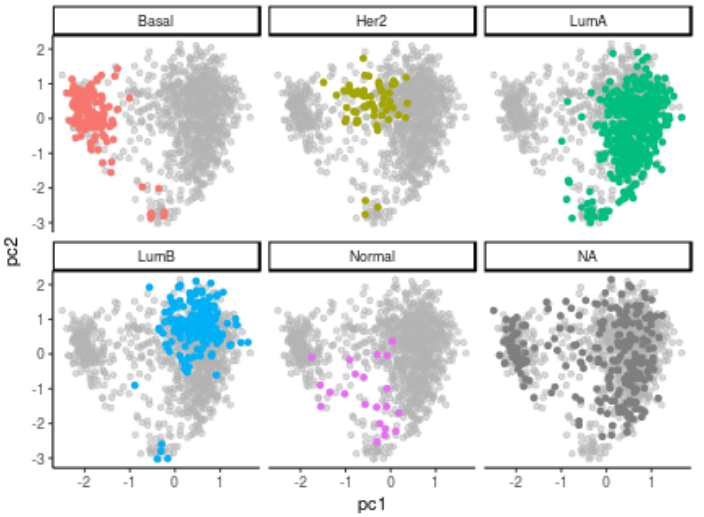




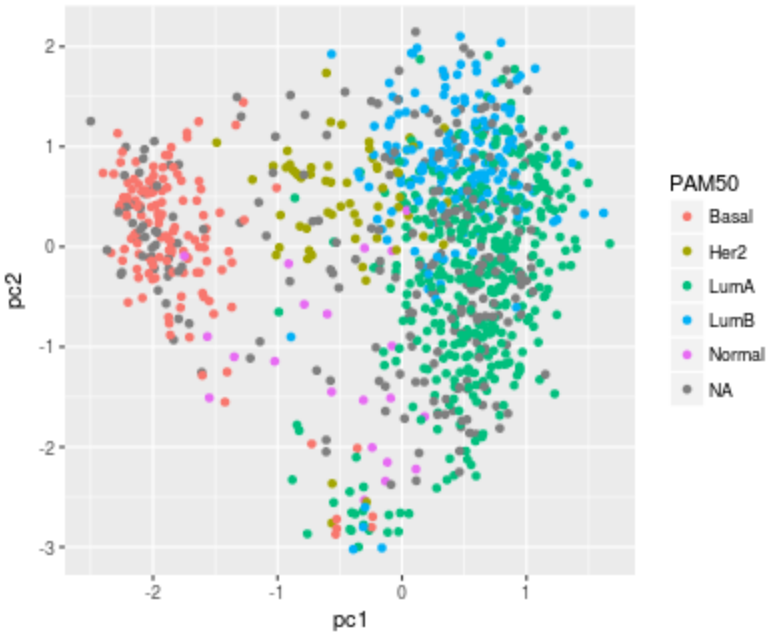
Further tweaking

Adding points to the background





Your turn



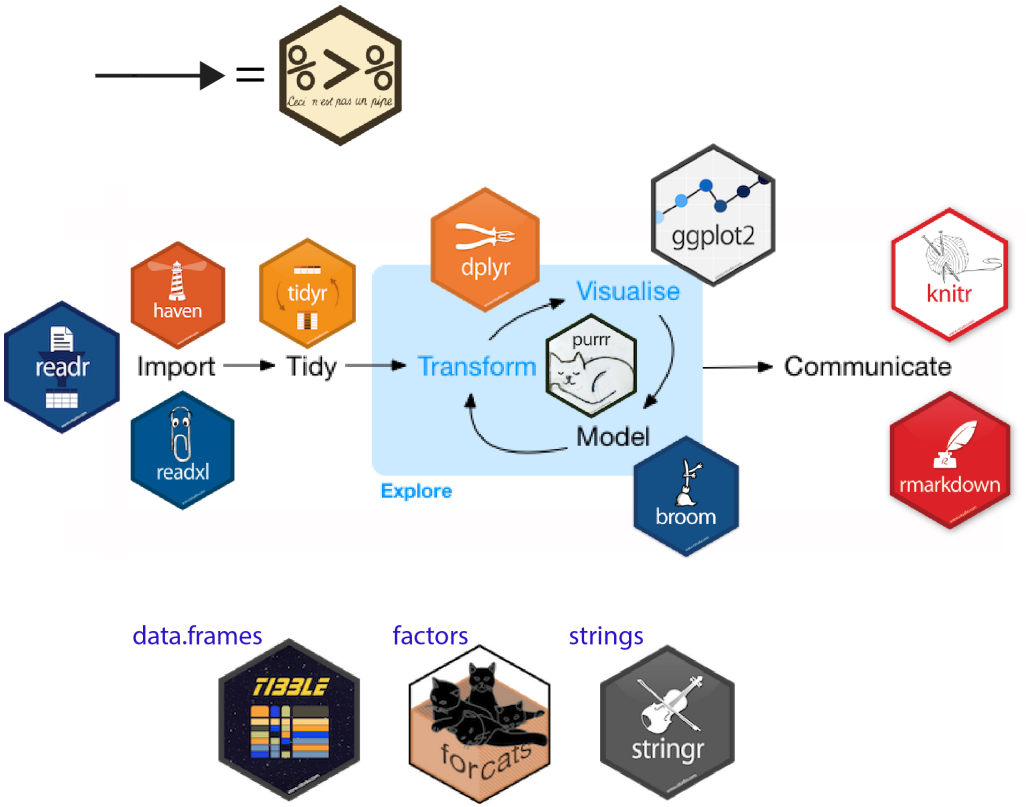
Design an informative (and nice) visualisation for showing how well PC1 distinguishes between PAM50 subtypes.

Note: So far we have been using a nice clean prepared data set. But in reality, they came from two different data sources:

- clinical variables / phenotypes
- gene expression data

=> In practice, data cleaning is the most time consuming part of the analysis.

Data exploration workflow



Source: https://lsru.github.io/tv_course

Data transformation using dplyr

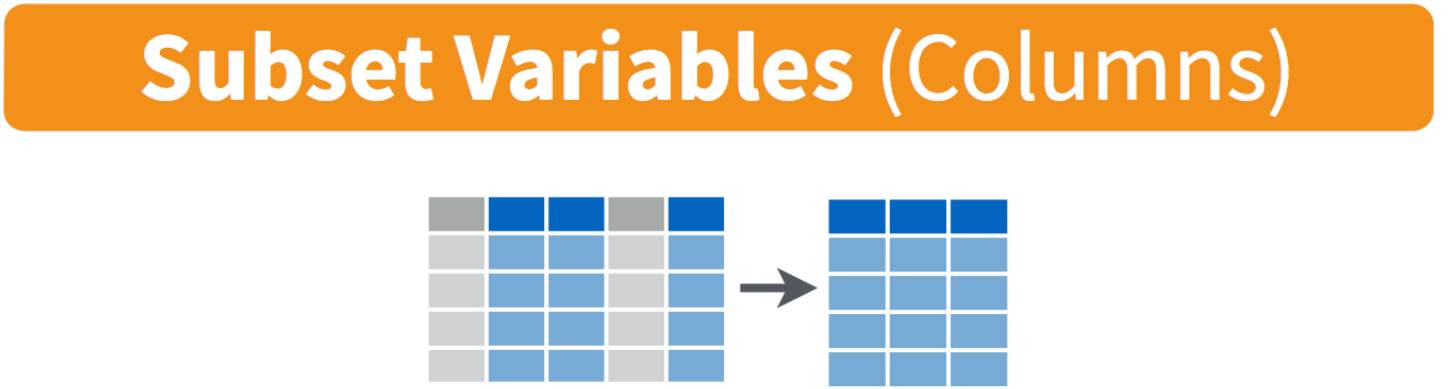
Subset rows with filter()

Example:

```
filter(df, ER_status == "Positive")
```

```
filter(df, ER_status == "Positive", PAM50 %in% c("LumA", "LumB"))
```

Subset columns with select()



Example:

```
select(df, id, pc1, pc2)
```

Or exclude certain columns

```
select(df, -ER_status, -PAM50)
```

Add a new column with mutate()

Example:

```
mutate(df, scaled_pc1 = 100 * pc1)
```

group_by() and summarise()

Example:

```
grouped_df <- group_by(df, ER_status)
summarise(grouped_df, count = n(), mean_pc1 = mean(pc1))
```

```
## # A tibble: 2 x 3
##   ER_status count   mean_pc1
##       <chr> <int>      <dbl>
## 1  Negative   235 -1.4394317
## 2  Positive   799  0.4233008
```

Pipes %>%

Introducing `%>%`, to help you write code in a way that easier to read and understand.

`g(f(x))` can be rewritten using the pipe `x %>% f() %>% g()`

Pipes %>%

Introducing `%>%`, to help you write code in a way that easier to read and understand.

`g(f(x))` can be rewritten using the pipe `x %>% f() %>% g()`

E.g. the following

```
df_filtered <- filter(df, ER_status == "Positive")
grouped_df <- group_by(df, PAM50)
summarise(grouped_df, count = n(), mean_pc1 = mean(pc1))
```

can be rewritten

```
df %>%
  filter(ER_status == "Positive") %>%
  group_by(PAM50) %>%
  summarise(count = n(), mean_pc1 = mean(pc1))
```

Pipes %>%

Introducing `%>%`, to help you write code in a way that easier to read and understand.

`g(f(x))` can be rewritten using the pipe `x %>% f() %>% g()`

E.g. the following

```
df_filtered <- filter(df, ER_status == "Positive")
grouped_df <- group_by(df, PAM50)
summarise(grouped_df, count = n(), mean_pc1 = mean(pc1))
```

can be rewritten

```
df %>%
  filter(ER_status == "Positive") %>%
  group_by(PAM50) %>%
  summarise(count = n(), mean_pc1 = mean(pc1))
```

- No need for storing intermediate variables.
- Easier to read and write.

Dplyr in one slide

- Pick observations by their values (`filter()`)
- Reorder the rows (`arrange()`, `arrange(desc())`)
- Pick a random subset of rows (`sample_frac()`, `sample_n()`)
- Pick top n rows (`top_n()`)
- Pick variables by their names (`select()`)
- Create new variables with functions of existing variables (`mutate()`)
- Collapse many values down to a single summary (`summarise()`)

Dplyr in one slide

- Pick observations by their values (`filter()`)
- Reorder the rows (`arrange()`, `arrange(desc())`)
- Pick a random subset of rows (`sample_frac()`, `sample_n()`)
- Pick top n rows (`top_n()`)
- Pick variables by their names (`select()`)
- Create new variables with functions of existing variables (`mutate()`)
- Collapse many values down to a single summary (`summarise()`)

All of these functions follow a common design:

- The first argument is a data frame.
- The subsequent arguments describe what to do with the data frame, using the variable names (without quotes).
- The result is a new data frame.

Your turn

1. Order the PAM50 subtypes by the number of observations in each subtype.
2. Calculate the mean PC1 and PC2 value for every PAM50 subtype. Can you add this to the PC1-PC2 scatterplot?

Tidy data

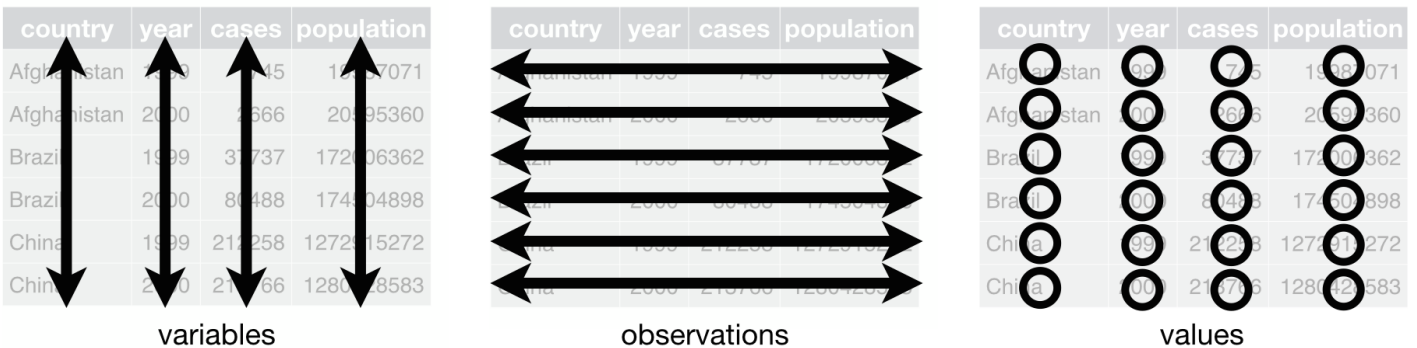
> “Tidy datasets are all alike, but every messy dataset is messy in its own way.” — Hadley Wickham

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

Tidy data

> "Tidy datasets are all alike, but every messy dataset is messy in its own way." — Hadley Wickham

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



When unsure, we should ask ourselves: Is our data trapped in column names? Column headers should be variable names (but not values).

Tidy vs untidy data

Wide-format

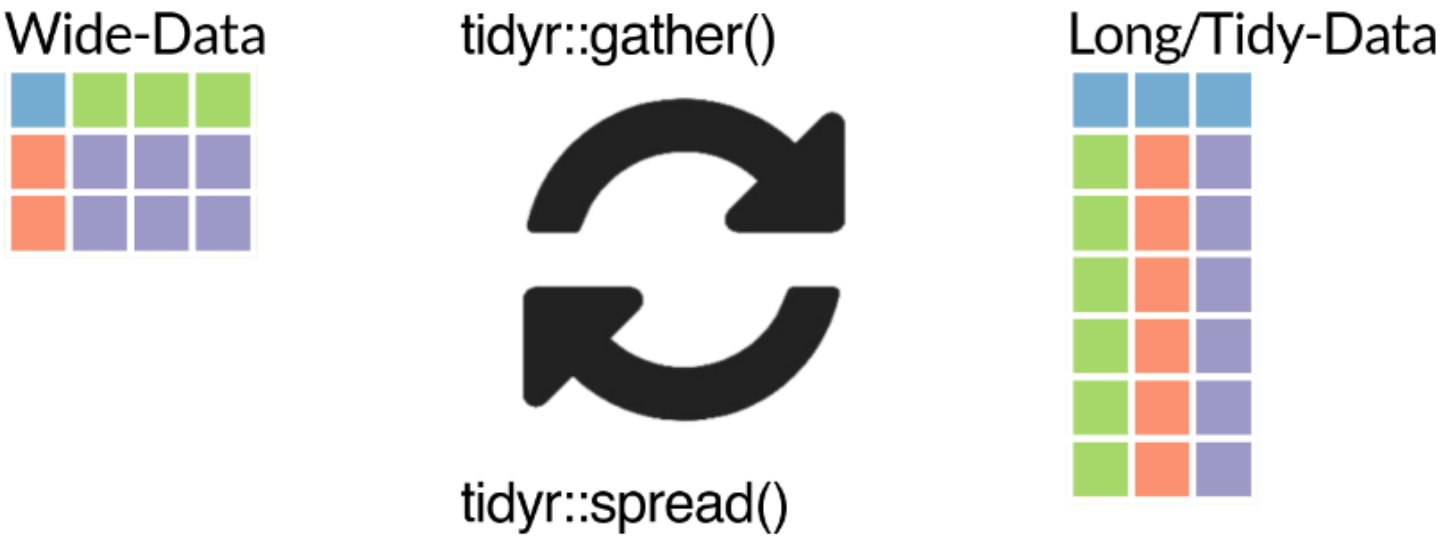
Year	Alice	Bob	Charlie
2010	105	100	90
2011	110	97	95

Long/Tidy-Data

Name	Year	Sales
Alice	2010	105
Alice	2011	110
Bob	2010	100
Bob	2011	97
Charlie	2010	90
Charlie	2011	95

Source: slides by David Zimmermann

tidyr for converting between the two formats



Wide-format

Year	Alice	Bob	Charlie
2010	105	100	90
2011	110	97	95

Long/Tidy-Data

Name	Year	Sales
Alice	2010	105
Alice	2011	110
Bob	2010	100
Bob	2011	97
Charlie	2010	90
Charlie	2011	95

Converting from wide format to tidy

```
data %>%
  gather(key = "Name", value = "Sales", -year)
```

Your turn

What is tidy format for a gene expression matrix?

```
df_exprs <- readRDS("data/TCGA_exprs.rds")
```

id	SLC7A2_ENSG00000003989	HSPB6_ENSG00000004776	PDK4_ENSG00000004799	ZMYND10_ENSG00000004838
TCGA-3C-AAAU	8.6058498	2.0863915	3.4674773	3.8919134
TCGA-3C-AALI	3.0177057	2.8843299	2.7492562	2.3850030
TCGA-3C-AALJ	4.3425394	3.0862352	5.3112761	2.7666318
TCGA-3C-AALK	4.8619597	2.7547870	3.4076714	4.4072346
TCGA-4H-AAAK	4.2986373	2.8731872	4.5942167	4.6344716
TCGA-5L-AAT0	2.5955476	2.4720156	3.1656451	3.4055321

Convert this gene expression data into a tidy format using `tidyr::gather()`.

Joining data frames

Source: slides by David Zimmermann

inner_join()

inner_join(df_a, df_b,
by = "id")

Output Data

full_join()


```
full_join(df_a, df_b,
          by = "id")
```

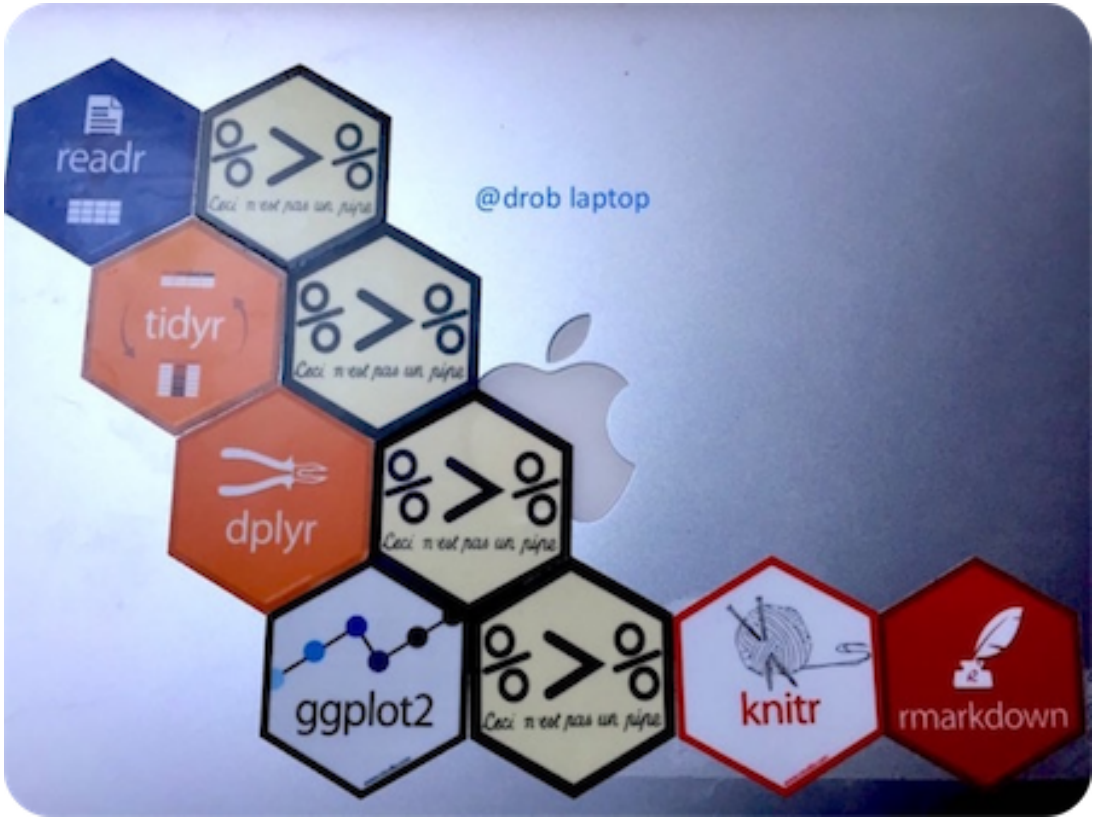
Output Data

left_join()


```
left_join(df_a, df_b,
by = "id")
```

Output Data

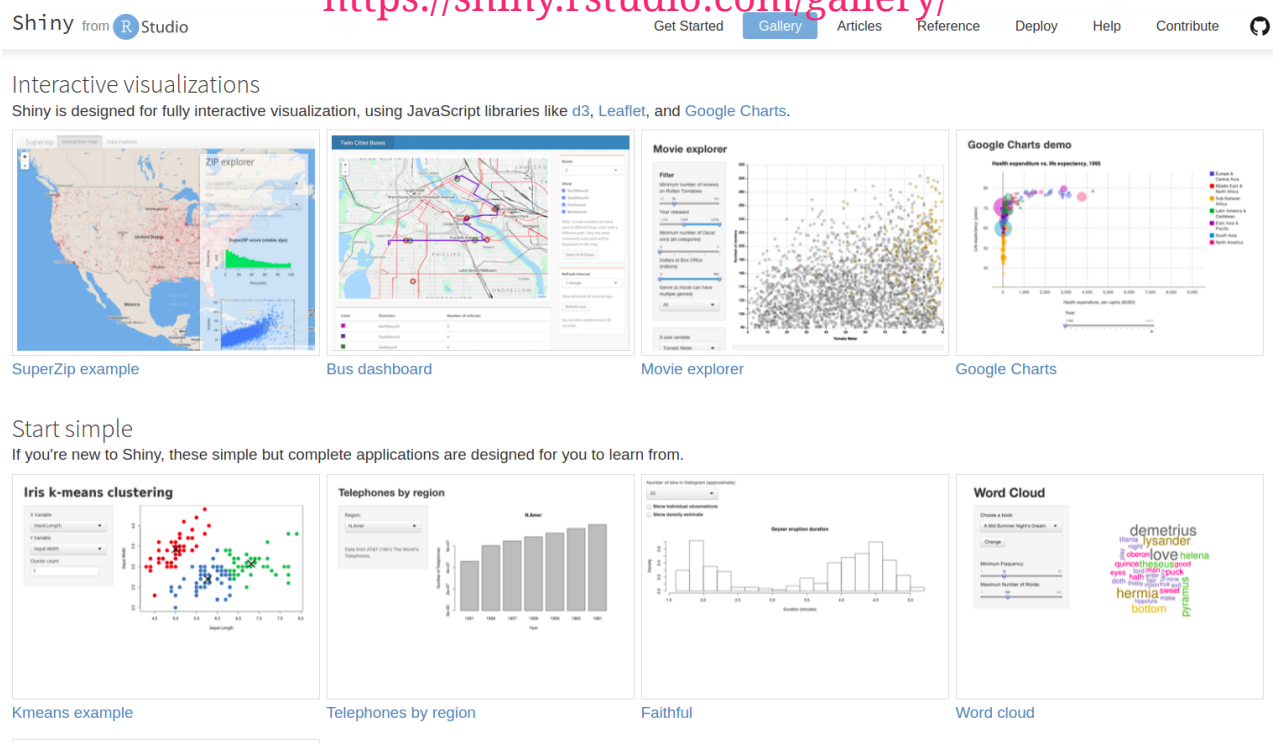
Data science workflow within tidyverse



Creating interactive visualisations

Turn your static figures into an interactive app with little effort using Shiny

https://shiny.rstudio.com/gallery/



kaspar.martens@gmail.com