SciencesPo
DEPARTMENT OF ECONOMICS

ScPoEconometrics

Session 2

Florian Oswald
SciencesPo Paris
2019-09-08

Working With Data

- Econometrics is about Data.
- In these slides we will start to look at this.
- Let's first all load a dataset with this command:

```
data("mpg",package="ggplot2")
```

Working With Data

- Econometrics is about Data.
- In these slides we will start to look at this.
- Let's first all load a dataset with this command:

```
data("mpg",package="ggplot2")
```

- Here are the first 6 rows

```
head(mpg)
```

```
##   manufacturer model displ year cyl      trans drv cty hwy fl   class
## 1         audi    a4   1.8 1999   4   auto(l5)   f  18  29  p compact
## 2         audi    a4   1.8 1999   4 manual(m5)   f  21  29  p compact
## 3         audi    a4   2.0 2008   4 manual(m6)   f  20  31  p compact
## 4         audi    a4   2.0 2008   4   auto(av)   f  21  30  p compact
## 5         audi    a4   2.8 1999   6   auto(l5)   f  16  26  p compact
## 6         audi    a4   2.8 1999   6 manual(m5)   f  18  26  p compact
```

The `mpg` dataset

- How many rows and columns?

```
dim(mpg)
```

```
## [1] 234  11
```

- `tail` gives you the last (6) rows.

```
tail(mpg)
```

- `names` gives the column names.

The `mpg` dataset: datatypes

- It's important to know how the data is stored.
- We can use `str` for that:

```
str(mpg)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    234 obs. of  11 variables:
##  $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
##  $ model       : chr  "a4" "a4" "a4" "a4" ...
##  $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr  "f" "f" "f" "f" ...
##  $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr  "p" "p" "p" "p" ...
##  $ class       : chr  "compact" "compact" "compact" "compact" ...
```

Summarizing Data

- One can learn only a limited amount from **looking** at a `data.frame`. 🔍

Summarizing Data

- One can learn only a limited amount from **looking** at a `data.frame`. 🔍
- Even if you *could* see all rows of the dataset, you would not know very much **about it**.

Summarizing Data

- One can learn only a limited amount from **looking** at a `data.frame`. 🔍
- Even if you *could* see all rows of the dataset, you would not know very much **about it**.
- We need to **summarize** the data for us to learn from it.

Summarizing Data

- One can learn only a limited amount from **looking** at a `data.frame`. 🔍
- Even if you *could* see all rows of the dataset, you would not know very much **about it**.
- We need to **summarize** the data for us to learn from it.
- In general, we can compute summary statistics, or visualize the data with plots.

Summarizing Data

- One can learn only a limited amount from **looking** at a `data.frame`. 🔍
- Even if you *could* see all rows of the dataset, you would not know very much **about it**.
- We need to **summarize** the data for us to learn from it.
- In general, we can compute summary statistics, or visualize the data with plots.
- Let's start with some statistics first!

Summarizing Data

- One can learn only a limited amount from **looking** at a `data.frame`. 🔍
- Even if you *could* see all rows of the dataset, you would not know very much **about it**.
- We need to **summarize** the data for us to learn from it.
- In general, we can compute summary statistics, or visualize the data with plots.
- Let's start with some statistics first!
- Let's look at two features: *central tendency* and *spread*.

Central Tendency

1. `mean(x)`: the average of all values in `x`.

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

2. `median`: the value x_j below and above which 50% of the values in `x` lie. m is the median if

$$\Pr(X \leq m) \geq 0.5 \text{ and } \Pr(X \geq m) \geq 0.5$$

3. The median is robust against *outliers*. 🤔? (later).

Central Tendency

1. `mean(x)`: the average of all values in `x`.

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

2. `median`: the value x_j below and above which 50% of the values in `x` lie. m is the median if

$$\Pr(X \leq m) \geq 0.5 \text{ and } \Pr(X \geq m) \geq 0.5$$

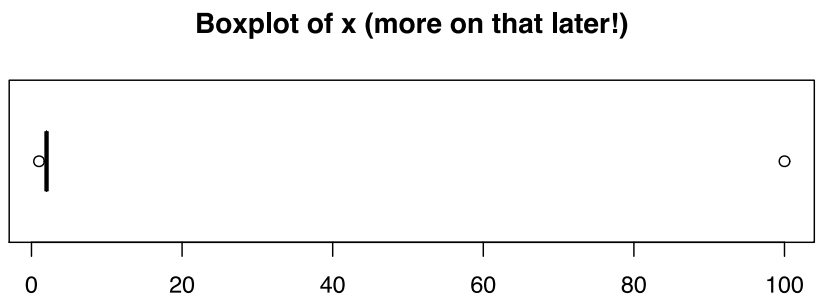
3. The median is robust against *outliers*. 🤔? (later).

```
x <- c(1,2,2,2,2,100)
mean(x)
```

```
## [1] 18.16667
```

```
mean(x) == sum(x) / length(x)
```

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




```
median(x)
```

```
## [1] 2
```

Spread

- Another interesting feature is how much a variable is *spread out* about it's center (the mean in this case).
- The *variance* is such a measure.

$$Var(X) = E[(X - \mu)^2]$$

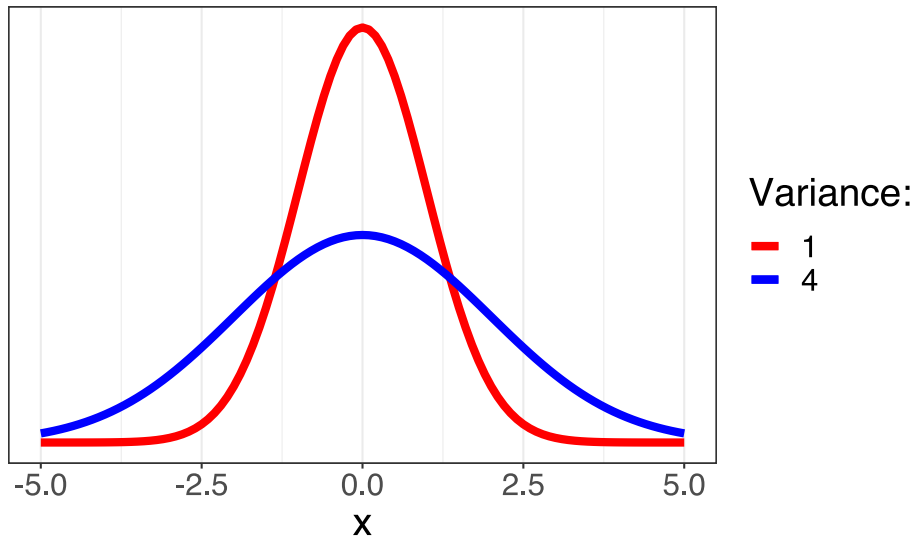
- Consider two Normal Distributions with equal mean at `0`:

Spread

- Another interesting feature is how much a variable is *spread out* about it's center (the mean in this case).
- The *variance* is such a measure.

$$Var(X) = E[(X - \mu)^2]$$

- Consider two Normal Distributions with equal mean at `0`:



- Compute with:

```
var(x)
range(x)    # range
```

The `table` function

- `table(x)` is a useful function that counts the occurence of each unique value in `x`:

```
table(x)
```

```
## x
##   1   2 100
##   1   4   1
```

```
table(mpg$trans)
```

```
##
##   auto(av)   auto(l3)   auto(l4)   auto(l5)   auto(l6)   auto(s4)
##          5          2         83         39          6          3
##   auto(s5)   auto(s6) manual(m5) manual(m6)
##          3         16         58         19
```

Crosstables

- Given two vectors, `table` produces a contingency table:

```
table(mpg$trans,mpg$drv)
```

```
##
##                4  f  r
##   auto(av)     0  5  0
##   auto(l3)     0  2  0
##   auto(l4)    34 37 12
##   auto(l5)    29  8  2
##   auto(l6)     2  2  2
##   auto(s4)     2  1  0
##   auto(s5)     1  2  0
##   auto(s6)     7  8  1
##   manual(m5) 21 33  4
##   manual(m6)  7  8  4
```

Crosstables

- Given two vectors, `table` produces a contingency table:

```
table(mpg$trans,mpg$drv)
```

```
##
##                4  f  r
##   auto(av)     0  5  0
##   auto(l3)     0  2  0
##   auto(l4)    34 37 12
##   auto(l5)    29  8  2
##   auto(l6)     2  2  2
##   auto(s4)     2  1  0
##   auto(s5)     1  2  0
##   auto(s6)     7  8  1
##   manual(m5) 21 33  4
##   manual(m6)  7  8  4
```

- with `prop.table`, we can get proportions:

```
prop.table(table(mpg$trans,mpg$drv),margin=2)
```

Plotting

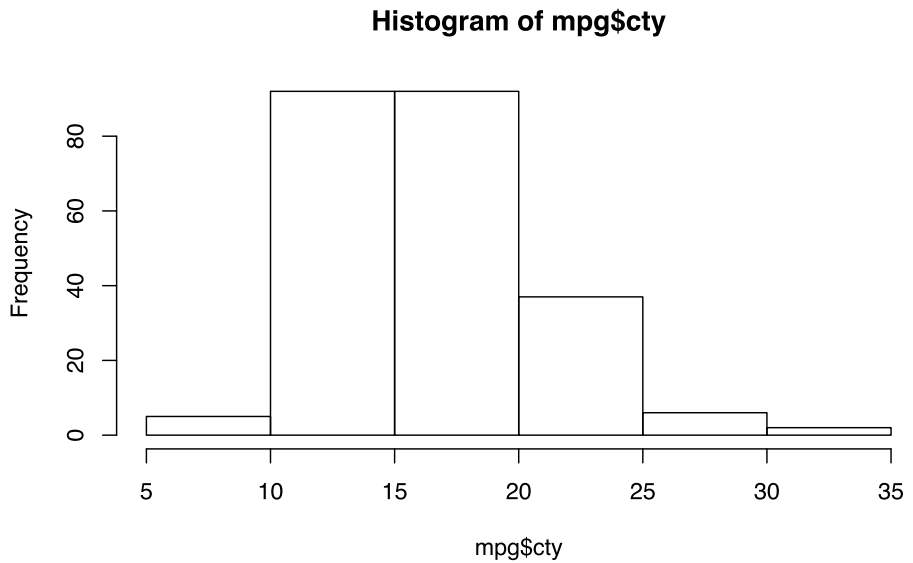
Plotting

- `R` base plotting is fairly good.
- There is an extremely powerful alternative in package `ggplot2`. We'll see both.
- First example: *histograms*. A histogram counts how many obserations fall within a certain bin.

Plotting

- `R` base plotting is fairly good.
- There is an extremely powerful alternative in package `ggplot2`. We'll see both.
- First example: *histograms*. A histogram counts how many obserations fall within a certain bin.

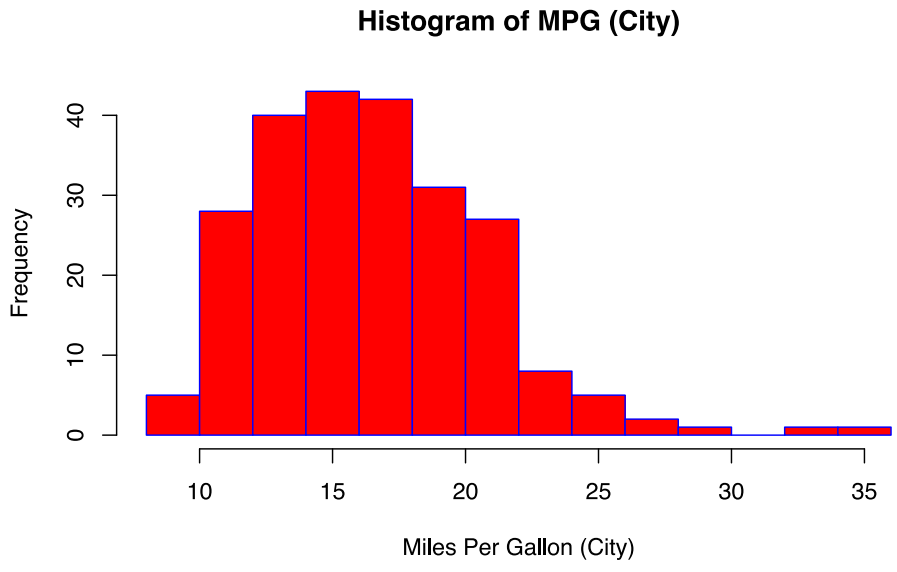
```
hist(mpg$cty)
```



A Nicer Histogram

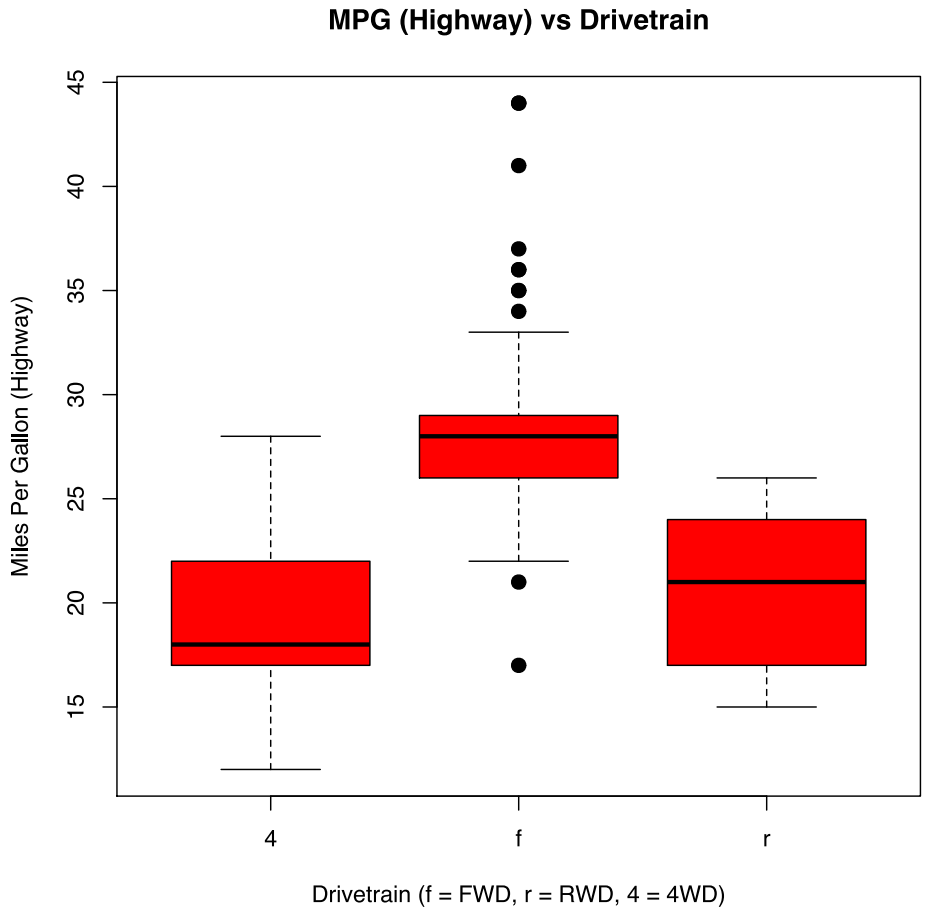
- We can give additional arguments to `hist`.
- Look at `?hist` for more.

```
hist(mpg$cty,
     xlab   = "Miles Per Gallon (City)",
     main   = "Histogram of MPG (City)",
     breaks = 12,
     col    = "red",
     border = "blue")
```



Looking for Outliers: Boxplots

- An *Outlier* is a datapoint far removed from the center of a distribution.
- Boxplots are an effective way to check: Outliers are dots.
- The thick line is the median.
- The *box* typically denotes the interquartile range.
- see `?boxplot`



Scatter Plots

- Two variables x and y

Scatter Plots

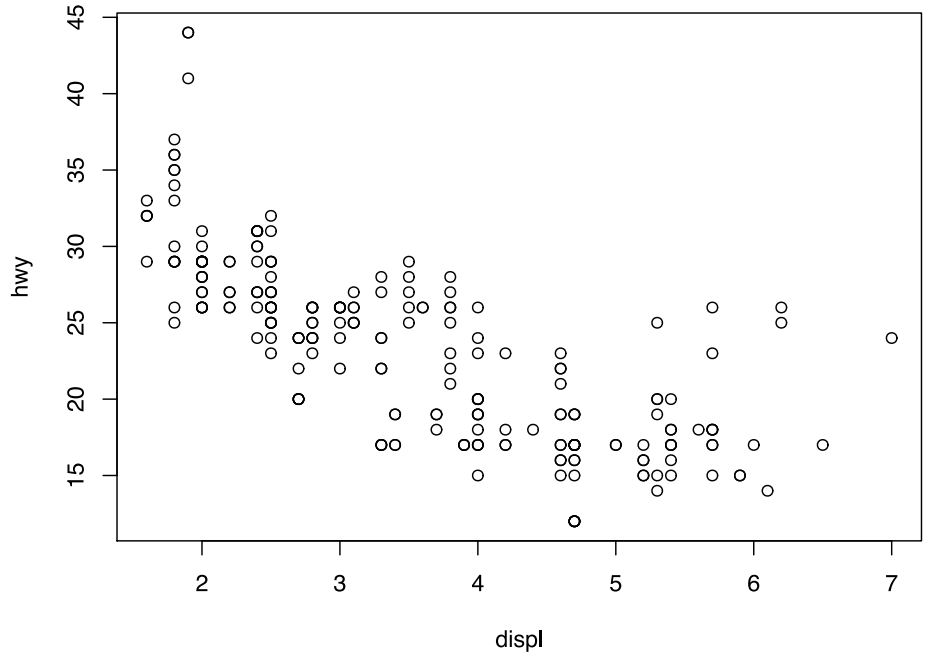
- Two variables x and y
- Natural to ask: How often do certain pairs of (x_i, y_i) occur?

```
head(mpg[,c("hwy","displ")])
```

```
## # A tibble: 6 x 2
##     hwy displ
##   <int> <dbl>
## 1    29   1.8
## 2    29   1.8
## 3    31   2
## 4    30   2
## 5    26   2.8
## 6    26   2.8
```

- That's what a scatter plots shows.

Scatter Plots



- Each dot is one pair (x_i, y_i).
- We often call it one *observation*.
- Corresponding to one *row* of the data.frame.
- Why do some dots appear *darker* than others here?

It's Tutorial Time!

Tutorial Chapter 2

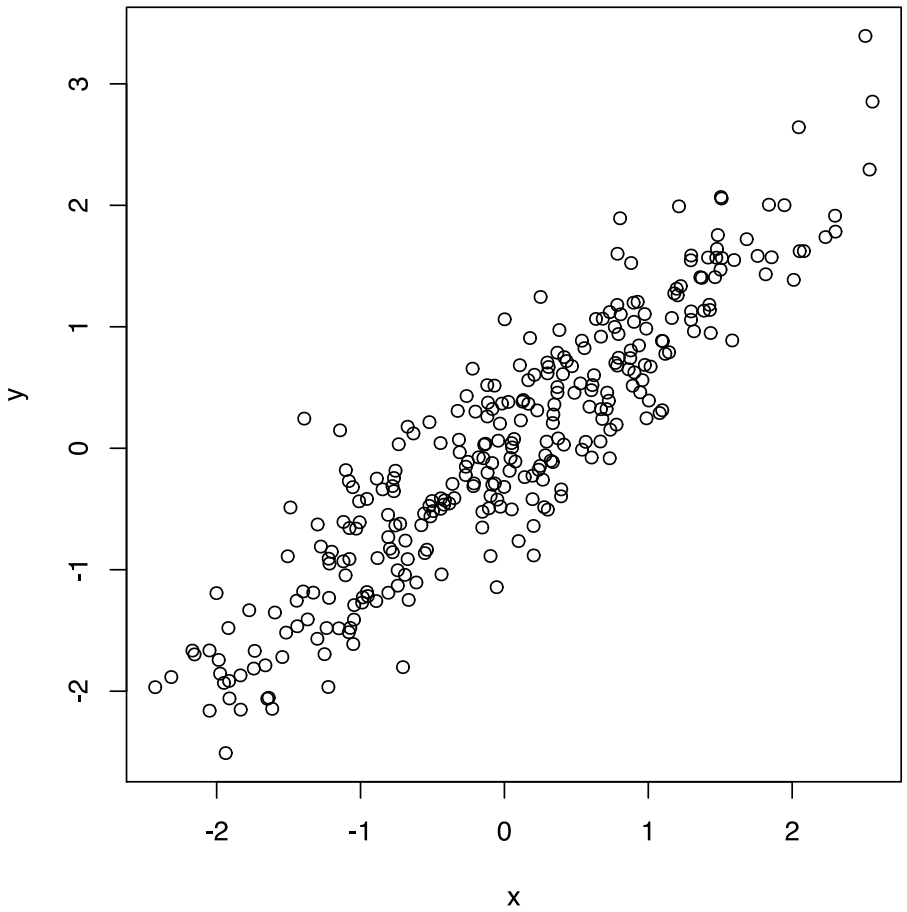
Time for our first tutorial!! Type this into your `RStudio` console:

```
library(ScPoEconometrics)
runTutorial('chapter2')
```

If you have trouble with the interactive doc, try this version (no interactive content):

```
ScPoEconometrics::runTutorial('chapter2-script')
```

How are x and y related? Covariance.



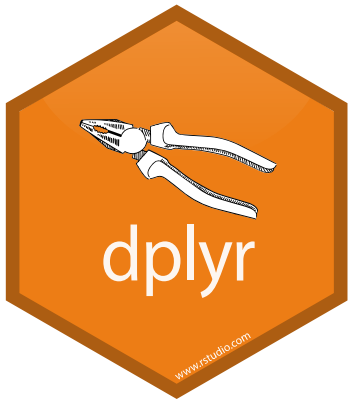
- How are `x` and `y` related here?
- This is the relevant section in the book about Covariance.
- This is **mandatory** reading.

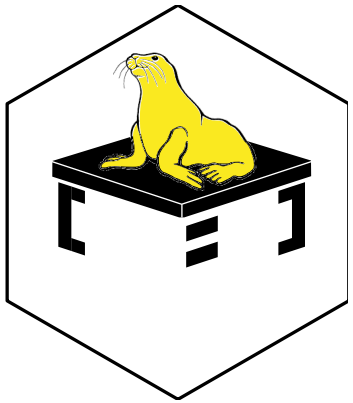
Correlation App

```
library(ScPoEconometrics)
runTutorial('correlation')
```

Intro do `dplyr`

- `dplyr` is part of the tidyverse package family.
- `data.table` is another alternative. I use it *a lot* in research.
- Both have pros and cons. We'll start you off with `dplyr`.





`dplyr` Overview

- You *must* read through Hadley Wickham's chapter. It's concise.
- The package is organized around a set of **verbs**, i.e. *actions* to be taken.
- We operate on `data.frames` or `tibbles` (*nicer looking* data.frames.)
- All *verbs*: First arg is a data.frame, subsequent args describe what to do, returns another data.frame.

`dplyr` Overview

- You *must* read through Hadley Wickham's chapter. It's concise.
- The package is organized around a set of **verbs**, i.e. *actions* to be taken.
- We operate on `data.frames` or `tibbles` (*nicer looking* data.frames.)
- All *verbs*: First arg is a data.frame, subsequent args describe what to do, returns another data.frame.

Verbs

1. Choose observations based on a certain value (i.e. subset): `filter()`
2. Reorder rows: `arrange()`
3. Select variables by name: `select()`
4. Create new variables out of existing ones: `mutate()`
5. Summarise variables: `summarise()`

R package `nycflights13`

```
library(nycflights13)
library(dplyr)
flights
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # … with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

🚨 This is a `tibble` (more informative `data.frame`)

Subset a data.frame with `filter()`

- `filter` has the same purpose than `subset`
- Which flights on 01/03/2013 departed between 5 and 6 AM with more than 10 minutes ahead of schedule?

```
filter(flights, day == 1, month == 3,
       dep_time >= 500 & dep_time <= 600, dep_delay < -5)
```

Subset a data.frame with `filter()`

- `filter` has the same purpose than `subset`
- Which flights on 01/03/2013 departed between 5 and 6 AM with more than 10 minutes ahead of schedule?

```
filter(flights, day == 1, month == 3,
       dep_time >= 500 & dep_time <= 600, dep_delay < -5)
```

```
## # A tibble: 10 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     3     1      505            515       -10      746
##  2  2013     3     1      521            530        -9      813
##  3  2013     3     1      549            600       -11      639
##  4  2013     3     1      550            600       -10      747
##  5  2013     3     1      551            559        -8      722
##  6  2013     3     1      551            600        -9      717
##  7  2013     3     1      551            600        -9      845
##  8  2013     3     1      552            600        -8      656
##  9  2013     3     1      554            600        -6      651
## 10  2013     3     1      558            605        -7      803
## # … with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

Create a Filter: Comparisons and Logical Ops

- We have standard suite of `>`, `<`, `>=`, `<=`, `!=`, `==`.
- Construct more complex filters with logical operators
  1. `x & y`: `x` **and** `y`
  2. `x | y`: `x` **or** `y`
  3. `!y`: **not** `y`
- `R` has the convenient `x %in% y` operator, `TRUE` if `x` is *a member of* `y`.

```
3 %in% 1:3
```

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

```
c(2,5) %in% 2:10  # also vectorized
```

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

```
c("S","Po") %in% c("Sciences","Po")  # also strings
```

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

Missing Values: `NA`

- Whenever a value is *missing*, we code it as `NA`.

```
x <- NA
```

- `R` propagates `NA` through operations:

```
NA > 5
```

```
## [1] NA
```

```
NA + 10
```

```
## [1] NA
```

- the function `is.na(x)` returns `TRUE` if `x` is an `NA`.

```
is.na(x)
```

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

Missing Values: NA

- Whenever a value is *missing*, we code it as `NA`.

```
x <- NA
```

- `R` propagates `NA` through operations:

```
NA > 5
```

```
## [1] NA
```

```
NA + 10
```

```
## [1] NA
```

- the function `is.na(x)` returns `TRUE` if `x` is an `NA`.

```
is.na(x)
```

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

- What is confusing is that

```
NA == NA
```

```
## [1] NA
```

- It's easy to illustrate like that:

```
# Let x be Mary's age. We don't know how old she
x <- NA

# Let y be John's age. We don't know how old he i
y <- NA

# Are John and Mary the same age?
x == y
```

```
## [1] NA
```

```
#> [1] NA
# We don't know!
```

Task 2.1

- You should read through 5.2.1 and learn more about *comparisons* and *logical operators*.

Then, find all flights that:

1. Had an arrival delay of two or more hours
2. Flew to Houston (IAH or HOU)
3. Were operated by United, American, or Delta
4. Departed in summer (July, August, and September)
5. Arrived more than two hours late, but didn't leave late
6. How many flights have a missing `dep_time`? What other variables are missing? What might these rows represent?

`dplyr` Self Study

We can also

1. *sort* a data.frame,
2. *select* some columns from it, and
3. add new columns.

For case study 1, you have to read those short sections yourself (click on function name):

1. `arrange()`
2. `select()`
3. `mutate()`

Split-Apply-Combine

- Often we do *some* operation **by** some group in our dataset:
    - Mean height by sex.
    - Maximum income by age, etc
- For this, we need to
    1. Split the data **by** `x`
    2. Apply to each chunk `xyz`
    3. Recombine all chunks
- in `dplyr`, that's `group_by()`.

Split-Apply-Combine

- Often we do *some* operation **by** some group in our dataset:
  - Mean height by sex.
  - Maximum income by age, etc
- For this, we need to
  1. Split the data **by** `x`
  2. Apply to each chunk `xyz`
  3. Recombine all chunks
- in `dplyr`, that's `group_by()`.

1. `group_by(x)` groups/splits `data.frame` by `x`:

```
g = group_by(iris, Species)
class(g)
```

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

2. `summarise` each chunk and re-combine

```
summarise(
  g, mean_l = mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    mean_l
##   <fct>       <dbl>
## 1 setosa       5.01
## 2 versicolor   5.94
## 3 virginica    6.59
```

Chaining 🔗 Commands Together: The Pipe

- `magrittr` gives us the *pipe* `%>%`.
- This is like the UNIX pipe `|`: it passes arguments on.
- `x %>% f(y)` becomes `f(x,y)`.
- With the *pipe* you construct data *pipelines*.

Our above example would become:

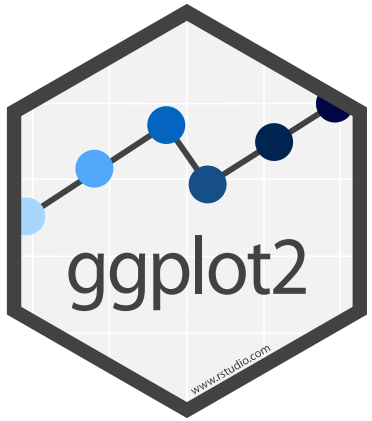
```
iris %>%
  group_by(Species) %>%
  summarise(mean_l = mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    mean_l
##   <fct>       <dbl>
## 1 setosa       5.01
## 2 versicolor   5.94
## 3 virginica    6.59
```

which is equivalent to, but nicer than:

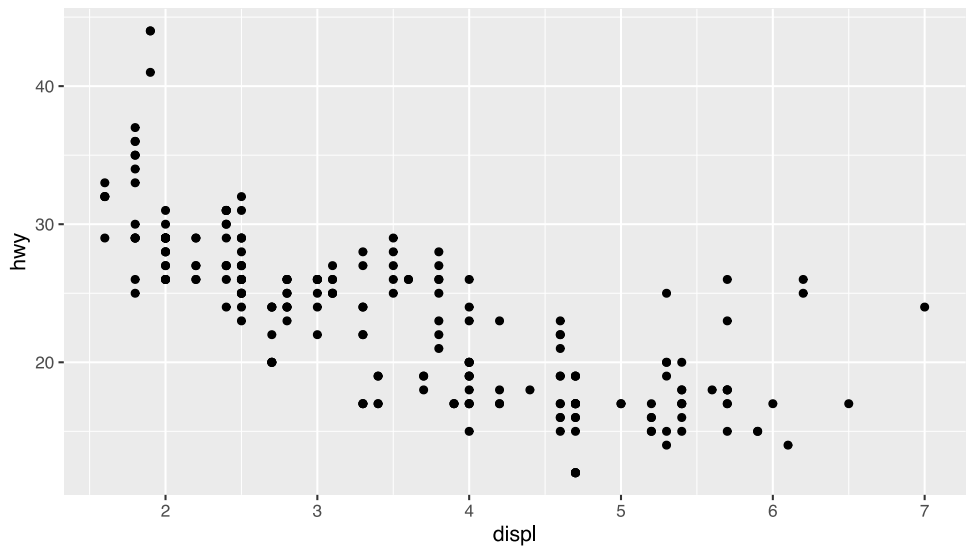
```
summarise(
  group_by(iris, Species),
  mean_l = mean( Sepal.Length))
```

Quick `ggplot2` Intro



- Excellent cheatsheet on project website.
- We construct a `ggplot` in layers. We `+` add layers.
- In `aes` (aestethics) we say how data maps onto plot.
- We choose a `geom_` function to choose the geometry.

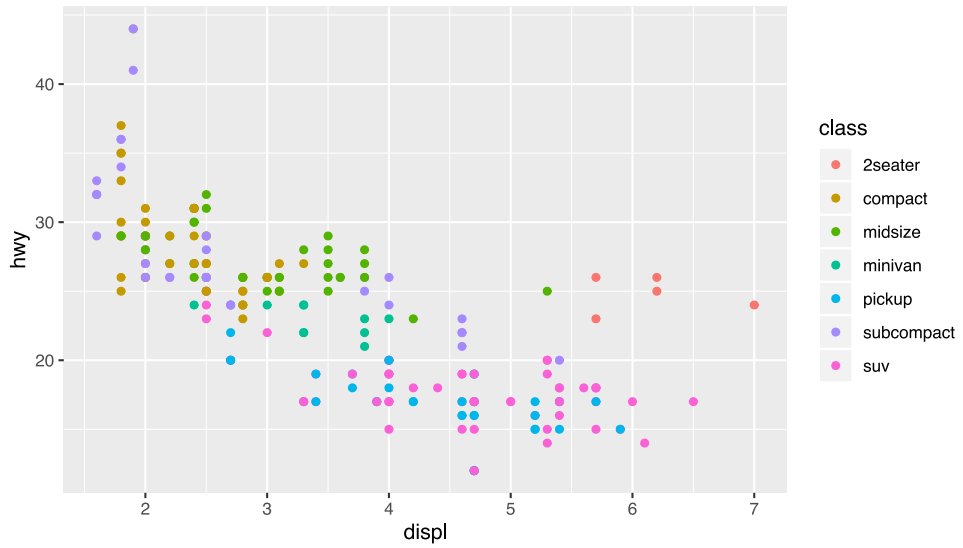
```
library(ggplot2)
ggplot(data = mpg,    # base layer
       mapping = aes(x = displ, y = hwy)) +
   geom_point()    # add geom_ layer
```



Quick `ggplot2` Intro

- We can add more layers to this plot.
- We can map another variable to another feature, like color, size, shape etc.
- We could also add another `geom_` function.

```
ggplot(data = mpg,
       aes(x = displ,
           y = hwy,
           color = class)) +  # map `class` to color
   geom_point()
```



SciencesPo
DEPARTMENT OF ECONOMICS

END

	florian.oswald@sciencespo.fr
	Slides
	Book
	@ScPoEcon
	@ScPoEcon