

# Tidyverse

## R for Data Science

Juan R Gonzalez  
[juanr.gonzalez@isglobal.org](mailto:juanr.gonzalez@isglobal.org)

BRGE - Bioinformatics Research Group in Epidemiology  
ISGlobal - Barcelona Institute for Global Health  
<http://brge.isglobal.org>

# Preliminaries

```
install.packages(c("tidyverse",  
                  "nycflights13"))
```

I assume you are familiar with:

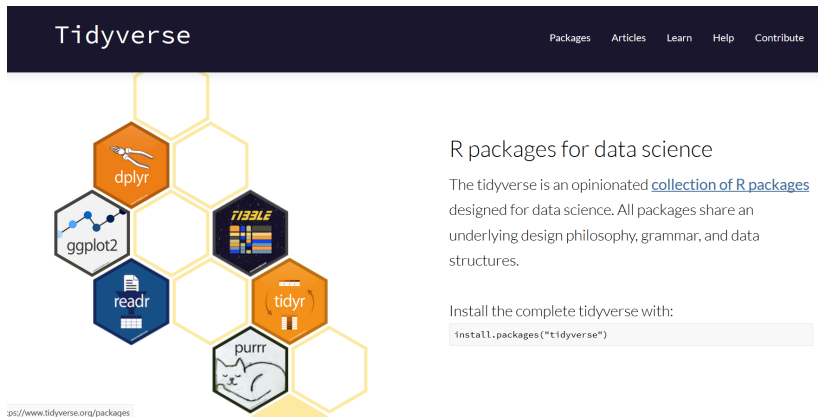
- ▶ R
- ▶ RStudio
- ▶ RMarkdown

# Introduction

- ▶ Data science is an exciting discipline that allows you to turn raw data into understanding, insight, and knowledge.
- ▶ R can help you learn the most important tools that will allow you to do data science.
- ▶ Data science is a huge field, and this lectures aim to introduce you on it

# Tidyverse

# Introduction



## R packages for data science

The tidyverse is an opinionated [collection of R packages](https://www.tidyverse.org/packages) designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

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

Figure 1: Tidyverse

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

# What you will learn

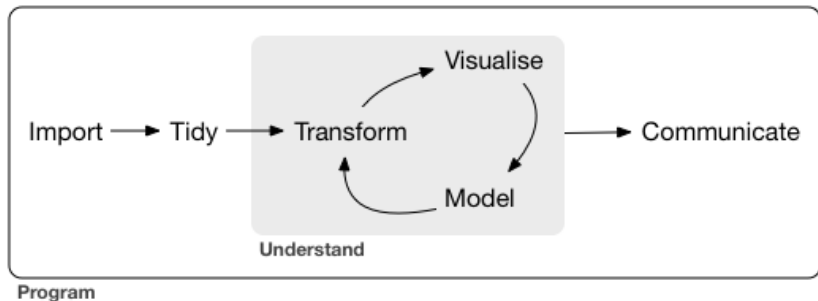


Figure 2: Data science

**Tidying** data means storing it in a consistent form that matches the semantics of the dataset with the way it is stored

# Data wrangling

# Data wrangling

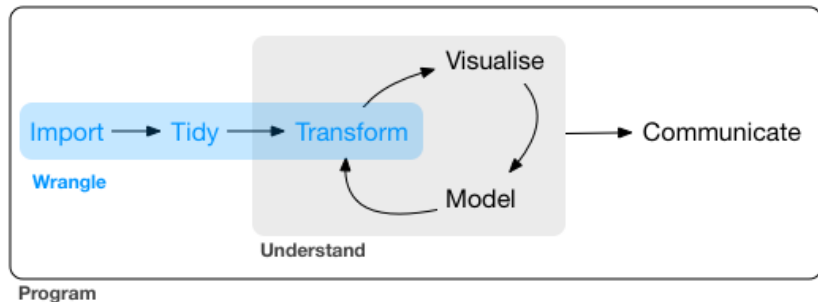


Figure 3: Data wrangling



# Data wrangling

- ▶ In **tibbles**, the counterpart of `data.frames` in tidyverse.
- ▶ In **data import** you get data from disk into R focusing on plain-text rectangular formats (other types are possible)
- ▶ In **tidy** data, a consistent way of storing your data that makes transformation, visualisation, and modelling easier.

# Data Wrangling

Also encompasses data transformation (not covered here) that facilitates:

- ▶ Relational data will give you tools for working with multiple interrelated datasets.
- ▶ Strings will introduce regular expressions, a powerful tool for manipulating strings.
- ▶ Factors are how R stores categorical data. They are used when a variable has a fixed set of possible values, or when you want to use a non-alphabetical ordering of a string.
- ▶ Dates and times will give you the key tools for working with dates and date-times.

# Tibbles

You can learn more by executing `vignette("tibble")`

```
library(tidyverse)
```

## ► Creating tibbles

```
head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

```
iris.tib <- as_tibble(iris)
iris.tib
```

```
# A tibble: 150 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
      <dbl>       <dbl>       <dbl>       <dbl> <fct>
1         5.1         3.5         1.4         0.2 setosa
2         4.9         3.0         1.4         0.2 setosa
3         4.7         3.2         1.3         0.2 setosa
4         4.6         3.1         1.5         0.2 setosa
5          5.0         3.6         1.4         0.2 setosa
6         5.4         3.9         1.7         0.4 setosa
7         4.6         3.4         1.4         0.3 setosa
```

A new tibble can be created by (data is recycled):

```
tibble(  
  x = 1:5,  
  y = 1,  
  z = x ^ 2 + y  
)
```

```
# A tibble: 5 x 3  
      x     y     z  
  <int> <dbl> <dbl>  
1     1     1     2  
2     2     1     5  
3     3     1    10  
4     4     1    17  
5     5     1    26
```

NOTE: It never changes the type of data (i.e. character to factor)

Tibbles have a refined print method that shows only the first 10 rows, and all the columns that fit on screen. This can be changed

```
print(iris.tib, n = 10, width = Inf)
```

```
# A tibble: 150 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
    <dbl>         <dbl>         <dbl>         <dbl> <fct>
1      5.1         3.5           1.4         0.2 setosa
2      4.9         3           1.4         0.2 setosa
3      4.7         3.2           1.3         0.2 setosa
4      4.6         3.1           1.5         0.2 setosa
5       5          3.6           1.4         0.2 setosa
6      5.4         3.9           1.7         0.4 setosa
7      4.6         3.4           1.4         0.3 setosa
8       5          3.4           1.5         0.2 setosa
9      4.4         2.9           1.4         0.2 setosa
10     4.9         3.1           1.5         0.1 setosa
# ... with 140 more rows
```

```
print(iris.tib, n = 10, width = 25)
```

```
# A tibble: 150 x 5
  Sepal.Length
      <dbl>
1         5.1
2         4.9
3         4.7
4         4.6
5          5
6         5.4
7         4.6
8          5
9         4.4
10        4.9
# ... with 140 more
#   rows, and 4 more
#   variables:
#     Sepal.Width <dbl>,
#     Petal.Length <dbl>,
#     Petal.Width <dbl>,
#     Species <fct>
```

## ► Subsetting

```
df <- tibble(  
  x = runif(5),  
  y = rnorm(5)  
)
```

```
# Extract by name  
df$x
```

```
[1] 0.617075207 0.441019911 0.007119353 0.642234022 0.582265506
```

```
df[["x"]]
```

```
[1] 0.617075207 0.441019911 0.007119353 0.642234022 0.582265506
```

```
# Extract by position  
df[[1]]
```

```
[1] 0.617075207 0.441019911 0.007119353 0.642234022 0.582265506
```

## Exercises (tibbles)

1. How can you know whether an object is a tibble? (Hint: try printing `mtcars`, which is a regular data frame).
2. If you have the name of a variable stored in an object, e.g. `var <- "mpg"`, how can you extract the reference variable from a tibble?
3. What option controls how many additional column names are printed at the footer of a tibble?
4. Practice creating new variables in the following data frame

```
tbl <- tibble(  
  age = c(14, 18, 22, 12, 16, 19, 21, 24),  
  chol = c(172, 180, 185, 170, 175, 188, 190, 192),  
  sex = c("male", "male", "female", "female", "female", "male", "male", "male")  
)
```

by: + Extracting the variable called `sex`. + Plotting a scatterplot of `age` vs `chol`. + Creating a new column called `chol2` which is `chol` to the power of 2. + Rename the columns to one, two and three.



## Data import

# Data import

The key package is *readr*

- ▶ *read\_csv()* reads comma delimited files, *read\_csv2()* reads semicolon separated files (common in countries where , is used as the decimal place), *read\_tsv()* reads tab delimited files, and *read\_delim()* reads in files with any delimiter.
- ▶ *read\_fwf()* reads fixed width files. You can specify fields either by their widths with *fwf\_widths()* or their position with *fwf\_positions()*. *read\_table()* reads a common variation of fixed width files where columns are separated by white space.
- ▶ *read\_log()* reads Apache style log files. (But also check out *webreadr* which is built on top of *read\_log()* and provides many more helpful tools.)

## Comparison with base R

- ▶ They are typically much faster ( $\sim 10\times$ ) than their base equivalents. Long running jobs have a progress bar, so you can see what's happening. If you're looking for raw speed, try `data.table::fread()`. It doesn't fit quite so well into the tidyverse, but it can be quite a bit faster.
- ▶ They produce tibbles, they don't convert character vectors to factors, use row names, or munge the column names. These are common sources of frustration with the base R functions [Hadley statement!].
- ▶ They are more reproducible. Base R functions inherit some behaviour from your operating system and environment variables, so import code that works on your computer might not work on someone else's.

```
library(readr)
system.time(dd1 <- read_delim("../../data/genome.txt"))
```

```
user  system elapsed
7.02   0.12   7.22
```

```
system.time(dd2 <- read_delim("../../data/genome.txt",
                                delim="\t"))
```

```
user  system elapsed
0.66   0.10   0.77
```

```
dim(dd2)
```

```
[1] 733202    5
```

## head(dd1)

	Name	Chr	Position	Log.R.Ratio	B.Allele.Freq
1	rs1000000	12	125456933	-0.002501764	1.000000000
2	rs1000002	3	185118461	-0.029741180	0.000336171
3	rs10000023	4	95952928	0.004015533	0.460671800
4	rs1000003	3	99825597	-0.142527700	0.541123600
5	rs10000030	4	103593179	0.365104000	1.000000000
6	rs10000037	4	38600725	-0.005177616	0.504625300

## dd2

# A tibble: 733,202 x 5

	Name	Chr	Position	Log.R.Ratio	B.Allele.Freq
	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	rs1000000	12	125456933	-0.00250	1
2	rs1000002	3	185118461	-0.0297	0.000336
3	rs10000023	4	95952928	0.00402	0.461
4	rs1000003	3	99825597	-0.143	0.541
5	rs10000030	4	103593179	0.365	1
6	rs10000037	4	38600725	-0.00518	0.505
7	rs10000041	4	165841405	-0.179	0
8	rs10000042	4	5288053	0.168	0.998
9	rs10000049	4	119167668	-0.00238	0
10	rs1000007	2	237416793	-0.00411	0

# ... with 733,192 more rows

## Data transformation

# Data transformation

- ▶ It is rare that you get the data in exactly the right form you need.
- ▶ Often you'll need to create some new variables or summaries.
- ▶ Or maybe you just want to rename the variables or reorder the observations in order to make the data a little easier to work with.

Let us illustrate how to manage available data using NYC flights database. `nycflights13::flights` data frame contains all 336,776 flights that departed from New York City in 2013. The data comes from the US Bureau of Transportation Statistics, and is documented in `?flights`.

```
library(nycflights13)
library(tidyverse)
```



## flights

```
# A tibble: 336,776 x 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>
1	2013	1	1	517	515	2	830	819
2	2013	1	1	533	529	4	850	830
3	2013	1	1	542	540	2	923	850
4	2013	1	1	544	545	-1	1004	1022
5	2013	1	1	554	600	-6	812	837
6	2013	1	1	554	558	-4	740	728
7	2013	1	1	555	600	-5	913	854
8	2013	1	1	557	600	-3	709	723
9	2013	1	1	557	600	-3	838	846
10	2013	1	1	558	600	-2	753	745

```
# ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,  
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,  
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

## dlpyr basics

- ▶ Pick observations by their values: `filter()`.
- ▶ Reorder the rows: `arrange()`.
- ▶ 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 verbs work similarly:

- ▶ 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.

## Filter rows

```
jan1 <- filter(flights, month == 1, day == 1)
```

R either prints out the results, or saves them to a variable. If you want to do both, you can wrap the assignment in parentheses:

```
(jan1 <- filter(flights, month == 1, day == 1))
```

```
# A tibble: 842 x 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
1  2013     1     1     517             515           2       830           819
2  2013     1     1     533             529           4       850           830
3  2013     1     1     542             540           2       923           850
4  2013     1     1     544             545          -1      1004          1022
5  2013     1     1     554             600          -6       812           837
6  2013     1     1     554             558          -4       740           728
7  2013     1     1     555             600          -5       913           854
8  2013     1     1     557             600          -3       709           723
9  2013     1     1     557             600          -3       838           846
10 2013     1     1     558             600          -2       753           745
# ... with 832 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

## Logical filtering

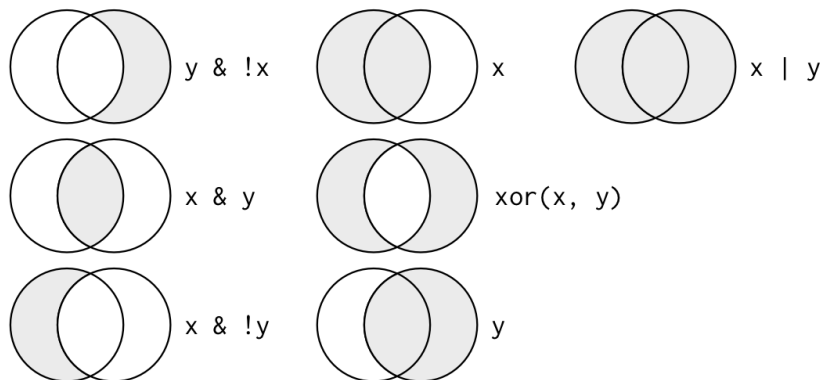


Figure 4: boolean operations

```
filter(flights, month == 11 | month == 12)
```

```
# A tibble: 55,403 x 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>
1	2013	11	1	5	2359	6	352	345
2	2013	11	1	35	2250	105	123	2356
3	2013	11	1	455	500	-5	641	651
4	2013	11	1	539	545	-6	856	827
5	2013	11	1	542	545	-3	831	855
6	2013	11	1	549	600	-11	912	923
7	2013	11	1	550	600	-10	705	659
8	2013	11	1	554	600	-6	659	701
9	2013	11	1	554	600	-6	826	827
10	2013	11	1	554	600	-6	749	751

```
# ... with 55,393 more rows, and 11 more variables: arr_delay <dbl>,  
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,  
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
```

```
# A tibble: 316,050 x 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>
1	2013	1	1	517	515	2	830	819
2	2013	1	1	533	529	4	850	830
3	2013	1	1	542	540	2	923	850
4	2013	1	1	544	545	-1	1004	1022
5	2013	1	1	554	600	-6	812	837
6	2013	1	1	554	558	-4	740	728
7	2013	1	1	555	600	-5	913	854
8	2013	1	1	557	600	-3	709	723
9	2013	1	1	557	600	-3	838	846
10	2013	1	1	558	600	-2	753	745

```
# ... with 316,040 more rows, and 11 more variables: arr_delay <dbl>,  
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,  
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

## Arrange rows

```
arrange(flights, year, month, day)
```

```
# A tibble: 336,776 x 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>
1	2013	1	1	517	515	2	830	819
2	2013	1	1	533	529	4	850	830
3	2013	1	1	542	540	2	923	850
4	2013	1	1	544	545	-1	1004	1022
5	2013	1	1	554	600	-6	812	837
6	2013	1	1	554	558	-4	740	728
7	2013	1	1	555	600	-5	913	854
8	2013	1	1	557	600	-3	709	723
9	2013	1	1	557	600	-3	838	846
10	2013	1	1	558	600	-2	753	745

```
# ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,  
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,  
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```



```
arrange(flights, desc(dep_delay))
```

```
# A tibble: 336,776 x 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>
1	2013	1	9	641	900	1301	1242	1530
2	2013	6	15	1432	1935	1137	1607	2120
3	2013	1	10	1121	1635	1126	1239	1810
4	2013	9	20	1139	1845	1014	1457	2210
5	2013	7	22	845	1600	1005	1044	1815
6	2013	4	10	1100	1900	960	1342	2211
7	2013	3	17	2321	810	911	135	1020
8	2013	6	27	959	1900	899	1236	2226
9	2013	7	22	2257	759	898	121	1026
10	2013	12	5	756	1700	896	1058	2020

```
# ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,  
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,  
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

NOTE: missing values are located at the end

# Select columns

```
select(flights, year, month, day)
```

```
# A tibble: 336,776 x 3
  year month   day
  <int> <int> <int>
1  2013     1     1
2  2013     1     1
3  2013     1     1
4  2013     1     1
5  2013     1     1
6  2013     1     1
7  2013     1     1
8  2013     1     1
9  2013     1     1
10 2013     1     1
# ... with 336,766 more rows
```

```
select(flights, year:day)
```

```
# A tibble: 336,776 x 3
   year month   day
  <int> <int> <int>
1  2013     1     1
2  2013     1     1
3  2013     1     1
4  2013     1     1
5  2013     1     1
6  2013     1     1
7  2013     1     1
8  2013     1     1
9  2013     1     1
10 2013     1     1
# ... with 336,766 more rows
```

```
select(flights, -(year:day))
```

```
# A tibble: 336,776 x 16
```

	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier
	<int>	<int>	<dbl>	<int>	<int>	<dbl>	<chr>
1	517	515	2	830	819	11	UA
2	533	529	4	850	830	20	UA
3	542	540	2	923	850	33	AA
4	544	545	-1	1004	1022	-18	B6
5	554	600	-6	812	837	-25	DL
6	554	558	-4	740	728	12	UA
7	555	600	-5	913	854	19	B6
8	557	600	-3	709	723	-14	EV
9	557	600	-3	838	846	-8	B6
10	558	600	-2	753	745	8	AA

```
# ... with 336,766 more rows, and 9 more variables: flight <int>,
```

```
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
```

```
#   hour <dbl>, minute <dbl>, time_hour <dtm>
```

There are a number of helper functions you can use within `select()`:

- ▶ `starts_with("abc")`: matches names that begin with "abc".
- ▶ `ends_with("xyz")`: matches names that end with "xyz".
- ▶ `contains("ijk")`: matches names that contain "ijk".
- ▶ `matches("(.)\\1")`: selects variables that match a regular expression. This one matches any variables that contain repeated characters. You'll learn more about regular expressions in strings.
- ▶ `num_range("x", 1:3)`: matches `x1`, `x2` and `x3`.

## Add new variables

```
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
```

# A tibble: 336,776 x 9

	year	month	day	dep_delay	arr_delay	distance	air_time	gain	speed
	<int>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2013	1	1	2	11	1400	227	-9	370.
2	2013	1	1	4	20	1416	227	-16	374.
3	2013	1	1	2	33	1089	160	-31	408.
4	2013	1	1	-1	-18	1576	183	17	517.
5	2013	1	1	-6	-25	762	116	19	394.
6	2013	1	1	-4	12	719	150	-16	288.
7	2013	1	1	-5	19	1065	158	-24	404.
8	2013	1	1	-3	-14	229	53	11	259.
9	2013	1	1	-3	-8	944	140	5	405.
10	2013	1	1	-2	8	733	138	-10	319.

# ... with 336,766 more rows

If you only want to keep the new variables, use `transmute()`:

```
transmute(flights,  
  gain = dep_delay - arr_delay,  
  hours = air_time / 60,  
  gain_per_hour = gain / hours  
)
```

```
# A tibble: 336,776 x 3  
   gain hours gain_per_hour  
   <dbl> <dbl>         <dbl>  
1    -9  3.78          -2.38  
2   -16  3.78          -4.23  
3   -31  2.67         -11.6  
4    17  3.05           5.57  
5    19  1.93           9.83  
6   -16  2.5           -6.4  
7   -24  2.63          -9.11  
8    11  0.883          12.5  
9     5  2.33           2.14  
10  -10  2.3           -4.35  
# ... with 336,766 more rows
```

## Grouped summaries

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 1 x 1  
  delay  
  <dbl>  
1  12.6
```



```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 365 x 4
# Groups:   year, month [12]
   year month   day delay
  <int> <int> <int> <dbl>
1  2013     1     1  11.5
2  2013     1     2  13.9
3  2013     1     3  11.0
4  2013     1     4   8.95
5  2013     1     5   5.73
6  2013     1     6   7.15
7  2013     1     7   5.42
8  2013     1     8   2.55
9  2013     1     9   2.28
10 2013     1    10   2.84
# ... with 355 more rows
```

## The pipe %>%

Imagine that we want to explore the relationship between the distance and average delay for each location. The steps are:

There are three steps to prepare this data:

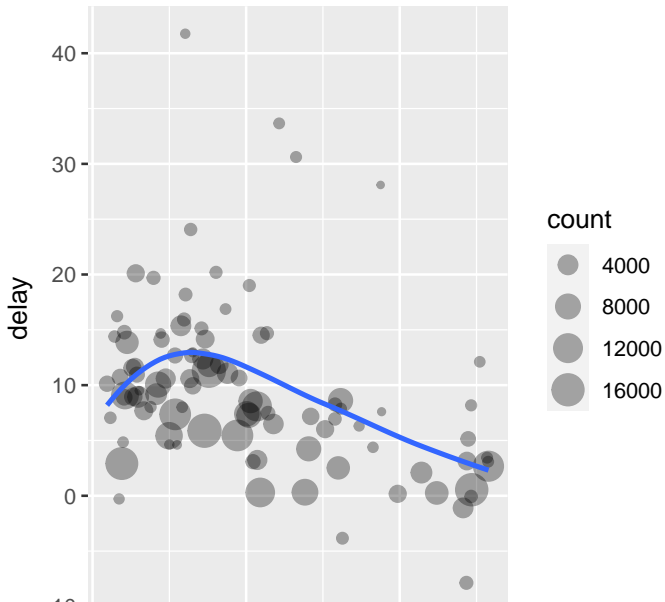
- ▶ Group flights by destination.
- ▶ Summarise to compute distance, average delay, and number of flights.
- ▶ Filter to remove noisy points and Honolulu airport, which is almost twice as far away as the next closest airport.

Using what you know about dplyr, you might write code like this:

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")
delay
```

```
# A tibble: 96 x 4
  dest   count   dist delay
  <chr> <int> <dbl> <dbl>
1 ABQ     254 1826   4.38
2 ACK     265  199   4.85
3 ALB     439  143  14.4
4 ATL    17215  757. 11.3
5 AUS    2439 1514.   6.02
6 AVL     275  584.   8.00
7 BDL     443  116   7.05
8 BGR     375  378   8.03
9 BHM     297  866. 16.9
10 BNA    6333  758. 11.8
# ... with 86 more rows
```

```
ggplot(data = delay, mapping = aes(x = dist, y = delay)) +  
  geom_point(aes(size = count), alpha = 1/3) +  
  geom_smooth(se = FALSE)
```



```
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
delays
```

```
# A tibble: 96 x 4
  dest   count   dist delay
  <chr> <int> <dbl> <dbl>
1 ABQ     254 1826   4.38
2 ACK     265  199   4.85
3 ALB     439  143  14.4
4 ATL   17215  757.  11.3
5 AUS    2439 1514.   6.02
6 AVL     275  584.   8.00
7 BDL     443  116   7.05
8 BGR     375  378   8.03
9 BHM     297  866.  16.9
10 BNA    6333  758.  11.8
# ... with 86 more rows
```

## Group by different variables

```
flights %>%  
  group_by(year, month, day) %>%  
  summarise(  
    avg_delay1 = mean(arr_delay, na.rm=TRUE),  
    avg_delay2 = mean(arr_delay[arr_delay > 0], na.rm=TRUE)  
  )
```

```
# A tibble: 365 x 5  
# Groups:   year, month [12]  
   year month   day avg_delay1 avg_delay2  
   <int> <int> <int>      <dbl>      <dbl>  
1  2013     1     1      12.7       32.5  
2  2013     1     2      12.7       32.0  
3  2013     1     3       5.73      27.7  
4  2013     1     4      -1.93      28.3  
5  2013     1     5      -1.53      22.6  
6  2013     1     6       4.24      24.4  
7  2013     1     7      -4.95      27.8  
8  2013     1     8      -3.23      20.8  
9  2013     1     9      -0.264      25.6  
10 2013     1    10      -5.90      27.3  
# ... with 355 more rows
```

## Useful summary functions

- ▶ `count()`
- ▶ `mean()`
- ▶ `median()`
- ▶ `min()`
- ▶ `max()`
- ▶ `quantile(x, 0.25)`
- ▶ `IQR()`
- ▶ `mad()`

## Exercises (data transform)

1. Using `flights` dataset, find all flights that
  - ▶ Had an arrival delay of two or more hours
  - ▶ Flew to Houston (IAH or HOU)
    - ▶ Arrived more than two hours late, but didn't leave late
  - ▶ Were delayed by at least an hour, but made up over 30 minutes in flight
2. Another useful dplyr filtering helper is `between()`. What does it do? Can you use it to simplify the code needed to answer the previous challenges?
3. Sort flights to find the fastest flights.
4. Create a new data frame having variables with the dep string.
5. Create a new data frame having the hour and minute of departure (Hint: information is in the variable `dep_time` with format HHMM or HMM. Use `%/%` or `%%` when appropriate)
6. Create a summary of each airline (variable `carrier`) describing the total number of flights, the average, median, IQR and



# Session info

## sessionInfo()

R version 4.0.2 (2020-06-22)  
Platform: x86\_64-w64-mingw32/x64 (64-bit)  
Running under: Windows 10 x64 (build 18362)

Matrix products: default

locale:

[1] LC\_COLLATE=Spanish\_Spain.1252 LC\_CTYPE=Spanish\_Spain.1252  
[3] LC\_MONETARY=Spanish\_Spain.1252 LC\_NUMERIC=C  
[5] LC\_TIME=Spanish\_Spain.1252

attached base packages:

[1] stats graphics grDevices utils datasets methods base

other attached packages:

[1] nycflights13\_1.0.1 forcats\_0.5.0 stringr\_1.4.0 dplyr\_1.0.2  
[5] purrr\_0.3.4 readr\_1.3.1 tidyr\_1.1.2 tibble\_3.0.3  
[9] ggplot2\_3.3.2 tidyverse\_1.3.0

loaded via a namespace (and not attached):

[1] tidyselect\_1.1.0 xfun\_0.16 lattice\_0.20-41 splines\_4.0.2  
[5] haven\_2.3.1 colorspace\_1.4-1 vctrs\_0.3.3 generics\_0.0.2  
[9] htmltools\_0.5.0 mgcv\_1.8-33 yaml\_2.2.1 utf8\_1.1.4  
[13] blob\_1.2.1 rlang\_0.4.7 pillar\_1.4.6 glue\_1.4.2  
[17] withr\_2.2.0 DBI\_1.1.0 dbplyr\_1.4.4 modelr\_0.1.8  
[21] readxl\_1.3.1 lifecycle\_0.2.0 munsell\_0.5.0 gtable\_0.3.0  
[25] cellranger\_1.1.0 rvest\_0.3.6 codetools\_0.2-16 evaluate\_0.14  
[29] labeling\_0.3 knitr\_1.29 fansi\_0.4.1 broom\_0.7.0  
[33] Rcpp\_1.0.5 scales\_1.1.1 backports\_1.1.9 jsonlite\_1.7.0  
[37] farver\_2.0.3 fs\_1.5.0 hms\_0.5.3 digest\_0.6.25