```r
library(dplyr)

rladies_global %>%
  filter(city == 'Bari')
```

Manipolare i dati con Tidyverse

Hello!

Welcome to R-Ladies Bari

1.
Introduction

R language, RStudio

Three things
you’ll need to install

1. **Install R** -- this is the open-source programming language we’ll use (download via CRAN -- Comprehensive R Archive Network)

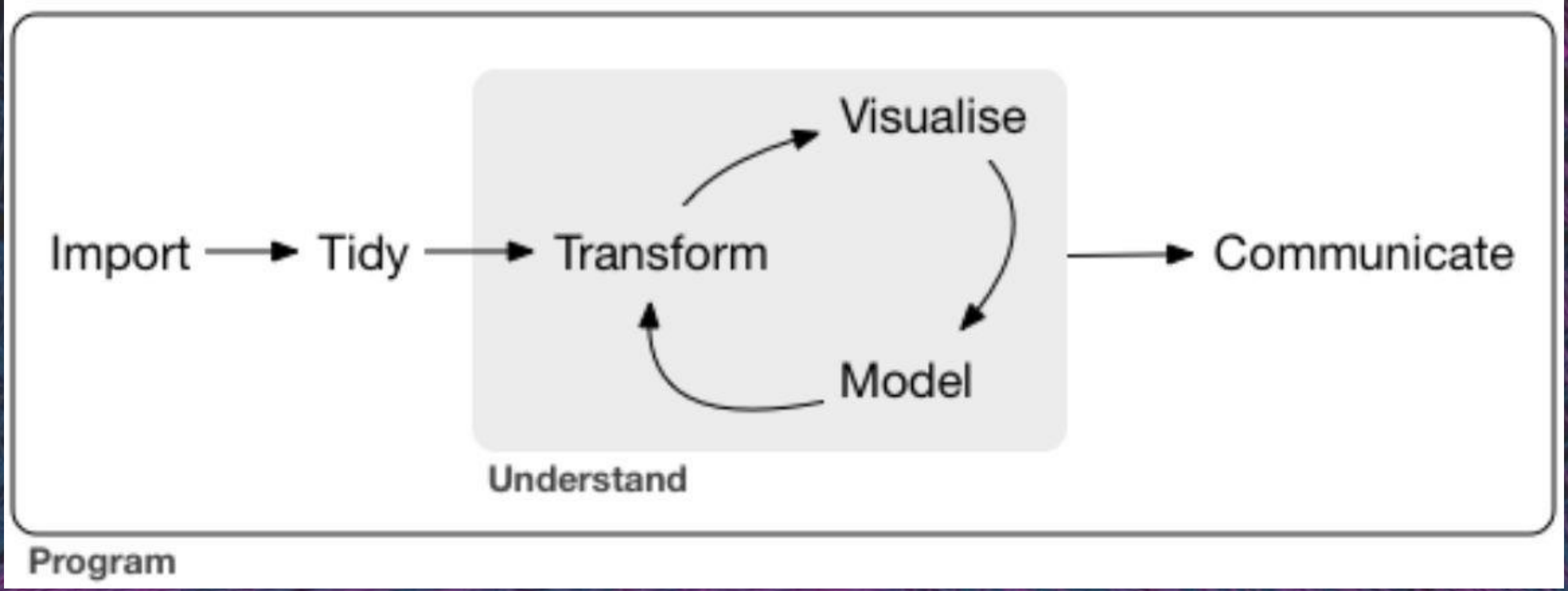
2. **Install RStudio** -- this is the most popular IDE for R and will make your life a lot easier (download from rstudio.com/download)

3. **Install the tidyverse** -- this is the group of packages we’ll use within R to work with data.

   Install with one line of code in R: `install.packages(“tidyverse”)`

2.
Manipolare i dati con Tidyverse

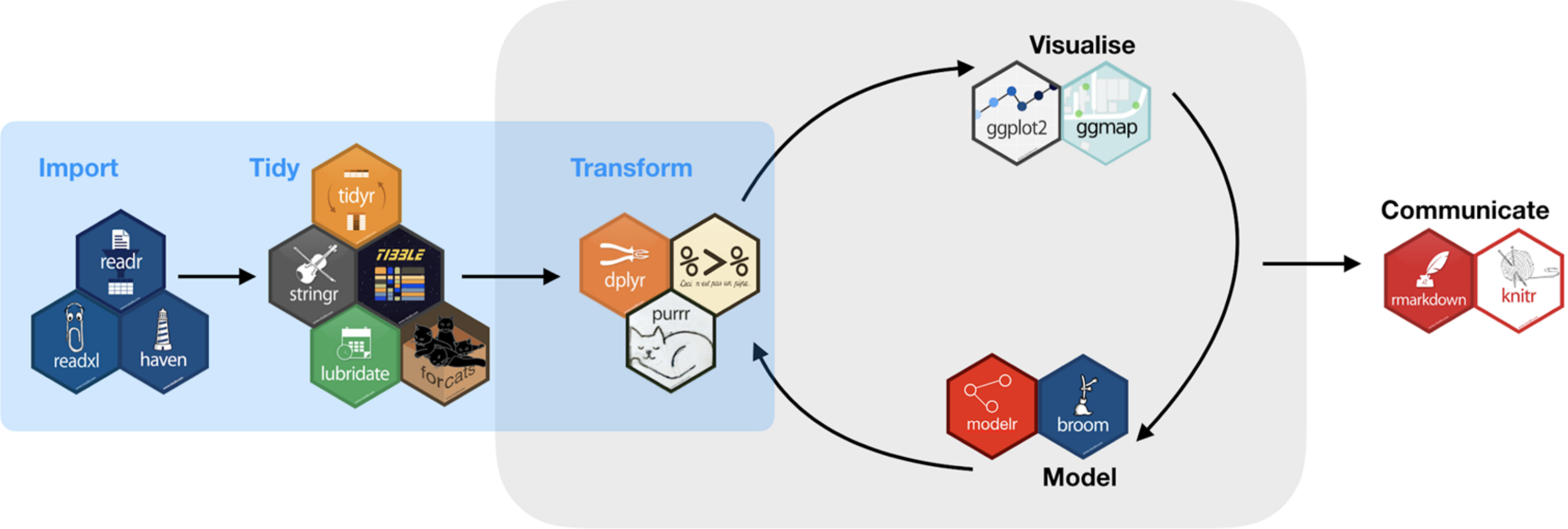
The data science process (tidied)



What is
the tidyverse?

- **Collection of R packages** based on tidy data principles
- Designed to **work together**
- An **easier** way to code!
- AKA “Hadleyverse” (most packages written by Hadley Wickham)

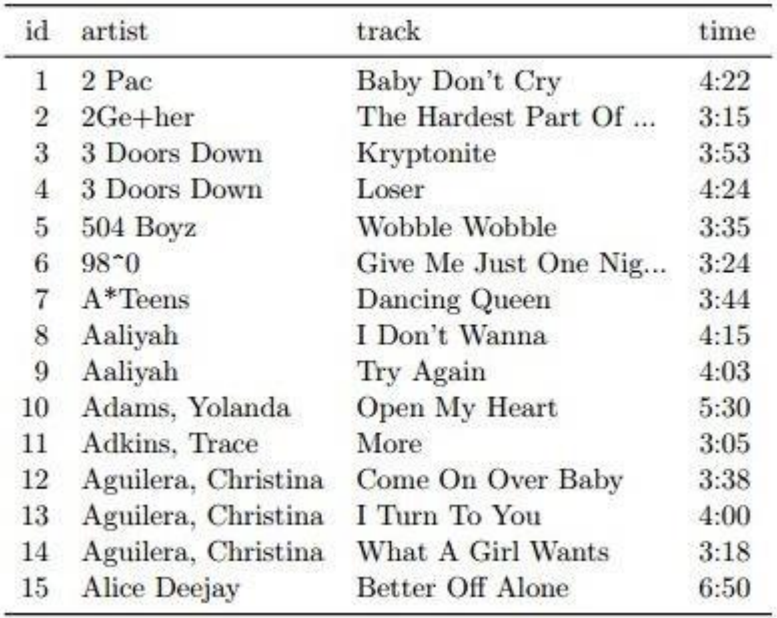
The data science process
with tidyverse



https://www.tidyverse.org/

What is
tidy data?

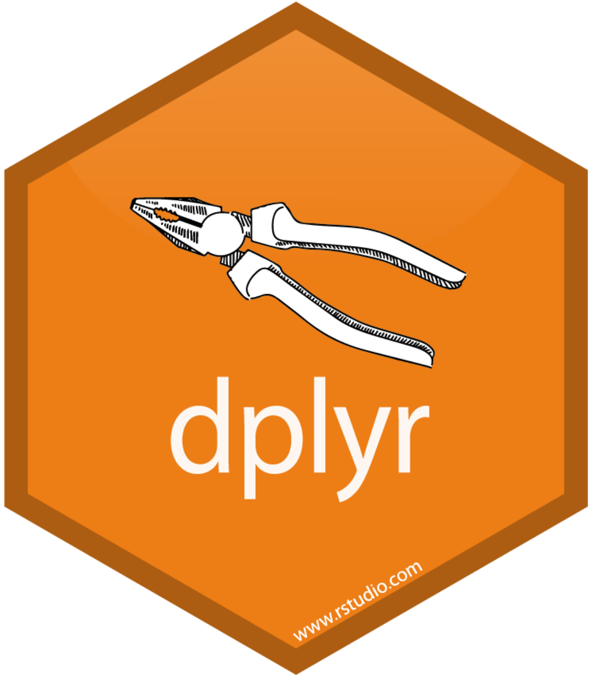
- Each **variable** is a column
- Each observation is a row
- Each type of observational unit is a table



id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98^0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

3.
dplyr

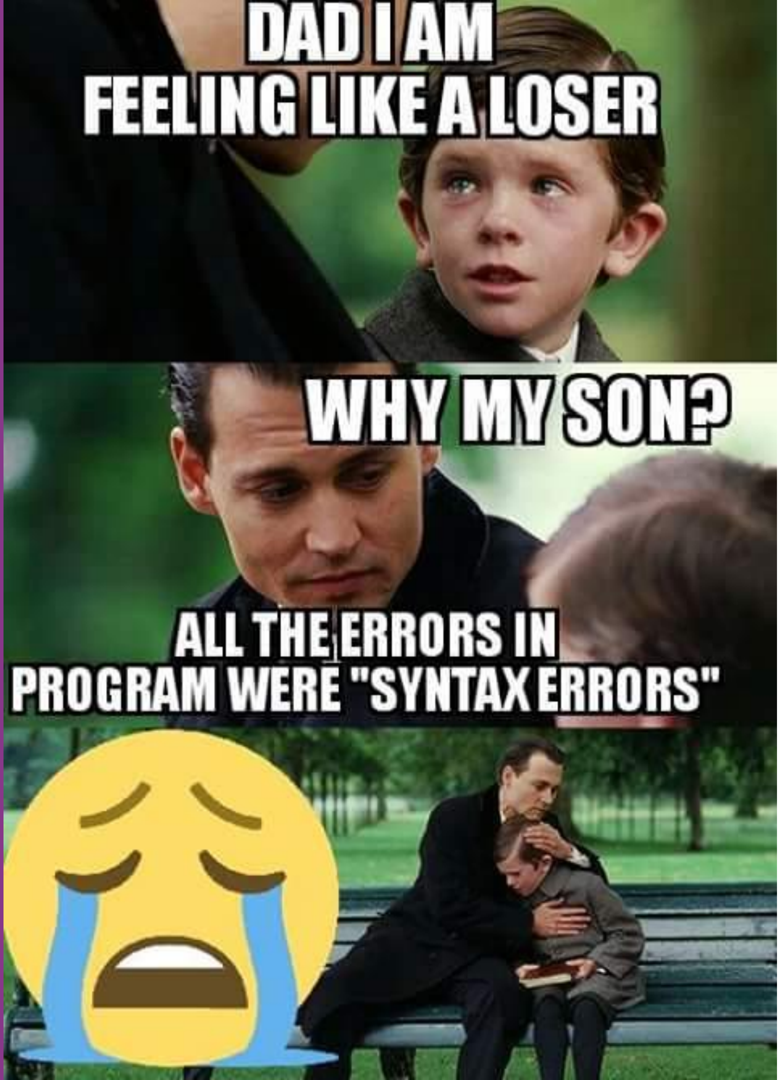
Let’s start with the first set of slides



dplyr

grammar of data manipulation

+ `mutate()` to create new variables from existing ones
+ `select()` picks variables based on their names
+ `filter()` allows pointed selection based on given criteria
+ `summarise()` reduces multiple values down to a single summary
+ `arrange()` changes the ordering of rows
+ `group_by()` performs any of the above on a group-by-group basis


DAD I AM FEELING LIKE A LOSER
WHY MY SON?
ALL THE ERRORS IN PROGRAM WERE "SYNTAX ERRORS"

dplyr
syntax

- All **calls** to dplyr verbs follow the same **format**:
  1. The first argument is a **dataframe**
  2. The subsequent arguments describe **what to do** to that dataframe, using unquoted variable names.

- Each call returns a new dataframe (rather than overwriting the 'old' one)

- Example:

```
filter(dataset_name,
          name == "Francesca")
```

What is magrittr?

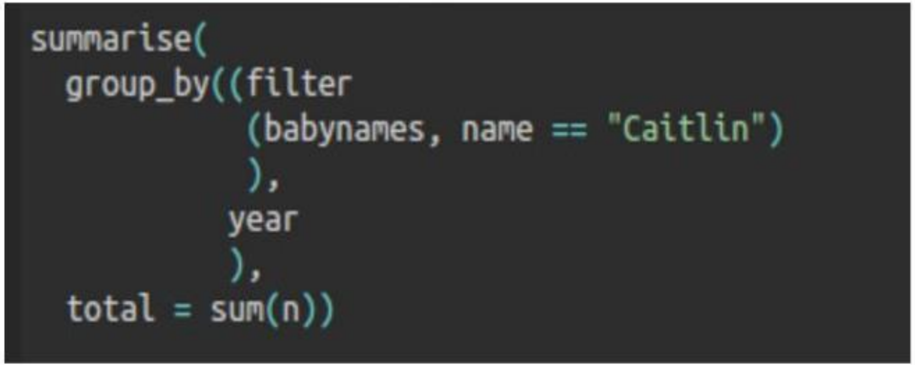
Simplifying R code with **pipes** (`%>%`)

- Easy way to pass data through functions without nesting
- First argument of each function is “piped” in to reduce redundancy

dplyr + magrittr

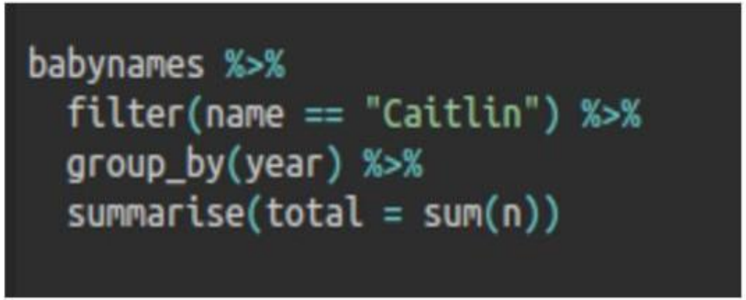
example

before



```
summarise(
  group_by((filter
            (babynames, name == "Caitlin")
            ),
           year
           ),
  total = sum(n))
```

after



```
babynames %>%
  filter(name == "Caitlin") %>%
  group_by(year) %>%
  summarise(total = sum(n))
```

Quick aside:
iris dataset

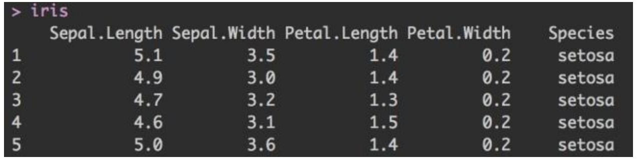
- Included in R (iris to view)
- 150 observations of 5 variables:
  iris type, sepal length, sepal width, petal length and petal width

select()

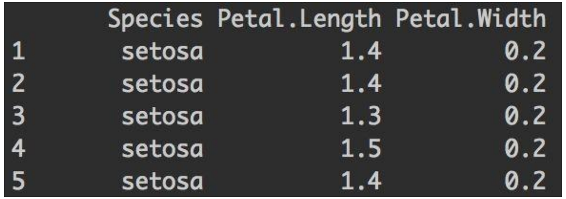
+ Picks variables based on their names

+ First argument is dataframe; subsequent arguments represent columns to select

```
iris %>% select(Species, Petal.Length, Petal.Width)
```



```
> 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
```

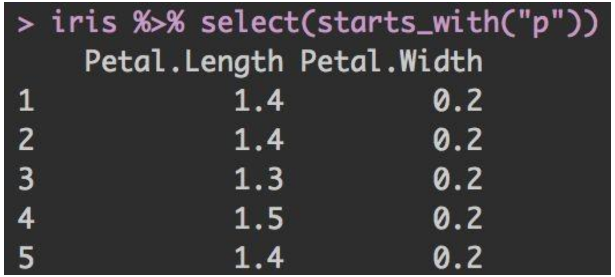


```
  Species Petal.Length Petal.Width
1  setosa          1.4         0.2
2  setosa          1.4         0.2
3  setosa          1.3         0.2
4  setosa          1.5         0.2
5  setosa          1.4         0.2
```

select() +
helper functions

Helper functions you can use within **select()**:

+ `starts_with(“a”)` matches names that begin with “a”
+ `ends_with(“z”)` matches names that begin with “z”
+ `contains(“lady”)` matches names that contain “lady”
+ `matches(<regex>)` allows you to do regex matching on names

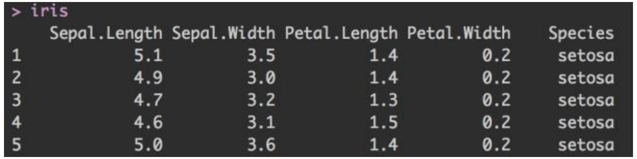


```
> iris %>% select(starts_with("p"))
  Petal.Length Petal.Width
1          1.4         0.2
2          1.4         0.2
3          1.3         0.2
4          1.5         0.2
5          1.4         0.2
```

filter()

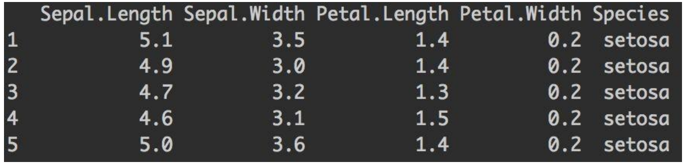
- Allows pointed selection based on given criteria
- First argument is the dataframe, subsequent arguments are expressions used to filter the dataframe

```
iris %>% filter(Species ==“setosa”)
```



```
> 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
```

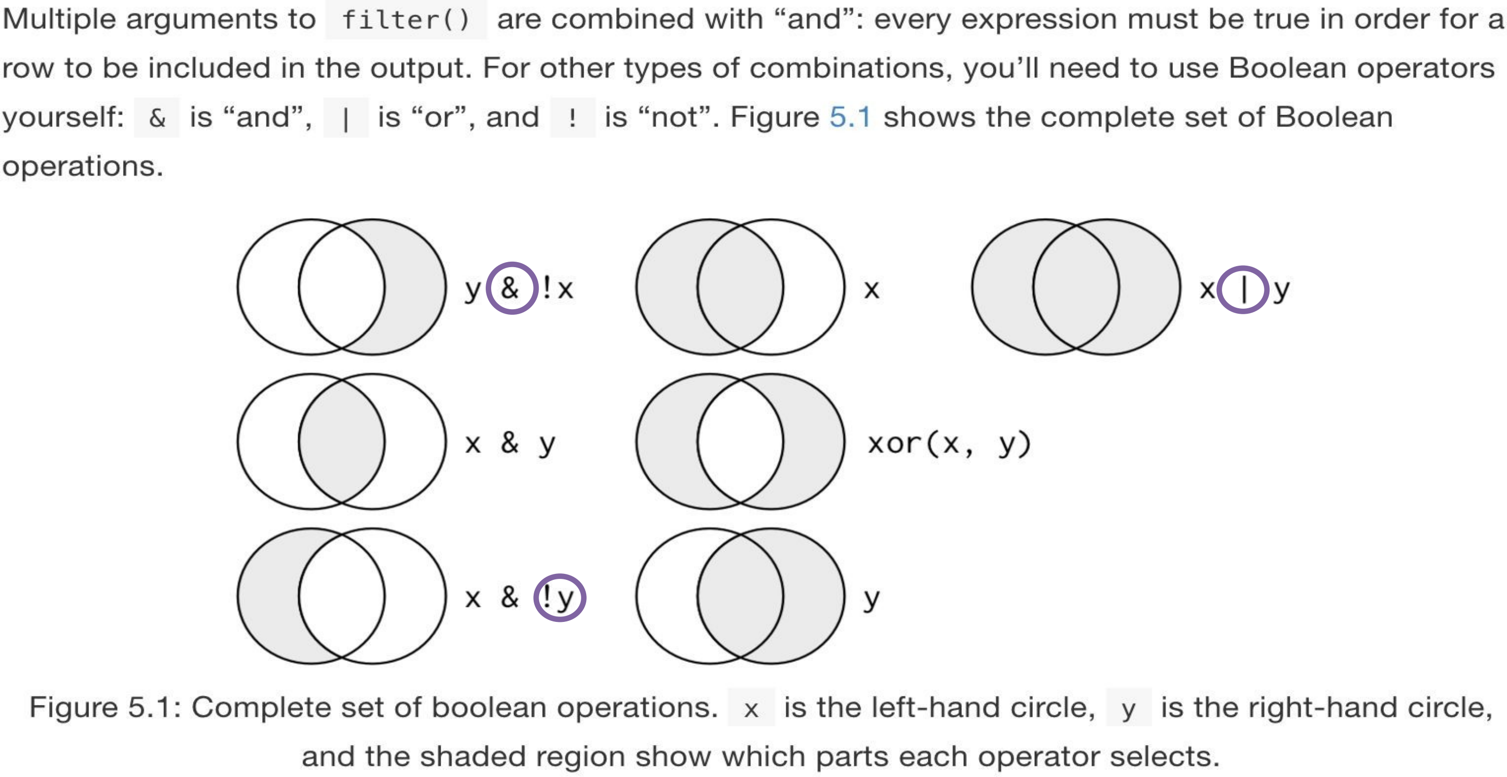
nrow = 150



```
  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
```

nrow = 50

filter() + booleans



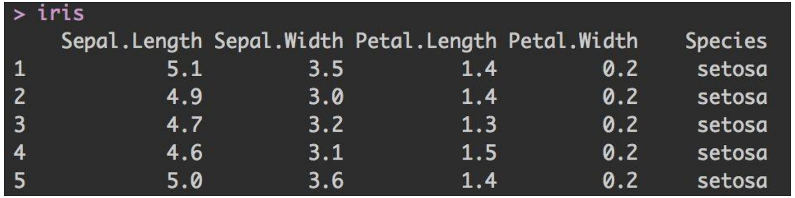
Multiple arguments to `filter()` are combined with “and”: every expression must be true in order for a row to be included in the output. For other types of combinations, you’ll need to use Boolean operators yourself: `&` is “and”, `|` is “or”, and `!` is “not”. Figure 5.1 shows the complete set of Boolean operations.

Figure 5.1: Complete set of boolean operations. `x` is the left-hand circle, `y` is the right-hand circle, and the shaded region show which parts each operator selects.

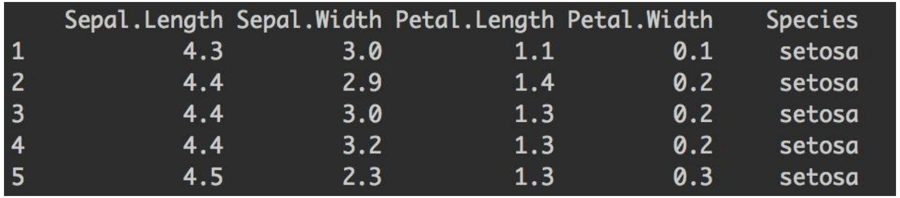
arrange()

- Changes the ordering of rows
- First argument is the dataframe, subsequent arguments are columns and/or expressions used to re-arrange the dataframe
- Note: default is ascending order, and NA's are always at the end

```
iris %>% arrange(Sepal.Length, Sepal.Width)
```



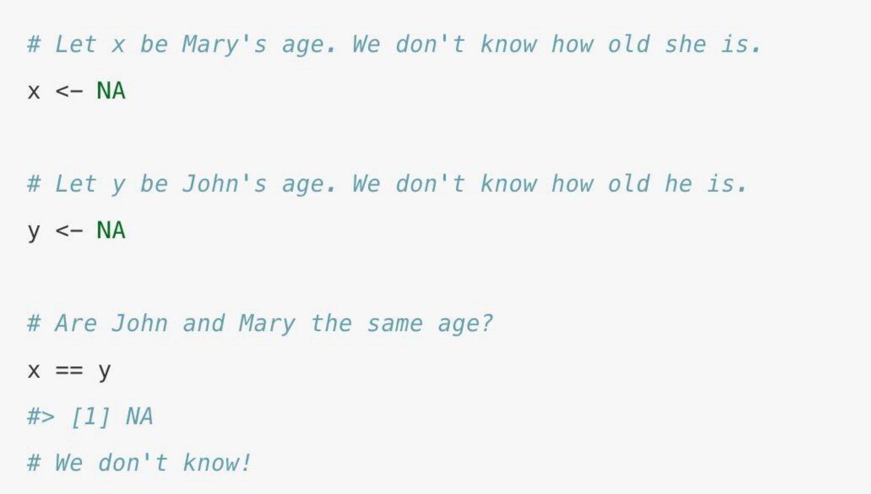
```
> iris
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          5.1         3.5          1.4         0.2     setosa
2          4.9         3.0          1.4         0.2     setosa
3          4.7         3.2          1.3         0.2     setosa
4          4.6         3.1          1.5         0.2     setosa
5          5.0         3.6          1.4         0.2     setosa
```



```
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          4.3         3.0          1.1         0.1     setosa
2          4.4         2.9          1.4         0.2     setosa
3          4.4         3.0          1.3         0.2     setosa
4          4.4         3.2          1.3         0.2     setosa
5          4.5         2.3          1.3         0.3     setosa
```

Quick aside:
Missing values

- *NA* represents a *missing (unknown)* value
- Comparisons involve unknown values typically result in unknown values
- To see whether a value is missing, use `is.na()`
- `filter()` only includes rows where the condition is **true** (not false or *NA*)

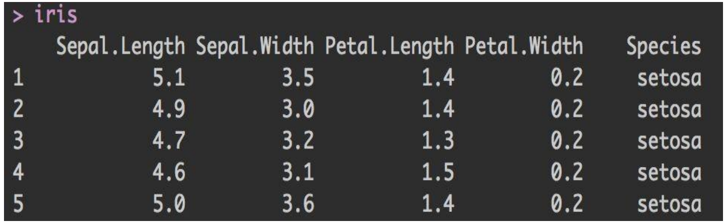


```
# Let x be Mary's age. We don't know how old she is.
x <- NA

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

# Are John and Mary the same age?
x == y
#> [1] NA
# We don't know!
```

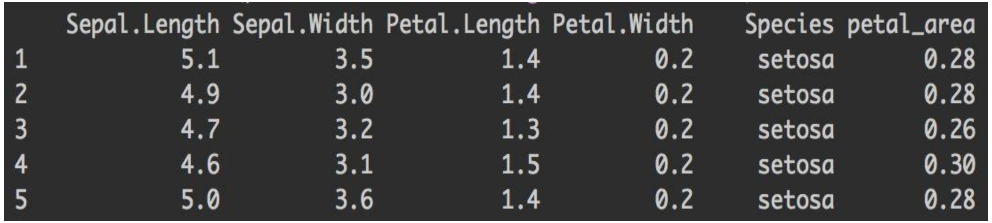
mutate()

- Creates new variables from existing ones
- Note: columns created with **mutate()** are always added to end of dataset

```
iris %>%  mutate(petal_area = Petal.Length * Petal.Width)
```



```
> 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
```



```
  Sepal.Length Sepal.Width Petal.Length Petal.Width  Species petal_area
1          5.1         3.5          1.4         0.2   setosa       0.28
2          4.9         3.0          1.4         0.2   setosa       0.28
3          4.7         3.2          1.3         0.2   setosa       0.26
4          4.6         3.1          1.5         0.2   setosa       0.30
5          5.0         3.6          1.4         0.2   setosa       0.28
```

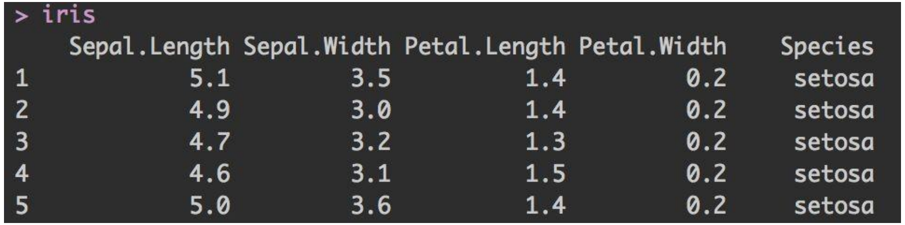
mutate()
Useful functions

- Arithmetic operators (`+`, `-`, `*`, `/`, `^`)
- Log functions (like `log10()`)
- Offsets like `lead()` and `lag()`
- Logical comparisons (`<`, `<=`, `>`, `>=`, `!=`)
- `Ifelse()` statements (*if this, then this, else this*)
- Cumulative and rolling aggregates
- Ranking (like `ntile()`)

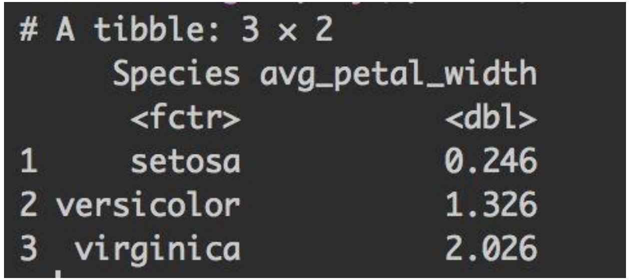
group_by() and summarise()

- group_by applies dplyr verbs by group
- summarise reduces multiple values down to a single summary

```
iris %>%
    group_by(Species) %>%
    summarise(avg_petal_width = mean(Petal.Width)
```



```
> 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
```



```
# A tibble: 3 × 2
     Species avg_petal_width
      <fctr>           <dbl>
1     setosa           0.246
2 versicolor           1.326
3  virginica           2.026
```

summarise()

Useful functions

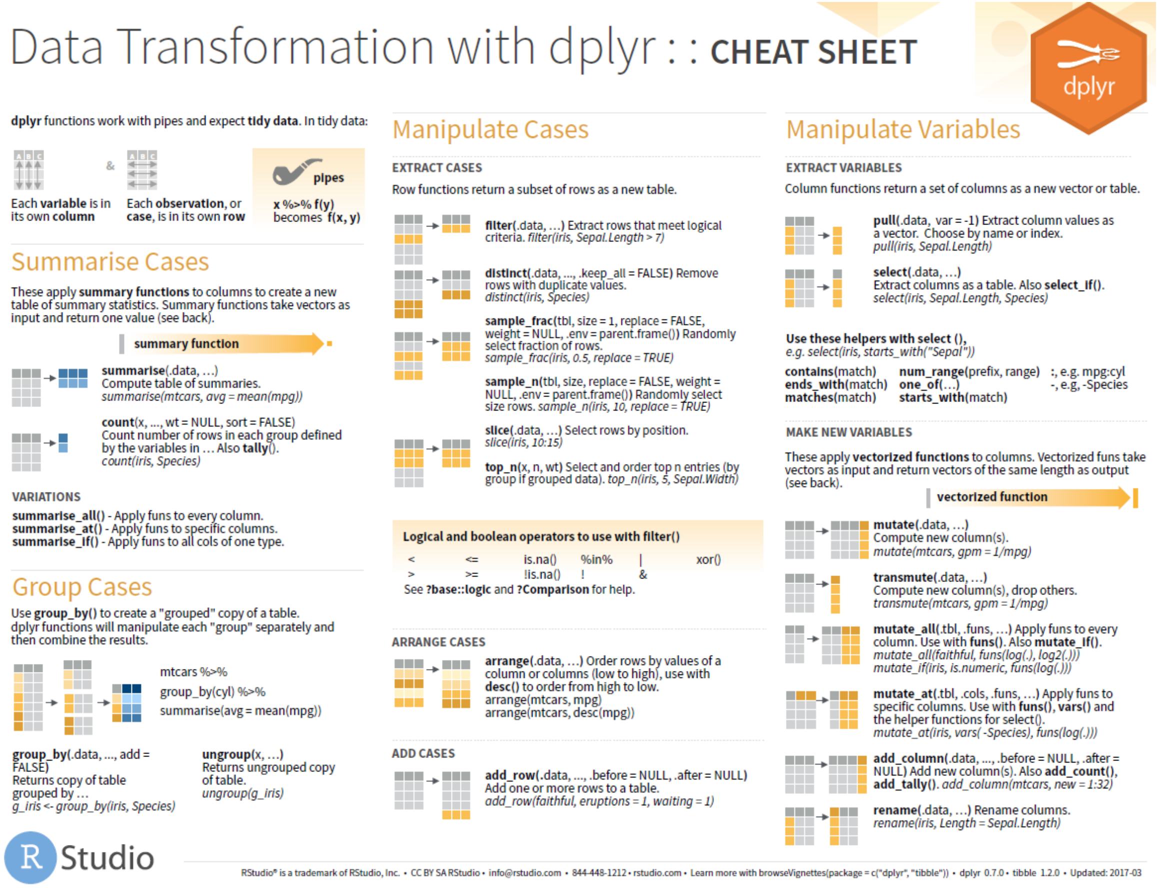
- Counts (`n()`, `n_distinct()`)
- Measures of location (`mean()`, `median()`)
- Measures of spread (`sd()`, `IQR()`)
- Measures of position (`first()`, `last()`)

Tips & Tricks

- If you don't have the result of a dplyr chain to a dataframe, it will print
- If you want to print and save, wrap assignment in parenthesis

  Example: `(iris_names <- iris %>% filter(Species == "setosa"))`
- `rename()` is a cool function to clean up messy column names
- After grouping with `group_by()`, you can `ungroup()` to remove groupings
- There is a cheat sheet for data wrangling!

Cheat sheet for data wrangling



Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …) Compute table of summaries. *summarise(mtcars, avg = mean(mpg))*

count(x, ..., wt = NULL, sort = FALSE) Count number of rows in each group defined by the variables in … Also **tally**(). *count(iris, Species)*

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by**() to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

group_by(.data, ..., add = FALSE) Returns copy of table grouped by … *g_iris <- group_by(iris, Species)*

ungroup(x, …) Returns ungrouped copy of table. *ungroup(g_iris)*

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values. *distinct(iris, Species)*

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows. *sample_frac(iris, 0.5, replace = TRUE)*

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position. *slice(iris, 10:15)*

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc**() to order from high to low. arrange(mtcars, mpg) arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table. *add_row(faithful, eruptions = 1, waiting = 1)*

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index. *pull(iris, Sepal.Length)*

select(.data, …) Extract columns as a table. Also **select_if**(). *select(iris, Sepal.Length, Species)*

Use these helpers with select (), *e.g. select(iris, starts_with("Sepal"))*

contains(match)	**num_range**(prefix, range)	:, e.g. mpg:cyl
ends_with(match)	**one_of**(…)	-, e.g, -Species
matches(match)	**starts_with**(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …) Compute new column(s). *mutate(mtcars, gpm = 1/mpg)*

transmute(.data, …) Compute new column(s), drop others. *transmute(mtcars, gpm = 1/mpg)*

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs**(). Also **mutate_if**(). *mutate_all(faithful, funs(log(.), log2(.)))* *mutate_if(iris, is.numeric, funs(log(.)))*

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs**(), **vars**() and the helper functions for select(). *mutate_at(iris, vars(-Species), funs(log(.)))*

add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count**(), **add_tally**(). *add_column(mtcars, new = 1:32)*

rename(.data, …) Rename columns. *rename(iris, Length = Sepal.Length)*

RStudio

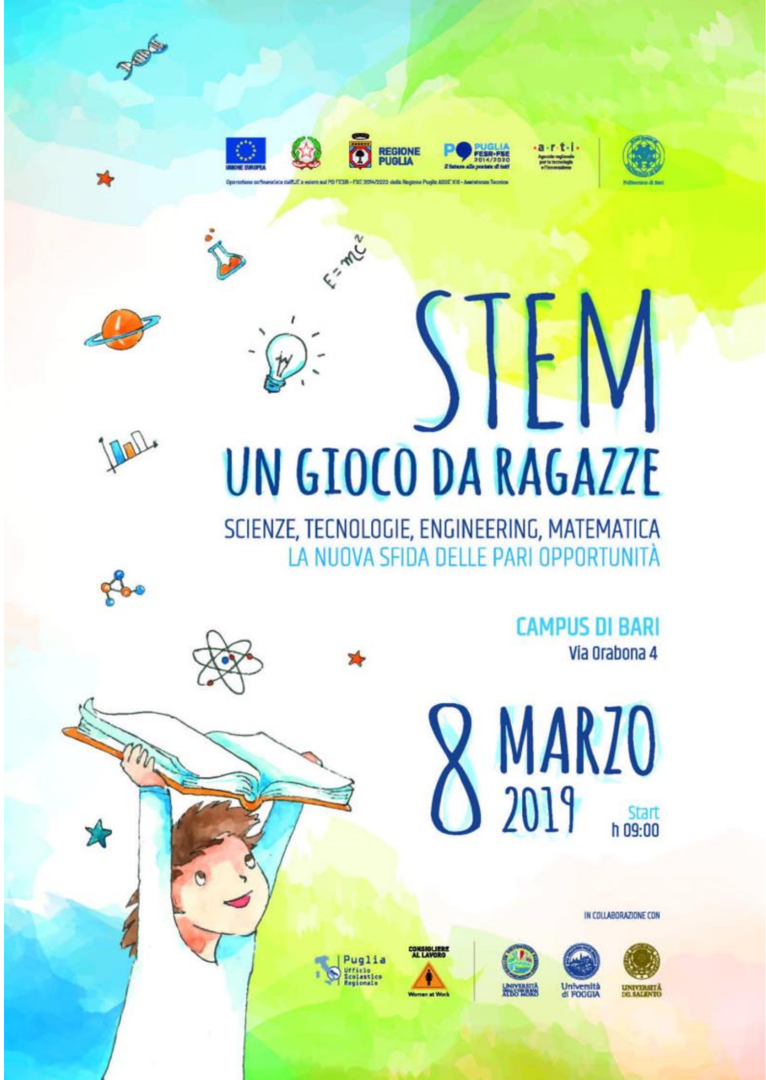
RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

4.
Wrap-up

Announcements, upcoming events, etc.

R-Ladies Bari

Upcoming Events



UNIONE EUROPEA

REGIONE PUGLIA

PO PUGLIA FESR-FSE 2014/2020

Il futuro alla portata di tutti

a·r·t·i

Agenzia regionale per la tecnologia e l'innovazione

Operazione cofinanziata dall'UE a valere sul PO FESR - FSE 2014/2020 della Regione Puglia ASSE XIII - Assistenza Tecnica

Politecnico di Bari

STEM

UN GIOCO DA RAGAZZE

SCIENZE, TECNOLOGIE, ENGINEERING, MATEMATICA

LA NUOVA SFIDA DELLE PARI OPPORTUNITÀ

CAMPUS DI BARI

Via Orabona 4

8 MARZO 2019

Start h 09:00

IN COLLABORAZIONE CON

Puglia Ufficio Scolastico Regionale

CONSIGLIERE AL LAVORO

Women at Work

UNIVERSITÀ ALDO MORO

Università di FOGGIA

UNIVERSITÀ DEL SALENTO

R-Ladies Bari
E-Learning

Esercitazioni gratuite

https://www.datacamp.com/courses/introduction-to-the-tidyverse

https://www.kaggle.com/rtatman/manipulating-data-with-the-tidyverse