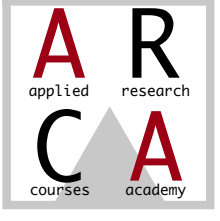
Introduzione a R

R and beyond



ARCA - @DPSS

Filipo Gambarota

What to do now? 🤔

Cosa abbiamo visto?

- I tipi di oggetti in R
- Strutture dati: come creare, accedere, manipolare
- Programmazione condizionale e iterativa (`for`, `if else`, `*apply`)
- Importare/esportare file

Un sacco di cose! 😄. Ma è tutto qui? Nope 😜

Manipolazione dataframe avanzata con `dplyr`

Possiamo utilizzare tutte le conoscenze su come manipolare i dataframe e sulle tipologie di dato con un pacchetto che ormai è diventato lo standard. `dplyr` permette di fare qualsiasi cosa con un dataframe:

https://dplyr.tidyverse.org/

- ordinare righe e colonne
- modificare colonne con operazioni complesse
- selezionare righe e colonne con operazioni complesse

Trasformare dataframe con `tidyr`

Spesso dobbiamo trasformare i dataframe in diversi formati ad esempio da formato `long` a formato `wide` e viceversa. Questa operazione è molto complessa ma `tiyr` la implementa in modo estremamente semplice. Creiamo un dataframe in formato `long`:

https://tidyr.tidyverse.org/

```
datlong <- expand.grid(
    id = 1:10,
    cond = c("a", "b", "c")
)

datlong$y <- rnorm(nrow(datlong))

head(datlong)
```

```
##   id cond           y
## 1  1    a  0.01326013
## 2  2    a  0.69350148
## 3  3    a -0.03093563
## 4  4    a -1.77298485
## 5  5    a  0.38531642
## 6  6    a -0.30612037
```

Trasformare dataframe con `tidyr`

Adesso possiamo trasformare il dataframe in formato `wide` con una semplice linea di codice con la funzione `pivot_wider()`

```
datwide <- tidyr::pivot_wider(datlong, names_from = cond, values_from = y)

head(datlong)
```

```
##   id cond           y
## 1  1    a  0.01326013
## 2  2    a  0.69350148
## 3  3    a -0.03093563
## 4  4    a -1.77298485
## 5  5    a  0.38531642
## 6  6    a -0.30612037
```

Trasformare dataframe con `tidyr`

Chiaramente possiamo anche ritornare al formato originale usando la funzione `pivot_longer()`:

```
datlong <- tidyr::pivot_longer(datwide, c(a, b, c), names_to = "cond", values_to = "y")

head(datlong)
```

```
## [90m# A tibble: 6 × 3 [39m
##      id cond        y
##    [3m [90m<int> [39m [23m  [3m [90m<chr> [39m [23m    [3m [90m<dbl> [39m [23m
## [90m1 [39m     1 a      0.013 [4m3 [24m
## [90m2 [39m     1 b      1.17
## [90m3 [39m     1 c     - [31m0 [39m [31m. [39m [31m529 [39m
## [90m4 [39m     2 a      0.694
## [90m5 [39m     2 b      0.100
## [90m6 [39m     2 c     - [31m0 [39m [31m. [39m [31m0 [39m [31m21 [4m5 [24m [39m
```

Combinare funzioni con `%>%` o `|>`

Quando vi capitera di cercare codice o soluzioni online che riguardano R, il 90% del codice avrà questi strani simboli `%>%` o `|>`. Abbiamo già imparato che questi sono operatori (e quindi funzioni). Questi operatori permettono di concatenare una serie di funzioni in modo molto intuitivo e leggibile. In altri termini `f(x)` è equivalente a `x %>% f()`. Queste si chiamano **pipes**:

```
x <- rnorm(100)
mean(x)
```

```
## [1] -0.2507534
```

```
x %>% mean()
```

```
## [1] -0.2507534
```

```
x |> mean()
```

```
## [1] -0.2507534
```

Combinare funzioni con `%>%` o `|>`

Con funzioni semplici non hanno molto senso ma con *pipeline* più complesse il codice diventa molto leggibile. Se volessimo concatenare una serie di funzioni di `dplyr` (ma funziona con tutte le altre funzioni):

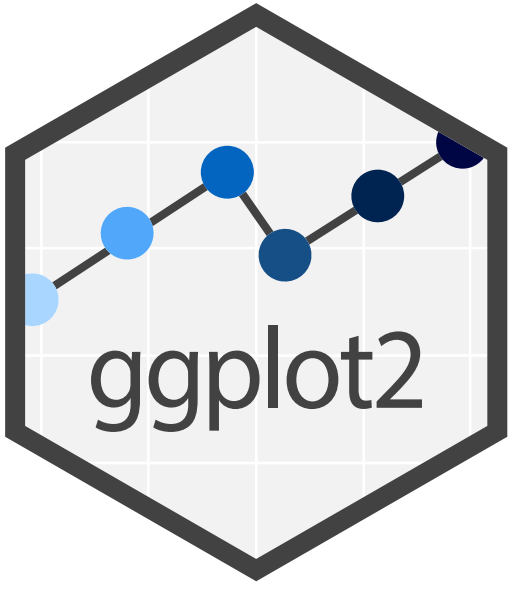
```
iris <- group_by(iris, Species)
iris <- summarise(iris, y = mean(Sepal.Width))
iris <- arrange(iris, y)
iris <- mutate(iris, z = y + 1)
iris <- filter(iris, y > 0)
```

Usando la **pipe**:

```
iris %>%
  group_by(Species) %>%
  summarise(y = mean(Sepal.Width)) %>%
  arrange(y) %>%
  mutate(z = y + 1) %>%
  filter(y > 0)
```

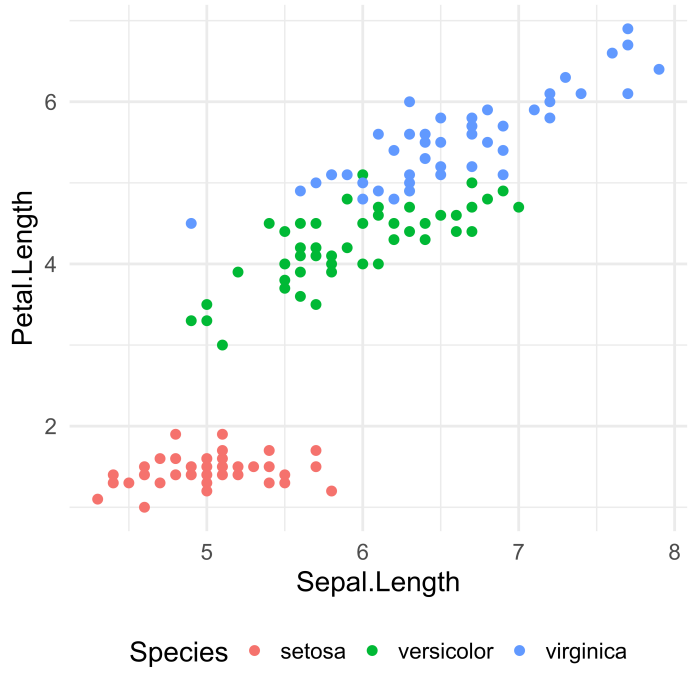
Grafici super con `ggplot2`

Un pacchetto estremamente potente che si basa su molte delle cose che abbiamo visto e sulla sintassi simil `dplyr` e **pipe-based** è `ggplot2`:



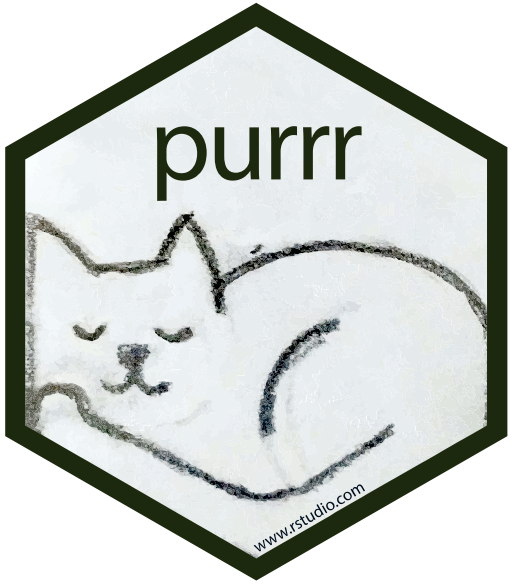
https://ggplot2.tidyverse.org/index.html

```
ggplot(iris, aes(Sepal.Length, Petal.Length, color = Species)) +
    geom_point(size = 3) +
    theme_minimal(base_size = 20) +
    theme(legend.position = "bottom")
```



Programmazione funzionale/iterativa

Il pacchetto `purrr` fornisce un insieme di funzioni che vanno a migliorare la `*apply` family con funzioni extra ed una maggiore robustezza. Se avete capito l'`*apply` family, `purrr` può essere un modo più efficiente di utilizzarle



https://tidyr.tidyverse.org/

```
sapply(mtcars, mean)
```

```
##        mpg        cyl       disp         hp       drat         wt
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
##       qsec         vs         am       gear       carb
##  17.848750   0.437500   0.406250   3.687500   2.812500
```

```
map_dbl(mtcars, mean) # controlla che l'output sia sempre numerico
```

```
##        mpg        cyl       disp         hp       drat         wt
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
##       qsec         vs         am       gear       carb
##  17.848750   0.437500   0.406250   3.687500   2.812500
```

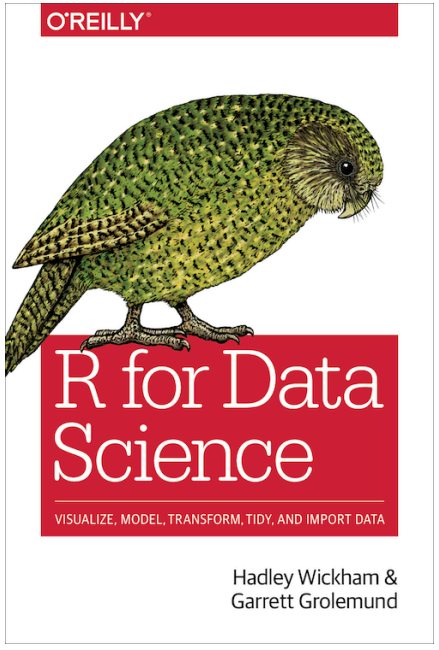
The amazing `tidyverse`

Tutto questo (e molto altro) fa parte di un insieme di pacchetti per lavorare con i dati raccolti in un meta-pacchetto chiamato `tidyverse`. Questo permette di lavorare in molto molto più consistente ed intuitivo per manipolare, analizzare e rappresentare dati di tutti i tipi.

https://www.tidyverse.org/

Come approfondire?

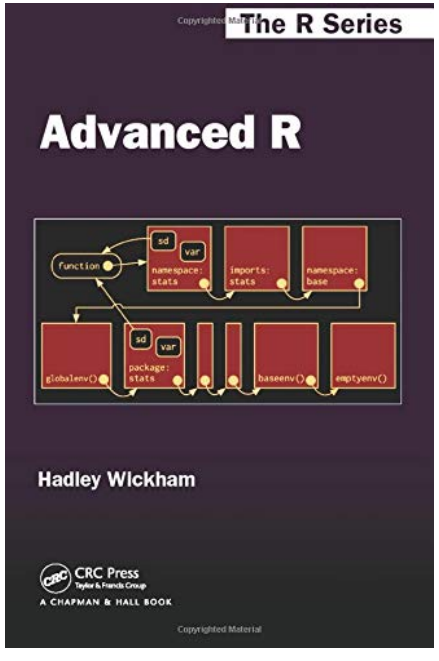
Adesso abbiamo le competenze di base per capire qualsiasi nuova funzione/pacchetto in R. Per approfondire il `tidyverse` vi consiglio:



https://r4ds.had.co.nz/

Come approfondire?

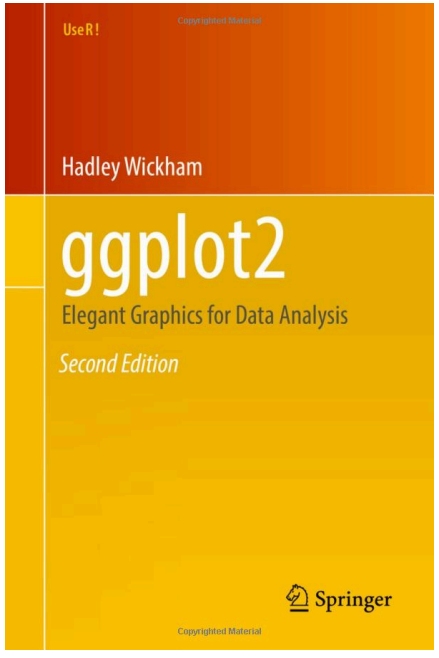
Per capire a fondo tutte le cose che abbiamo visto ed espandere al massimo la conoscenza di R come linguaggio di programmazione vi consiglio il libro **advanced R**



https://adv-r.hadley.nz/

Come approfondire?

Anche `ggplot2` ha un suo libro associato che oltre a fornire un framework teorico sulle rappresentazioni grafiche in generale chiamato *grammar of graphics* spiega nel dettaglio il funzionamento del pacchetto



https://ggplot2-book.org/

Altri argomenti utili...

Lavorare in modo avanzato con le stringhe

Le stringhe sono uno strumento molto potente da conoscere per organizzare al meglio le vostre analisi. E' anche un argomento vasto e complesso ma questi riferimenti sono un buon punto di partenza:

- Capitolo 16 del libro `introduction2R`
- Capitolo 14 del libro `R4DS`
- Regex ovvero Regular Expressions

Literate programming - R Markdown

R Markdown è uno strumento estramente potente che permette di combinare testo e codice per creare documenti, slides (queste ad esempio), siti web, tesi, articoli scientifici etc.

knitr website

https://rmarkdown.rstudio.com/

Creare tabelle

Spesso dobbiamo creare delle tabelle da inserire nei documenti o in file R Markdown. Ci sono diversi pacchetti per questo ad esempio `flextable()`, `kableExtra()` o anche `sjPlot()`.

Creiamo una tabella di statistiche descrittive:

```
df_sum <- iris |>
    group_by(Species) |>
    summarise(mean = mean(Sepal.Length),
              sd = sd(Sepal.Length),
              se = sd / sqrt(n()),
              min = min(Sepal.Length),
              max = max(Sepal.Length))
df_sum
```

```
## [90m# A tibble: 3 × 6[39m
##   Species     mean    sd     se   min   max
##   [3m[90m<fct>[39m[23m      [3m[90m<dbl>[39m[23m  [3m[90m<dbl>[39m[23m   [3m[90m<dbl>[39m[23m  [3m[90m<dbl>[39m[23m  [3m[90m<dbl>[39m[23m
## [90m1[39m setosa      5.01 0.352 0.049[4m8[24m   4.3   5.8
## [90m2[39m versicolor  5.94 0.516 0.073[4m0[24m   4.9   7
## [90m3[39m virginica   6.59 0.636 0.089[4m9[24m   4.9   7.9
```

Creare tabelle con `flextable`

`flextable` permette di creare tabelle molto complesse e di salvarle in vari formati tra cui `docx` per poterle usare poi in formato `word`. Il punto di partenza è sempre un dataframe. Il sito di `flextable` contiene tantissa documentazione.

```
df_sum |>
    flextable() |>
    theme_vanilla() |>
    autofit() |>
    colformat_double(digits = 2)
```

Species	mean	sd	se	min	max
setosa	5.01	0.35	0.05	4.30	5.80
versicolor	5.94	0.52	0.07	4.90	7.00
virginica	6.59	0.64	0.09	4.90	7.90

Creare tabelle con `kableExtra`

`kableExtra` è un altro ottimo pacchetto molto utilizzato per documenti R Markdown per creare tabelle in **html** e **pdf**. La documentazione per tabelle in **html** e **pdf** è ottima. Si parte sempre da un dataframe:

```
df_sum |>
    kable() |>
    kable_styling(bootstrap_options = c("striped"),
                  full_width = FALSE)
```

Species	mean	sd	se	min	max
setosa	5.006	0.3524897	0.0498496	4.3	5.8
versicolor	5.936	0.5161711	0.0729976	4.9	7.0
virginica	6.588	0.6358796	0.0899270	4.9	7.9

Creare tabelle da modelli statistici con `sjPlot`

`sjPlot` è un pacchetto tuttofare che tra le altre cose crea grafici e tabelle partendo non da dataframe ma da oggetti di modelli inserendo una serie di informazioni utili in automatico.

```
fit <- lm(Sepal.Length ~ Petal.Length,
          data = iris)

sjPlot::tab_model(fit)
```

	Sepal Length		
Predictors	*Estimates*	*CI*	*p*
(Intercept)	4.31	4.15 – 4.46	**<0.001**
Petal Length	0.41	0.37 – 0.45	**<0.001**
Observations	150		
R^2 / R^2 adjusted	0.760 / 0.758		

Creare tabelle da modelli statistici con `broom`

`broom` ha un approccio diverso (quello che preferisco) creando dei dataframe da modelli statistici e poi lasciando all'utente la creazione della tabella con `flextable` o `kableExtra`:

https://broom.tidymodels.org

.pull-right[

```
tidfit <- broom::tidy(fit)
tidfit
```

```
## [90m# A tibble: 2 × 5 [39m
##   term         estimate std.error statistic   p.value
##   [3m [90m<chr> [39m [23m            [3m [90m<dbl> [39m [23m      [3m [90m<dbl> [39m [23
## [90m1 [39m (Intercept)     4.31     0.078 [4m4 [24m      54.9 2.43 [90me [39m [31m-10
## [90m2 [39m Petal.Length    0.409    0.018 [4m9 [24m      21.6 1.04 [90me [39m [31m- 4
```

]

Materiale

Nel sito del corso arca-dpss.github.io/course-R è presente una sezione con articoli, libri e materiale di approfondimento di tutto quello che abbiamo visto e di questi argomenti extra. Nel tempo questo materiale potrebbe crescere ma sarà sempre disponibile nel link indicato 😄