Programmazione statistica in R: corso teorico-pratico

Parte 1

Damiano G. Preatoni

Unità di Analisi e Gestione delle Risorse Ambientali – *Guido Tosi Research Group*
Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria

prea@uninsubria.it

7–9/7/2021

Contenuti

Sommario

Dalla teoria alla pratica...

L'analisi dei dati è un elemento imprescindibile.[1]

Quali strumenti?

- riproducibilità
- flessibilità
- falsificabilità

- 1970-1980: pochi pacchetti generici (SAS, SPSS, minitab, S-PLUS)
  - uso specialistico per le procedure più comune, programmi *ad hoc* in generale
- 1980-1990: pacchetti proprietari e specializzati (SYSTAT, stata, Origin, ecc.)
  - tendenza a legarsi ad uno specifico *software*
- 1990-oggi: banalizzazione e *canned software*
  - lo *spreadsheet* è l'unico strumento di "calcolo"[2]

[1] Science, 340(6134): 814-815 doi:10.1126/science.1231535

[2] http://www.scientific-computing.com/features/feature.php?feature_id=218

Perché R?

- È libero e gratuito[3]
- È uno standard in ambiente scientifico (e non solo!)
- Riesce a gestire lavori complessi in modo semplice (Big Data!)
- Moduli specializzati per *qualunque analisi*[4]
- Elaborati grafici di qualità tipografica (*press-ready*)

[3] Ihaka R., Gentleman R. 1996. R: a language for data analysis and graphics. Journal of Computational and Graphical Statistics 5: 299-314.

[4] marzo 2009: 1723 pacchetti differenti; ottobre 2013: 4953, luglio 2014: 5721, novembre 2015: 7542, ottobre 2016: 9369, aprile 2017: 10374, novembre 2020: 16568, luglio 2021: 17812

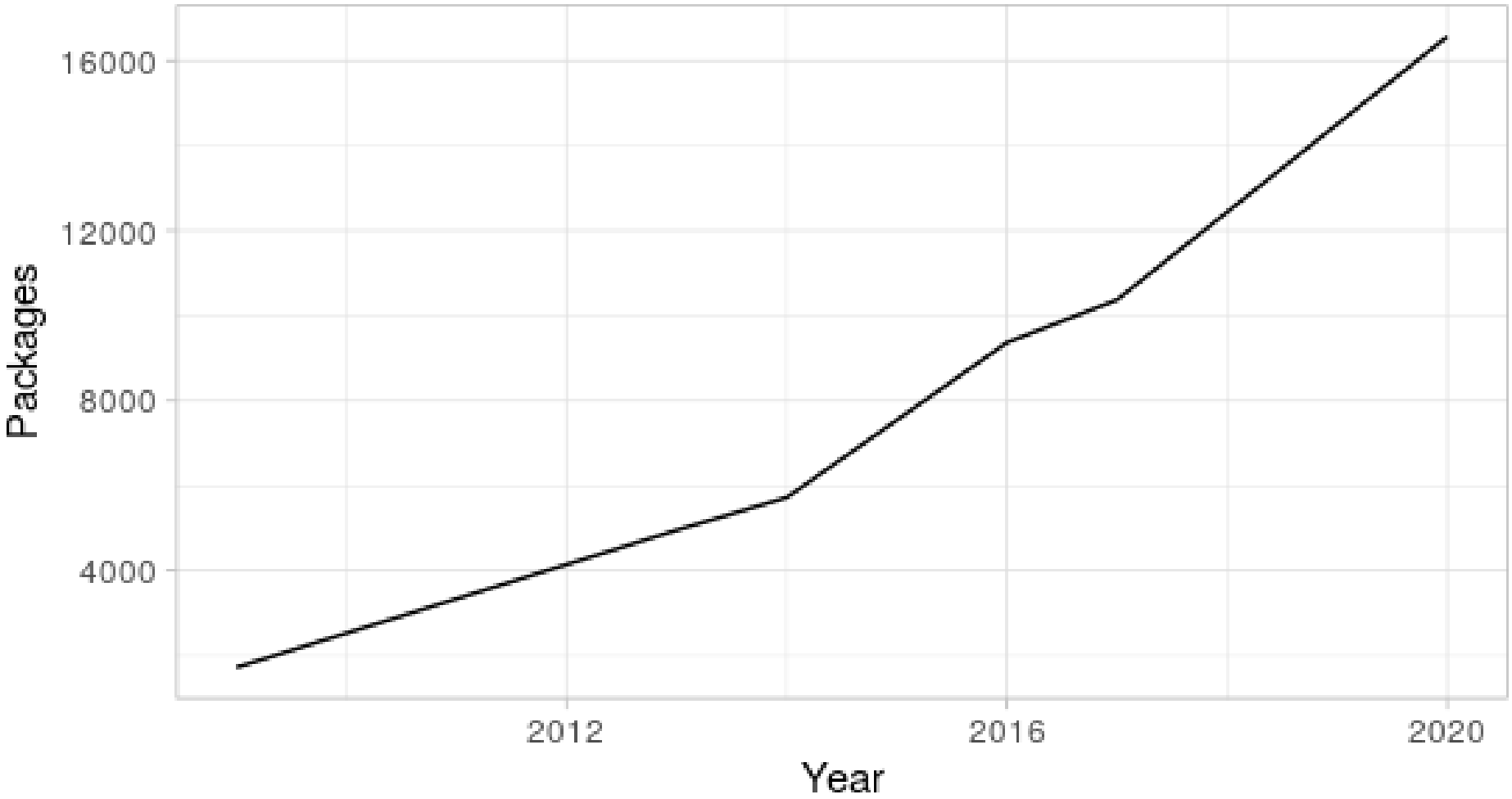
Perché R?

- È libero e gratuito[3]
- È uno standard in ambiente scientifico (e non solo!)
- Riesce a gestire lavori complessi in modo semplice (Big Data!)
- Moduli specializzati per *qualunque analisi*[4]
- Elaborati grafici di qualità tipografica (*press-ready*)
- Costringe a ragionare (*no wizards, no monkey-typing*)
- Dà dipendenza

[3] Ihaka R., Gentleman R. 1996. R: a language for data analysis and graphics. Journal of Computational and Graphical Statistics 5: 299-314.

[4] marzo 2009: 1723 pacchetti differenti; ottobre 2013: 4953, luglio 2014: 5721, novembre 2015: 7542, ottobre 2016: 9369, aprile 2017: 10374, novembre 2020: 16568, luglio 2021: 17812

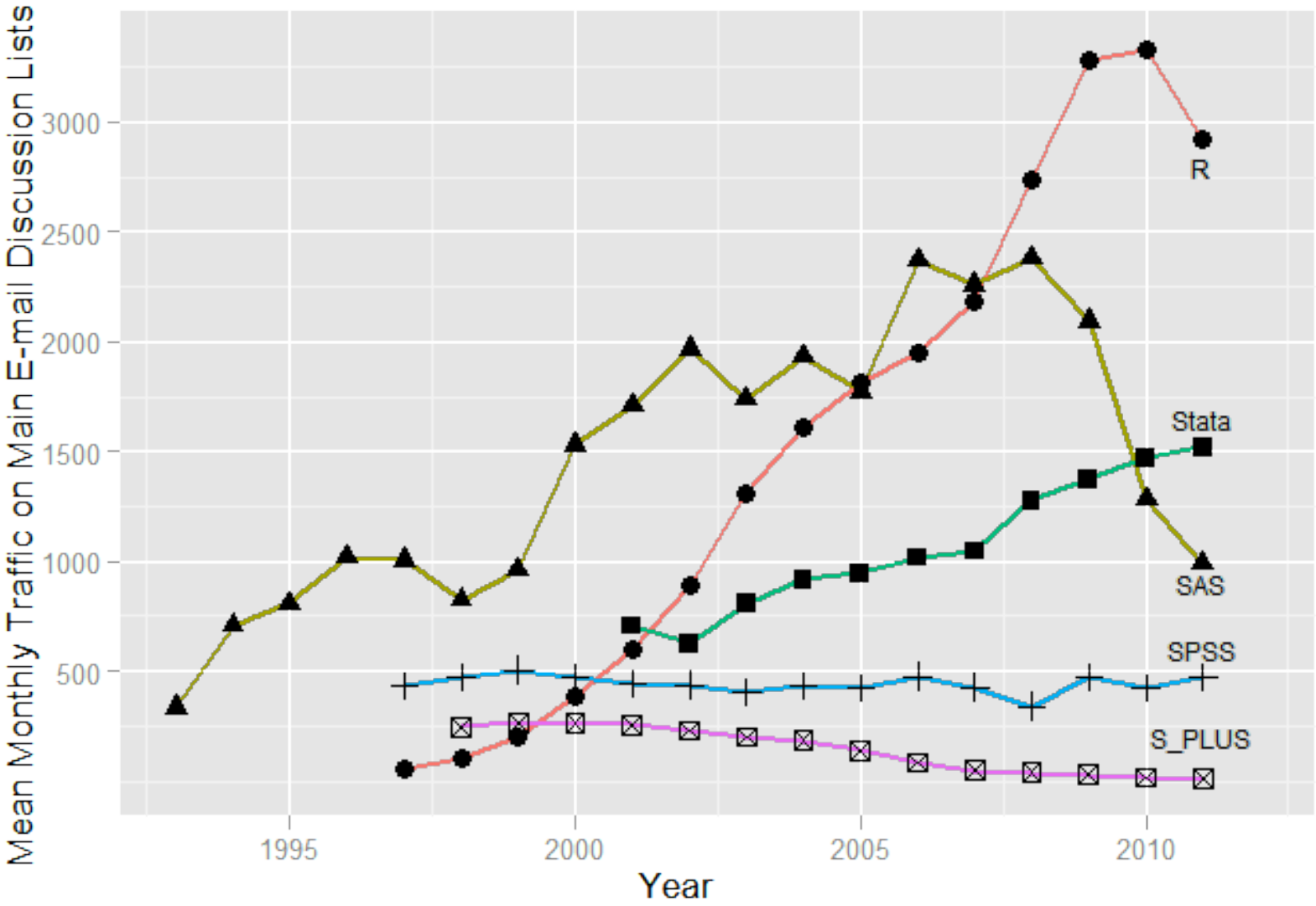
Perché R?



[3] Ihaka R., Gentleman R. 1996. R: a language for data analysis and graphics. Journal of Computational and Graphical Statistics 5: 299-314.

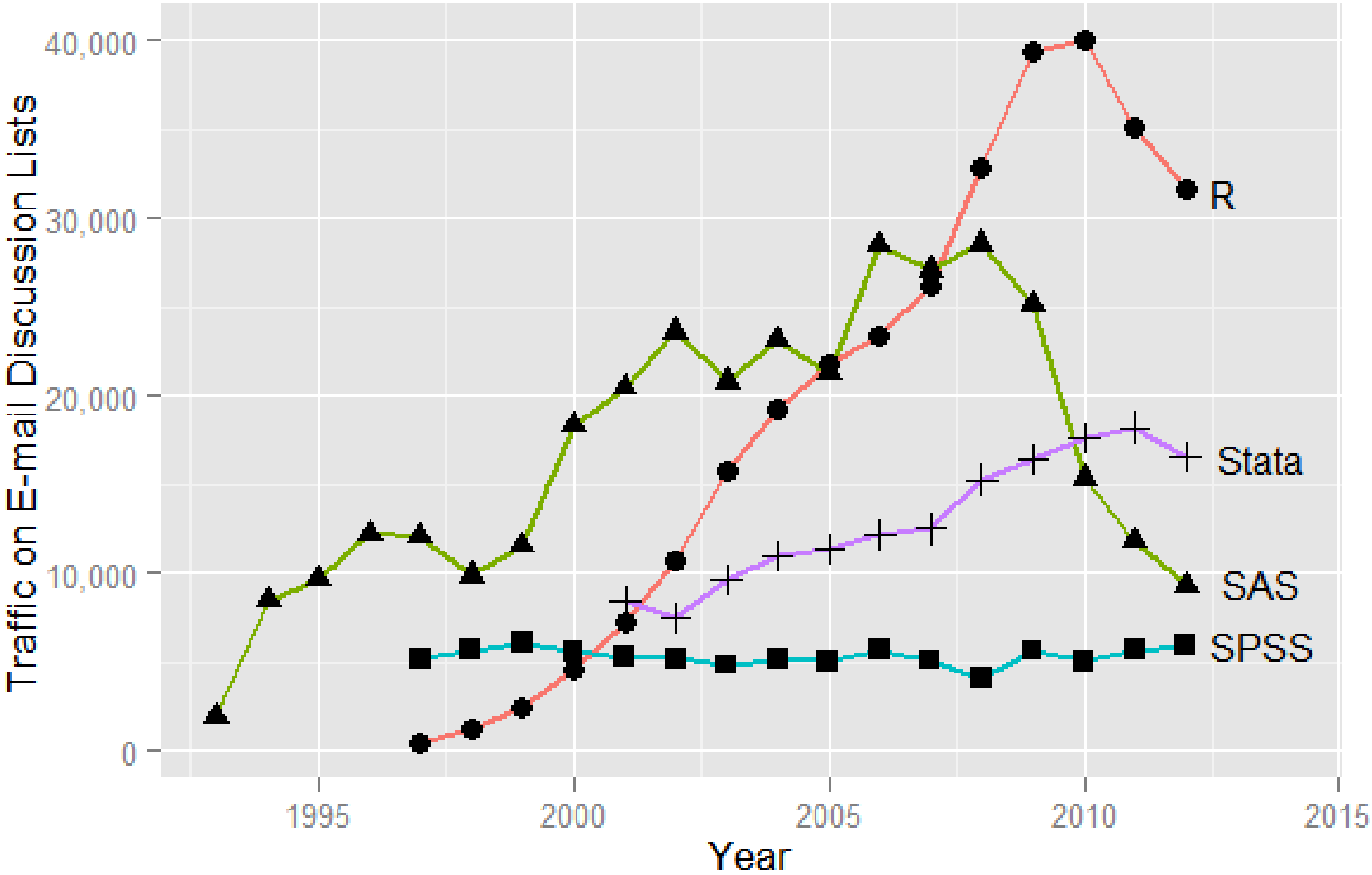
[4] marzo 2009: 1723 pacchetti differenti; ottobre 2013: 4953, luglio 2014: 5721, novembre 2015: 7542, ottobre 2016: 9369, aprile 2017: 10374, novembre 2020: 16568, luglio 2021: 17812

Perché R?



[5] http://r4stats.com/articles/popularity/

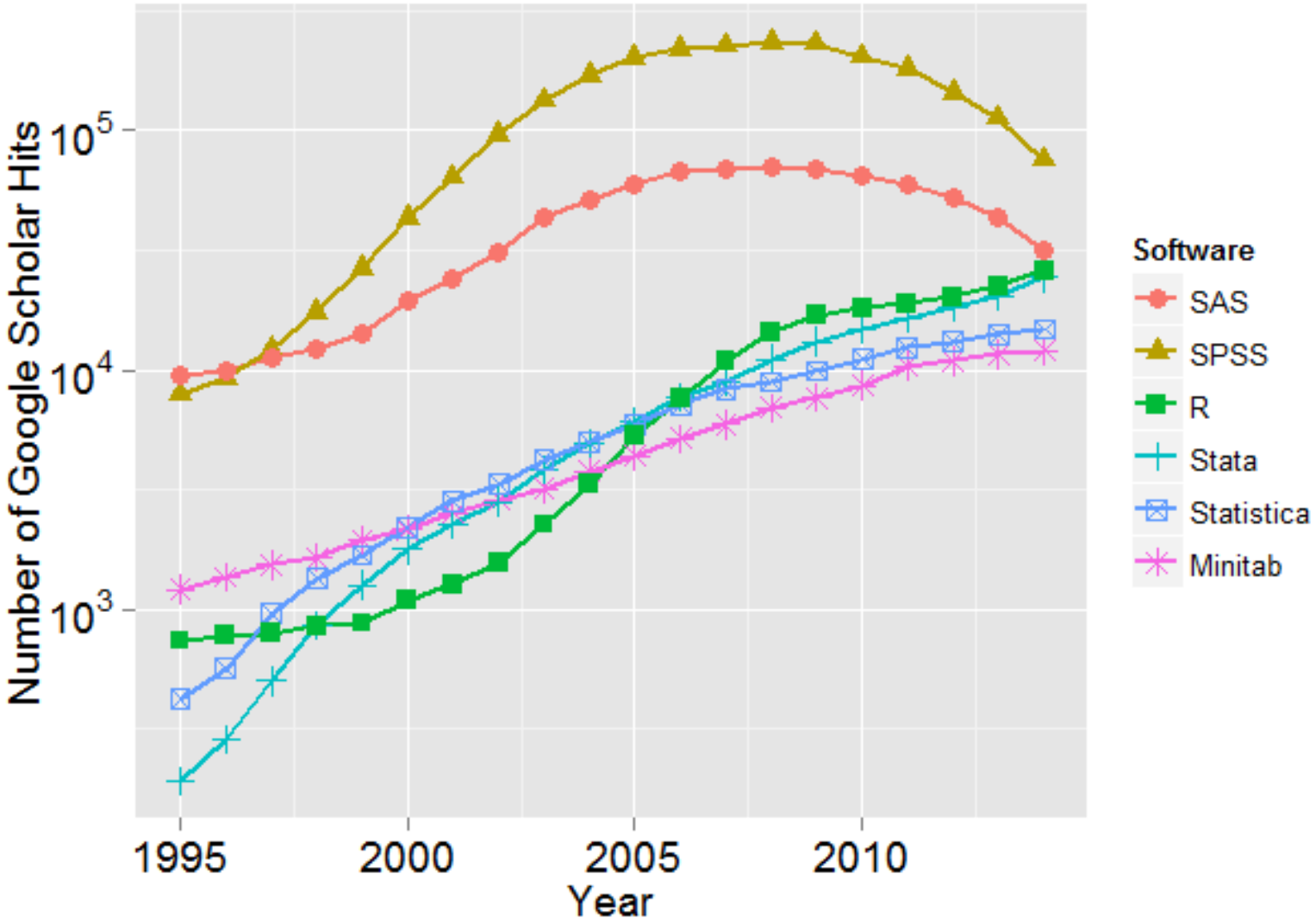
Perché R?



5

[5] http://r4stats.com/articles/popularity/

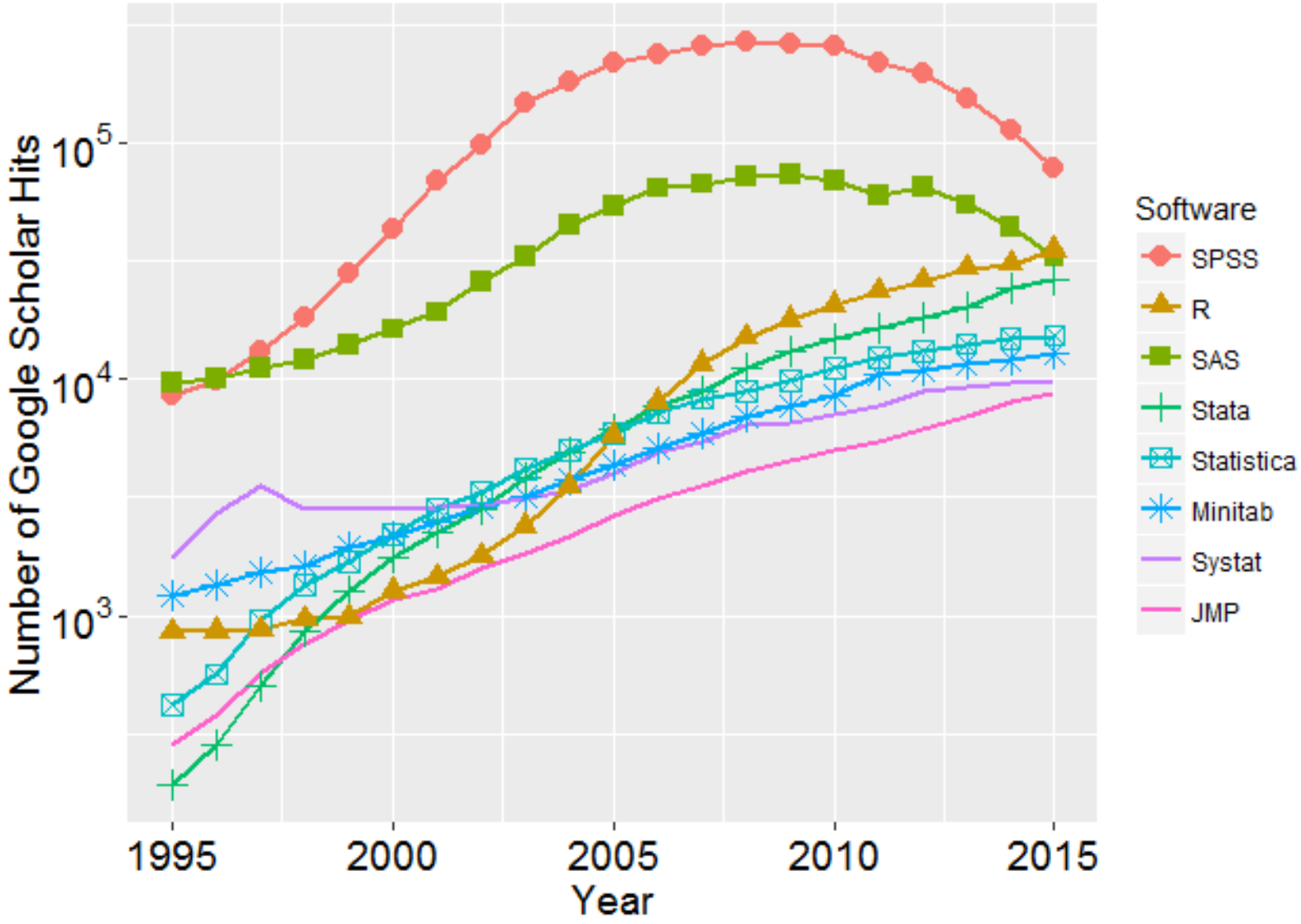
Perché R?



5

[5] http://r4stats.com/articles/popularity/

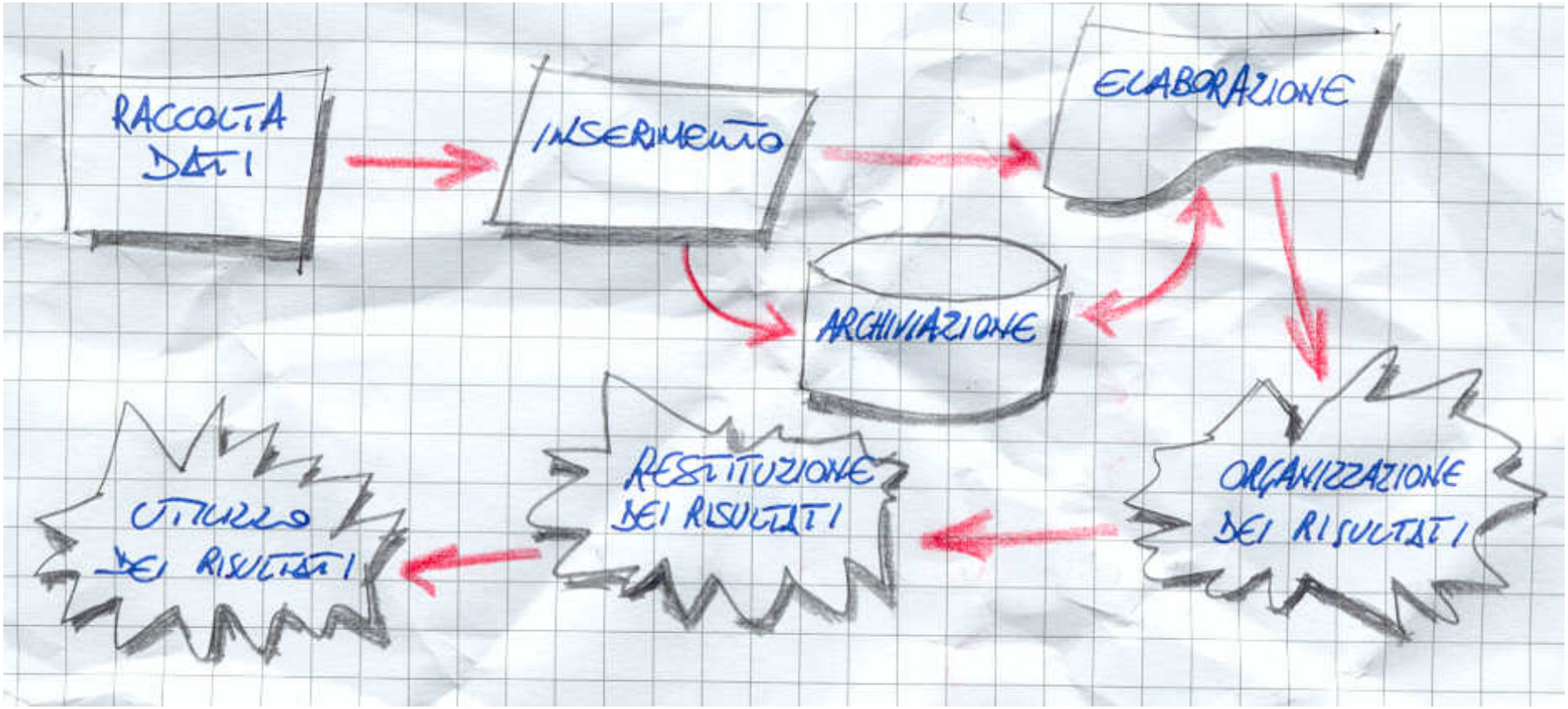
Perché R?



5

[5] http://r4stats.com/articles/popularity/

Il flusso di lavoro



Il flusso di lavoro: *step by step* o *one shot?*

R

- un'analisi comprende una serie di passaggi
- ogni passaggio produce dei risultati intermedi, che possono essere utilizzati nei passaggi successivi
- non vengono mostrati a video ("*stampati*") risultati, se non a richiesta

Altri programmi

- un'analisi è eseguita in un unico passaggio
- il risultato non è "riutilizzabile"
- il risultato è un *output* lungo e prolisso

Altri software

Vantaggi

- velocità nell'eseguire analisi standard
- interattività
- visibilità del singolo dato

Altri software

Vantaggi apparenti

- velocità nell'eseguire analisi standard
  - e con le analisi non standard? quali e quante analisi sono previste?
- interattività
  - interattività significa interagire con i *dati*, non con i comandi del programma
- visibilità del singolo dato
  - *big data*: a che serve "vedere" una matrice di 7000 righe e 200 colonne?

Altri software

Vantaggi

- velocità nell'eseguire analisi standard
- interattività
- visibilità del singolo dato

. . . ma a che prezzo?

- impossibile effettuare analisi "non standard"
- interattività significa interagire con i dati, non con i comandi
- ha senso poter esaminare una matrice di 7000 righe e 200 colonne?
- scarsa estensibilità
- scarsa qualità e inflessibilità dell'output
- impossibile (o quasi) da automatizzare

Perché R è meglio!

Vantaggi

- suddivisione del problema
- possibilità di salvare e riutilizzare gli intermedi
- *single point of truth*

Perché R è meglio!

Vantaggi

- suddivisione del problema
- possibilità di salvare e riutilizzare gli intermedi
- *single point of truth*

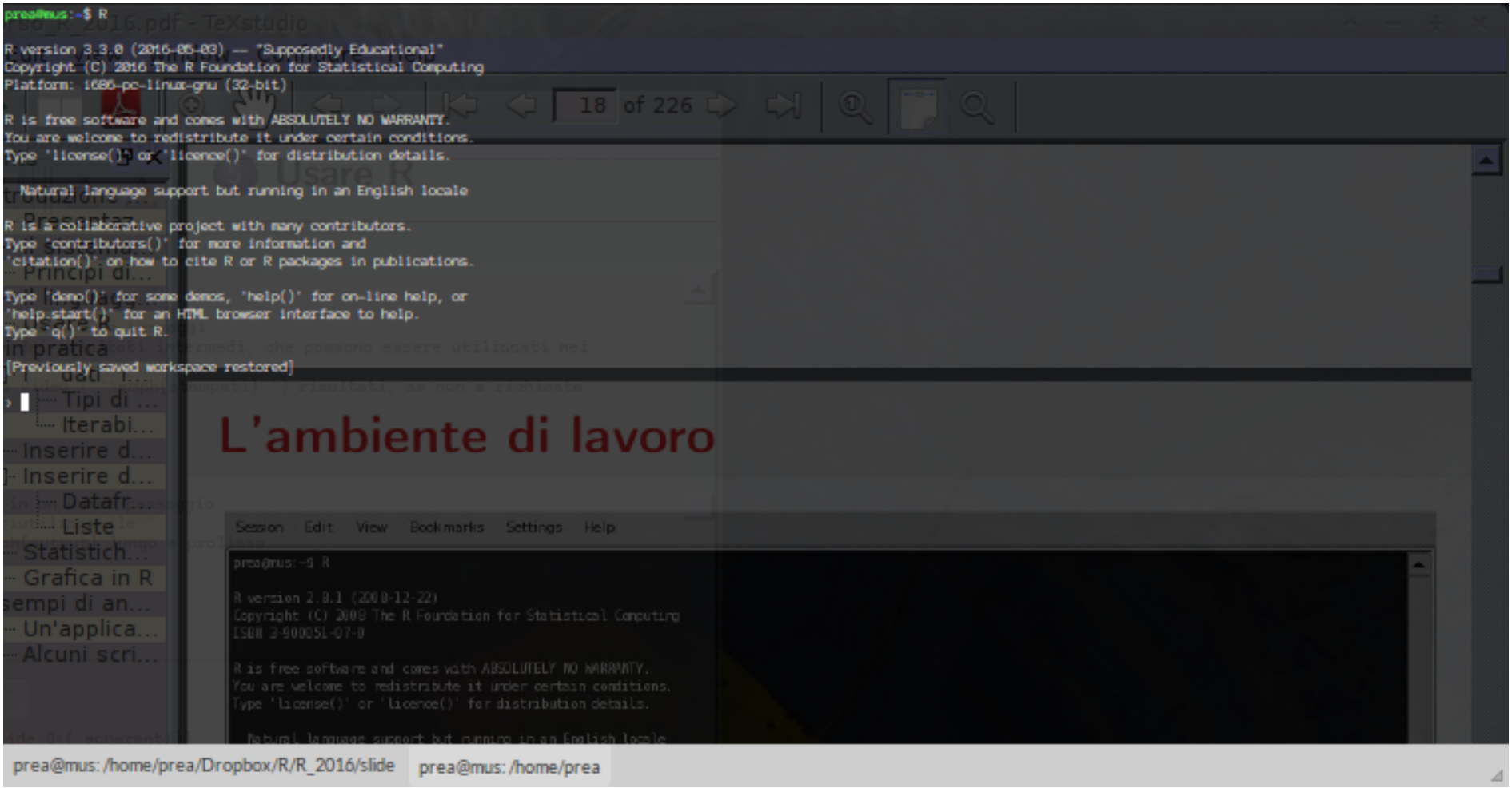
. . . e completa libertà di azione!

- R è un *linguaggio*, quindi è possibile programmare *qualunque cosa*
- la CLI[a] è più produttiva di una GUI[b], e garantisce ripetibilità
- . . . se "complicato" significa "sapere cosa si sta facendo", R è semplice, SPSS no!

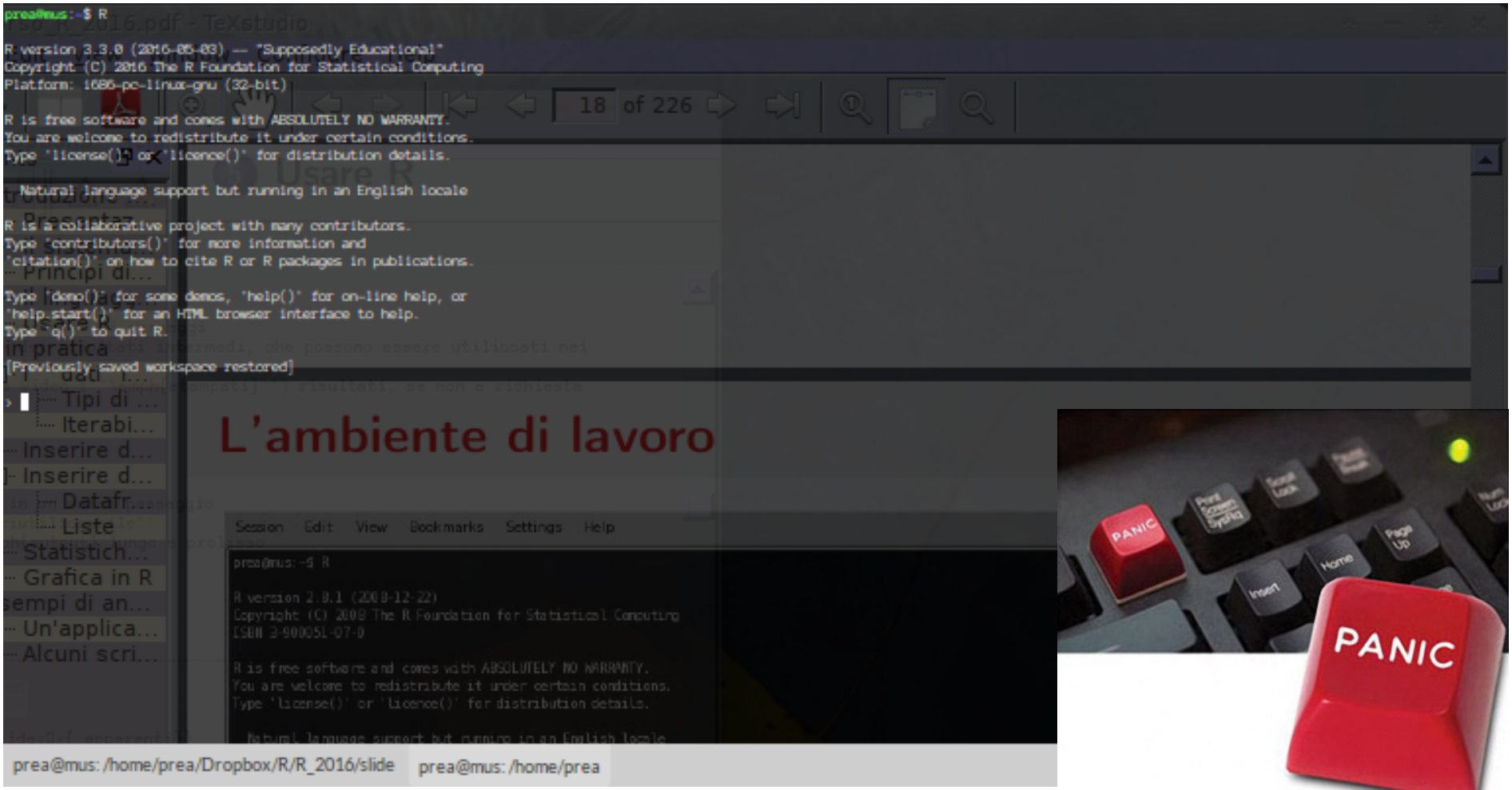
[a] Command Line Interface
[b] Graphical User Interface

Sommario

L'ambiente di lavoro



L'ambiente di lavoro



L'ambiente di lavoro

La *shell* R non è sicuramente la "postazione di lavoro" più comoda...

...ma l'apparenza inganna!

Dato che un comando viene digitato da tastiera, un insieme di comandi può essere contenuto in un file di testo: normalmente si utilizza un *text editor* come *front end*[6]

Editor

- Costruzione di uno *script*
- Sviluppo di funzioni
- Riutilizzo di codice precedentemente scritto

Console

- Verifica di un singolo comando
- Esame interattivo dei dati

[6] o meglio, di un *editor* integrato in un ambiente dedicato a R

Apt-get it![7]

I software indicati sono solo alcune delle numerose possibilità!

*NIX

KDE, IDE: RKward

emacs: ESS (emacs speaks statistics)

OSX

R.app

Komodo Editor

Windows

Editor: Tinn-R

IDE: RKward

Multipiattaforma!!

RStudio (`https://www.rstudio.com/products/rstudio/`)

Benvenuti a casa! ...Windows

RStudio

File Edit Code View Project Workspace Plots Tools Help

Project: (None)

diamondPricing.R* | formatPlot.R | diamonds

Source on Save | Source

```
1  library(ggplot2)
2  source("plots/formatPlot.R")
3
4  view(diamonds)
5  summary(diamonds)
6
7  summary(diamonds$price)
8  aveSize <- round(mean(diamonds$carat), 4)
9  clarity <- levels(diamonds$clarity)
10
11 p <- qplot(carat, price,
12             data=diamonds, color=clarity,
13             xlab="Carat", ylab="Price",
14             main="Diamond Pricing")
15
```

15:1 (Top Level) R Script

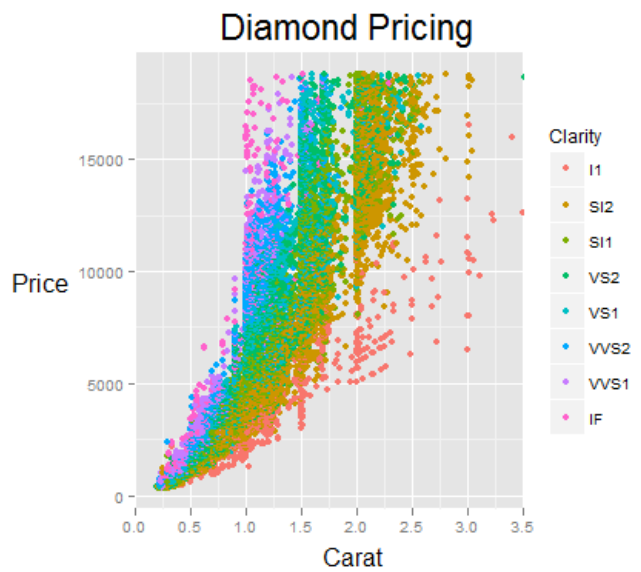
Workspace | History

Load | Save | Import Dataset | Clear All

Data	
diamonds	53940 obs. of 10 variables
Values	
aveSize	0.7979
clarity	character[8]
p	ggplot[8]
Functions	
format.plot(plot, size)	

Files | Plots | Packages | Help

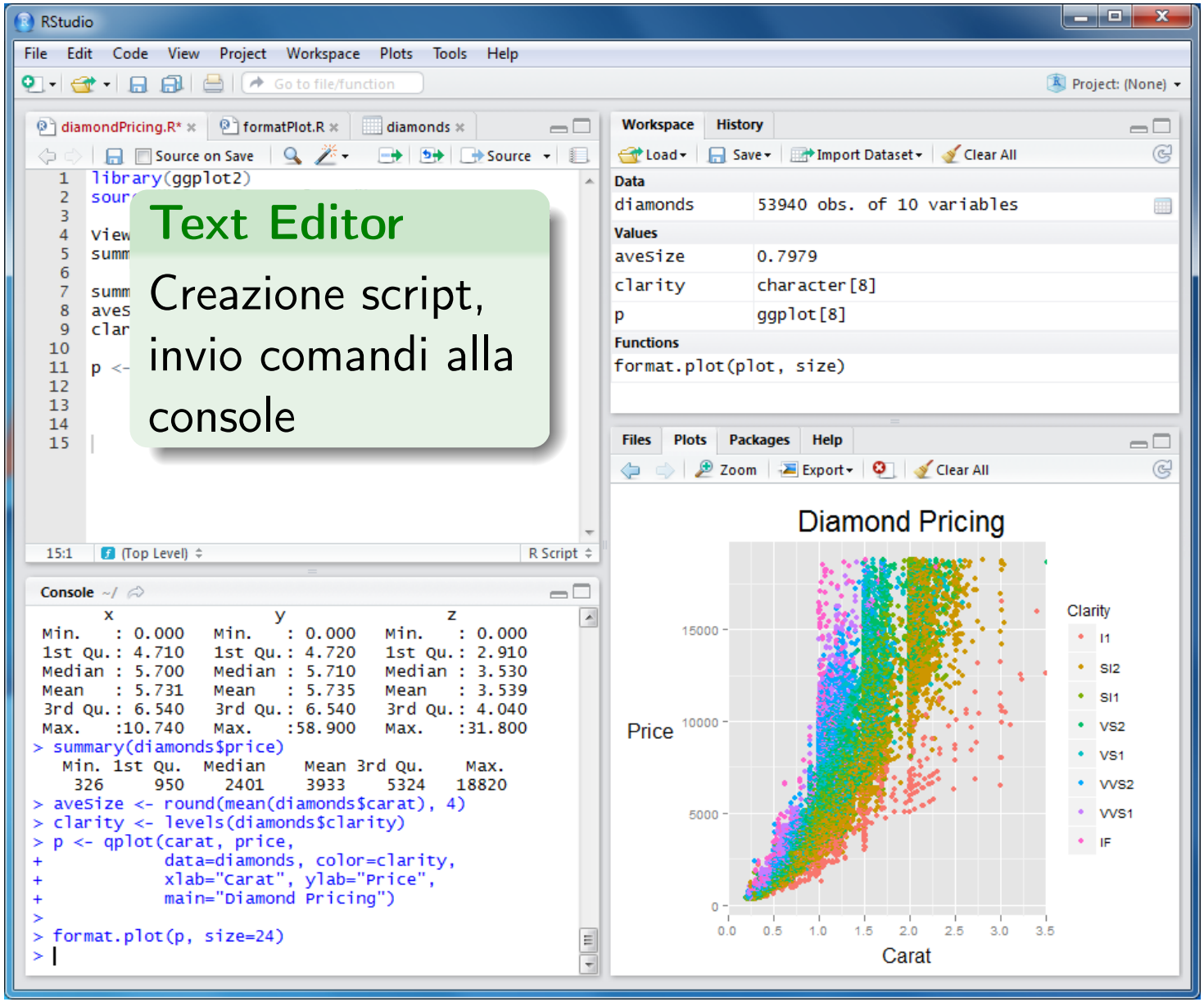
Zoom | Export | Clear All



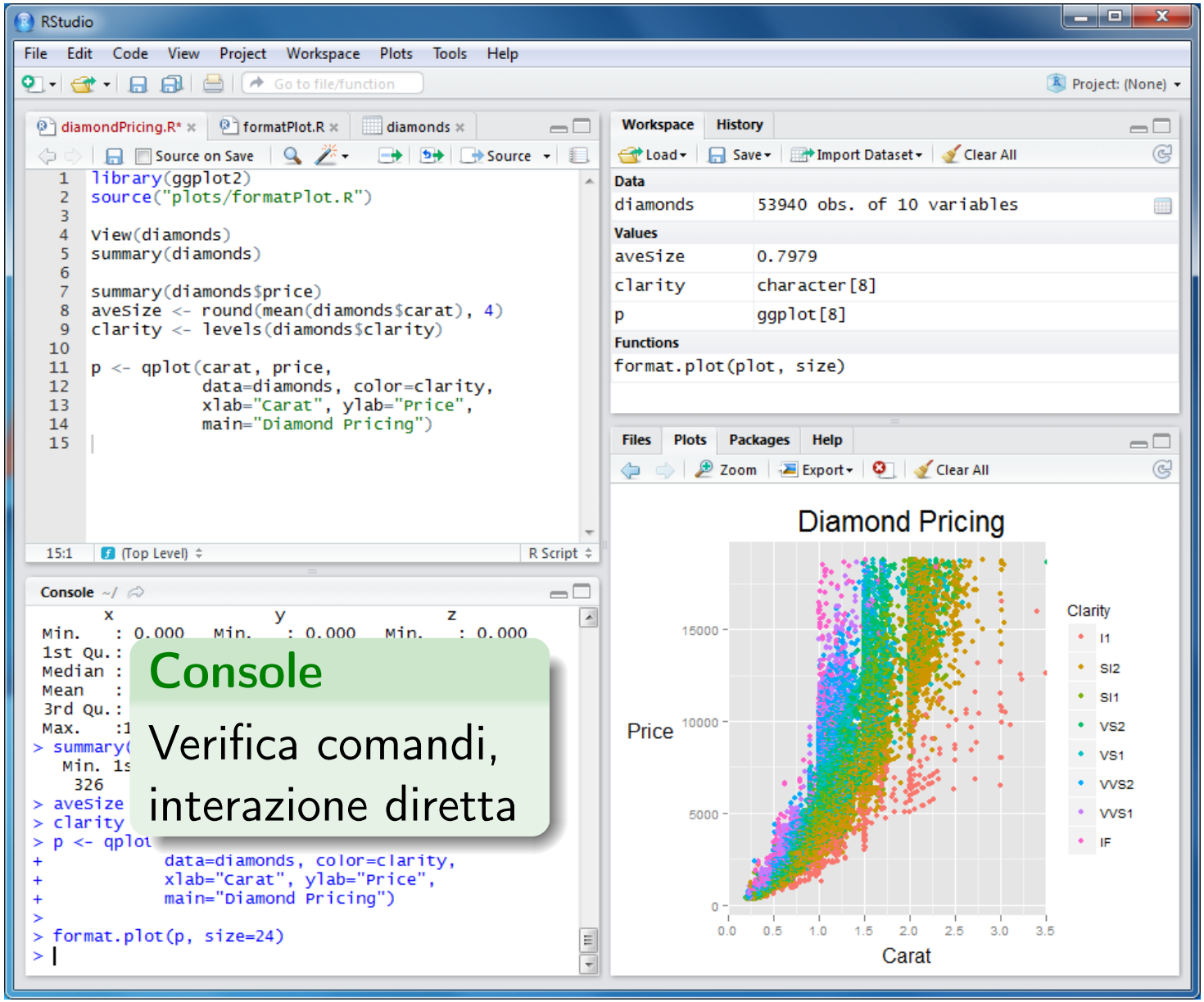
Console ~/

```
       x                y                z
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
 Median : 5.700   Median : 5.710   Median : 3.530
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
 Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(p, size=24)
>
```

Benvenuti a casa! ...Windows



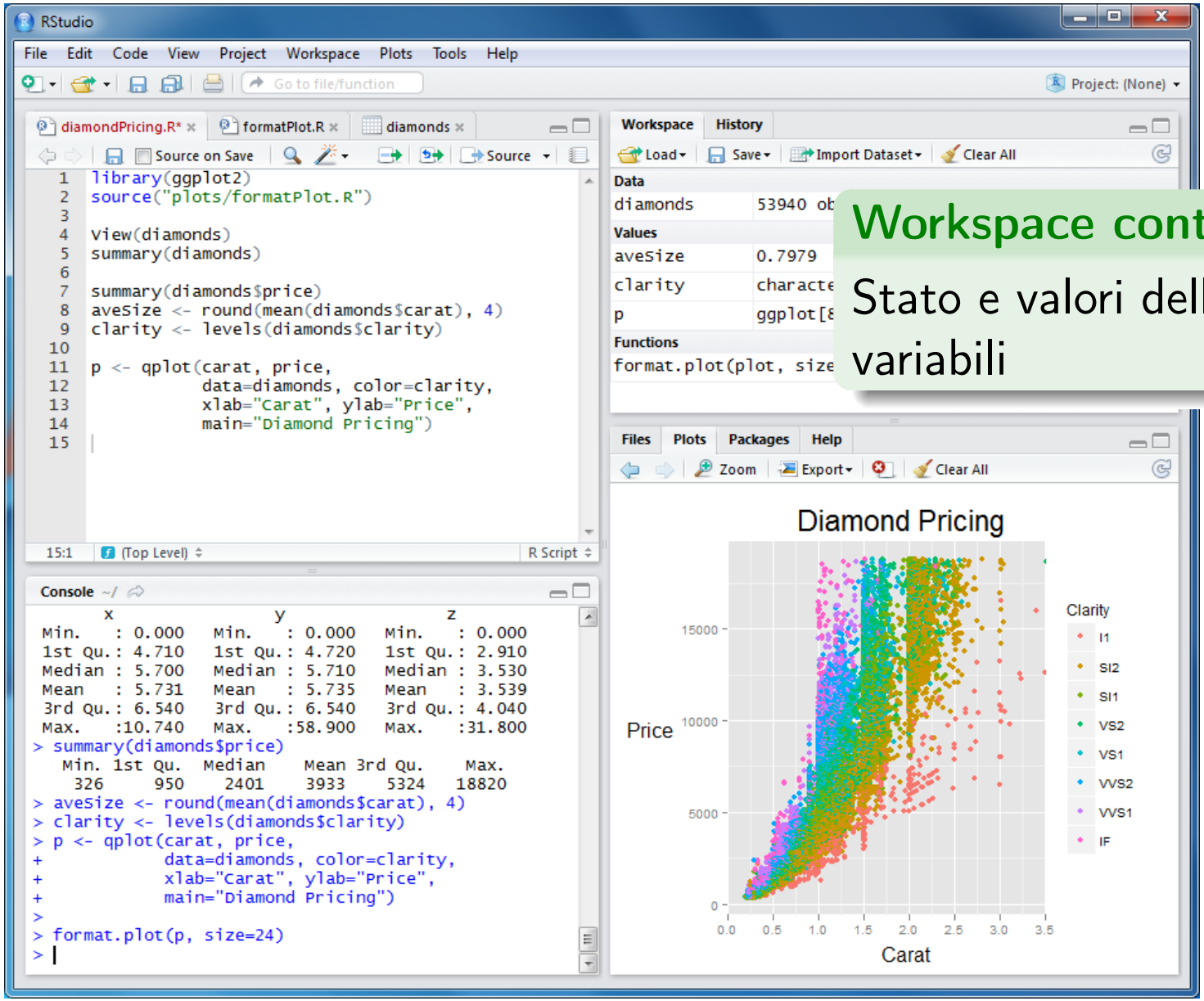
Benvenuti a casa! ...Windows



Console

Verifica comandi, interazione diretta

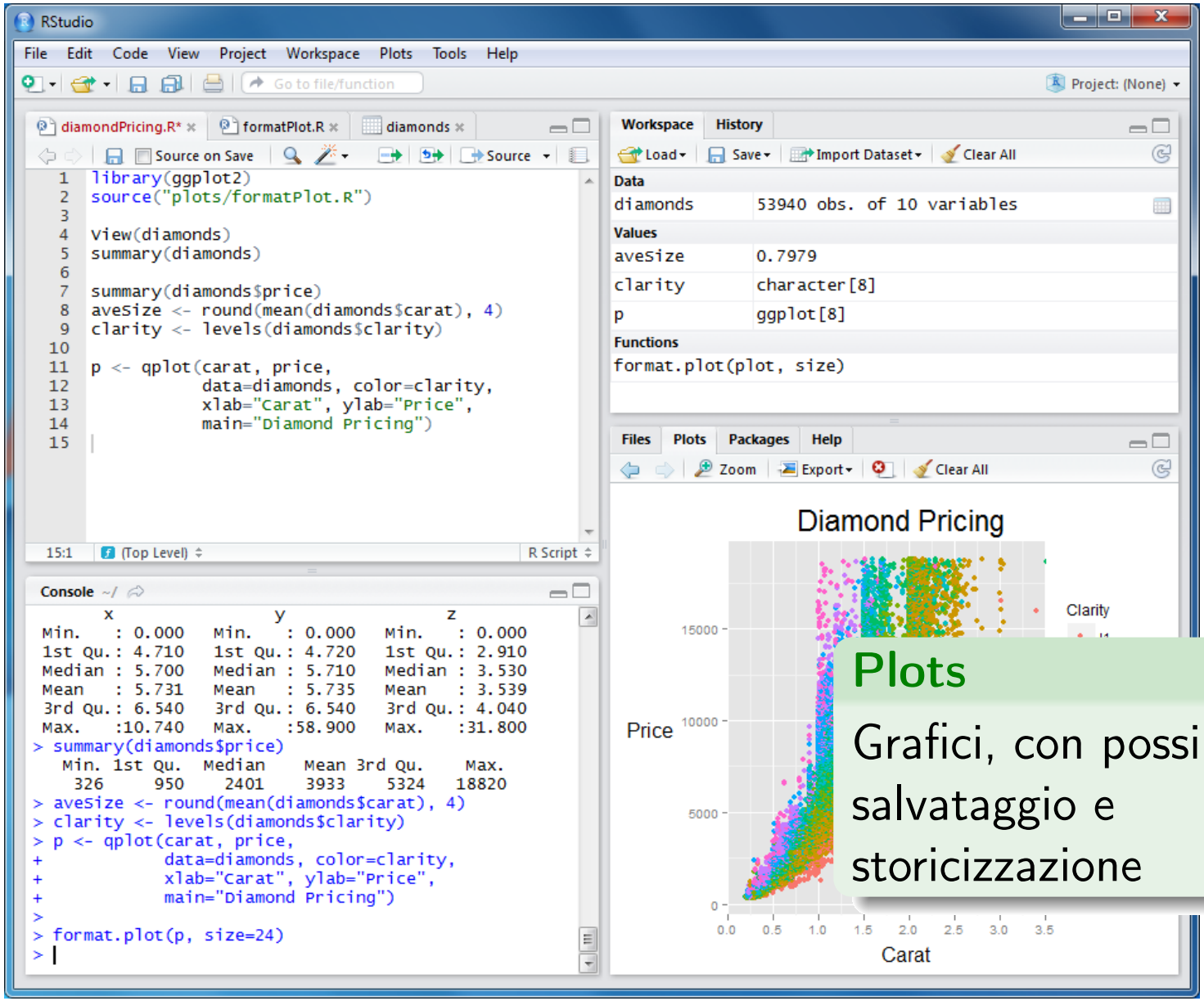
Benvenuti a casa! ...Windows



Workspace content

Stato e valori delle variabili

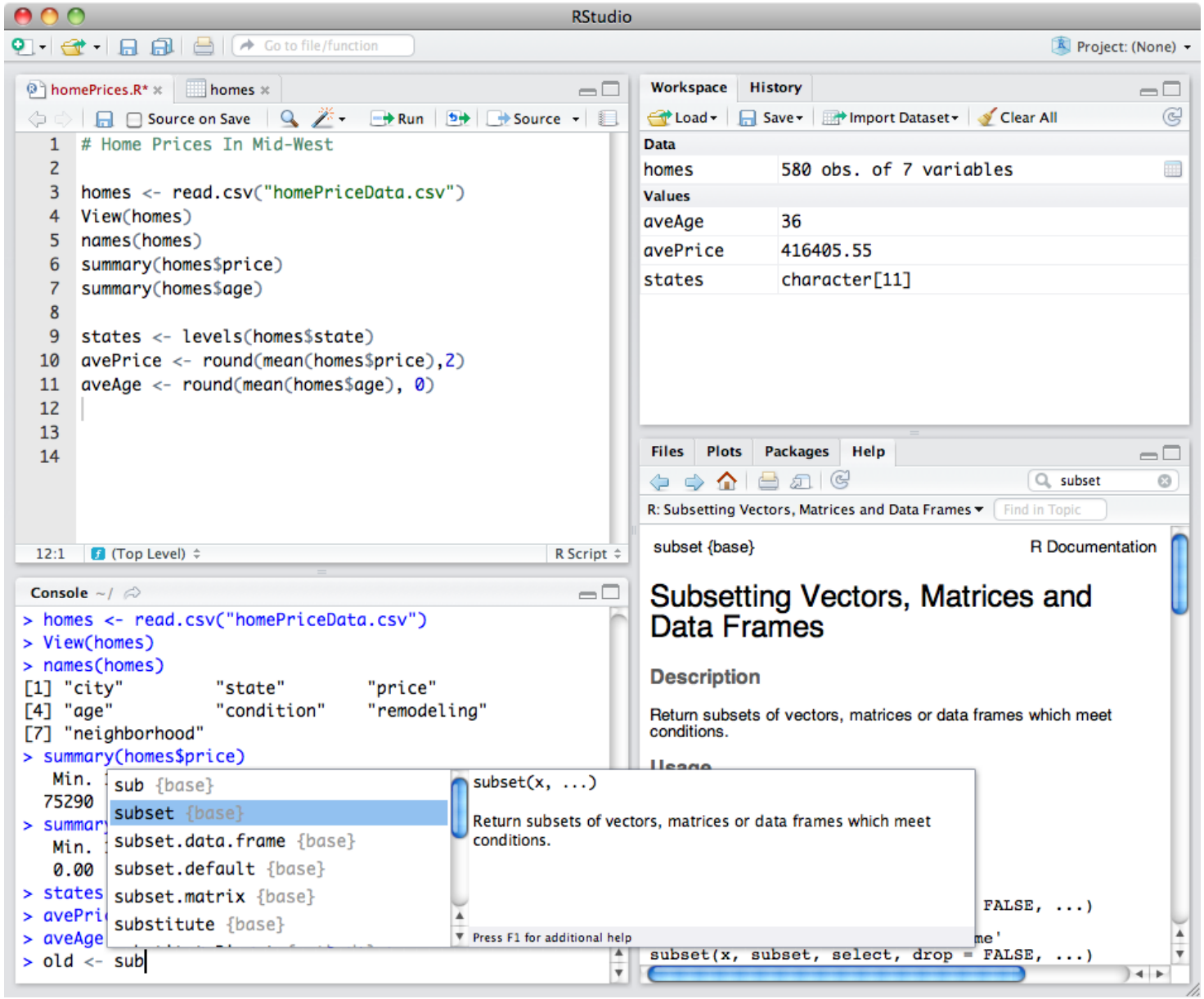
Benvenuti a casa! ...Windows



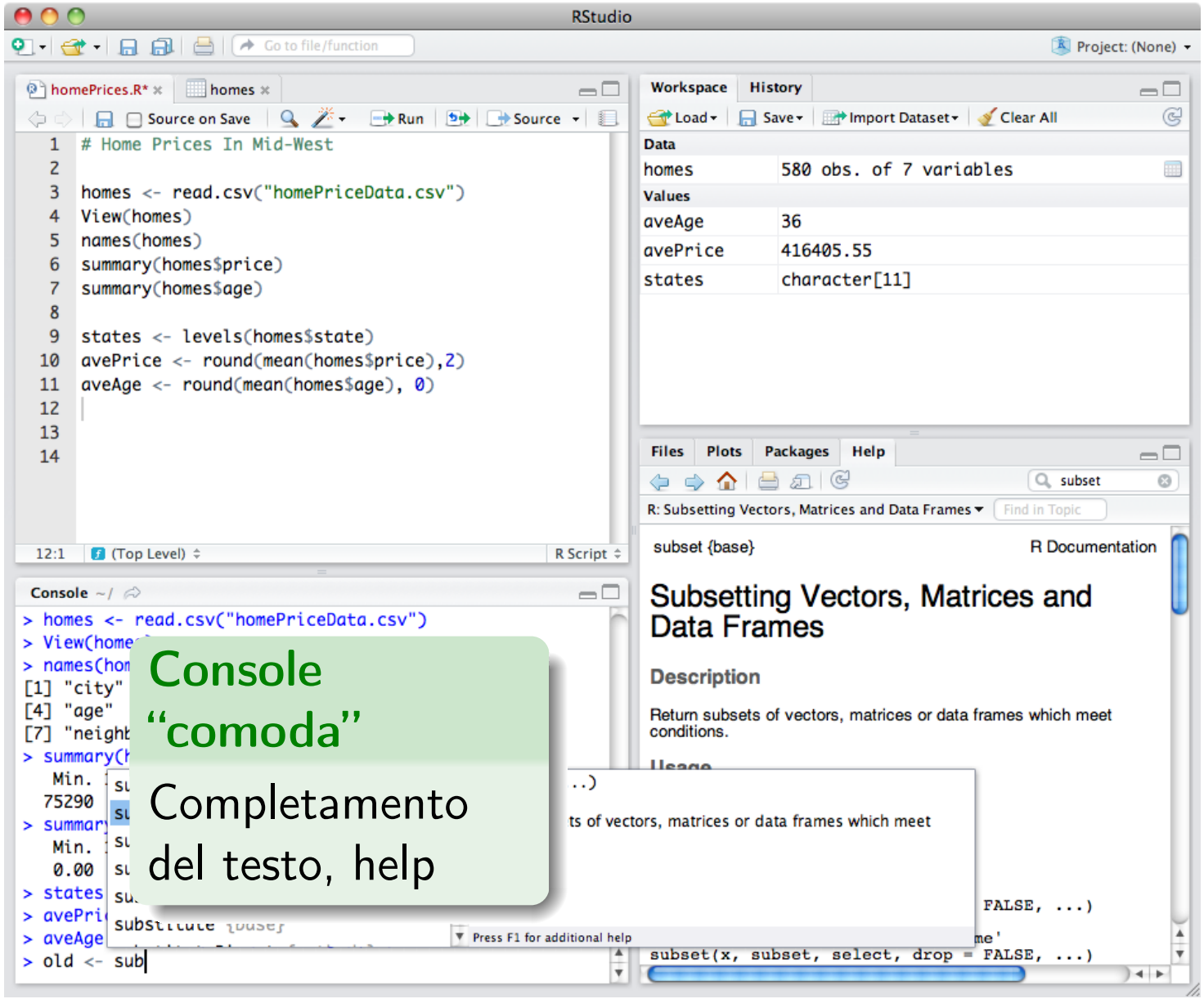
Plots

Grafici, con possibilità di salvataggio e storicizzazione

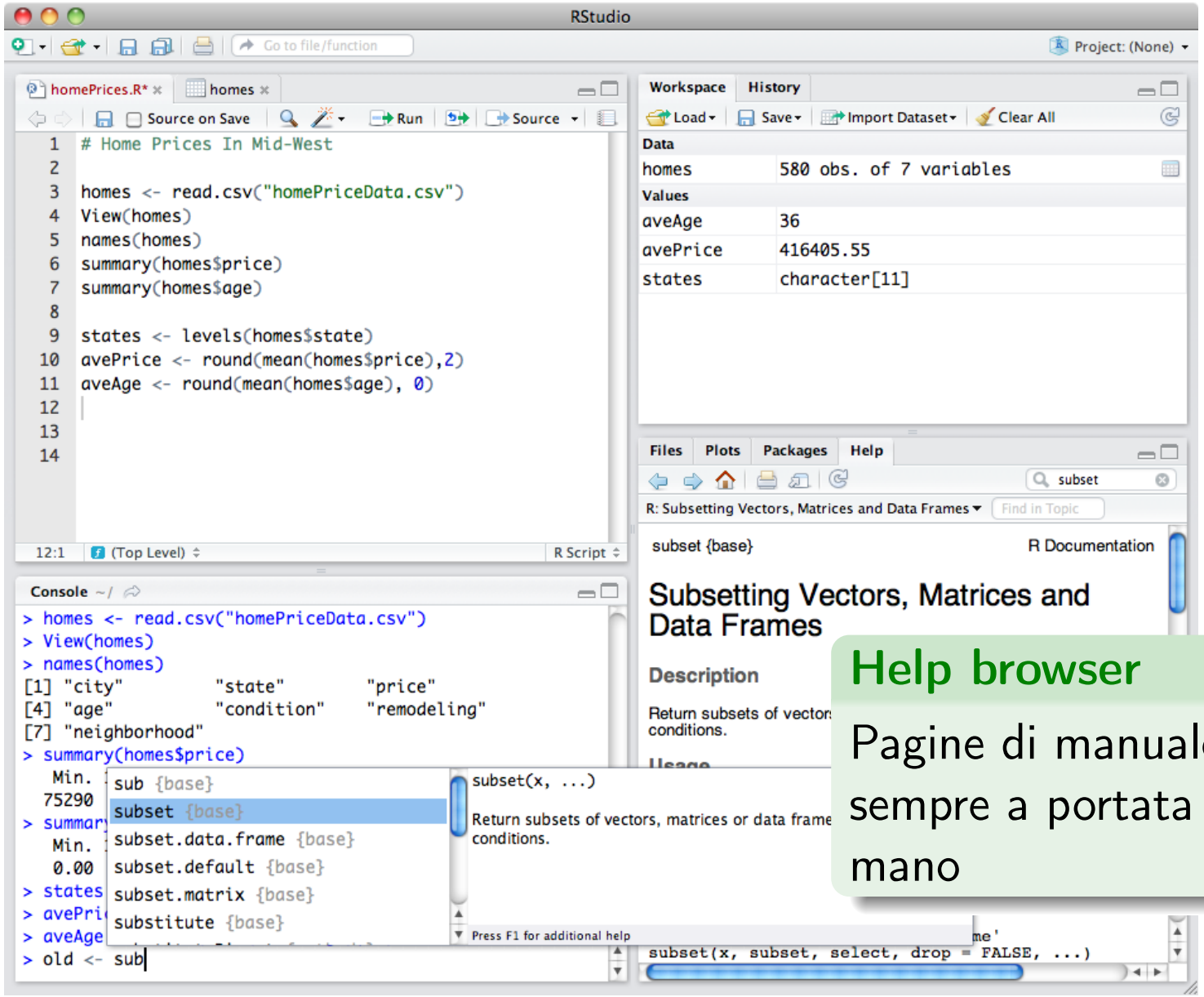
Benvenuti a casa! ...Apple OSX



Benvenuti a casa! ...Apple OSX



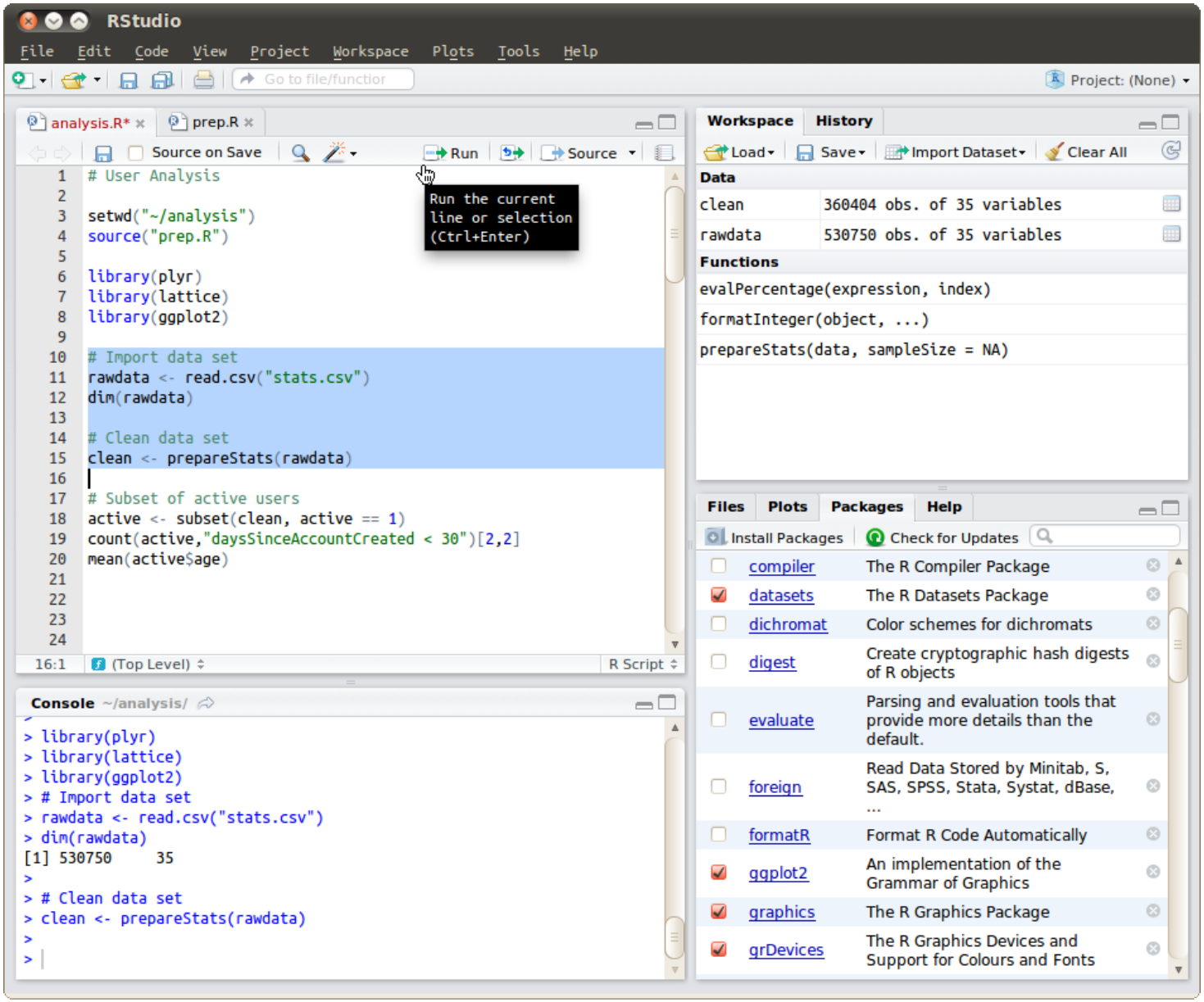
Benvenuti a casa! ... Apple OSX



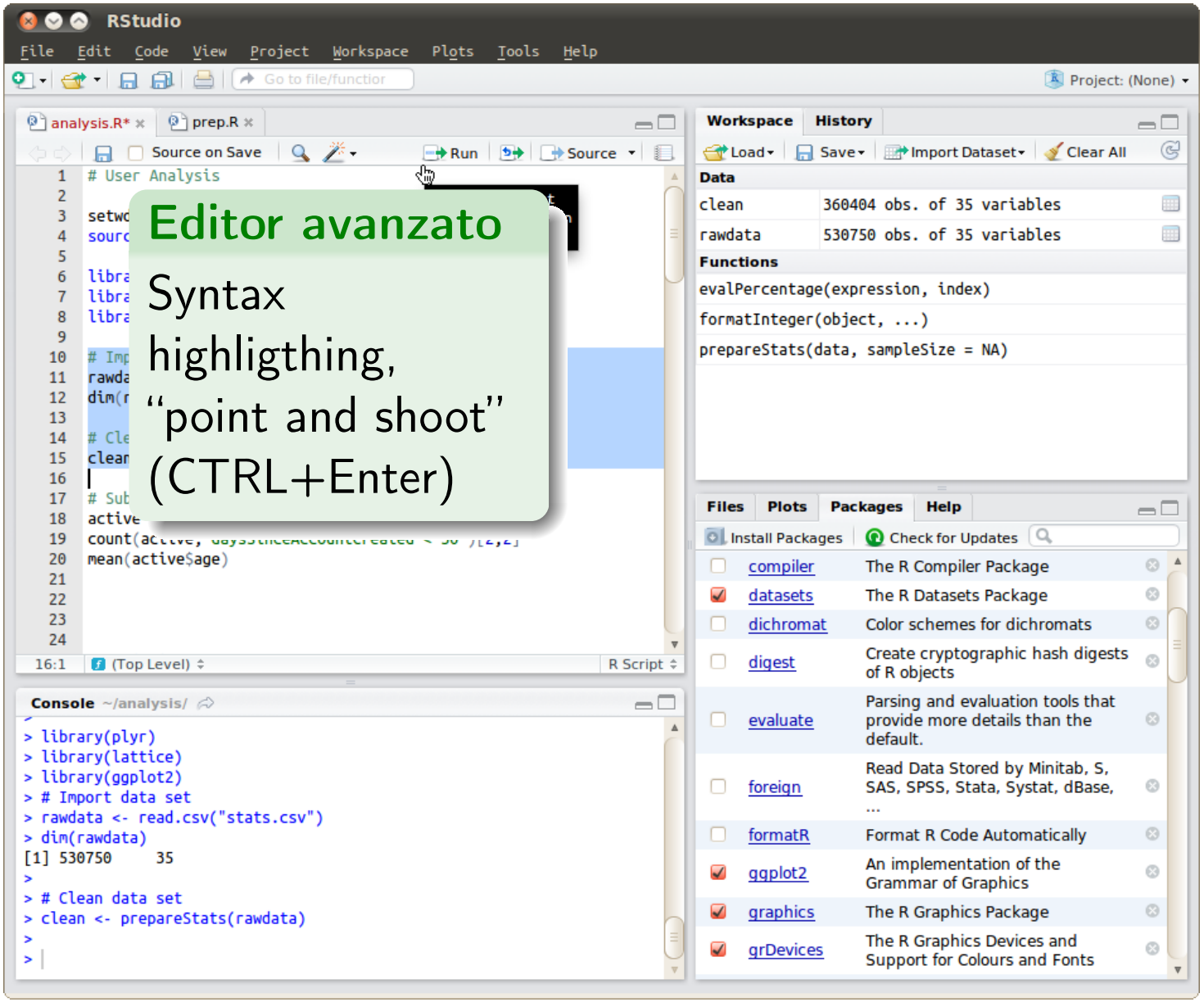
Help browser

Pagine di manuale sempre a portata di mano

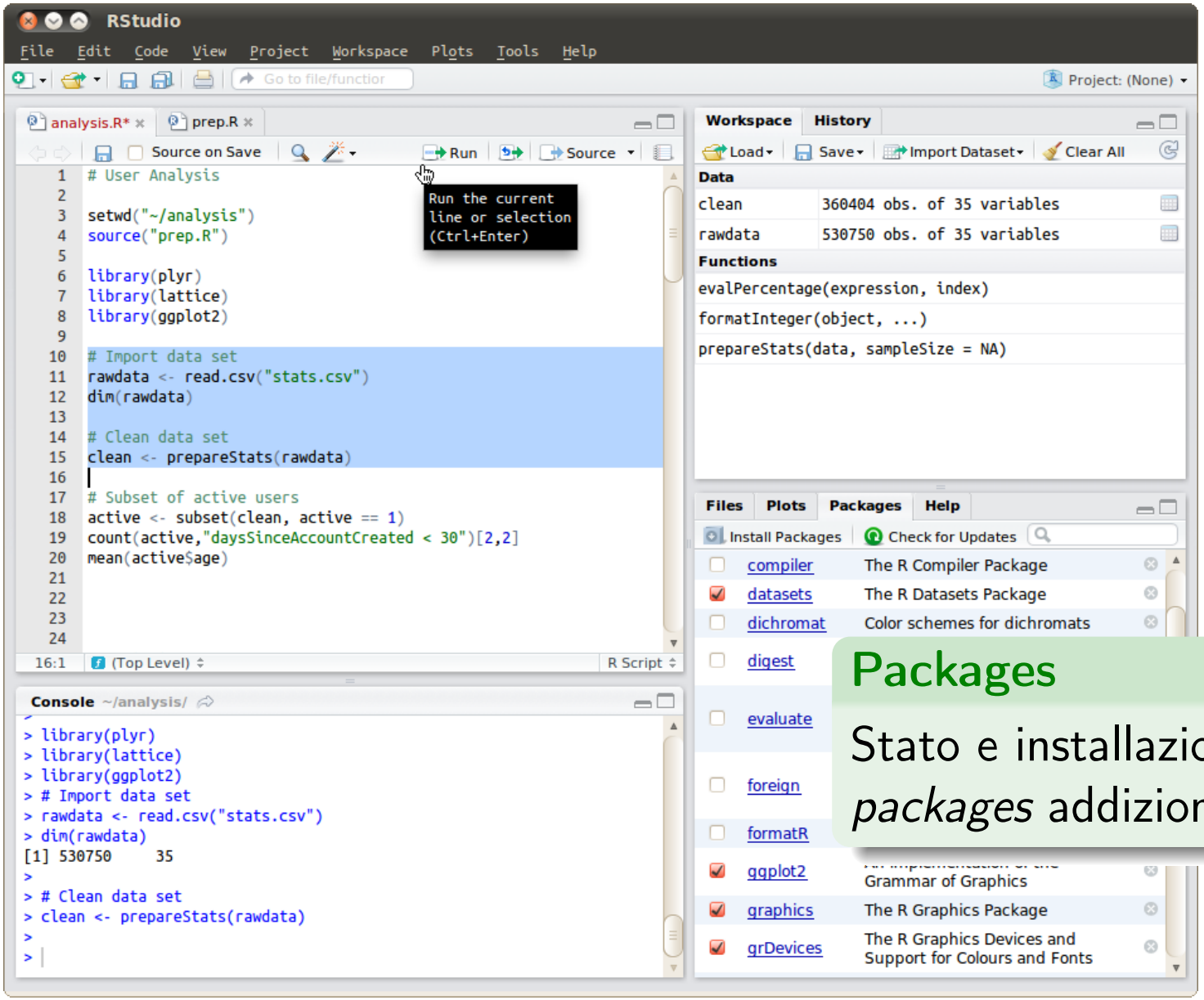
Benvenuti a casa! ...Ubuntu (GNOME)



Benvenuti a casa! ...Ubuntu (GNOME)



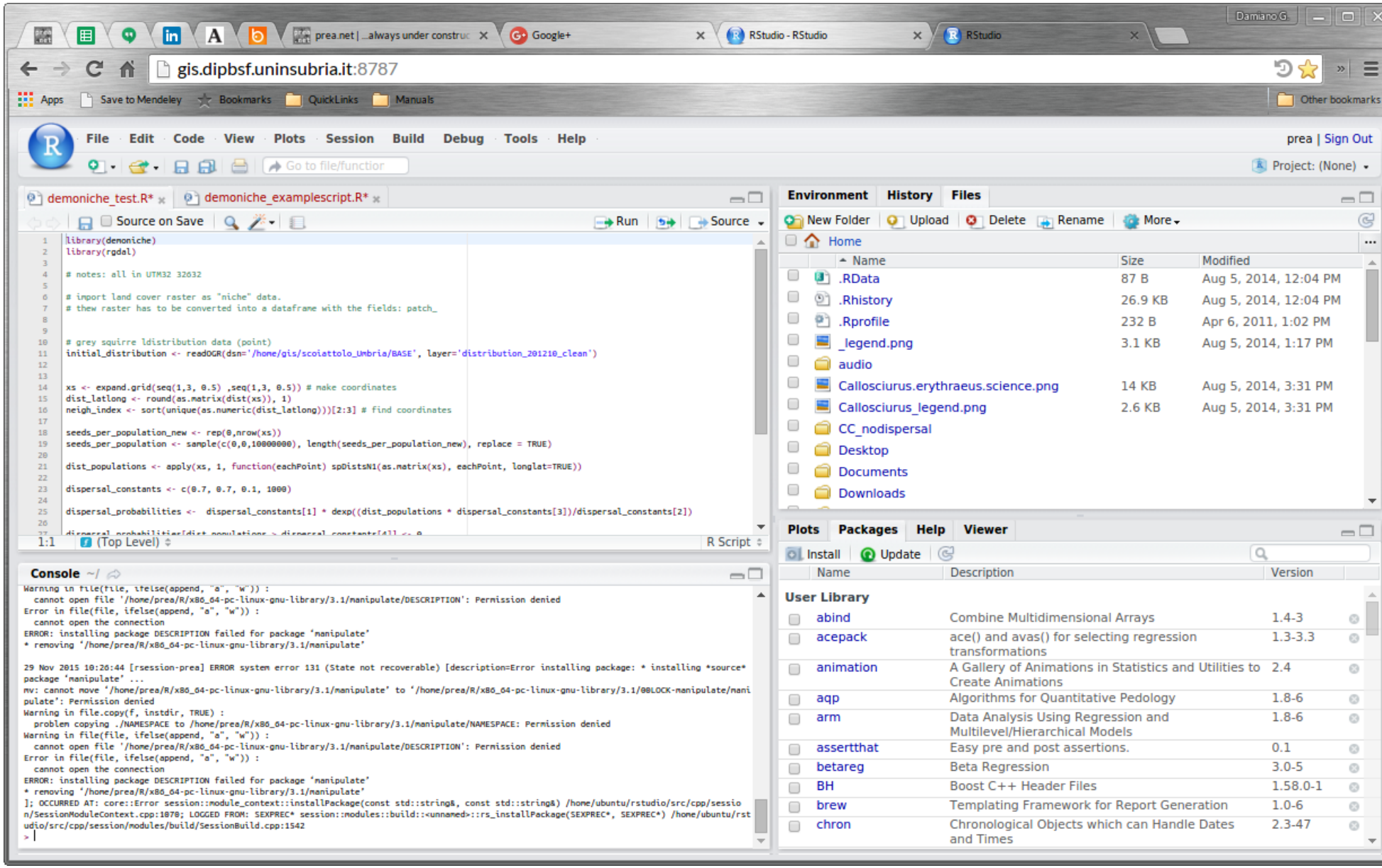
Benvenuti a casa! ...Ubuntu (GNOME)



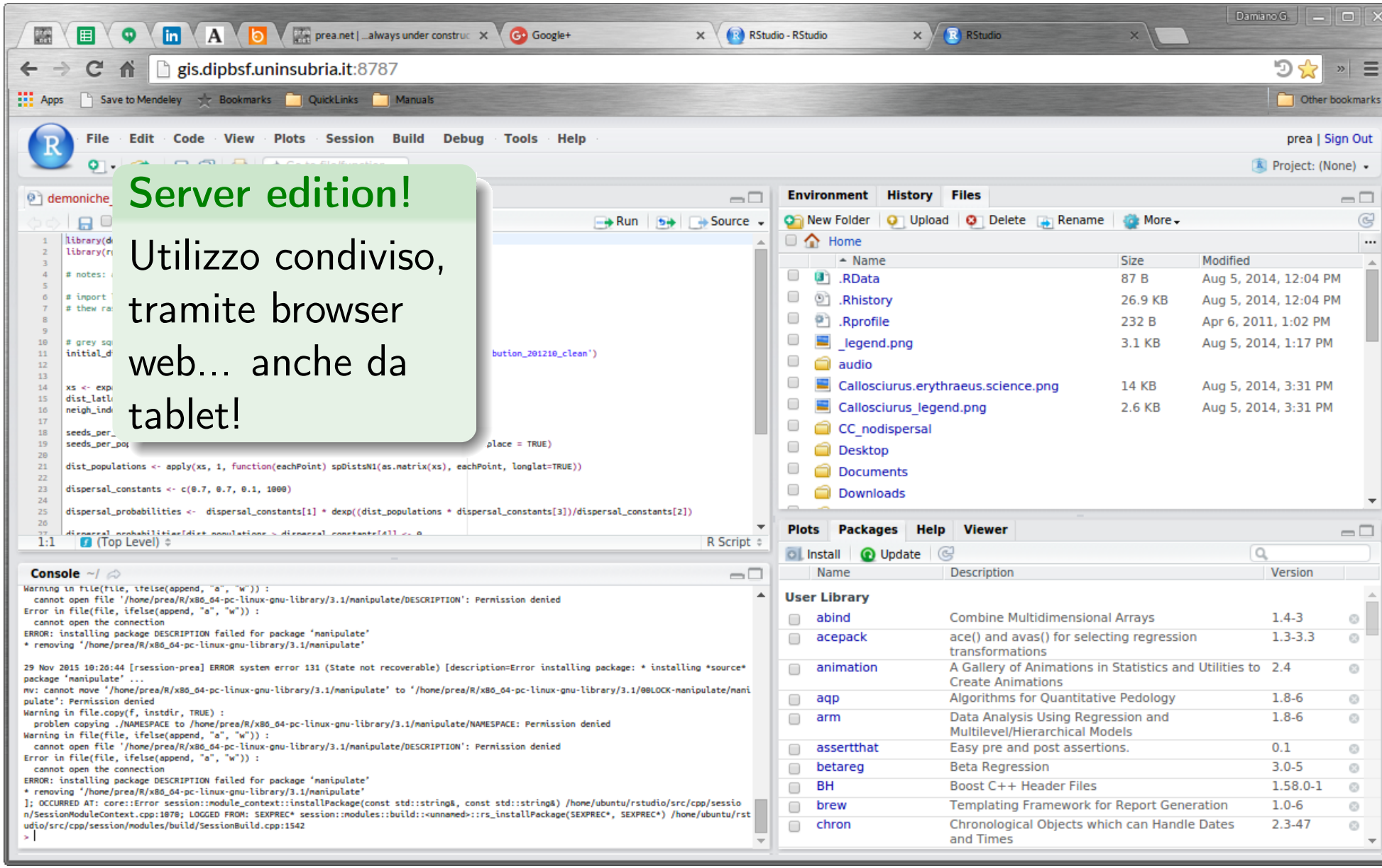
Packages

Stato e installazione di *packages* addizionali

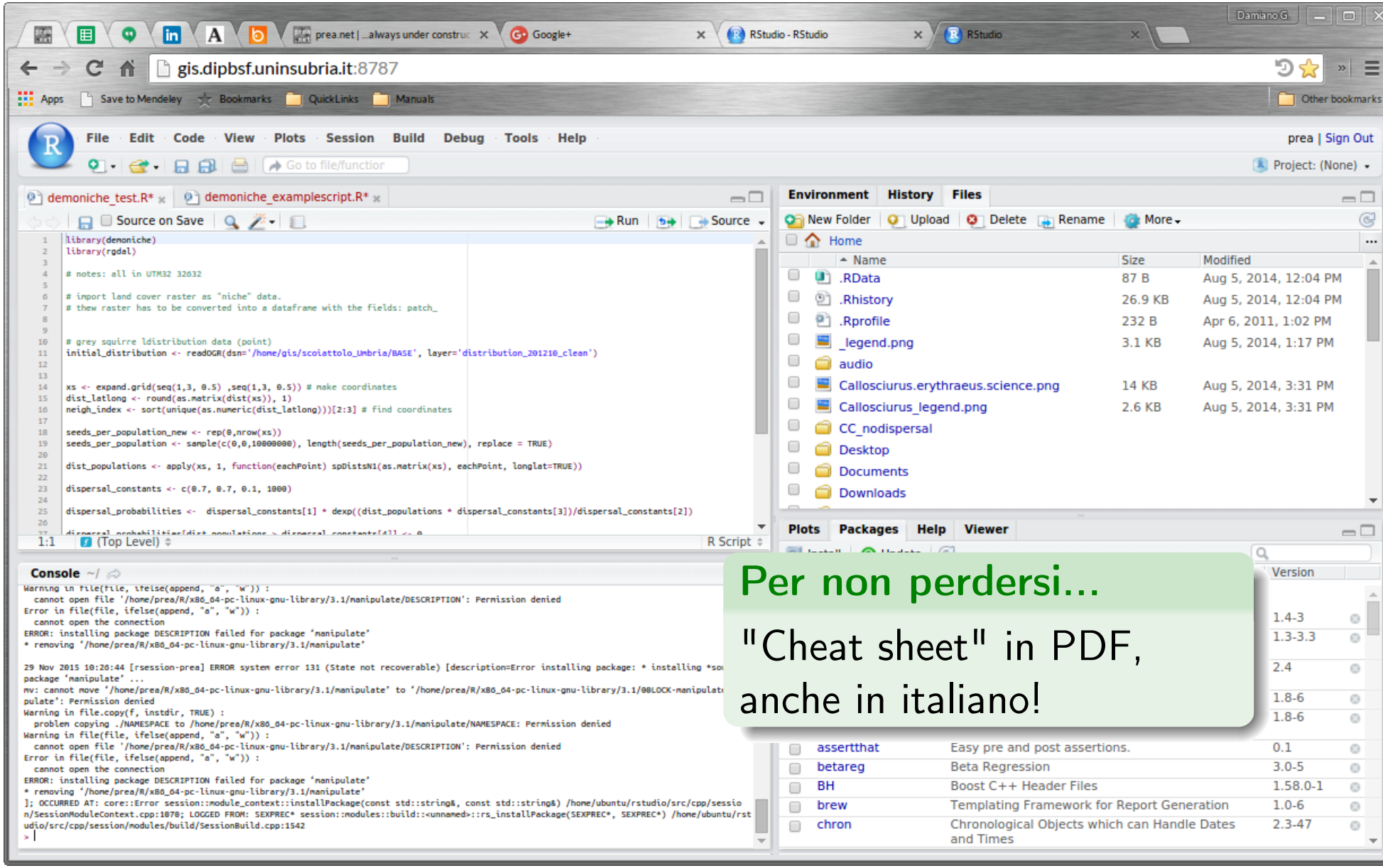
Benvenuti a casa! ...RStudio server, Chrome browser



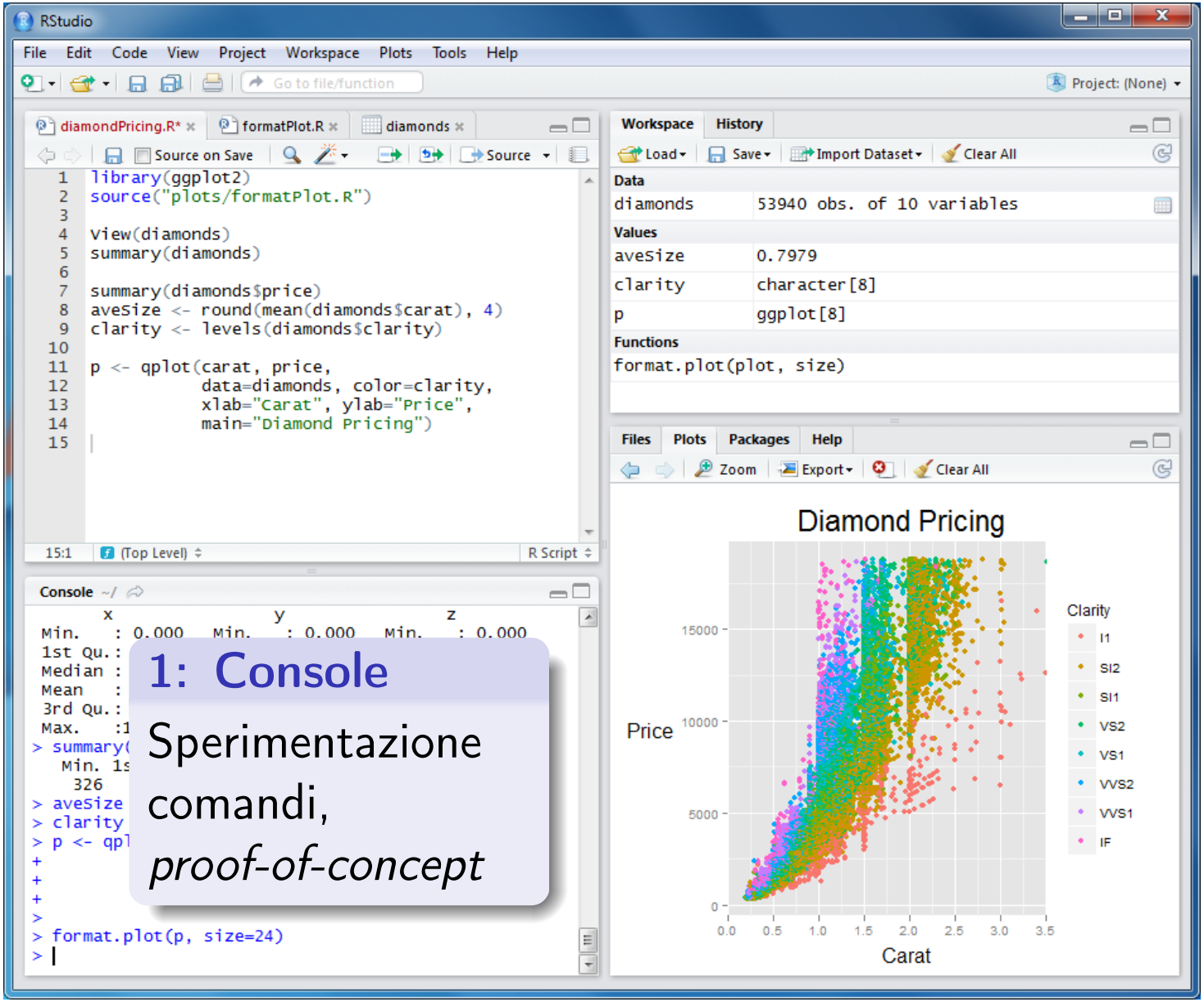
Benvenuti a casa! ... RStudio server, Chrome browser



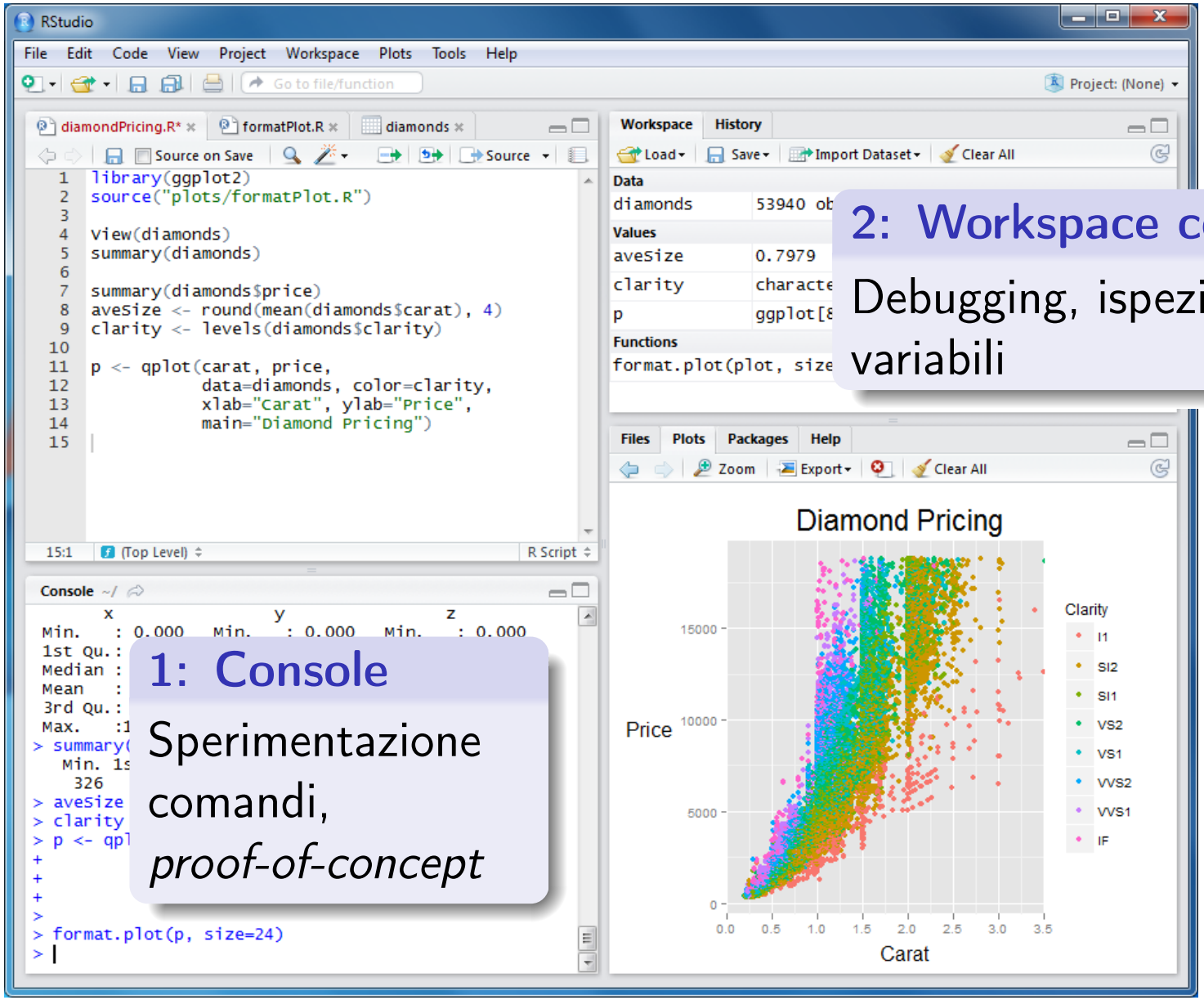
Benvenuti a casa! ...RStudio server, Chrome browser



IDE e flusso di lavoro



IDE e flusso di lavoro



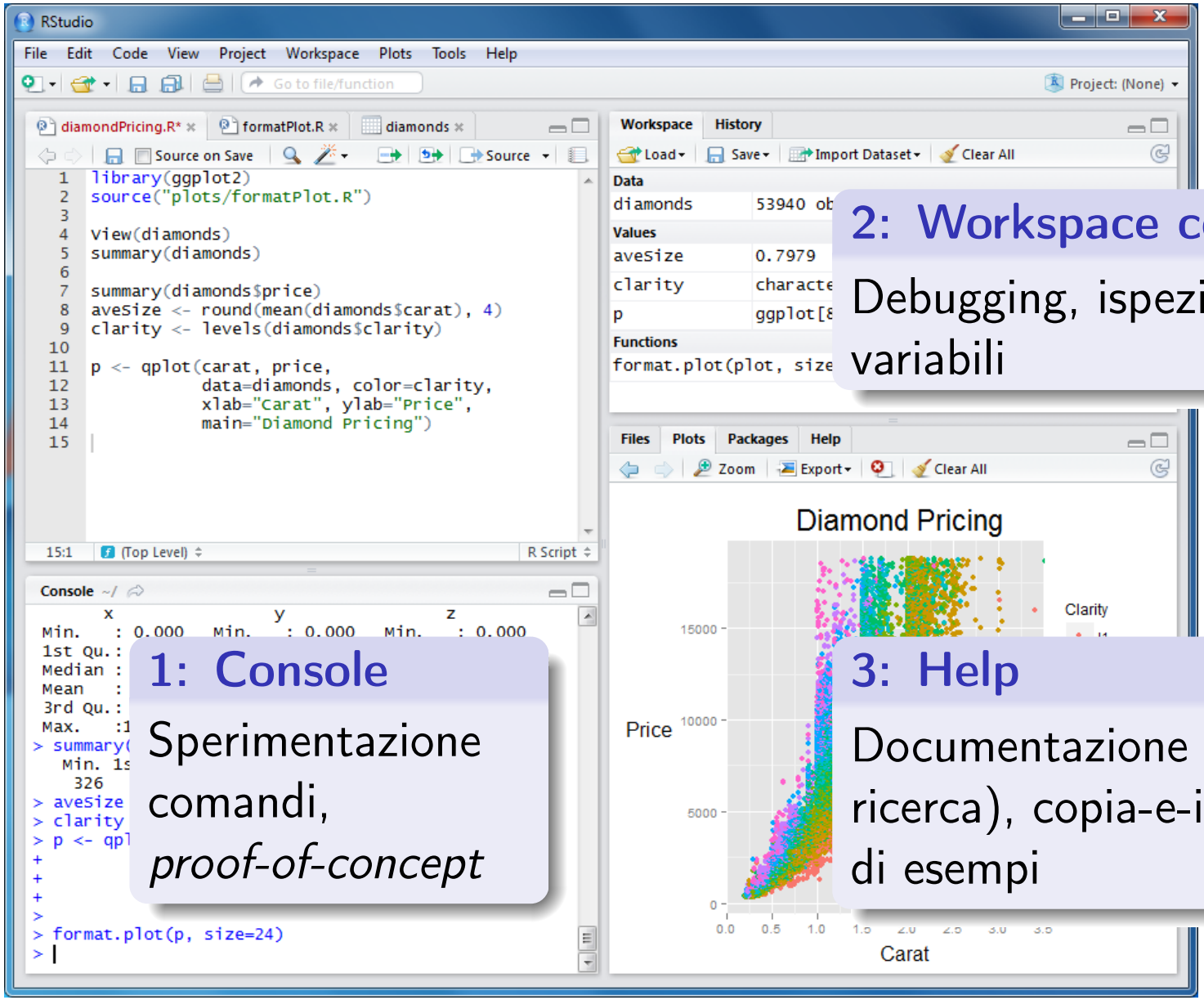
1: Console

Sperimentazione comandi, *proof-of-concept*

2: Workspace content

Debugging, ispezione variabili

IDE e flusso di lavoro



2: Workspace content

Debugging, ispezione variabili

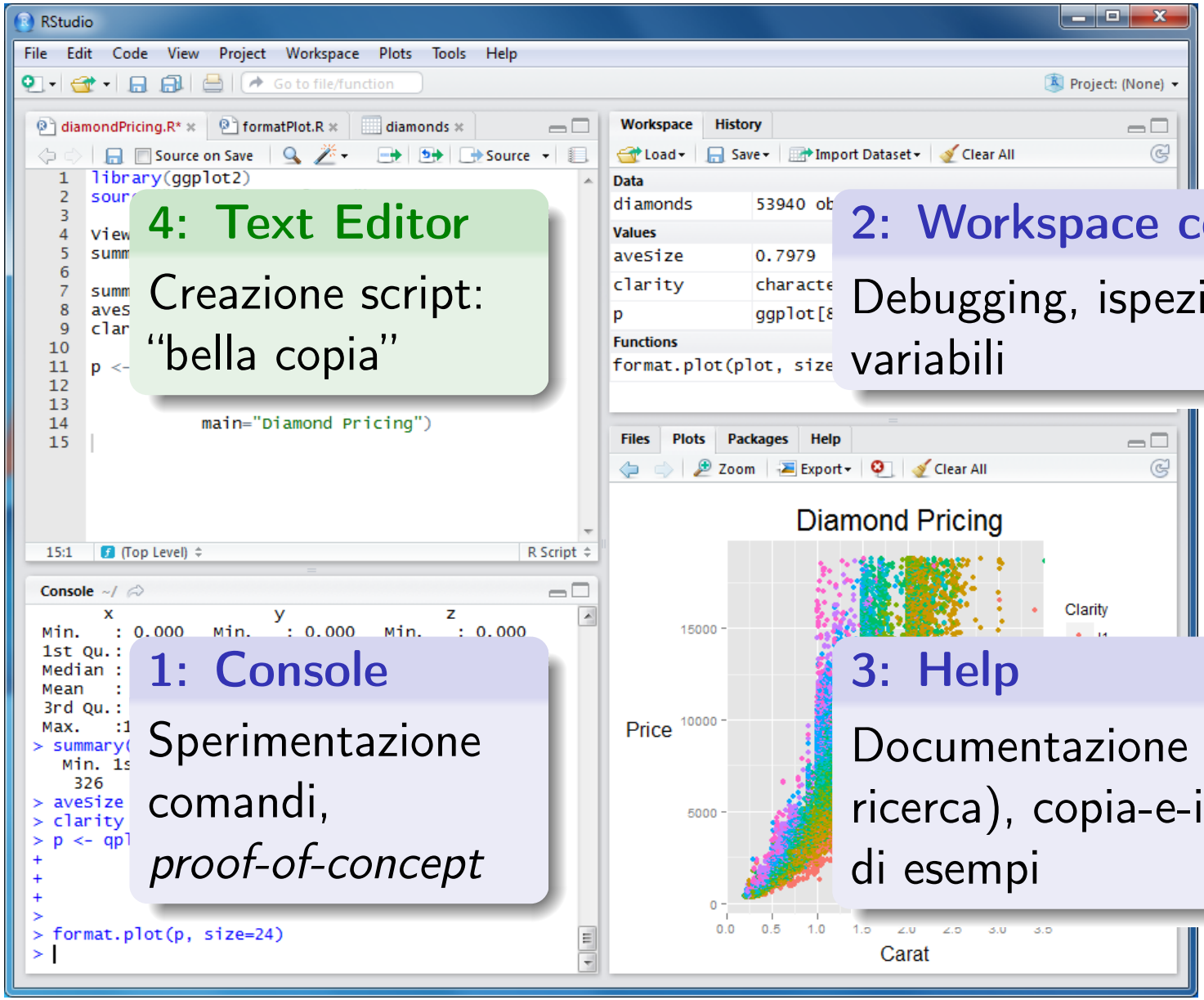
1: Console

Sperimentazione comandi, *proof-of-concept*

3: Help

Documentazione (inclusa ricerca), copia-e-incolla di esempi

IDE e flusso di lavoro



4: Text Editor

Creazione script: "bella copia"

2: Workspace content

Debugging, ispezione variabili

1: Console

Sperimentazione comandi, *proof-of-concept*

3: Help

Documentazione (inclusa ricerca), copia-e-incolla di esempi

Sì, ma... cosa devo scrivere?

R è un *linguaggio*.
Si tratta di apprenderne *grammatica* e *vocabolario*.
Occorre anche qualche idea riguardo ad alcuni concetti di *programmazione*...

Sì, ma... cosa devo scrivere?

R è un *linguaggio*.
Si tratta di apprenderne *grammatica* e *vocabolario*.
Occorre anche qualche idea riguardo ad alcuni concetti di *programmazione*...
Più si utilizza R, più si arricchisce il proprio "vocabolario".

Sommario

What is a programming language?

Un linguaggio di programmazione è un *linguaggio*, cioè una notazione di comodo per esprimere *idee*.
Un linguaggio di programmazione è un metodo pratico per definire e organizzare concetti riguardanti l'elaborazione di informazioni.

Requisiti di un linguaggio di programmazione

- rendere pratica l'elaborazione
- rendere efficiente l'uso degli elaboratori

Programmazione: perché studiarla?

- per migliorare la propria capacità di comprensione
- per migliorare le proprie capacità di *problem solving*
- per utilizzare al meglio uno strumento

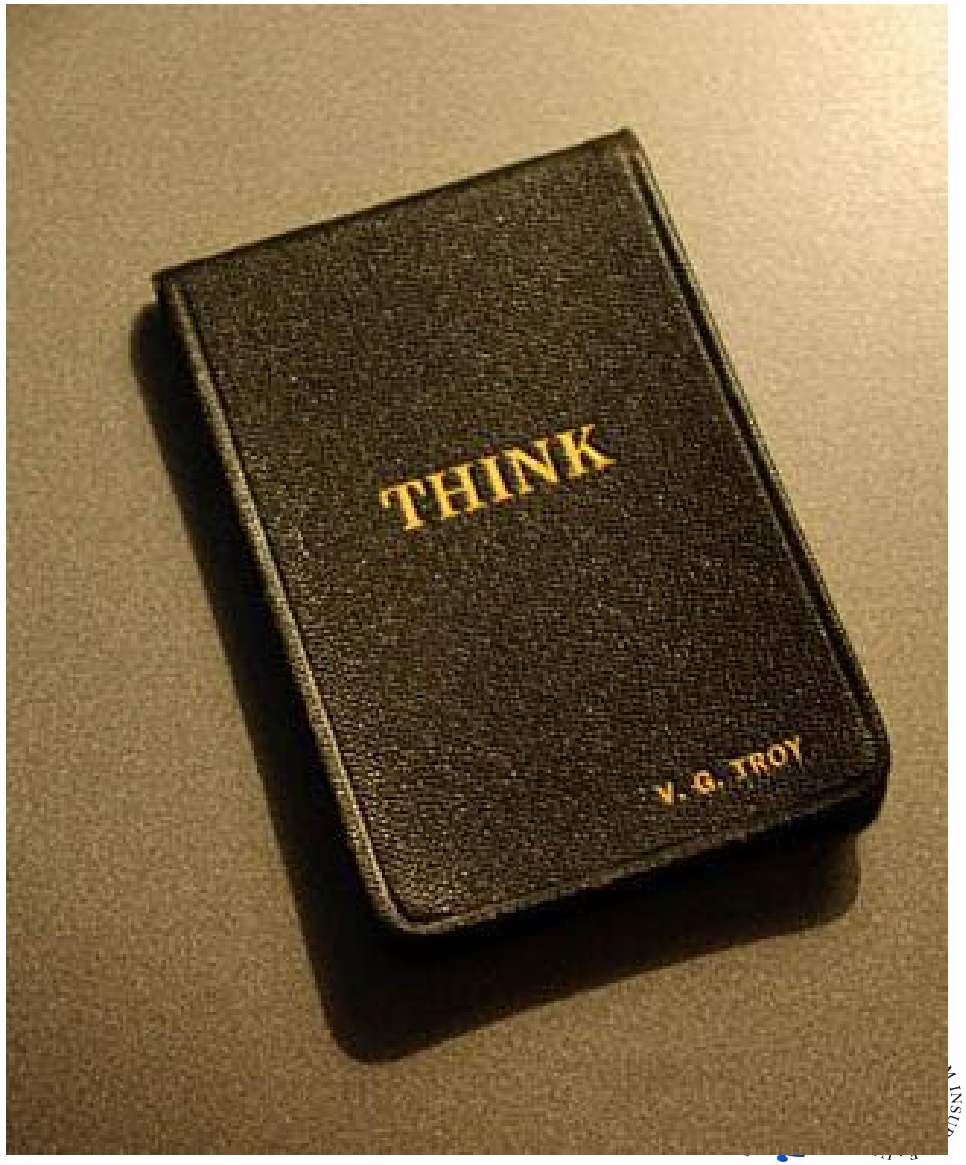
Programmare significa *risolvere problemi*.

What is programming?

Programmare non è soltanto ordinare a un calcolatore di svolgere determinate operazioni. È un metodo per organizzare le idee riguardo a un dato fenomeno o processo.

Programmare è un metodo per pensare.

Qualunque linguaggio di programmazione offre un metodo[8] per identificare concetti semplici e organizzarli in modo complesso, al fine di risolvere un problema.



[8] più o meno comodo...

Ingredienti

Nella programmazione si ha a che fare con due ingredienti di base:

- dati (informazione)
- procedure (operazioni da effettuare)

Dati e procedure vengono definiti e utilizzati mediante tre meccanismi:

primitive (o espressioni primitive), la entità concettuali più elementari su cui un linguaggio si basa;

combinazione di primitive, per generare espressioni via via più complesse;

astrazione tramite la quale poter generalizzare e manipolare concetti complessi

Un esempio di programma...

Ingredients

3 eggplants, peeled and cut lengthwise into 1/2 inch thick slices
salt
1/4 cup olive oil
1 tablespoon butter
1 pound lean ground beef
salt to taste
ground black pepper to taste
2 onions, chopped
1 clove garlic, minced
1/4 teaspoon ground cinnamon
1/4 teaspoon ground nutmeg
1/2 teaspoon fines herbs
2 tablespoons dried parsley
1 (8 ounce) can tomato sauce
1/2 cup red wine
1 egg, beaten

4 cups milk
1/2 cup butter
6 tablespoons all-purpose flour
salt to taste
ground white pepper, to taste

1 1/2 cups freshly grated Parmesan cheese
1/4 teaspoon ground nutmeg

Un programma è una sorta di "ricetta"...

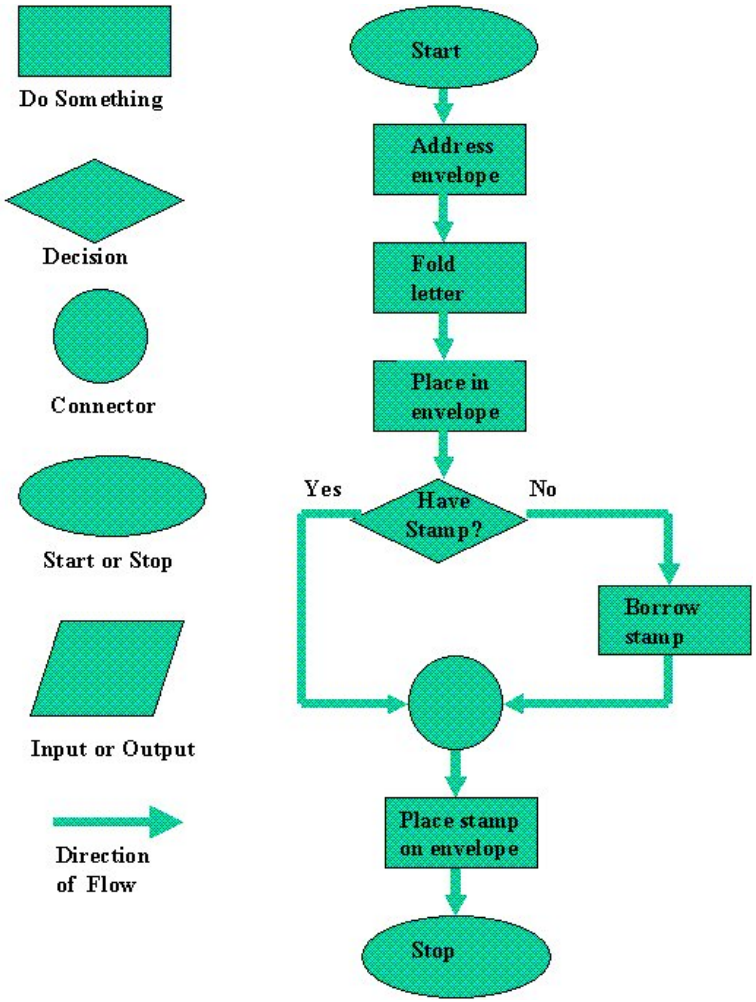
Un esempio di programma...

Directions

1. Lay the slices of eggplant on paper towels, sprinkle lightly with salt, and set aside for 30 minutes to draw out the moisture. Then in a skillet over high heat, heat the olive oil. Quickly fry the eggplant until browned. Set aside on paper towels to drain.
2. In a large skillet over medium heat, melt the butter and add the ground beef, salt and pepper to taste, onions, and garlic. After the beef is browned, sprinkle in the cinnamon, nutmeg, fines herbs and parsley. Pour in the tomato sauce and wine, and mix well. Simmer for 20 minutes. Allow to cool, and then stir in beaten egg.
3. To make the bechamel sauce, begin by scalding the milk in a saucepan. Melt the butter in a large skillet over medium heat. Whisk in flour until smooth. Lower heat; gradually pour in the hot milk, whisking constantly until it thickens. Season with salt, and white pepper.
4. Arrange a layer of eggplant in a greased 9x13 inch baking dish. Cover eggplant with all of the meat mixture, and then sprinkle 1/2 cup of Parmesan cheese over the meat. Cover with remaining eggplant, and sprinkle another 1/2 cup of cheese on top. Pour the bechamel sauce over the top, and sprinkle with the nutmeg. Sprinkle with the remaining cheese.
5. Bake for 1 hour at 350 degrees F (175 degrees C).

Un programma è una sorta di "ricetta"...

Un esempio di programma...



Un programma è una sorta di "ricetta"... ...per "cucinare" soluzioni a problemi definiti.

Cosa fa un programmatore?

- Definisce un problema

Cosa fa un programmatore?

- Definisce un problema
- Trova una soluzione

  (. . . o più di una)

Cosa fa un programmatore?

- Definisce un problema
- Trova una soluzione

  (. . . o più di una)
- Traduce la soluzione in istruzioni per il calcolatore

  (cioè scrive un programma!)

Cosa fa un programmatore?

- Definisce un problema
- Trova una soluzione

  (. . . o più di una)
- Traduce la soluzione in istruzioni per il calcolatore

  (cioè scrive un programma!)
- Verifica che il programma funzioni correttamente

  (il cosiddetto *debugging*)

Cosa fa un programmatore?

- Definisce un problema
- Trova una soluzione

  (. . . o più di una)
- Traduce la soluzione in istruzioni per il calcolatore

  (cioè scrive un programma!)
- Verifica che il programma funzioni correttamente

  (il cosiddetto *debugging*)
- Se necessario individua eventuali errori e li corregge

Cosa fa un programmatore?

- Definisce un problema
- Trova una soluzione

  (. . . o più di una)
- Traduce la soluzione in istruzioni per il calcolatore

  (cioè scrive un programma!)
- Verifica che il programma funzioni correttamente

  (il cosiddetto *debugging*)
- Se necessario individua eventuali errori e li corregge
- Documenta ciò che ha fatto

  (per facilitare l'uso del programma e l'eventuale manutenzione)

Definire il problema

La definizione di un problema consiste nell'identificare *ciò che è noto*, cioè quali sono le informazioni in ingresso e *ciò che si vuole ottenere* (i dati in uscita).
È la parte più difficile del lavoro...

Definire il problema

Identificare ciò che è noto

Spesso anche un problema apparentemente banale, quando deve essere tradotto in un programma, presenta difficoltà.

- I termini utilizzati per descrivere il problema sono ambigui?
- Identificare i processi che determinano il problema.

Comunicare con chi dovrà usare (o chi ha richiesto[9]) il programma.

[9] . . . o con la vostra papera di gomma. Greg Duff, su IBM 3270, circa 1980.

Trovare una soluzione[10]

Schematizzare le possibili soluzioni può essere d'aiuto per definire la *strategia di programmazione*.
Esistono due tecniche di base:

Flowcharting schematizzare i vari passaggi creando un *diagramma di flusso*

Pseudocoding delineare i vari passaggi *per iscritto*

[10] . . . o anche due.

Trovare una soluzione

Flowcharting

Un diagramma di flusso è una rappresentazione grafica, passo per passo, delle "azioni" che un programma svolge.
Si utilizzano poligoni per rappresentare le singole azioni e frecce per indicarne lo svolgimento.
È una **mappa** del programma.
Esiste una norma ANSI che definisce i simboli utilizzabili.

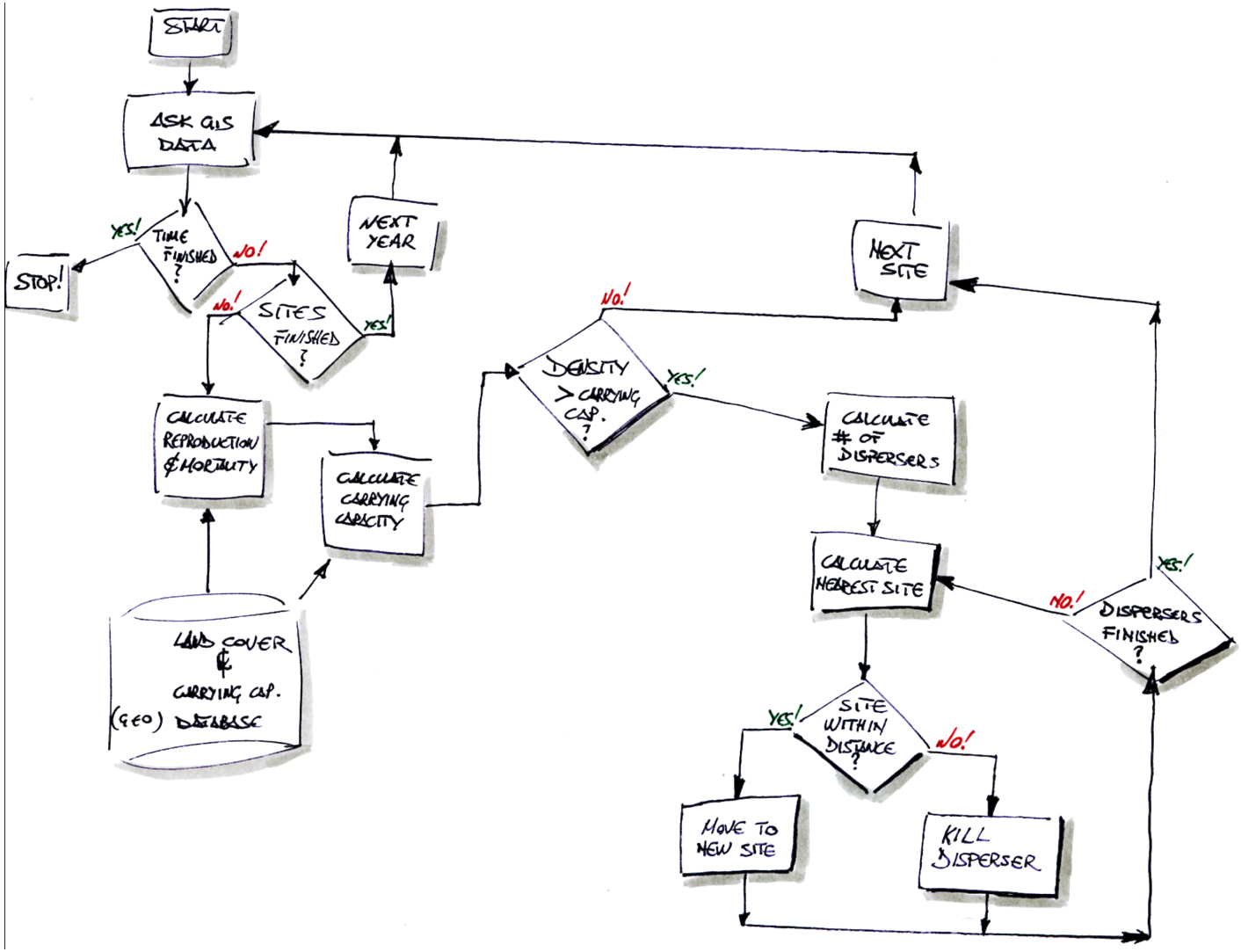
Trovare una soluzione

Pseudocoding

La pseudocodifica consiste nello scrivere uno *pseudo-programma*, in una sorta di linguaggio che non è un vero e proprio linguaggio (formale) di programmazione, ma è già molto simile ad esso.

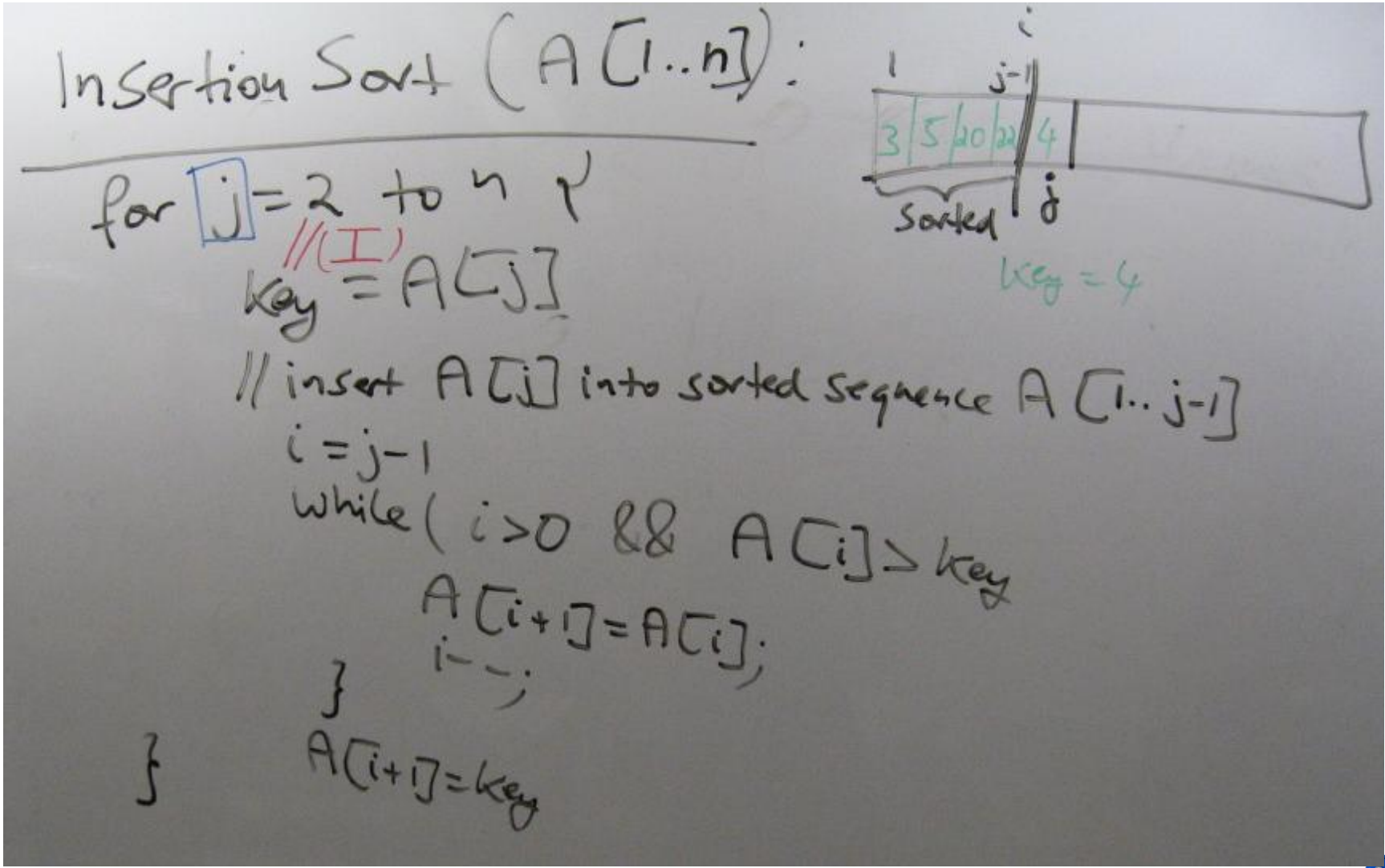
Trovare una soluzione

Un esempio di *flowchart*



Trovare una soluzione

Un esempio di pseudocodice



Trovare una soluzione

Lo scopo della rappresentazione *informale* (pseudocodifica, diagramma di flusso) del programma, in ultima analisi, è quello di ottenere un *modello concettuale* del programma stesso, propedeutico alla *rappresentazione formale*, cioè al programma vero e proprio.
Sia la pseudocodifica che il diagramma di flusso *non* dipendono da nessuna specifica formale di linguaggio.

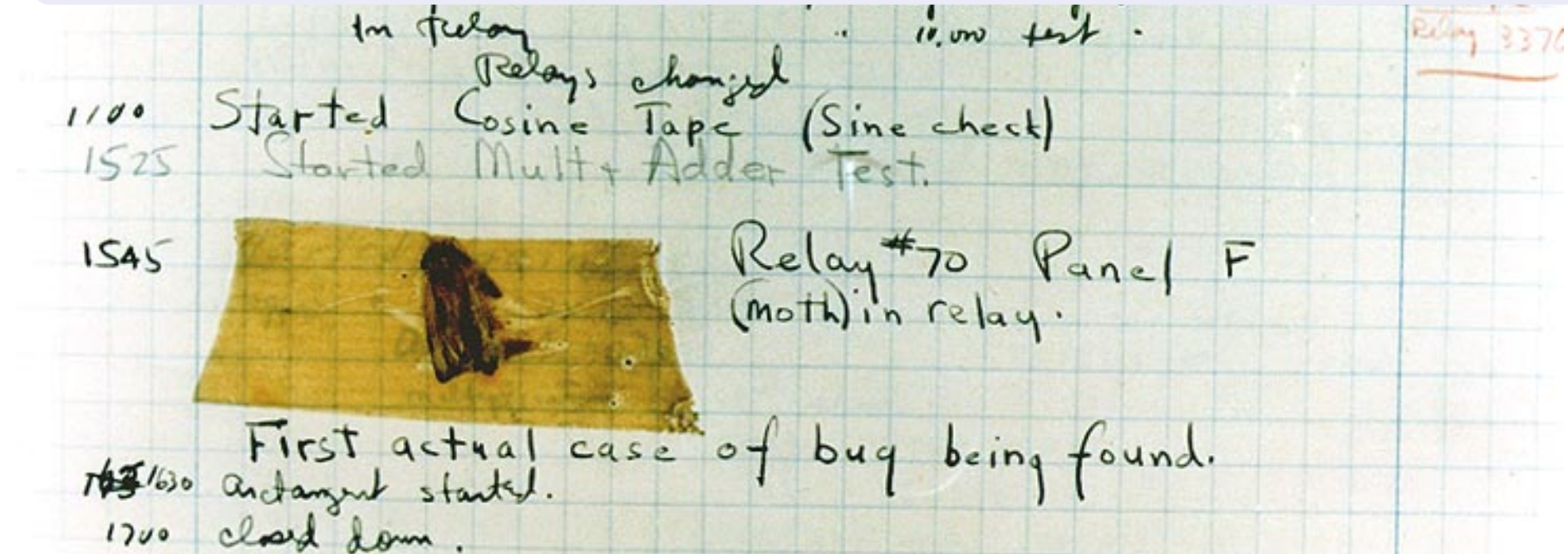
Scrivere il programma

Il "codice sorgente"

Il modello concettuale (informale), viene tradotto secondo precise "regole di grammatica" in un dato *linguaggio formale* (linguaggio di programmazione). Come nei linguaggi umani (scritti e parlati), non c'è nessuna garanzia che una "frase" grammaticalmente corretta abbia un qualsiasi significato...

Verificare il programma

Nonostante in campo pratico si sostenga che un programma ben progettato può essere scritto correttamente sin dall'inizio...
Nonostante alcuni teorici insistano sul fatto che esistano metodi formali per provare che un programma è "corretto"...



Verificare il programma

. . . Inevitabilmente un programma *può* contenere errori.

errori di "grammatica" refusi, sgrammaticature, segni di interpunzione. . .

errori di "sintassi" costrutti errati, uso scorretto o inesatto di tecniche o strutture dati. . .

errori "ideologici" cattiva progettazione. . .

Verificare il programma

Caccia all'errore

Esistono diverse tecniche per scovare e correggere eventuali errori:

Rilettura esattamente come si rilegge un testo scritto, rileggere (o meglio *far rileggere a un'altra persona* il proprio codice)

Verificare il programma

Caccia all'errore

Esistono diverse tecniche per scovare e correggere eventuali errori:

Rilettura esattamente come si rilegge un testo scritto, rileggere (o meglio *far rileggere a un'altra persona* il proprio codice)

Desk-checking rileggere il codice, seguendo il flusso delle operazioni del programma[11]. In alcuni casi, è previsto che il codice venga riletto e verificato da altri programmatori[12]!

[11] È sempre possibile rileggerlo e spiegarlo alla propria papera di gomma...

[12] Tipicamente, nel campo *Open Source*

Verificare il programma

Caccia all'errore

Esistono diverse tecniche per scovare e correggere eventuali errori:

Rilettura esattamente come si rilegge un testo scritto, rileggere (o meglio *far rileggere a un'altra persona* il proprio codice)

Desk-checking rileggere il codice, seguendo il flusso delle operazioni del programma[11]. In alcuni casi, è previsto che il codice venga riletto e verificato da altri programmatori[12]!

Traduzione un traduttore (o *compilatore*) è un programma che traduce il codice sorgente in forma utilizzabile direttamente dal calcolatore: eventuali errori vengono intercettati e notificati.

[11] È sempre possibile rileggerlo e spiegarlo alla propria papera di gomma...

[12] Tipicamente, nel campo *Open Source*

Verificare il programma

Caccia all'errore

Esistono diverse tecniche per scovare e correggere eventuali errori:

Rilettura esattamente come si rilegge un testo scritto, rileggere (o meglio *far rileggere a un'altra persona* il proprio codice)

Desk-checking rileggere il codice, seguendo il flusso delle operazioni del programma[11]. In alcuni casi, è previsto che il codice venga riletto e verificato da altri programmatori[12]!

Traduzione un traduttore (o *compilatore*) è un programma che traduce il codice sorgente in forma utilizzabile direttamente dal calcolatore: eventuali errori vengono intercettati e notificati.

Debugging si tenta di "far girare" il programma, seguendone passo per passo il funzionamento, verificando la correttezza dello svolgimento delle operazioni.

[11] È sempre possibile rileggerlo e spiegarlo alla propria papera di gomma...

[12] Tipicamente, nel campo *Open Source*

Documentare il programma

Alcuni programmatori pensano che un programma ben scritto si documenti da sè...
La documentazione è parte integrante di un programma, e non ha senso *scriverla dopo*.

- Origine e natura del problema
- Strategia adottata per risolverlo
- Rappresentazioni logiche (diagrammi di flusso, pseudocodice)
- Descrizione dei test di verifica e dei risultati attesi
- Modalità di utilizzo

Documentare il programma

Codice che "si documenta da sè"

La documentazione è essenziale

- per tenere traccia del lavoro svolto
- per lasciare una traccia a chi dovrà lavorare sul codice
- per poter riutilizzare in futuro codice già scritto
- per comunicare con altri

> *I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: "Literate Programming".*
>
> *– Donald E. Knuth (1984), Literate Programming*

Sommario

Alcune convenzioni

- R è *case sensitive*
- Comandi (in realtà *funzioni*) e *parole riservate*
- Variabili: possono avere qualunque nome
- Un comando viene terminato con `Enter` (o `Invio`), o con ";"
- È possibile creare comandi *compositi* racchiudendoli tra graffe ({ e })[13]
- È possibile inserire *commenti* dovunque: un commento inizia con "#", i caratteri seguenti sino a fine riga vengono ignorati
- Se un comando non è grammaticalmente completo, R lo rende noto mediante il simbolo "+"

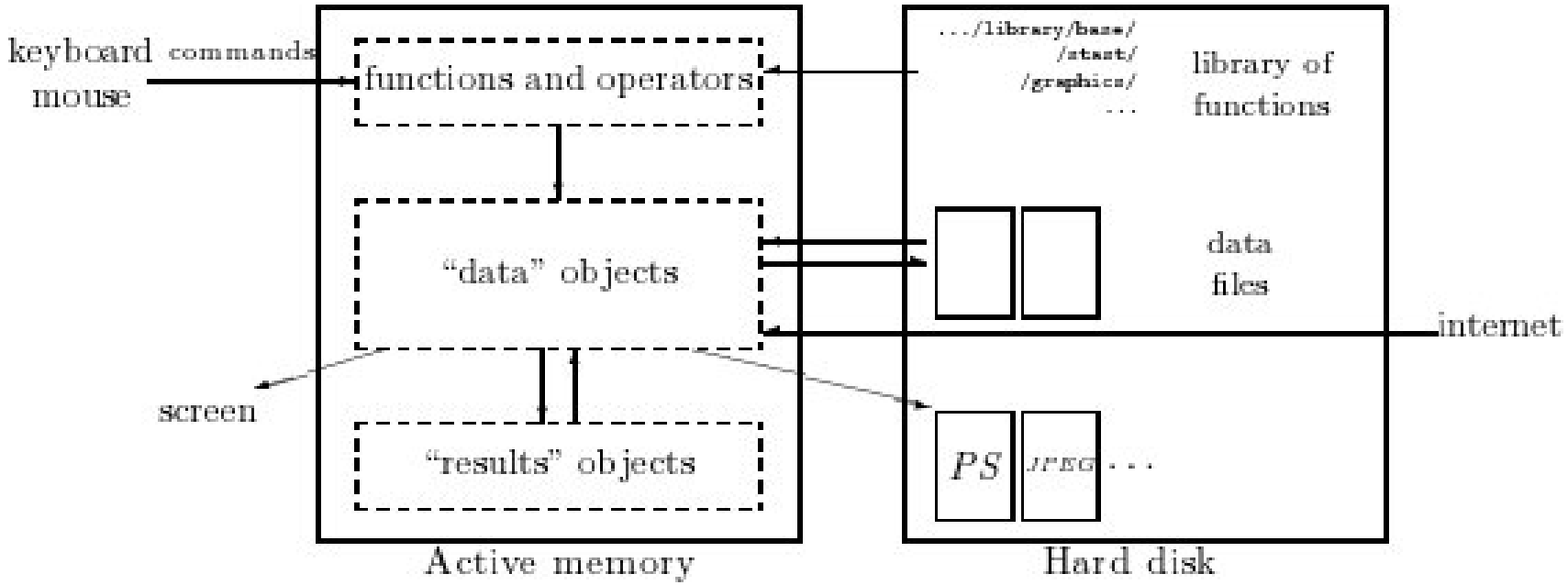
[13] in R, come in molti progammi e linguaggi di derivazione UNIX, sono comuni caratteri non presenti sulla tastiera italiana, quali { } ~ ecc.

I "personaggi"

- R immagazzina **tutto** in memoria (RAM) variabili, dati, funzioni, risultati ecc.[14]
- in R qualunque "cosa" è un *oggetto*, cioè un insieme di dati e di funzioni per manipolarli

[14] Il che non esclude che si possa lavorare con la *memoria di massa* (disco fisso, dischi remoti, web, ecc.)

In R "tutto è un oggetto"



Un esempio

File: src/example_01.R

```
# let a new variable be, called 'n', with value equal to 10 + 2
n <- 10 + 2
# o R, what is the value of n?
n

# now n will be 3 + a random gaussian deviate (mean 1, sd 1)
n <- 3 + rnorm(1)
# o R, what is the value of n?
n

# and now?
n <- 3 + rnorm(100)
```

Un esempio

File: src/example_01.R

```
1  # let a new variable be, called 'n', with value equal to 10 + 2
   n <- 10 + 2
3  # o R, what is the value of n?
   n
5
   # now n will be 3 + a random gaussian deviate (mean 1, sd 1)
7  n <- 3 + rnorm(1)
   # o R, what is the value of n?
9  n

11 # and now?
   n <- 3 + rnorm(100)
```

Ma che succede?

- `n` inizialmente è un numero intero
- successivamente (riga 7) `n` diventa un numero reale...
- infine, `n` diventa un *vettore* di 100 numeri reali!!!

Tutto in R è un oggetto...

- Un oggetto è una combinazione di dati e dei metodi per manipolarli
- Un oggetto "sa" gestirsi
- Un oggetto può "discendere" da altri oggetti, e ne eredita le "capacità" (*ereditarietà*)
- Diversi oggetti "sanno" compiere azioni simili (*polimorfismo*)

Tutto in R è un oggetto...

File: src/example_02.R

```
# load some canned data...
data(iris)
data(sunspots)
data(precip)
# what sort of R objects we have?
ls()
# and of what kind are they?
class(iris)
class(sunspots)
class(precip)
# a dataframe, a time series, a named numeric vector...and yet
summary(iris)
summary(sunspots)
summary(precip)
```

Ma... che cos'è un *oggetto*?

Un oggetto (o *classe*) è un modello astratto di *dati* e *metodi* per operare su di essi: si parla in questo caso di *incapsulamento* di dati e funzioni collegate ai dati.

La programmazione a oggetti permette di sfruttare alcune tecniche:

incapsulamento "tutto è contenuto nell'oggetto": una volta creata una classe di oggetti, li si può utilizzare sensa preoccuparsi di come sia il loro "funzionamento interno"

polimorfismo diversi oggetti possono "fare cose simili in modo diverso": è possibile creare metodi con lo stesso nome, che assolvono a funzioni simili

ereditarietà una classe di oggetti può *discendere* da una classe *progenitrice*: la classe di oggetti "figlia" avrà automaticamente tutte le caratteristiche ereditate dalla classe "madre"

Le conseguenze (vantaggiose!)

- Grazie al *polimorfismo*, è frequente che un dato comando R, una volta appreso, serva in molti altri casi (es.: `summarize`)
- È possibile *estendere* il linguaggio[15], basandosi sulla classi già esistenti e modificandole grazie all'*ereditarietà*
- È possibile operare con strutture di dati parecchio complesse, senza che sia necessario conoscere il loro funzionamento interno, grazie all'*incapsulamento*

[15] . . . e siete caldamente incoraggiati a farlo, R è anche un linguaggio *funzionale*!

Comandi base

- Per assegnare un valore a una variabile si usa il simbolo <-
- Utilizzare come comando il nome di un oggetto, dà come risultato il *vedere* l'oggetto stesso
- Ricordarsi sempre che:
  - una funzione R ha obbligatoriamente le parentesi
  - R è *case-sensitive*
  - non si usa \ (*backslash*) ma / (*slash*)

Comandi base

- Per assegnare un valore a una variabile si usa il simbolo <-
- Utilizzare come comando il nome di un oggetto, dà come risultato il *vedere* l'oggetto stesso
- Ricordarsi sempre che:
  - una funzione R ha obbligatoriamente le parentesi
  - R è *case-sensitive*
  - non si usa \ (*backslash*) ma / (*slash*)

Cheat sheet!

È umanamente impossibile ricordarsi *tutti* i comandi...
R Cheat sheet

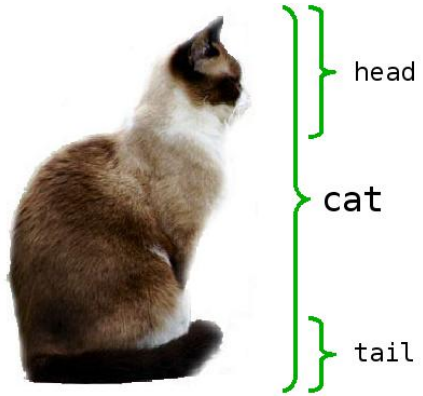
Manuale di sopravvivenza...

Molti comandi e costrutti i R imitano comandi e costrutti comuni in UNIX[16]

`ls()` `list`, elenco degli oggetti

`rm()` `remove`, elimina oggetti

`str()` `structure`, visualizza la struttura di un oggetto



`cat()` `concatenate`, concatena stringhe di testo, visualizza in forma grezza un oggetto

`head()` visualizza le prime parti di un oggetto

`tail()` visualizza le ultime parti di un oggetto

`help(),?` richiama il manuale

`help.search(),??` ricerca nel manuale

[16] Master Foo Discourses on Returning to Windows `http://catb.org/esr/writings/unix-koans/returning.html`

Un esempio: elenco oggetti

File: src/example_03.R

```
# what objects are present in the current R space?
ls

# ops.. I meant 'call the function ls', but what I wrote was instead
# 'show me how the object called `ls` is made inside'

# more politely
ls()
```

Un comando R è una *funzione*[17], e deve *sempre* "avere le parentesi"!!

[17] proprio nel senso di "$f(x)$!

R è *vettoriale*

Anziché operare con singoli valori (*scalari*), R assume di lavorare *sempre* su vettori.
Questo comporta:

1. l'enfasi sul fatto che una *variabile*[18] è un vettore;
2. il fatto che un'operazione si effettui applicandola *implicitamente* a tutti gli elemanti dell'intero vettore (R "ripete" da solo!)
3. un vettore in R non si limita solo a un vettore numerico...

File: src/example_04.R

```
1 # let's have 100 random numbers, normally distributed
  my.variable <- rnorm(100)
3
  # calculate a mean
5 mean(my.variable)

7 # what if I 'apply' a function?
  log.myvar <- log10(my.variable)
```

[18] Più appropriatamente, un campione di n realizzazioni di una variabile aleatoria

R è *estensibile*

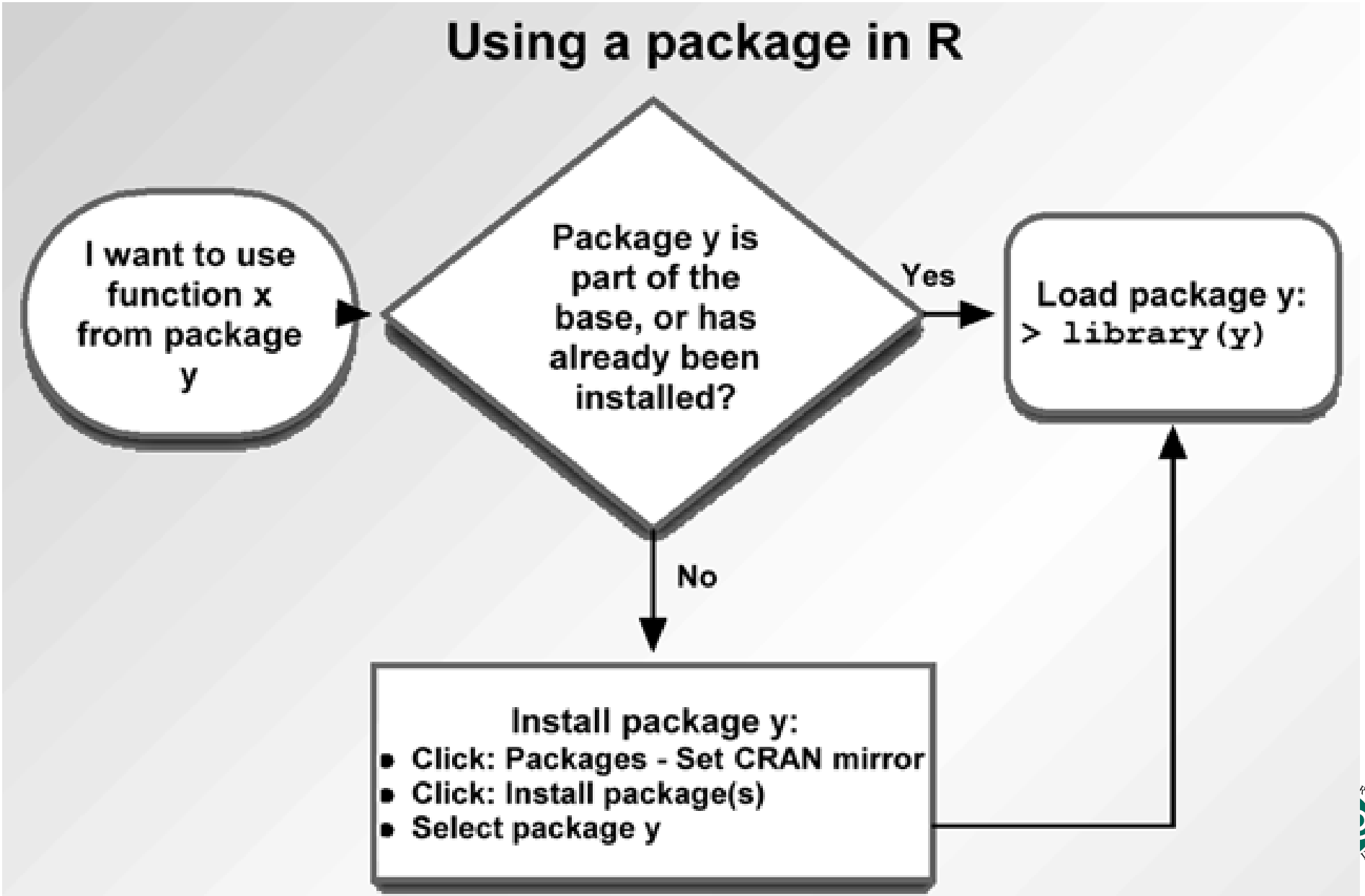
Mi servirebbe una funzione R per fare...

1. controllare gli *abstract* dei pacchetti esistenti
   `http://cran.r-project.org/web/packages/`
2. installare il pacchetto desiderato con
   `install.packages("a_package_name", dep=TRUE)`[19]
3. caricare il pacchetto con
   `library(a_package_name)`[20]

[19] selezionare il mirror CRAN più vicino

[20] esiste una sottile differenza tra `library()` e `require()`...

Mi servirebbe una funzione R per fare...



Alcune estensioni utili

`foreign`	legge/scrive i formati dati più comuni
`XLConnect`	legge/scrive i fogli elettronici Excel (occorre avere installato Java!)
`readxl`	legge i fogli elettronici Excel ("puro R")
`lattice`	*trellis graphics*
`ggplot2`	*Wilkinson-Wickham "grammar of graphics"*
`circular`	statistica circolare
`nlme`	*mixed-model* ANOVA
`rggobi`	analisi esplorativa di dati multidimensionali[21]
`ade4`	analisi multivariata
`bioconductor`	tutto per la bioinformatica (`http://www.bioconductor.org/`)

. . . e molti molti altri ancora!!

[21] purtroppo, dà problemi con RStudio. . .

L'unico comando da ricordare a memoria

?

L'unico comando da ricordare a memoria

?

This is all documented in TFM. Those who WTFM don't want to have to WTFM again on the mailing list. RTFM[22].

– Barry Rowlingson, R-help mailing list, October 2003

[22] Read That F... Manual

L'unico comando da ricordare a memoria

R: Cross Tabulation and Table Creation ▾ Find in Topic

table {base} R Documentation

Cross Tabulation and Table Creation

Description

`table` uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

Usage

```
table(..., exclude = if (useNA == "no") c(NA, NaN), useNA = c("no",
    "ifany", "always"), dnn = list.names(...), deparse.level = 1)

as.table(x, ...)
is.table(x)

## S3 method for class 'table'
as.data.frame(x, row.names = NULL, ...,
```

Come leggere una *manpage*

Qualsiasi comando R possiede una pagina di manuale (o *manpage*)
Una qualsiasi *manpage* è organizzata in *sezioni*

intestazione nome del comando, breve descrizione

Description descrizione per esteso delle funzionalità

Usage "grammatica" del comando

Arguments descrizione di dettaglio di quanto specificato nella sezione *Usage*

Details eventuali approfondimenti, riferimenti alla letteratura, peculiarità

Value com'è organizzato il *risultato*

Source riferimenti all'implementazione utilizzata

References bibliografia!

See Also rimandi ad altri comandi o a concetti generali

Examples esempi pratici (provare con il copia-e-incolla!)

Come usare una *manpage*

Ma come arrivo a *quella* pagina?

- Se non si conosce il nome di un comando:

  ```
  ??<comando, o qualcosa di simile>
  ```

- Se si conosce il comando ma non ci si ricorda le "regole di grammatica":

  ```
  ?<comando>
  ```

Come usare una *manpage*

Ma come arrivo a *quella* pagina?

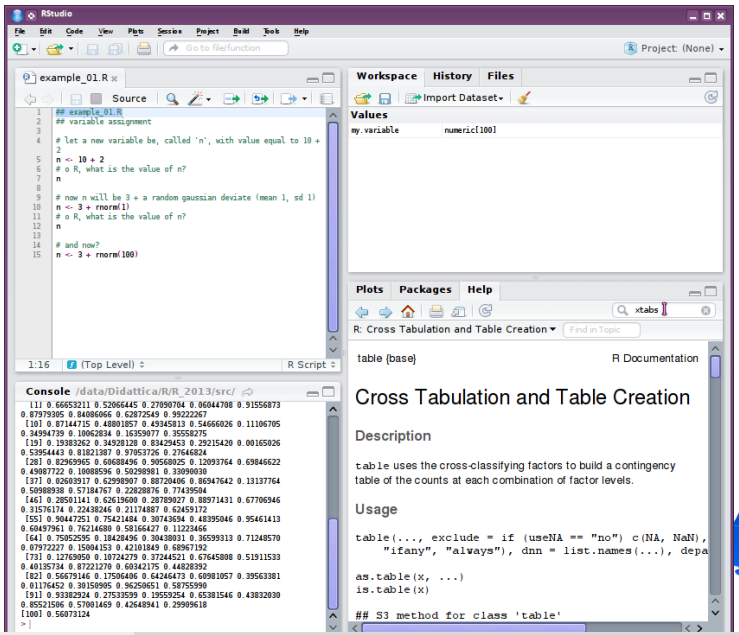
- Se non si conosce il nome di un comando:

  `??<comando, o qualcosa di simile>`

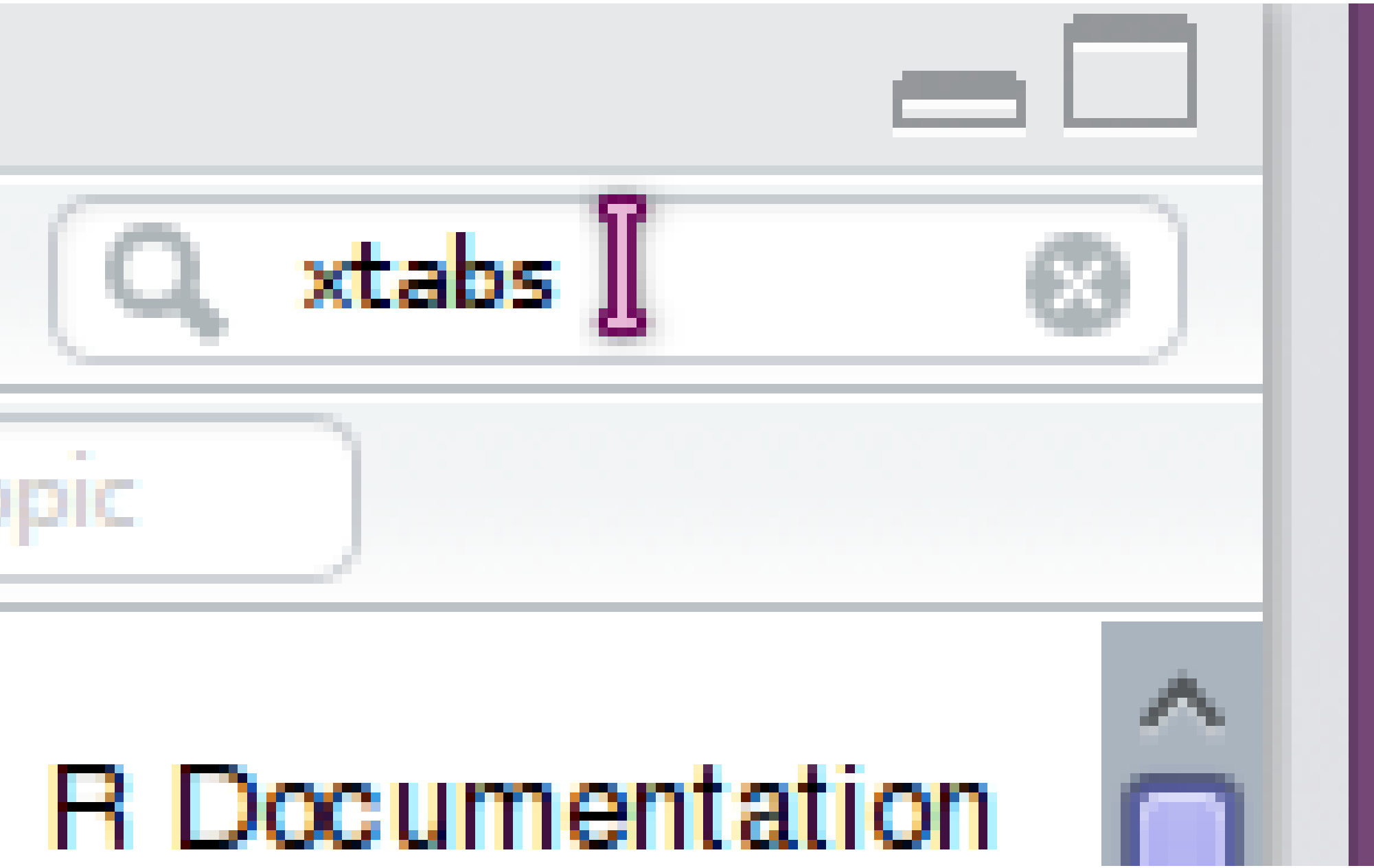
- Se si conosce il comando ma non ci si ricorda le "regole di grammatica":

  `?<comando>`

RStudio ha l'help integrato!



Come usare una *manpage*



Sommario

Analisi in R

The formulation of a problem is often more essential than its solution which may be merely a matter of mathematical or experimental skill.

– Albert Einstein

Analisi in R

The formulation of a problem is often more essential than its solution which may be merely a matter of mathematical or experimental skill.

– Albert Einstein

È più importante la *formulazione* di un problema che la sua soluzione.

Analisi in R

The formulation of a problem is often more essential than its solution which may be merely a matter of mathematical or experimental skill.

– Albert Einstein

È possibile scrivere un programma una volta che i termini del problema sono *ben definiti*

Analisi in R

Come definire un problema?

1. Comprendere la natura e l'origine dei dati.
2. Definire degli obiettivi.
3. *Keep it simple and stupid*
4. Tradurre il problema in termini statistici[a]
5. Non basta identificare un metodo per processare i dati... i risultati potrebbero comunque essere privi di senso!

[a] "Se riesci a fare un grafico, riesci a fare l'analisi"

Per iniziare. . .

1. importazione dei dati
2. "organizzazione" dei dati (*condizionamento*)
3. statistiche descrittive (`summary`, `stem`)
4. analisi grafica: `plot`, `xyplot`[22], `bwplot`[22], `qplot`[23], `ggplot`[23]
5. Ricerca (ed eliminazione) di eventuali *outlier*
6. Se necessario, *imputazione* di dati mancanti
7. Test per la normalità e relative conseguenti trasformazioni

[22] package `lattice`

[23] package `ggplot2`

Per iniziare. . .

1. importazione dei dati
2. "organizzazione" dei dati (*condizionamento*)
3. statistiche descrittive (`summary`, `stem`)
4. analisi grafica: `plot`, `xyplot`[22], `bwplot`[22], `qplot`[23], `ggplot`[23]
5. Ricerca (ed eliminazione) di eventuali *outlier*
6. Se necessario, *imputazione* di dati mancanti
7. Test per la normalità e relative conseguenti trasformazioni
8. Analisi statistica vera e propria
9. Restituzione dei risultati (grafici e/o testuali)

[22] package `lattice`

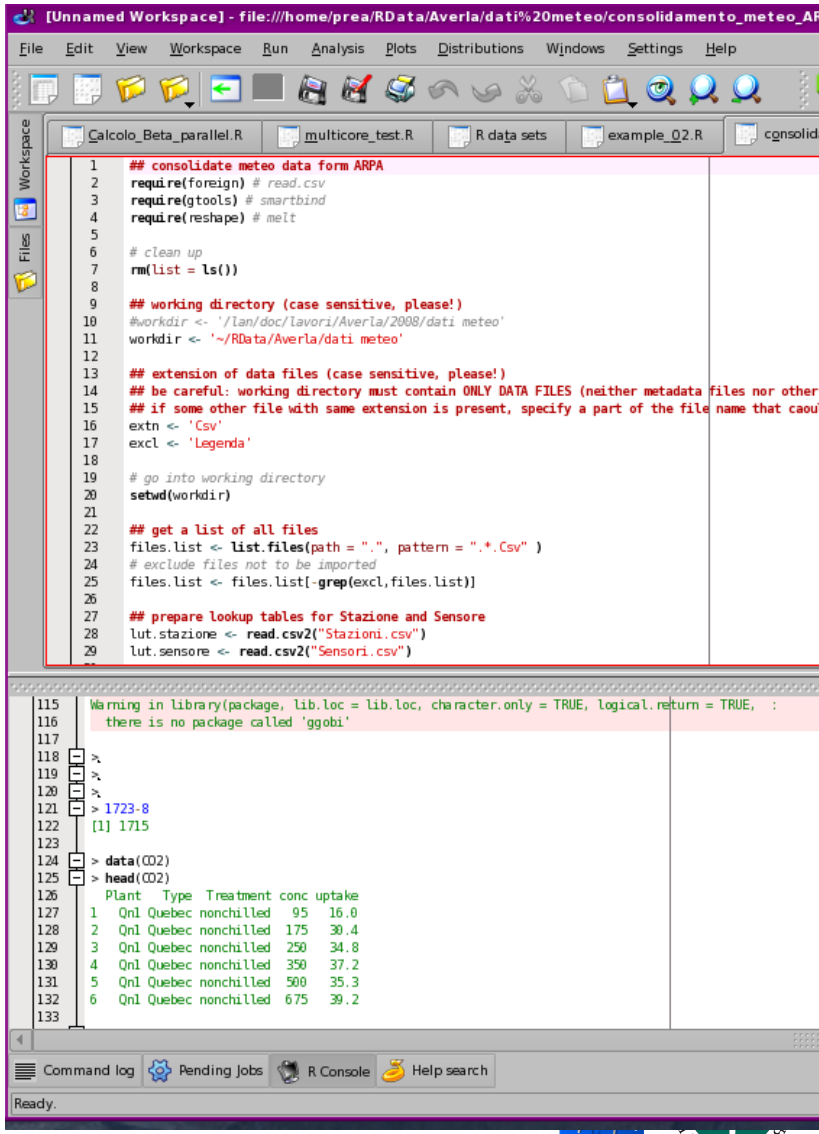
[23] package `ggplot2`

Ma. . . in pratica?

Le varie fasi del processo di elaborazione dati si concretizzano in differenti sezioni (consecutive, come in una ricetta di cucina...) di uno *script* in linguaggio R

Struttura base di uno *script* R

1. importazione dati
2. condizionamento dati
3. EDA (*Exploratory data analysis*[24])
4. analisi propriamente detta
5. grafici



[24] *sensu* Tukey, 1977

And now... for something completely different!

An Introduction to R http://cran.r-project.org/doc/manuals/R-intro.html

R for beginners http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

icebreakeR http://cran.r-project.org/doc/contrib/Robinson-icebreaker.pdf

Introduction to the R Project for Statistical Computing for use at ITC
http://cran.r-project.org/doc/contrib/Rossiter-RIntro-ITC.pdf

Esercitazioni di statistica biomedica–alcune note su R http://cran.r-project.org/doc/contrib/DellOmodarme-esercitazioni-R.pdf

R reference cards http://cran.r-project.org/doc/contrib/Short-refcard.pdf

Rootless `root`–The Unix Koans of Master Foo
http://catb.org/esr/writings/unix-koans/

R Search Engine http://rseek.org (ma esiste anche il comando `RSiteSearch`...)

Practical Regression and Anova using R
http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf


Programmazione statistica in R: corso teorico-pratico

Parte 2

Damiano G. Preatoni

Unità di Analisi e Gestione delle Risorse Ambientali – *Guido Tosi Research Group*
Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria

prea@uninsubria.it

7–9/7/2021

Contenuti

Sommario

Ma dove stanno i dati?

In R esistono diversi tipi di oggetti *"contenitore"*[25]
Ciascun tipo di "contenitore" (in R: oggetto dati) può contenere dati di diverso *"tipo"* (in R: modo)

[25] e numerosi altri vengono messi a disposizione da pacchetti specifici

Ma dove stanno i dati?

In R esistono diversi tipi di oggetti *"contenitore"*[25]
Ciascun tipo di "contenitore" (in R: oggetto dati) può contenere dati di diverso *"tipo"* (in R: modo)

Modi

- `logical`
- `integer`
- `numeric`
- `character`
- `factor`
- `...`

Data objects

- `vector`
- `matrix`
- `list`
- `dataframe`
- `...`

[25] e numerosi altri vengono messi a disposizione da pacchetti specifici

R “itera” da solo

In R si dà per scontato che un oggetto dati sia “iterabile”: qualsiasi operazione viene “applicata” a tutti gli elementi di quell’oggetto.

R "itera" da solo

In R si dà per scontato che un oggetto dati sia "iterabile": qualsiasi operazione viene "applicata" a tutti gli elementi di quell'oggetto.

Applicazione iterativa

File: src/example_01.R

```
1 n <- 3 + rnorm(100)
```

R aggiunge 3 a tutti gli elementi del vettore *n*

R “itera” da solo

In R si dà per scontato che un oggetto dati sia “iterabile”: qualsiasi operazione viene “applicata” a tutti gli elementi di quell’oggetto.
Se un’operazione coinvolge elementi di lunghezza diversa, l’elemento più corto viene *riciclato*

R "itera" da solo

In R si dà per scontato che un oggetto dati sia "iterabile": qualsiasi operazione viene "applicata" a tutti gli elementi di quell'oggetto.
Se un'operazione coinvolge elementi di lunghezza diversa, l'elemento più corto viene *riciclato*

R "ricicla"

File: src/example_05.R

```
1 a <- c(1,1,1,1,1,1,1,1)
  b <- c(4,0)
3
  a + b
```

R "ripete"il vettore *b* in modo da poter iterare su tutti gli elementi del vettore *a*

Sommario

Creare dati in R

Un po' di biometrie... (*Ammodramus caudacutus*, passero codacuta, N America [26])

Wing cord	Tarsus	Head	Weight
59	22.3	31.2	9.5
55	19.7	30.4	13.8
53.5	20.8	30.6	14.8
55	20.3	30.3	15.2
52.5	20.8	30.3	15.5
57.5	21.5	30.8	15.6
53	20.6	32.5	15.6
55	21.5	NA	15.7

[26] Chris Elphick, University of Connecticut, unpublished data, da Zuur et al., 2009, dataset originale: 1100 osservazioni.

Creare dati in R

Un po' di biometrie... (*Ammodramus caudacutus*, passero codacuta, N America [26])

Wing cord	Tarsus	Head	Weight
59	22.3	31.2	9.5
55	19.7	30.4	13.8
53.5	20.8	30.6	14.8
55	20.3	30.3	15.2
52.5	20.8	30.3	15.5
57.5	21.5	30.8	15.6
53	20.6	32.5	15.6
55	21.5	NA	15.7

Come inserire questi dati?

[26] Chris Elphick, University of Connecticut, unpublished data, da Zuur et al., 2009, dataset originale: 1100 osservazioni.

Creare dati in R

Esistono alcuni comandi di base per creare dati.

Creare dati in R

Esistono alcuni comandi di base per creare dati.

- `c()`
- `rep()`
- `matrix()`
- `list()`
- `data.frame()`

Creare dati in R . . . "a mano" I

File: src/ZUUR_CodeforChapter2.r

```
Wingcrd <- c(59, 55, 53.5, 55, 52.5, 57.5, 53, 55)
Wingcrd[1]

S.win <- sum(Wingcrd)
S.win


Tarsus <- c(22.3, 19.7, 20.8, 20.3, 20.8, 21.5, 20.6, 21.5)
Head <- c(31.2, 30.4, 30.6, 30.3, 30.3, 30.8, 32.5, NA)
Wt <- c(9.5, 13.8, 14.8, 15.2, 15.5, 15.6, 15.6, 15.7)

sum(Head, na.rm = TRUE)

BirdData <- c(Wingcrd, Tarsus, Head, Wt)
Id <- c(1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
        2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4)


rep(c(1, 2, 3, 4), each = 8)
rep(1 : 4, each = 8)
a <- seq(from = 1, to = 4, by = 1)
a

a <- seq(from = 1, to = 4, by = 1)
rep(a, each = 8)
```

Creare dati in R . . . "a mano" II

```
27
29 VarNames <- c("Wingcrd", "Tarsus", "Head", "Wt")
   Id <- rep(VarNames, each = 8)
31

33 Z <- cbind(Wingcrd, Tarsus, Head, Wt)
   n <- dim(Z)
35 n
   n <- dim(Z)[1]
37 n

39 W <- vector(length = 8)
   W[1] <- 59
41 W[2] <- 55
   W[3] <- 53.5
43 W[4] <- 55
   W[5] <- 52.5
45 W[6] <- 57.5
   W[7] <- 53
47 W[8] <- 55

49
   Dmat <- matrix(nrow = 8, ncol = 4)
51 Dmat

53 Dmat[, 1] <- c(59, 55, 53.5, 55, 52.5, 57.5, 53, 55)
   Dmat[, 2] <- c(22.3, 19.7, 20.8, 20.3, 20.8, 21.5,
```

Creare dati in R . . . "a mano" III

```
55                    20.6, 21.5)
   Dmat[, 3] <- c(31.2, 30.4, 30.6, 30.3, 30.3, 30.8,
57                    32.5, NA)
   Dmat[, 4] <- c(9.5, 13.8, 14.8, 15.2, 15.5, 15.6,
59                    15.6, 15.7)

61 Dmat

63 colnames(Dmat) <- c("Wingcrd", "Tarsus", "Head", "Wt")
   Dmat
65
   Dmat2 <- as.matrix(cbind(Wingcrd,Tarsus,Head, Wt))
67
   Dfrm <- data.frame(WC = Wingcrd, TS = Tarsus,
69                    HD = Head, W = Wt)
   Dfrm
71

73 M <- lm(WC ~ Wt, data = Dfrm)

75
   AllData <- list(BirdData = BirdData, Id = Id, Z = Z,
77                    VarNames = VarNames)

79 AllData <- list(BirdData, Id, Z, VarNames)
```

Creare “a mano”? . . . ma ne vale la pena?

File: src/ZUUR_CodeforChapter2_dataframe.R

```
DFrame <- data.frame(
  Wingcrd <- c(59, 55, 53.5, 55, 52.5, 57.5, 53, 55),
  Tarsus <- c(22.3, 19.7, 20.8, 20.3, 20.8, 21.5, 20.6, 21.5),
  Head <- c(31.2, 30.4, 30.6, 30.3, 30.3, 30.8, 32.5, NA),
  Wt <- c(9.5, 13.8, 14.8, 15.2, 15.5, 15.6, 15.6, 15.7)
)
```

Creare dati in R

Esistono alcuni comandi di base per creare dati.

- `c()`
- `rep()`
- `matrix()`
- `list()`
- `data.frame()`

Creare dati in R

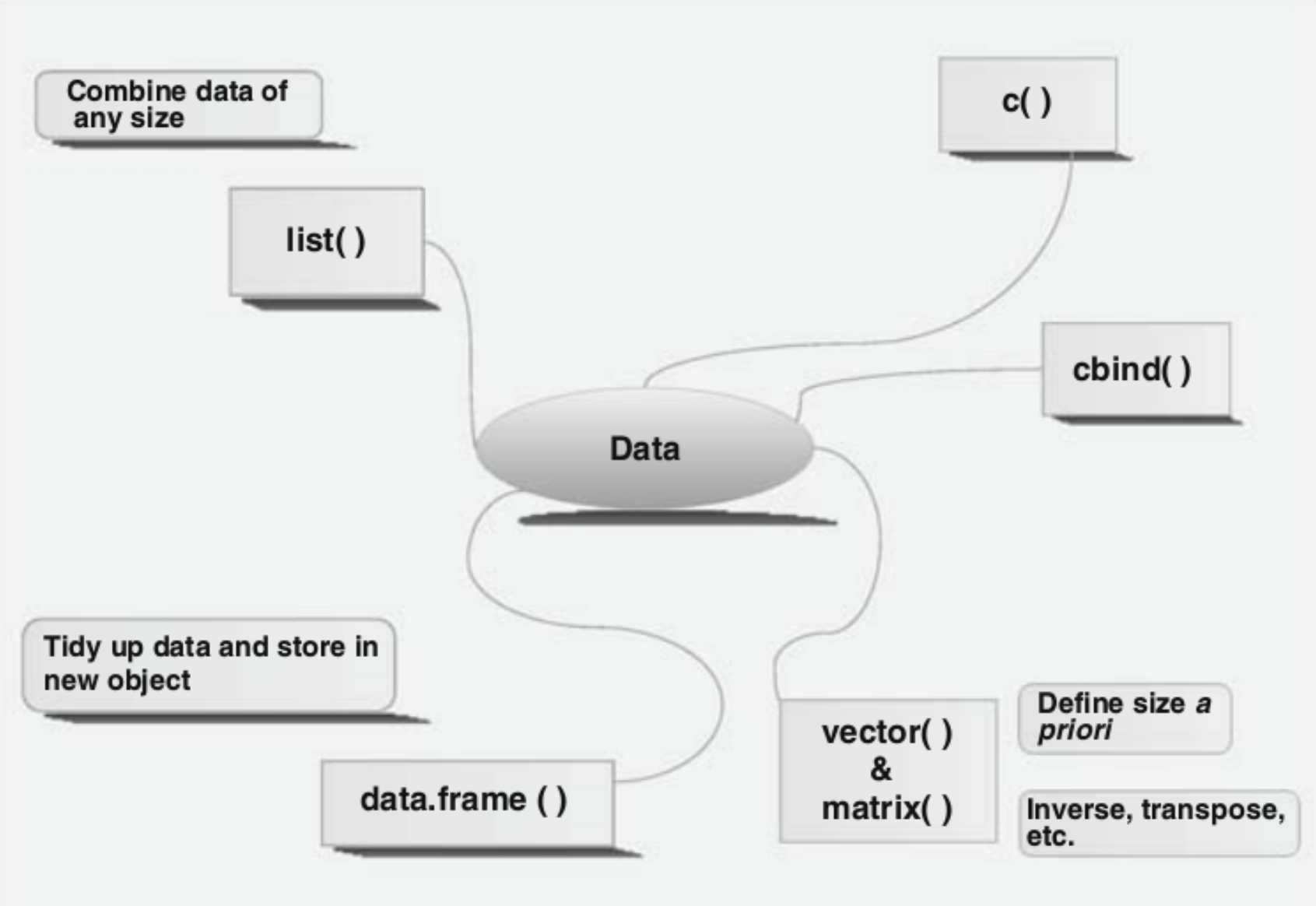
Esistono alcuni comandi di base per creare dati.

- `c()`
- `rep()`
- `matrix()`
- `list()`
- `data.frame()`

. . . ma i dati di solito sono già presenti in file esterni a R

Di norma, si tratta quasi sempre di individuare il sistema più comodo per *importare* file di dati esistenti in R
Tutti (o quasi) i comandi per importare creano dei *dataframe*

Creare dati in R



Comandi per caricare file di dati

`data(x)` carica un set di dati "di serie" in R

`read.table(...)` carica dati da un file di testo

`read.csv(...)` carica dati da un file di testo in formato `.csv`

`read.*(...)` forma generale...

Comandi per caricare file di dati

`data(x)` carica un set di dati "di serie" in R

`read.table(...)` carica dati da un file di testo

`read.csv(...)` carica dati da un file di testo in formato `.csv`

`read.*(...)` forma generale...

I comandi in R non sono difficili da imparare!

Quale potrebbe essere il comando per leggere un file Excel (`.xls`)?

Sommario

Dataframe

Il contenitore di dati più comune è il *dataframe*.

- analogo a un foglio elettronico ("righe e colonne")
- colonne "tipizzate" (una data colonna è in un dato "modo")
- un dataframe funziona come una matrice (indicizzazione per righe e colonne)
- le colonne hanno un nome

Dataframe

Il contenitore di dati più comune è il *dataframe*.

- analogo a un foglio elettronico ("righe e colonne")
- colonne "tipizzate" (una data colonna è in un dato "modo")
- un dataframe funziona come una matrice (indicizzazione per righe e colonne)
- le colonne hanno un nome

Un esempio

File: src/example_06.R

```
1 data(iris)
  # see the 'head' ;)
3 head(iris)
  # see the structure
5 str(iris)
  # see column names
7 names(iris)
```

Dataframe

Il contenitore di dati più comune è il *dataframe*.

- analogo a un foglio elettronico ("righe e colonne")
- colonne "tipizzate" (una data colonna è in un dato "modo")
- un dataframe funziona come una matrice (indicizzazione per righe e colonne)
- le colonne hanno un nome

Lavorare con i dataframe

File: src/example_06.R

```
1 # refer to a column
  iris$Species
3 iris[,'Species']
  iris[,5]
```

Ma... com'è fatto "dentro" un oggetto R?

In R è possibile creare dei "contenitori per dati" anche estremamente articolati.
È sempre possibile esaminare la *struttura* di un oggetto R mediante il comando `str()`.

Ma... com'è fatto "dentro" un oggetto R?

In R è possibile creare dei "contenitori per dati" anche estremamente articolati.
È sempre possibile esaminare la *struttura* di un oggetto R mediante il comando `str()`.

"Contenitori" e strutture dati

Poter disporre di una struttura definita (e apparentemente **rigida**) in cui immagazzinare i dati è un vantaggio:

- si "demanda alla macchina" il compito di tenere in ordine i dati, evitando incoerenze!
- si può comunque accedere a qualunque elemento di una struttura dati!

Dataframe: *slicing*

Un dataframe è una matrice "specializzata".
È possibile quindi utilizzare la notazione comunemente utilizzata per riferirsi a degli elementi della "matrice"

dataframe[<riga>, <colonna>]

Dataframe: *slicing*

Un dataframe è una matrice "specializzata".
È possibile quindi utilizzare la notazione comunemente utilizzata per riferirsi a degli elementi della "matrice"

dataframe[<riga>, <colonna>]

Notazione "tra quadre"

Entro parentesi quadre è possibile specificare *qualsiasi espressione* che sia valida per righe e/o colonne:

- indici (numerici o "per nome")
- espressioni logiche o algebriche ("filtraggio")

Dataframe: *slicing*

Un dataframe è una matrice "specializzata".
È possibile quindi utilizzare la notazione comunemente utilizzata per riferirsi a degli elementi della "matrice"

dataframe[<riga>, <colonna>]

Notazione "tra quadre"

File: src/example_06.R

```
1 # make 'slices'
  iris[1:3,'Sepal.Length']
3 # select by row: only a species
  iris[iris$Species=='setosa',]
5 # select by row: two species
  iris[iris$Species==c('setosa', 'versicolor'),]
7 # select by row: sepal length <= 5
  iris[iris$Sepal.Length <= 5,]
9 # select by row: sepal length under the mean
  iris[iris$Sepal.Length <= mean(iris$Sepal.Length),]
```

Dataframe: *slicing*

Un dataframe è una matrice "specializzata".
È possibile quindi utilizzare la notazione comunemente utilizzata per riferirsi a degli elementi della "matrice"

dataframe[<riga>, <colonna>]

Si può lavorare "dentro" un dataframe

Un dataframe può essere collegato al *percorso di ricerca* dell'ambiente R mediante il comando `attach()`: non è più necessario specificare il nome del dataframe.
Il comando "opposto" è `detach`.
Non è comodo[a] utilizzare `attach` quando si lavora con più dataframe contemporaneamente.

[a] ... anzi. è *vivamente sconsigliato!*

Liste

Un altro utile "contenitore" è la *lista* (`list`)

- collezione *eterogenea* di oggetti: può contenere *di tutto*!
- gli elementi di lista hanno un *nome*, oltre che una posizione
- funziona come un vettore, con qualche comodità in più
- è "naturalmente iterabile"

Liste: un esempio I

File: src/example_06b.R

```r
## Lists examples
n <- c(2, 3, 5) # a vector (of integers)
s <- c("aa", "bb", "cc", "dd", "ee") # a vector (of characters)
b <- c(TRUE, FALSE, TRUE, FALSE, FALSE) # a vector (of logicals)
x <- list(n, s, b, 3) # a list: x contains copies of n, s, b

# list slicing: square brackets
x[2]
x[c(2, 4)]
# direct access: _double_ square brackets
x[[2]]
# direct access is needed to _modify_ list elements
x[[2]][1] <- "ta"
x[[2]]
# the list contains a copy of s, which is unaffected
s
# named lists
v <- list(bob=c(2, 3, 5), john=c("aa", "bb"))
v
# named lists slicing
v["bob"]
# named lists direct access
v[["bob"]]
v$bob
```

Dataframe: *do it yourself* ! I

File: src/example_07.R

```
## example_07.R
## data frames, part II
data(mtcars)
# is there any documentation on this built-in data set?
?mtcars
# show the dataframe
mtcars
# size up the dataframe
nrow(mtcars)
ncol(mtcars)
str(mtcars)
# print cell value from the first row, second column
mtcars[1, 2]
# the same, using row and column names
mtcars["Mazda RX4", "cyl"]
# print a single column: type of transmission
mtcars[ ,9]
mtcars[[9]]
mtcars[["am"]]
mtcars$am
mtcars[,"am"]
# column slicing: get column 1
mtcars[1]
mtcars["mpg"]
```

Dataframe: *do it yourself*! II

```
26 # get miles per gallon and horsepower
   mtcars[c("mpg", "hp")]
28 # row slicing: print a single row
   mtcars[24,]
30 # rows 3 and 24
   mtcars[c(3, 24),]
32 # get a row 'by name'
   mtcars["Camaro Z28",]
34 mtcars[c("Datsun 710", "Camaro Z28"),]
   # place in a new dataframe all the cars with automatic transmission
36 automatic <-mtcars$am == 0
   mtcars[automatic,]
38 # the same, in one line
   mtcars[mtcars$am == 0,]
40 # mileage for cars with automatic transmission
   mtcars[mtcars$am == 0,]$mpg
```

Sommario

Per iniziare. . .

1. importazione dei dati
2. "organizzazione" dei dati (*condizionamento*)
3. statistiche descrittive (`summary`, `stem`)
4. analisi grafica: `plot`, `xyplot`[22], `bwplot`[22], `qplot`[23], `ggplot`[23]
5. Ricerca (ed eliminazione) di eventuali *outlier*
6. Se necessario, *imputazione* di dati mancanti
7. Test per la normalità e relative conseguenti trasformazioni
8. Analisi statistica vera e propria
9. Restituzione dei risultati (grafici e/o testuali)

[22] package `lattice`

[23] package `ggplot2`

Statistiche di base e analisi esplorativa

Una volta "acquisiti" i dati grezzi, è *indispensabile* "dare un'occhiata:

- è avveunto qualche pasticcio nell'importazione?
- i dati sono del tipo atteso (stringhe che sembrano numeri...)
- esistono valori "estremi" (*outliers*)?
- i dati richiedono qualche trasformazione?
- i dati devono essere "normali"?
- ...

Comandi per "esplorare"

`summary(x)`	fa il "sommario" di qualunque tipo di dato: statistiche descrittive per un intero dataframe
`stem(x)`	*stem-and-leaf plot*
`mean(x)`	media
`median(x)`	mediana
`sd(x)`	deviazione standard
`unique(x)`	(per variabili testuali) elenco dei valori, senza duplicati
`quantile(x,probs=)`	calcolo dei quantili (default: quartili)
	...

Comandi per "esplorare"

`summary(x)`	fa il "sommario" di qualunque tipo di dato: statistiche descrittive per un intero dataframe
`stem(x)`	*stem-and-leaf plot*
`mean(x)`	media
`median(x)`	mediana
`sd(x)`	deviazione standard
`unique(x)`	(per variabili testuali) elenco dei valori, senza duplicati
`quantile(x,probs=)`	calcolo dei quantili (default: quartili)
	...

Curiosamente, *non esistono* in R i comandi per calcolare l'errore standard e la varianza. . . come mai?

Comandi per "esplorare"

`summary(x)`	fa il "sommario" di qualunque tipo di dato: statistiche descrittive per un intero dataframe
`stem(x)`	*stem-and-leaf plot*
`mean(x)`	media
`median(x)`	mediana
`sd(x)`	deviazione standard
`unique(x)`	(per variabili testuali) elenco dei valori, senza duplicati
`quantile(x,probs=)`	calcolo dei quantili (default: quartili)

Curios

R è un linguaggio *funzionale*...

```
se <-function(x) sd(x)/sqrt(length(x))
se <-function(x) sqrt(var(x)/length(x))
```

standard e la vari

Analisi esplorativa: *do it yourself* !

Dati epidemiologici da Vicente et al. (2006).
Presenza di *Elaphostrongylus cervi* in esemplari di cervo e cihghiale (portatore sano).

- i dati sono in un file Excel? Come leggerlo?
- che struttura hanno i dati? Quante righe? Quante colonne?
- Cosa contengono le varie colonne?
- Il campione è omogeneo per *sesso*?
- Il campione è omogeneo per *anno*?
- Il campione è omogeneo per *allevamento*?
- I valori di lunghezza testa-corpo sono diversi tra maschi e femmine?

Analisi esplorativa: *do it yourself*! I

File: src/example_07b.R

```
## example_07b.R
## load and explore deer data (Vicente et al. 2006) from Zuur et al. 2009.

rm(list=ls()) # this clears the R workspace

## Where are the data?
## What if data are moved or renamed?

# store in a variable the path where the Excel file 'Deer.xls' is
wd <- '/home/prea/Dropbox/R/R_2015/slide/src/data/'
# Also store in a variable the file name
datafile <- 'Deer.csv'

# chenge R 'working directory' into the one where the spreadsheet is
setwd(wd)

# load the Excel file
Deer <- read.csv(datafile)

# explore: how big is the dataset?
nrow(Deer)
ncol(Deer)

# how are data organized?
str(Deer)
```

Analisi esplorativa: *do it yourself*! II

```
26
28 # how many observations for each sex?
   table(Deer$Sex)
30
   # better recode Sex...
32 Deer$SEX <- factor(Deer$Sex, levels=c(1,2), labels=c('M', 'F'))

34 # recode as well clas1_4
   Deer$CLASS <- factor(Deer$clas1_4, levels=c(1,2,3,4), labels=c('1', '2', '3', '4'
       ))
36

38 # go on... how many observation for each sex?
   table(Deer$SEX)
40 # observations by year?
   table(Deer$Year)
42 xtabs( ~ Year, data=Deer)
   # observations by farm
44 table(Deer$Farm)
   # by sex and year
46 table(Deer$Sex, Deer$Year)
   xtabs( ~ SEX + Year, data=Deer)
48
   # average LCT?
50 mean(Deer$LCT)
   # NA?
52 mean(Deer$LCT, na.rm=TRUE)
   # mean by sex
```

Analisi esplorativa: *do it yourself*! III

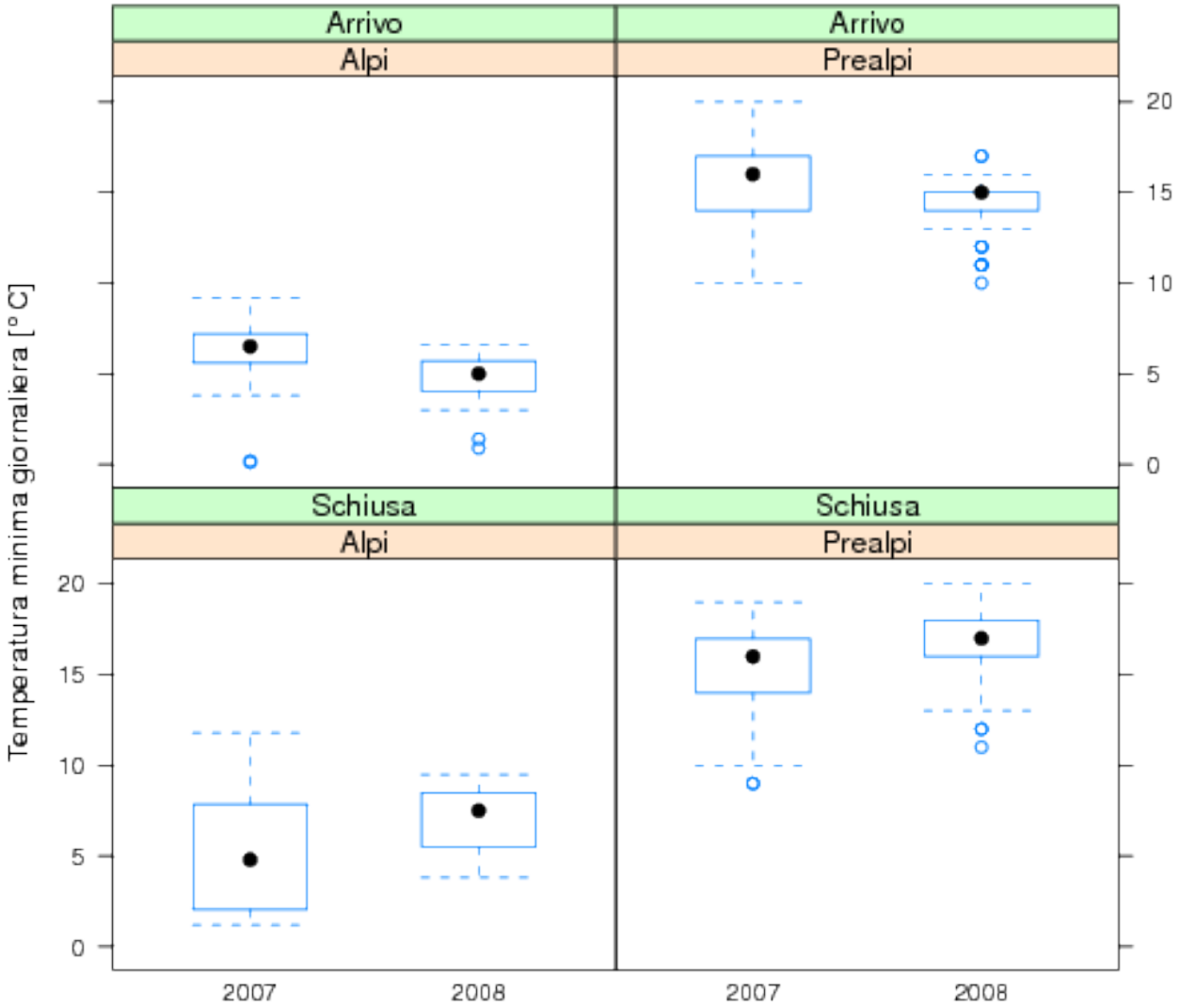
```
54 aggregate(LCT ~ SEX, data=Deer, FUN=mean, na.rm=TRUE)
   # mean by sex and age class
56 aggregate(LCT ~ SEX + clas1_4, data=Deer, FUN=mean, na.rm=TRUE)
```

Bibliografia

- https://cran.r-project.org/other-docs.html
- Venables WN, Smith DM, R Core Team, 2015. An Introduction to R. https://cran.r-project.org/doc/manuals/R-intro.pdf
- Paradis E, 2005. R for beginners. https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf
- Rossiter DG, 2014. Introduction to the R Project for Statistical Computing for use at ITC http://www.css.cornell.edu/faculty/dgr2/pubs/list.html#pubs_m
- Verzano J., 2002. SimpleR. Using R for Introductory statistics. https://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf
-
- Zuur AF, Ieno EN, Meesters EHWG, 2009. A Beginner's Guide to R. UseR! Series, Springer.
- Logan M, 2010. Biostatistical design and analysis using R : a practical guide. Wiley.


Sommario

Cosa c'è dietro a un grafico?



Cosa c'è dietro a un grafico?

Creare un grafico in R significa programmare le "regole" per costruirlo.
Questo (ovviamente) non significa indicare "quali sono le serie di dati", ma *come deve essere "calcolato"* il grafico.
Se un grafico non riesce come vi aspettavate...
probabilmente ha ragione R!

Cosa c'è dietro a un grafico?

Creare un grafico in R significa programmare le "regole" per costruirlo.
Questo (ovviamente) non significa indicare "quali sono le serie di dati", ma *come deve essere "calcolato"* il grafico.
Se un grafico non riesce come vi aspettavate...
probabilmente ha ragione R!

La "grafica" non si inventa!

Cleveland W, 1985. Elements of Graphic Data. Brooks Cole.
Wilkinson L, 1999. The Grammar of Graphics. Springer.

Tre modi di fare grafica

`base` `plot` e famiglia... approccio per "serie di dati"

`lattice` grafici come *formule*

`ggplot2` *Grammar of Graphics*

Un esempio con `plot` I

File: src/example_07c.R

```
1
rm(list=ls())
3
data(iris)
5
attach(iris)
7 # specify X and Y
plot(Petal.Length, Petal.Width)
9
# same, but using the 'formula interface"
11 # note: Y ~ X
plot(Petal.Width ~ Petal.Length)
13
# color by Species
15 plot(Petal.Length, Petal.Width, pch=(21:23)[as.numeric(Species)], cex=1.2,
     xlab="Petal length (cm)", ylab="Petal width (cm)",
17   main="Anderson Iris data",
     col=c("slateblue", "firebrick", "darkolivegreen")[as.numeric(Species)]
19 )
21 # add trendlines
abline(v=mean(Petal.Length), lty=2, col="red")
23 abline(h=mean(Petal.Width), lty=2, col="red")
abline(v=median(Petal.Length), lty=2, col="blue")
25 abline(h=median(Petal.Width), lty=2, col="blue")
```

Un esempio con `plot` II

```
# add a grid
grid()

# add centroids
points(mean(Petal.Length), mean(Petal.Width),
       cex=2, pch=23, col="black", bg="red")
points(median(Petal.Length), median(Petal.Width),
       cex=2, pch=23, col="black", bg="blue")

# add a subtitle
title(sub="Centroids: mean (green) and median (gray)")

# free-format annotations
text(1, 2.4, "Three species of Iris", pos=4, col="navyblue")

# a legend
legend(1, 2.4, levels(Species), pch=21:23, bty="n",
         col=c("slateblue", "firebrick", "darkolivegreen"))

# regression lines
abline(lm(Petal.Width ~ Petal.Length), lty="longdash", col="red")
library(MASS) # for lqs function
abline(lqs(Petal.Width ~ Petal.Length), lty=2, col="blue")
```


Programmazione statistica in R: corso teorico-pratico

Parte 3

Damiano G. Preatoni

Unità di Analisi e Gestione delle Risorse Ambientali – *Guido Tosi Research Group*
Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria

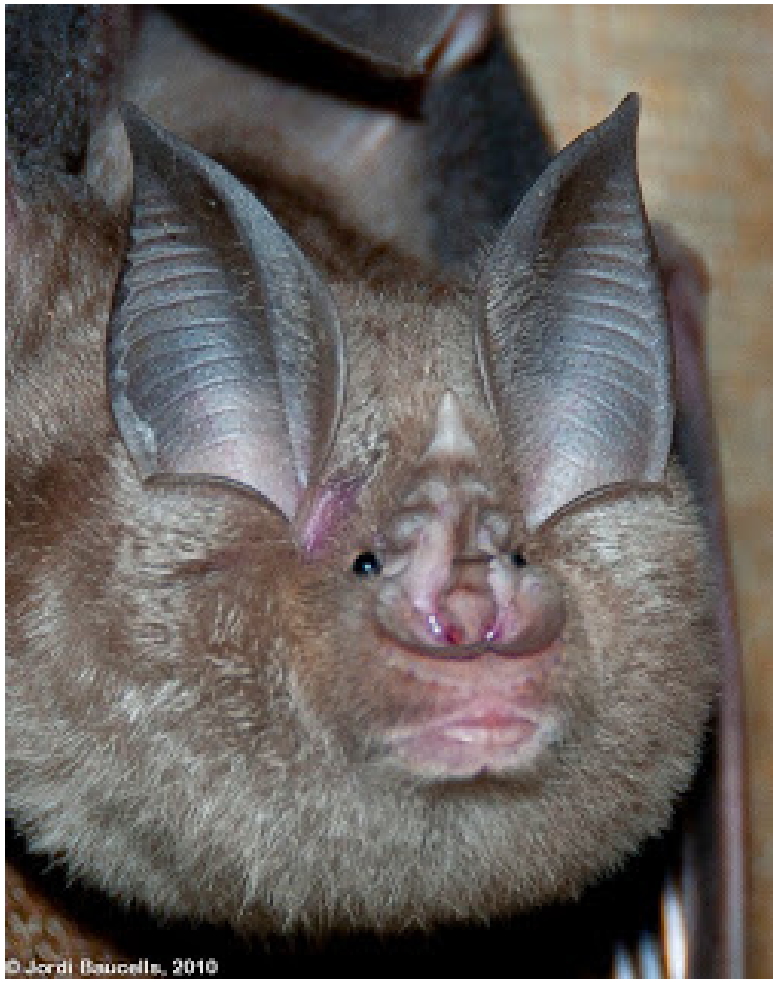
prea@uninsubria.it

7–9/7/2021

Sommario

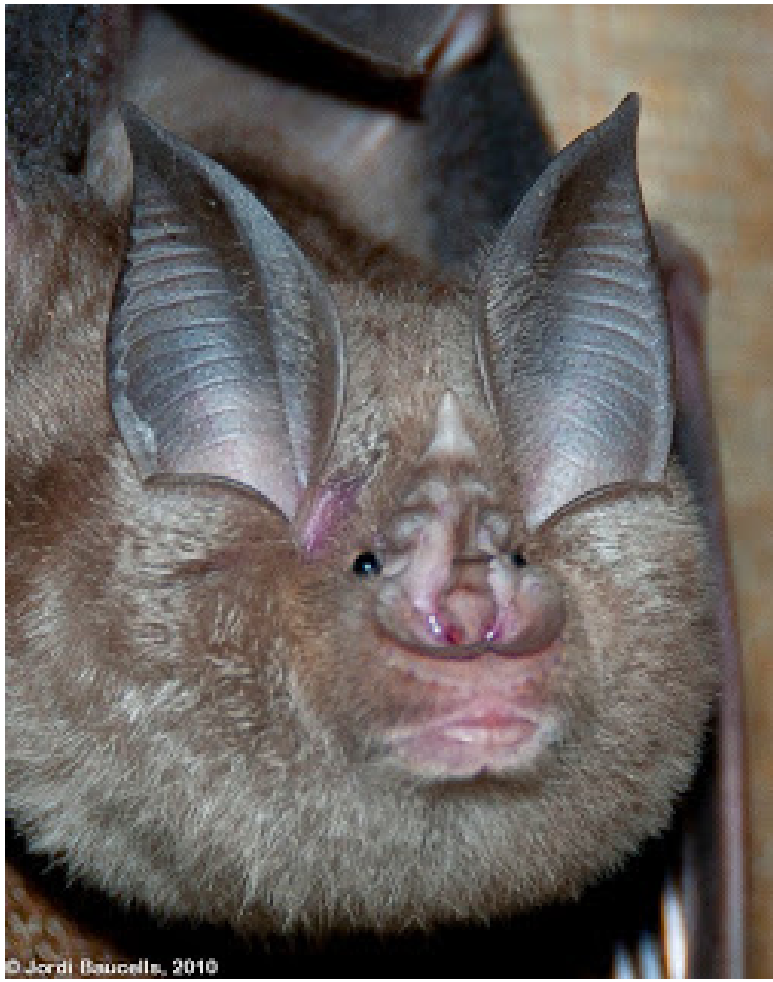
Un caso reale

In molte specie di Mammiferi, il *Body Condition Index* è indice indiretto della *fitness* degli individui.
Esistono differenze nella *fitness* di femmine riproduttive di *Rhinolophus ferrumequinum* in tre differenti siti?



Un caso reale

In molte specie di Mammiferi, il *Body Condition Index* è indice indiretto della *fitness* degli individui.
Esistono differenze nella *fitness* di femmine riproduttive di *Rhinolophus ferrumequinum* in tre differenti siti?

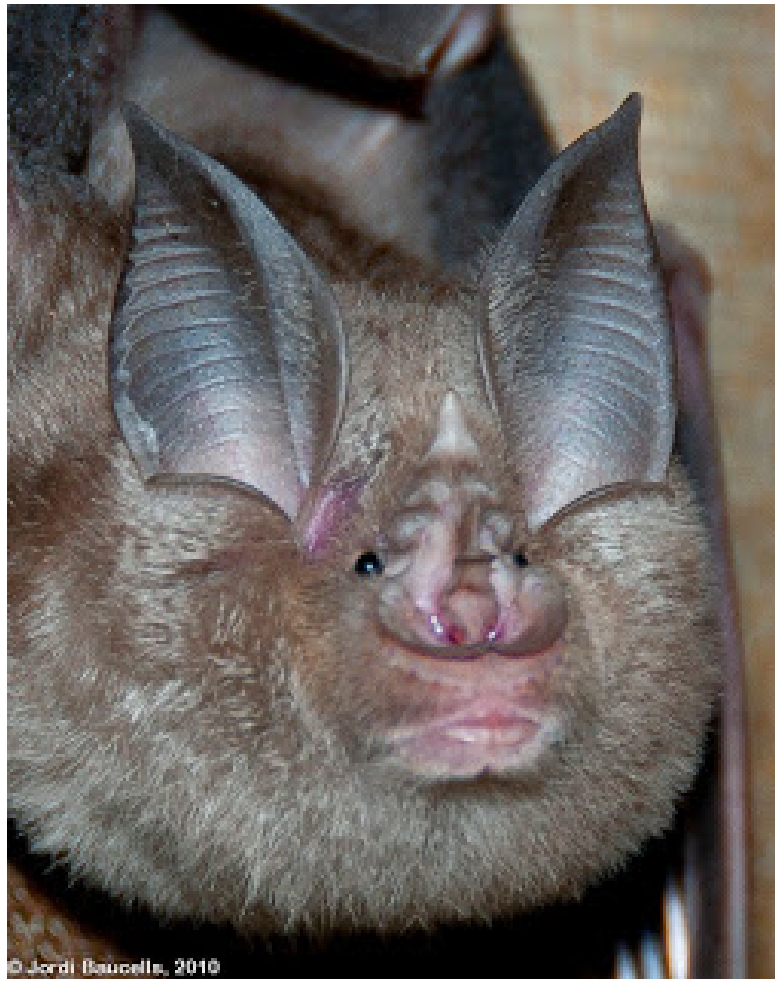


Dati disponibili

- biometrie
- numero di individui presenti per ogni colonia
- quota
- metriche di paesaggio

Un caso reale

In molte specie di Mammiferi, il *Body Condition Index* è indice indiretto della *fitness* degli individui.
Esistono differenze nella *fitness* di femmine riproduttive di *Rhinolophus ferrumequinum* in tre differenti siti?

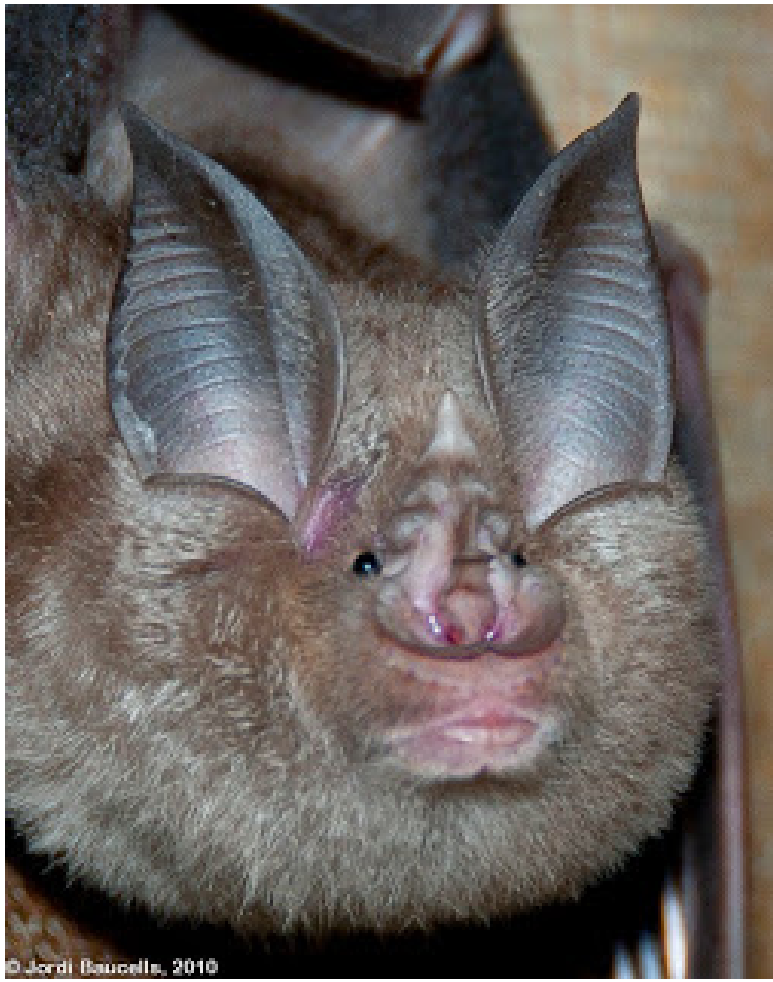


Problemi

- i dati sono relativi a 4 siti
- gli individui sono 40
- quota e metriche di paesaggio sono relative ai 4 siti

Un caso reale

In molte specie di Mammiferi, il *Body Condition Index* è indice indiretto della *fitness* degli individui.
Esistono differenze nella *fitness* di femmine riproduttive di *Rhinolophus ferrumequinum* in tre differenti siti?



Soluzioni

- accorpare (media, sd) le biometrie per sito
- unire i dati biometrici accorpati ai dati di paesaggio

Un caso reale I

File: src/example_08_bats.R

```
## 1. prepare working environment

# clean up the workspace
rm(list=ls())

# we need some libraries:
library(foreign)   # read.csv, read.csv2
require(Hmisc)     # xYplot
require(lattice)   # lattice plots

# specify a 'working directory'
setwd('/data/Didattica/R/R_2013/src/data')

## 2. get data
# site data (direct input)
site.data <- data.frame(localita=c("Cormons","Pandona","S.Cesario","Toggiano"),
                        quota=c(58,140,56,600),
                        consistenza=c(190,360,30,26))
# read in bci data
bci.data <- read.csv2('BCI.csv', as.is=TRUE)
# read in landscape data
land.data <- read.csv('landscape.csv')

```

Un caso reale II

```
26 ## 3. data conditioning

28 # condition some variable in bci.data
   bci.data$localita <- factor(bci.data$localita,
30                             levels=c('JAP', 'COR', 'SCP', 'TOG'),
                               labels=c('Pandona', 'Cormons', 'S.Cesario', 'Toggiano
                                   '))
32 bci.data$sesso <- factor(bci.data$sesso)
   bci.data$eta <- factor(bci.data$eta)
34 # we need only female adults
   bci.data <- bci.data[bci.data$sesso == 'f' & bci.data$eta == 'ad',]
36 # aggregate BCI by site
   bci.avg <- aggregate(bci.data$BCI, by=list(bci.data$localita), FUN="mean")
38 names(bci.avg) <- c('localita','BCI.avg')
   bci.sd <- aggregate(bci.data$BCI, by=list(bci.data$localita), FUN="sd")
40 names(bci.sd) <- c('localita','BCI.sd')
   bci.avg <- merge(bci.avg, bci.sd, by="localita")
42 rm(bci.sd)
   bci.avg <- merge(bci.avg, site.data, by="localita")
44 # now, dataframe bci.avg contains all average stats

46 # condition data in landscpe metrics data
   land.data$localita <- factor(land.data$LID, levels=c(' D:\\w_rhifer\\uso_4 ',' D
       :\\w_rhifer\\uso_6 ',' D:\\w_rhifer\\uso_9 ','D:\\w_rhifer\\uso_11 '),
       labels=c('Pandona', 'Cormons', 'S.Cesario', 'Toggiano'))
48
   # merge bci + landscape data
50 land <- merge(bci.avg, land.data,by="localita")
   land$LID <- NULL # delete LID column
```

Un caso reale III

```
52
54 # 4. Exploratory Data Analysis
56 # BCI vs. elevation
   xYplot(Cbind(BCI.avg,BCI.sd) ~ quota, groups=localita, bci.avg, type=c("p"),
        method="bars", ylim=c(.25,.4), label.curves=TRUE, auto.key=TRUE, ylab="BCI",
        xlab="Quota (m s.l.m.)")
58
   cor.test(bci.avg$BCI.avg, bci.avg$quota
60
62 # BCI vs. colony size
   xYplot(Cbind(BCI.avg,BCI.sd) ~ consistenza, groups=localita, bci.avg, type=c("p")
        , method="bars", ylim=c(.25,.4), label.curves=TRUE, auto.key=TRUE, ylab="BCI
        ",xlab="Consistenza stimata della colonia (individui)")
64
   cor.test(bci.avg$BCI.avg, bci.avg$consistenza)
66
68 # Automate correlation analysis between BCI and Landscape level metrics
   # We need to repeat the same analysis for 70 variables!!
70
   # get landscape variables names
72 land.vars <- names(land.data)[3:length(names(land.data))-1]

74 # define correlation tests 'limits'
   corr.sig <- 0.06  # passes test if less than or equal to
76 corr.val <- 0.3   # passes test if greater then or equal to
```

Un caso reale IV

```
78 land.vars.good <- vector() # to store interesting variable names

80 # open a 'sink file' to write tests results, but close it immediately
   #  to avoid writing garbage into
82 sink(file='lansdcape_correlations.txt', split=TRUE)
   sink()
84
   # begin plotting & making correlations
86 for(v in land.vars) {
     filename <- paste("plots/land_BCI",v,"png",sep=".")
88   cat("plotting",filename,"...")
     x <- land[[v]]
90   c <- cor.test(land$BCI.avg,x)
     # a little data stashing to have a vector of 'interesting' variables
92   if(!is.na(c$estimate)) {
       s <- paste("r=",formatC(c[["estimate"]],3),", p=",formatC(c[["p.value"]],3),
            sep="")
94     sink(file='lansdcape_correlations.txt', split=TRUE, append=TRUE)
       cat("correlation of", v, "with BCI", s, "\n")
96     sink()
       if (abs(c[["estimate"]]) >= corr.val & c[["p.value"]] <= corr.sig) {
98       land.vars.good <- c(land.vars.good, v)
         cat("SIGNIFICANT!")
       }
     } else {
02     s <- "regression not possible"
     }
04   plt <- xyplot(BCI.avg ~ x, land, type=c('p','r'), xlab=v, ylab="BCI", sub=s)
```

Un caso reale V

```
    #direct.label(plt)
06  png(filename=filename)
    print(plt)
08  dev.off()
    cat("\tDone.\n")
10  }
sink()
12 rm(v,x,c,s,plt)

14 ## End of File
```

Qualche commento...

Nello *script* sono stati utilizzati diversi comandi per

- importare dati
- modificare dataframe
- aggregare dati
- automatizzare analisi ripetitive

Import/export

Esistono numerosi *package* per leggere e scrivere praticamente qualsiasi tipo di dato

foreign legge e scrive `csv`, `dbf` e formati di altri software per l'elaborazione dati

gdata fornisce una funzione `read.xls`, ma richiede il linguaggio `perl`[27]

[27] che su UNIX è di serie...

Import/export

Esistono numerosi *package* per leggere e scrivere praticamente qualsiasi tipo di dato

foreign legge e scrive `csv`, `dbf` e formati di altri software per l'elaborazione dati

gdata fornisce una funzione `read.xls`, ma richiede il linguaggio `perl`[27]

Il problema dei fogli elettronici

Il foglio elettronico è *comodo*, ma il formato fisico dei file cambia continuamente. In più, è preferibile utilizzare un *package* che funzioni su qualsiasi piattaforma.
Attualmente, conviene utilizzare o `XLConnect` (che utilizza Java), o `readxl`, che è "puro R".

[27] che su UNIX è di serie...

Manipolazione dataframe

Esistono innumerevoli comandi per "lavorare" con i dataframe

cbind unisce dataframe "accostando" le colonne

rbind unisce dataframe "accostando" le righe

merge unisce due dataframe in base a una colonna chiave presente in entrambi

Manipolazione dataframe

Esistono innumerevoli comandi per "lavorare" con i dataframe

cbind unisce dataframe "accostando" le colonne

rbind unisce dataframe "accostando" le righe

merge unisce due dataframe in base a una colonna chiave presente in entrambi

R ricicla...

`cbind` richiede che i due dataframe abbiano lo stesso numero di righe.
`rbind` richiede che i dataframe abbiano la stessa struttura.
`merge` effettua invece una vera e propria operazione di *giunzione*.

Manipolazione dataframe

Esistono innumerevoli comandi per "lavorare" con i dataframe

cbind unisce dataframe "accostando" le colonne

rbind unisce dataframe "accostando" le righe

merge unisce due dataframe in base a una colonna chiave presente in entrambi

Il *package* `plyr` è in grado di "piegare" un dataframe in ogni modo possibile, in base a regole di accorpamento, ma utiliza una sintassi complessa[28].

[28] che alla fine conviene imparare

Aggregare e tabulare dati

È possibile *applicare* a una variabile di un dataframe una *funzione di aggregazione*[29] e ottenere un nuovo dataframe "aggregato" in base ai valori di una seconda variabile.
Per casi più complessi (es. aggregazione su più variabili) è più conveniente adottare un'altra strategia, di tipo *map-reduce*[30].
Si tratta di:

1. dividere il dataframe in più dataframe indipendenti (es. `split()`)
2. processare ciascun dataframe separatamente (es. `lapply()`)
3. "rimettere insieme i pezzi" (es. `plyr::ldply()` o brutalmente con `do.call(rbind, ...)`)

[29] ad esempio `mean`, `sum`, `sd`...

[30] Una strategia come *map-reduce* rende il programma di fatto *già parallelizzabile!*

Il tipo di dati `factor`

Serve per rapresentare variabili *nominali*
Viene rappresentato *internamente* con numeri interi (“levels”)
Viene rappresentato *esternamente* con stringhe (“labels”)

Aggregare e tabulare dati: `xtabs`

È possibile *crosstabulare*[31] una variabile di un dataframe in base ad altre variabili (che vangono automaticamente convertite in factor.
`xtabs` utilizza un'*interfaccia a formula*, tecnica frequentissima in R.

[31] cioè creare delle tabelle di contingenza

Aggregare e tabulare dati: `xtabs`

È possibile *crosstabulare*[31] una variabile di un dataframe in base ad altre variabili (che vangono automaticamente convertite in factor.
`xtabs` utilizza un'*interfaccia a formula*, tecnica frequentissima in R.

Oggetti "formula"

```
<left side term> ~ <right side term>
```

[31] cioè creare delle tabelle di contingenza

Aggregare e tabulare dati: `xtabs`

È possibile *crosstabulare*[31] una variabile di un dataframe in base ad altre variabili (che vangono automaticamente convertite in factor.
`xtabs` utilizza un'*interfaccia a formula*, tecnica frequentissima in R.

Oggetti "formula": esempi

`data$y ~a + b * data$x`: un'equazione per una regressione
`d$response ~d$dose * d$age + Error(d$subject)`: un modello con errore casuale per un'analisi della varianza

...

[31] cioè creare delle tabelle di contingenza

Aggregare e tabulare dati: `xtabs`

È possibile *crosstabulare*[31] una variabile di un dataframe in base ad altre variabili (che vangono automaticamente convertite in factor.
`xtabs` utilizza un'*interfaccia a formula*, tecnica frequentissima in R.

Oggetti "formula": esempi

`<variabile da tabulare> ~<dimensioni della tabella>: xtabs`
Se non viene specificato nulla nel termine di sinistra, `xtabs` *conta le righe*.
Se viene specificata una variabile numerica, `xtabs` *somma*.

[31] cioè creare delle tabelle di contingenza

Aggregare e tabulare dati: xtabs

È possibile *crosstabulare*[31] una variabile di un dataframe in base ad altre variabili (che vangono automaticamente convertite in factor.
xtabs utilizza un'*interfaccia a formula*, tecnica frequentissima in R.

Oggetti "formula": xtabs

<variabile da tabulare> ~<dimensioni della tabella>

File: src/xtabs.R

```
xtabs( ~ Specie, data)
  Specie
  lepcom  lepvar
  745     432
```

[31] cioè creare delle tabelle di contingenza

Automazione: flusso del programma

Programmare significa innanzitutto demandare alla macchina i compiti ripetitivi.
R offre un linguaggio di programmazione sufficientemente ricco: `?Control`

if(cond) expr test logici

if(cond) cons.expr else alt.expr test logici a due alternative

for(var in seq) expr iterazione per un numero finito di ripetizioni

while(cond) expr iterazione indefinita in base a una condizione

repeat expr iterazione indefinita

ifelse test logico "in line"

switch test logico a più "uscite"

Automazione: flusso del programma

Programmare significa innanzitutto demandare alla macchina i compiti ripetitivi.
R offre un linguaggio di programmazione sufficientemente ricco: `?Control`

if(cond) expr test logici

if(cond) cons.expr else alt.expr test logici a due alternative

for(var in seq) expr iterazione per un numero finito di ripetizioni

while(cond) expr iterazione indefinita in base a una condizione

repeat expr iterazione indefinita

ifelse test logico "in line"

switch test logico a più "uscite"

Dato che R "itera naturalmente", è meglio *evitare* di utilizzare cicli `for`.
In più, un `for` è difficilmente parallelizzabile, mentre `lapply` (e famiglia) lo è (*package* `parallel`)

Automazione: funzioni

R è un *linguaggio funzionale*, cioè si basa sulla definizione di *funzioni*

- funzioni *sensu* $f(x)$
- non necessariamente operanti solo su dati numerici
- *nuovi comandi* a tutti gli effetti
- operano indipendentemente dallo stato del programma

L'approccio funzionale consente di *estendere* le capacità di R, ad esempio allestendo librerie di funzioni personalizzate[32]

[32] . . . che possono anche essere utilizzate per creare nuovi *package*

Sommario

Proviamo da soli...

Creare una directory locale[33] nella quale copiare di volta in volta i *dataset* per ciascun esercizio.
Un *dataset* comprende uno *script* R e i relativi file di dati.
Ciascuno *script* è commentato, in modo da dare una spiegazione delle tecniche e dei comandi utilizzati.
In ogni *script* sono citati i riferimenti alle pagine di manuale relative ai diversi comandi utilizzati.
In caso di dubbio su qualunque comando ***usare l'help***

R help system

`?<nome comando>` mostra la pagina di manuale
`??<keyword>` ricerca
`RSiteSearch<keyword>` cerca sul sito R
`http://rseek.org` "specializzazione" di Google per ricerche su R

[33] ad esempio `C:\R`

Attenzione ai metacommenti!

`@TODO`

In alcuni punti degli *script* alcuni comandi dovranno necessariamente essere adattati, ad esempio nel caso dati e *script* siano in una cartella differente. Tali comandi sono preceduti da un commento che inizia con il metacommento `@TODO`.

`@Q&A`

In alcuni punti degli *script* l'uso di alcuni comandi particolari o di alcune tecniche *dovrebbe* far venire alcuni dubbi o quantomeno suscitare un minimo id curiosità. Tali comandi sono preceduti da un commento che inizia con il metacommento `@Q&A`.

`Riferimenti`

I commenti che iniziano con il testo `Riferimenti` rimandano alle pagine di manuale.

Lepri e gradienti altitudinali

Script file: `src\example_09_hares2.R`

Dataset: `src\data\hares.csv`

Descrizione: osservazioni di lepre alpina e lepre comune (2000–2008) in 5 Comprensori Alpini di Caccia della provincia di Sondrio.

Obiettivo

In Valtellina, *Lepus timidus* e *L. europaeus* occupano orizzonti altitudinali diversi?

Lepri e gradienti altitudinali I

File: src/example_09_hares2.R

```
# Gli oggetti di una sessione di lavoro R rimangono in memoria (i.e. nel
    workspace).
# Prima di iniziare una nuova sessione di analisi, e' opportuno "svuotare"
# lo spazio di lavoro
# Riferimenti: ?rm
rm(list=ls())

# Risulta comodo immagazzinare dati, script e risultati nella stessa directory.
#  Conviene utilizzare una directory separata per ciascuna analisi.
#  Spesso - pero' - succede di dover spostare i vari file, e al contempo non
# desiderabile dover rivedere l'intero script a causa del cambiamento del
# nome di un file o dello spostamento di dati e script in una differente
# directory.
#  Una tecnica elementare consiste nel dichiarare in anticipo gli 'elementi
# variabili', in pratica definendoli come costanti. In caso di variazioni
# sara' sufficiente sistemare i nuovi valori in un punto solo dello script.
#   Definiamo come costante la directory nella quale stanno i dati
##@TODO modificare rispetto alla propria directory di lavoro
workdir <- "/data/Didattica/R/R_2013/src/"
# definisce la 'cartella di lavoro' di R
setwd(workdir)
##@Q&A: e se non uso setwd()? Dove 'lavora' (ad esempio salva i risultati) R?


##@TODO modificare togliendo in carattere di commento a uno (e uno solo!)
```

Lepri e gradienti altitudinali II

```
##@TODO dei due blocchi di codice seguenti

## legge i dati direttamente da un foglio Excel (occorre avere installato
## il package XLConnect
#require(XLConnect)
#data <- readWorksheetFromFile(paste("data","hares.xls", sep="/"), sheet="totale
    ")

## legge i dati da un file di testo CSV (Comma Separated Values, occorre avere
## il package foreign)
library(foreign)
data <- read.csv(paste("data","hares.csv", sep="/"))

##@Q&A: che differenza c'e' tra i comandi require() e library()?

##@Q&A: cha cosa fa il comando paste()? Provare a copiare e incollare nella linea
    di comando solo la funzione paste("data","SOVRAPP_LC-VAR.def.csv", sep="/")

##@Q&A: che razza di oggetto e' data, dal momento che non  stato specificato
    nulla di particolare?
#Riferimenti: ?data.frame

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# rapido esame della struttura dei dati
head(data)
str(data)

##@Q&A: quante osservazioni ho per ciascuna specie?

```

Lepri e gradienti altitudinali III

```
##@Q&A: quante osservazioni ho per anno?

##@Q&A: e combinando specie e anno?
#Riferimenti: ?xtabs, ?formula

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Condizionamento dei dati
##@Q&A: le variabili sono del 'tipo' corretto?
#Specie
#Comprensorio
#Quota
#Anno
#Riferimenti: ?class, ?is, ?str

# Sarebbe meglio se Specie e Comprensorio fossero variabili di tipo 'factor'
    anzich di tipo carattere

##@Q&A: E' possibile fare di meglio, rispetto a usare delle sigle incomprensibili
    ?
#  Una variabile di tipo factor possiede dei _livelli_, codificati con numeri
    interi
# e a ciascun livello pu essere associata un'_etichetta_ di comodo, che viene
# utilizzata ad esempio nei titoli delle tabelle e nei grafici
#Riferimenti: ?factor
data$Specie <- factor(data$Specie, levels=c('lepcom','lepvar'), labels=c('L.
    europaeus','L. timidus'))

data$Comprensorio <- factor(data$Comprensorio)
```

Lepri e gradienti altitudinali IV

```
#  La variabile Comprensorio ha un numero di livelli non noto. Sarebbe meglio
# esaminare i valori esistenti e quindi ricodificarli con qualcosa di piu'
    comprensibile
#  I comandi utilizzati sono presentati esclusivamente a titolo di esempio.
# In questo caso, e' comune in uno script R prima del blocco di codice commentato
     che
# funge da esempio, ma non e' essenziale ai fini della procedura di elaborazione,
# aggiungere un commento che tradizionalmente indica che il codice seguente e'
# appunto un esempio e non  importante o necessario eseguirlo.

## Not Run:
# summary(data$Comprensorio)
#> AV  CH  MO  SO  TI
#>157  94 218 314 394

data$Comprensorio <- factor(data$Comprensorio, levels=c("CH", "MO", "SO", "TI", "
    AV"), labels=c("Val Chiavenna", "Morbegno", "Sondrio", "Tirano", "Alta Valle
    "))

## Not Run:
# summary(data$Comprensorio)

##@Q&A Il 'cambio di etichette' dovrebbe dare un risultato migliore ad esempio
##@Q&A nel caso delle tabulazioni...

## Not Run:
# xtabs( ~ Specie + Comprensorio, data)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Lepri e gradienti altitudinali V

```
# Analisi esplorativa: le quote medie almeno 'sembrano' diverse o no?
# si potrebbe pensare che sia necessario 'aggregare' le osservazioni in modo
# da avere un 'totale' per ciascuna quota. A pensarci bene, e' fondamentalmente
# tempo perso: e' molto piu' pratico lavorare sui dati grezzi!

# Eccoun esempio di inutile aggregazione
## Not Run:
## (data.freq <- data.frame(xtabs(~Quota+Specie, data)))

# il sistema piu' rapido per visualizzare 'le medie' di piu' 'gruppi' e' il box-
    and-whisker plot
boxplot(Quota ~ Specie,data)

# Il package lattice e' in grado di produrre grafici piu' complessi e in modo piu
    ' comodo.
#   Lattice usa un'interfaccia a formula!
#   In lattice, il comando equivalente a boxplot e' bwplot
bwplot(Quota ~ Specie,data)

##@Q&A,@TODO perche' il comando bwlpot non funziona? E' presente ed e' stato
    caricato il package lattice?

# ggplot2 e' ancora meglio, da un punto di vista estetico
#   GGplot2 si basa sulla metodologia creata da Leland Wilkinson nota come "
    grammar of Graphics"
ggplot(data, aes(x=Specie, y=Quota)) + geom_boxplot()

##@Q&A,@TODO E' stata caricata la libreria ggplot2?
```

Lepri e gradienti altitudinali VI

```
# Un altro modo di visualizzare i dati, utilizzando la grafica a pannelli di
    lattice
histogram(~Quota | Specie, data=data, type="count", layout=c(1,2))
histogram(~Quota | Comprensorio + Specie, data=data, type="count")
##@TODO provare ad invertire l'ordine dei fattori nel comando precedente
histogram(~Quota | Anno + Specie, data=data, type="count")

# in ggplot2
ggplot(data, aes(x=Quota)) + geom_histogram() + facet_grid(Specie ~ .)
ggplot(data, aes(x=Quota)) + geom_histogram() + facet_grid(Specie ~ Comprensorio)

# Un altro modo di visualizzare i dati, utilizzando uno specifico grafico
# ('violin plot')
require(vioplot)
vioplot(data[data$Specie == "L. europaeus","Quota"], ##@Q&A: che significano quei
     nomi di variabile racchiusi tra quadre?
        data[data$Specie == "L. timidus","Quota"],
        names= c('L. europaeus','L. timidus'),        # specifica i nomi degli
            assi
        col='lightgray',                              # (quasi) nessuna rivista
            pubblica grafici a colori!
        ylim=c(0,3000)                                # forziamo la scala dell'
            asse Y su un intervallo 'comodo'
       )


#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#
# Per avere un'evidenza statistica delle differenze, occorre eseguire un test
# sulla differenza tra le medie.
```

Lepri e gradienti altitudinali VII

```
# Ad esempio, il t di Student, che pero' richiede che le variabili oggetto del
# test abbiano distribuzione normale.
# Prima ancora – quindi – occorrer effettuare un test per la normalita' (ad
# esempio il test di Shapiro–Wilks).
#Riferimenti: ?shapiro.test
shapiro.test(data[data$Specie == "L. europaeus","Quota"])
shapiro.test(data[data$Specie == "L. timidus","Quota"])
##@Q&A: sono distribuite normalmente? Indizio: cosa e' possibile capire dall'
##@Q&A: esempio nella pagina di manuale di shapiro.test?

# Un'alternativa (che non e' un test statistico) in forma grafica: il quantile–
# quantile plot mette in grafico i valori di una data variabile rispetto ai
# valori che quella variabile avrebbe _se fosse_ normale.
# Riferimenti: ?qqnorm, ?qqplot, ?qline
qqnorm(data[data$Specie == "L. europaeus","Quota"])
qqline(data[data$Specie == "L. europaeus","Quota"],col='red')
qqnorm(data[data$Specie == "L. timidus","Quota"])
qqline(data[data$Specie == "L. timidus","Quota"],col='red')

##@Q&A: c'e' per caso qualche ripetizione?

# Creare del codice contenente ripetizioni e' fondamentalmente una tecnica
    suicida,
# in particolare nel caso in cui si debbano fare in seguito delle modifiche.
# La soluzione intelligente e' quella di creare una _funzione_, per poi
    utilizzarla
# tutte le volte che si vuole.

myqqplot <- function(x) {
```

Lepri e gradienti altitudinali VIII

```
  qqnorm(x)
78  qqline(x,col='red')
}
80
myqqplot(data[data$Specie == "L. europaeus","Quota"])
82 myqqplot(data[data$Specie == "L. timidus","Quota"])

84 # c'e' ancora qualche ripetizione...
lapply(split(data$Quota, data$Specie) , myqqplot)
86
##Q&A: E' possibile verificare la differenza tra quote con un t-test?
88 # Riferimenti: ?t.test

90 ## Not Run:
##t.test(Quota ~ Specie,data)
```

Chirotteri e *optimal foraging*

Script file: `src\example_10_distance.R`

Dataset: `src\data\dist_tab.dbf`

Descrizione: distanze percorse tra sito di ricovero diurno (*roost*) e aree di foraggiamento da femmine di *Myotis emarginatus* in differente stato riproduttivo.

Obiettivo

Le femmine che si sono riprodotte dovrebbero avere dei costi energetici maggiori, e quindi sfrutterebbero aree di foraggiamento più vicine al *roost*.

Chirotteri e optimal foraging I

File: src/example_10_distance.R

```
# 'pulizia' del workspace
rm(list = ls())

# I dati sono in formato dBASE IV (.dbf). Occorre il package foreign per poterli
# caricare
require(foreign)

#require(graphics)
#require(lattice)
#require(nlme)

# definizione della directory di lavoro
workdir <- "/data/Didattica/R/R_2013/src/"

# nome del file di dati
datafile <- "data/dist_tab.dbf"

# attiva la directory di lavoro, carica i dati
setwd(workdir)
dist_tab <- read.dbf(datafile)

##@TODO: il condizionamento dei dati    tutto da rifare!
# Data dictionary per il file dist_tab.dbf
# SRIP: stato riproduttivo: 0 -> non riproduttiva; 1 -> riproduttiva
# Riferimenti: ?factor
```

Chirotteri e optimal foraging II

```
## Not Run:
## dist_tab$SRIP <- factor(dist_tab$SRIP, levels=c(0,1), labels=c('non
    riproduttiva','riproduttiva'))
#    pi  opportuno dare alle variabili dei nomi pi  comprensibili:
# Riproduzione, Mese, Individuo, Distanza
# Riferimenti: ?names
## Not Run:
## names(dist_tab) <- c('Riproduzione', 'Mese', 'Individuo', 'Distanza')

##@TODO: esplorazione dei dati: quali sono i fattori che possono influenzare la
##@TODO: distanza tra roost e aree di foraggiamento?

##@Q&A: L'analisi della varianza richiede che la variabile responso sia
##@Q&A: distribuita normalmente...
## Not Run:
## shapiro.test(dist_tab$Distanza)

#############################################################################
## Analisi della varianza
# modello saturo
aov.0 <- aov(Distanza ~ Riproduzione * Mese, data=dist_tab)
summary(aov.0)
##@Q&A:    possibile escludere il fattore 'Mese', per aumentare la sensibilit
##@Q&A: del test su 'Riproduzione'?
## Not Run:
## aov.1 <- aov(Distanza ~ Riproduzione, data=dist_tab)
## summary(aov.1)
##@Q&A: Riproduzione ha un effetto significativo?
##@Q&A: Se s  , occorrerebbe effettuare un test post-hoc...
```

Chirotteri e optimal foraging III

```
# Riferimenti: TukeyHSD
## Not Run:
## TukeyHSD(aov.1)

#########################################################################################
## Ma quali sono le distanze medie?
# Occorre calcolare la media per i due gruppi (Riproduttive e non)
# Due possibili soluzioni:
# 1) metodo a forza bruta: estrarre i dati dal dataframe e calcolare media e
     deviazione standard
dist_tab.R <- dist_tab[dist_tab$Riproduzione == 'riproduttiva',]
mean(dist_tab$Distanza)
sd(dist_tab$Distanza)

dist_tab.R <- dist_tab[dist_tab$Riproduzione == 'non riproduttiva',]
mean(dist_tab$Distanza)
sd(dist_tab$Distanza)

# 2) approccio un po' pi  intelligente, faciamo gestire a R le 'ripetizioni'
# Riferimenti: ?by
by(dist_tab$Distanza, dist_tab$Riproduzione, summary)

## End of File ##
```

Bilancio energetico in *Emys orbicularis* e *Trachemys scripta*

Script file: `example_11_terrapin.R`

Dataset: `terrapin_vs_slider.txt`

Descrizione: esperimenti di termoregolazione controllata in *Emys orbicularis* e *Trachemys scripta*.

Obiettivo

Esistono delle differenze nell'efficienza della termoregolazione che possono dare un vantaggio alla specie alloctona di tartaruga *Trachemys scripta* nella competizione con la testuggine palustre europea *Emys orbicularis*?

Bilancio energetico in testuggini autoctone e alloctone I

File: src/example_11_terrapin.R

```
# 'pulizia' workspace
rm(list=ls())
##@TODO modificare rispetto alla propria directory di lavoro
workdir <- "/data/Didattica/R/R_2013/src/"
setwd(workdir)

# caricamento dati
data <- read.table('data/terrapin_vs_slider.txt', header=TRUE)


##############################################################################
## Nota
## I comandi utilizzati per le varie fasi dell'analisi sono il pi  delle volte
## commentati (Not Run:). Confrontare ciascun comando con le rispettive pagine
## di manuale, e, una volta compreso il funzionamento del comando, togliere i
## caratteri di commento ed eseguire il comando.
##    anche possibile trovare delle soluzioni alternative...


##Q&A: Le misure di temperatura sono state effettuate ogni 15'.
##Q&A: Che cosa c'  che non va nella variabile data$tempo e com'  possibile
     sistemarlo?
# Soluzione: tempo  stato codificato come numero sequenziale, che si riferisce
# a intervalli successivi di 15'.  sufficiente moltiplicare per 15!
```

Bilancio energetico in testuggini autoctone e alloctone II

```
25 timespan <- 15 # minutes
   data$tempo <- data$tempo * timespan
27

29 ##Q&A: Come varia la temperatura in funzione del tempo?
   ##Q&A:    possibile visualizzare le curve di temperatura separatemante per le due
         specie?
31 # Soluzione: quando si tratta di 'ripetere lo stesso grafico' in funzione di una
   # variabile fattoriale (in questo caso data$specie),    il caso di usare lattice.
33 require(lattice)
   xyplot(temperatura~tempo, group=specie, type=c('r','p'), data, auto.key=TRUE)
35 bwplot(temperatura~tempo| specie, data, horizontal=FALSE)

37 # Soluzione (2): utilizzare un conditioning plot, facendo passare una polinomiale
   #  per i punti sperimentali
39 # Riferimenti: ?coplot, ?panel.smooth
   ## Not Run:
41 ## coplot(temperatura ~ tempo | specie, data, panel=panel.smooth)
   ## coplot(temperatura ~ tempo | specie * sesso, data, panel=panel.smooth)
43
   # Soluzione (3): panel plot in ggplot2, facendo passare una polinomiale
45 #  per i punti sperimentali
   # Riferimenti: ?ggplot, ?stat_smooth
47 require(ggplot2)
   require(scales)
49 plt <- ggplot(data, aes(x=tempo, y=temperatura, color=specie)) + geom_point()
```

Bilancio energetico in testuggini autoctone e alloctone III

```
plt + facet_grid(specie ~ .)
plt + facet_grid(. ~ specie)
plt + facet_grid(specie ~ sesso)
plt + facet_grid(specie ~ sesso) + stat_smooth(method='lm')
plt + facet_grid(specie ~ sesso) + stat_smooth(method='loess')

##Q&A: Qual    il modello pi  adatto per rappresentare la variazione di
    temperatura?
##Q&A: (In altri termini:    sufficiente una retta?)
# Riferimenti: ?lm (regressione lineare), ?nls (regressione non lineare)

##Q&A: Le analisi vanno ripetute _due_ volta, una per ciascuna specie.
##Q&A:     possibile identificare una soluzione al problema che utilizzi delle
##Q&A: funzioni?


################################################################################
## Soluzione 1 - regressione lineare
# non viene utilizzata una funzione, ma un approccio _iterativo_ mediante una
# iterazione finita (comando 'for').
# Riferimenti: ?Control

# Il comando for deve avere una variabile _iterabile_ (un vettore, una lista, ecc
    .)
# in base alla quale ripetere di volta in volta.
# Creiamo un elenco delle specie.
# Riferimenti: ?unique, ?as.character
```

Bilancio energetico in testuggini autoctone e alloctone IV

```
75 lista.specie <- as.character(unique(data$specie))

77 # Se eseguiamo le analisi in modo iterativo, non    possibile accedere
   # ai risultati se non quando il ciclo for    terminato.
79 #  Occorre quindi creare un 'contenitore' che riceva i dati mentre il ciclo    in
   # funzione, in modo da poterli consultare in seguito.
81 #  Creiamo una lista, vuota (per ora!)
   # Riferimenti: ?list
83 results <- list()

85 # ciclo for sugli elementi di lista.specie per il calcolo della regressione
       lineare
   for (s in lista.specie) {                    # ripetere per ciascuna specie
87   d <- subset(data,subset=specie==s)         # estrae i dati relativi alla specie
         's' in un dataframe
     results[[s]] <- lm(temperatura ~ tempo, d) # esegue la regressione lineare, e
         immagazzina il risultato
89 }

91 # un p   meglio, usando lapply, senza 'for'
   splist <- split(data, data$specie)
93 results2 <- lapply(splist, function(x) lm(x$temperatura ~ x$tempo))

95 ##Q&A: Che cosa contengono adesso gli oggetti results e results2?
   # Riferimenti: ?str, ?names
97 # Nota: per accedere a un elemento di una lista, si utilizza la notazione con
   #  le doppie parentesi quadre (vedi riga 89 sopra)
```

Bilancio energetico in testuggini autoctone e alloctone V

```
99
##Q&A: Come si fa a vedere i risultati delle regressioni?
01 # Riferimenti: ?summary, ?plot

03 ## Not Run:
   ## visualizza i risultati delle regressioni
05 # summary(results[['trachemys']])
   # summary(results[['emys']])
07
   ## Not Run:
09 ## visualizza i risultati delle regressioni
   # plot(results[['trachemys']])
11 # plot(results[['emys']])

13 ##########################################################################
   ## Soluzione 2 - regressione non lineare
15 # Modello: esponenziale negativa
   #  In questo caso, all'interno del ciclo sono stati inseriti comandi per la
17 # visualizzazione immediata dei risultati.
   results.nls <- list()
19 for (s in lista.specie) {
     d <- subset(data,subset=specie==s)
21   # regressione non lineare (notare i 'guess values' per i parametri del modello)
     results.nls[[s]] <- nls(temperatura ~ Tmin + Tdiff * (1 - exp(-(k/100)*tempo)),
           d, start=list(Tmin=19,Tdiff=5,k=1))
```

Bilancio energetico in testuggini autoctone e alloctone VI

```
23    cat("
          ================================================================
          n","Non linear regression for id = ", s, "\n\n")
      print(summary(results.nls[[s]]))
25    cat("\n\n")
  }
27
  ## stima dei parametri: viene eseguita una regressione non lineare per ciascun
        animale
29 ## i coefficienti e i valori dei parametri vengono immagazzinati in un dataframe
  ## creato in precedenza
31
  # struttura per immagazzinare i risultati
33 animali <- unique(data[,c('id','specie', 'sesso')])
  animali$AIC <- NA      # Akaike Information Criterion
35 animali$Tmin <- NA  # parametro: T iniziale
  animali$Tmax <- NA  # parametro: delta T
37 animali$k <- NA      # parametro: tasso di riscaldamento

39 # calcolo iterativo delle regressioni
  for (b in animali$id) {
41   d <- subset(data,subset=id==b)
     r <- nls(temperatura ~ Tmin + Tdiff * (1 - exp(-(k/100)*tempo)), d, start=list(
          Tmin=19,Tdiff=5,k=1))
43   cat("
          ================================================================
          n","Non linear regression for id = ", b, "\n\n")
```

Bilancio energetico in testuggini autoctone e alloctone VII

```
    print(summary(r))
    cat("\n\n")
    animali[animali$id == b,]$AIC <- AIC(r)
    animali[animali$id == b,]$Tmin <- coef(r)[['Tmin']]
    animali[animali$id == b,]$Tmax <- coef(r)[['Tdiff']] + coef(r)[['Tmin']]
    animali[animali$id == b,]$k <- coef(r)[['k']]
}

# analisi della varianza: k    differente tra specie?
cat("
    ================================================================================
    n","ANOVA for k (negative exponential model)\n\n")
aov.k <- aov(k ~ specie * sesso, animali, family='gaussian')
summary(aov.k)
TukeyHSD(aov.k)
bwplot(k ~ specie , animali)

# analisi della varianza: Tmax    differente tra specie?
cat("
    ================================================================================
    n","ANOVA for k (negative exponential model)\n\n")
aov.Tmax <- aov(Tmax ~ specie * sesso, animali)
summary(aov.Tmax)
TukeyHSD(aov.Tmax)
bwplot(Tmax ~ sesso | specie, animali)
## End of File ##
```

Predatori di semi, *pulsed resource systems* e cambiamenti climatici

Script file: `example_12_cones.R`

Dataset: `Pigne.xlsx`, `Pigne2.xls`

Descrizione: produttività di diverse specie di conifere in diverse aree di studio (Valle d'Aosta e Valtellina).

Obiettivo

Possono verificarsi degli anni di "sovrapproduzione" (*mast* o *pasciona*) di semi: è un meccanismo evolutivo (*predator satiation hypothesis*), ma è possibile che gli anni di "pasciona" dipendano anche da fattori climatici: questo potrebbe essere vero se in aree diverse e specie vegetali diverse gli anni in cui avviene sovrapproduzione sono sincroni.

Predatori di semi, *pulsed resource systems* e cambiamenti climatici I

File: src/example_12_cones.R

```
1  ##########################################################
   ### Testing conifer cones value acrosss areas and years ###
3  ##########################################################
   #Q: cones production of different conifers species has been registered in several
         sites and in consecutive years. Cones production change across years... why
        ?
5  #1. Seed predator or 2.Climate change? If climate change influence cones
        production is there any correlation between species, years and different
        areas?
   # 'clean' workspace
7  rm(list=ls())

9  ########WORKING ENVIRONMENT SETTINGS###########
   ##@TODO change according to your workdir
11 workdir <- "/data/Didattica/R/R_2014/src/data"
   setwd(workdir)
13
   ##ATTENZIONE: i pacchetti che permettono di leggere e gestire i file xls/xlsx si
        possono appoggiare ad altri programmi che devono essere presenti sulla
        vostra macchina. la funziona read.xls del pacchetto gdata utilizza il
        programma perl. Se utilizzate SO Windows dovete specificare al comando dove
        risiede Perl. Se il vostro SO ? UNIX non ? necessario
15 #perl="C:/Perl64/bin/perl.exe"##it is necessary to read xls files, at least on
        Windows...
```

Predatori di semi, *pulsed resource systems* e cambiamenti climatici II

```
17
#######PACKAGES#########
19 library(gdata)##read xls
library(ggplot2)## graphics package
21 library(reshape2)## modify dataset
#library(XLConnect)## read and write xls and xlsx (it needs java)
23
#########################################################
25 #######LOADING DATA#####################################
#########################################################
27
#####CONE PER PLOT#######################################
29 xlsxfile <- 'Pigne.xlsx'
#vegx <- read.xls(xlsxfile, sheet=1, stringsAsFactors=FALSE, perl=perl)## su
    windows
31 vegx <- read.xls(paste(workdir,'data',xlsxfile, sep='/'), sheet=1,
    stringsAsFactors=FALSE)

33 ### have a look of data
str(vegx)
35 head(vegx)

37 ########## FIXING DATA FOR ANALYSIS ################
vegx[vegx$tree.species%in% 'P.abies',]$tree.species <- 'P. abies'
39 vegx[vegx$Area%in% 'CEDRASCO',]$Area <- 'CED'
```

Predatori di semi, *pulsed resource systems* e cambiamenti climatici III

```
vegx$cones <- as.integer(vegx$cones)#number of cones is a character variable, but
    we need it as numeric
vegx <- vegx[!is.na(vegx$cones),]#dropping cones NaN values,
##ATTENTION: be careful at the differences between 0 (zero) and NA!

#########################################################
####folding data to create proper dataframe##############
#########################################################

## we need to have mean, standard deviation and CV of number of cones
vegmean <- aggregate(cones~Area+year+tree.species,vegx,FUN=mean)
names(vegmean) <- c("Area","year","tree.species","mean")
vegsd <- aggregate(cones~Area+year+tree.species,vegx,FUN=sd)
names(vegsd) <- c("Area","year","tree.species","sd")
vegtot <- cbind(vegmean, vegsd$sd)
names(vegtot) <- c("Area","year","tree.species","mean",'sd')
vegtot$cv <- vegtot[, "mean"] / vegtot[, "sd"]

#########################################
##can we optimize the code above?########
############## TRY ######################
#vegtot <- setNames(aggregate(cones~Area+year+tree.species,vegx,FUN= function(x)
    c(mean(x), sd(x))), c("Area","year","tree.species","mean",'sd'))
#names(vegtot) <- c("Area","year","tree.species","mean",'sd')

## as usual while your are running your experiments you get new data and most of
    the time format is different...
```

Predatori di semi, *pulsed resource systems* e cambiamenti climatici IV

```
65 ##########################################
   #### load other data  ##############
   ##########################################
67 xlsvda <- 'Pigne2.xls'
   ###read in from the same file two different sheets ###
69 #rhemes <- read.xls(paste(workdir,'data',xlsvda, sep='/'), sheet='Rhemes',
       stringsAsFactors=FALSE, perl=perl) ##windows
   rhemes <- read.xls(paste(workdir,'data',xlsvda, sep='/'), sheet='Rhemes',
       stringsAsFactors=FALSE)
71 #cogne <- read.xls(paste(workdir,'data',xlsvda, sep='/'), sheet='Cogne',
       stringsAsFactors=FALSE, perl=perl) ##windows
   cogne <- read.xls(paste(workdir,'data',xlsvda, sep='/'), sheet='Cogne',
       stringsAsFactors=FALSE)
73
   ##have a look of the new dataset...
75 str(rhemes)
   str(cogne)
77
   ##obviously format is different...
79 ##FIX the new dataset according to previous dataframe
   vdarh <- melt(rhemes, id=c("Area", "Area.code","ID",'Trap','Tree.species'),
       variable.name = "year")## metl dataset according to previous dataframe
81 vdarh$year <- substring(as.character(vdarh$year),2,5)##something strange with
       year!

83 vdaco <- melt(cogne, id=c("Area", "Area.code","ID",'Trap','Tree.species'),
       variable.name = "year")## melt dataset according to previous dataframe
```

Predatori di semi, *pulsed resource systems* e cambiamenti climatici V

```
vdaco$year <- substring(as.character(vdaco$year),2,5)##something strange with
    year!
85
vda <- rbind(vdarh,vdaco)##join of the two dataframe
87 names(vda) <- c('Area', "Area.code","ID",'Trap','tree.species', 'year','cones')
vdamean <- aggregate(cones~Area+year+tree.species,vda,FUN=mean)
89 names(vdamean) <- c("Area","year","tree.species","mean")
vdasd <- aggregate(cones~Area+year+tree.species,vda,FUN=sd)
91 names(vdasd) <- c("Area","year","tree.species","sd")
vdatot <- cbind(vdamean, vdasd$sd)
93 names(vdatot) <- c("Area","year","tree.species","mean",'sd')
vdatot$cv <- vdatot[, "mean"] / vdatot[, "sd"]
95
###what if factor levels are different??###
97 vdatot[vdatot$tree.species%in%'abete',]$tree.species <- 'P. abies'
vdatot[vdatot$tree.species%in%'Abete',]$tree.species <- 'P. abies'
99 vdatot[vdatot$tree.species%in%'larice',]$tree.species <- 'Larix'
vdatot[vdatot$tree.species%in%'Larice',]$tree.species <- 'Larix'
01 vdatot$year <- as.factor(vdatot$year)

03 ###Finally the complete dataframe!###
vegetation <- rbind(vegtot,vdatot)
05
## Q: Is any correlation between cones production and area+species####
07 ##prepare data for correlation analysis
vegetation$tree.species <- as.factor(vegetation$tree.species)
09
```

Predatori di semi, *pulsed resource systems* e cambiamenti climatici VI

```
##dcast stretch dataframe
zz <- dcast(vegetation, year ~ Area+tree.species, value.var='mean',fun.aggregate=
    mean)
##have a look of the matrix
str(zz)
head(zz)
##correlation test
tt <- cor(zz[,2:19], use="pairwise.complete.obs")

#####################################
###write matrix to file###########
#####################################
##package XLConnect it is useful not just to read but also to write matrix
##where does it write it?
writeWorksheetToFile('cor_cones_speciesXarea.xls', tt, 'spec_vs_area',
    styleAction = XLC$STYLE_ACTION.XLCONNECT)

writeWorksheetToFile('matrice_pigne.xls', zz, 'spec_x_area', styleAction = XLC$
    STYLE_ACTION.XLCONNECT)

#############################
####some useful graphics####
#############################
vegetation$year <- as.factor(vegetation$year)

##base plot
```

Predatori di semi, *pulsed resource systems* e cambiamenti climatici VII

```
plot.0 <- ggplot(vegetation,aes(x=year,y=mean,fill=tree.species))+facet_wrap(~
    tree.species,scale='free_y')+geom_bar(stat = "identity")
plot.0

##Fix axes text and rearrange scale
plot.1 <- plot.0 +theme(axis.text.x = element_text(angle = 90, hjust = 0.5))
plot.1

##Fix axes labes
plot.2 <- plot.1 + ylab('Number of Cones')+ xlab('Year')
plot.2

##Remove legend
plot.3 <- plot.2 + theme(legend.position="none")
plot.3

##changing plot placement
plot.4 <- plot.3 + facet_wrap(~tree.species,scale='free_y',ncol = 1)

##underline mast year
plot.5 <- plot.4 + geom_vline(xintercept=as.numeric(vegetation$year[c(1,3,6,8,45)
    ]),linetype = "longdash")
plot.5

##add title and print out graphic choosing dimension and resolution
plot <- plot.5 +ggtitle('MAST')
png(filename='PLOT.png', width=15, height=20, units='cm', res=300)
```

Predatori di semi, *pulsed resource systems* e cambiamenti climatici VIII

```
59 plot
   dev.off()
```