**ANALISI DATI**

R è un linguaggio di analisi dei dati, nato per analizzarli; esso si compone di un programma aperto il quale presenta dei vantaggi:

* Non ha costo
* Chiunque lo utilizza può replicare le analisi svolte da altri soggetti. Questo è permesso grazie alla condivisione del codice generato il quale rende l’analisi replicabile (è possibile utilizzare gli stessi dati e codice).

Il programma R compone uno strumento di analisi dei dati, se lo guardiamo nel dettaglio troviamo la sua interfaccia a riga di comando, si tratta di un modo di visualizzazione diverso da quanto siamo abituati normalmente. In generale la maggior parte dei programmi che utilizziamo sono settati ad interfaccia grafica.

Di recente era stato proposto di modificare il line up di visualizzazione ma si è optati per mantenerlo in modalità riga di comando al fine di permettere all’utente l’inserimento dei codici. Questo perché i nuovi utenti tendono ad essere abbastanza pigri in quanto non pensano a ciò che stanno facendo. Attraverso l’interfaccia grafica l’utente potrebbe pertanto svolgere funzioni sconosciute senza conoscere l’esatto motivo e utilità.

L’utilizzo della linea di comando permette inoltre di mantenere traccia di tutti i passaggi scritti per ottenere il codice finale, questo comporta notevoli vantaggi:

1. Totale trasparenza nei processi di analisi dei dati
2. Riducibilità: noi attraverso uno script possiamo riprodurre una stessa analisi a distanza di tempo
3. Condivisione: condividere con la comunità scientifica il codice utilizzato. Permette la diffusione di nuove metodiche di analisi comportando una notevole vantaggio comunitario

# Terminologia

R è composto da un linguaggio statistico di tipo open-source dove gli utenti possono scrivere determinati codici e successivamente condividerli. Nel corso del tempo sono stati sviluppati diversi pacchetti di funzioni aggiuntive al programma R, essi contengono un’insiemi di codici argomento specifici scaricabili. In questo modo gli utenti che sono interessati a particolari argomenti scientifici possono trovare un insieme di funzioni già sviluppate e funzionanti per migliorare il loro workflow di lavoro.

I pacchetti a loro volta come possono essere resi disponibili, possono essere anche rimossi dagli amministratori di R. Questo perché il programma è in continuo sviluppo e ogni circa 6 mesi subisce un aggiornamento, ecco che in questo caso non è detto che pacchetti sviluppati precedentemente rimangano compatibili con le ultime versioni del programma (se non risultano essere compatibili vengono rimosse).

Generalmente gli utenti di R sono interessati ad analisi e dati molto specifici che sono accessibili attraverso determinati pacchetti, i pacchetti non sono tutti uguali infatti possiamo avere:

* Metodi per analisi
* Dati da utilizzare come esempi di analisi

In particolar modo per il corso che svolgeremo, tutti i metodi di analisi sono già contenuti in opportuni pacchetti

# Concetti generali

* Bioconductor: compone di una collezione di dati o pacchetti di R specializzati per metodiche di analisi biologiche. I pacchetti in questo caso sono limitati ma specifici per analizzare dati biologici
* R-Studio: per facilitare l’utilizzo di R una compagnia ha sviluppato un’interfaccia grafica chiamata R-Studio che viene installata in seguito all’installazione di R. Attraverso questo programma abbiamo lo stesso il terminal ma, possiamo analizzare i differenti file in modo più comodo permettendoci di visualizzare nuovi file, grafici, pacchetti ed help. Troviamo poi diverse funzioni come quella di Editor che ci permette di scrivere il nostro codice da utilizzare successivamente nel terminal.

Durante le analisi con R è consigliato l’utilizzo di un terminal esterno in quanto quando si analizzano grandi moli di dati può succedere che il programma vada in blocco “crasha” portando inoltre alla perdita dei dati. Alcuni esempi sono:

* Sblime text
* WeBr: viene utilizzato da browser, la sua interfaccia ricorda vagamento R-Studio
* Jumpyter Nootbook: nato per generare tutorial e successivamente utilizzato per il coding.

L’idea in generale è quella di poter dividere il codice in celle a nostra scelta; se infatti ho delle figure complesse con molti punti, l’utilizzo di Jupyter, e anche su R è infattibile in quanto il programma diventa estremamente lento. In questi casi sarebbe più facile stampare direttamente i dati con PDF ed impiegare il PDF reader per visualizzarli. Jumpyter è utile per l’analisi di piccoli dataset statistici.

# Intro R

R è organizzato in Data Structure, si tratta di un’organizzazione in un linguaggio di programmazione. I Data Structure non sono tutte uguali, le principali sono:

* Vettori
* Array e matrici
* Fattori
* Liste
* Data Frame

Oltre alle tipologie di Data structure citate sopra, ne possiamo avere molte altre scaricabili dai vari pacchetti. Un chiaro esempio lo troviamo con Datatable il quale genera un’altra struttura di dati.

A loro volta è importante evidenziare che i vari pacchetti posso creare un datastructure al fine di riorganizzare al meglio i dati. Solitamente si tratta di versioni speciali

# Vettori

I vettori sono una sequenza ordinata di elementi omogenei. Con ordinata intendiamo che un elemento localizzato in una sequenza è diverso rispetto ad un altro elemento contenuto in una diversa sequenza, si tratta di una cosa ovvia che però non risulta essere vera in tutte le tipologie di datastructure.

I vettori sono di tipo omogeneo, ovvero gli elementi che lo costituiscono devono essere tutti uguali, nella sequenza possiamo avere solo numeri di una certa classe (tutti interi o tutti decimali etc.).

Il “tipo” compone un concetto universale nei vari linguaggi di programmazione. A sua volta questi linguaggi possiedono un determinato grado di tipizzazione dove la tipologia in analisi può essere più o meno specializzata. Con TIPI in R intendiamo numeri o testo.

# Funzione

In un linguaggio di programmazione, un operatore è un elemento che permette di eseguire un’operazione. Un esempio molto semplice è quello degli operatori matematici classici, come +, - o la divisione.

In **R**, invece, le operazioni vengono eseguite principalmente tramite **funzioni**, che hanno sempre un nome seguito da parentesi tonde. Ad esempio, una funzione generica si scrive così: funzione().

Per creare un **vettore** in R, si utilizza la funzione c(), abbreviazione di concatenate (concatena). Questa funzione permette di unire più elementi in un unico vettore.

Se si desidera ottenere maggiori informazioni su una funzione in R, si può utilizzare il punto di domanda (?) seguito dal nome della funzione. Questo comando apre la documentazione della funzione, fornendo diverse informazioni utili, tra cui:

* **Il pacchetto di appartenenza** della funzione.
* **La sintassi e gli argomenti** che la funzione accetta. Ad esempio, nel caso di c(), si apprende che tutti gli elementi inseriti nel vettore vengono convertiti allo stesso tipo. Questo perché un vettore in R è omogeneo, ovvero tutti i suoi elementi devono appartenere allo stesso tipo di dato. Se si inseriscono elementi di tipo diverso, R li converte automaticamente affinché siano compatibili.
* Ad esempio, se concateno due stringhe di testo e un numero, ma scrivo il numero tra virgolette, il risultato sarà un vettore composto solo da stringhe.
* Se invece inserisco il numero senza virgolette, R lo convertirà comunque in una stringa, affinché tutto il vettore abbia lo stesso tipo.
* Questo comportamento può essere utile in alcuni casi, ma in altri può causare errori. Ad esempio, se si utilizza una funzione che richiede numeri, ma si passa un valore testuale, si otterrà un errore.
* **Una descrizione dettagliata della funzione**, spiegando il suo funzionamento e l’output che genera.
* **L’uso della funzione**, con esempi pratici.
* **L’elenco degli argomenti accettati**, con una spiegazione per ciascuno.
* **Dettagli aggiuntivi**, tra cui il modo in cui R gestisce la conversione dei dati. R segue una gerarchia di conversione automatica, scegliendo il tipo più "alto" tra quelli presenti nel vettore.
* **L’output della funzione (value)**, ovvero il risultato che restituisce. Se non vengono forniti argomenti validi, il valore restituito è NULL.
* **Oggetti di tipo S4**, per chi utilizza la programmazione a oggetti in R.
* **Riferimenti e suggerimenti**, con collegamenti ad altre funzioni correlate.
* **Esempi pratici**, che permettono di comprendere meglio l’uso della funzione.

Tutte le pagine di aiuto in R seguono una struttura simile: iniziano con una breve descrizione della funzione e poi forniscono informazioni più approfondite, rendendo più semplice l’apprendimento e l’utilizzo delle funzioni disponibili

In **R**, le funzioni possono appartenere a due principali sistemi di programmazione a oggetti: **S3** e **S4**.

* **S3** è il sistema più semplice e più vecchio. Le funzioni S3 sono flessibili, ma meno strutturate.
* **S4**, invece, è un sistema più moderno e complesso, che introduce concetti più avanzati come metodi e oggetti con proprietà ben definite.

# Programmazione Procedurale vs. Programmazione a Oggetti

Quando scriviamo un codice per eseguire un’operazione, possiamo farlo in **modo procedurale**, ovvero scrivendo una serie di istruzioni che vengono eseguite una dopo l’altra. Questo approccio è ancora molto usato, ma ha un limite: se dobbiamo ripetere la stessa operazione più volte, siamo costretti a riscrivere parti di codice, rendendo il programma meno efficiente e più difficile da mantenere.

Per superare questo problema, si è sviluppata la **programmazione a oggetti** (OOP), che permette di creare strutture di codice riutilizzabili. In questo paradigma, il codice è organizzato in **oggetti**, che possono essere richiamati quando servono. Anche in R, ogni elemento è considerato un oggetto: ad esempio, una funzione è un **oggetto funzione** che possiamo utilizzare per svolgere un compito specifico.

# Gli oggetti in R

In R, qualsiasi entità creata è un **oggetto**. Ad esempio, se creiamo un **vettore**, esso diventa un **oggetto di tipo vettore**, che appartiene a una delle cinque strutture dati principali di R.

Uno dei vantaggi della programmazione a oggetti è la possibilità di **ereditare proprietà**:

* Chi ha progettato R ha definito che un oggetto di tipo **vettore** sia sempre un insieme ordinato di elementi **omogenei**.
* Ogni classe di oggetto può avere metodi associati. Ad esempio, un oggetto numerico può avere associato un metodo per eseguire una **somma**.

# Metodi e Automazione

Un aspetto potente della programmazione a oggetti è che, se un certo **metodo** esiste già per un tipo di oggetto, R lo eseguirà automaticamente. Ad esempio:

* Se creiamo un oggetto e vogliamo generare un **grafico** (plot), R verifica se esiste un metodo di **plot** per quel tipo di oggetto.
* Se il metodo è già stato definito, il grafico viene generato automaticamente senza bisogno di specificare ulteriori istruzioni.
* Ad esempio, non dobbiamo dire a R come fare il **plot** di un vettore, perché esiste già un metodo dedicato.

# Nomi degli Oggetti

Nel nostro **spazio di lavoro** possono esserci molti oggetti, e alcuni potrebbero avere nomi molto lunghi. Per questo motivo, è sempre una buona pratica assegnare nomi chiari e significativi agli oggetti, in modo da capire facilmente il loro scopo anche a distanza di tempo.

Vettori possono essere di tipo:  
 - numerici, numeri interi o in virgola mobile  
 - caratteri, singoli caratteri o stringhe di caratteri  
 - logici, è un tipo diverso di valore che può assumere il vettore e può essere TRUE o FALSE

Vettore Logico  
 Si crea nel caso in cui noi ci chiediamo se una cosa è vera o falsa, basta ad esempio fare una richiesta come ad esempio 5 >3 dove R ci ritornerà che è vero p falso se metto il <.  
Per assegnare un valore a un oggetto possiamo usare = o la <-.  
Per chiedere a R se un oggetto è uguale all’altro si usa un ==.   
Nel momento in cui noi mettiamo solo un = tra due oggetti (non tra due numeri come scrivere 5 = 3), non mi darà errore ma lui interpreterà che un oggetto vogliamo che sia uguale all’altro; quindi, bisogna stare attenti quando si mettono gli uguali.

Se noi creiamo un vettore numerico, ad esempio, mynumvec <- c (3,4,5,6) e dopo inseriamo su R munumvec > 4, il programma ci dirà per ogni elemento numerico del vettore quando è vero e falso.  
 Se io voglio creare il vettore logico ad esempio: mylogvec <- (mynumvec > 4), mi crea in vettore logico mylogvec che se lo chiamo sul programma mi da il risultato di tutti i confronti del vettore numerico con >4.  
Le operazioni  
Io creo i miei vettori numerici x e y tipo x <- (1,2,3) e y <- (4,5,6) se io faccio la somma quindi z <- x + y mi da il vettore z numerico con i risultati sommando numero per numero ovvero nell’esempio di x + y avrò come risultato 5 7 9 (è una somma elemento per elemento). Uguale così sia per - / e \*. Posso siamo inserire x + y su schermo e ottenere il risultato o salvarlo in un vettore numerico come può essere z per salvarlo e richiamarlo più avanti.  
 Se io sommo (o faccio un'operazione) tra due vettori numerici di lunghezza diversa; quindi, uno con 4 numeri e uno con 3 mi darà non un errore ma un warning. Mi dal risultato con il warning ma usa il Riciclo, ovvero riutilizza un elemento del vettore più corte. Tramite questa posso fare in modo che richiamando il nome del vettore mettendo a fianco l’operazione come ad esempio: nome vettore + 10, ad ognuno degli elementi numerici è aggiunto 10.  
Se lo rifacciamo con un altro vettore ancora più corto che è la meta di quello più lungo non mi darà il warning perché questo esce solo nel momento in cui il vettore più piccolo non è un multiplo dell’altro, nel caso in cui sia un multiplo per R va bene. Molto utile se si vogliono manipolare i vettori a proprio piacimento.

Operazioni sui vettori  
 Sono operazioni numeriche da fare sui vettori come:  
- media: mean(x)   
- varianza: var (x)  
- range: range (x)  
- valore min e max: min(x) oppure max(x)  
- somma: sum (x)  
- prodotto: prod (x)  
Media  
L’help ci dice le solite informazioni, e negli argomenti ci dice che nell’x della media che deve essere un oggetto R e può farla di oggetti di tipo numerico, logico o date.   
E’ possibile fare un trim con la media, che è una frazione che va dallo 0 a 0.5 delle osservazioni che devono essere tolte da ciascuna parte di x prima che la media sia completata.  
Nel caso noi avessimo le nostre misurazioni come ad esempio il valore di espressione di un gene o di una proteina, sapendo che i sistemi biologici sono molto rumorosi ovvero sono sotto l’effetto di numerose variabili che fanno si che in varie prove l’espressione non è sempre più o meno simile e ci sono degli outliner, per evitare che gli outliner mi diano un risultato poco rappresentativo, io faccio un trim che va da 0 a 0.5 (da 0% a 50%) che mi elimina gli outliner più o meno estremi in base al valoro che do da 0 a 0.5.  
Prima di fare una media e vedere se ci sono outliner potrei:  
- andare a visualizzare con un plot i vari punti e vedere se ce ne sono alcuni fuori scala e questo è un metodo visivo per osservare outliner  
- calcolare il percentile con la funzione quantile(x, probs= ) dove probs mi dice la percentuale che può andare da 0 a 1 e valuto tutti i percentili che mi da su schermo inserendo il valore di probs.  
  
Immaginando di fare un esperimento da cui ottengo diversi valori e mi chiedo se questi abbiano un significato statistico, il segnale dell’esperimento lo confronto con un controllo, quindi creo due vettori numerici uno che è il mio test e uno il mio controllo entrambi di lunghezza uguale, e verifico per entrambi che non vi siano outliner.   
Posso plottare entrambi e per vedere che non vi siano troppo differenze posso fare un rapporto e lo faccio con la fuzione ad esempio: plot (test/ctrl) per vedere come oscilla il rapporto e osservare se non vi è troppa differenza tra le repliche biologiche.  
Per dare un supporto statistico devo fare un test, ad esempio il t.test con cui uso o il controllo e test separati oppure uso la ratio  
t.test  
Il t.test fa un test su uno o due campioni su dati in vettori e confronta le medie e la variazione dei gruppi.  
Nel help del t.test trovo nell’uso:  
- x, che è un argomento  
- y che è il secondo argomento o vettore che posso inserire e di default è nullo  
- alternative dove posso avere “two.sided”, “greater” o “less”, ovvero se noi abbiamo due elementi prendiamo come significativo se sono diversi in una determinata soglia sia che il controllo sia piu basso o alto o entrambi i casi. Il risultato che poi prendo come significativo dipende da cosa mi aspetto dal mio esperimento.   
- paired=FALSE/TRUE considerare i dati dei nostri vettori accoppiati o meno e di default è falso. - var.equal per dire che la varianza tra i due vettori sia uguale, e si intende la varianza dei vettori che considero, inteso sul piano della grandezza, se è diversa non ha senso fare il t.test perché se la varianza è molto diversa è inutile il test.   
- conf.level è il livello di confidenza che di default è del 0.95 ovvero 95% che è la stringenza che diamo al test e va da 0 a 1.  
- mu è un numero che indica il valore della vera media o la differenza media se faccio un two sample t.test (ovvero dove assegno vettori sia a x che y), e di default è 0. Noi con il t.test testiamo l’ipotesi che è che non ci sia differenza tra x e y e valutiamo che l’ipotesi alternativa ovvero che vi sia differenza. Si lavora quindi contro ipotesi, il valore di p value che ci da come risultato è il valore che l’ipotesi alternativa sia vera e che quindi ci sia differenza.  
Scrivere tutti i parametri elencati prima è opzionale, se rispetto l’ordine dell’help in caso contrario li devo inserire.  
  
Se io faccio il t.test two sample tra x = test e y = ctrl, la cui funzione è t.test (x,y) e di default tutti i valori che trovo nell’help lui mi da come risultato il t che mi dice quanto robusto è il test, il df che è il grado di liberta del test e poi vi è il p value che mi da la significatività del test. Di default mi fa i percentili e quindi le confidenze di intervallo.  
Essendo dati biologici se io metto paired = TRUE mi cambia tutti i risultati cambiando anche i pvalue.   
Posso inserire anche alternative che di default è “two.sides”, io lo metto “greater” in modo da prendermi il caso in cui vi è un aumento rispetto al controllo. O in caso contrario se metto “less”.  
 (comando da ricordare è ls() che mi da il nome dei vettori che ho nelle box space)  
  
Io posso calcolare la ratio del test contro il mio controllo e posso con un t.test andare a stimare il mio vettore contro una media ovvero un numero, che mi dice il vero valore della media mu.  
In questo caso faccio t.test (x= ratio (è il nome del vettore), mu = 1), non ha senso parlare di paired e varianza, la nostra ipotesi in questo caso è che la media non sia uguale a 1 e ancora una volta in questo caso il risultato è positivo al one sample t.test perché ha un pvalue molto basso. La mu nel one sample t.test può essere diversa e quindi specifica a seconda dell’elemento che noi vogliamo vedere.  
  
Il t.test è un test parametrico ovvero ha parametri tali che impongono limitazioni come la distribuzione di dati. Tipicamente i dati dovrebbero avere una distribuzione normale per poter essere usati nel t.test, ma come si fa a definire che i dati sono normali? Esistono test per farlo, ma nella maggior parte dei casi questi prevedono un numero di campioni sufficientemente elevati ma spesso non si è in quella condizione, a senso fare questi test nel momento in cui ad esempio si hanno 100 rilevazioni. I test parametrici dicono che i dati abbiano una normale distribuzione, ma spesso non potendo avere questa assunzione usiamo test non parametrici di cui ne esistono diversi, e il caso più semplice è il test di Wilcoxon.  
  
Wilcoxon test  
Il Wilcoxon test è un test non parametrico ovvero che prevede che i dati non siano distribuiti in modo normale, questo test può essere applicato a uno o due set di dati (one sample o two sample data). Lungo l’help questo assomiglia come il t.test a livello di usi. È un test che si basa sui ranghi, quindi io prendo i miei due vettori usati come esempio test e ctrl, e vedo ciascun elemento come si dispone all’interno del proprio spazio di dati del proprio insieme di elementi e anche in quello contrario. Prima di farlo il test gli elementi dei vettori devono essere uniformi, in modo che questo test possa confrontarmi dove sarebbero gli elementi di un vettore rispetto all’altro, mi fa un confronto della posizione dell’elemento di un vettore con i ranghi dell’altro vettore e fa la somma dei ranghi, più i vettori sono diversi più sarà maggiore la differenza e viceversa in caso contrario. Nel caso in cui in un vettore vi siano due elementi uguali, ti da un warning, in quanto il pvalue viene calcolato in modo approssimativo dato dal fatto che le due i due elementi uguali condividono due posizioni.  
Il t.test potrebbe fallire nel momento in cui i valori sono in una scala piu grande e quindi abbiamo vettori numerici piu complessi. Il t.test è molto sensibile al numero dei campioni, quindi aumentandoli si potrebbe abbassare sensibilmente il pvalue, mano a mano che aggiungiamo campioni il t.test ha un p-value che si abbassa sempre di più, e vuol dire che se si hanno tanti numeri anche se la differenza di media tra le due popolazioni è molto piccola, la pvalue andrà sotto la soglia di significatività (è 0.05), e questa cosa puo capitare spesso se si hanno data set pubblici dove si hanno molti campioni con una differenza tra le medie molto bassa.  
Questo per dire che la p-value nel t.test è un elemento che mi dice quanto significativa statisticamente è la differenza, non mi dice quanto la differenza è rilevante, importante quindi quando il numero di campioni crescono, anche piccole differenze sono statisticamente significative, ma non è detto che lo è anche biologicamente e sta all’operatore di laboratorio interpretarlo statisticamente a livello biologico.   
Nel wilcoxon test quanto i due vettori hanno dati molto diversi non funziona bene. Quindi prima di applicarlo bisogna vedere che tipo di dati abbiamo.  
  
Visualizzazione dei dati  
Oltre ai plot (x) e plot (x/y), però posso fare uno scatter plot tra x e y facendo il comando plot (x=….,y=….) e vedere come tra loro si discostano i dati.   
Si possono fare vari plot che si possono vedere sul sito ggplot.   
Altri tipi di plot sono i bar plot, fatti col comando barplot (x), che non è sempre la miglior rappresentazione.  
Immagine che contiene testo, diagramma, schermata, Rettangolo

Il contenuto generato dall'IA potrebbe non essere corretto.Sono suggeriti molto i box plot, la cui funzione è boxplot (x). (puo essere sia in orizzontale che verticale)  
  
Il segno in mezzo al boxo è la mediana, il box di per se comprende i dati tra il 25esimo percentile e il 75esimo percentile, quindi nel box ci sono il 50% dei dati.  
La lunghezza delle due righe che spuntano dai lati è calcolata come il valore massimo del dato Q3+1.5\*IQR. I punti fuori da questo valore sono i possibili outliner.  
Nell’help di boxplot, vi è la descrizione dove mi dice che genera questi box plot di una serie di valori raggruppati e non. È più complesso degli altri plot in quanto qua si puo avere una formula, ovvero un fatto per raggruppare campioni in due gruppi diversi. Immaginando di avere sempre i vettori numeri test e ctrl, possiamo creare una variabile gruppo che mi descrive che alcuni sono test e ctrl, della stessa dimensione dei due vettori testo e controllo. Immaginando di voler ripetere un character, ad esempio, il character grouptest 8 volte, per farlo faccio la funzione rep (“grouptest”, 8). Altrimenti posso fare tutto in una volta rep (“grouptest”, length(x)), e poi lo nominiamo come un vettore character, e di questi ne facciamo due uno per il vettore test e uno per il vettore ctrl:  
  
Creiamo poi la variabile groups (un nome a caso) che è la somma dei due gruppi 1 e 2.  
  
Serve poi avere la somma dei dati dei due vettori tra test e controllo, e creo quindi un altro vettore.  
Immagine che contiene testo, diagramma, linea, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.  
Si puo quindi fare il boxplot  e mi da come risultato visivo:  
Vedendo l’immagine direi che sono evidentemente diversi, ma per la statistica non lo è a causa della varianza. I test statistici non misurano solo le medie ma anche la distribuzione e la varianza dei dati, per questo i test statistici ti possono confermare che non vi è in realtà tanta differenza a causa della grossa varianza.

# Alcune funzioni utili:

* sort (x, decreasing=FALSE) ordina gli elementi numerici di un vettore in modo crescente di default. Queste perché di default è decreasing=FALSE, se io cambio l’opzione in TRUE faccio in modo decrescente. Vi è anche l’opzione na.last= FALSE che mi mette gli NA per primi0, e se l’attivo mettendo TRUE, li inserisce come ultimi (se noi scriviamo un’opzione con un nome parziale R inserirà quella più vicina come opzione a quello scritto dall’operatore, se non può scegliere da errore).
* order(x) che crea un vettore che riporta gli indici che servirebbero per ordinare il vettore per metterlo ordinato in modo crescente, ordina il mio vettore non ordinando direttamente i valori numerici, ma ordinando le loro posizioni, quindi dice che prima va casella numero tot, poi altra casella numero etc…

Ci sono varie funzioni per creare vettori numerici:  
+ posso fare sia c(1:10) e vare un vettore crescente da 1 a 10, viceversa se faccio c(10:1)  
+ posso creare una sequenza da 1:10 tramite sec(1:10), che è più potente, perché con lui posso specificare maggiori cose ad esempio posso fare sec (from=1, to=4, by=0.5).  
+ esiste anche una versione per vettori caratteri seq\_along(x)

La selezione ci permette di selezionare e ottenere elementi o subset di elementi, e la si può fare in 4 modi diversi, tutti giusti ma alcuni più utili di altri. In generale tutte queste operazioni prendono vita mediante l’uso dell’operatore [ ], che selezione un elemento mediante gli indici presenti all’interno delle parentesi quadre.  
Quindi posso selezionare l’elemento di un vettore, ad esempio con x[3] prendo il terzo elemento del vettore x, la funzione sarebbe nome vettore [x], e con length(nome vettore) so quanto è lungo.

-se io voglio ottenere il massimo valore di un vettore massimo, faccio maxvalue <-max(x).  
nel momento in cui io voglia fare la stessa cosa di max(x) senza usare quella funzione devo usare queste 3 righe di codice sottostanti (i nomi tra le parentesi sono i nomi dei vettori.  
  
Posso fare la stessa cosa tramite il comando sottostante.  
  
Noi di qualsiasi oggetto possiamo selezionare più di un valore ad esempio tramite: nomevettore [ c(1,5)], dove 1 e 5 sono le due posizioni che voglio.

Se io invece creo un altro vettore myindexorder<- order (myvector) e poi faccio myvector[myindexorder], faccio prima la funzione order dove mi mette in ordine le posizioni, poi lo riordino, quindi faccio la stessa roba di sort (i nomi myvector e myindexorder sono i nomi dei vettori scelti in questo caso).

È possibile fare la stessa cosa con un vettore carachter. Noi abbiamo il vettore e possiamo ordinare con lo stesso ordine di un vettore numerico il vettore character, lo sorto con lo stesso ordine del vettore numerico. Dove myindexorder è un vettore numerico. È utile in quanto nelle analisi abbiamo un vettore con character associato a un vettore numerico.  


-se io voglio togliere un elemento uso sempre le [] con indice negativo, quindi x[-2], dove in x ho il nome del vettore e con 2 indico la posizione da eliminare all’interno del vettore. Un altro modo per farlo è il seguente (tra le parentesi quadre vi sono le due posizioni da eliminare).  
  
Se voglio togliere il primo e l’ultimo del quale non so la posizione faccio il comando sottostante  
  
Un’altra opzione per selezionare oltre agli indici positivi e negativi sono i vettori logici, ad esempio se faccio  prende tutti gli elementi che sono veri.   
Se io voglio prendere gli elementi superiori a un determinato valore, posso andarli a selezionare creando anche un nuovo vettore numerico, dove gli dico di prendere i valori maggiori di un determinato valore. Cosi poi da poter selezionare con l’operatore [ ] i valori TRUE, ovvero i numeri al di sopra di un determinato valore.  
Immagine che contiene testo, Carattere, schermata

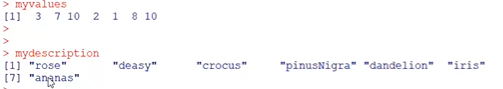
Il contenuto generato dall'IA potrebbe non essere corretto.Missing Data

Spesso accade che non di tutti gli elementi e variabili non si abbia a disposizione tutti i dati. Siccome è una cosa nota, in R vi è un valore speciale NA, che non è un elemento carachter, ma è un valore speciale, con valore diverso. In generale quando si ha a che fare con un operatore che definisce una cosa non disponibile, le operazioni con questo danno come risultato non disponibile ovvero NA. Per identificare quali elementi NA ho in un vettore faccio la funzione: is.na (x). Se io voglio escludere dai calcoli NA, metto tra le parentesi tonde dell’operatore na.rm= TRUE, anche se non tutte le funzioni è possibile.

Se io voglio toglier NA dal mio vettore, x <- is.na(y) poi x [y], dove x e y sono nomi di vettore, e in questo modo selezione NA, a me serve invertire la cosa in quanto così seleziono solo NA, quindi lo riscrivo cosi x <- !is.na(y) poi x [y], e per avere un vettore senza NA ne creo uno nuovo, w <- x[y].  
Immagine che contiene testo, Carattere, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

Se io voglio che un elemento nel mio vettore numerico diventi un altro valore   
, dove faccio il modo che il primo elemento sia sotituito con un 5. Può essere utile nel momento in cui la funzione che vogliamo fare non sa gestire gli NA, oppure li vogliamo togliere perché vogliamo plottare i valori.   
Ad esempio, è possibile farlo tramite il seguente comando.  
  
Dove vado a sostituire l’NA con uno zero.

Immaginando di avere dei valori risultanti da un t.test o wilcoxon test, quindi un vettore numerico con valori molto piccoli, dobbiamo plottarli, devo trasformarli in scala logaritmica per visualizzarli meglio, per far ciò:  
  
Abbiamo convertito le pvalue e messe positive. Le pvalue sono strumenti utili a noi per capire se un risultato è significativo o meno a livello statistico, e incrementando il numero di repliche le pvalue iniziano a scendere. Può capitare che in una serie di test statistici vi siano valori molto vicini allo zero, che non permettono la corretta visualizzazione in un grafico barplot.  
Possiamo anche inserire una soglia che ci dice che se il nostro valore è superiore a una soglia, allora il nostro valore viene messo pari al valore di treshold.  
Devo costruire una soglia, un elemento che mi faccia da TRUE o FALSE, riprendo il mio elemento e seleziono utilizzando il TRUE o FALSE che mi sono costruito.  
  
  
Character Index Vecotr  
Per adesso abbiamo visto solo vettori nudi, in R possiamo aggiungere ad ogni elemento del vettore una descrizione. Dobbiamo aggiungere al vettore l’attributo names.   
  
Partiamo dai due tipi di vettori della stessa lunghezza, per poi usare l’attributo names.  
  
Ottengo questo risultato:  
Immagine che contiene testo, Carattere, schermata, ricevuta

Il contenuto generato dall'IA potrebbe non essere corretto.  
E’ possibile poi andare a fare una selezione mediante il nome, ad esempio possiamo selezionare solo rosa o iris, pero tra le “ “ perché sono due character.  
Immagine che contiene testo, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

Comandi utili per lo svolgimento degli esercizi:

* Set.seed e runif

Algoritmo seed per generare numeri casuali in modo riproducibile, questo perché non sono veri e propri numeri casuali ma partono da un valore, ad esempio l’ora, e da questo valore iniziale vengono sviluppati i numeri casuali un algoritmo. R è robusto, buoni numeri random. Ad esempio, se scrivo set.seed (42) allora 42 è il numero da cui parte a generare numeri casuali. Ogni volta si esegue runif dopo aver impostato set. seed, si otterranno gli stessi numeri casuali.

> set.seed (42)

> vet <- runif (100)

Verrà quindi creato un vettore di 100 numeri casuali uniformi. Se riscrivo quanto scritto sopra otterrò un vettore con gli stessi numeri. Possiamo verificare ciò:

> set.seed (42)

> vet <- runif (100)

> set.seed (42)

> vet2 <- runif (100)

> vet == vet2

Il risultato sarà un vettore con tutti i valori “TRUE”.

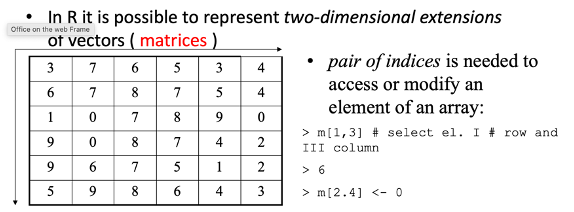
* data.score

Funzione per confrontare l’andamento tra due serie: se fossero su diversa scala potremmo avere risultati non corretti, quindi si normalizzano i dati con data.score.

# MARTICI:

Vettori se hanno più dimensioni si chiamano matrici.

* **Vettore**: unidimensionale, tra quadrate singolo indice di sub setting.
* **Matrice**: due dimensioni, dati disposti in griglia con colonne e righe. Godono delle stesse proprietà dei vettori, ma avendo due dimensioni c’è la necessità di individuare gli elementi bi-dimensionalmente, necessitiamo quindi di due indici, uno che indica la riga (primo indice) e uno la colonna (secondo indice).

Esempio nell’immagine: m [1,3] restituisce 6

Per R gli **array** sono l’estensione multidimensionali delle matrici, n dimensionale, ad esempio: a [1,3,4]. L’unico limite è la memoria. La matrice, quindi, è considerabile sottocategoria degli array. Utile usare array quando ogni dimensione descrive un parametro dello stesso oggetto, ad esempio nel caso di una pianta un elemento ne descrive la lunghezza, uno colore dei fuori, uno spessore del fusto etc. in goni punto n dimensionale trovo descrittore con i valori della pianta in esame. Molto bello, ma poco utilizzato.

Le matrici sono estensioni di vettori in bidimensione, gli elementi devono essere omogenei al suo interno come per i vettori, quindi o tutti numeri, o logical o character.

**Come si costruisce una matrice:**

ho un vettore creato con una sequenza, ad esempio, x <- 1:24 e vogliamo metterlo in una matrice dobbiamo indicare come è formata la matrice, ad esempio ha 4 righe e 6 colonne. m è la matrice quindi m <- matrix (x, nrow = 4, ncol = 6, byrow = TRUE).

> m

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 1 2 3 4 5 6

[2,] 7 8 9 10 11 12

[3,] 13 14 15 16 17 18

[4,] 19 20 21 22 23 24

Se ho 24 elementi e metto 7 colonne ho dello spazio vuoto che R non sa riempire quindi dà un warning, non un errore, e dice che 24 non è un multiplo o sottomultiplo; waring perché ha restituito una tabella usando la regola del riciclo che ha riempito con i dati iniziali gli spazi vuoti. Se do un ncol = 12 non mi dà nessun warning inquanto multiplo, ma bisogna prestare attenzione perché c’è il riciclo dei valori.

# Proprietà delle matrici:

- lenght (m): restituisce la lunghezza della matrice, conta gli elementi della matrice indipendentemente dalle dimensioni.

- mode (m): restituisce il tipo di valori, se numerici, logical o carattere.

* dim (m): restituisce il numero di righe e di colonne. (NB: sempre le righe prima). Se chiedessi dim del vettore mi darebbe NULL perché non ha dimensioni.

# Funzioni utili per costruire o unire le matrici:

con la funzione bind si uniscono rispetticamente

* cbind (m): le colonne, unisce vettori o matrici in modo orizzontale

> x <- 1:3

> y <- 4:6

> m <- cbind(x,y)

> m

x y

[1,] 1 4

[2,] 2 5

[3,] 3 6

* rbind (m): le righe, unisce vettori o matrici in modo verticale

> x<- 1:3

> y <- 4:6

> m <- rbind(x,y)

> m

[,1] [,2] [,3]

x 1 2 3

y 4 5 6

Vengono conservate tutte le info che posso richiamare se tra parentesi quadre indico gli indici di righe e colonne, posso anche richiamare i dati di una intera riga, ad esempio, se scrivo m [1, ] vengono restituiti tutti i valori della prima riga.

> m [1, ]

[1] 1 2 3

Posso fare bind anche tra due matrici m e m2. m e m2 hanno elementi sempre che si chiamano x e y ma R non li confonde, li riconosce come appartenenti a due matrici diverse. Attenzione quando fai bind tra due matrici con x y e x2 y2 chiamerà gli elementi del bind x e y.

# Operazioni:

- Somme di due matrici come somme di due vettori, sommate per indici (prima riga prima colonna delle due matrici etc.) questa vale per tutte le operazioni.

- Il prodotto tra due matrici serve quando si vuole far interagire le matrici in maniera che si semplifichino, operatore per il prodotto di matrici è %\*% e non \*. Il numero di righe di una matrice deve essere il numero di colonne dell’altra.

- Trasporre matrici permette di girare colonne con righe mediante l’operatore t (m). Se traspongo un vettore x con t (x) allora lo trasforma in una matrice avente su una sola riga, se queto risultato lo trasponiamo di nuovo otteniamo una matrice avente una sola colonna.

> x <- 10:12

> x

[1] 10 11 12

> t(x)

[,1] [,2] [,3]

[1,] 10 11 12

> traspostodix <- t(x)

> t(traspostodix)

[,1]

[1,] 10

[2,] 11

[3,] 12

Attenzione! Se vuoi richiamare una riga di una matrice bidimensionale essa verrà semplificata come vettore, scrivendo m [, “y”] dove la colonna che si chiama Y, allora verrà restituito un vettore avente i valori della colonna Y.

> m

x y

[1,] 10 14

[2,] 11 15

[3,] 12 14

> m [,"y"]

[1] 14 15 14

Se chiediamo con class cosa sia m [, “y”] ti dice che è un vettore:

> class (m [,"y"])

[1] "integer"

# Calcolo matriciale:

Calcolo matriciale usa funzioni specifiche, ad esempio per trovare la diagonale di una matrice, risolvere una matrice, il determinante di una matrice etc. ma a noi non frega un cazzo.

# Accedere agli elementi delle matrici:

Le regole di accesso per array e matrici seguono quelle già viste per i vettori, considerando però l'esistenza di più indici e quindi la possibilità di utilizzare un vettore per ogni dimensione:

* Vettori di indici interi positivi
* Vettore di indici interi negativi
* Vettore degli indici logici
* Vettori dell'indice dei caratteri

Viene quindi utilizzato un vettore di indici per ogni dimensione dell'array.

Esempio, costruisco la seguente matrice

> m <- matrix (1:12, nrow=2)

> m

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 1 3 5 7 9 11

[2,] 2 4 6 8 10 12

Voglio estrarre il valore in posizione riga 1 e colonna 2:

> m [1,2]

[1] 3

Voglio estrarre i valori in riga 1 e colonne da 3 a 4:

> m [1,3:4]

[1] 5 7

Voglio estrarre i valori nelle righe da 1 a 2 e colonne da 4 a 6:

> m [1:2,4:6]

[,1] [,2] [,3]

[1,] 7 9 11

[2,] 8 10 12

Voglio estrarre tutti i valori della prima riga:

> m [1,]

[1] 1 3 5 7 9 11

Voglio estrarre tutti i valori della terza colonna:

> m [, 3]

[1] 5 6

Voglio estrarre i valori di tutte le righe e delle colonne 1, 2 e 6

> m [, c(1,2,6)]

[,1] [,2] [,3]

[1,] 1 3 11

[2,] 2 4 12

Voglio estrarre i valori di tutte le righe e tutte le colonne eccetto la 4:

> m [,-4]

[,1] [,2] [,3] [,4] [,5]

[1,] 1 3 5 9 11

[2,] 2 4 6 10 12

Voglio estrarre tutti i valori maggiori di 4 all’interno della matrice (ottengo un vettore)

> m [m>4]

[1] 5 6 7 8 9 10 11 12

# Dare nomi a righe e colonne di una matrice:

Uso la funzione dimnames a cui attribuisco i nomi di colonne e righe mediante la funzione list (nomi righe, nomi colonne). In questo caso avendo solo due righe possiamo creare un vettore dove scriviamo direttamente i nomi, quindi c(“r1”,”r2”). Per le colonne che invece sono più numerose anziché creare un vettore c(“c1”,”c2”…) sino a c6 usiamo la funzione paste che unisce più stringhe di testo, il primo testo è “c”, il secondo il numero della colonna che va da 1 a 6, ed infine specifichiamo di non volere un separatore tra le stringhe di testo “c” ed il suo numero, quindi indichiamo sep=”” senza nulla tra le virgolette.

> dimnames(m)<- list (c("r1","r2"), paste ("c",1:6, sep=""))

> m

c1 c2 c3 c4 c5 c6

r1 1 3 5 7 9 11

r2 2 4 6 8 10 12

Se vogliamo prendere un sottoset o semplice colonna da una matrice, consiglia di usare i nomi di queste righe/colonne (meglio specificare nomi e non numeri, mentre se usiamo numeri viene difficile leggere il codice). Ad esempio:

> m["r1","c2"]

[1] 3

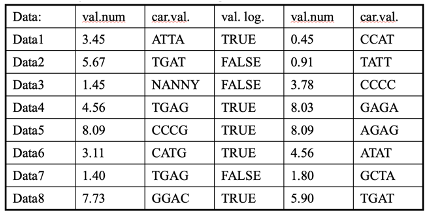
> m["r1",]

c1 c2 c3 c4 c5 c6

1 3 5 7 9 11

# DATA FRAME:

Sino ad ora siamo stati limitati da un constrain ovvero il fatto che matrici e vettori dovevano essere composti da elemento omogenei. In alcuni casi è utile avere nello stesso momento annotazioni e valori delle osservazioni, è necessaria una struttura dati che accolga entrambe queste affermazioni ovvero i data frame.

**Dataframe**: strutture dati che ospitano dati disomogenei. Ad esempio, matrice contenenti dati numerici in colonna 2 e 5, logici in colonna 4 e testi in colonne 1,3, 6. (immagine)

Le proprietà viste sino ad ora si riflettono anche in dataframe.

Dataframe sono una classe detta liste. Gli elementi dei dataframe possono essere alti data frame, situazione innestate in cui un elemento è un altro dataframe.

# Come si costruiscono i dataframe:

Possiamo partire da uno o più vettori come si fa anche per le matrici.

> x <- 1:4

> y <- 5:8

> z <- paste("A",1:4,sep="")

> da.fr <- data.frame(x,y,z)

> da.fr

x y z

1 1 5 A1

2 2 6 A2

3 3 7 A3

4 4 8 A4

Se scrivo str(df) viene descritta la struttura del dataframe.

> str(df)

'data.frame': 4 obs. of 3 variables:

$ x: int 1 2 3 4

$ y: int 5 6 7 8

$ z: chr "A1" "A2" "A3" "A4"

Posso aggiungere una colonna costituita dal vettore j

> j←c(5,6,7,8)

> df2←data.frame(df,j)

> df2

x y z j

1 1 5 A1 5

2 2 6 A2 6

3 3 7 A3 7

4 4 8 A4 8

Se volessimo unire j ma cambiargli il nome in varj:

> df2←data.frame(df,varj=j)

> df2

x y z varj

1 1 5 A1 5

2 2 6 A2 6

3 3 7 A3 7

4 4 8 A4 8

Dando un’occchiata su help di data.frame di default check names = TRUE, quindi i nomi devono essere diversi perché sono considerati due variabili, a differenza che nel caso della matrice. Ad esempio, se volessimo chiamare j con il nome x, ma x è già il nome di una colonna allora R si incazza e te la chiama x.1

> df2←data.frame(df,x=j)

> df2

x y z x.1

1 1 5 A1 5

2 2 6 A2 6

3 3 7 A3 7

4 4 8 A4 8

Creiamo in dataframe più complesso composta da una matrice e un vettore:

> m1 <- matrix(1:12,nrow=2)

> v <- c("A","C")

> daf3 <- data.frame(m1,v)

> daf3

X1 X2 X3 X4 X5 X6 v

1 1 3 5 7 9 11 A

2 2 4 6 8 10 12 C

La cui struttura è:

> str(daf3)

'data.frame': 2 obs. of 7 variables:

$ X1: int 1 2

$ X2: int 3 4

$ X3: int 5 6

$ X4: int 7 8

$ X5: int 9 10

$ X6: int 11 12

$ v : chr "A" "C"

Attenzion! Se cerco di inserire numero dispari di valori del vettore v1 allora R mi dà errore:

> v1 <- c("A", "C", "G")

> v1

[1] "A" "C" "G"

> daf4 <- data.frame(m1,v1)

Error in data.frame(m1, v1) :

arguments imply differing number of rows: 2, 3

Se pari usa regola riciclo:

> v1 <- c("A", "C", "G", "T")

> v1

[1] "A" "C" "G" "T"

> daf4 <- data.frame(m1,v1)

> daf4

X1 X2 X3 X4 X5 X6 v1

1 1 3 5 7 9 11 A

2 2 4 6 8 10 12 C

3 1 3 5 7 9 11 G

4 2 4 6 8 10 12 T

È possibile associare in un set di valori di un colore:

> daf4 <- data.frame(m1,v1)

> daf4

X1 X2 X3 X4 X5 X6 v1

1 1 3 5 7 9 11 A

2 2 4 6 8 10 12 C

3 1 3 5 7 9 11 G

4 2 4 6 8 10 12 T

> daf4 <- data.frame(daf4, color="red")

> daf4

X1 X2 X3 X4 X5 X6 v1 color

1 1 3 5 7 9 11 A red

2 2 4 6 8 10 12 C red

3 1 3 5 7 9 11 G red

4 2 4 6 8 10 12 T red

> daf3

X1 X2 X3 X4 X5 X6 v

1 1 3 5 7 9 11 A

2 2 4 6 8 10 12 C

> m1 <- matrix(10:22,nrow=2)

Warning message:

In matrix(10:22, nrow = 2) :

data length [13] is not a sub-multiple or multiple of the number of rows [2]

> v <- c("Z","X", " J", "K")

> daf3 <- data.frame(m1,v)

> daf3

X1 X2 X3 X4 X5 X6 X7 v

1 10 12 14 16 18 20 22 Z

2 11 13 15 17 19 21 10 X

3 10 12 14 16 18 20 22 J

4 11 13 15 17 19 21 10 K

> daf3 <- data.frame(daf3, color="green")

> daf3

X1 X2 X3 X4 X5 X6 X7 v color

1 10 12 14 16 18 20 22 Z green

2 11 13 15 17 19 21 10 X green

3 10 12 14 16 18 20 22 J green

4 11 13 15 17 19 21 10 K green

> daf4

X1 X2 X3 X4 X5 X6 v1 color

1 1 3 5 7 9 11 A red

2 2 4 6 8 10 12 C red

3 1 3 5 7 9 11 G red

4 2 4 6 8 10 12 T red

# Accedere agli elementi di un dataframe:

Ai accede agli elementi dei dataframe allo stesso di come si accede agli elementi di una matrice. C’è solo una cosa in più ovvero il $ che indica il nome della colonna. Ad esempio, se scrivo daf4$X1 mi restituisce tutta la prima colonna; ma nessuno ha indicato il nome X1 ai nomi delle colonne, ma dato che abbiamo imposto il nome “color” all’ottava colonna allora lui ha dato i nomi a quelle precedenti. Ma porcoidddio.

> daf4$X1

[1] 1 2 1 2

# Come dare i nomi a colonne e righe:

rownames e colnames, sia per dataframe che per matrici. Ad esempio

> rowname <- c (“ex1”, “ex2”, “ex3”, “ex4”)

> daf4

X1 X2 X3 X4 X5 X6 v1 color

ex1 1 3 5 7 9 11 A red

ex2 2 4 6 8 10 12 C red

ex3 1 3 5 7 9 11 G red

ex4 2 4 6 8 10 12 T red

# Subset di elementi:

subset: ritorna un subset di vettori, matrici, dataframe (prima c’è una condizione TRUE/FALSE). Missing values presi come FALSE.

Subset mediante funzione subset, ad esempio subset (daf4, X3>5)

> subset(daf4,X3>5)

X1 X2 X3 X4 X5 X6 v1 color

ex2 2 4 6 8 10 12 C red

ex4 2 4 6 8 10 12 T red

Stesso risultato anche con which:

> which(daf4[,"X3"]>5)

[1] 2 4

> daf4[which(daf4[,"X3"]>5) , ]

X1 X2 X3 X4 X5 X6 v1 color

ex2 2 4 6 8 10 12 C red

ex4 2 4 6 8 10 12 T red

# Operatore %in%:

Altro operatore utile è %in% che cerca gli elementi all’interno di un vettore, matrice, dataframe. Ad esempio voglio cercare attraverso il nome delle variabili:

> colnames(daf4)

[1] "X1" "X2" "X3" "X4" "X5" "X6"

[7] "v1" "color"

> daf4\_colnames ← colnames(daf4)

> daf4\_colnames

> daf4\_colnames <- colnames(daf4)

> daf4\_colnames

[1] "X1" "X2" "X3" "X4" "X5" "X6"

[7] "v1" "color"

> daf4\_colnames %in% “X2” questo è in realtà uguale a > daf4\_colnames == ”X2”

[1] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE

> daf4\_colnames %in% c(“X2”,”X6”) questo è in realtà uguale a daf4\_colnames == ”X2” | daf4\_colnames == ”X6”

[1] FALSE TRUE FALSE FALSE FALSE TRUE FALSE FALSE

Ma con %in% restringo gli spazi senza scrivere ogni volta daf4\_colnames == ”” | …

Esercizi slide 51 (chiede di usare iris data set, prova a capire da solo come inserire questo dataset di R nel codice)

Il dataframe può tenere diversi tipi di dati al suo interno.  
I dati solitamente sono rappresentati a forma tabellare di solito almeno che non ci siano casi specifici i dati sono una tabella sempre.  
Ci sono elementi di base di R utili per caricare le tabelle. Fino ad adesso abbiamo creato dataset all’interno del nostro elemento.   
Come salvare dati sia nella forma tabellare che quella di R.   
Esistono diversi pacchetti per caricare dati. Normalmente i dati possono rappresentarsi come colonne che sono label, e le colonne possono essere numerici o character.  
Nel momento in cui abbiamo i dati su excel, i dati sono scritti in una forma (t)xml e poi zippati, i file di excel sono quindi zippati. Nel momento in cui noi andiamo a leggere il file di excel come file di testo, avremo un file binario da cui non capiamo nulla.   
Se salviamo il file come file di test e lo importiamo su R possiamo leggerlo volendo, il file di testo è quindi il più versatile e più facile da importare.   
  
Esistono diverse funzioni per importare file di testo e la più semplice è:  
Read Table  
Che è nel pacchetto utils presente nel pacchetto di base di R e la descrizione ci dice che legge un file in formato tabella per poi creare un dataframe da esso, con gli elementi corrispondenti alle linee e alle variabili ai campi nel file.   
Trami Help vediamo che nell’uso di Read teable è richiesto:

* File
* header FALSE/TRUE, ovvero solitamente la prima riga che ha informazioni riguardanti le colonne, e di default è FALSE
* sep “ “, in quanto vuole sapere qual è il separatore dei campi, ovvero da cosa sono separati gli elementi, di default è messo come niente, ma noi sappiamo che in excel passiamo da un valore all’altro con tab perché di norma i file a loro volta sono salvati con il tab,
* quote “ \ “ ‘ “, vuole sapere qual è il carattere a fare le virgolette e possono essere “ o ‘
* dec “.”, dove di default è il .
* se deve leggere i col.names o i row.names
* na.string, ovvero qual è la stringa per quanto riguarda gli NA dove di default è NA, dove lui mette NA dove li legge
* nrows, per sapere quante righe ci sono nel file che di default è -1 ovvero infinite, dove leggo tutte le righe che trovo
* skip, che mi dice se deve saltare alcune righe e di default è =0 per dire che non ne deve saltare neanche una
* comment.char, dove R legge il file e se incotnra il simbolo # (di default è comment.char=”#”), ovvero se incontra # ignora tutto quello che vi è nella riga

Noi per specificare a R da dove andare a prendere i file, ad esempio prima di fare read table, bisogna fare il comando stwd(“/home) per dire dove è la nostra work directory, dove se c’è da specificare un’altra cartella dopo home inserisco \ con il nome della cartella da cui prendere il file.  
Oppoure dentro R studio è possibile settare la working directory andando su session dove compare il menu a tendina con set working directory, per poi scegliere l’opzione choose working directory e selezionare la cartella interessata.  
  
Dentro R esistono dei comandi che servono per manipolare i file prima di leggerli, per esempio esiste il comando **dir()** che serve a leggere le directory**.**Se vediamo l’help di dir(), questo fa parte del pacchetto base, di default dir ha:  
-path = “.”, ovvero mi dice difault ifile presenti nella direcotry in cui siamo in quel momento, ovvero la cartella, per selezionare un’altra cartella basta scriverlo entro parentesi, ad esempio, dir (“home/silvanopiazza”).

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.Se chiamiamo la funzione read.table (file=”tabellavirgola.text”, sep = “,”) per creare il dataframe e poi creiamo un vettore con il nome dfvirgola con dentro il dataframe, se facciamo la struttura str(dfvirgola) ci dice che è un dataframe, possiamo anche chiedere la classe con il comando class (), per avere la conferma che è un dataframe.  
  
Il warning che è uscito mi dice che la linea finale è incompleta, perché bisonga andare a capo. In questo caso è stato caricato il file con la virgola.  
Provando a caricare il file con il tab, purtroppo tab non è espresso a parole ma con il simbolo \t.  
Immagine che contiene testo, Carattere, schermata, bianco

Il contenuto generato dall'IA potrebbe non essere corretto.  
Vi è un altro parametro da inserire che può essere \n per andare a capo.  
  
Possiamo provare anche a salvare i file e per questo vi è **Write table ()** e tramite l’help abbiamo le funzioni possibili con questo comando, ovvero nell’uso è specificato: **-**  file ovvero l’argomento da salvare  
- append di default FALSE, ma se il l’argomento deve essere aggiunto al file va messo TRUE  
-quote = TRUE di default, ovvero se vogliamo che le virgolette vadano tra i nostri elementi  
- sep = “ “, di default, ci chiede il separatore  
- eol = “\n” ci chiede quando finisce la linea e di default è \n  
- na= “ NA “, ci dice come trattare gli NA  
- dec = “. “, dove è chiesto qual è il decimale  
-row.names = TRUE e col.anmes = TRUE, di default sono così  
  
Nel secondo e terzo abbiamo prima i row names e poi sia i col che i row e come risultato abbiamo:  
  
solo row  
  
Sia col che row   
  
Se io importo da un file word la tabella con read table normale, i row names e col names sono importati correttamente, mentre se importo da excel risultano sballati.  
  
Questo è come dovrebbero essere su excel, mentre se li importo su R li avrò cosi:  
Immagine che contiene testo, Carattere, tipografia, bianco

Il contenuto generato dall'IA potrebbe non essere corretto.  
Se andiamo a vedere le opzione di default del comando read table, vediamo che l’header è di default falso, quindi dobbiamo metterlo TRUE (sempre nel caso in cui si importa da excel). Già solo mettendo header = TRUE abbiamo che:  
  
Row names funziona solo se gli elementi sono minori a quelli di un'altra riga, se invece ci sono solo righe omogeno ovvero con lo stesso numero di elementi allora non funziona. Quindi nel caso in cui è un numero uguale, posso settarli io attraverso il comando rownames(), come nell’esempio qua sotto.  
Immagine che contiene testo, schermata, Carattere, algebra

Il contenuto generato dall'IA potrebbe non essere corretto.  
Se ci da fastidio la prima colonna possiamo toglierla tramite un indice negativo, ad esempio:  
Immagine che contiene testo, Carattere, bianco, calligrafia

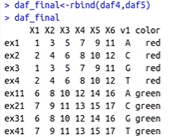
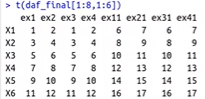
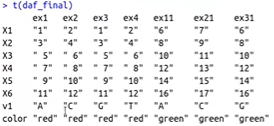
Il contenuto generato dall'IA potrebbe non essere corretto.  
Se si hanno dati da Excel solitamente è meglio trasformarli in testo e poi importarli come testo, per comodità. Esistono anche altre opzioni più moderne dove si può leggere e scrivere da Excel ma sono più complesse, e queste opzioni sono in diversi pacchetti ed ognuno ha la sua sintassi  
  
Ci sono migliaia di pacchetti per varie task ed alcuni sono ad esempio specifici solo per la bioinformatica. Bioconductor project si chiama e appunto è specifico per la bioinformatica. Come installare pacchetti? Mediante il comando instal.packages (“nome pacchetto”).

Bioconductor project ha un proprio installer di pacchetti, ovvero BiocManager da installare sempre mediante la funzione instal.packages (“biocmananger”) e di seguito scrivere biocmanager::insall(“monocel”). Monocel è un pacchetto di single cell RNA seq, esso si appoggia ad altri pacchetti che si trovano elencati su bioconductor sotto la voce depends; se non hai già installati questi pacchetti da cui dipende li installa lui.

Come trovare i pacchetti? Vado su bioconductor cercando manualmente, mediante tasks e descrizioni; oppure chiedo a Google o chat gpt.  
  
Per importare ed esportare su Excel, vi è write.xlsx e read.xlsx, sono comodi nel momento in cui si ha a che fare con persone meno esperte, oppure se il file da salvare è più complesso ed è necessario organizzarlo in più sheet.  
  
È possibile salvare il workspace tramite il menu a tendina che esce da session, dove posso trovare l’opzione save workspace. E poi andare nella cartella e selezionare il file ed aprilo con R studio.  
Se noi vogliamo cambiare l’elemento all’intero di un dataframe come ad esempio il nome di una colonna usiamo il seguente comando, che funziona grazie alla regola del riciclo.  
Immagine che contiene testo, Carattere, bianco, algebra

Il contenuto generato dall'IA potrebbe non essere corretto.  
Se io voglio cambiare anche i numeri, possiamo quindi prendere le porzioni con i numeri e le possiamo dare un valore, ad esempio vogliamo aumentarle di 5:  
Immagine che contiene testo, Carattere, ricevuta

Il contenuto generato dall'IA potrebbe non essere corretto.  
E poi andiamo ad inserirlo nel dataframe daf5 tramite un subset  
Immagine che contiene testo, Carattere, bianco, algebra

Il contenuto generato dall'IA potrebbe non essere corretto.  
Possiamo anche unire i due dataframe tramite:  
  
Di questo è possibile fare un box plot di una o più colonne, nel caso di una colonna e nel caso di due:  
  
Oppure è possibile farlo di tutte e 6:  
  
Posso anche invertire la posizione delle colonne verticali con quelle orizzontali tramite il seguente comando e ottengo la trasposta:  
  
Di questo posso fare anche fare un boxplot con i valori sulle x e y invertiti ma senza i colori. In questo momento l’oggetto con cui stiamo lavorando è la trasposta del dataframe originale e si hanno infatti i colori da un’altra parta (se prendo tutti i valori da 1:8 e basta):  
  
  
Posso fare un boxplot di questo e dare un colore in base ai colori assegnati nel dataframe tramite il comando:  
  
Per fare una PCA estraiamo la parte di dati  che sono trasposti in questo caso (il bro btw sta cacciando tutto su chat gpt, ma fa qualcosa anche lui?). E stato dato il dataset completo su chat gpt e chiesto che codice usare dando tutte le informazioni riguardanti il tipo di plot che si vuole ottenere.

# FATTORI:

Strutture dati che rappresentano valori che possono essere presi da un pool di valori discreti, ad esempio avendo una categoria come ad esempio i colori, i valori possono essere i vari colori rosso, giallo, verde etc.; oppure categorie generiche come maschio e femmina. I fattori si distinguono dal vettore classico perché automaticamente contengono gli elementi unici che sono presenti, ovvero le stesse categorie che sono chiamate livelli.

> sexvector<-c("male","male","female","male","female","female")

> class(sexvector)

[1] "character"

> is.vector(sexvector)

[1] TRUE

> unique (sexvector)

[1] "male" "female"

Un vettore può essere trasformato in fattore con la funzione as.factor:

> sexfactor <- as.factor(sexvector)

> sexfactor

[1] male male female male female female

Levels: female male

I fattori sono per forza delle categorie, dunque, sono sempre dei testi, character, ragione per cui non troviamo le virgolette; inoltre vengono indicati i livelli (nel caso del sexfactor saranno female e male).

Posso chiedere se un oggetto è un fattore con la funzione is.factor:

> is.factor (sexvector)

[1] FALSE

> is.factor(sexfactor)

[1] TRUE

Se chiedo str (sexfactor) ottengo una interessante info per alcuni precisi scopi; ad esempio, voglio trasformare li valori delle categorie in numeri; si creano vettori numerici a cui corrispondono i vari livelli.

> str(sexfactor)

Factor w/ 2 levels "female","male": 2 2 1 2 1 1

Di default l’ordine dei numeri è alfabetico f viene prima di m quindi female hanno numero 1 e male 2. Se chiedo as.numeric (sexfacotr) mi vengono restituiti i valori numerici dei livelli.

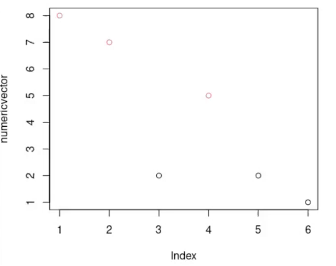
> as.numeric(sexfactor)

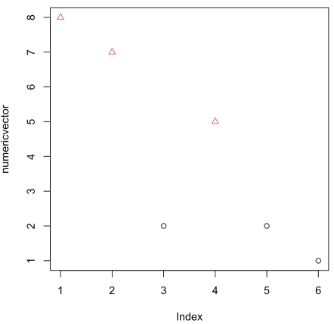
[1] 2 2 1 2 1 1

Se ho un vettore numerico corrispondente ai valori delle diverse categorie posso distinguere le categorie per colori:

> numericvector<- c (8,7,2,5,2,1)

> plot (numericvector, col = as.numeric(sexfactor))

Posso cambiare anche i colori del plot se sono autistico, i colori possono essere chiamati per nome o per numero (lui li cerca su google R color cheatsheet).

Posso cambiare anche la forma dei pallini se sono ancora più autistico mediante la funzione pch, anche in questo caso per distinguere i pallini delle categorie pch = as.numeric(sexfactor) come facevo per i colori. (lui non sa come farlo e lo cerca su google).

> plot (numericvector, col = as.numeric(sexfactor), pch = as.numeric(sexfactor))

È inutile la struttura dati del fattore? Sì XD. Però tante funzioni possono creare come output un elemento che ha nella sua struttura dati dei fattori.

Ad esempio: costruiamo un dataframe mydataframe i cui valori saranno quelli del nostro numericvector e le classi corrisponderanno al sexfactor e sexvector. Chiediamo poi la struttura mediante str(mydf)

mydf <- data.frame (value = numericvector, class = sexfactor, class2 = sexvector)

> mydf

value class class2

1 8 male male

2 7 male male

3 2 female female

4 5 male male

5 2 female female

6 1 female female

> str(my.df)

'data.frame': 6 obs. of 3 variables:

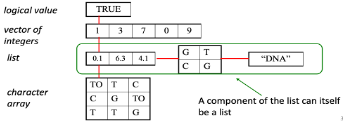
$ value : num 8 7 2 5 2 1

$ class : Factor w/ 2 levels "female","male": 2 2 1 2 1 1

$ class2: chr "male" "male" "female" "male" ...

La prima classe in questo caso è un fattore, se quindi si vogliono fare operazioni su quella colonna bisogna tenere in considerazione si tratti di un fattore e non un vettore, altrimenti poi quando fai le operazioni le cose non tornano e bestemmi.

# LISTE

Strutture molto usate perché versatili, varie funzioni che usano liste permettono di velocizzare il calcolo. È uno degli oggetti di R più generici che c’è, i data frame infatti sono dei tipi di lista. Le liste rappresentano un set di oggetti ordinati dette componenti, esse possono non essere dello stesso tipo, ad esempio la prima componente piò essere un dataframe, la seconda un vettore etc; quindi, liste sono set eterogenei di oggetti. Gli elementi all’interno possono essere a loro volta una lista.

Si costruisce con la funzione list, i suoi componenti sono sempre numerarti, ma si può attribuire un nome ai componenti. Le doppie quadrate si riferiscono al il numero della componente e permettono di distinguerli da oggetti numerici.

> mylist <- list (value = numericvector, class = sexfactor, class2 = sexvector)

> mylist

$value

[1] 8 7 2 5 2 1

$class

[1] male male female male female female

Levels: female male

$class2

[1] "male" "male" "female" "male" "female" "female"

> str(mylist)

List of 3

$ value : num [1:6] 8 7 2 5 2 1

$ class : Factor w/ 2 levels "female","male": 2 2 1 2 1 1

$ class2: chr [1:6] "male" "male" "female" "male" ...

Se non avessi dato dei nomi alle componenti allora esse verranno identificate da numeri tra [[]]:

mylist2 <- list (numericvector, sexfactor, sexvector)

> mylist2

[[1]]

[1] 8 7 2 5 2 1

[[2]]

[1] male male female male female female

Levels: female male

[[3]]

[1] "male" "male" "female" "male" "female" "female"

# Come accedere alla lista:

Alla lista si può accedere in varie maniere:

1. Via numeric index

Uso le doppie parentesi quadre per richiamare l’argomento delle componenti:

> mylist2 [[1]]

[1] 8 7 2 5 2 1

> class (mylist2[[1]])

[1] "numeric"

NB. attenzione differenza [] e [[]]: la differenza è che con le singole parentesi io non voglio il valore della componente, ma voglio quella specifica componente; dato che le componenti esistono solo dentro una lista faccio un subsetting della lista. Quindi basta togliere una parentesi e creo una nuova lista formata da una componete, se chiedo la classe di [2] dell’output mi dice che è una lista, mentre la classe di [[2]] è un carattere. Se chiedo le classi noto pene la differenza, nel caso delle doppie parentesi ottengo la classe dell’argomento, in questo caso un numero, nel caso delle singole parentesi ottengo la classe della componente e quindi sarà una lista.

> mylist2[1]

[[1]]

[1] 8 7 2 5 2 1

> class (mylist2[1])

[1] "list"

La differenza è cosa chiedo mi venga resitutio dal comano: con [[]] chiedo l’argomento, con [] chiedo la componente.

1. Via compoent name

Chiamo mediante il nome e utilizzo il dollaro

> mylist$class2

[1] "male" "male" "female" "male" "female" "female"

1. Via “character”

oppure posso fare la chiamata del componente stesso utilizzando il suo nome in character

> mylist[[“class2”]]

[1] "male" "male" "female" "male" "female" "female"

# Operazioni:

Operazioni con le liste sono sempre le stesse, sempre uguali.

* Aggiungere o sovrascrivere elementi

Se si vuole aggiungere un elemento ad una lista posso banalmente imporlo, ad esempio se ho due elementi decido quale deve essere il terzo:

> li<-list(0.6798,

paste(c(rep("A",4),rep("T",4)),

collapse=""))

> li[3] <- list(TRUE)

> there

[[1]]

[1] 0.6798

[[2]]

[1] "AAAATTTT"

[[3]]

[1] TRUE

Stessa cosa vale anche per sovrascrivere un elemento.

* Concatenare, unire liste:

Le liste si uniscono allo stesso modo dei vettori, se ho due liste con diverse componenti e le voglio unire uso c come se unissi dei vettori.

> li1 <- list (TRUE,2)

> li2 <- list ("foo")

> li3<-list ( c(1,2,3), c("T","A","C"))

> li123 <- c (li1,li2,li3)

> li123

[[1]]

[1] TRUE

[[2]]

[1] 2

[[3]]

[1] "foo"

[[4]]

[1] 1 2 3

[[5]]

[1] "T" "A" "C"

# PCA

Quando abbiamo a che fare con un vasto numero di dati e non sappiamo come organizzarli in modo semplice, un metodo è l’analisi delle componenti principali che è una trasformazione lineare dei dati che porta ad avere i dati plottati rispetto alle loro componenti più importanti del dataset. I dati possono essere scomposti, analizzati mediante la ricerca delle componenti principali che chiamiamo riassunti dei dati.

Aumentando il numero delle componenti si suddividono meglio i dati che risultano più fedeli alla loro distribuzione originale, ma aumenta la complessità; il nostro scopo però è quello di semplificare i dati senza perdere però completamente il loro significato, quindi, riduciamo al minor numero di componenti possibili che permettono di rappresentare in maniera utile il dato.

Esempio immagine delle foglie: le foglie sono distinte da colore, forma etc., ma ad esempio all’interno della categoria del colore possiamo ritrovare infinite sfumature, dobbiamo scegliere un numero di colori utile alla rappresentazione significativa e allo stesso tempo semplificata del dato. Se facciamo una media di colore e forma si ottiene un risultato che non descrive nulla dei dati di partenza; avendo dei campioni ognuno dei quali descritto da determinati dati per ogni componente possiamo riconoscere le principali componenti; man mano che aumentiamo i componenti aumenta la fedeltà all’originale. Con la PCA si vuole ridurre il numero di componenti comunque riuscendo ad avere una rappresentazione accettabile dei dati originali.

# PCA su dataset Iris:

Esempio, uso il dataset “iris” (pacchetto – dataset):

* Inserisco il dataset e guardo come è struturato

Per richiamare questo dataset già presente su R basta scrivere

> data (iris)

E scrivendo iris comparirà l’intero dataset, per vederlo in maniera semplificata possiamo scrivere

> str(iris)

'data.frame': 150 obs. of 5 variables:

$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...

$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...

$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...

$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...

$ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

In questo modo possiamo sapere che iris è un pacchetto di 150 oggetti e 5 variabili ovvero lunghezza delle foglie, dimensione delle foglie, lunghezza dei petali, larghezza dei petali e specie che sono dei factor, per sapere quali livelli sono presenti nel fattore delle specie basta richiamare levels:

> levels(iris$Species)

[1] "setosa" "versicolor" "virginica"

Indica siano presenti questi tre tipi di specie.

* Togliere il nome delle specie

Togliamo l’ultima colonna perché non vi sono valori numerici utili alla analisi delle componenti principali. Ad esempio, se vogliamo are la media delle colonne con il dataset avente componenti character R non prosegue.

> colMeans(iris)

Error in colMeans(iris) : 'x' must be numeric

Quindi tolgo la quarta colonna:

> iris\_clean <- iris[, 1:4]

> str(iris\_clean)

'data.frame': 150 obs. of 4 variables:

$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...

$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...

$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...

$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...

Possiamo creare un oggetto a parte per le specie:

species <- iris$Species

* Come costruisco una PCA

Basta usare il semplice comando:

> pca\_iris <- prcomp (iris\_clean, scale = TRUE)

Dove scale = TRUE permette la normalizzazione dei dati; la PCA infatti è un sistema di trasformazione lineare per capire quale elemento è più importante di alti per quanrto riguarda la variabilità dei dati; ad esempio, una componente è in scala milioni e l’altra in scala unità ed in trasformazioni lineari ciò ha un peso, mediante la normalizzazione mettiamo tutti i dati sulla stessa scala; quindi fondamentale è la normalizzazione (che di default avviene).

Se chiediamo la classe di pca\_iris otteniamo come risultato “prcomp”, oggetto che ha una struttura sua detto “prcomp”,

> class(pca\_iris)

[1] "prcomp"

Andando a chiedere la sua struttura con str(pca\_iris) ci viene detto essere una lista.

> str(pca\_iris)

List of 5

$ sdev : num [1:4] 1.708 0.956 0.383 0.144

$ rotation: num [1:4, 1:4] 0.521 -0.269 0.58 0.565 -0.377 ...

..- attr(\*, "dimnames")=List of 2

.. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"

.. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"

$ center : Named num [1:4] 5.84 3.06 3.76 1.2

..- attr(\*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"

$ scale : Named num [1:4] 0.828 0.436 1.765 0.762

..- attr(\*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"

$ x : num [1:150, 1:4] -2.26 -2.07 -2.36 -2.29 -2.38 ...

..- attr(\*, "dimnames")=List of 2

.. ..$ : NULL

.. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"

- attr(\*, "class")= chr "prcomp"

Ci viene così data la deviazione standard (=sdev), rotazione (= rotation), il centro (= center), la scala (= scale) ed il valore x (= x). è una lista ma l’output sarà sempre questo: ci saranno 5 elementi della lista che si chiameranno sempre sdev, rotation, center, scale ed x. Questo permette di ottenere sempre una medesima struttura quando si vuole analizzare le componenti principali usando questa funzione. Se l’output fosse semplicemente una lista non sarebbe sempre uguale la sua lunghezza, il nome delle componenti etc.

* Summary per riassumere i risultati:

Il summary è una funzione che permette di riassumere i risultati.

> summary\_pca\_iris <- summary (pca\_iris)

> summary\_pca\_iris

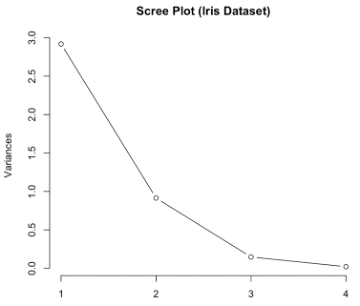
Importance of components:

PC1 PC2 PC3 PC4

Standard deviation 1.7084 0.9560 0.38309 0.14393

Proportion of Variance 0.7296 0.2285 0.03669 0.00518

Cumulative Proportion 0.7296 0.9581 0.99482 1.00000

Gli elementi sono stati scomposti in elementi principali in questo caso chiamati PC1, PC2, PC3, PC4, ognuno con le seguenti proprietà: deviazione standard, proporzione della variane e proporzione della varianza in senso cumulativo. Guardando alla PC1 la proprozione delle variande è di circa 0.73, questo vuol dire che questa singola componente riassume il 73% della varianza dei dati usati da questa analisi. Quindi semplicemente guardando questa proporzione ho riassunto molte info, infatti la variabilità dei dati può essere anche chiamata informazione dei dati. Prendendo la seconda PC2 la proporzione della varianza, quindi l’informazione, variabilità dei dati, è del 23% circa, la terza PC3 ha solo il 3% di informazione e la PCA4 ancora meno. La terza riga ovvero la proporzione cumulativa fa la somma della proporzione della varianza delle varie componenti e quindi ci dice che con solo le prime due componenti principali PC1 e PC2 abbiamo descritto il circa 96% dei dati; se uso tre componenti ed includo anche PC3 descrivo fino al 99% dei dati.

* Plot della varianza in funzione delle varie principal compoents

Possiamo quindi plottare l’oggetto, abbiamo appena visto che pca\_iris è una lista, come è possibile quindi plottarlo? Questo perché è una lista “prcomp” e chi ha scritto questa tipologia di lista ha anche scritto una metodologia di plotting partendo da questo oggetto lista. Scrivo quindi:

> plot (pca\_iris, type = "l", main = "Scree Plot (Iris Dataset)")

Si ottiene una semplice rappresentazione delle componenti principali e la varianza delle info principali.

* Visualizzazione dei componenti secondo le principal compoents

Nel pca\_iris nella componente x abbiamo un fattore avente i 150 elementi come righe e le 4 diverse PC come colonne.

> pca\_iris$x

PC1 PC2 PC3 PC4

[1,] -2.25714118 -0.478423832 0.127279624 0.024087508

[2,] -2.07401302 0.671882687 0.233825517 0.102662845

[3,] -2.35633511 0.340766425 -0.044053900 0.028282305

Etc.

I valori riportati sono le coordinate cartesiane nello spazio creati con questa analisi PCA.

Per vedere meglio ciò possiamo creare un oggetto ovvero un dataframe con questi elementi del fattore x e ci attacchiamo ad essi le rispettive specie; creato il dataframe andiamo a guardare la sua struttura e vado a plottare il tutto.

> scores <- data.frame (pca\_iris$x, Species = species)

> str(scores)

'data.frame': 150 obs. of 5 variables:

$ PC1 : num -2.26 -2.07 -2.36 -2.29 -2.38 ...

$ PC2 : num -0.478 0.672 0.341 0.595 -0.645 ...

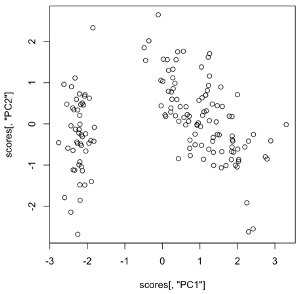
$ PC3 : num 0.1273 0.2338 -0.0441 -0.091 -0.0157 ...

$ PC4 : num 0.0241 0.1027 0.0283 -0.0657 -0.0358 ...

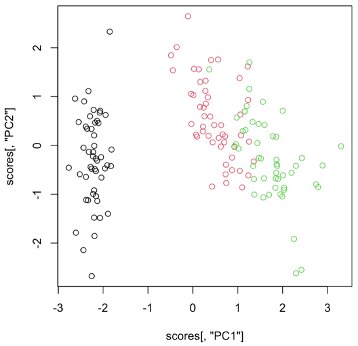
$ Species: Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

Plotto così:

> plot (x = scores[,"PC1"], y = scores[,"PC2"])

Ottengo un grafico che rappresenta nello spazio riassuntivo le posizioni di ogni singola osservazione delle 150 piante. Si vede già ad occhio due grandi gruppi di campioni.

Inizialmente avevo 4 osservazioni, larghezza e lunghezza di foglie e petali, se volevo dividere in base a queste osservazioni i dati avrei dovuto fare dei plot per ognuna di queste variabili: su un lato lunghezza di foglie dall’altro l’lunghezza dei petali, plottare e vedere se i campioni vengono raggruppati. In questo caso avrei dovuto fare questi plot per ogni combinazione di osservazioni (che in questo caso sono 4), ma se avessi moltissime osservazioni sarebbe un puttanaio. In una espressione genica differenziale non si può andare gene per gene a vedere se questi sono in grado di dividermi i campioni in gruppi; quindi, voglio riassumere le informazioni per individuare la presenza di campioni da un lato e dall’altra.



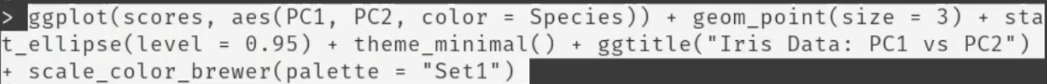
Per rendere più comprensibile il prot posso distinguere per colori

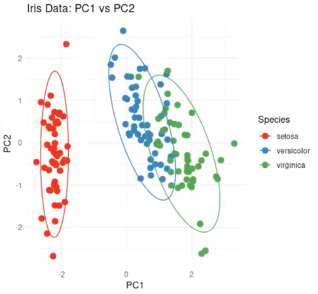
> plot (x = scores[,"PC1"], y = scores[,"PC2"], col = as.numeric(scores$Species))

* Plottare con ggplot2

È possibile plottare con un pacchetto R aggiuntivo chiamato ggplot2 (ce la spiegherà meglio la Giulia nelle prossime lezioni).

> library (ggplot2)

Il grafico ha la legenda, varie colorazioni, campioni suddivisi sulla base delle prime due componenti principali.

La PC3 aveva poca informazione, il 3%, ma se avesse avuto maggiore informazione? Allora avrei dovuto plottare in 3D (e ti trovi u tutorial su google); se ne avessi 4, 5 o 6 allora si può fare per coppie di componenti principali con maggiore informazione.

In questo caso i campioni sono divisi in due macrogruppi: la setosa rispetto alle altre due specie rispetto alle osservazioni fatte su lunghezza di petali e foglie. Se trasliamo questo concetto su un esempio biologico potremmo aver osservato, anziché la lunghezza delle foglie, l’espressione dei geni e che i nostri elementi di suddivisione non sono la specie di pianta, ma ad esempio maschi e femmine, trattati e non trattati, responder o non-respondre ad un farmaco. Osservando dei geni posso capire se la persona è maschio o femmina, se una persona risponde o meno ad un farmaco; insomma, osservando i parametri posso distinguere i campioni. Permette a partire da una enorme quantità di informazione di riuscire a capire da ogni info presente se è possibile distinguere i campioni sulla base di osservazioni. In questo caso ci sono due componenti ovvero PC1 e PC2 che sono più importanti rispetto alle altre nel permettere la divisione in gruppi del campione.

(Nel documento tutorial che ha caricato ci sono anche le info per affrontare un altro dataset R possibile da analizzare in PCA sulle performance accademiche, infine c’è una conclusione di confronto dei due dataset).

La PCA è solo uno dei metodi di riduzione della dimensione della complessità di un dataset, altri sono MDS, t-SNE etc. il principio è simile, cambia il metodo matematico alla base dei calcoli. Permette una descrizione iniziale dei campioni, utile a capire chi va confrontato con chi. Sapere che due dati sono simili mi predispone nel momento in cui faccio t-test, ad esempio, di aspettarmi che, in questo caso, le specie virginica e versicolor sono più simili tra loro rispetto a setosa.

Inoltre, è possibile mettere la varianza delle componenti principali, ad esempio PC1 ha il 90% dell’informazione quindi divide molto il campione a differenza della seconda che aveva circa il 20%, quindi elementi più distanti sull’asse della X hanno molta più importanza di dati distanti sull’asse Y perché l’info che c’è lungo l’asse Y è molto più piccola rispetto a quella sull’asse X. Quindi c’è una varianza all’interno di ciascun gruppo, i dati non sono tutti concentrati, ma la loro distanza in X è più importante di pella in Y perché c’è più varianza, più info sulle componenti principali.

Immagine che contiene testo, schermata, Carattere, design

Il contenuto generato dall'IA potrebbe non essere corretto.  
Lezione 12 maggio (FaBiO tOcCa i BaMbInI)  
Un esempio di analisi di data set  
Molti pacchetti di R hanno dentro un data set per far vedere loro stessi come funziona il loro pacchetto. Se con il mio data set non funziona ad esempio so che posso aver sbagliato io in qualche impostazione, perché ho la prova che il loro pacchetto funzioni.  
Nel momento i cui carichiamo il nostro pacchetto dentro dovrebbe esserci il data set da lor fornito. Se vogliamo pasticciare coi dati è meglio salvarne una copia, soprattutto se il data set è molto grande.  
Nel momento in cui abbiamo il data set osserviamo la sua struttura, e vediamo nel caso di questo pacchetto che il data set lung è un data frame  
Immagine che contiene testo, schermata, Carattere, ricevuta

Il contenuto generato dall'IA potrebbe non essere corretto.  
Vediamo che gli elementi sono di classe e tipo diverso, abbiamo cose numeriche e cose che sono fattori. Vediamo che alcuni elementi contengono anche NA.   
Le variabile di questo data set all’interno di questo pacchetto sono:   
Quando le persone sono malte vi è un periodo di tempo dove le persone sono trattate a seconda del tipo di tumore, e poi vi è un tempo di osservazione su come sta andando rispetto alla cura, puo succedere che a una persona a un determinato tempo puo essere viva o morta. Vuol dire che nello studio le persone non sono osservate per sempre, vi è un tempo di studio determinato. Quindi una persona operata e tratta e alla fine dello studio puo essere ancora viva, o morta ed uscita dallo studio, oppure uscita per altre situazioni come trasferita o morta per altre cause (serva a contestualizzare l’ambiente su cui è stato fatto questo data frame, ovvero uno studio di persone col cancro in seguito a trattamento).  
Quando abbiamo molti numeri possiamo fare un summary() del data set, e ci dira per ogni variabile la sua distribuzione. Ci da una prima indicazione di com’è la distribuzione dati dei variabili, e ci dice quanti sono gli NA per ogni variabile per sapere quale variabile è piu debole che rende piu difficile le comparazioni tra Immagine che contiene testo, schermata, diagramma, Rettangolo

Il contenuto generato dall'IA potrebbe non essere corretto.variabili.

Immagine che contiene testo, schermata, Carattere, bianco

Il contenuto generato dall'IA potrebbe non essere corretto.  
La qualità dei dati è fondamentale. Se io faccio un plot in cui prendo in considerazione una variabile come sesso che è codificato 1 e 2, dovremo operare per rendere il plot visibile in modo migliore facendo chimare un asse maschio e l’altro donna. Puo essere utile fare all’inizio cambiare i label da numerici a carattere o viceversa per renderci piu facili le operazioni successive.   
Per farlo va fatto un cambiamento di status il comando è   
Immagine che contiene testo, Carattere, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.  
  
I box plot sono utili per valutare distribuzioni, mode, variazioni, valori anomili di variabili numeriche in diversi gruppi.   
Ad esempio l’eta puo essere un valore che cambia il relazione al sesso, con il boxplot lo possiamo visaulizzare. Il comando del blox plot vuole una formula che si ail valore delle y rispetto al gruppo (Y verso x) ad esempio   
  
Per veder ese vie è una differenza significativa tra le due classi si puo testare con un t.test (possiamo copia incollare age – sex, data=lung perche anche questo ha formula)  
  
Immagine che contiene testo, ricevuta, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.  
Non siamo signifcativi a livello di differenza. Noi dobbiamo decidere se la differenza è o meno rilevante, perche possiamo dire di scartare i valori estremi e vedere se è significativa la differenza. L’analisi ci puo dire che vi è una differenza significativa, e questa è pari a 4 o 3 ad esempio su tot insieme di dati, a quel punto siamo noi a dover dire se è rilevante o meno la differenza rispetto alla distribuzione dei dati. Abbiamo visto quindi le variabili categoriche rispetto alle variabili numeriche.  
  
Andiamo adesso a vedere i valori numerici rispetto ad altri variabili numeriche, per vedere questa cosa si vanno a vedere le correlazioni, quindi quando una variabile ha una tendenza ad a vedere cosa succede alle altre variabili. Vediamo correlazioni semplici, in quante a volte i dati possono esser interpretati in modo piu complesso rispetto alle correlazioni. Nel senso che le variabili possono essere manipolate prima di fare le correlazioni, o si possono fare correlazioni complesse.  
Ad esempio creiamo una variabile 1   
°  
Facciamo poi l’istogramma di questa variabile  
  
Facciamo poi una seconda variabile  


E facciamo anche l’istogramma di questa  


Facciamo poi un plot della var1 rispetto alla var 2  
  
Otteniamo un andamento di tipo esponenziale.   
Immagine che contiene testo, schermata, linea, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.  
Vedendo un andamento di questo tipo sappiamo che cercare una correlazione della variabile 1 rispetto alla variabile potrebbe aver senso ma avrebbe senso fare il log variabile 2 rispetto alla 1, giusto per fare una prova.  
Calcoliamo la correlazione con un test  
  
Immagine che contiene testo, Carattere, ricevuta, algebra

Il contenuto generato dall'IA potrebbe non essere corretto.

Il sistema ci dive che è positivo perche quando cresce una crece anche l’altra. Adesso proviamo pero con il logaritmo  
  
Immagine che contiene testo, Carattere, schermata, algebra

Il contenuto generato dall'IA potrebbe non essere corretto.  
In questo caso abbiamo una correlazione positiva pero è piu giusto fare cosi dal punto di vista statistico perche noi cerchiamo correlazioni lineari, quindi c’è una fase in cui noi osserviamo due variabile con un plot in modo lineare per poi vedere la correlazione con la funzione, ma esisterebbero i modelli lineari che sono molto piu complessi ma che noi non affronteremo. E sono una serie di strumenti per vedere se una variabile cambia in funzione dell’altra. I modelli lineari servono molto nel caso in cui ci siano delle variabili di disturbo.  
Come visto prima abbiamo opzioni nelle funzioni che ci permettono di gestire le NA, anche con i sisestemi di correlaizoni abbiamo sistemi sistemi simili, nella correlazione normale c’è un opzione per usare le osservazioni complete.  
Se andiamo nell’help di ?cor, in use c’è la possibilità di scegliere di usare le correlazioni complete, e Immagine che contiene testo, Carattere, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.ci sono poi altre opzioni.  
  
  
  
  
  
  
Ci sono diveri modi per vedere se gli elementi del data frame lung sono numerici  
Immagine che contiene testo, Carattere, schermata, bianco

Il contenuto generato dall'IA potrebbe non essere corretto.  
I data frame sono liste bidimensione, e se gli voglio applicare una funzione come ad esempio il numero delle righe  
tramite cui vedo che tutti le variabili hanno la stessa lungheza. Posso chiedere anceh appunto se le variabili sono numeriche  
Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.  
Abbiamo quindi creato un indice di true o false, da cui possono ottenere un secondo data frame con solo queste informazioni, ottenendo un sub set delle variabili numeriche di lung  
  
Adesso facciamo il nostro correleationi delle informazioni complete da cui otteniamo una matrice quadrata   
Immagine che contiene testo, Carattere, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.  
E’ una matrice di correlazione dove tutti gli elementi contro tutti. Ha la diagonale 1 che sono le corellazioni tra la stessa variabile (tipo time contro time ecc…). Quando noi poi vogliamo fare un plot della correlazione degli elementi  
Immagine che contiene schermata, testo, diagramma, design

Il contenuto generato dall'IA potrebbe non essere corretto.  
Da questo grafico possiamo vedere le correlazioni che possono essere positive o negative. Ci sono vari tipi di corplot con diverse modalità, si possono mettere p-value per vedere la significatività, questo perche nelle correlazioni c’è da tener conto del valore stesso e della pvalue associata.  
Immagine che contiene testo, schermata, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.  
  
Otteniamo un overview delle correlazioni da cui poi possiamo andare af are un’analisi piu specifica.  
Ad esempio possiamo farlo della perdita del peso rispetto Karnofsky  
Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.  
Vediamo che vi è una bassa correlazione e ha un pvalue significativo. Poi volendo si puo anche fare un plot di base Immagine che contiene testo, schermata, linea, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.  
  
Vediamo appunto che i pochi non sono ben correlati per questo abbiamo un valore basso. Si plotta per vedere se vediamo un andamaneto dei punti uno rispetto all’altro che magari ci puo essere sfuggito.  
  
Volendo possiamo rendere delle variabili numeriche categoriche, ad esempio l’eta, la categorizzo in diverse fascie. Anche la perdita di peso ad esempio.  
Immagine che contiene testo, Carattere, schermata, algebra

Il contenuto generato dall'IA potrebbe non essere corretto.  
A questo punto abbiamo due variabili categoriche che possiamo confrontare  
Immagine che contiene testo, schermata, Carattere, ricevuta

Il contenuto generato dall'IA potrebbe non essere corretto.   
In cui è stato fatto anche un test del chi-quadro per vedere se le due variabili cateogirche sono correlate tra lor. Quindi con le variabili numeriche ci facciamo le correlazioni, con le variabili categoriche ci facciamo test del chi-quadro o anova.  
Possiamo rappresentarlo con il plot a mosaico.  
Immagine che contiene schermata, testo, Rettangolo, design

Il contenuto generato dall'IA potrebbe non essere corretto.

A questo tipo di data set è possibile fare un’analisi di sopravvivenza, ovvero un data set dove vi è tempo come categoria e l’evento di esser vivo o morto a quel determinato tempo.  
In questo esempio abbiamo gia il pacchetto dentro survival senno dovremoo caricare la library.  
Immagine che contiene testo, schermata, linea, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

Appunto mettiamo a confronto in surv\_obj il tempo con lo stato. Che è poi è stato messo assieme al sesso in fit per vedere in relazione l’evento e il tempo con la variabile sesso. Per poi plottare tutta l’analis con la formula vista sopra.  
Per ottenere una statistica che supporta l’anlisi visiva, vi è la funzione di questo pacchetto chiamata survdiff, a cui chiediamo se vi è una differenza tra le curve uomo donna.  
Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.  
Da cui otteniam un pvalue che ci dice che c’è differenza nella sopravvivenza tra uomo e donna. Posso quindi pensare di tornare indietro e considerare i dati del dataset in modo diviso tra uomo e donna e fare le correlazione in modo separato in quanto vi è una differenza di sopravvivenza tra i due sessi.  
  
Possiamo fare anche la PCA per visualizzare i campioni a livello iniziale. Da cui vediamo che non si evince una differenza tra uomo e donna visualizzando i dati in questo modo. Noi possiamo analizzare i dati e l’analisi non suggerisce quella correlazione è vero per quanto riguarda l’analisi singola, ma possiamo sulla base di questo reinterpretare poi i dati a nostro modo.  
  
Immagine che contiene testo, schermata, software

Il contenuto generato dall'IA potrebbe non essere corretto.