Програмиране за биолози. Анализ и визуализация на научни данни

Веселин Баев

Университетско издателство
"Паисий Хилендарски"

Учебникът е предназначен за студенти от специалност Биоинформатика, както и други биологични специалности към Биологическия факултет на ПУ „Паисий Хилендаркси“. Учебникът има за цел да разясни основните принципи на програмния език R, във връзка с анализа и визуализацията на научни данни в биологията.

2022
Онлайн версия на учебника: 1.0
(Източник на изображенията в корицата Freepik)

ISBN 978-619-202-779-7

Съдържание:

Глава 1

Защо да изучаваме R?

Добре дошли в програмирането за биолози! В тази книга ще разгледаме програмният език R, който е един от най-популярните платформи за анализ и визуализация на научни данни, налични до момента. Това е безплатен език с отворен код, достъпен за операционни системи Windows, Mac OS X и Linux. Този учебник ще ви помогне да придобиете основните умения, необходими да овладеете този изчерпателен език и да го приложите ефективно към вашите собствени данни. R се използва изключително много в науката, в това число и в биологията, тъй като благодарение на него лесно и бързо може да анализиране вашите данни и да генерирате нужните графики за визуализацията им. Освен това в последните години са натрупани огромен брой готови пакети за R, както и софтуерни продукти, специфични за обработката на биологични данни. Така R в днешно време се превръща в неразделна част от биоинформатиката и научните анализи.

В началото на учебника ще се запознаем с основите на инсталирането на софтуера, ще се научим да навигираме в интерфейса на RStudio, както и импортирането на данни и тяхното преформатиране в удобен формат за последващ анализ. След като сте запознати с R-интерфейса, следващото предизвикателство е да въведем данните си в програмата. В днешния богат на информация свят данните могат да идват от много източници и в много формати.

След това ще се запознаем със същината на R. Ще започнем с преглед на структурите от данни, които R използва за съхраняване на данни и ще продължим с R и функциите, които го правят толкова мощна платформа за съвременен анализ на научни данни. Да не забравяме също, че и една от силните страни на R в анализа на данни е статистиката - чрез готови функции и пакети, лесно ще може да обработите вашите данни дори и да нямате задълбочени познания в статистическите методи.

Няма да пропуснем и много важна част от научните изследвания – визуализацията на научни данни, които са неотменна част от графичното

представяне на резултатите ви. Все пак знаем, че човек много по-лесно възприема и обобщава визуална информация, отколкото текстова или таблична такава. Тук R също няма да ни подведе, той притежава уникални модули за визуализация на данни, с които можем лесно да създадем високо персонализирани и красиви графики и диаграми.

Затова дали сте молекулярен биолог, микробиолог, биоинформатик, еколог, еволюционист или генетик, инвестирайте своето време в R и няма да сгрешите, защото той ще ви се отплати във вашата работа!

Какво е R и RStudio?

Този учебник ще запознае читателите с RStudio Integrated Development Environment (IDE) за използване и програмиране на R, широко използван език за анализ на данни с отворен код. RStudio (https://www.rstudio.com) е отделен проект с отворен код, който обединява много мощни инструменти за писане на код в интуитивен, лесен за научаване интерфейс. RStudio работи във всички основни операционни системи (Windows, Mac, Linux), както и чрез уеб браузър (използвайки инсталация на отдалечен сървър). Тази част ще е полезна за по-новите потребители на R, студенти и учени, които искат да проучат интерфейса на тази платформа, за да извлекат максимума от R, както и за дългогодишни потребители на R, които търсят по-модерна среда за работа.

Ще започнем с бърз преглед на R и какво е IDE, преди да се потопим в RStudio.

Какво е R? R език с отворен код за анализ на данни и графики. Софтуерният проект R е стартиран за първи път от Робърт Джентълман и Рос Ихака. Езикът е силно повлиян от езика S, който първоначално е разработен в Bell Laboratories от Джон Чембърс и колеги. Оттогава, под ръководството на основния екип за разработка на R, той се превръща в *ligua franca* за статистически изчисления и визуализация в много дисциплини на академичните среди и различни индустрии.

R е много повече от просто базов език. Той има световна система за хранилище, Comprehensive R Archive Network (CRAN) (http://cran.r-project.org) за пакети с добавки, предоставени от потребителите, които допълват базовата дистрибуция. Към 2022 г. има повече от 19000 готови пакета, хоствани в CRAN и още много на други сайтове. Като цяло R понастоящем има функционалност за справяне с огромен набор от проблеми и все още има място за развитие, чрез нови пакети разработвани от потребителите.

R е проектиран около своя основен скриптов език, но също така позволява интеграция с код, написан на Python, C, C++, Java и т.н., за интензивни изчислителни задачи или за използване на инструменти, предоставени за други езици.

Какво е IDE? R, подобно на други езици за програмиране, се разширява чрез функции, написани от потребителя. Интегрирана среда за разработка (integrated development environment, IDE) като RStudio, е предназначена да улесни такава работа, както и всекидневната работа на работещият с R. Освен това, за разлика от много други софтуерни пакети за анализ на данни, в които се използва графичен потребителски интерфейс, типичният потребител взаимодейства с R предимно чрез командния ред. Тогава IDE за R трябва също да включва средство за интерактивно издаване на команди. R не е уникален в това отношение и IDE за интерактивните езици за програмиране често включват функции като:

• Конзола за издаване на команди;

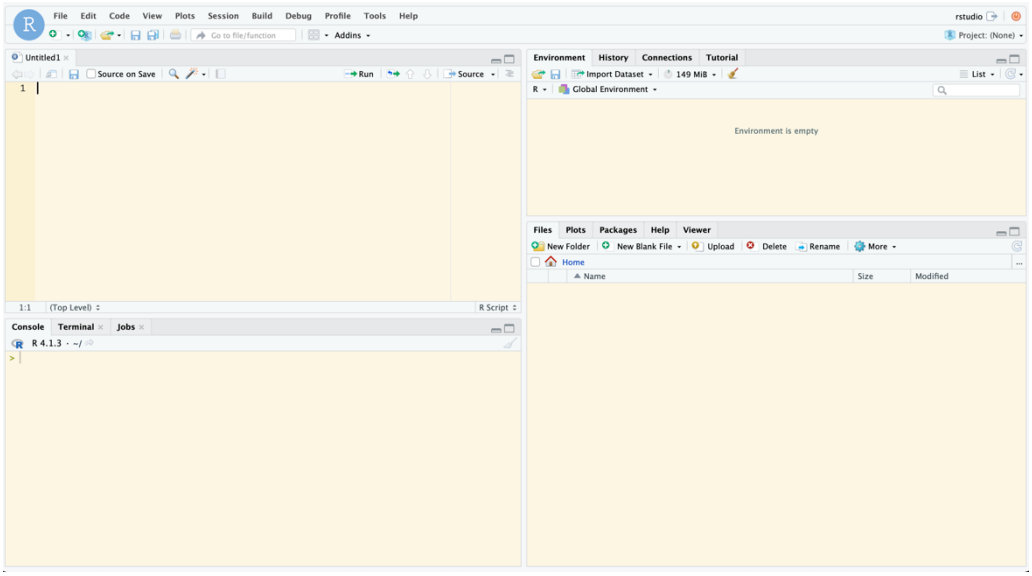
• Редактор на изходен код, който позволява процеса на самото програмиране. Такива редактори съществуват от известно време и очакванията към тях вече са доста взискателни. Типичният набор от техните функции включват: богат набор от клавишни комбинации; автоматично форматиране на изходния код, помощ със скоби, подчертаване на ключови думи; помощ със синтаксиса на езика; контекстна помощ; управление на проекти; помощ за отстраняване на грешки и т.н.

Какво е RStudio? Проектът RStudio (https://www.rstudio.com/products/rstudio/) предоставя повечето от желаните функции за IDE, което го прави средство за по-лесна и по-продуктивна работа и използване на R. Някои от акцентите и функционалностите са:

- Основните компоненти на IDE са добре интегрирани в оформление от четири панела (фиг. 1), което включва: конзола за интерактивна R сесия, редактор на изходен код, панел за организиране на файловете от проекта и допълнителни раздели за графики и организиране на по-малко централни компоненти.
- Редакторът на изходния код е богат на функции и е интегриран с конзолата.
- Създаването на различни проекти е бързо, а превключването между тях е още по-лесно.
- RStudio предоставя много удобни и лесни за използване административни инструменти за управление на пакети, работно пространство, файлове, графики и др.

- IDE е достъпна за трите основни операционни системи и може да се изпълнява през уеб браузър, ако е инсталирана на отдалечен сървър.
- Безплатна платформа с отворен код.

В този учебник ние ще използваме R през RStudio, защото това е много удобна среда/платформа, която ще ни помогне за по-доброто усвояване на езика и оптимизиране на нашата работа.



Фиг.1 Интерфейсът на RStudio и неговите основни панели за работа.

Интерфейсът на RStudio се състои от няколко основни компонента/панела, разположени под лентата с инструментите. Въпреки че това разположение може да се регулира и да се смени според предпочитанията, оформлението по подразбиране използва четири основни панела (Фиг.1):

• В горния ляв ъгъл е редакторът на изходния код, както и редактор на файлове с данни;

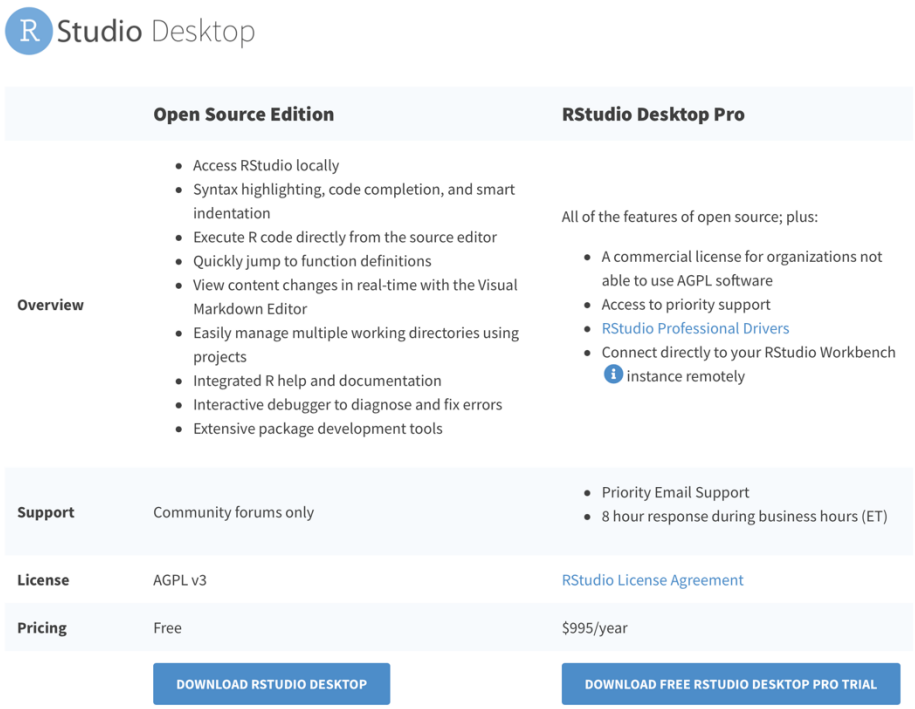
• В долния ляв ъгъл е конзолата за взаимодействие с R („Console”);

• В горния десен ъгъл има няколко подпанела като история на командите, както („History”) и таблица на обектите („Environment“);

• В долния десен ъгъл има също няколко важни подпанела като този за работа с файлове („Files”), плотове и графики („Plots”), налични пакети (“Packages”) и др.

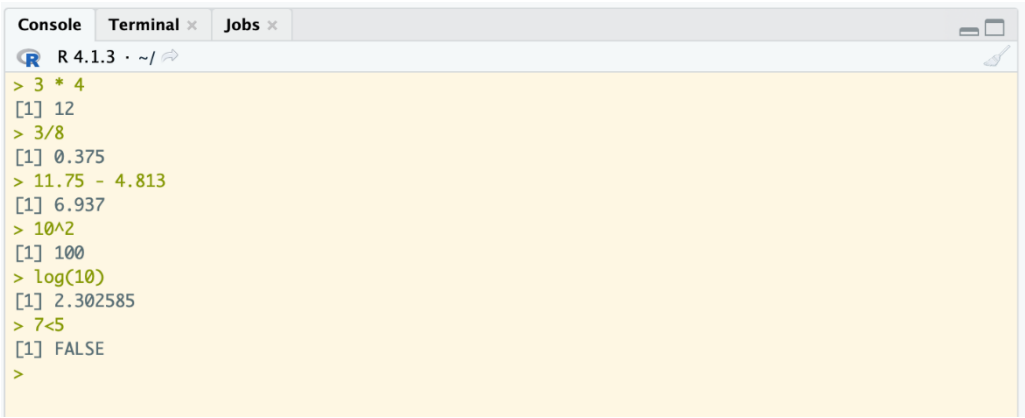
Потребителят може да промени разпределението на всеки от панелите по подразбиране и да добавя нови раздели в зависимост от нуждите. Всеки от панелите също може да бъде регулиран във връзка с неговата големина.

Можем да инсталираме десктоп версията на RStudio от https://www.rstudio.com/products/rstudio/, в зависимост от наша версия ОС. Платформата е налична за повечето ОС – Windows, MacOS, Linux (Фиг. 2).



Фиг.2 Сайтът на RStudio предлага сваляне на десктоп версията за редица ОС.

За да започнем да си взаимодействаме с R, можем да използваме панела с конзолата, където можем да задаваме команди, които да бъдат изпълнявани. За да започнем, нека използваме R като един огромен калкулатор ☺ и да изискаме някой елементарни математически изчисления да бъдат изпълнени (в конзолата виждаме, че редът започва със знака „>“, което означава, че R очаква нашата команда):

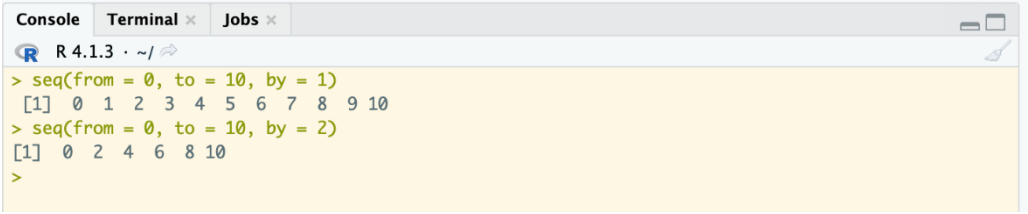


След всяка команда за изчисление натискаме клавиша за нов ред и R съобщава резултата от командата на нов ред. Основните математически операции са подобни на останалите езици за програмиране и се представят от символите: „*“ за умножение, „/“ за делене, „^“ за степен и т.н. Последният въпрос е „7 по-малко от 5?“. R разбира това правилно, с отговора „ГРЕШНО“. Знакът „по-малко от“ е известен като „логически оператор“. Други такива оператори включват == (равно/еднакво), != (не е еднакво), > (по-голямо), <= (по-малко или равно на) , >= (по-голямо или равно на), | (символът с вертикална линия) е логическо „или“; и & логическо „и“.

Всички изчисления по-горе дават само един отговор. Това е вид математика, с която вероятно сме свикнали. Но R може да отговори на няколко въпроса едновременно. Например можем да попитаме R: „Бихте ли ни дали целите числа

от 1 до 10 включително?“. Ще видим как, но преди това трябва да си отговорим на въпроса:

Какво са функциите в R? Ние до момента сме използвали функцията log(). Изискването от нас R да прави неща, обикновено изисква използване на т.нар. функции. R използва функции, за да прави всякакви неща и да ви връща информация. Всички функции в R си имат име, представляващо дума или комбинация от думи, които не съдържат интервали, последвани от отваряща скоба „(“ и затваряща скоба „)“. Вътре в тези скоби е информацията, която подаваме на функцията. Тази информация се нарича „аргументи“ на функцията. Аргументите се разделят със запетаи. И сега да се върнем към нашата задача - да поискаме R да ни даде числата от 1 до 10. За тази цел ще използваме функцията seq(). Функцията е seq(), а вътре в скобите има три аргумента, разделени с две запетаи (задължително). Първият аргумент е стойността, която трябва да се използва за началото на последователността, вторият е стойността, която трябва да се използва за край на последователността, а третият е размерът на стъпката:



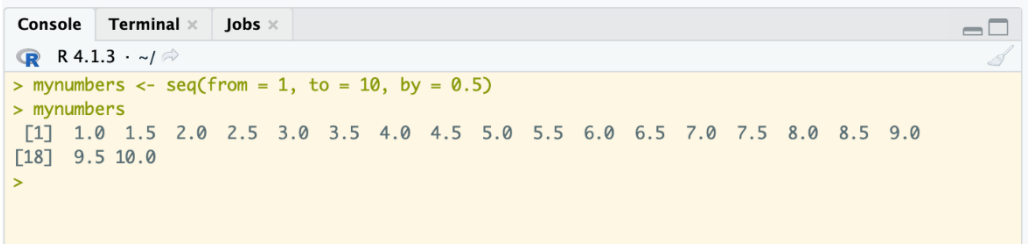
Тук на втория път променихме аргументите, като модифицирахме стъпката, така получихме числата от 1 до 10 със стъпка 2.

Всичко изглежда чудесно! Но досега R е отпечатал отговора на въпросите ни в конзолата. R не е запазил отговора някъде. В резултат на това не можем да направим нищо друго с отговора. На практика това е изчезнало от паметта на R.

Често, когато се занимаваме с програмиране или просто с анализа на данни, ще искаме да използваме отговора на един въпрос в следващ въпрос. В този случай е удобно, ако не и задължително, R да запази отговора. За да накараме R да направи това, ние присвояваме отговора на въпроса на т.нар. обект. За обектите може да мислите като за променливи, както във всеки един друг език за програмиране.

Какви са структурите от данни в R? R има голямо разнообразие от обекти, за да съхраняваме данни - например вектори, фактори, матрици, масиви, датафрейм и списъци. Те се различават по отношение на типа данни, които могат да съхраняват, как са създадени, както и тяхната структурна сложност. Нека започнем с най-лесните - векторите. Тези обекти имат едно измерение и могат да пазят поредица от еднотипни елементи (например само числа или само стрингове/текст).

Извършваме присвояването (да накараме R да запомни отговор в нашия обект), като използваме стрелката за присвояване, която е знак за по-малко, последван (без интервал) от знак минус „<-„. Стрелката сочи от дясно наляво, така че информацията върви от дясно наляво. Ние присвояваме отговора на команда към mynumbers, което сме задали като име на нашия обект. Всеки обект трябва да има собствено уникално име (съставено от букви, цифри, долна черта или точка). Насърчаваме ви да бъдете внимателни: вероятно ще трябва да напишете това име на обекта отново, така че не го правете твърде дълго или сложно, но го направете информативно, така че да ви подсказва каква информация съхранява.

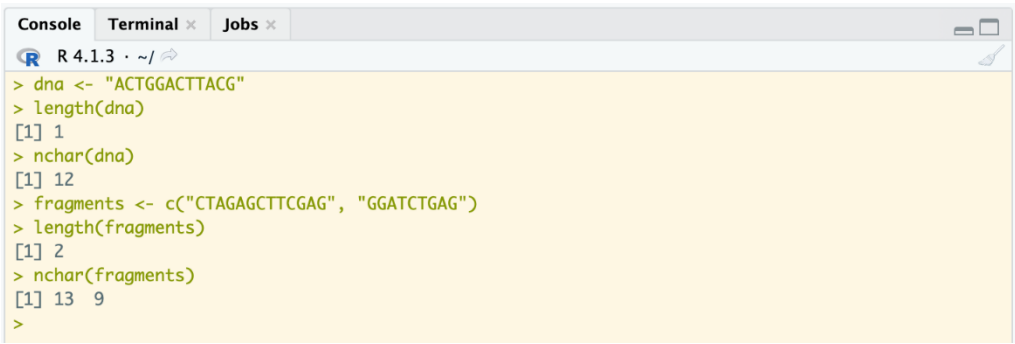


Така с познатата ни функция seq() казваме да запазим числата от 1 до 10 със стъпка 0.5 в обект с име mynumbers. След като натиснем нов ред, на следващия ред се появява нова подкана за команда. Не виждаме отговора на нашия въпрос, защото R е присвоил (запазил) стойностите, произведени от дясната страна на командата към обекта в лявата страна. Тоест, ако R е в състояние да интерпретира нашата инструкция, той прави каквото му е казано и след това просто казва, че е готов за следващата инструкция. Нека да свикнем, R няма да ни поздрави, когато изпълним успешно команда ☺, но ако сме направили грешка, той няма да пропусне да ни критикува! За да видим отговора на въпроса, просто трябва да въведем името на обекта и R ще върне каква информация е запазил в него. По-горе виждаме, че в отговора, който R ни дава, поредността от елементите в обекта от тип вектор са маркирани, като първият елемент [1] е 1, а [18] елемент е 9.5.

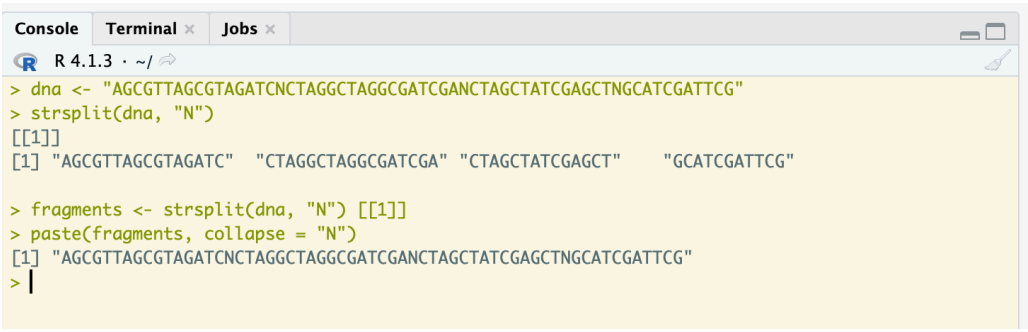
Тук можем да използваме още две от многото базови функции, с които разполага R – max() и min(), които могат да ни дадат най-голямата и респективно най-малката стойност на елемент нашия обект mynumbers:

```
> min(mynumbers)
[1] 1
> max(mynumbers)
[1] 10
>
```

Освен вектори с числови стойности, ние можем да създаваме и такива от текстови елементи. Нека да направим един обект вектор, наречен dna, и да присвоим малка ДНК секвенция. С функциите length() и nchar(), ние можем да проверим колко елемента съдържа нашият вектор както и дължината на текста. С функцията вектори c() ние можем да създадем вектор, съдържащ повече от един стринг/дума (както и други еднотипни елементи):

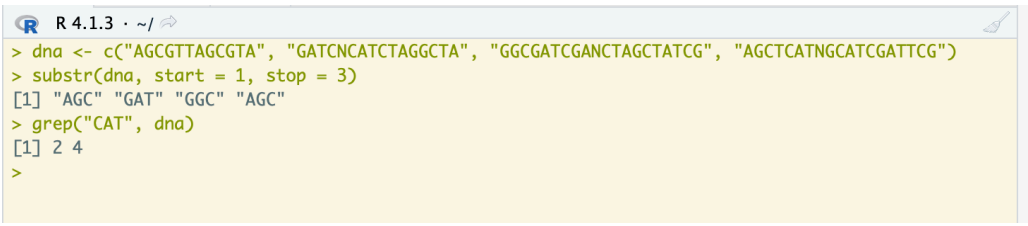


В молекулярната биология и биоинформатиката стринговете се използват изключително много, тъй като първичната секвенция на базите в молекулите ДНК, РНК и протеини се представя от символи и стрингове. Затова е много важно ние да можем да манипулираме стрингове, когато работим с такива данни. Винаги, когато работите със стрингове, трябва да можете да свързвате „думите" (да ги нареждате заедно) и да ги разделяте. В R използваме функцията paste() за конкатенация и функцията strsplit() за разделяне на стрингове:



Така можем да използваме тези функции, за да раздели ДНК секвенция, която съдържа бази N на фрагменти, както и след това отново да ги обединим в една молекула. Аргументите и на двете функции са обектът, както и по какъв символ ще се разделя или конкатенира, в случая по базите N. Характерното е, че strsplit() връща обект списък и за разлика от векторите, обектите списъци могат да запаметяват цяла колекция от различни обекти. Тъй като списъците могат да съдържат множество обекти в себе си, указваме със [[1]] кой вектор искаме да извадим, за да работим с него, а той е и единственият в този случай в списъка. Списъците са най-сложните обекти в R, възприемайте ги като група (колекция) от други обекти.

Освен тези манипулации друго важно нещо е да можем да намираме част от текст в друг текст (субстринг). Например това е много полезно в случаите, когато искаме да намерим някакъв мотив в ДНК секвенция. Можем да търсим текст в стринг по позиция или по шаблон. Със substr() можем да извличаме определени позиции на символи от текст, а с grep() да търсим наличието на даден текст или шаблон. Когато казваме шаблон тук, трябва да напомним, че също можем да използваме регулярни изрази, които са много мощно средство за работа с текст, вероятно познати ви от други езици например Python, Perl и др.



За момента се научихме как да създаваме обекти вектори и да ги манипулираме (да извършваме различни действия с тях). Освен тях, както споменахме, в R има различни типове обекти, в зависимост от това какви данни те могат да съхраняват:

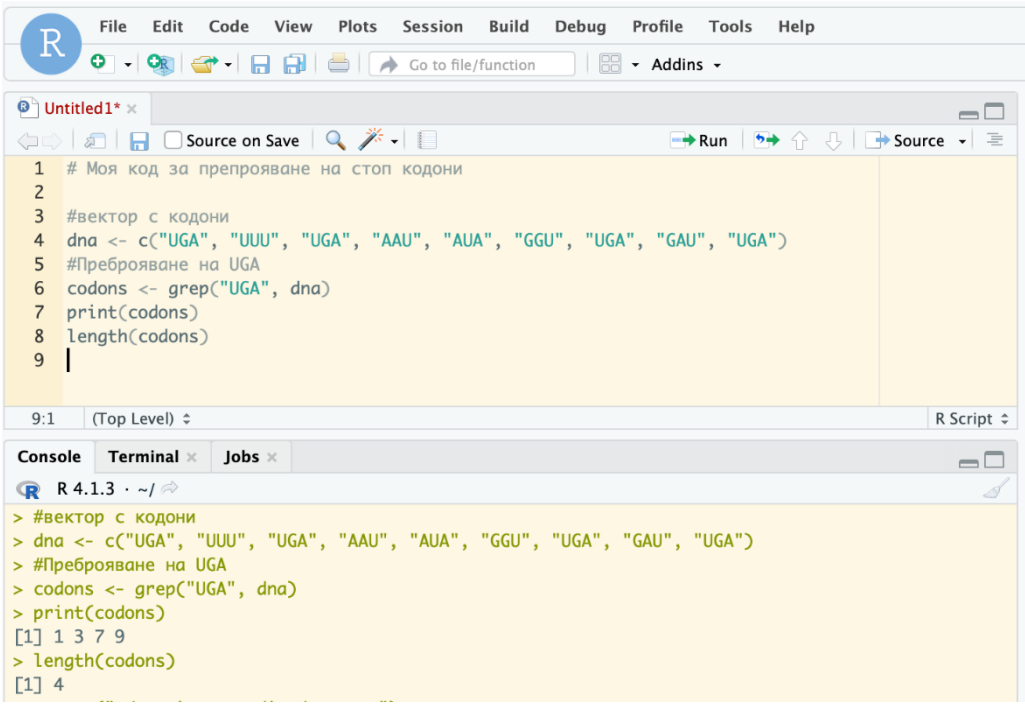
- Матрици – това са двуизмерни обекти, които могат да съдържат еднотипни елементи (само числа или само текст). Представете си матрицата като таблица с редове и колони.
- Масивите – това са обекти, подобни на матриците, но имат повече от две измерения, представете си масивите като естествено продължение на матриците (множество матрици) и отново могат да съхраняват еднотпни данни.
- Датафрейм – това са подобни обекти като матриците, но могат да съхраняват едновременно данни от различен тип. Те са едни от най-използваните обекти в R. В подобни обекти се съхраняват данни например от често срещани статистически софтуери, например SPSS.
- Списъци – това е колекция от обекти. Могат да съхраняват различен тип обекти.
- Фактори – представляват данни за категоризация. Те са специални структури от данни. Тясно свързани са с векторите от текстов тип, защото всеки един такъв може да бъде представен като фактор. Факторите са от решаващо значение в R, защото определят как данните се анализират и представят визуално. Повече за тях ще обясним, когато разгледаме някои примери при анализа на различни данни.

Отново ще подчертаем, че в биологията ще се срещаме и ще работим изключително често с данни от тип датафрейм, защото те са много удобен формат за съхраняване на данни. Представете си ги като една двуизмерна таблица в която имаме както числова, така и текстова информация, описваща нашите биологични обекти. И предполагам и вие имате данни от този тип? Погледнете във вашия Excel, вероятно вече имате подобни тип таблици?

Нашият първи скрипт на R. Пакети в R

Нека да погледнем горния десен панел на RStudio. Това е редакторът на код (скриптове) в R. Това означава, че не е задължително да подаваме в конзолата команда по команда, а можем да напишем нашия код с всички команди и той да бъде изпълнен след това. За да отворите нов скрипт, отидете в менюто Файл, след това Нов файл и щракнете върху R Script. Също така можете да щракнете върху горния ляв бутон в лентата с инструменти на скрипта, който изглежда като документ със зелен плюс, и след това щракнете върху R Script в падащото меню.

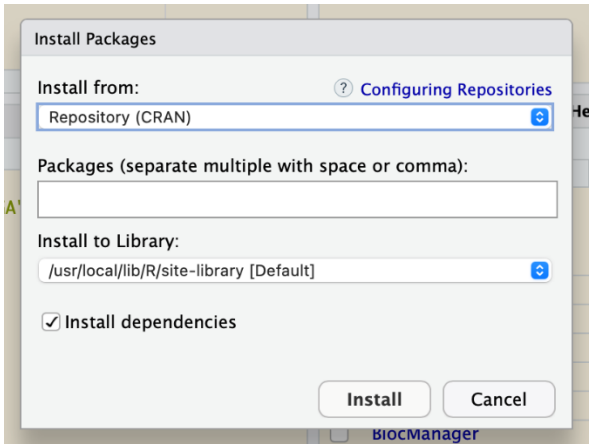
Вече имаме къде да напишем нашите инструкции към R, както и да ги запазим, ако искаме да изпълним отново същите след време. Нека започнем да пишем в скрипта с много специален символ - символ #. Той е много специален знак и както и при други програмни езици означава коментар. Там в свободен текст може да опишете частите на вашия код, което е много важно, за да може други, когато четат вашия код да се ориентират лесно, а дори и когато си четете кода след време. R ще игнорира тези редове, започващи с #, това е важно, защото те са само за ваша информация и R знае, че те не са команди за него.



Когато напишем нашия скрипт с команди, можем да го изпълним с натискане на бутона “Run”. Имайте предвид, че изпълнението отново ще е ред по ред. Тук можем да видим нашия първи скрипт, с който можем да преброим колко стоп кодони имаме в нашите данни, които са запазени в един стрингов вектор.

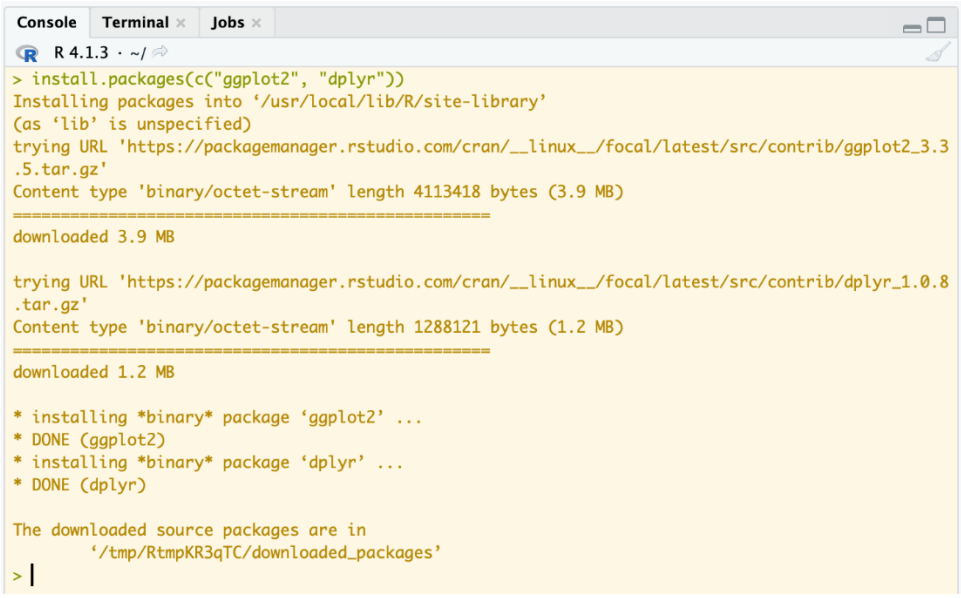
Функционалността на базовия R може да се разшири благодарение на хиляди налични допълнителни пакети. Всеки такъв пакет има определен фокус - например пакетът stats съдържа функции, които прилагат общи статистически методи, а графичният пакет има функции, отнасящи се до графиката. Когато изтеглите R, автоматично получавате набор от базови пакети. Това са основни пакети, които съдържат широко използвана статистическа и графична функционалност. Тези базови пакети на R представляват малка част от всички пакети, които можете да използвате с R. Всъщност към момента на писане на този учебник има повече от 19000. Тези други пакети наричаме „add-on“ пакети, защото трябва да ги добавите към R, от CRAN архива допълнително.
R пакетите могат да бъдат инсталирани по различни начини. Но, както може да очакваме, RStudio ви дава лесен начин да направите това. Разделът „Пакети“ (Packages) в долния десен панел има бутон Инсталиране (Install Packages). След

щракването върху бутона ще се появи малък прозорец с три основни полета: Инсталиране от, Пакети и Инсталиране в библиотека. Важно е полето за имената на пакетите, които искаме - Packages; другите две могат почти винаги да бъдат оставени по подразбиране.



Когато започнете да въвеждате първите няколко букви от името на пакета (например dplyr), RStudio ще предостави списък с наличните пакети, които отговарят на това. Напълно възможно е да инсталирате повече от един пакет, като поставите запетая или интервал между тях. След като ги намерим, всичко което трябва да направим, е да щракнем върху бутона Инсталиране и да оставим RStudio да направи своята магия ☺.

Нека изпробваме това сега. В тази книга ще работим с два пакета: dplyr и ggplot2. Бихме искали да ги въведете и да ги инсталирате. Следвайте инструкциите по-горе. Трябва да е доста лесно. След като приключите, разгледайте бързо конзолата. Ако всичко работи, ще видите, че всичко, което RStudio е направил, е да изпрати R функцията install.packages() към конзолата, за да инсталира пакетите вместо вас. С тази команда вие също може да инсталирате пакети в конзолата:



Сега, когато имаме инсталирани нужните пакети, можем да използваме функциите от тези пакети, като предварително, в началото на нашия скрипт, декларираме кои пакет ще използваме, за да се заредят със library(ggplot2). Можем да декларираме зареждането на колкото искаме пакети, нужни за нашата работа. Вече сме почти готови да започнем работа с R и RStudio. Не забравяйте, че когато искате да получите помощ и информация за някоя функция, винаги може да напишете в конзолата ?име_на_функция, например ?seq(). Освен това от адреса https://www.rstudio.com/resources/cheatsheets/ може да свалите страхотни бързи справочници за синтаксиса на базовия R и на някои много използвани пакети.

Глава 2

Как да въведем нашите данни в R и RStudio?

За да започнем анализа на данните, с които разполагаме по даден проблем, ние трябва да можем да ги въведем в R. Една от най-честите спънки за студентите и опитни изследователи, използващи R, е именно тази стъпка. Това е наистина жалко, тъй като това е и първият досег на повечето хора с R! Нека направим тази първа стъпка възможно най-лесна. Вашите данни може да са записани в различни формати - CSV, TXT, или файлове от Excel, SPSS, и тук ще научим как да ги въвеждаме (импортираме) в R.

Както в много научни сфери, внимателната подготовка е ключът към добрите резултати. Така че, да се надяваме, че четете това, преди да сте събрали, придобили данните, с които ще работите.

R харесва данни, в които има ред с т.нар. наблюдение и всяка променлива е представена от своя собствена колона. Някои хора наричат това „дълъг формат“; други го наричат "подредени данни" (tidy data). R харесва данни, които изглеждат така в смисъл, че много от инструментите, които използвате в R (инструментите са функции), работят върху данни, които изглеждат в подобен формат.

Какво точно означава да имате данни с наблюдение и всяка променлива само в една колона? Представете си набор от данни, представляващи теглото на мъжки и женски мишки. Това може да бъде подредено в две колони, едната съдържа теглото на мъжките, а другата на женските (Фиг. 3).

	A	B
1	Male.mouses	Female.mouses
2	26	18
3	27	19
4	25	17
5	28	20
6	25	22
7	22	19
8	20	17
9	24	24
10	26	22
11	25	23
12	28	21
13	26	14
14	26	19
15	27	18
16	26	18

Фиг.3 Данни за теглото на мъжки и женски мишки.

Това обаче няма да се хареса на R! Във всеки ред има две наблюдения - стойностите на височините са в две колони, както и променливата за пол, въпреки че е една и съща променлива. Алтернативното, удобно за R подреждане, е да се генерира една колона, наречена „пол“ и втора, наречена „тегло“ (Фиг. 4). Тази подредба има повече редове и затова се нарича „дълъг формат“.

	A	B
1	Mouses	Weight
2	male	26
3	male	27
4	male	25
5	male	28
6	male	25
7	male	22
8	male	20
9	male	24
10	male	26
11	male	25
12	male	28
13	male	26
14	male	26
15	male	27
16	male	26
17	female	18
18	female	19
19	female	17
20	female	20
21	female	22
22	female	19
23	female	17
24	female	24
25	female	22
26	female	23
27	female	21
28	female	14
29	female	19
30	female	18
31	female	18
32		

Фиг.4 Подреден формат (tidy data) на данните за теглото на мишките.

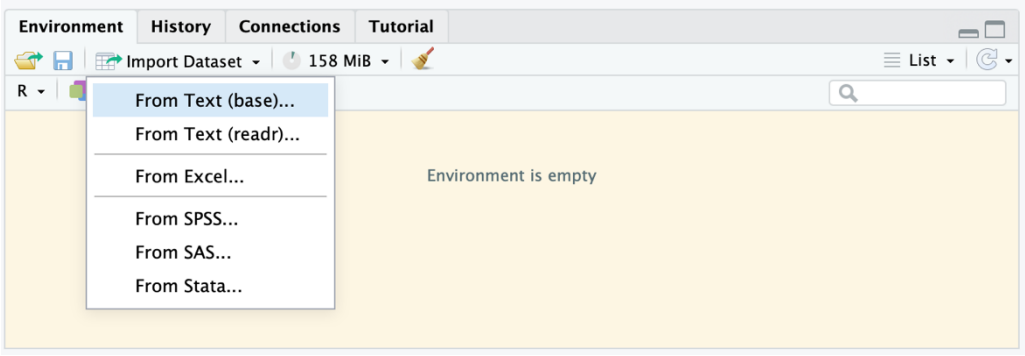
Ако имаме малко данни, ние можем да ги трансформираме лесно и „ръчно“ в Excel, но ако данните са в хиляди редове, това ще е трудно. По-късно ще научим как да правим това автоматично с помощта на пакет от R.

След като имаме данните във формат, който е подходящ, нека запишем файла от Excel в CSV. Това е обикновен текстови формат, в който стойностите от таблицата и нейните колони са разделени със символ (например запетая или точка и запетая), затова се нарича comma separated values (CSV) формат. Той е удачен формат за запис на данни, които ще бъдат импортирани другаде, по-удачен от XLS, защото формата на стандартните екселски файлове може да съдържа много други данни, които са специфични и са нужни само за Excel, но другите програми не могат да ги прочетат. Подобен формат са т.нар. файлове с табулирани данни, те представляват отново текстови файлове, в които данните от дадена таблица са форматирани по такъв начин, че отделните колони се формират чрез

специален символ (например табулация). Нека сега да разгледаме различните начини да импортираме нашите данни, които имаме във файл.

Rstudio има специално функционалност за импортиране на данни чрез бутона Import Dataset в горния десен панел на RStudio. Това ще ви позволи да отидете до вашия файл с данни, да го изберете и да щракнете върху бутона Отвори. Когато се опитате да използвате това за първи път, RStudio може да ви помоли да инсталирате или актуализирате някои от пакетите; просто кажете да! Това са пакети, които правят импортирането на данни по-бързо и по-лесно.

Нека да импортираме файл, който съдържа данни за дължината и теглото на даден биологичен обект с две групи (контрола и третиран). От падащото меню за импорт виждате различни формати. Изберете From Text (readr), за да импортирате CSV. Ще се отвори диалоговият прозорец, разкриващ много удобен инструмент за импорт (Фиг. 5).

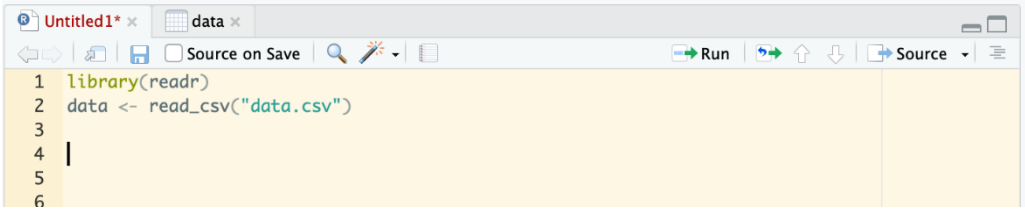


Фиг. 5 Импортиране на данни от файл.

В Data Preview ще видите как R интерпретира вашите данни (все още не са в R, вие просто получавате предварителен преглед). В този момент можете да видите някои от опциите за импортиране в долната лява част на прозореца (например тип разделител, опции за първи ред и т.н.). Често не е нужно да променяте нищо и можете просто да щракнете върху бутона Импортиране.

Преди да направите нещо повече, копирайте първите два реда в полето за преглед на кода (Фиг. 6). Там RStudio ви показва всички тези действия на какъв код отговарят. След това щракнете върху Импортиране. RStudio ще стартира целия код в полето за предварителен преглед. Импортирането става с помощта на функцията read_csv() от пакета за четене на файлове readr. По време на импортирането също така отваря прозорец, показващ данните.

Сега поставете двата реда код във вашия скрипт. След като направите това, не е нужно да използвате инструмента за импортиране отново. Просто стартирате тези редове код и вашите данни ще се прочетат. Вашият скрипт вече съдържа информацията за това кой набор от данни се използва. Изпълнявате този код всеки път, когато работите със скрипта. Така, когато декларирате пакета readr, може да използвате функцията read_csv() и с две лесни команди код да импортирате бързо вашия файл в обект data:

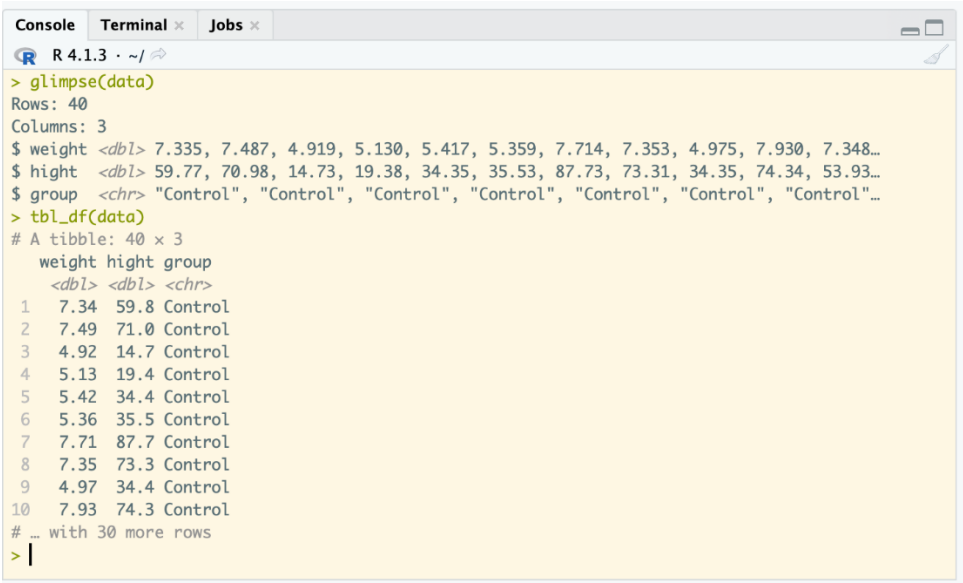


Сега е време да проверим в конзолата, дали правилно сме си импортирали данните. Това можем да направим с няколко лесни и полезни функции. Не е изненадващо, че names() ви казва имената, които сте присвоили на всяка колона (т.е. имената на вашите променливи, имената на колоните във нашия .csv файл, които сте въвели в Excel или в друга програма). Функцията head() връща първите шест реда от набора от данни (и сега нека познаем какво прави tail()?). dim() ви казва броя на редовете и колоните – измерението – на набора от данни. И накрая str() връща структурата на данните, комбинирайки почти всички предишни функции в една удобна такава. За измерението от данните виждаме, че имате

таблица с данни на обект и броя на наблюденията (редове) и променливи (колони). Имаме редове за всяка от вашите променливи, даващи името на променливата, типа на променливата (числови, фактори или цели числа, например) и първите няколко стойности на тази променлива. По този начин със str() може да гарантирате, че импортираните от вас данни са това, с което възнамерявате да работите, и също да си напомним за характеристиките на нашите данни.

Управление, манипулиране, и обработка на данни с dplyr.

Нека сега да направим нашите първи стъпки с пакета dplyr. В предишната част ви накарахме да инсталирате пакета dplyr. Ако сме работили заедно и сте стартирали примерните команди, dplyr трябва да е готов за използване. Две функции от dplyr също са полезни за разглеждане на данните, които току-що сте импортирали. Едната е glimpse() и осигурява хоризонтален изглед на данните. Другата е tbl_df() и осигурява вертикален изглед. И двете показват имената на колоните/променливите и типа данни.

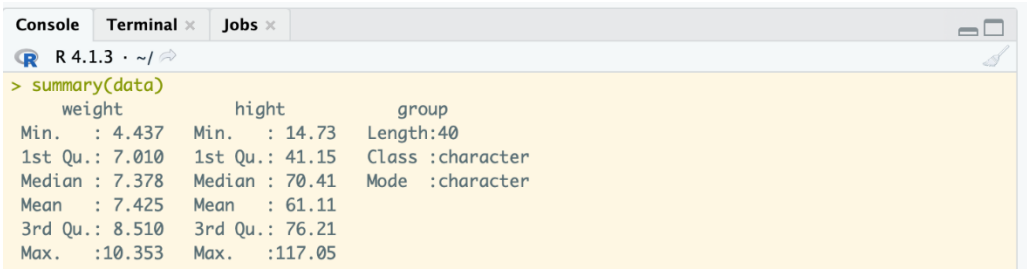


Проучването, манипулирането и визуализацията на вашите данни са основни части от процеса на научния анализ. Нашият работен процес започва с опознаване на данните, както и с обработката им (манипулирането им), преди да направим каквито и да е или по-сложни анализи, както и да генерираме графики за визуализацията на текущите данни или резултатите.

Има два основни набора от инструменти за тези стъпки: инструменти за манипулиране и обработка на данни и инструменти за графики. В тази глава представяме пакета dplyr за извършване на различни общи манипулации с данни. Това е изключително важно, тъй като почти винаги трябва да правим някаква

манипулация на данни, като например подмножество или изчисляване на средна стойност, ± SE, или статистичен метод (ще научим за тях в Глава 4). Трябва да извършвате цялата манипулация на данните в R (и никъде другаде) и тази глава ще ви даде знанията за това. Така целият алгоритъм (workflow) на работния процес ще може да се съдържа на едно място: вашия R скрипт.

До момента разгледахме няколко удобни инструмента за разглеждане на части от нашите данни. Докато names(), head(), dim(), str(), tbl_df() и glimpse() са добри, за да ви кажат как изглеждат вашите данни (т.е. да видим, дали имаме правилен брой редове и колони), те не дават много информация за конкретните стойности на всяка променлива или обобщена статистика за тях. За да получите тази информация, използвайте функцията summary() за вашия датафрейм, рамка от данни (Споменахме за този тип обекти – нашите импортирани данни са запаметени по този начин, защото представляват двумерна таблица с елементи от различен тип числа и текст). Продължете и приложете функцията summary() към тези данни:



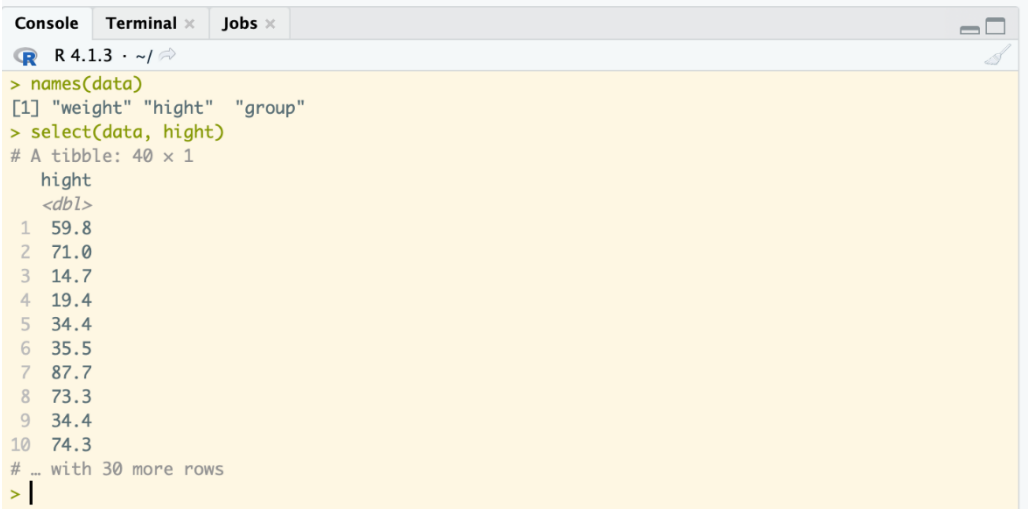
Функцията summary() в R ни дава някои параметри от дескриптивната статистика - медианата, средната стойност, интерквартилния диапазон, минимума и максимума за всички числови колони (променливи) и „нивата“ и размера на извадката за всяко ниво на всички категорични колони (променливи). Струва си да разгледате внимателно тези обобщени статистически данни. Те могат да ви кажат, дали има неочаквано екстремни, неправдоподобни или дори невъзможни стойности – т.е. добро първо анализиране на вашите данни. Сега нека продължим да използваме тези данни, за да научим как dplyr може да изследва и манипулира данни.

В този пакет има т.нар. „глаголи“, които всъщност са функции, фокусирани върху манипулирането на данни. Функциите са: select(), slice(), filter(), arrange() и

mutate(). Както имената подсказват от английски език, select() е за избор на колони, slice() е за избор на редове, filter() е за избор на подмножества от редове, зависими от стойностите в колона (филтриране на данни), order() е за сортиране на редове, а mutate() е за създаване на нови колони/променливи в датафрейма. Това са основни функции за основни дейности, съсредоточени върху избора на части от или поднабора на нашите данни. Разбира се, има още много полезни функции, може да ги разгледате от справочниците, които споменахме на адрес: https://www.rstudio.com/resources/cheatsheets/

Ключът към използването на dplyr е да запомните, че първият аргумент за всички dplyr функции е вашият обект – датафрейм, съдържащ данните!

Нека да изпробваме select(), който взима определени колони от вашите данни. Разбира се, в случаят ще ни помогне да се знаят имената на променливите (колоните), така че ако е необходимо, първо ще използваме names(), за да си ги припомним. Ето как можем да го използваме, за да получим само колоната от нашите данни за височина:

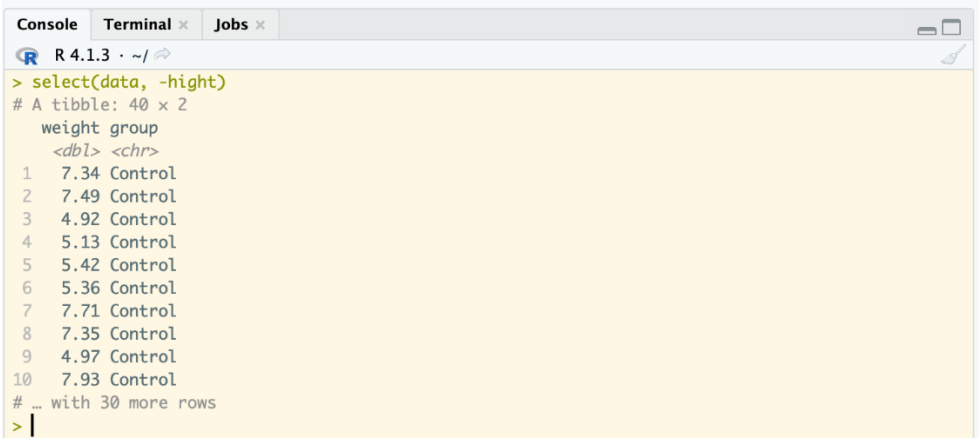


Важно! Ако получите грешка от типа *Error: not could find function "select"*, значи или не сте декларирали предварително библиотеката (dplyr) в горната част на вашия скрипт, или не сте изпълнили този ред код!

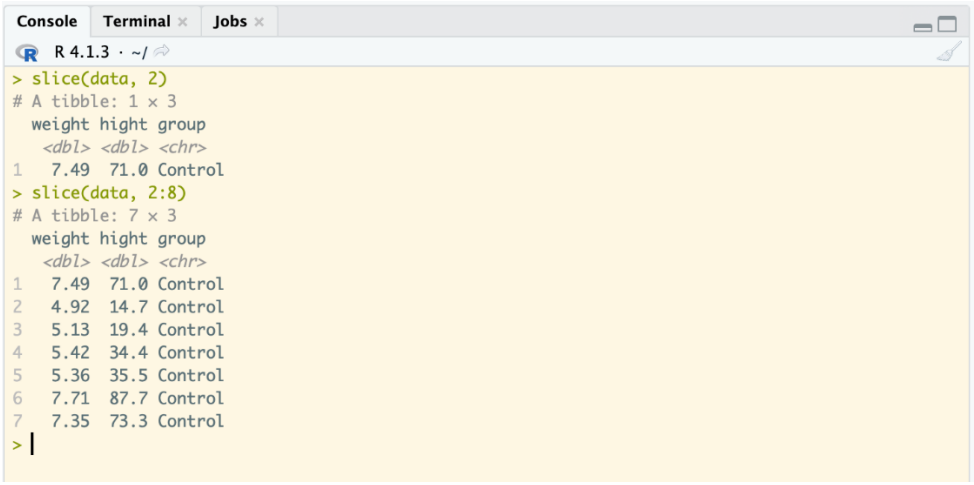
И така, нека разгледаме някои подробности за това как работи dplyr. Първо, можете да видите, че този пакет е доста лесен за използване. Ако искаме колона, казваме на dplyr в кой набор от данни да търси и коя колона да вземе.

select(), като dplyr функция, използва рамка от данни (датафрейм) и връща също такъв обект. Не всички R функции действат по този начин, някои връщат различен обект от този, с който работят, и това трябва да го имаме в предвид.

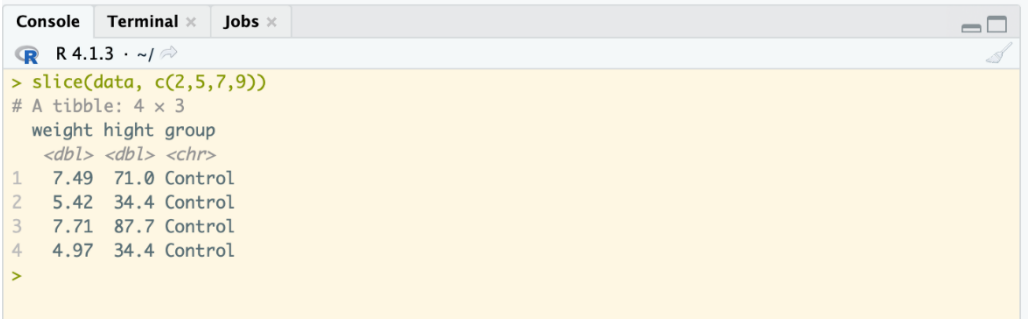
И накрая, може да забележите, че select() изглежда прави само едно нещо. Това е напълно вярно. Всички функции на dplyr правят едно нещо, и то много бързо и много ефективно. select() може да се използва и за избор на всички колони с изключение на една. Например, ако искаме да пропуснем колоната hight, оставяйки само колоните weight и group:



Сега да видим какво прави slice() – функцията взима редове от данните (наблюдения). Работи, като изискваме конкретни номера на редове, които поискаме. Можете да поискате един ред, последователност или прекъснат набор. Например нека да поискаме да вземем 2ри ред, а после цяла последователност от 2ри до 8ми:



или определени редове, които изреждаме (и подаваме като вектор с числа) с познатата ни функция c():

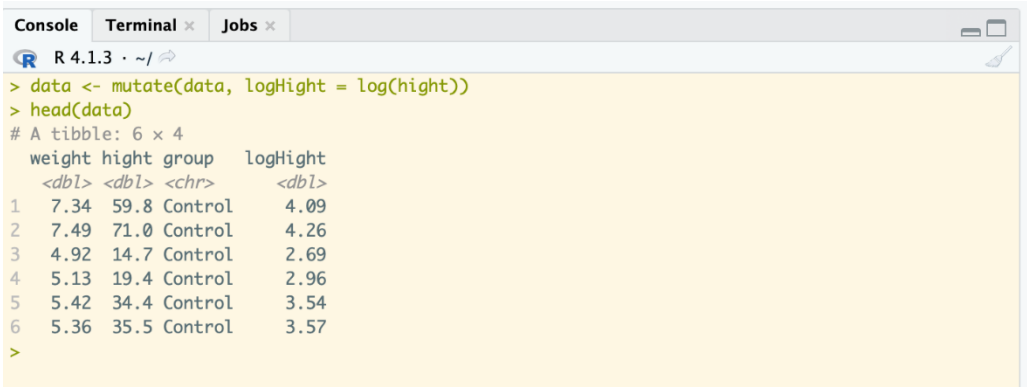


Следващата функция filter() е много полезна. Тя обаче изисква някои основни познания за логически оператори и булеви оператори в R. R има пълен набор от логически оператори, за които вече споменахме. И сега можем да комбинираме функцията filter() с логическо условие, за да филтрираме нашите данни, например нека филтрираме и изберем редовете от нашите данни, при които височината е по-голяма от 80 и тежестта е по-голяма от 7. Ще видим, че резултатът е 9 реда, които отговарят едновременно и на двете условия, които сме групирали с логическо „и“ -„&“:

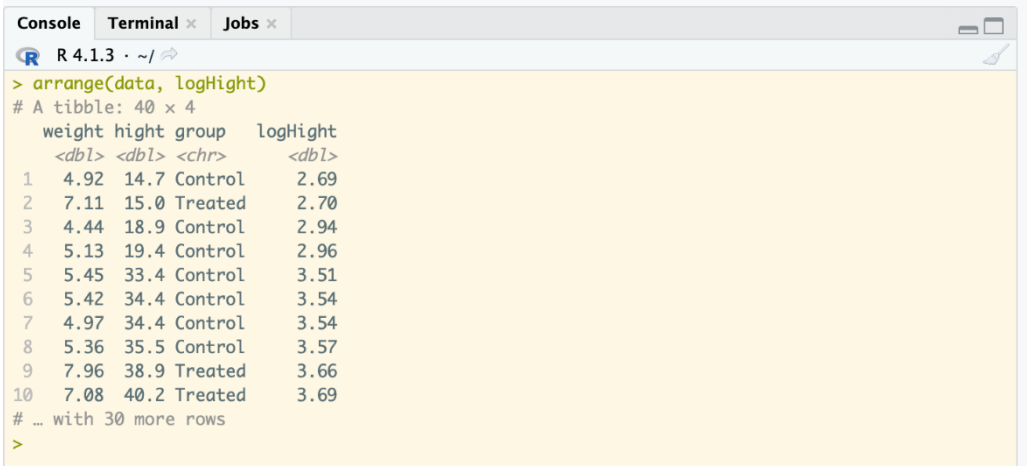
За да се уверим, че ще можем да запомним нашия резултат, трябва да го запаметим в нов обект, като преди командата за филтриране добавим променливата и знака за присъединяване „<-„. Това ще осигури работата по-нататък само с тази част от филтрираните данни.

Трансформирането на колони от нашите данни е обичайна практика в много биологични области. Например обичайно е да се log-трансформират променливи за изобразяване на графики или анализ на данни. Може също да се окажете в ситуация, в която искате да представите или анализирате променлива, която е функция на други променливи във вашите данни. Тук разкриваме основния подход за използване на mutate() за решаване на тази задача.

Както при всички функции на dplyr, синтаксисът на mutate() започва датафрейма, в който се намират данните, и след това обозначава име на новата колона (променлива, която трябва да се изчисли с трансформацията). Например нека log-трансформираме hight и го наречем logHight. Ще направим тази нова колона да се добави в нашата работна рамка с данни, като използваме трик, присвоявайки стойностите, върнати от mutate() на обект със същото име като оригиналните данни. По същество презаписваме датафрейма! Ще използваме също head(), за да ограничим броя на редовете, които виждаме, само за яснота:



Понякога е важно да сортираме наблюденията (редовете) на нашите данни в определен ред. Може просто да искате да разгледате набора от данни и да предпочетете конкретно подреждане на редовете. Можем да използваме функцията arrange(). Нека сортираме данните по новата колона която log-трансформирахме:



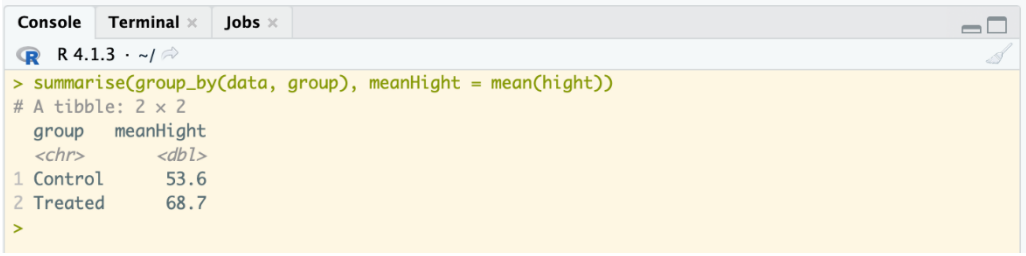
В този момент трябва да се чувствате доста уверени. Можете да импортирате и да изследвате структурата на вашите данни и дори да манипулирате различни части от тях, като използвате петте функции от dplyr, включително filter() и select().

В нашите данни имаме променлива за категории, group. Тя има две нива, Control и Treatment. Тази структура на данните означава, че може да сме в състояние да направим изчисления за всяка една категория. Всеки път, когато

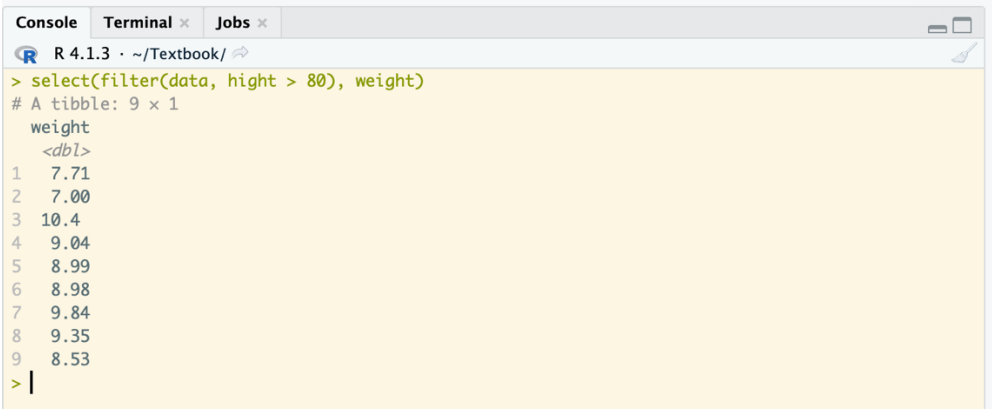
имате структура или групи, можете да генерирате с R доста невероятна и бърза обобщена информация.

Двете ключови функции на dplyr за това са group_by() и summarise(). Също така нека въведем и функциите mean() и sd() (стандартно отклонение).

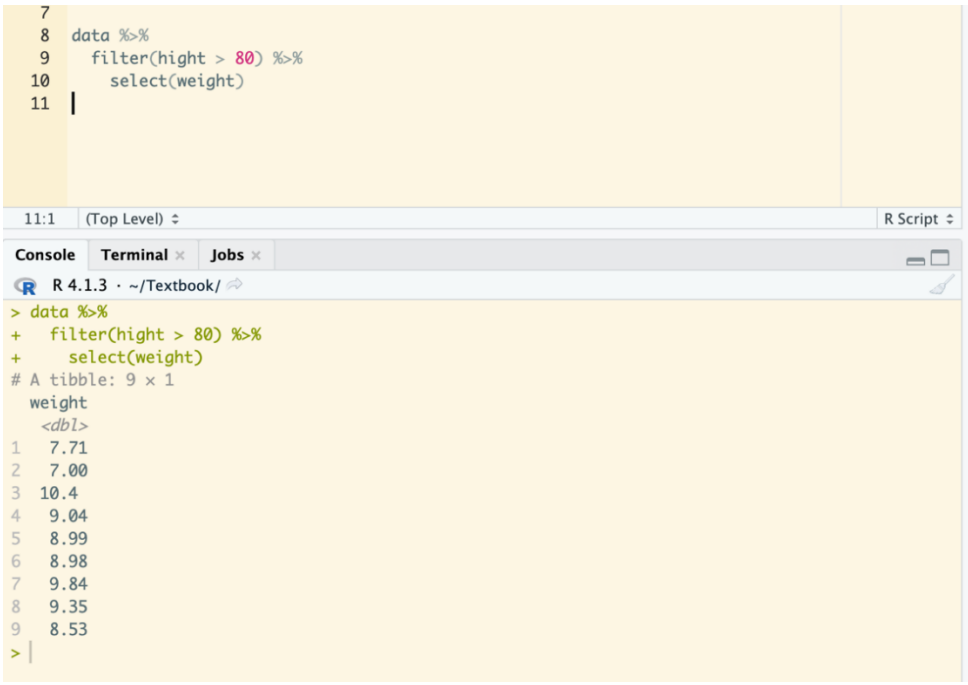
Обобщаването се извършва по следния начин – подаваме датафрейма с данни като аргумент на математичната функция, с която да обобщим данните:



Нека сега научим нещо още по-вълнуващо и интересно - можем да използваме повече от една функция dplyr в един ред код! Да изпробваме това със filter() и select() :



Така едновременно филтрираме данните по едната променлива и взимаме колоната по другата променлива. И сега нека научим за един магически символ – „%>%“ от dplyr, него можем да разчитаме като „сложи отговора от лявата команда във функцията вдясно. Нека изпробваме по горния пример но чрез „%>%“:



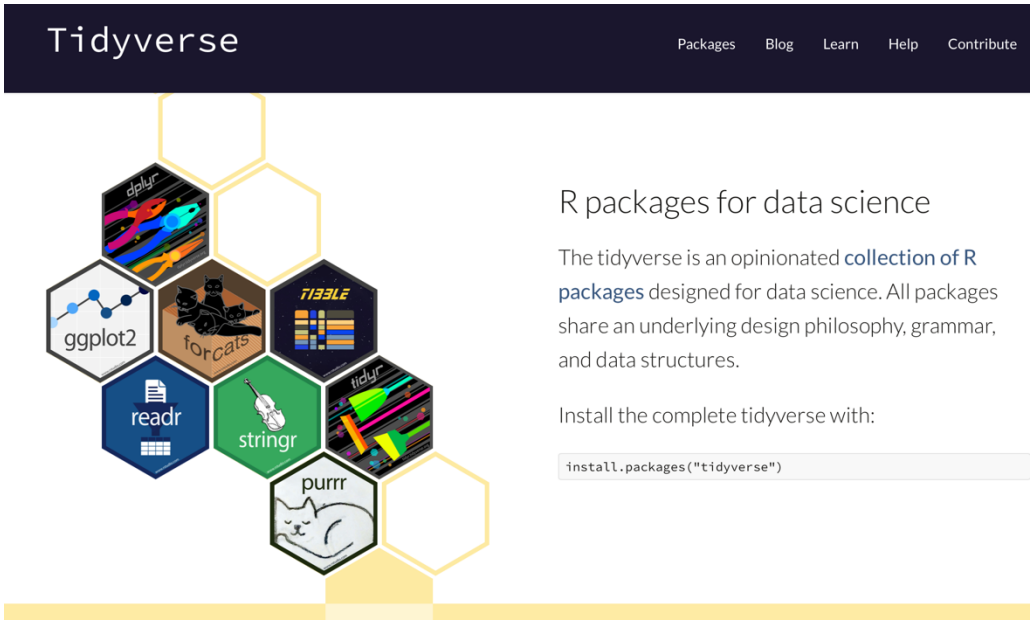
Тук сме написали примера в полето за скрипт, защото е на няколко реда. Но това може да го направим и в конзолата. Ако имаме команда, която е по-дълга и е на няколко реда, може да поставите в края на първия ред знака „+“ и това ще покаже на R, че следва да бъде въведен код и ще изчака да въведете и продължението на следващия ред. Така визуално може да форматирате и кода в редактора и ако имате много дълги редове код, може да ги пренасяте на следващите редове с „+“, както и да правите отстъп на следващите редове. Това прави кода ви по-четим и прегледен. И сега нека да изпробваме %>% с примера по-горе за обобщаването на информация по групи:

Важно! Един от най-често срещаните проблеми в обработката на данни е непълният набор от данни. За да се справи с липсващите стойности, R използва запазената ключова дума NA, която е съкращение от Not Available. Можете да използвате NA като валидна стойност, може да я присвоявате или да изпълнявате манипулации с нея. Резултатът на всички аритметични операции или логически такива със NA стойности отново ще водят до NA. Така тези данни в някои случаи няма да са ни необходими и трябва да намерим начин да се избавим от тях. Това можем да направим с функцията na.omit(). Ако използвате този метод за подмножество на вашите данни или за изчистване на липсващи стойности, не забравяйте да съхраните резултата в нов обект. R не променя нищо в оригиналния датафрейм, освен ако изрично не го презапишете. Това е хубаво нещо, защото не можете случайно да объркате данните си.

Важно! Когато искаме да работим само с един компонент/променлива от нашата рамка от данни, можем да използваме специалния символ $. Той взима само един компонент от обекта и ако това е датафрейм (нарушава неговото измерение), връща обект вектор. Нека изпробваме това, като разделим височината на теглото в нашите данни и запазим отговора в нов обект:

```
Console  Terminal  Jobs
R 4.1.3 · ~/Textbook/
> calc <- data$hight / data$weight
> head(calc)
[1] 8.148603 9.480433 2.994511 3.777778 6.341148 6.629968
>
```

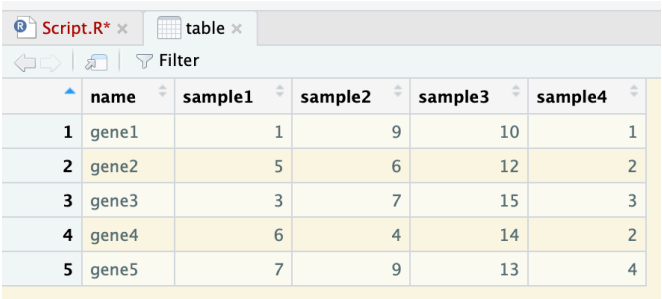
Пакетът, с който работихме в тази глава dplyr, е част от групата с пакети за обработка и манипулация на данни, наречена Tidyverse (https://www.tidyverse.org) (Фиг. 7). Може да разгледате още подробности за пакета на неговата страница: https://dplyr.tidyverse.org.



Фиг. 7 Tidyverse (https://www.tidyverse.org) пакети за анализ на данни.

Как да преформатираме нашите данни?

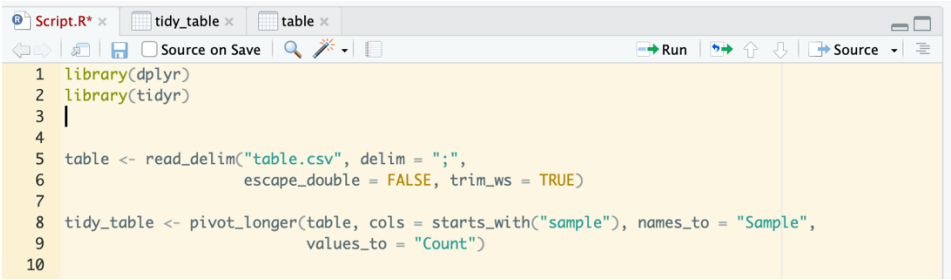
В началото на учебника говорихме, че R харесва подредени данни (tidy data). В много случаи, обаче в биологията ние имаме данни, които не са в този формат и трябва да ги преформатираме. Друг пакет от Tydiverse ще ни помогне за това. Това е пакетът tidyr. И така, ако имаме следната таблица с данни:


Script.R* × table ×

Filter

	name	sample1	sample2	sample3	sample4
1	gene1	1	9	10	1
2	gene2	5	6	12	2
3	gene3	3	7	15	3
4	gene4	6	4	14	2
5	gene5	7	9	13	4

можем да я преформатираме, като използваме функцията pivot_longer() и да не забравим да декларираме новия пакет в началото на скрипта ни с library(tidyr).


Script.R* × tidy_table × table ×

Source on Save | Run | Source

```
1  library(dplyr)
2  library(tidyr)
3
4
5  table <- read_delim("table.csv", delim = ";",
6                      escape_double = FALSE, trim_ws = TRUE)
7
8  tidy_table <- pivot_longer(table, cols = starts_with("sample"), names_to = "Sample",
9                             values_to = "Count")
10
```

Използваме cols = starts_with("sample"), за да кажем на функцията, че искаме да преформатираме колоните, чиито имена започват с "sample". След това pivot_longer() ще преформатира посочените колони в две нови колони, които наричаме „Sample“ и „Count“. Names_to = "Sample" указва, че искаме новата колона, съдържаща колоните, които сме подали с cols, да бъде наречена "Sample", а values_to = "Count" указва, че искаме новата колона, съдържаща стойностите, да бъде наречена "Count" . Така след командата нашата таблица придобива следния формат:

Script.R* × tidy_table × table ×

Filter

	name	Sample	Count
1	gene1	sample1	1
2	gene1	sample2	9
3	gene1	sample3	10
4	gene1	sample4	1
5	gene2	sample1	5
6	gene2	sample2	6
7	gene2	sample3	12
8	gene2	sample4	2
9	gene3	sample1	3
10	gene3	sample2	7
11	gene3	sample3	15
12	gene3	sample4	3
13	gene4	sample1	6
14	gene4	sample2	4
15	gene4	sample3	14
16	gene4	sample4	2
17	gene5	sample1	7
18	gene5	sample2	9
19	gene5	sample3	13
20	gene5	sample4	4

Бихме могли също да посочим диапазон от колони за преформатиране. Това може да направим с командата *cols = sample1:sample3* или дори да изключим някои колони, които не желаем в преформатирането с условието cols = -c("X1", "gene_symbol").

Може да разгледате какви други интересни функции ви дава този пакет за преформатиране на данни на неговата страница: https://tidyr.tidyverse.org/

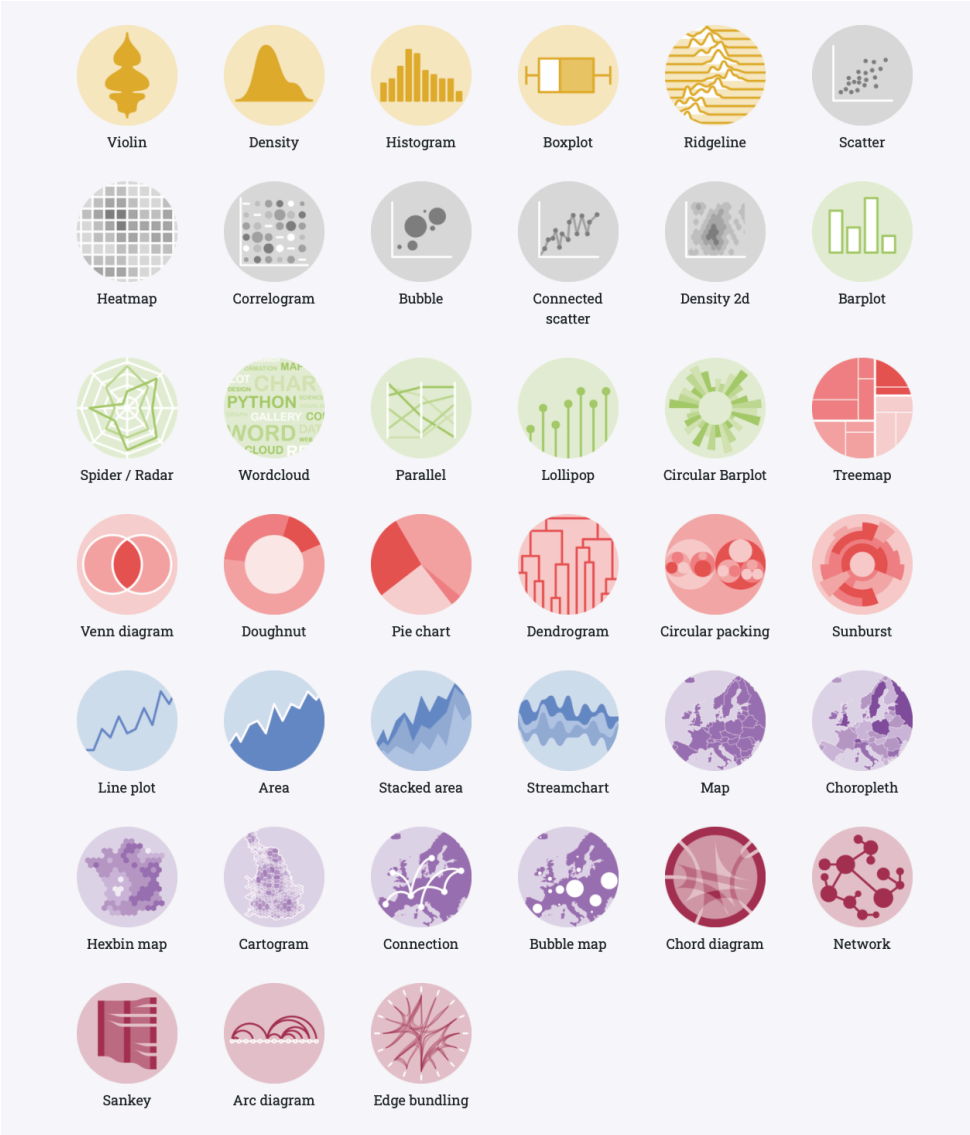
Глава 3

Как да визуализираме нашите данни?

Едно от фундаменталните правила за анализ на данни е да не започва анализът направо с изчисления или статистически анализ. Първо, винаги се опитайте да видите как изглеждат данните ви визуално. Направете графика, която трябва да ви покаже, дали вашите данни са добре подбрани, какво е тяхното качество, има ли липсващи данни. Дори и да не сте започнали с изчисленията, графиката ще ви подскаже, дали има някакви взаимовръзки или очаквани модели и хипотези във вашите данни.

Съществуват много различни типа фигури и визуализации, чрез които ние може да представим нашите данни и резултатите от тях. Например такива са диаграми на разсейването, (скатърплот, scatterplot), хистограми, вероятностни диаграми, боксплот диаграми (още: диаграми тип „кутия“, boxplot), блокплот диаграми (blockplot), кръгови диаграми (pie chart), стълбови диаграми (bar chart), радарни диаграми (radar chart) и др. (Фиг 8).

В тази част ще покажем как от вашите данни можем да генерираме подобни графики и плотове, които да визуализират определени моменти от вашия анализ, на които искате да акцентирате. И все пак, както сме споменавали и по-напред в този учебник, човек възприема много по-добре визуална информация, отколкото текстова или таблична, и затова е важно да видим някои от основните методи, с които ние можем да построим нашите графики и така да подплатим анализа ни с данни. Както знаете, всички научни изследвания са съпътствани от подобни графики и това не трябва да ни плаши, защото точно визуализацията на данни е една от силните страни на R.

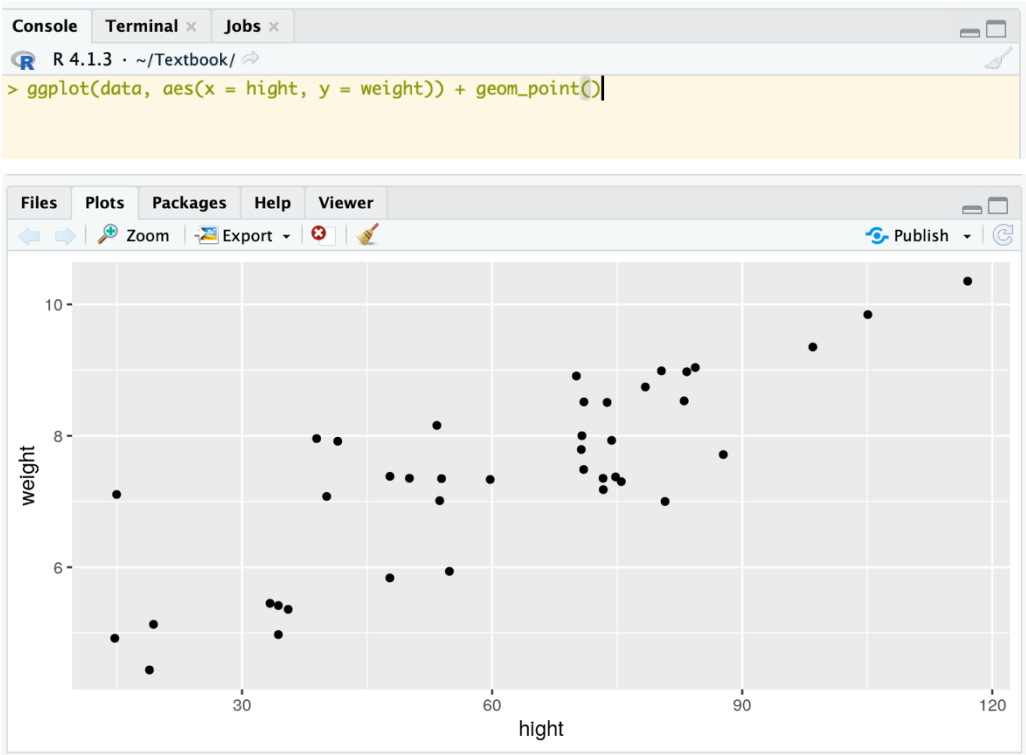


Фиг. 8 Различни типове диаграми и начини да визуализираме данни (Източник https://www.data-to-viz.com/).

В тази глава ние ще се фокусираме върху пакета за графики ggplot2. Той не само е популярен и има огромни онлайн ресурси и помощни материали, но и също така е изключително ефективен при работа с подредени данни и при взаимодействие с dplyr (все пак са от един ресурс с пакети - Tidyverse). Въпреки това, той има специфики, така че ще видим някои основни неща и ще започнем с

изобразяване на графика от нашите данни, с които вече работихме в предходните глави.

Нека започнем с основния синтаксис. Ето как казваме на функцията ggplot() (която се намира в ggplot2 пакета) как да направи проста, обикновена диаграма на разсейване (scatterplot) на данните:



Подобно на функциите в пакета dplyr, първият аргумент, който даваме на функцията ggplot(), е обектът с данните, в която се намират вашите променливи. Следващият аргумент, aes(), е интересен. Първо, той сам по себе си е функция. Второ, той определя естетиката на графиката – т.е. картографирането между променливите в набора от данни и характеристиките на графиката. Именно тук казваме на R в този пример да свърже x-позицията на точките от данни със стойностите на променлива hight и y-позицията на точките със стойностите в променливата weigth.

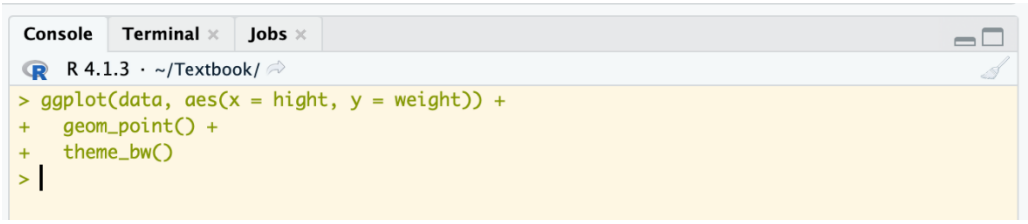
Другото важно нещо, което трябва запомним относно ggplot2, е, че работи чрез добавяне на слоеве и компоненти към нашата графика. Данните и естетиката (тип графика) винаги формират първия слой. След това добавяме геометричните

обекти като точки (в нашия случай са точно такива), линии и други елементи, които отчитат данните ни. Можем също да добавяме/настройваме други компоненти, като персонализиране на x- и y- осите, както и теми с цветове и т.н.. Важно е да знаем, че символът „+“ е начинът, по който добавяме тези слоеве и компоненти към основния първи слой.

И така, в горния пример добавяме с „+“ геометричния слой като точки. Използваме за това функцията geom_point().

Може да не ви хареса първоначалният резултат на графиката (ще кажете, че е леко скучен и неразбираем), но когато добавим още слоеве, с които можем да персонализираме нашата графика, тя ще придобие по-завършен вид.

Сега сме готови да направим няколко неща, за да направим графиката ни по-представителна и красива. Ще ви представим три неща: как може да се отървем от сивия фон, как да променим размера на точките и текста и как да накарате цветовете да съответстват на групите за нашия фактор (group). Ето един бърз начин да премахнем сивото. Вградени в ggplot2 са набор от теми. Една от тях е theme_bw(). Такива теми е добре да поставите като последния компонент на вашия ggplot():



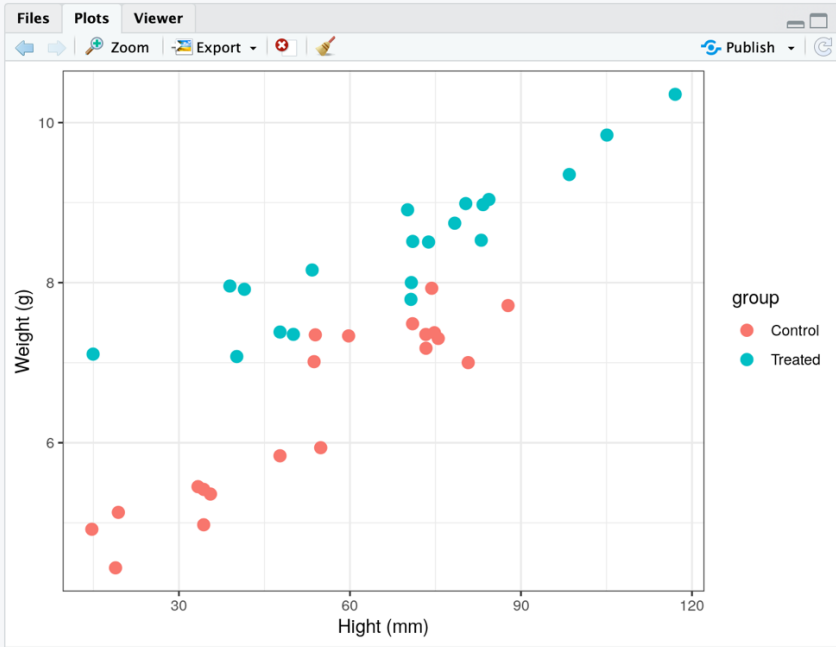
След това искаме да увеличим размера на точките. Лесно е да кажем, че искаме размерът на точките да е по-голям в слоя geom_points(), като използвате аргумент за размер:

Можем да променим етикетите на оста *x* и *y*. Тук изрично модифицираме чрез компонентите xlab() и ylab():

```
> ggplot(data, aes(x = hight, y = weight)) +
+   geom_point(size = 3) +
+   xlab("Hight (mm)") +
+   ylab("Weight (g)") +
+   theme_bw()
>
```

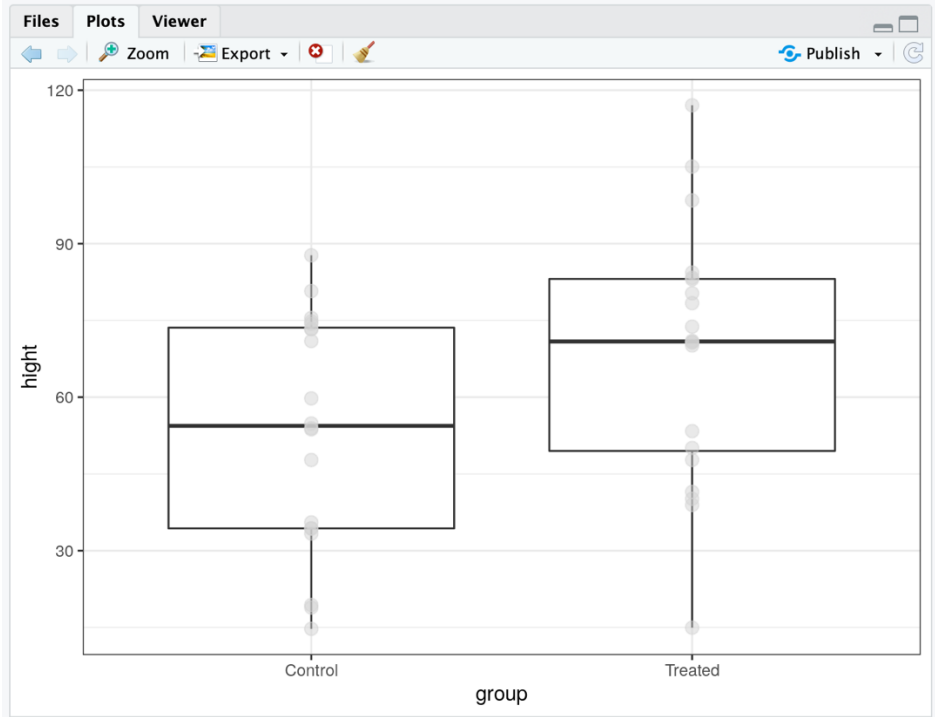
И последното персонализиране, което ще видим, е как да коригирате цветовете на точките, за да съответстват на конкретните групи. Това е толкова удобно за свързване на цифри със статистически модели, че е почти задължително да го знаете. И е много лесно с ggplot2, само добавяме colour=group и това дори ще добави и легенда за цветовете автоматично:

```
> ggplot(data, aes(x = hight, y = weight,  colour = group)) +
+   geom_point(size = 3) +
+   xlab("Hight (mm)") +
+   ylab("Weight (g)") +
+   theme_bw()
>
```



Диаграмите на разсейване или скатърплот, като по-горе, са много добри при показването на необработени данни. Въпреки това, има много начини за представяне на данни чрез някаква версия на тяхната централна тенденция (средна стойност или медиана) и някаква оценка на тяхната вариация (например стандартно отклонение или стандартна грешка). В биологичните науки диаграмите със стълбчета са често срещани. Въпреки това, тези видове графики могат да крият твърде много информация и не са подходящи в някои случаи.

В подкрепа на това ще предложим използването на добре установена алтернатива – боксплот графика. Нека да разгледаме отново данните ни и да се съсредоточим върху това как променливата за височина варира. ggplot2 има вграден geom_ за боксплот графики, не е изненадващо наречен geom_boxplot(). Ето как можем да го използваме:

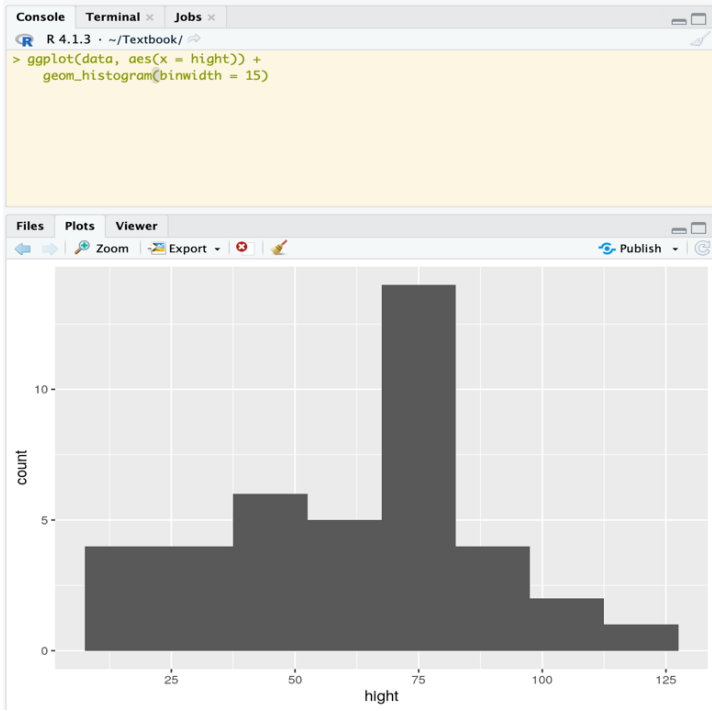


Резултатът от това е чудесна графика на данните с медианата и разпределението на данните за височината по групи. Вмъкнахме няколко детайла за персонализиране в слоя geom_point(). Можете да промените размера, цвета и прозрачността (alpha) на всички точки в този слой.

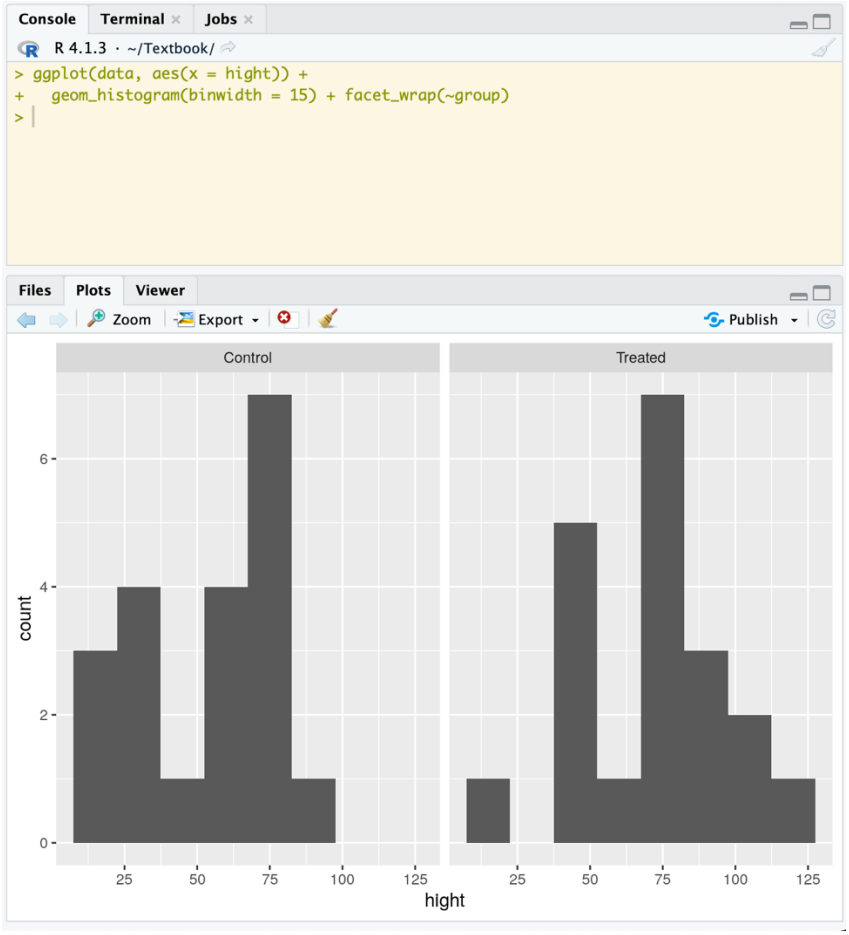
Изследването на разпределението на данните в нашите променливи е изключително важно. Можем да получим идея за формата на разпределението, за централната тенденция, за разпространението и дали може да има някои доста екстремни стойности.

Визуализацията на разпределения с помощта на ggplot() включва geom_histogram(). Това изисква да помислим малко за това какво представлява хистограмата. Данните от хистограмата са изобразени като правоъгълници, които представляват отделните части на данните, без да се припокриват и показват определени популации според относителното разпределение.

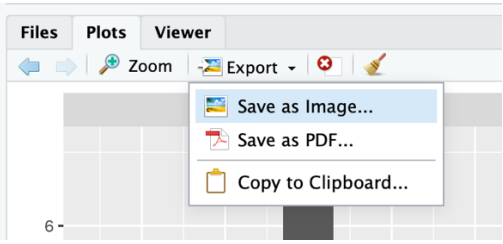
Това означава, че R прави някои неща вместо вас - произвежда оста *y*, а не вие! От гледна точка на ggplot() това става в аргумента на aes(), там има само една променлива:



Използваме възможността тук, чрез хистограмите, да представим още един инструмент в ggplot2. Това е много удобна функционалност за данни, които са групирани. Тези групи се наричат фасети. Фасетирането или решетката е за създаване на матрица от графики, автоматично структурирана от променлива фактор (категория) във вашите данни. Имайте предвид, че този трик работи за почти всички типове графики в пакета с инструменти на ggplot2. Но ние ще го демонстрираме тук с хистограмата. Изисква се да посочи специална функция на фигурата facet_wrap(), за да направим фасетите. За да демонстрираме, ние вземаме хистограмата на данните за височина и позволяваме на ggplot() да раздели данните по групи, осигурявайки автоматично две хистограми. Важното, което трябва да забележите при facet_wrap() е, че символът „~“ предхожда променливата за групиране.



След като приключим с добавянето на слоеве към графиката и с нейното персонализиране и тема, ние можем да запишем и свалим нашата графика от RStudio в различни формати, например растерна графика или PDF, от бутона Export в панела за плотове:



Как да направим нашите графики още по-добри?

Научихме основните неща, с които можем да създадем основни типове графики, както и някои методи, за да ги персонализираме. Но все пак това е само върхът на айсберга от възможности, които ggplot2 може да ни предостави.

Нека да направим отново нашия скатърплот и да го присъединим в обект. Функцията ggplot() връща обект графика, която ние можем да запазим. Така в последствие ние можем да добавяме всеки слой към нашата графика, като ги добавяме към обекта:

С добавяне само на един ред код eg_scatter + ggtitle("Заглавие") ние можем да сложим заглавие на нашата графика. Дори можем да направим това на кирилица!

Следващото важно нещо за нашите графики е мащабът на диаграмата. Функциите scale_() са дълбоко обвързани с променливите, които визуализираме. ggplot2, както всички добри пакети за графики, задава някои настройки по подразбиране. Разбира се, можем да променим това. Нека например коригираме мащаба на осите, като използваме границите като аргумент. В някой случай ще искаме дори да се средоточим в определен диапазон:

```
8
9  scatter + ggtitle("Заглавие")
10 scatter + scale_x_continuous(limits = c(30, 90))
11
```

Още повече можем да използвате функционалността на scales_(), за да log-трансформираме нашата ос. Това ще е полезно в някои случаи , за да подчертаем

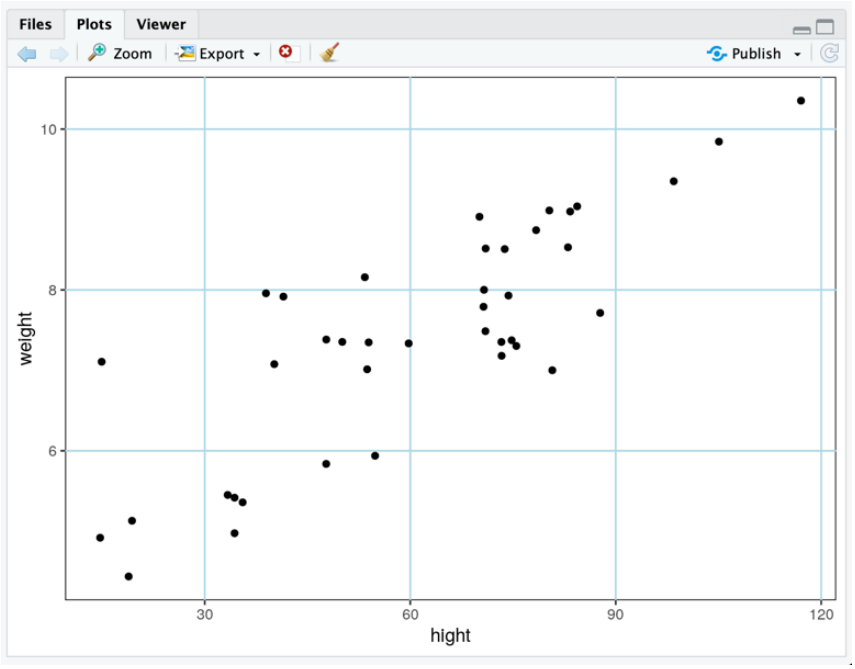
степента на вариация. Можем да направим това „в движение“ с ggplot2. Използваме графиката, което създадохме, и генерираме оста log-y. Използваме аргумента „trans =“ в scale_y_continuous(). Със *trans* = може да направите много видове трансформации към едната или и двете оси x и y на графиката:

```
scatter +
  scale_y_continuous(breaks = seq(from = 10, to = 150, by = 20), trans = "log10")
```

И сега да поговорим още за персонализирането на графиките чрез т.нар. теми в ggplot2. Функцията theme() е много мощно средство за коригиране на всички неестетични части на графиката и създаване, както на красиви, така и направо нелепи фигури. Така че, е добре да се запознаем с тази функция, за да се гордеем по-скоро с нашите графики, отколкото обратното ☺. Трябва да знаем, че има т.нар. готови теми, които можем да използваме. Например theme_bw(). Има осем теми по подразбиране, включени в ggplot2, които могат да бъдат извикани чрез съответните функции: theme_gray, theme_bw, theme_linedraw, theme_light, theme_dark, theme_minimal, theme_classic и theme_void (вижте http://ggplot2/referencedy/verse. ggtheme.html за повече информация).

Нека разгледаме синтаксиса на няколко елемента на темата, с които можем да променим изгледа на нашата графика. Например нека да видим как да се отървем от сивия фон и да променим линиите на мрежата във фона, като генерираме светлосини такива:

```
9
10 scatter +
11   theme(
12   panel.background = element_rect(fill = NA, colour = "black"),
13   panel.grid.minor = element_blank(),
14   panel.grid.major = element_line(colour = "lightblue")
15 )
16
```



Да разгледаме по-подробно тези редове код, за да си обясним как персонализирахме графиката:

• Панелната група от елементи на темата имат логически имена, като „фон“ и „решетка“, съответстващи на ясни характеристики на фигурата.

• element_() указва аргументи за геометричната композиция на панела, като „rect“ за „правоъгълник“ или „line“ за „линии“. Така очертаваме нашата графика с черен правоъгълник.

• Всеки от тези аргументи element_() може да бъде персонализиран с вече познати аргументи като fill = или colour =.

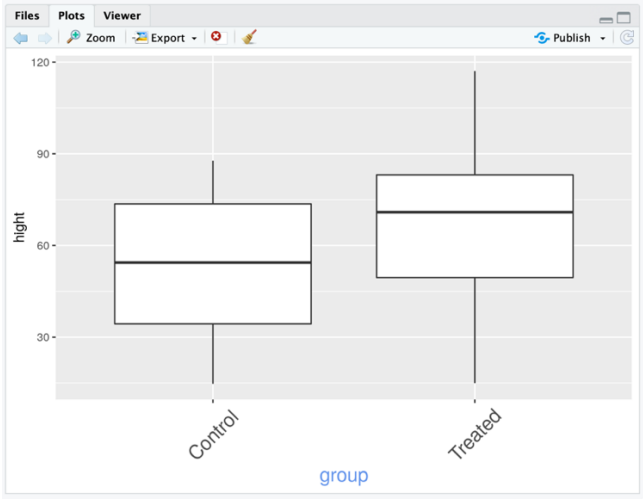
• element_blank() указва дадения елемент да не се генерира изобщо, тоест така можем да го премахнем.

Ако искаме да променим начина, по който са форматирани отметките на осите, трябва да манипулираме темата, като зададем атрибути на компонентите axis.title() и axis.text().

Например ето как можем да регулираме цвета и размера на заглавието на оста X и ъгъла на етикетите за отметка на оста X. Правим това с примера за боксплот, където завъртането може да бъде супер удобно:

```
Script.R* | data | tidy_table | table
Source on Save | Run | Source
1  library(ggplot2)
2  library(readr)
3  data <- read_csv("data.csv")
4
5  box <- ggplot(data, aes(x = group, y = hight)) + geom_boxplot()
6
7  box + theme(
8          axis.title.x = element_text(colour = "cornflowerblue", size = 15),
9          axis.text.x = element_text(angle = 45, size = 15, vjust = 0.5))
10
```

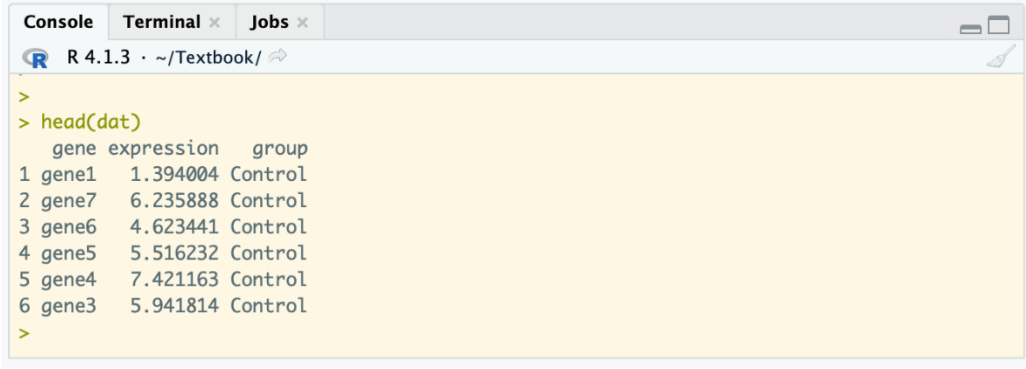
Още веднъж можем да видим, че този клас атрибути axis() сме използвали за манипулирането на текстовите елементи (element_text()) и те приемат няколко познати аргумента. Например size = може да приеме абсолютен размер (например 15). Освен това отбелязваме аргумента vjust =, който приема стойности между 0 и 1, за да коригираме вертикално етикетите, което често е необходимо, когато въртим етикетите под определен ъгъл. И естествено с angle= 45 да завъртим нашите етикети на 45 градуса.

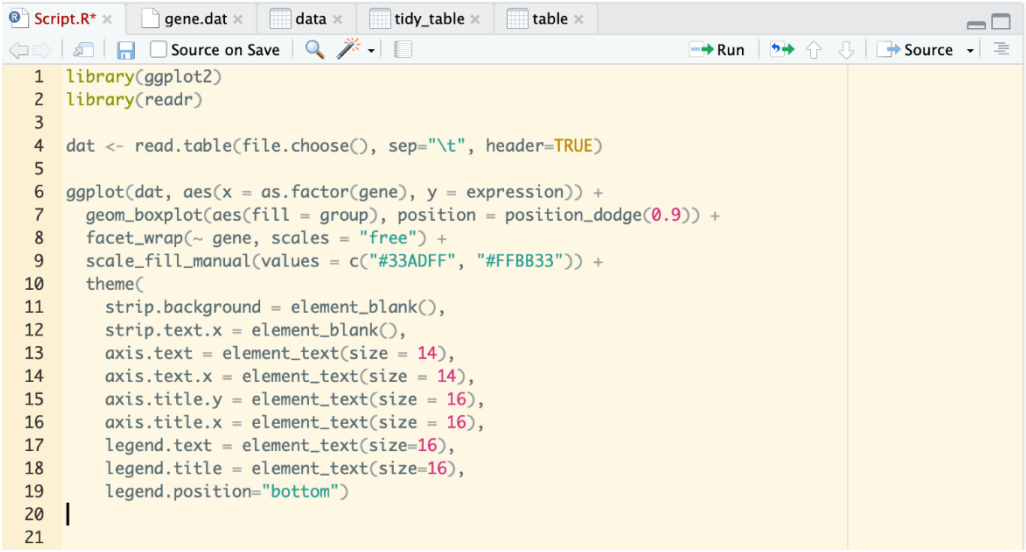


Можете да научите повече за пакета ggplot2, както и много от аргументите, с които може да персонализирате всички елементи на графиката ви, от неговият сайт: https://ggplot2.tidyverse.org/.

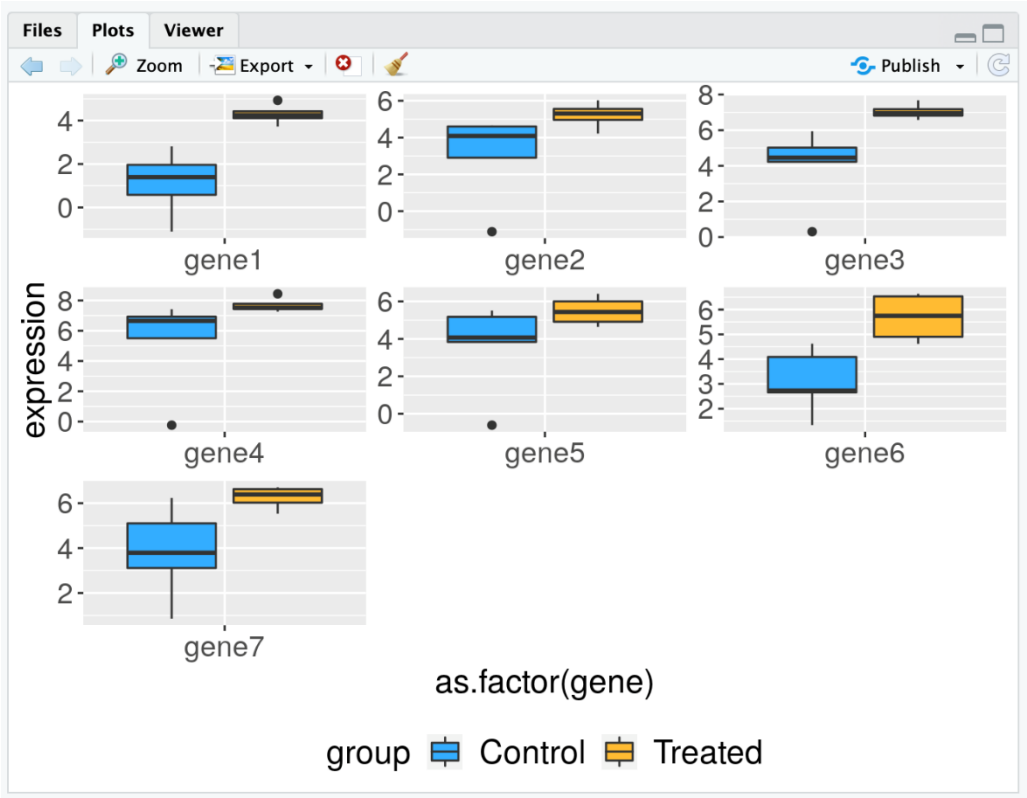
Пример от практиката

Нека да разгледаме сега един пример от практиката с построяване на графика. Това е важно, защото, когато виждате конкретни примери, ще можете по-лесно да запомните някои елементи от кода и синтаксиса им в практиката. Често в молекулярната биология ще имате например данни за експресията на дадени гени в контролна група и в третирана група и ще искате да видите, дали има разлика в експресията. Тази задача можем да направим с вече познатия ви тип графика боксплот. Ето как изглеждат нашите данни (таблица с колони – име на ген, експресия и група (фактор). Чудесно – нашите данни са подредени и са готови за използване от ggplot2 и не е нужно да трансформираме нашата таблица:



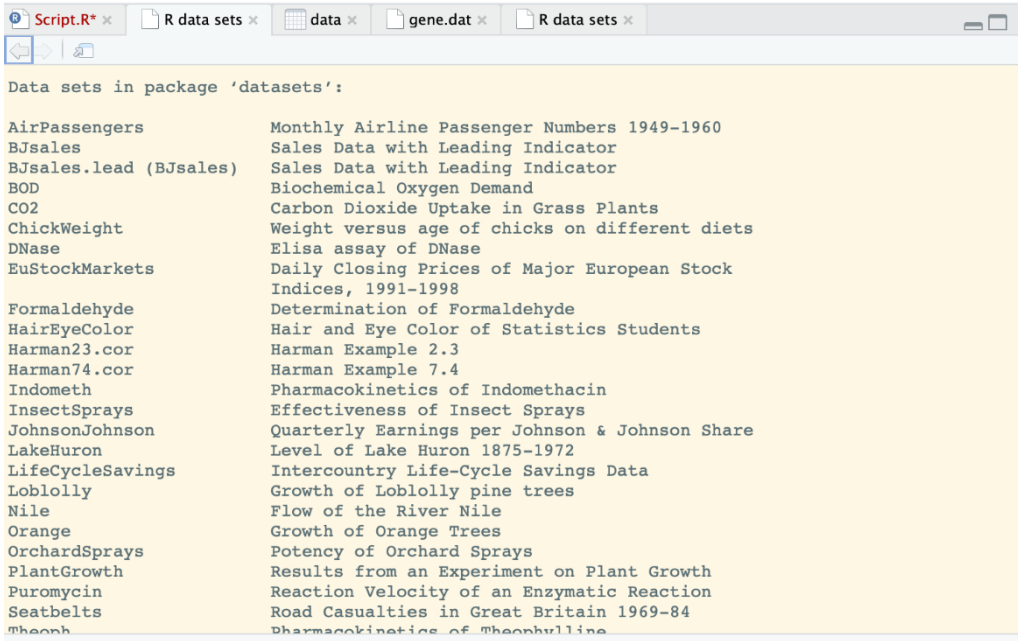


Нека да разгледаме кода, за да изясним какво прави всяка команда. С ggplot() създаваме нашата графика, като казваме, че данните в променливата gene искаме да бъдат фактор (защото ще искаме да фасетираме нашата графика и да направим малки графики за всеки ген). На у оста ще са данните с експресията на гена. Вече знаем как да направим фасетирането на графиката с facet_warp(). Задаваме и нашия машаб да е свободен, а не фиксиран с scale = “free”. След това задаваме ръчно цветове, как да бъдат оцветени нашите правоъгълници от боксплота. Използваме theme(), за да зададем и големина на шрифта на текста на повечето елементи, както и да изчистим елементите, които не искаме да се генерират с element_blank(). И резултатът от нашия код е това. Имаме чудесна фасетирана графика за всеки ген и как са разпределени данните за експресията в двете групи:



Вградени данни в R

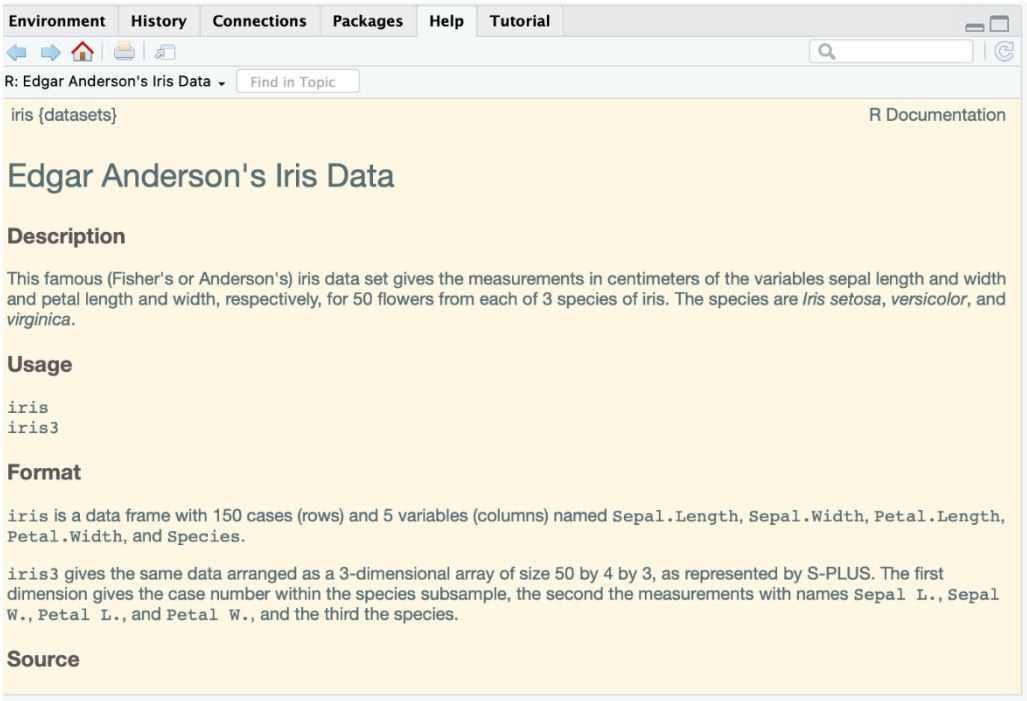
Когато се упражняваме с R, изключително важно е да имаме чисти и качествени данни, за да можем да усвоим добре материала и основните методи за обработка и визуализация на данни с R. Тук трябва да споменем нещо, която ще ни помогне – R съдържа вградени примери на данни от много области, които можем да използваме наготово (в пакет datasets). С функцията data() в конзолата можем да видим списък на примерните данни в R. С нея можем да видим и още примерни данни от пакетите, които сме заредили.



За да получим повече информация за дадени данни, можем в конзолата да напишем ?, следвана от името на примерните данни. Нека пробваме това с данните iris, като напишем в конзолата ?PlantGrowth, ще видим информация за данните и че те са свързани с резултати от експеримент за сравняване на добивите (измерени чрез изсушено тегло на растенията), получени при контрола и две различни условия на третиране.

Още видове графики!

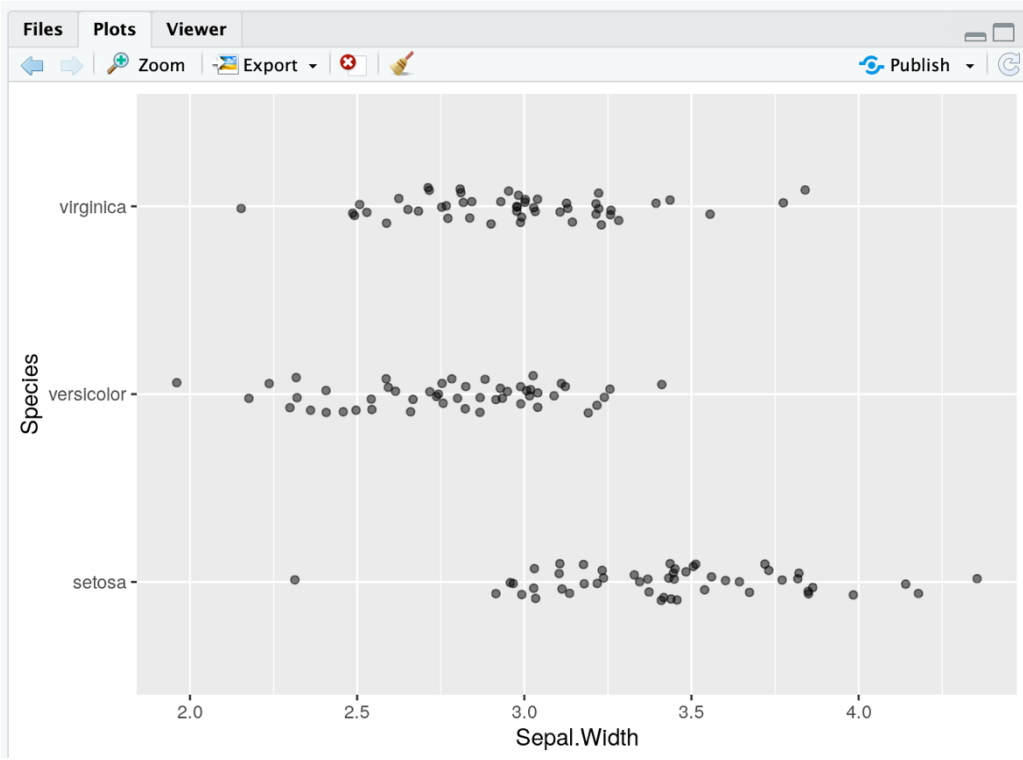
За да илюстрираме още от възможностите на ggplot2, ще използваме набор от примерни данни, наречен iris. Наборът от данни за видовете ирис на Фишър е набор от данни, въведен от британския статистик и биолог Роналд Фишър в неговата статия от 1936 г. Понякога се нарича набор от данни за ириса на Андерсън, тъй като Едгар Андерсън събира данните, за да определи количествено морфологичните вариации на цвета на три вида ирис. Тези данни се състоят от четири морфометрични измервания за екземпляри на три различни вида ирис (*Iris setosa*, *I. versicolor* и *I. virginica*). Нека да използваме помощта на R, за да прочетем повече информация за този набор от данни (?iris). Ще ги използваме, за да онагледим възможностите за още видове графики в ggplot2.



```
Console  Terminal  Jobs
R 4.1.3 · ~/Textbook/
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
>
```

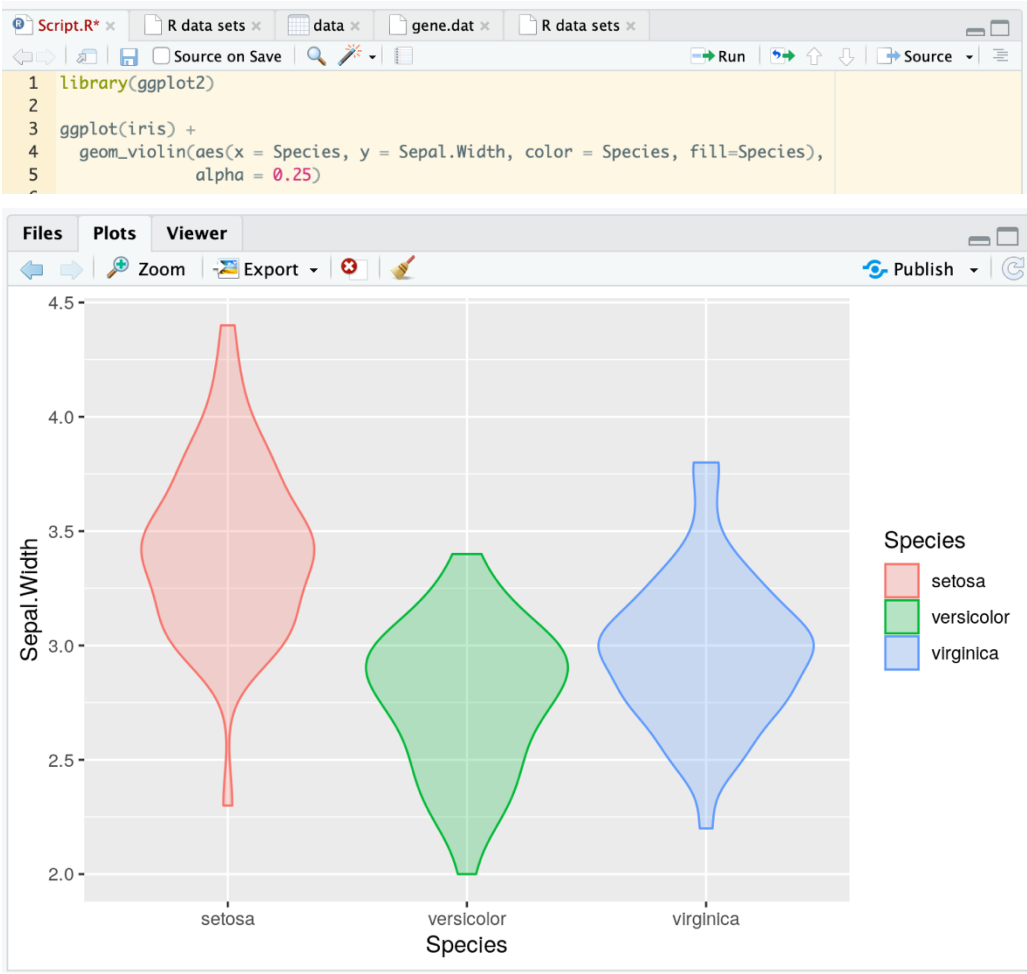
Нека да огледаме как изглеждат данните за Sepal.With чрез geom_jitter():

```
Script.R*  R data sets  data  gene.dat  R data sets
Source on Save   Run   Source
1 library(ggplot2)
2
3 ggplot(data = iris) +
4   geom_jitter(aes(x = Sepal.Width, y = Species),
5               width=0.05, height=0.1, alpha=0.5)
```



Използваме променливата за вид (Species), за да разделим данните, все пак имаме такива за три вида. Можем да добавим цвят, а също така да завъртим нашата графика на 90 градуса, понякога това е нужно, за да имаме по-добър поглед чрез coord_flip():

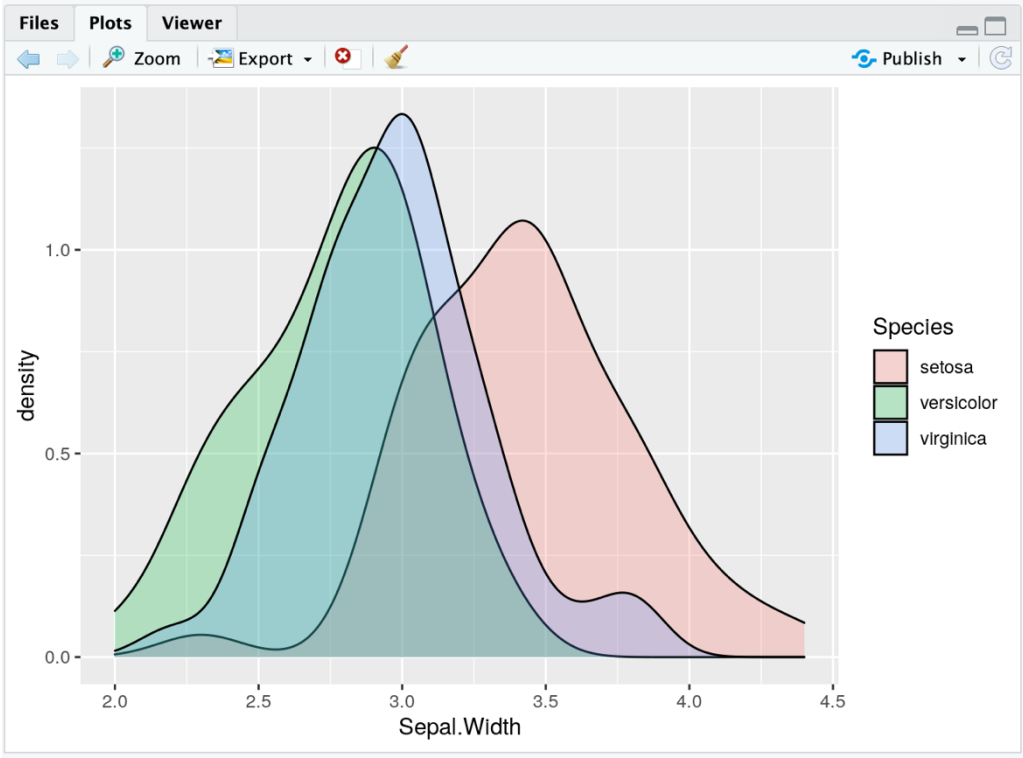
Нека да изпробване нов тип графика, т.нар. графика тип виола, чрез geom_violin():



Диаграмите *виола* позволяват да се визуализира разпределението на числова променлива за една или няколко групи. Той е наистина близо до боксплот, но позволява по-задълбочено разбиране на разпространението. Този вид визуализация са особено полезни, когато количеството данни е огромно и показването на индивидуални наблюдения става невъзможно.

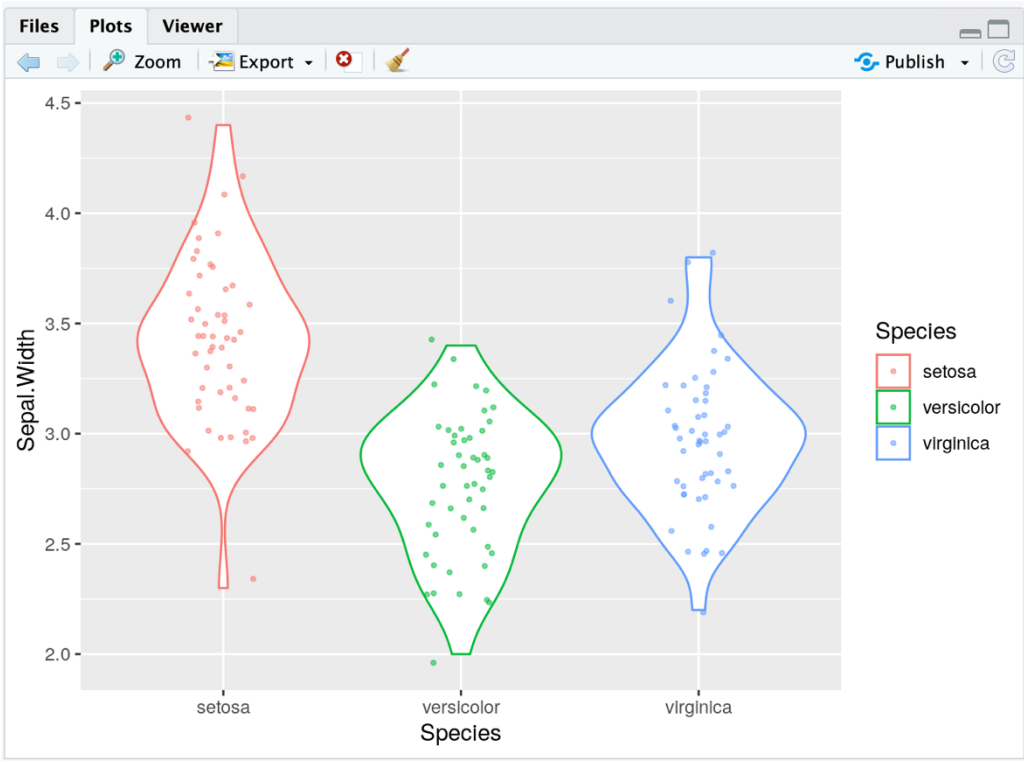
Друг тип графика, подобен на хистограмите, са графиките за плътност. Може да мислите за тях като за по-плавни и красиви хистограми с geom_density():

```
library(ggplot2)

ggplot(iris) +
  geom_density(aes(x = Sepal.Width, fill = Species), alpha=0.25)

```



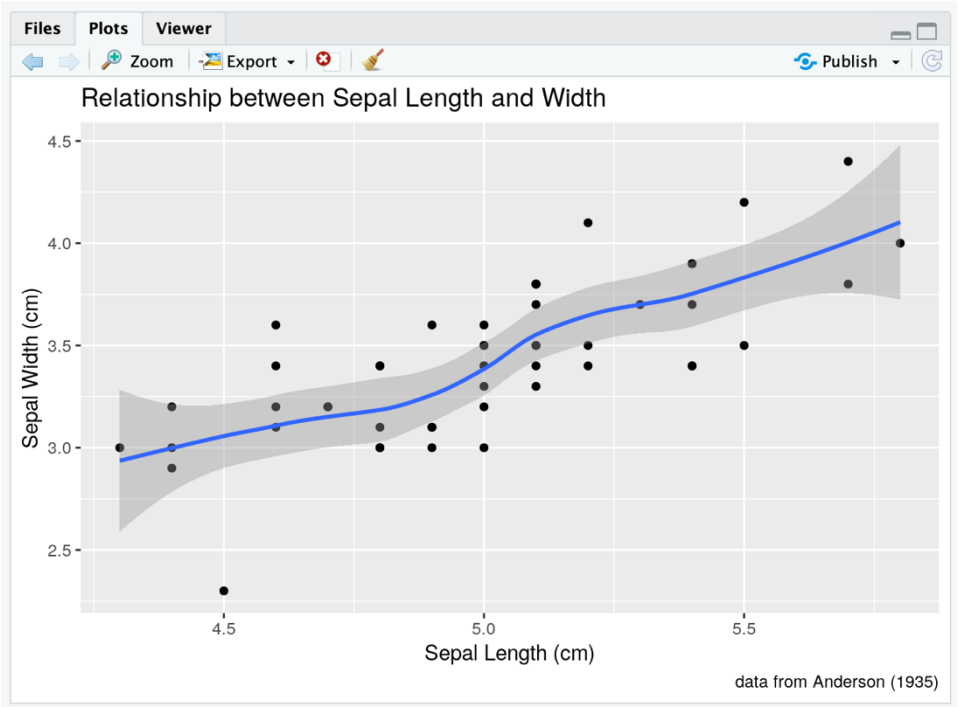
Тук нека отново повторим, че можем да запазваме всеки слой на графиката в обект и така можем да имаме два типа визуализация, като съберем двата слоя в една графика:

```
library(ggplot2)

myplot <- ggplot(iris, mapping = aes(x = Species, y = Sepal.Width, color = Species))

violin <- geom_violin()
jitter <- geom_jitter(width=0.15, height=0.05, alpha=0.5, size=0.75)

myplot + violin + jitter

```

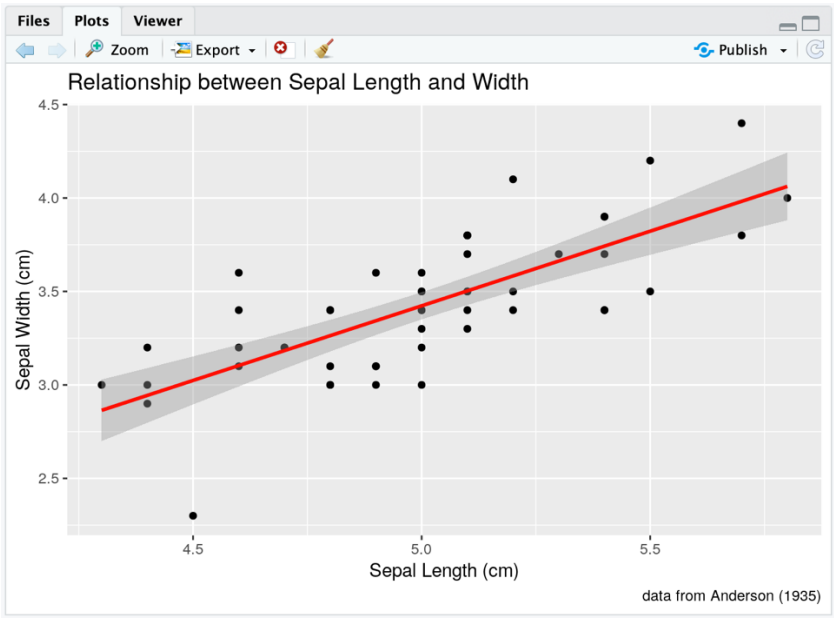


Да се върнем към скетърплота и да опитаме да му добавим още един компонент – линия на тенденцията (trend). Първо, нека се ограничим само с данните на единия вид ирис - *I. Setosa* чрез базова функция, наречена subset(). subset() ще върне подмножества, които отговарят на посочените условия. Нека запазим нашите данни за вида в нов обект setosa:

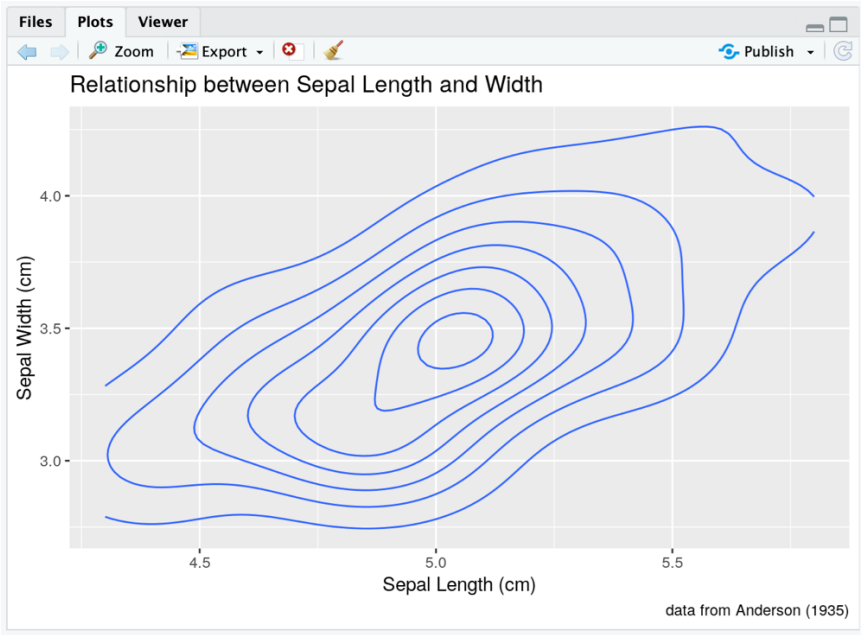
Нашият скатърплот ще илюстрира връзката между две променливи. Нека направим и отделен слой на графиката за етикетите. Със geom_smooth() добавяме лесно нашата линия на тенденцията, с която по-лесно можем да видим тенденцията на връзките между двете променливи.



Тази линия е построена по метод за нелинейна крива, ако вместо това искаме права линия на тенденцията, както обикновено се изобразява при модел на линейна регресия, можем да посочим различен статистически метод в аргументите geom_smooth(method = "lm", color = "red"):

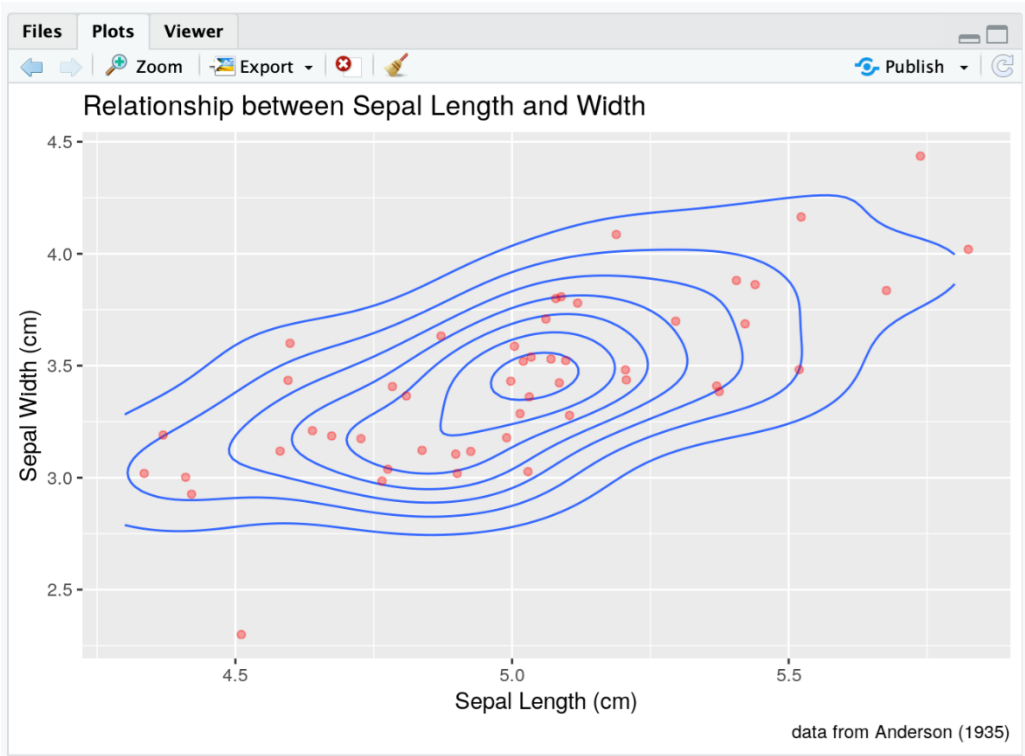


Помните ли графиката на плътността, която въведохме като визуализация за едномерни данни? Тя може да бъде разширена с двуизмерни такива. В 2D графика за плътност, вложените контури (или контури плюс цветове) показват региони с по-висока локална плътност. Нека илюстрираме това с пример, като използваме geom_density2d():



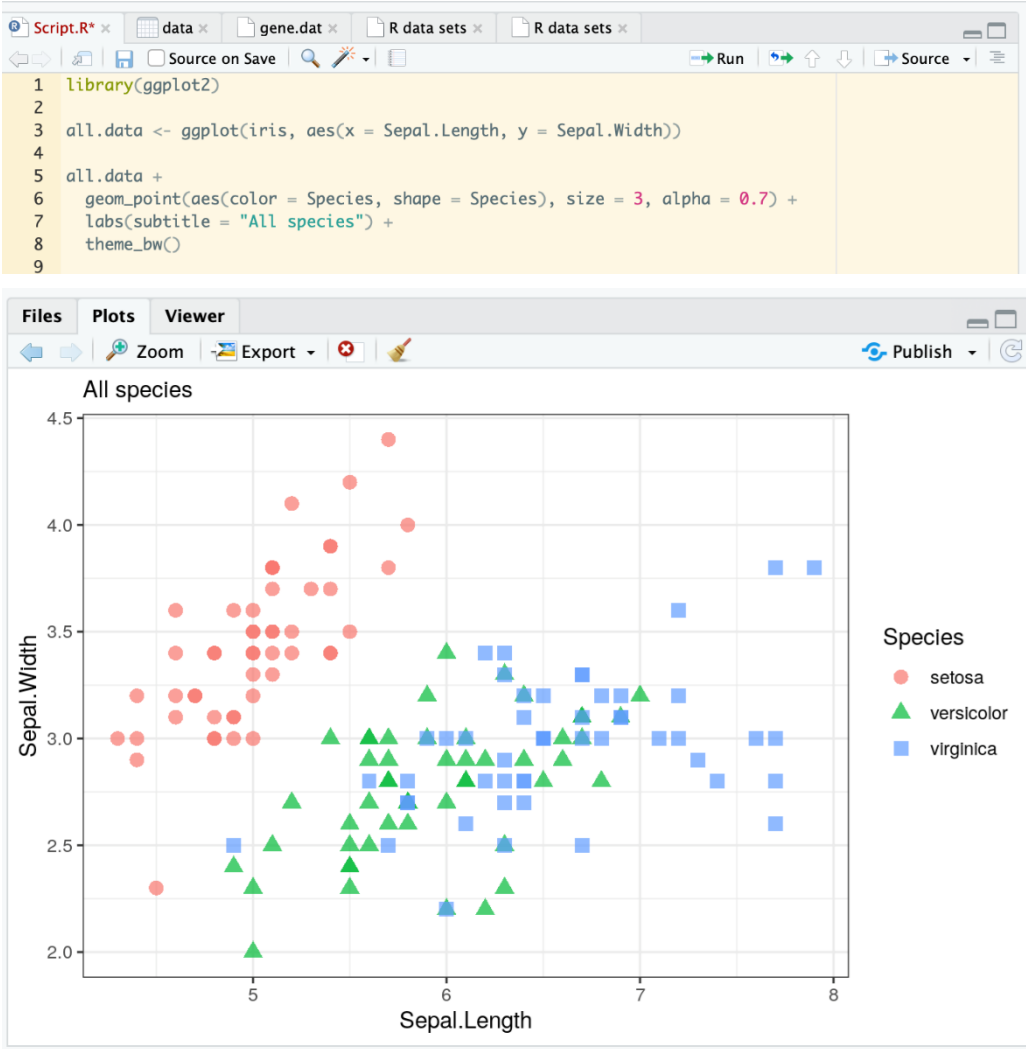
Да усложним още повече нашата графика на плътност, като добавим данните като точки в отделен слой (и да им дадем прозрачност и друг цвят):

```
library(ggplot2)

setosa <- subset(iris, Species == "setosa")
setosa.sepals <- ggplot(setosa,
                        mapping = aes(x = Sepal.Length, y = Sepal.Width))

labels <- labs(x = "Sepal Length (cm)", y = "Sepal Width (cm)",
                      title = "Relationship between Sepal Length and Width",
                      caption = "data from Anderson (1935)")

setosa.sepals  + geom_density2d() + geom_jitter(alpha=0.35, colour = "red") + labels
```



Така с комбинирането на различни типове графики и слоеве, в случаят скарътплот и плътност, можем в някои случаи да придобием по-ясна представа за данните или за тенденции, които искаме да анализираме.

Какво ще кажете сега да изобразим всички данни и да не се ограничаваме само до един вид ирис. Нека направим това като в geom_point() и aes() подаваме, че искаме цветът и формата на точките да съответстват на променливата Species, която съдържа 3 вида и така ще постигнем различната визуализация на данните:



Виждаме, че в скатърплота данните на 3-те вида се припокриват, и ни е трудно да ги отдиференцираме. Да направим фасетиране, за да имаме отделни скарътплотове за всеки вид, както и да добавим графика на плътност, ето сега вече можем да видим разликата в трите типа данни по-лесно:

```r
library(ggplot2)

all.data <- ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))

all.data +
  geom_point(aes(color = Species, shape = Species), size = 1, alpha = 0.7) +
  labs(subtitle = "All species") +
  geom_density_2d(aes(color = Species), alpha = 0.5) +
  facet_wrap(~ Species) +
  theme_bw() +
  theme(aspect.ratio = 1)
```

Понякога, когато визуализираме едни и същи данни, но използваме различни типове диаграми, може да искаме да ги направим в обща фигура. Това може да направим с пакета cowplot и неговата функция plot_grid(), с която да съединим няколко различни плота в една графика:

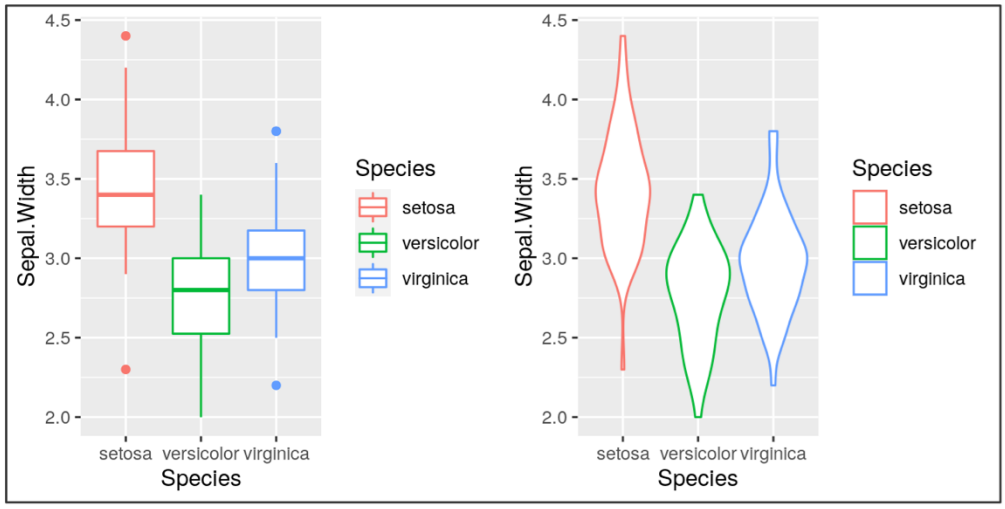
Script.R* | data | gene.dat | R data sets | R data sets

Source on Save | Run | Source

```
1  library(ggplot2)
2  library(cowplot)
3
4  myplot <- ggplot(iris,
5             mapping = aes(x = Species, y = Sepal.Width, color = Species))
6
7  plot1 <- myplot + geom_boxplot()
8  plot2 <- myplot + geom_violin()
9
10 plot_grid(plot1, plot2) + theme_bw() + theme(aspect.ratio = 0.5)
11
```

Files | Plots | Viewer

Zoom | Export | Publish



Глава 4

Статистика в R

Статистическите методи, използвани в биологията, помагат на учените да получат представа за процеси, които са или твърде обширни, или твърде многобройни, за да бъдат анализирани с други методи. Основната роля на статистиката в биологията е да тества хипотези. Въпреки това и други статистически методи се използват в биологията, за да помогнат за интерпретирането на научните резултати.

Някои статистически концепции могат да помогнат при избора на размера на извадката или кои организми да се изследват от група. Въпреки че може да изглежда, че изборът на субекти от група на случаен принцип би осигурил най-добрата група за анализ, произволните проби могат случайно да произведат модели, които не се срещат естествено извън групата на извадката. Също така много биологични експерименти изискват интерпретация на големи масиви от данни, които са твърде големи и твърде сложни, за да могат учените да анализират на ръка. По тези причини използването на статистиката в биологията е ключов момент за успешното анализиране на научните данни и резултати.

За извършване на много от статистическите методи, използвани в биологията, изследователите работят със специализирани инструменти и статистически софтуер, специално проектиран за обработка на данни.

R е език създаден от статистици. В него съществуват готови пакети и функции, които може да използвате, за да прилагате статистика, дори и да не сте запознат в дълбочина за статистическите методи и не сте статистик. R се смята за алтернатива с отворен-код на редица комерсиални статистически софтуери като SPSS, SAS, Stata, и др.

Знаем, че биолозите не са едни от най-големите фенове на математиката и статистиката, но с помощта на R ще видим, че това не е толкова страшно. Някои от основните преимущества, когато извършвате статистически анализи с R (и не само такива), са по-голямата повторяемост на анализа – вашият скрипт служи като рецепта за анализа и след време вие може да го повторите с точните параметри, които сте използвали и преди. А дори и да използвате готови

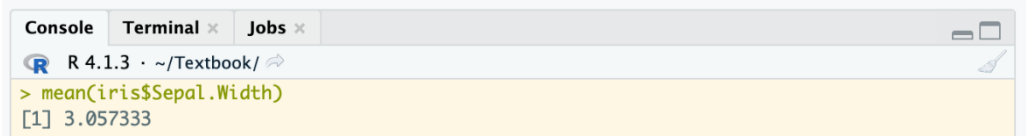
„рецепти“, които са споделяни в научната общност. Тук няма да се спираме на подробности за разясняването на статистическите похвати като методика, а по-скоро ще имаме за задача да видим как можем да ги извършим с помощта на R.

Дескриптивната статистика с R

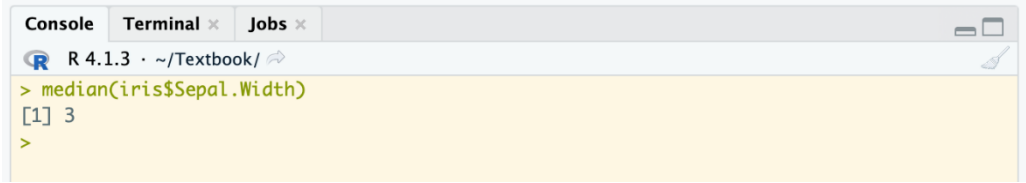
Дескриптивната статистика (или описателната статистика) често е първата стъпка и важна част от всеки статистически анализ. Позволява да се провери качеството на данните, като имаме по-ясен преглед върху тях. Дескриптивната статистика е добра отправна точка за по-нататъшни анализи. Съществуват много мерки за обобщаване на набор от данни.

В дескриптивната статистика съществуват показатели за местоположение, които дават разбиране за централната тенденция на данните, и такива за дисперсия, даващи разбиране за разпространението на данните. В R, както се досещате, имаме не само пълен набор от готови функции за аритметични изчисления, но и такива за изчисляване на редица от тези статистически показатели.

Нека се върнем отново към нашите данни за видовете ирис (iris). Първото нещо, за което много хора се сещат във връзка със статистиката, е т.нар. средна стойност. В R изчисляването на средната стойност е лесно. Всичко, от което се нуждаем, е функцията mean() и вектор от числа. В нашия пример въвеждаме mean(iris$Sepal.Width):

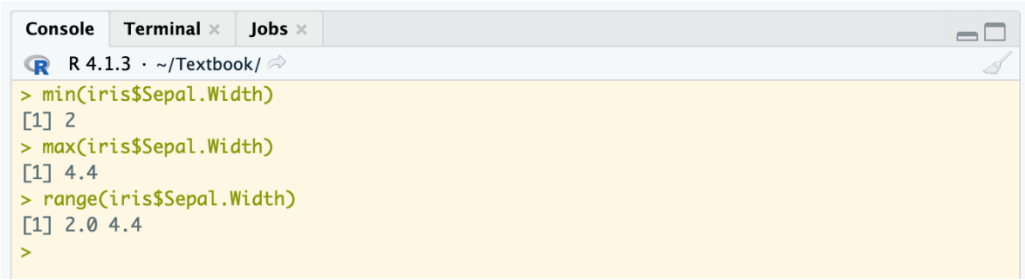


Припомняме за знака $, чрез него може да взимате отделни променливи от вашата таблица с данни (датафрейм). Освен средната стойност, друг общ статистически показател е медианата. В R получавате медиана с функцията, която сама говори за себе си - median().

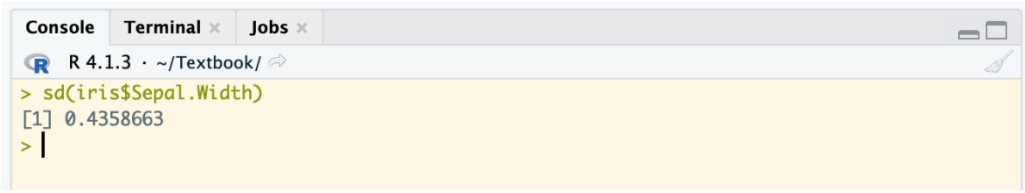


За конкретните данни за ириса няма голяма разлика между медианата и средната стойност. Това често е така (например при т.нар нормално или Гаусово разпределение). Но когато средната стойност и медианата са видимо различни, ще искаме да отчетем и двете стойности.

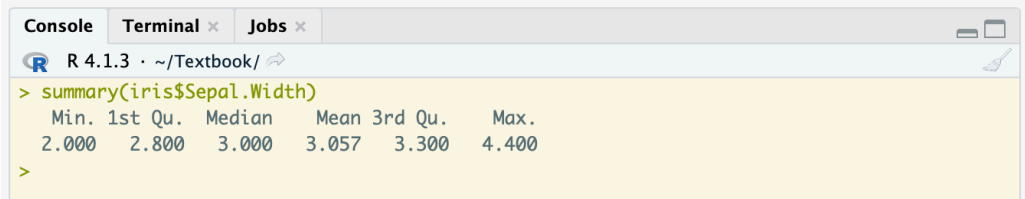
Понякога искаме и мярка за разпространението на данните. Например кои са минималната и максималната стойност? R връща минимума на вектор с функцията min(), а максимума - с max(). Можем да получим и двете стойности едновременно с range():



Докато range() ни казва крайностите в разпределението (минимум и максимум), стандартното отклонение дава по-добра картина за отклонението от средната стойност. В R изчислявате стандартното отклонение е с функцията sd(). Това е важно, когато например искаме да създаваме графики, в които да присъства този параметър:



Вместо да пишете функциите mean(), median(), range() една по една, можем да получим цялата тази информация с функцията summary().



Статистически модели и тестове с R

След като разгледахме нашите данни чрез дескриптивните показатели, както и начините да визуализираме данните чрез графика, ние можем да изведем някаква хипотеза, която да тестваме чрез статистически модел и това можем да направим в R. В биологията често експериментите включват сравняване на обекти и доказване на хипотези, с които можем да разкрием определени характеристики и правила в биологията. Така, след като докажем или отхвърлим нашата хипотеза чрез определени статистически методи и тестове, ние можем да направим надеждни заключения.

Когато става въпрос за сравняваме на средната стойност за две или повече групи, ANOVA (Анализ на дисперсията) и t-тестът са двете предпочитани най-добри практики. Въпреки че между двете има разлика. t-тестът се провежда, когато работим с две групи, когато има три или повече от три групи, трябва да използваме ANOVA тест.

Както t-тестът, така и ANOVA са статистическите методи за тестване на хипотеза. И двете предполагат следното:

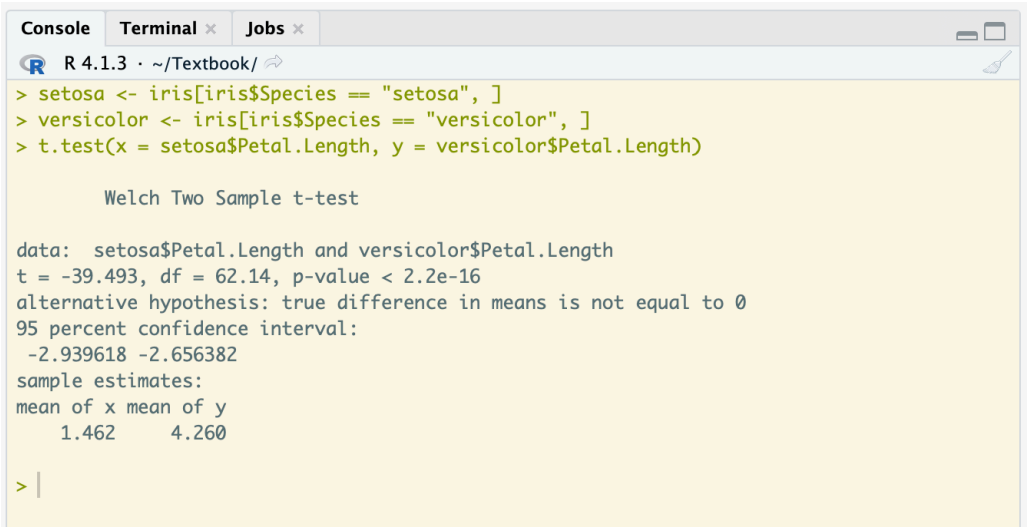
- Извадката, взета от популацията, има нормално разпределение;
- Хомогенна дисперсия;
- Извадка е от произволни данни;
- Наблюденията са независими;

Нулевата хипотеза е предположението по подразбиране, че не съществува разлика в средната стойност на групите.

Ще започнем, като направим едно-единствено сравнение, като разгледаме дължините на венчелистчетата на два вида (отново от данните iris). Използваме t-тест, за да попитаме дали стойностите за двата вида са различни. Само като

погледнем данните за два вида ириси, изглежда, че дължините на венчелистчетата са различни, но дали са значимо различни?

За да сравним две средни стойности на две групи, можем да направим с помощта t-test на Стюдънт. Ще започнем със сравняване на данните за *Iris setosa* и *Iris versicolor*, така че трябва да създадем два нови обекта с данни, единият съответстващ само на данните от *I. setosa* и един за *I. versicolor*:



Резултатът от t-теста е малко по-различен от това, което ще видим по-късно в друг статистически метод, който също е много използван в биологията - ANOVA. Резултатите включват редица параметри на теста като t-стойност на теста, степента на свобода, p-стойност (нивото на значимост на t-теста), интервал за достоверност на разликата в средните стойности между двата набора от данни.

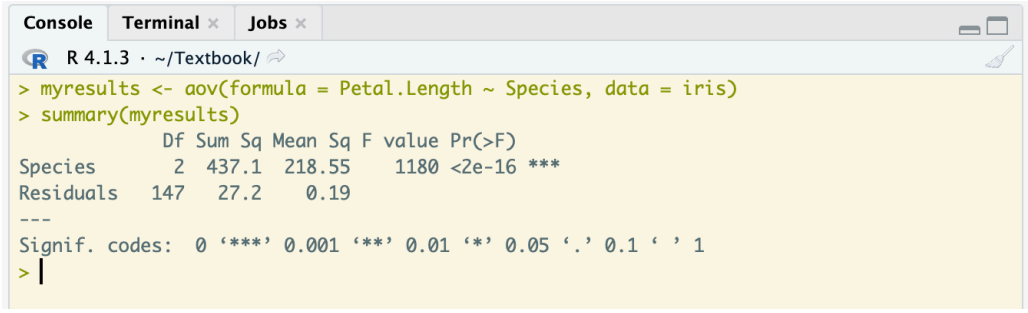
За да отхвърлим хипотезата, че при тези два вида стойността, която изследваме е еднаква (тоест разликата е значима), трябва да имаме p-стойност < 0.05. Виждаме че R ни дава стойност 2.2e-16 – какво означава тя? R тук използва т.нар. научна (експоненциална) нотация, която е равносилна на стойността 2.2×10^{-16}, или 0.0000000000000022. Е, това е доста по-малко от 0.05, нали? Така достигаме до извода, че разликите при тези два вида по този показател са значими.

В зависимост от сравняваните групи също така можем да имаме t-тест на независими или на двойка групи (unpaired и paired). При t-тест на двойка групи (свързани групи) се използва, за да се определи дали промяната в средните

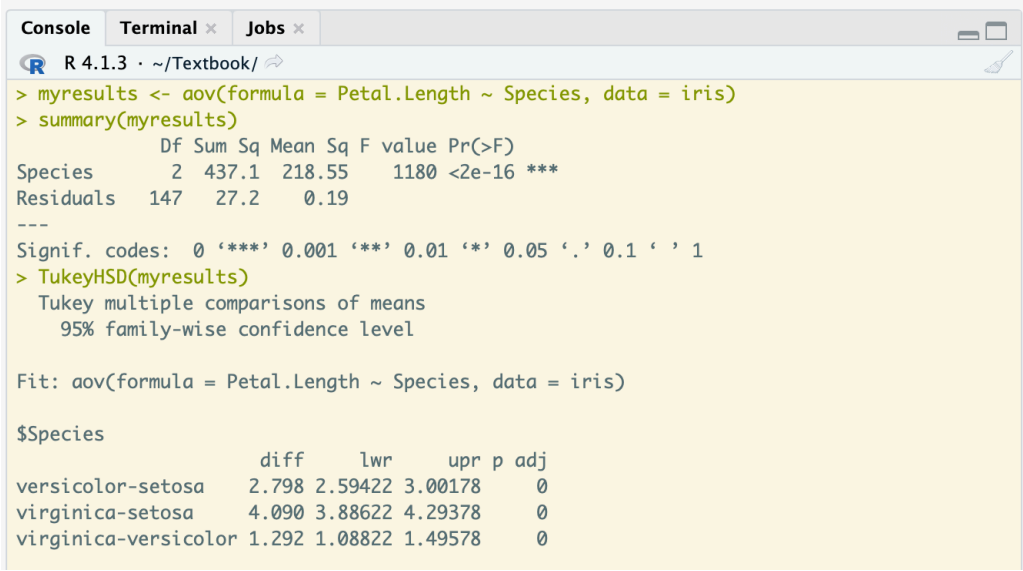
стойности между две сдвоени наблюдения е статистически значима. Например едни и същи субекти се измерват в две времеви точки или се наблюдават по два различни метода или когато едната група е извадка от другата. За да укажем на R, че искаме да направим t-тест на двойка групи, използваме аргумента paired = TRUE.

Но какво ще направим, ако имаме повече от 2 групи? Статистическа техника, използвана за сравняване на средните стойности между три или повече групи, е известна като ANOVA (или F-тест). При този тест, p-стойността показва, че има поне една двойка сред групите, в която средната разлика е статистически значима. Нужно е да се определи обаче на коя точно конкретна двойка се провеждат допълнителни тестове (post-hoc тестове). Има два основни типа ANOVA - еднофакторна (one-way) ANOVA (за тестване на една независима променлива) и двуфакторна (two-way) ANOVA (тества две независими променливи).

Нека да се опитаме да отговорим на въпроса има ли разлики в дължината на венчелистчетата между трите вида? Започваме с изграждането на модела на дисперсията с функцията aov(), като запазваме резултатите в нов обект и ползваме summary(), за да ги разпечатаме:



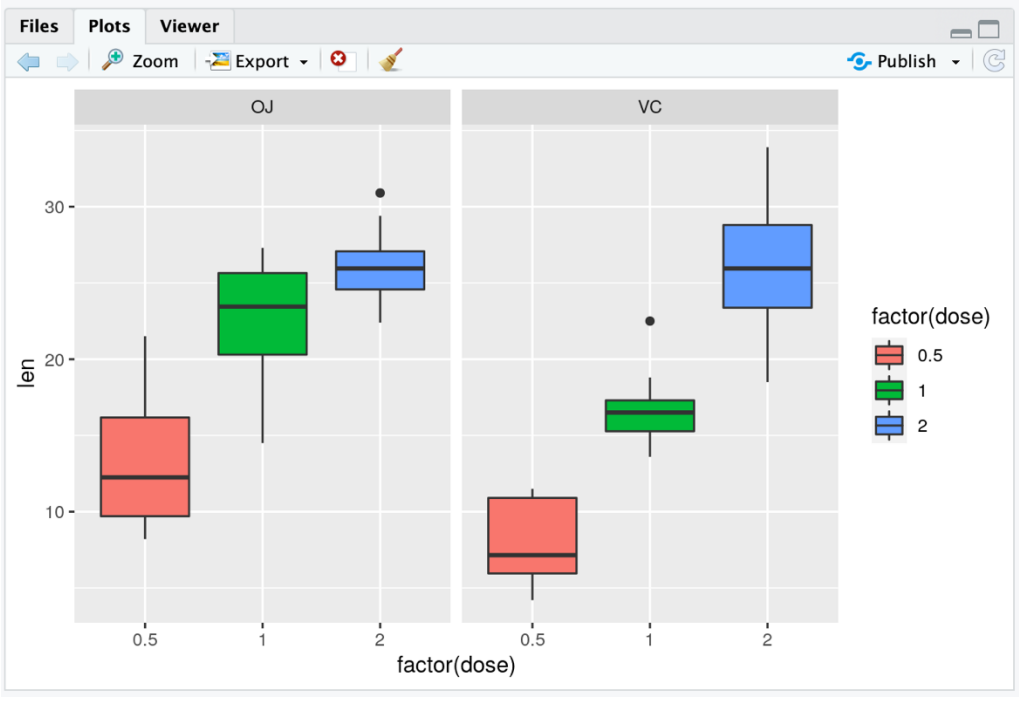
Видовете имат значително различна дължина на венчелистчетата (P <0.05). Ако обаче искаме да видим точно между кои видове е тази значима разлика, трябва да проведем последващ анализ ("post-hoc"), за да оценим как видовете са различни по между си, удачен тест за тази цел е т.нар. тест на Тюки с функцията TukeyHSD():



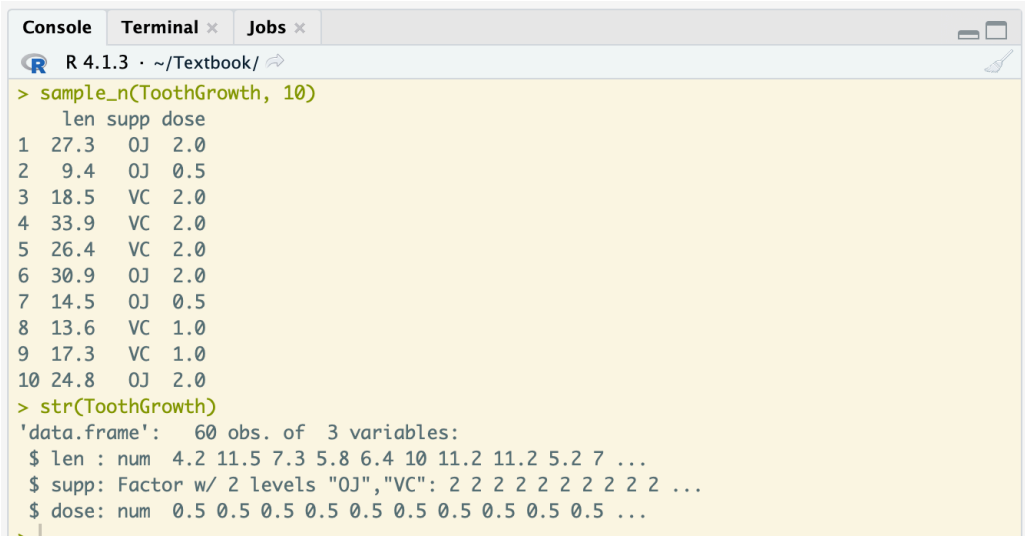
Така вече имаме индивидуални p-стойности, които показват дали разликите са значими между всека от 3-те двойки видове.

Нека направим и двуфакторен (two-way) ANOVA тест. Този път ще използваме други вградени данни, които са налични в R, защото ни трябват два фактора за анализа. Наборът от данни за ToothGrowth съдържа данни за ефекта на витамин C върху растежа на зъбите при морски свинчета. Данните от този експеримент са дължината на одонтобластите при 60 морски свинчета - 10 за всяка комбинация за дозата на витамин C (0,5, 1 и 2 mg), както и методът на доставяне (чрез портокалов сок „OJ“ или аскорбинова киселина „VC“). Нека разгледаме и визуализираме данните:

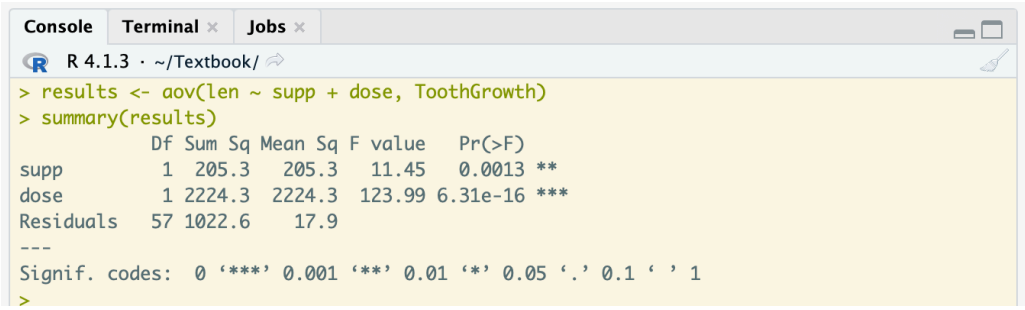
```
Console  Terminal  Jobs
R 4.1.3 · ~/Textbook/
   len supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5
4  5.8   VC  0.5
5  6.4   VC  0.5
6 10.0   VC  0.5
>
> ggplot(ToothGrowth, aes(x=factor(dose), y=len, fill=factor(dose))) +
+     geom_boxplot() +
+     facet_grid(. ~ supp)
```



Чрез sample_n() функция от dplyr можем да вземем произволни редове от данните, за да добием повече представа, както и за структурата им с вече познатата ви функция str():

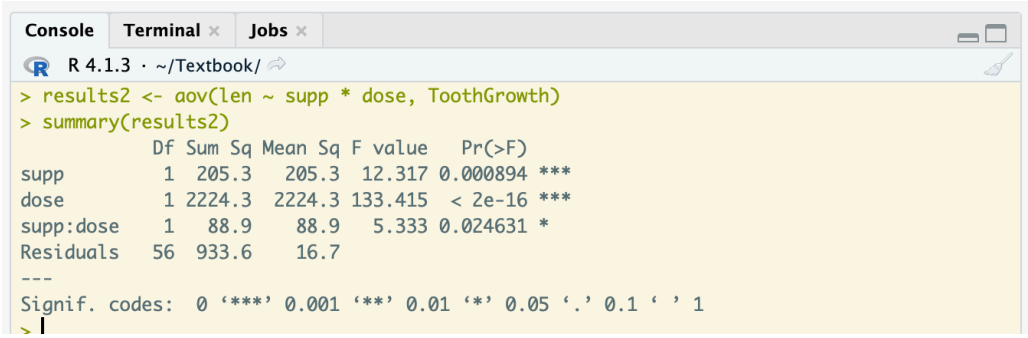


Нека сега проведем теста по двата фактора supp и dose:



От ANOVA резултатите можем да заключим, че както supp, така и dose факторите са статистически значими. Дозата е най-значимата факторна променлива. Тези резултати биха ни накарали да вярваме, че промяната на методите за доставка (supp) или дозата на витамин C ще повлияе значително на средната дължина на зъба.

Горният модел се нарича адитивен модел. Предполага се, че двете факторни променливи са независими. Ако смятате, че тези две променливи могат да взаимодействат, за да създадат синергичен ефект, заменете символа плюс (+) със звездичка (*):



Друг статистически метод е Тест на Пирсън (χ2-тест), чрез който можем да определим, дали има значима зависимост между две променливи. За този пример (отново ще използваме данните iris) ще тестваме в R, дали има връзка между променливите вида и размера на венчелистчетата. Тъй като имаме само една категория променлива, а тестът Хи-квадрат изисква две категории променливи, ние ще създадем нова категория променлива за размер, който съответства на малък (small), ако дължината на венчелистчето е по-малка от медианата, в противен случай размерът ще е голям (big).

След това създаваме таблица с наблюдавания брой случаи във всеки вид на новата променлива чрез table(). Тази функция изчислява броя на наблюдения на всяка комбинация от категориите:

```
Console  Terminal  Jobs
R 4.1.3 · ~/Textbook/
> table(dat$Species, dat$size)

            big small
  setosa      1    49
  versicolor 29    21
  virginica  47     3
```

За да илюстрираме създадената таблица с новите данни, правим барплот графика:

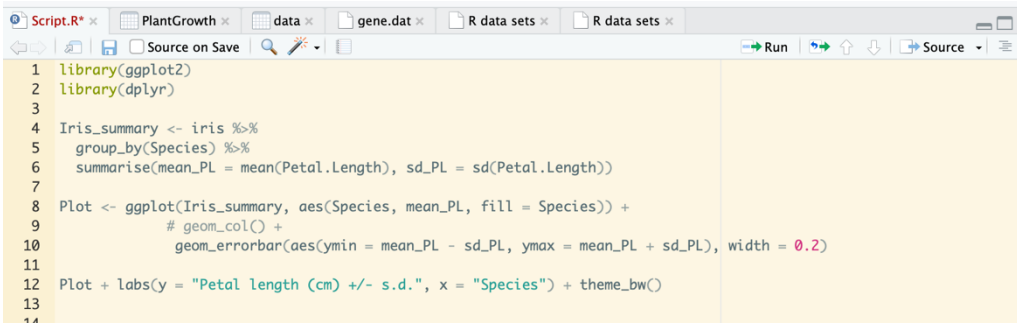
И накрая провеждаме и същинския хи-тест чрез chisq.test(), за да видим дали има връзка между големината на венчелистчетата и вида ирис:

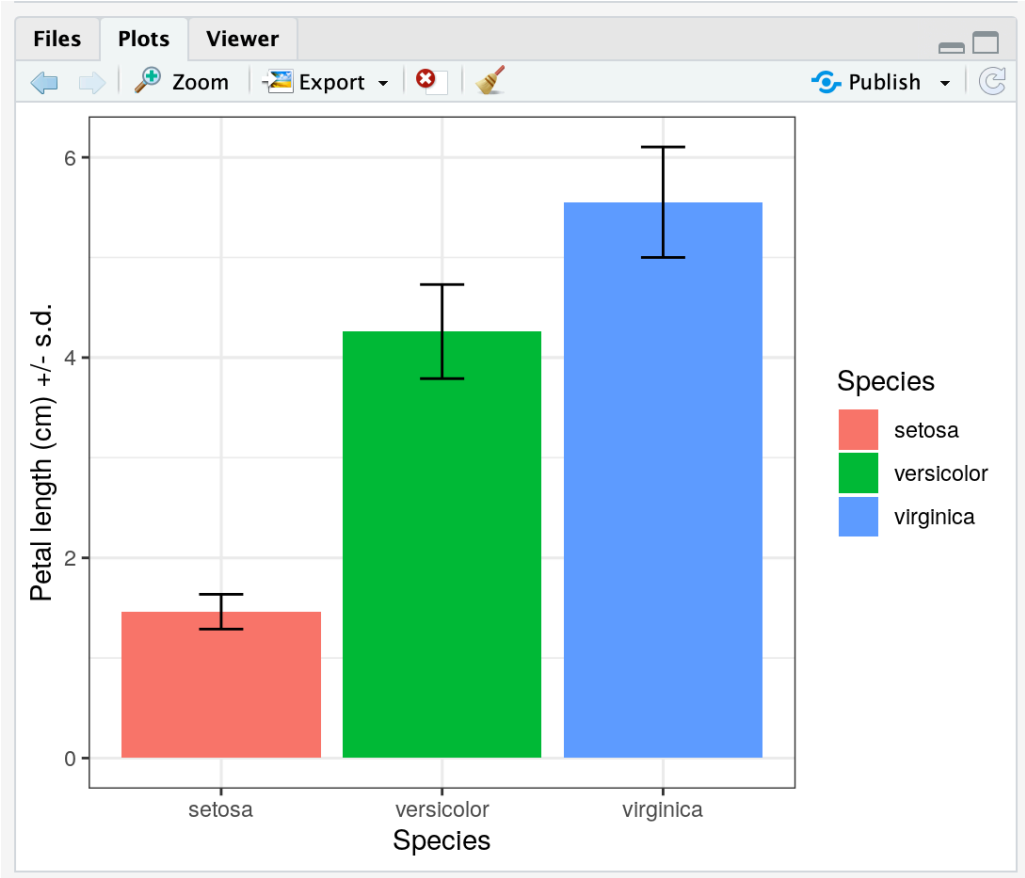
```
Console  Terminal  Jobs
R 4.1.3 · ~/Textbook/
> chisq.test(table(dat$Species, dat$size))

        Pearson's Chi-squared test

data:  table(dat$Species, dat$size)
X-squared = 86.035, df = 2, p-value < 2.2e-16
```

От нашия резултат отхвърлянето на нулевата хипотеза за хи-квадрат теста за независимост означава, че има значителна връзка между двете променливи - вид и размер.

Да комбинираме графиките и статистическите резултати.

След като научихме как да извършваме някои основни статистически методи, това ще ни помогне да направим и визуализациите на данни още по-точно. Представете си, че трябва да направите барплот графика и да отчетете стандартното отклонение. Ще използваме данните за ириса, за да направим барплот за дължината на чашелистчетата по вид със стандартното отклонение, като използваме функцията geom_errorbar():



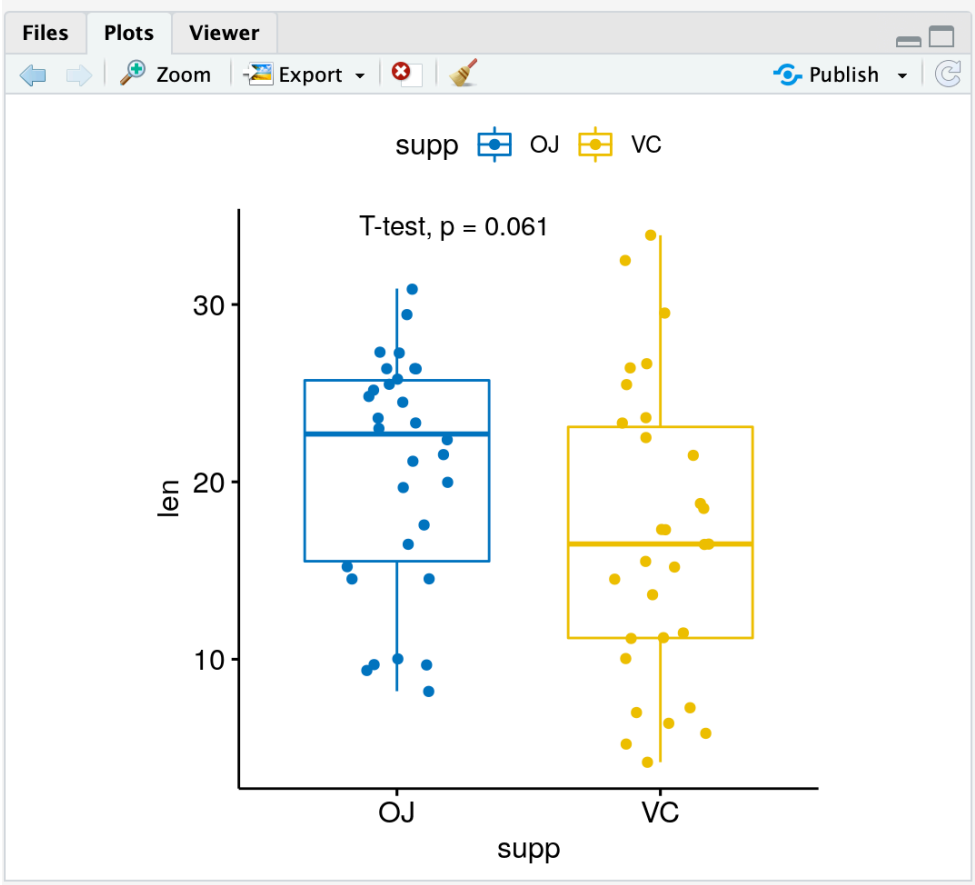


За да осъществим задачата, първо трябва да обобщим данните, за да получим средни стойности и стандартно отклонение за всеки от трите видове в набора от данни. Има няколко начина да направите това в R, но ние харесваме функциите summize() и group_by() от пакета dplyr, за които вече споменахме. Използваме geom_col() със стойността на променливата mean_PL като височина на стълбчетата. В аргументите на geom_errorbar(), ymin и ymax са горната и долната граница на елементите, визуализиращи отклоненията (дефинирани тук като средна стойност +/- sd), а width определя колко широки са лентите за грешки.

Когато говорихме за провеждане на статистически тестове и сравняване на различни параметри, в много от случаите ще ни се наложи да визуализираме графика на данните и да укажем резултата от преведения статистически метод. Така например, когато имаме боксплот или барплот графика, ние може да укажем

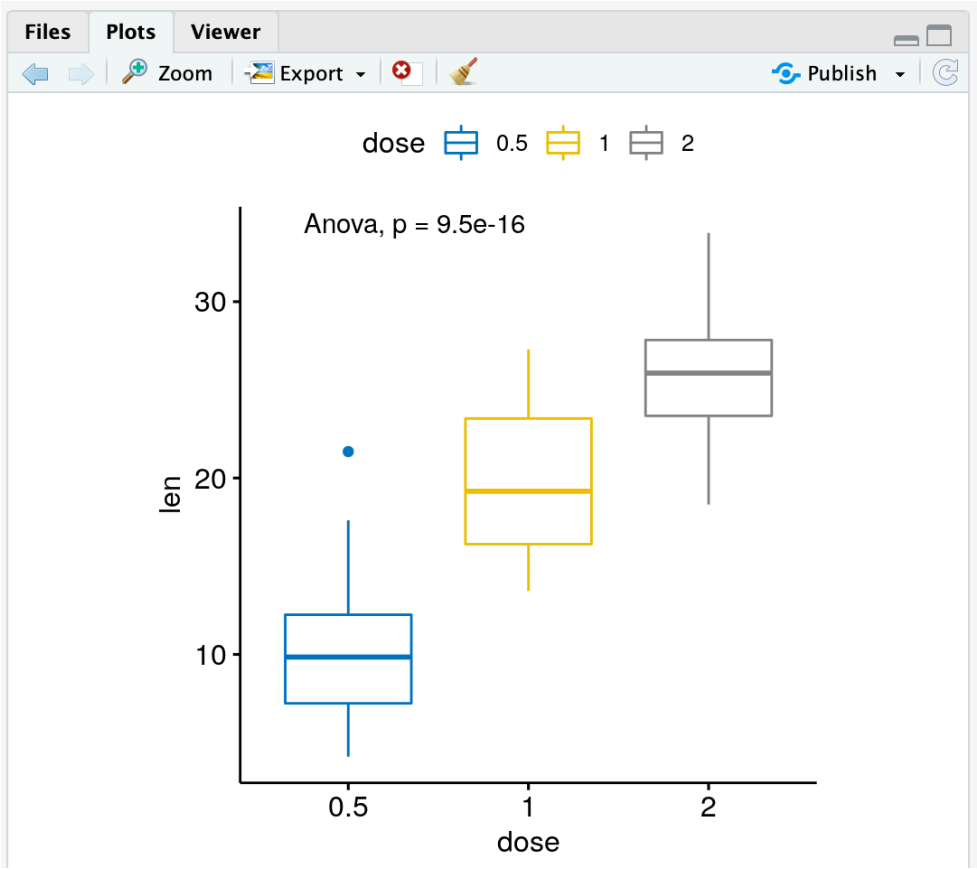
кои групи сме сравнявали, по кой метод и получената p-стойност. В ggplot2 има функции, които позволяват лесно вграждане на статистиката в графиките ни. Такива са compare_means() и stat_compare_means() от пакета ggpubr. За да илюстрираме това, нека използваме данните за морските свинчета.

```
Script.R* | data | gene.dat | R data sets | R data sets | PlantGrowth
Source on Save | Run | Source

1  library(ggplot2)
2  library(ggpubr)
3
4  plot <- ggboxplot(ToothGrowth, x = "supp", y = "len",
5                    color = "supp", palette = "jco",
6                    add = "jitter")
7
8  plot + stat_compare_means(method = "t.test") + theme(aspect.ratio=1)
9
```



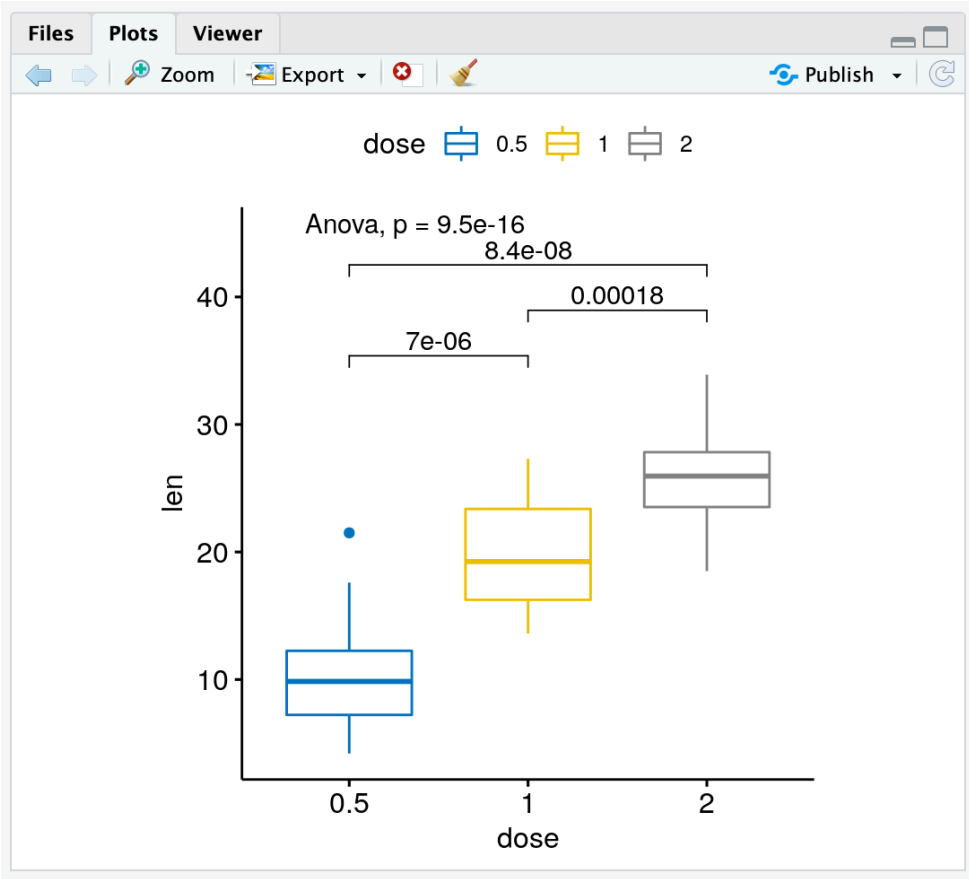
Ето и графиката с повече от една групи и метод ANOVA:

```
Script.R* × | data × | gene.dat × | R data sets × | R data sets × | PlantGrowth ×
Source on Save | Run | Source

1  library(ggplot2)
2  library(ggpubr)
3
4  ggboxplot(ToothGrowth, x = "dose", y = "len",
5            color = "dose", palette = "jco") +
6            stat_compare_means(method = "anova") +
7            theme(aspect.ratio = 1)
```



Когато имаме повече от една група, може да искаме да покажем и индивидуалните p-стойности, а не общата такава в графиката. Това може да постигнем по следния начин:

```
Script.R* | data | gene.dat | R data sets | R data sets | PlantGrowth
Source on Save | Run | Source

1  library(ggplot2)
2  library(ggpubr)
3
4  compare <- list( c("0.5", "1"), c("1", "2"), c("0.5", "2") )
5  ggboxplot(ToothGrowth, x = "dose", y = "len",
6            color = "dose", palette = "jco") +
7
8      stat_compare_means(comparisons = compare) +
9      stat_compare_means(method = "anova", label.y = 45) +
10     theme(aspect.ratio = 1)
11
```



Тук първо създаваме един списък с вектори за това, което искаме да сравняваме, и ще го използваме като аргумент в stst_compare_means(). Ако искаме вместо числова стойност на p-стойността, да имаме значимостта чрез „*“ , можем да използваме аргумента label = "p.signif". Като етикетите са следните „ns: p > 0.05“, „* : p <= 0.05“ „** : p <= 0.01“„*** : p <= 0.001“ „**** : p <= 0.0001“.

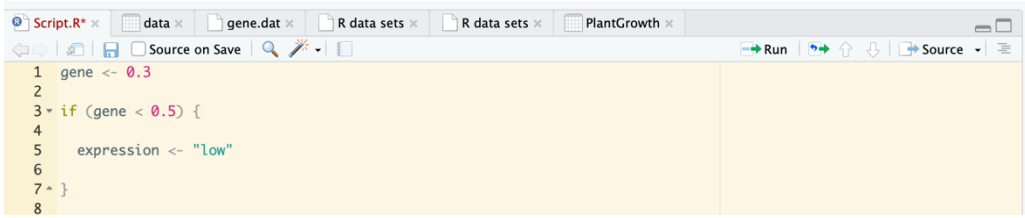
Глава 5

Базови изрази за контрол на потока в R.

Начините за контрол на потока в програмните езици проверяват как се изпълняват частите на кода. Те могат да се използват за извършване на команди само когато са изпълнени определени условия, както и за итерация през структури от данни, за повтаряне на команди, докато се случи определено събитие и т.н. Контролният поток често се използва при писане на функции или извършване на сложни трансформация на данни.

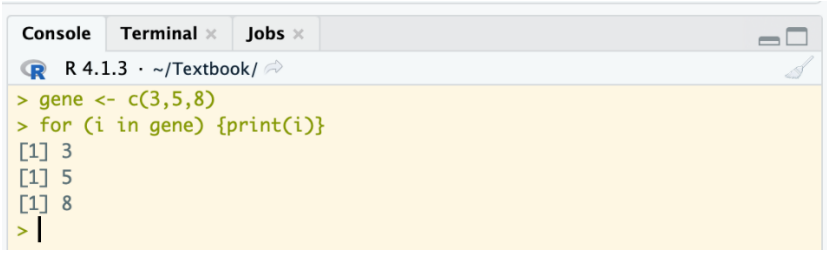
R не се различава от останалите програмни езици и командите за условия и цикли, както и техния синтаксис са много подобни – if, if-else, for, while и т.н.

Условия можем да създаваме с if, следвано от логическото условие в “( ), последвано от кода, който искаме да се изпълни в “{ }” , ако условието е изпълнено:



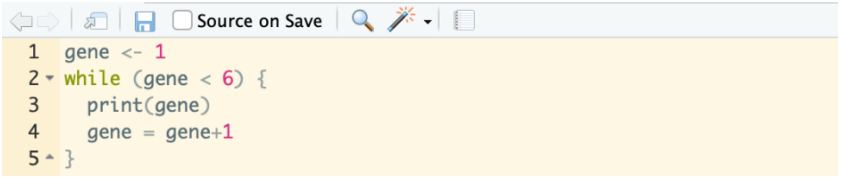
Можем да имаме множествено условие със if, if else, else:

Инструкцията for итерира елементите на обекти (като вектори или списъци). Обичайната употреба на for е да се извърши например изчисление за всеки елемент от обекта като „обходи“ обекта и извърши определен код с всеки един от елементите, кода, който ще се повтори при всяка итерация е ограден с „{ }”:



По този начин указваме с for да обходим числовия вектор gene, както и код който да бъде изпълнен всеки път за всеки елемент (в случая да бъде разпечатана стойността на елемента).

С while можем да направим цикъл, който да итерира толкова пъти, докато не се изпълни дадено условие:



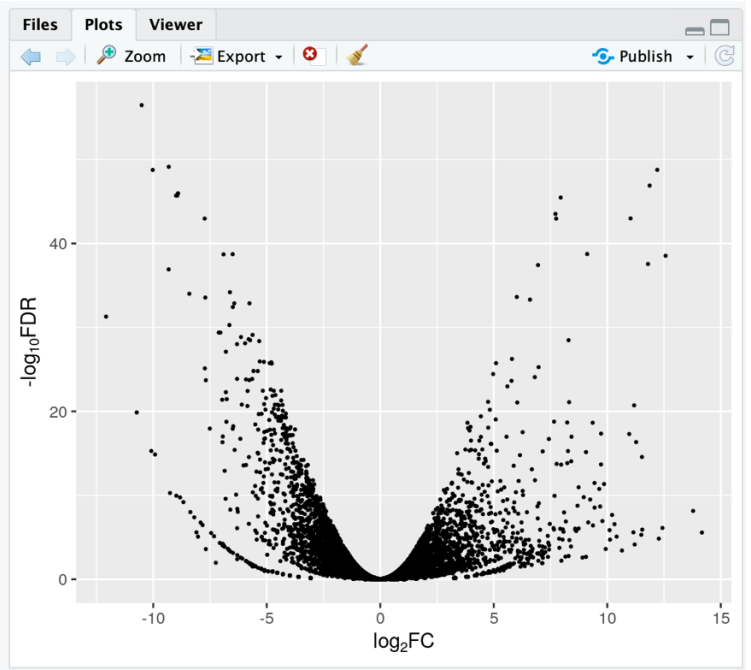
Съществуват и други начини за контрол на потока на кода, но това са едни от най-основните, които, както споменахме, съществуват и в други езици за програмиране. Освен това, в различни пакети съществуват готови по-сложни функции, чрез които можем да изпълняваме сложни условия и лесно да контролираме процеса на анализ на данни в нашия скрипт.

Още за контрола на потока.

Когато извършваме манипулиране на данни и изчисления в R, често ще ни се наложи да извършваме действия въз основа на някакви условия. И някои готови функции могат да ни помогнат с това. Например case_when() от пакета dplyr, представлява векторизиран if-else, с който можем да проверяваме условие и да изпълни код за всеки елемент на вектор или променлива в датафрейм. Това би ни спестило писане на код, който да направим чрез for и if-else.

Нека да направим пример със създаването на т.нар. графика вулкан. Тази графика е тип скатерплот и с нея може да визуализираме например данни за гени, които са диференциално експресирани, когато имаме изчисления за експресията на гени (например изчислени с пакета deseq2). Обикновено такива данни представляват датафрейм с променливи като log2FC, p-стойност и FDR за всеки ген. Такива данни можем да видим на адрес: https://raw.githubusercontent.com/sdgamboa/misc_datasets/master/L0_vs_L20.tsv

Можем да построим скатърплот по променливите logFC и FDR със стъпка на трансформиране на FDR със -log10, така в графиката колкото повече един елемент е по-високо на у оста, толкова по-значима е стойността:



Но в момента не можем да различим кои гени са с повишена експресия (up-regulated) и кои са с понижена (down-regulated). Гени с променена експресия се считат за такива, ако имат обикновенно |log2FC| > 2 и FDR < 0.05. Така че нека направим нашата графика малко по-информативна, като добавим цветове за променените гени. За да направим това, ще е необходимо да генерираме нова променлива в датафрейма „expression", която да показва дали генът е с повишена, понижена експресия или е не повлиян. И при тази стъпка използваме case_when(), за да направим множествени условия към всеки един елемент на log2FC и FDR и да дадем стойност на новата променлива за всеки един ген:

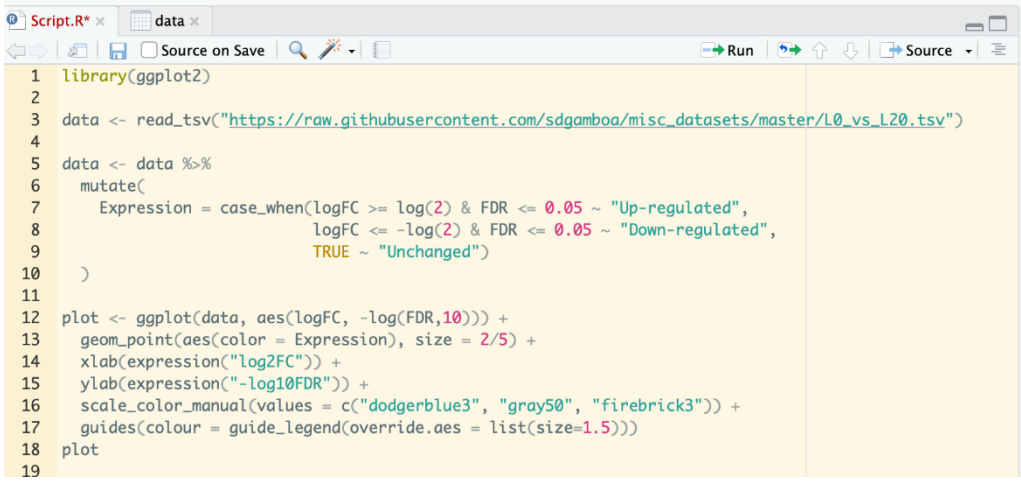
Script.R* × data ×

Source on Save | Run

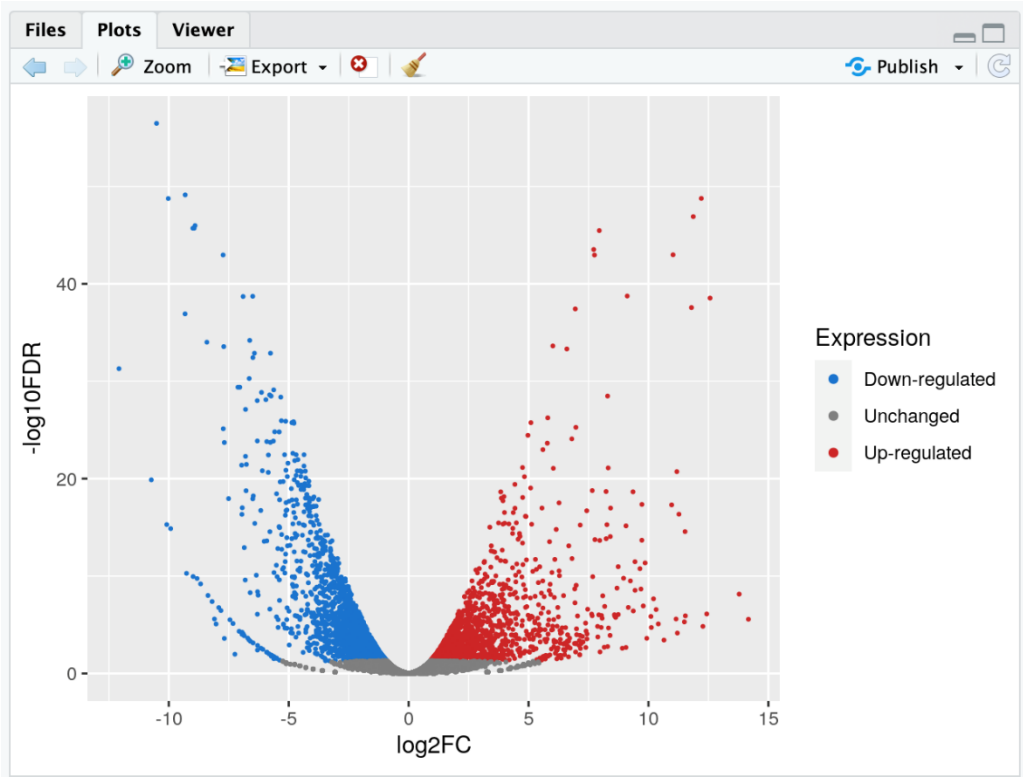
```
1  library(ggplot2)
2
3  data <- read_tsv("https://raw.githubusercontent.com/sdgamboa/misc_datasets/master/L0_vs_L20.tsv")
4
5  data <- data %>%
6    mutate(
7      Expression = case_when(logFC >= log(2) & FDR <= 0.05 ~ "Up-regulated",
8                             logFC <= -log(2) & FDR <= 0.05 ~ "Down-regulated",
9                             TRUE ~ "Unchanged")
10   )
11
```

Така изглеждат данните ни с новата променлива в датафрейма „Expression":

И нека генерираме новата графика с новите данни, като задаваме различен цвят на всеки ген в зависимост от променливата експресия:



Сега вече можем да различим отделните групи от гени, които са с повишена и понижена експресия визуално, или са останали непроменени. Освен това, дори можете, ако желаете, да сложите етикет на някои гени, от които се интересувате, или на топ 10-те гена, за да стане вашият скатърплот по-професионален, но това ще оставим на вас за домашно ☺.


Files
Plots
Viewer
Zoom
Export
Publish
-log10FDR
40
20
0
-10
-5
0
5
10
15
log2FC
Expression
Down-regulated
Unchanged
Up-regulated

Как да надградим знанията си?

Bioconductor

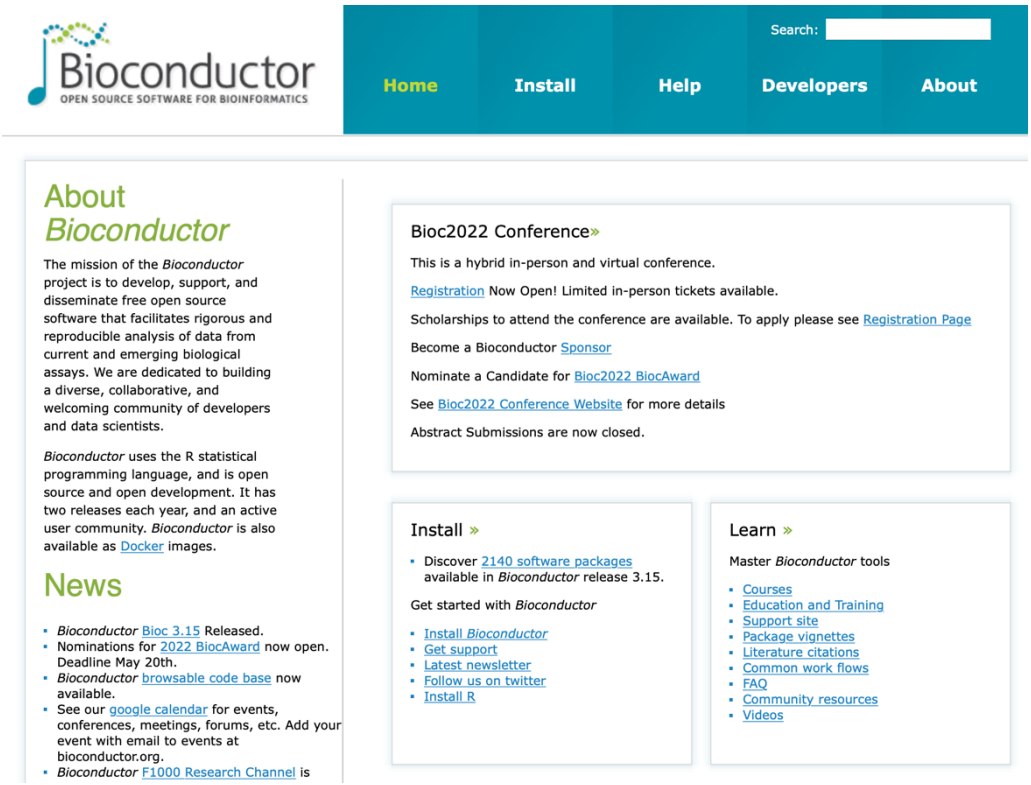
Какво е Bioconductor? Проектът Bioconductor (www.bioconductor.org/) стартира през есента на 2001 г. като инициатива за съвместно създаване на софтуер за изчислителна биология и биоинформатика. Проектът представлява набор от пакети и програми за анализ на биологични данни. От самото начало заявената мисия на проекта е да се разработят инструменти за статистически анализ и анализ на големи масиви от данни в строго и стабилно проектирани експерименти. Освен статистическите анализи, множество пакети поддържат интерпретация и анализ на резултати с биологичен контекст, визуализация и възпроизводимост (Фиг.9).

През годините софтуерните пакети, допринесли за проекта Bioconductor, отразяват еволюцията и подпомагането на някои широкомащабни технологии в молекулярната биология, от ДНК чиповете до секвенирането от ново поколение (NGS, например метагеномика, sRNA-seq, RNA-seq, ChIP-seq, DNA-seq). Някои от пакетите са фокусирани върху генетични анализи (например вариация на секвенциите, вариация на броя на копията, полиморфизми на единичени нуклеотиди (SNP)), а други помагат за анализа на данни в цитометрията, протеомиката, микроскопията и анализа на изображения. Например едни от най-популярните статистически пакети, които изчисляват диференциална експресия на гени от NGS са deseq2, edgeR, limma.

От решаващо значение е, че проектът не само има софтуерни пакети, внедряващи нови статистически тестове и методологии за специфични биологични проблеми, но също така създаде такива, предоставящи достъп до бази данни с молекулярни анотации и експериментални набори от данни. Тези пакети имат широк набор от приложения в редица биологични науки и днес Bioconductor версия 3.15 предоставя 2140 пакета:

Software:

- ResearchField (1022)
  - BiomedicalInformatics (74)
  - CellBiology (61)
  - Cheminformatics (16)
  - ComparativeGenomics (9)
  - Epigenetics (84)
  - Epitranscriptomics (2)
  - FunctionalGenomics (80)
  - Genetics (210)
  - ImmunoOncology (437)
  - Lipidomics (12)
  - MathematicalBiology (9)
  - Metabolomics (96)
  - Metagenomics (33)
  - Pharmacogenetics (14)
  - Pharmacogenomics (17)
  - Phylogenetics (9)
  - Proteomics (168)
  - StructuralPrediction (8)
  - SystemsBiology (99)
  - Transcriptomics (189)
- StatisticalMethod (800)
  - Bayesian (77)
  - Classification (160)
  - Clustering (269)
  - DecisionTree (8)
  - DimensionReduction (64)
  - FeatureExtraction (52)
  - GraphAndNetwork (143)
  - HiddenMarkovModel (17)
  - MultidimensionalScaling (1)
  - NeuralNetwork (5)
  - PatternLogic (3)
  - PrincipalComponent (46)
  - Regression (125)
  - StructuralEquationModels (1)
  - SupportVectorMachine (14)
  - Survival (31)
  - TimeCourse (61)



Фиг. 9. Страницата на Bioconductor (www.bioconductor.org/)

Литература:

1. Kabacoff, R. (2015). R in Action: Data Analysis and Graphics with R (2nd ed.). Manning.
2. Lander, J. P. (2017). R for Everyone: Advanced Analytics and Graphics (Addison-Wesley Data & Analytics Series) (2nd ed.). Addison-Wesley Professional.
3. Verzani, J. (2011). Getting Started with RStudio. Van Duuren Media.
4. Vries, D. A., & Meys, J. (2015). R For Dummies (2nd ed.). For Dummies.
5. Wickham, H., & Grolemund, G. (2017). R for Data Science: Import, Tidy, Transform, Visualize, and Model Data (1st ed.). O'Reilly Media.
6. Statistical Computing for Biologists (https://bio723-class.github.io/Bio723-book/index.html)
7. R for Biology Data Science (https://jmhulbert.github.io/r/bio/s1/)
8. Introduction to R for Biologists (https://melbournebioinformatics.github.io/r-intro-biologists/intro_r_biologists.html)
9. Starting with data (https://datacarpentry.org/R-ecology-lesson/02-starting-with-data.html)
10. Introduction to R Statistics (https://jcoliver.github.io/learn-r/002-intro-stats.html)
11. Descriptive statistics in R (https://statsandr.com/blog/descriptive-statistics-in-r/)
12. Visualising of volcano plots in with R (https://samdsblog.netlify.app/post/visualizing-volcano-plots-in-r/)

13. Mishra P, Singh U, Pandey CM, Mishra P, Pandey G. Application of student's t-test, analysis of variance, and covariance. Ann Card Anaesth. 2019;22(4):407-411. doi:10.4103/aca.ACA_94_19