R programming

Dr. Umesh Ghoshdastider
ETH Zurich

R

Visualization

Open source

Data science

Platform agnostic

Computational statistics

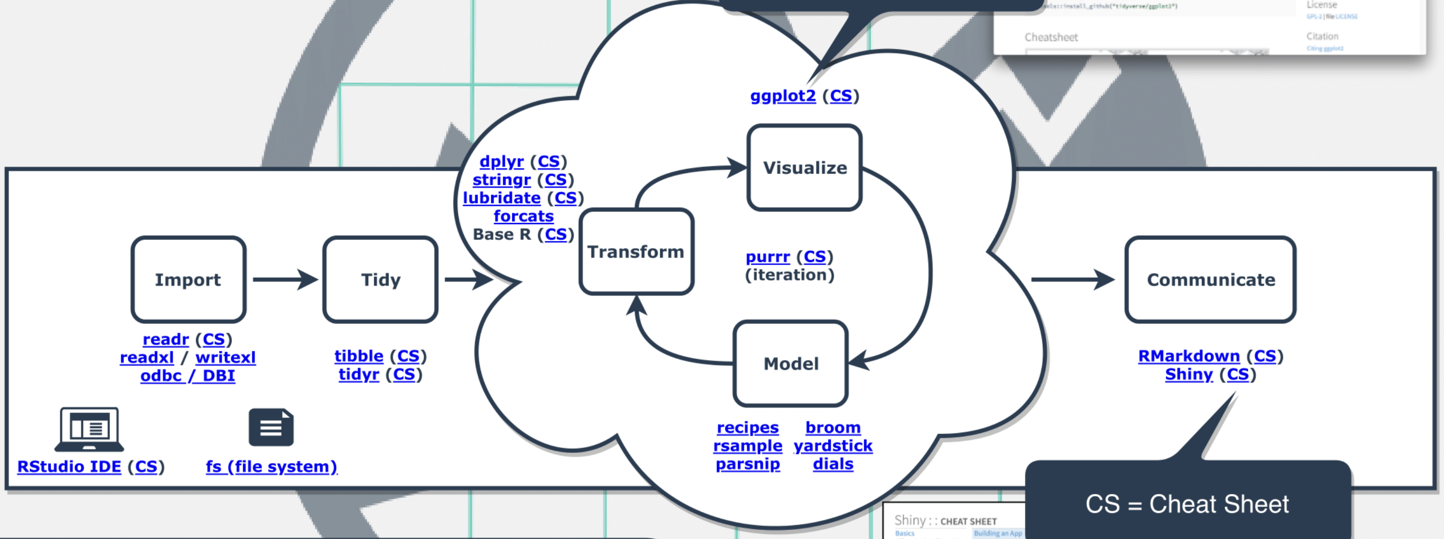
R

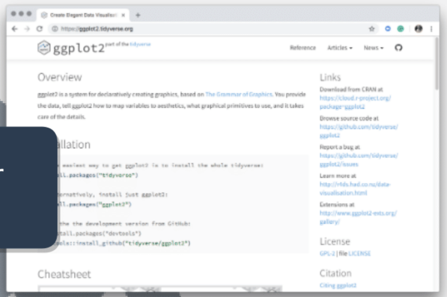
R is a powerful, extensible environment. It has a wide range of statistics and general data analysis and visualization capabilities.

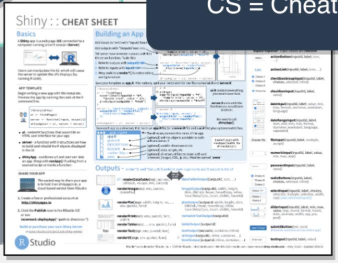
Data handling, wrangling, and storage
Wide array of statistical methods and graphical techniques available
Easy to install on any platform and use (and it’s free!)
Open source with a large and growing community of peers

Data Science with R Workflow

The Data Science With R Workflow is available in the book: R For Data Science. If you want to learn R and this workflow *for business*, take the **R For Business Analysis (DS4B 101-R) course** through Business Science University.







Important Resources

- **R For Data Science Book:** http://r4ds.had.co.nz/
- **Rmarkdown Book:** https://bookdown.org/yihui/rmarkdown/
- **Data Visualization Book**: https://rkabacoff.github.io/datavis/
- **More Cheatsheets:** https://www.rstudio.com/resources/cheatsheets/
- **tidyverse packages:** https://www.tidyverse.org/
- **Connecting to databases:** https://db.rstudio.com/
- **RMarkdown website:** https://rmarkdown.rstudio.com/
- **Shiny web applications website:** http://shiny.rstudio.com/
- **Jenny Bryan's purrr tutorial:** https://jennybryan.org/

"Business Science University: Enterprise-Grade Data Science Education"

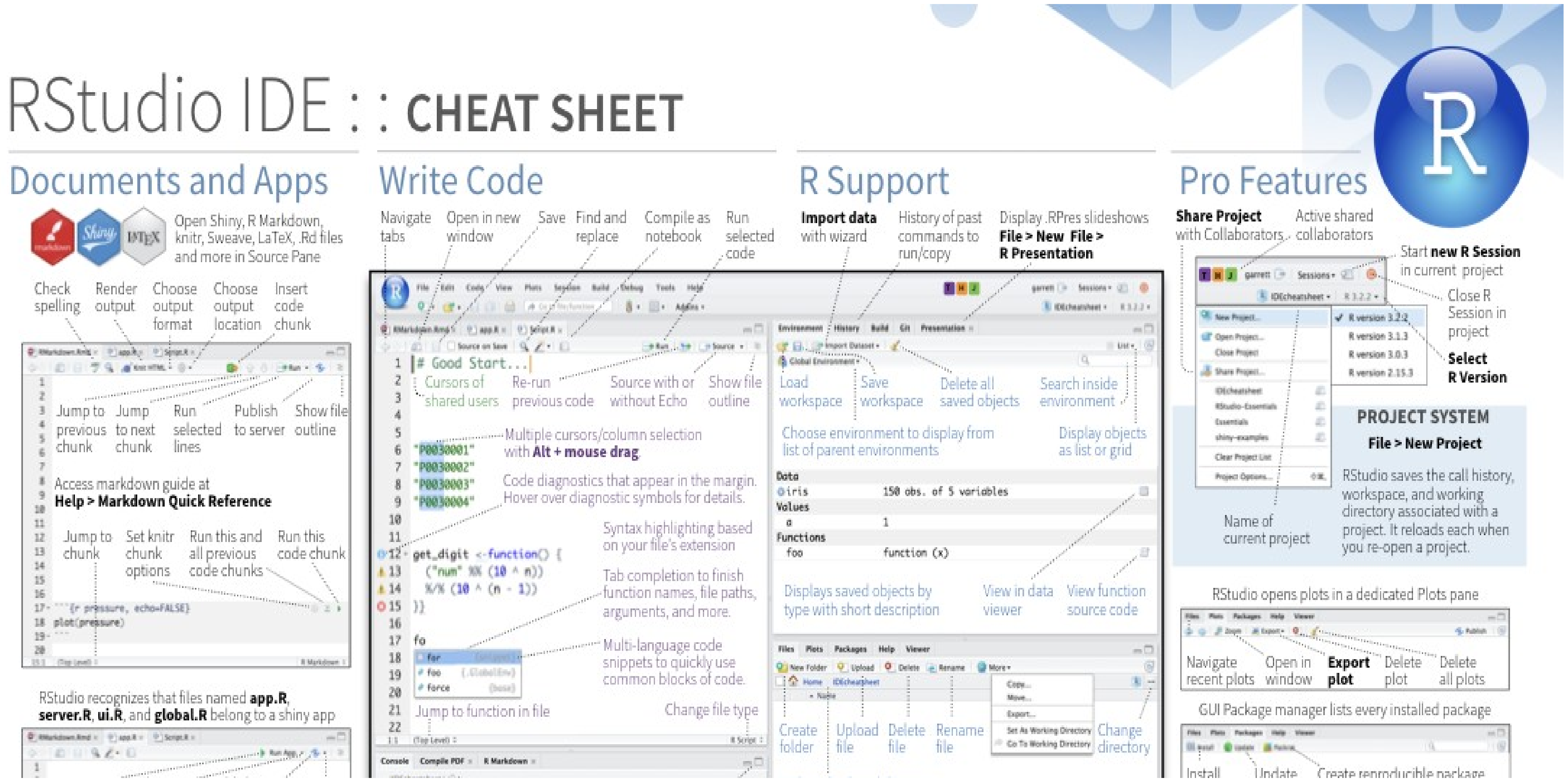
Business Science University
university.business-science.io

version: 0.0

RStudio is freely available open-source Integrated Development Environment (IDE).

RStudio provides an environment with many features to make using R easier and is a great alternative to working on R in the terminal.

R Studio



RStudio IDE : : CHEAT SHEET

Documents and Apps

Open Shiny, R Markdown, knitr, Sweave, LaTeX, .Rd files and more in Source Pane

Check spelling · Render output · Choose output format · Choose output location · Insert code chunk

Jump to previous chunk · Jump to next chunk · Run selected lines · Publish to server · Show file outline

Access markdown guide at **Help > Markdown Quick Reference**

Jump to chunk · Set knitr chunk options · Run this and all previous code chunks · Run this code chunk

~~~
```{r pressure, echo=FALSE}
plot(pressure)
```
~~~

RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app

Write Code

Navigate tabs · Open in new window · Save · Find and replace · Compile as notebook · Run selected code

```
# Good Start...

"P0030001"
"P0030002"
"P0030003"
"P0030004"

get_digit <- function() {
  ("num" %% (10 ^ n))
  %/% (10 ^ (n - 1))
})

fo
```

Cursors of shared users · Re-run previous code · Source with or without Echo · Show file outline

Multiple cursors/column selection with **Alt + mouse drag**.

Code diagnostics that appear in the margin. Hover over diagnostic symbols for details.

Syntax highlighting based on your file's extension

Tab completion to finish function names, file paths, arguments, and more.

Multi-language code snippets to quickly use common blocks of code.

Jump to function in file · Change file type

R Support

Import data with wizard · History of past commands to run/copy · Display .RPres slideshows **File > New File > R Presentation**

Load workspace · Save workspace · Delete all saved objects · Search inside environment

Choose environment to display from list of parent environments · Display objects as list or grid

```
Data
iris          150 obs. of 5 variables
Values
a             1
Functions
foo           function (x)
```

Displays saved objects by type with short description · View in data viewer · View function source code

Create folder · Upload file · Delete file · Rename file · Change directory

Pro Features

Share Project with Collaborators · Active shared collaborators · Start **new R Session** in current project · Close R Session in project · **Select R Version**

R version 3.2.2 · R version 3.1.3 · R version 3.0.3 · R version 2.15.3

Name of current project

PROJECT SYSTEM

File > New Project

RStudio saves the call history, workspace, and working directory associated with a project. It reloads each when you re-open a project.

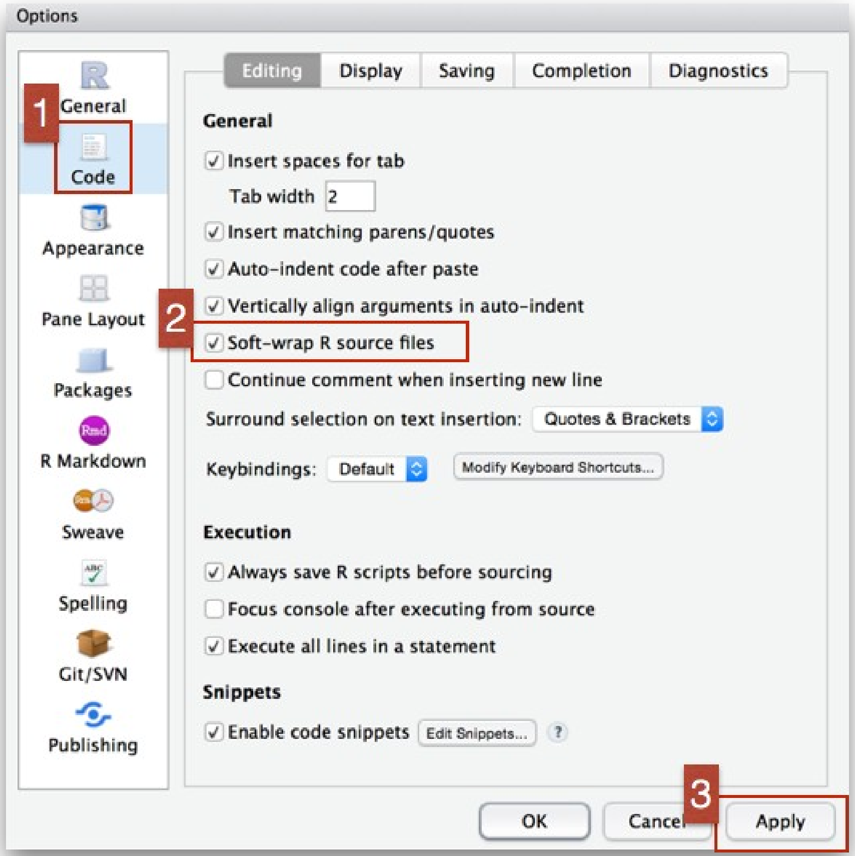
RStudio opens plots in a dedicated Plots pane

Navigate recent plots · Open in window · **Export plot** · Delete plot · Delete all plots

GUI Package manager lists every installed package

Install · Update · Create reproducible package

Rstudio



General

- [✓] Insert spaces for tab
  - Tab width: 2
- [✓] Insert matching parens/quotes
- [✓] Auto-indent code after paste
- [✓] Vertically align arguments in auto-indent
- [✓] Soft-wrap R source files
- [ ] Continue comment when inserting new line

Surround selection on text insertion: Quotes & Brackets

Keybindings: Default   Modify Keyboard Shortcuts...

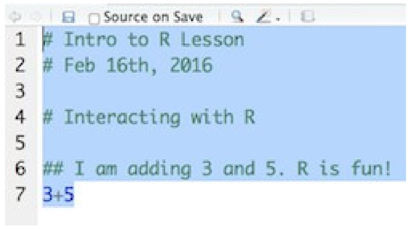
Execution

- [✓] Always save R scripts before sourcing
- [ ] Focus console after executing from source
- [✓] Execute all lines in a statement

Snippets

- [✓] Enable code snippets   Edit Snippets...   ?

The Rstudio script editor allows you to 'send' the current line or the currently highlighted text to the R console by clicking on the Run button in the upper-right hand corner of the script editor. Alternatively, you can run by simply pressing the Ctrl and Enter keys at the same time as a shortcut.



```
# Intro to R Lesson
# Feb 16th, 2016

# Interacting with R

## I am adding 3 and 5. R is fun!
3+5
```

Getting Help

Accessing the help files

`?mean`
Get help of a particular function.

`help.search('weighted mean')`
Search the help files for a word or phrase.

`help(package = 'dplyr')`
Find help for a package.

More about an object

`str(iris)`
Get a summary of an object's structure.

`class(iris)`
Find the class an object belongs to.

Using Packages

`install.packages('dplyr')`
Download and install a package from CRAN.

`library(dplyr)`
Load the package into the session, making all its functions available to use.

`dplyr::select`
Use a particular function from a package.

`data(iris)`
Load a built-in dataset into the environment.

Working Directory

`getwd()`
Find the current working directory (where inputs are found and outputs are sent).

`setwd('C://file/path')`
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Variable Names

Variables can be given almost any name, such as x, current_temperature, or subject_id. However, there are some rules / suggestions you should keep in mind:

- Make your names explicit and not too long.
- Avoid names starting with a number (2x is not valid but x2 is)
- Avoid names of fundamental functions in R (e.g., if, else, for, see here for a complete list). In general, even if it's allowed, it's best to not use other function names (e.g., c, T, mean, data) as variable names. When in doubt check the help to see if the name is already in use.
- Avoid dots (.) within a variable name as in my.dataset. There are many functions in R with dots in their names for historical reasons, but because dots have a special meaning in R (for methods) and other programming languages, it's best to avoid them.
- Use nouns for object names and verbs for function names
- Keep in mind that R is case sensitive (e.g., genome_length is different from Genome_length)
- Be consistent with the styling of your code (where you put spaces, how you name variable, etc.). In R, two popular style guides are Hadley Wickham's style guide and Google's.

Data Types

Variables can contain values of specific types within R. The six data types that R uses include:

"numeric" for any numerical value
"character" for text values, denoted by using quotes ("") around value
"integer" for integer numbers (e.g., 2L, the L indicates to R that it's an integer)
"logical" for TRUE and FALSE (the Boolean data type)
"complex" to represent complex numbers with real and imaginary parts (e.g., 1+4i) and that's all we're going to say about them
"raw" that we won't discuss further
The table below provides examples of each of the commonly used data types:

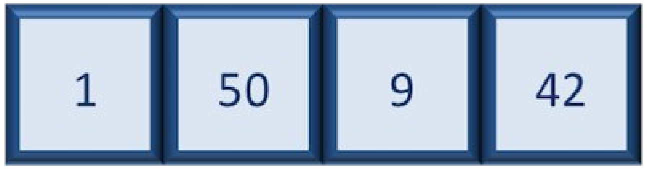
Data Type	Examples
`Numeric:`	`1, 1.5, 20, pi`
`Character:`	`“anytext”, “5”, “TRUE”`
`Integer:`	`2L, 500L, -17L`
`Logical:`	`TRUE, FALSE, T, F`

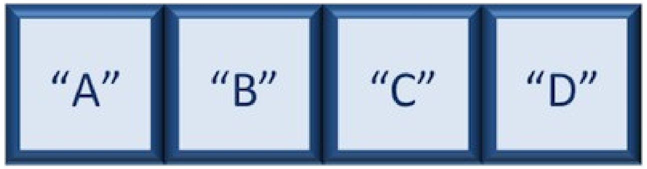
Data Structures

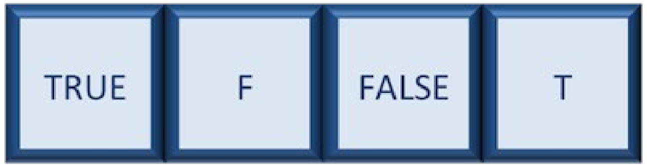
vectors (c),
factors (factor),
matrices (matrix),
data frames (data.frame)
lists (list).

Vectors

A vector is the most common and basic data structure in R, and is pretty much the workhorse of R. It's basically just a collection of values, mainly either numbers, All must be same data types.







Vectors

Creating Vectors

`c(2, 4, 6)`	2 4 6	Join elements into a vector
`2:6`	2 3 4 5 6	An integer sequence
`seq(2, 3, by=0.5)`	2.0 2.5 3.0	A complex sequence
`rep(1:2, times=3)`	1 2 1 2 1 2	Repeat a vector
`rep(1:2, each=3)`	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

`sort(x)` Return x sorted.	**`rev(x)`** Return x reversed.
`table(x)` See counts of values.	**`unique(x)`** See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

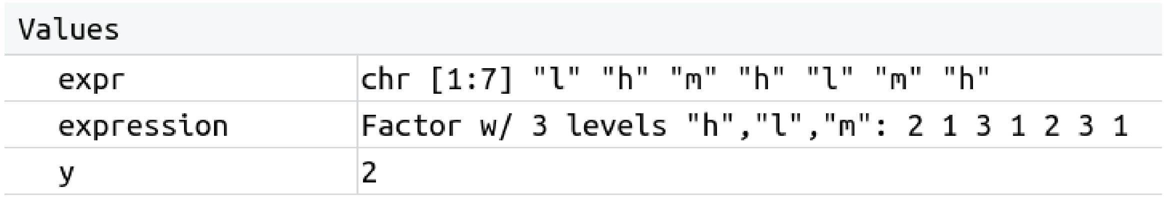
Named Vectors

`x['apple']`	Element with name 'apple'.

Factors

A factor is a special type of vector that is used to store categorical data. Each unique category is referred to as a factor level (i.e. category = level). Factors are built on top of integer vectors such that each factor level is assigned an integer value, creating value-label pairs.

```
expr <- c("l", "h", "m", "h", "l", "m", "h")
expression <- factor(expr)
```

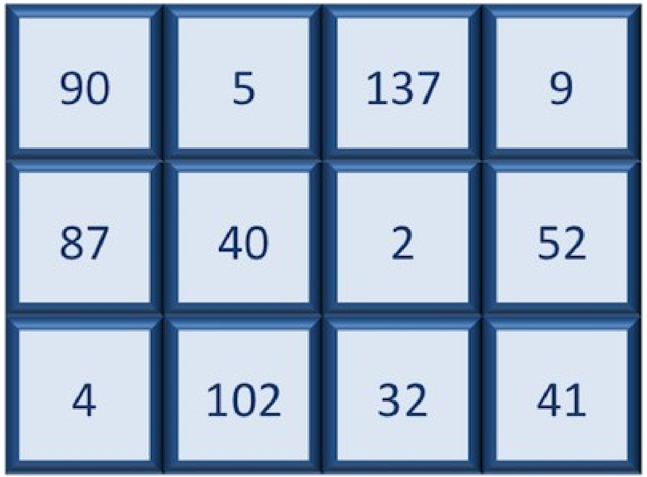


```
Values
   expr          chr [1:7] "l" "h" "m" "h" "l" "m" "h"
   expression    Factor w/ 3 levels "h","l","m": 2 1 3 1 2 3 1
   y             2
```

Matrix

A matrix in R is a collection of vectors of same length and identical datatype. Vectors can be combined as columns in the matrix or by row, to create a 2-dimensional structure.

Matrices are used commonly as part of the mathematical machinery of statistics. They are usually of numeric datatype and used in computational algorithms to serve as a checkpoint. For example, if input data is not of identical data type (numeric, character, etc.), the matrix() function will throw an error and stop any downstream code execution.



90	5	137	9
87	40	2	52
4	102	32	41

Data Frame

A data.frame is the de facto data structure for most tabular data and what we use for statistics and plotting. A data.frame is similar to a matrix in that it's a collection of vectors of the same length and each vector represents a column. However, in a dataframe each vector can be of a different data type (e.g., characters, integers, factors).

A data frame is the most common way of storing data in R, and if used systematically makes data analysis easier.

We can create a dataframe by bringing vectors together to form the columns. We do this using the data.frame() function, and giving the function the different vectors we would like to bind together. This function will only work for vectors of the same length.

Also see the **dplyr** package.

Data Frames

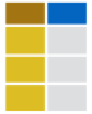
```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
```

A special case of a list where all elements are the same length.

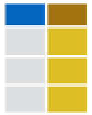
x	y
1	a
2	b
3	c

List subsetting

`df$x`



`df[[2]]`

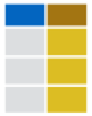


Understanding a data frame

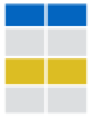
`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

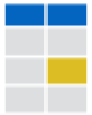
`df[ , 2]`



`df[2, ]`



`df[2, 2]`

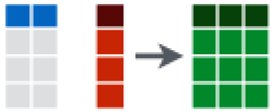


`nrow(df)`
Number of rows.

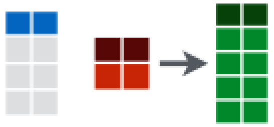
`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

cbind - Bind columns.

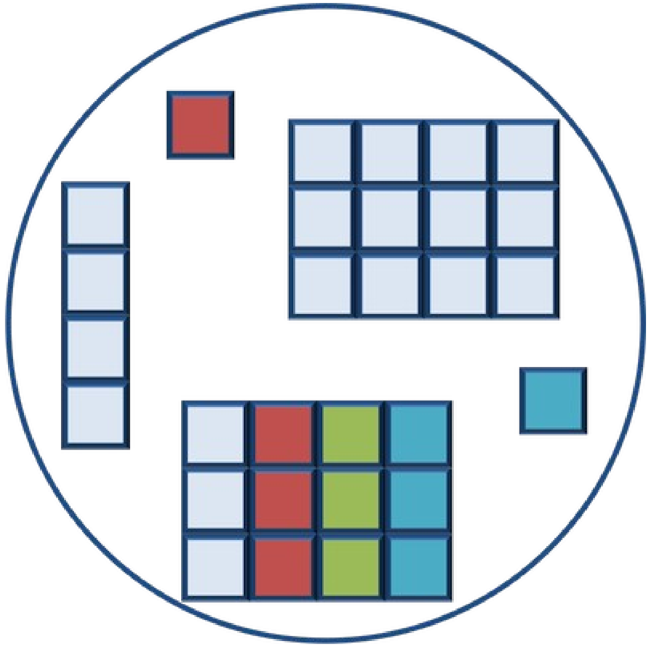


rbind - Bind rows.



Lists

Lists are a data structure in R that can be perhaps a bit daunting at first, but soon become amazingly useful. A list is a data structure that can hold any number of any types of other data structures. If you have variables of different data structures you wish to combine, you can put all of those into one list object by using the list() function and placing all the items you wish to combine within parentheses:



Lists

list1 <- list(species, df, number)

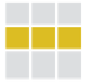
Print out the list to screen to take a look at the components:

```
list1
       [[1]]
[1] "ecoli" "human" "corn"

[[2]]
  species glengths
1   ecoli      4.6
2   human   3000.0
3    corn  50000.0

[[3]]
[1] 5
```

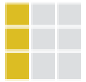
Matrices

`m <- matrix(x, nrow = 3, ncol = 3)`

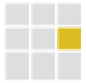
Create a matrix from x.



`m[2,  ]` - Select a row



`m[ , 1]` - Select a column



`m[2, 3]` - Select an element

`t(m)`

Transpose

`m %*% n`

Matrix Multiplication

`solve(m, n)`

Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`

A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

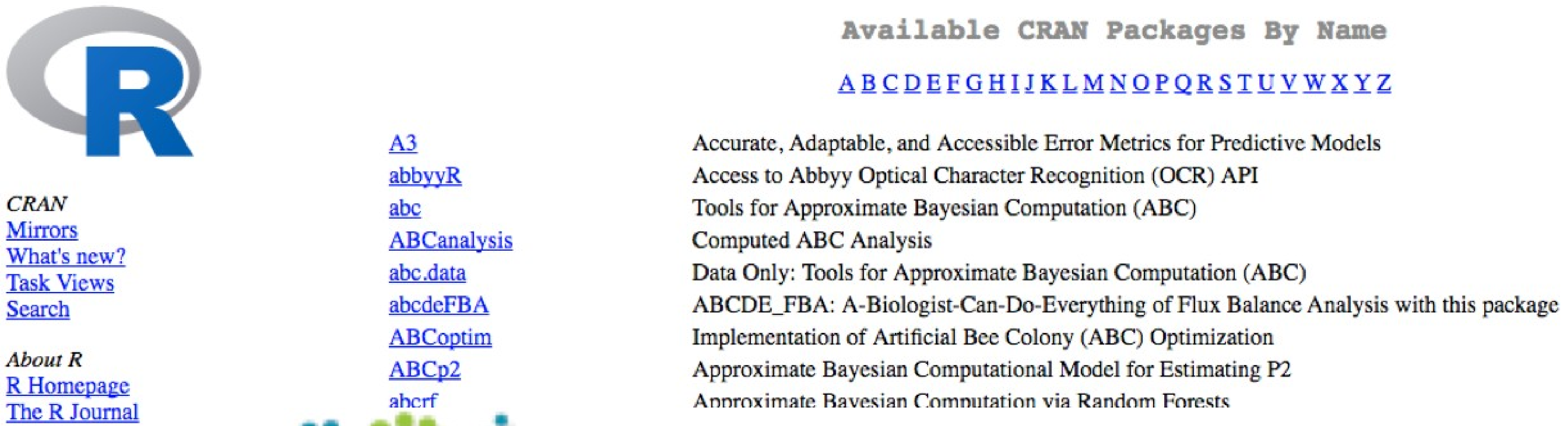
Functions

A key feature of R is functions. Functions are "self contained" modules of code that accomplish a specific task. Functions usually take in some sort of data structure (value, vector, dataframe etc.), process it, and return a result.

```
sqrt(100)
Round(3.333,3)
```

Package Source

A key feature of R is functions. Functions are "self contained" modules of



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Available CRAN Packages By Name

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A3	Accurate, Adaptable, and Accessible Error Metrics for Predictive Models
abbyyR	Access to Abbyy Optical Character Recognition (OCR) API
abc	Tools for Approximate Bayesian Computation (ABC)
ABCanalysis	Computed ABC Analysis
abc.data	Data Only: Tools for Approximate Bayesian Computation (ABC)
abcdeFBA	ABCDE_FBA: A-Biologist-Can-Do-Everything of Flux Balance Analysis with this package
ABCoptim	Implementation of Artificial Bee Colony (ABC) Optimization
ABCp2	Approximate Bayesian Computational Model for Estimating P2
abcrf	Approximate Bayesian Computation via Random Forests

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

Package Source

```
install.packages("ggplot2")

install.packages("BiocManager")
library(BiocManager)
install("DESeq2")


install.packages("~/Downloads/ggplot2_1.0.1.tar.gz", type="source",
repos=NULL)

library(ggplot2)
sessionInfo()
```

Save Session

```
save(iris, file="/tmp/iris.Rdata")

some_data <- load(file="~/Downloads/iris.RData")
```

Programming

For Loop

```
for (variable in sequence){

   Do something

}
```

Example

```
for (i in 1:4){

   j <- i + 10

   print(j)

}
```

While Loop

```
while (condition){

   Do something

}
```

Example

```
while (i < 5){

   print(i)

   i <- i + 1

}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){

   Do something

   return(new_variable)
}
```

Example

```
square <- function(x){

   squared <- x*x

   return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

Conditions

`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

Strings

Also see the **stringr** package.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)`

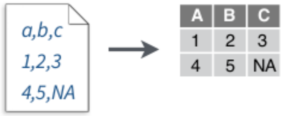
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`

Turn a numeric vector into a factor by 'cutting' into sections.

Read Tabular Data - These functions share the common arguments:

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),
  quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
  n_max), progress = interactive())
```



```
a,b,c
1,2,3
4,5,NA
```

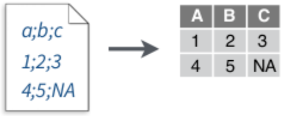
A	B	C
1	2	3
4	5	NA

Comma Delimited Files

read_csv("file.csv")

To make file.csv run:

`write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")`



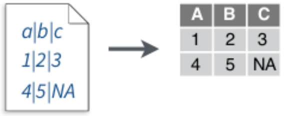
```
a;b;c
1;2;3
4;5;NA
```

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files

read_csv2("file2.csv")

`write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")`



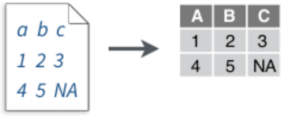
```
a|b|c
1|2|3
4|5|NA
```

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter

read_delim("file.txt", delim = "|")

`write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")`



```
a b c
1 2 3
4 5 NA
```

A	B	C
1	2	3
4	5	NA

Fixed Width Files

read_fwf("file.fwf", col_positions = c(1, 3, 5))

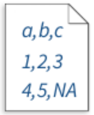
`write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")`

Tab Delimited Files

read_tsv("file.tsv") Also **read_table()**.

`write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")`

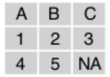
USEFUL ARGUMENTS



```
a,b,c
1,2,3
4,5,NA
```

Example file

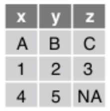
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"



A	B	C
1	2	3
4	5	NA

No header

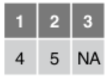
read_csv(f, **col_names = FALSE**)



x	y	z
A	B	C
1	2	3
4	5	NA

Provide header

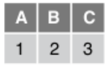
read_csv(f, **col_names = c("x", "y", "z")**)



1	2	3
4	5	NA

Skip lines

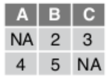
read_csv(f, **skip = 1**)



A	B	C
1	2	3

Read in a subset

read_csv(f, **n_max = 1**)



A	B	C
NA	2	3
4	5	NA

Missing Values

read_csv(f, **na = c("1", ".")**)

Data Inspection

We already saw how the functions head() and str() can be useful to check the content and the structure of a data.frame. Here is a non-exhaustive list of functions to get a sense of the content/structure of data.

All data structures - content display:
str(): compact display of data contents (env.)
class(): data type (e.g. character, numeric, etc.) of vectors and data structure of dataframes, matrices, and lists.
summary(): detailed display, including descriptive statistics, frequencies
head(): will print the beginning entries for the variable
tail(): will print the end entries for the variable
Vector and factor variables:
length(): returns the number of elements in the vector or factor
Dataframe and matrix variables:
dim(): returns dimensions of the dataset
nrow(): returns the number of rows in the dataset
ncol(): returns the number of columns in the dataset
rownames(): returns the row names in the dataset
colnames(): returns the column names in the dataset

Save Session

```
save(iris, file="/tmp/iris.Rdata")

some_data <- load(file="~/Downloads/iris.RData")
```

The tidyverse

Components

dplyr
forcats
ggplot2
haven
lubridate
%>%
Ceci n'est pas un pipe.
purrr
readr
readxl
stringr
TIBBLE
tidyr
tidyverse

The tidyverse is a collection of R packages that share common philosophies and are designed to work together. This site is a work-in-progress guide to the tidyverse and its packages.

dplyr

The most useful tool in the tidyverse is dplyr. It's a swiss-army knife for data wrangling. dplyr has many handy functions that we recommend incorporating into your analysis:

select() extracts columns and returns a tibble.
arrange() changes the ordering of the rows.
filter() picks cases based on their values.
mutate() adds new variables that are functions of existing variables.
rename() easily changes the name of a column(s)
summarise() reduces multiple values down to a single summary.
pull() extracts a single column as a vector.
_join() group of functions that merge two data frames together, includes (inner_join(), left_join(), right_join(), and full_join()).

dplyr

select()
To extract columns from a tibble we can use the select() function.

```
# Convert the res_tableOE data frame to a tibble
res_tableOE <- res_tableOE %>%
               rownames_to_column(var="gene") %>%
              as_tibble()

# extract selected columns from res_tableOE
res_tableOE %>%
    select(gene, baseMean, log2FoldChange, padj)
```

Conversely, you can remove columns you don't want with negative selection.

```
res_tableOE %>%
    select(-c(lfcSE, stat, pvalue))
```

dplyr

```
## # A tibble: 23,368 x 4
##         gene    baseMean log2FoldChange         padj
##          <chr>       <dbl>          <dbl>        <dbl>
##  1 1/2-SBSRNA4  45.6520399    0.266586547 2.708964e-01
##  2        A1BG  61.0931017    0.208057615 3.638671e-01
##  3    A1BG-AS1 175.6658069   -0.051825739 7.837586e-01
##  4        A1CF   0.2376919    0.012557390           NA
##  5       A2LD1  89.6179845    0.343006364 7.652553e-02
##  6         A2M   5.8600841   -0.270449534 2.318666e-01
##  7       A2ML1   2.4240553    0.236041349           NA
##  8       A2MP1   1.3203237    0.079525469           NA
##  9      A4GALT  64.5409534    0.795049160 2.875565e-05
## 10       A4GNT   0.1912781    0.009458374           NA
## # ... with 23,358 more rows
```

Let's save that tibble as a new variable called sub_res:

```
sub_res <- res_tableOE %>%
    select(-c(lfcSE, stat, pvalue))
```

dplyr

```
arrange(sub_res, padj)
## # A tibble: 23,368 x 4
##       gene   baseMean log2FoldChange          padj
##      <chr>      <dbl>          <dbl>         <dbl>
##  1    MOV10 21681.7998      4.7695983  0.000000e+00
##  2     H1F0  7881.0811      1.5250811 2.007733e-162
##  3    HSPA6   168.2522      4.4993734 1.969313e-134
##  4 HIST1H1C  1741.3830      1.4868361 5.116720e-101
##  5    TXNIP  5133.7486      1.3868320  4.882246e-90
##  6    NEAT1 21973.7061      0.9087853  2.269464e-83
##  7    KLF10  1694.2109      1.2093969  9.257431e-78
##  8   INSIG1 11872.5106      1.2260848  8.853278e-70
##  9    NR1D1   969.9119      1.5236259  1.376753e-64
## 10    WDFY1  1422.7361      1.0629160  1.298076e-61
## # ... with 23,358 more rows
```

dplyr

filter()
Let's keep only genes that are expressed (baseMean above 0) with an adjusted P value below 0.01. You can perform multiple filter() operations together in a single command.

```
sub_res %>%
   filter(baseMean > 0 & padj < 0.01)
## # A tibble: 4,959 x 4
##    gene     baseMean log2FoldChange     padj
##    <chr>       <dbl>          <dbl>    <dbl>
##  1 A4GALT       64.5          0.798 2.40e- 5
##  2 AAGAB      2614.          -0.390 1.68e-11
##  3 AAMP       3157.          -0.380 9.11e-13
##  4 AARS       3690.           0.167 2.10e- 3
##  5 AARS2      2255.          -0.204 3.77e- 4
##  6 AASDHPPT   3561.          -0.293 3.79e- 7
##  7 AASS       1018.           0.347 7.94e- 5
##  8 AATF       2613.          -0.290 1.97e- 7
##  9 ABAT        384.           0.384 1.99e- 4
## 10 ABCA1       108.           0.833 4.19e- 7
## # ... with 4,949 more rows
```

dplyr

mutate()
mutate() enables you to create a new column from an existing column.
Let's generate log10 calculations of our baseMeans for each gene.

```
sub_res %>%
  mutate(log10BaseMean = log10(baseMean)) %>%
  select(gene, baseMean, log10BaseMean)
## # A tibble: 23,368 x 3
##    gene        baseMean log10BaseMean
##    <chr>          <dbl>         <dbl>
##  1 1/2-SBSRNA4   45.7          1.66
##  2 A1BG          61.1          1.79
##  3 A1BG-AS1     176.           2.24
##  4 A1CF           0.238       -0.624
##  5 A2LD1         89.6          1.95
##  6 A2M            5.86         0.768
##  7 A2ML1          2.42         0.385
##  8 A2MP1          1.32         0.121
##  9 A4GALT        64.5          1.81
## 10 A4GNT          0.191       -0.718
## # ... with 23,358 more rows
```

dplyr

rename()
You can quickly rename an existing column with rename(). The syntax is new_name = old_name.

```
sub_res %>%
  rename(symbol = gene)
## # A tibble: 23,368 x 4
##    symbol      baseMean log2FoldChange       padj
##    <chr>          <dbl>          <dbl>      <dbl>
##  1 1/2-SBSRNA4   45.7          0.268    0.264
##  2 A1BG          61.1          0.209    0.357
##  3 A1BG-AS1     176.          -0.0519   0.781
##  4 A1CF           0.238        0.0130  NA
##  5 A2LD1         89.6          0.345    0.0722
##  6 A2M            5.86        -0.274    0.226
##  7 A2ML1          2.42         0.240   NA
##  8 A2MP1          1.32         0.0811  NA
##  9 A4GALT        64.5          0.798    0.0000240
## 10 A4GNT          0.191        0.00952 NA
## # ... with 23,358 more rows
```

dplyr

pull()
In the recent dplyr 0.7 update, pull() was added as a quick way to access column data as a vector. This is very handy in chain operations with the pipe operator.

```
# Extract first 10 values from the gene column
pull(sub_res, gene) %>% head()
```

dplyr

_join()
Dplyr has a powerful group of join operations, which join together a pair of data frames based on a variable or set of variables present in both data frames that uniquely identify all observations. These variables are called keys.

inner_join: Only the rows with keys present in both datasets will be joined together.

left_join: Keeps all the rows from the first dataset, regardless of whether in second dataset, and joins the rows of the second that have keys in the first.

right_join: Keeps all the rows from the second dataset, regardless of whether in first dataset, and joins the rows of the first that have keys in the second.

full_join: Keeps all rows in both datasets. Rows without matching keys will have NA values for those variables from the other dataset.

dplyr

```
 ID <- c(546, 983, 042, 952, 853, 061)
 diet <- c("veg", "pes", "omni", "omni", "omni", "omni")
 exercise <- c("high", "low", "low", "low", "med", "high")
 behavior <- data.frame(ID, diet, exercise)

 # Creating blood dataframe

 ID <- c(983, 952, 704, 555, 853, 061, 042, 237, 145, 581, 249, 467, 841, 
546)
 blood_levels <- c(43543, 465, 4634, 94568, 134, 347, 2345, 5439, 850, 
6840, 5483, 66452, 54371, 1347)
 blood <- data.frame(ID, blood_levels)
```

dplyr

To join only the IDs present in both data frames, we could use the inner_join() function:

```
# Inner join
inner_join(blood, behavior)
```

Alternatively, if we wanted to return all blood IDs, but include only the behavior IDs that match, we could use the left_join() function:

```
# Left join
left_join(blood, behavior)
```

We could also do the same thing but return all behavior IDs and matching blood IDs using right_join():

```
# Right join
right_join(blood, behavior)
```

Finally, we could return all IDs from both data frames regardless whether there is a matching key (ID):

```
# Full join
full_join(blood, behavior)
```