# Introduction to

**Introduction to Statistics and R**

May 25,26,29,30, 2017

Sarah Bonnin – Bioinformatics facility

Estefania Mancini – lab Juan Valcárcel

CRG Centre for Genomic Regulation

prbb Barcelona Biomedical Research Park

# Introduction to Statistics and R

- **Module 0: Introduction to R**
  May 25th, 26th, 29th and 30th

- **Module I: Descriptive Statistics & Intro to Probability**
  June 6th

- **Module II: Statistical Inference**
  June 8th

- **Module III: Statistical modeling & Regression**
  June 9th

**Bioinformatics room**
**10:00-13:00**

**Lectures:**
**Ramon y Cajal**
**10:00-13:00**

**Practicums:**
**Bioinformatics room**
**14:00-17:00**

# Introduction to Statistics and R – Course page

https://biocore.crg.eu/wiki/CRG_Introduction_to_Statistics_and_R_2017

# Outline

## *day 1 – May 25th*

- R Studio
- R basics
- Data types
- Data structures: Vectors
- Input/Output

# Outline

## *day 2 – May 26th*

- Data structures:
  - Matrix
  - Data Frames
- More on Input and Output

# Outline

## *day 3 – May 29th*

- Lists
- Library / packages
  - CRAN
  - Bioconductor
- (Writing functions in R)

# Outline

## *day 4 – May 30th*

- Graphing in R:
  - basic graphing
  - introduction ggplot2 package

# What is R?

- Used for **data manipulation**, **calculation** and **graphical display.**

- **Open source !**

https://www.r-project.org/

- **Interactive, flexible**

- **Very active community of developers and users!**

# R Studio

# R studio?

- **Free and open source** Integrated Development Environment for R

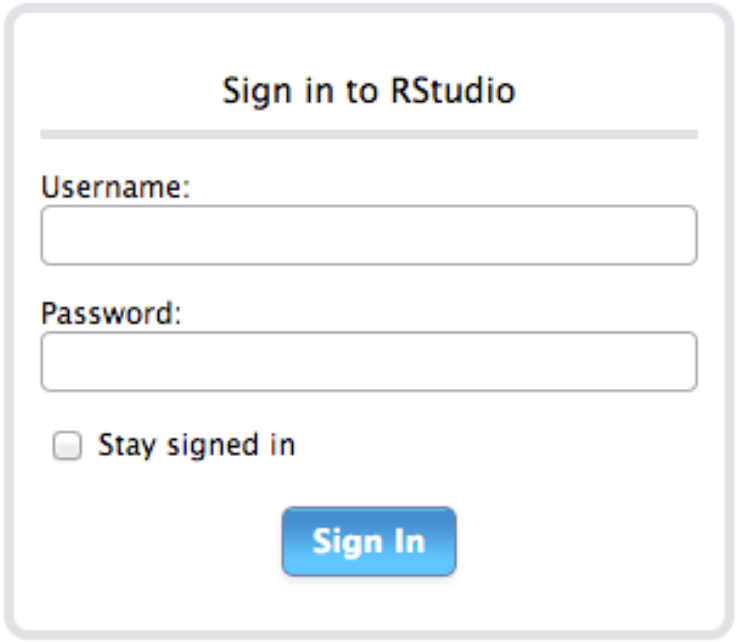
- Available for Windows, Mac OS and Linux

# Install R Studio locally

- **https://www.rstudio.com/products/rstudio/download2/**

# Connect to R Studio on the CRG server via a web browser

- **http://rstudio.linux.crg.es/**



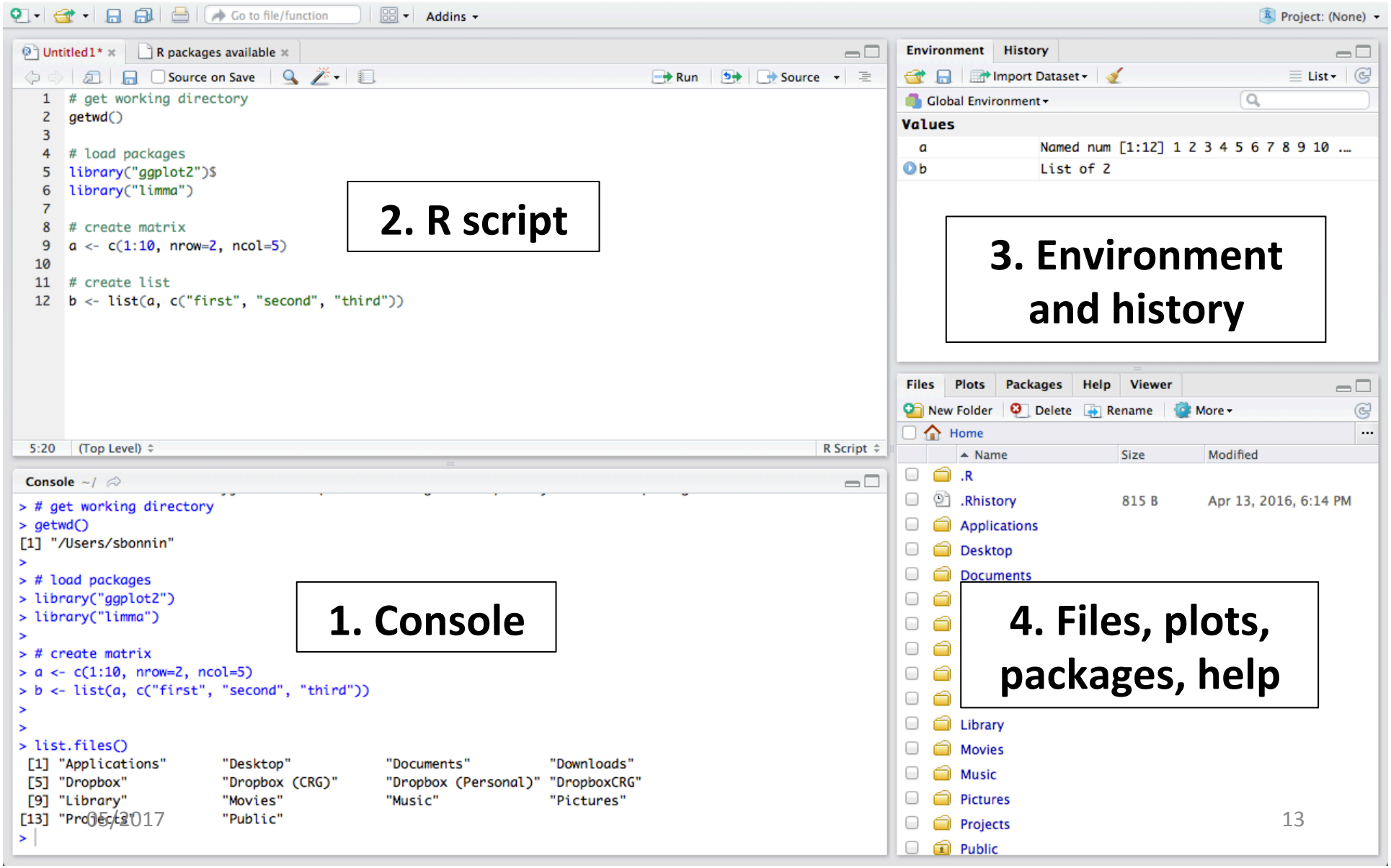
# Connect to R Studio on the CRG server via a web browser

- **http://rstudio.linux.crg.es/**

- **Connect with your CRG mail credentials**

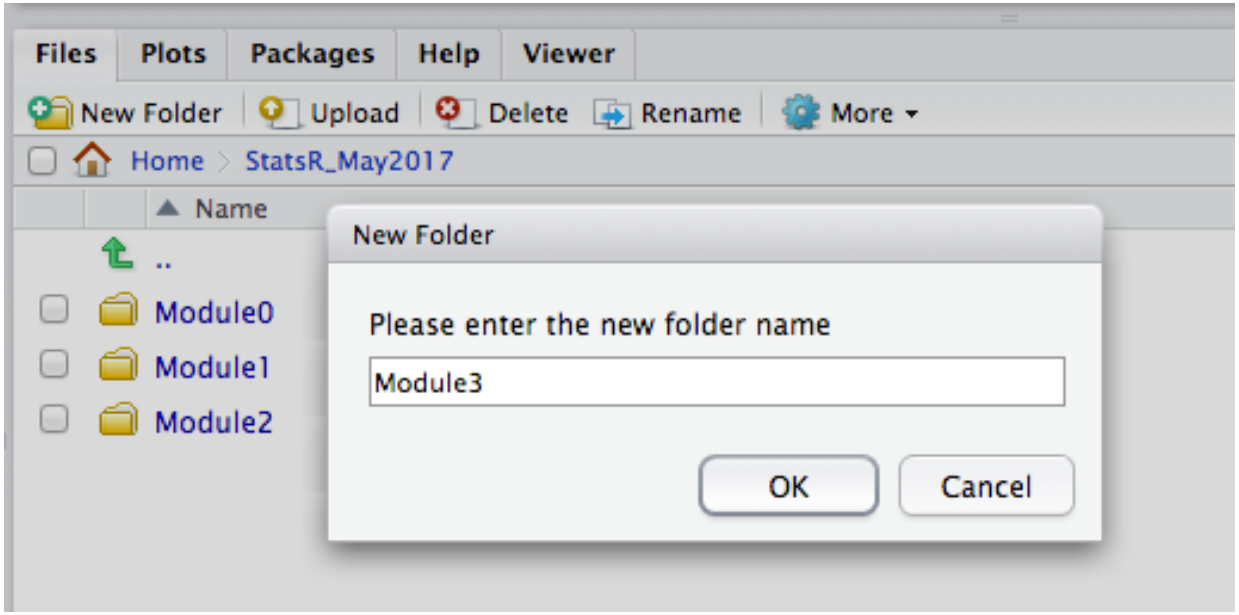
## *R Studio*

# Screen: 4 windows



# Create a directory and subdirectories for the course

**StatsR_May2017/**
**|-- Module0**
**|-- Module1**
**|-- Module2**
**|-- Module3**



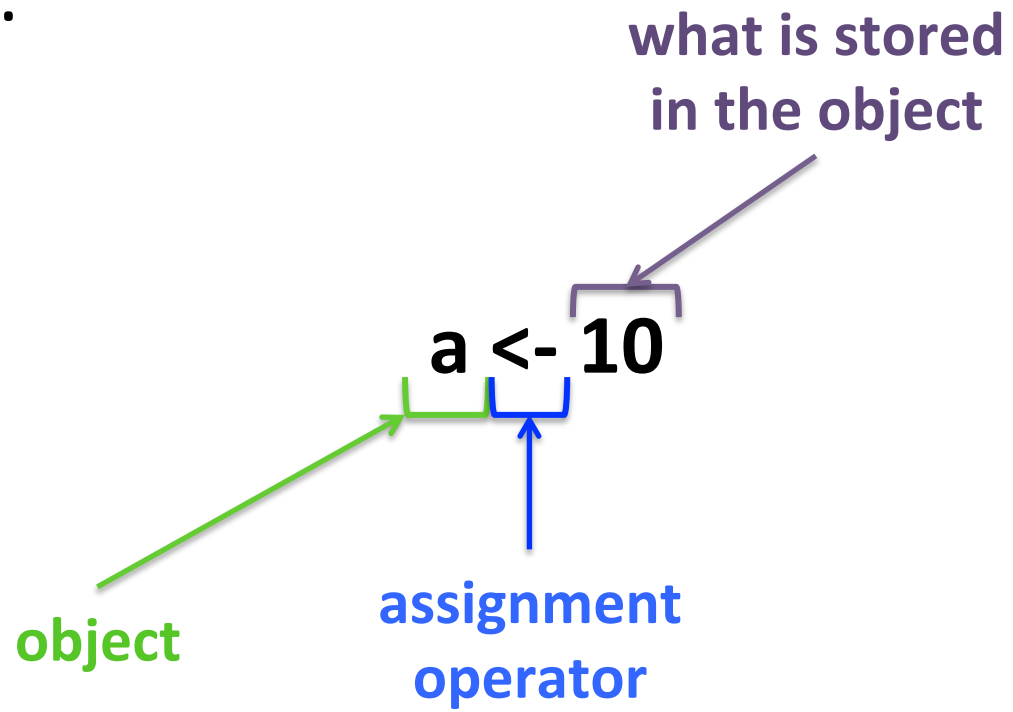
# R Basics

# Elementary arithmetic operators

| | |
|---|---|
| addition | **+** |
| subtraction | **-** |
| division | **/** |
| multiplication | ***** |
| exponentiation | **^** or ** |

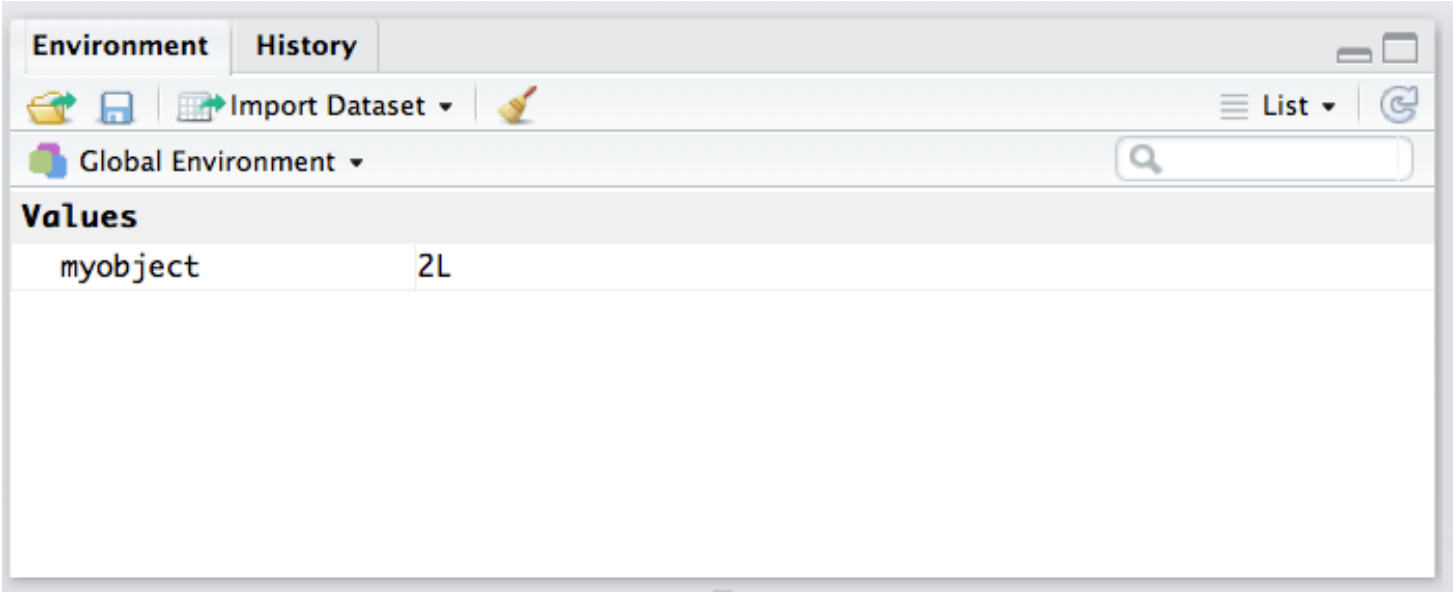
# R base syntax

example:



# Objects in R Studio

The object can be found in the "Environment" tab of R Studio.

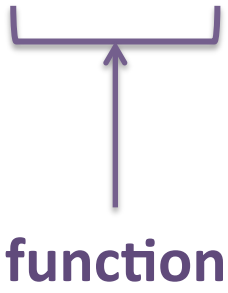


# R functions

## **function()**

example:

**getwd()**

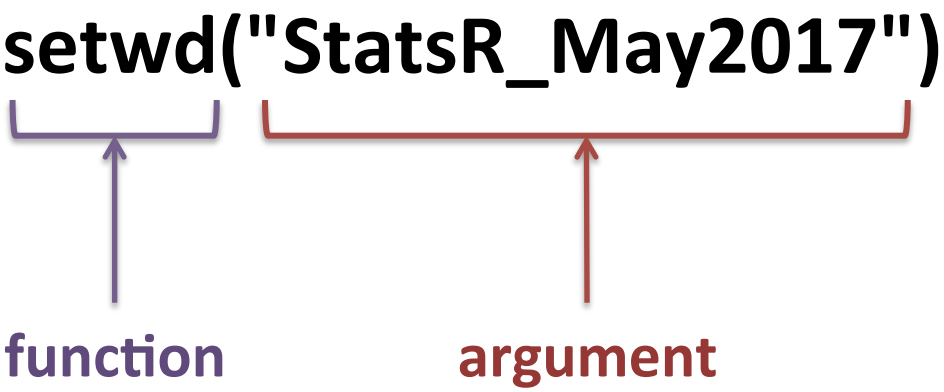


**function**

# R functions

**function(argument)**

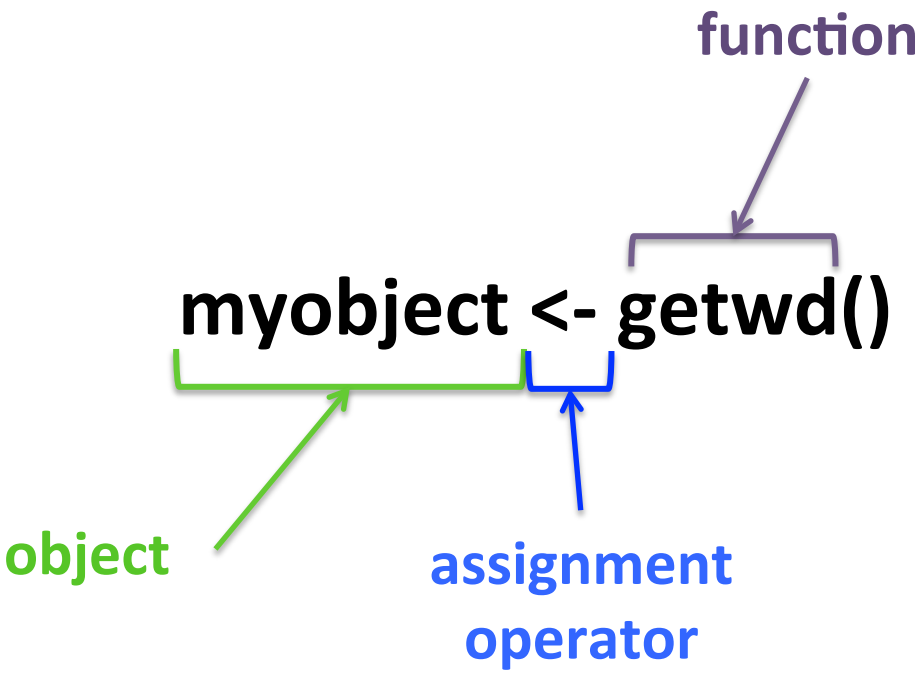
example:



# R functions

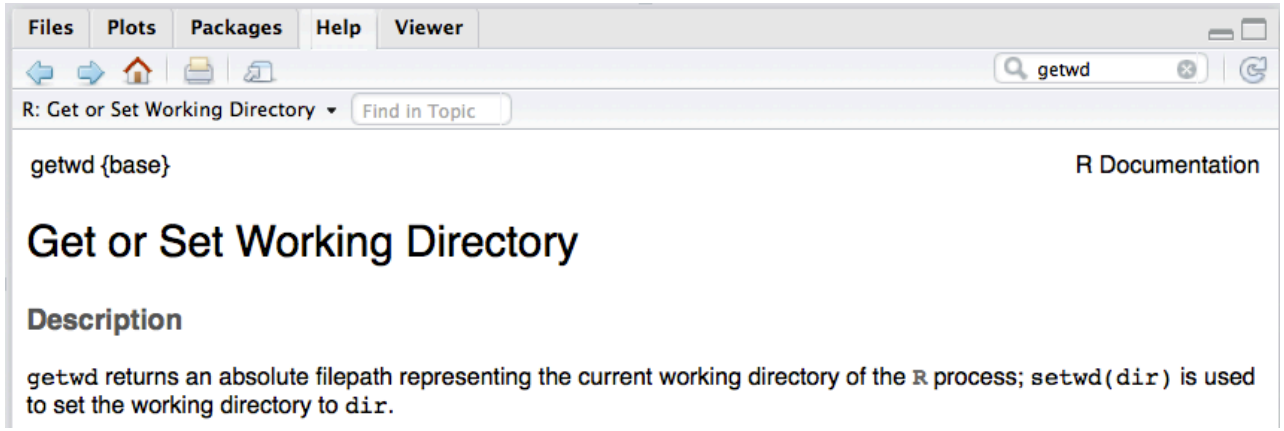
## a <- function()

example:



# Getting help on a function

- help(**getwd**)
- ?**getwd**
- Tab "help" in R Studio



# Example of functions

- example(**mean**)

```
mean> x <- c(0:10, 50)
mean> xm <- mean(x)
```

# R script

- Store commands in a ".R" script
- Execute commands or blocks of commands from R studio

# R syntax

- Case sensitive
- Comment lines start with **#**
- Commands separated by **;** or a new line

# Data Types

**Numeric**

**Character**

**Logical**

# Data Types

- Every object has a **data type** that tells what sort of value it is:
  - Numeric (Numbers)
  - Character (Text)
  - Logical (True / False)

# Data Types

- a <- 10
  - str(a)

  num 10

- b <- "word"
  - str(b)

  chr "word"

# Data Structures

**Vectors**

**Matrices**

**Data frames**

**List**

# Data Structures

**Vectors**

Matrices

Data frames

List

# Vectors

- Sequence of data elements of the same type

- Assignment of values to vector using the

  **c** command (combining elements)

```
a <- c(1, 9, 4, 8, 0, 11, 7)
```

# Numeric vectors

- Create a sequence of consecutive numbers

```
a <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
a <- 1:10
```

# Character vectors

- each element is between (double) quotes

```
b <- c("test", "ok")
```

# Vector manipulation

- Fetch elements of a vector **a**:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **values** | **1** | **9** | **4** | **8** | **0** | **11** | **7** |

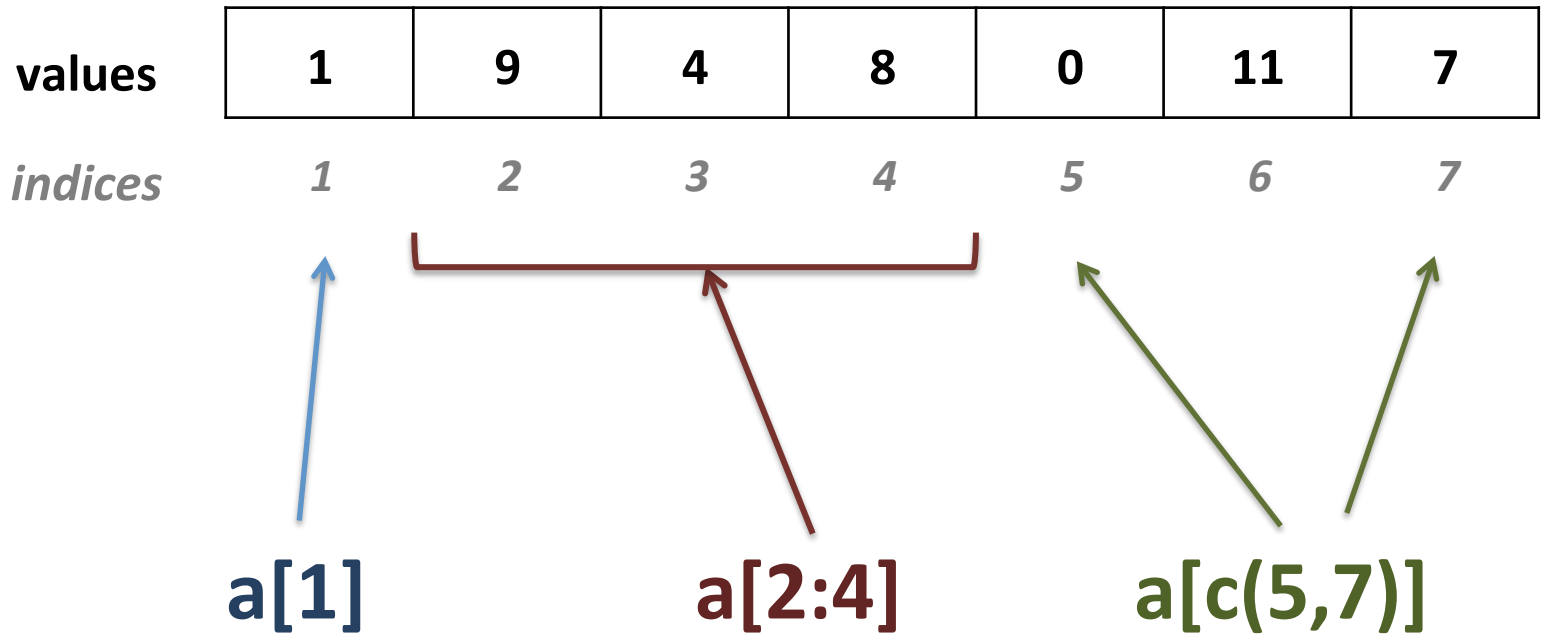
# Vector manipulation

- Fetch elements of a vector **a**:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **values** | **1** | **9** | **4** | **8** | **0** | **11** | **7** |
| ***indices*** | *1* | *2* | *3* | *4* | *5* | *6* | *7* |

# Vector manipulation

- Fetch elements of a vector **a**:



# Vector manipulation

- Get length of a vector

```
a <- 1:5
[1] 5
length(a)
```

# Vector manipulation

- Replacing a vector's value:

```
a <- 1:5
[1] 1 2 3 4 5

a[2] <- 3
[1] 1 3 3 4 5
```

# Numeric vector manipulation

- Add 2 to each element of a numeric vector

```
a <- 1:5
a + 2
[1] 3 4 5 6 7
```

# Numeric vector manipulation

- a <- 1:5
- Is **2** present in **a**?

```
a == 2
```

```
[1] FALSE  TRUE FALSE FALSE FALSE
```

- Which element of **a** are superior to **2**

```
a > 2
```

```
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

# Numeric vector manipulation

- a <- 1:5
- Select elements superior or equal to 2

a >= 2

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

[1] FALSE TRUE TRUE TRUE TRUE

**a[a >= 2]**

| 2 | 3 | 4 | 5 |
|---|---|---|---|

# Vectors comparison
## the **%in%** operator

- a <- 2:6
- b <- 4:20

- What elements of **a** are in **b**?

**a %in% b**

```
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

- Get the actual values of **a** that are in **b**

**a[a %in% b]**

```
[1] 4 5 6
```

# Logical operators

| | |
|---|---|
| inferior | **<** |
| inferior or equal | **<=** |
| superior | **>** |
| superior or equal | **>=** |
| equality | **==** |
| inequality | **!=** |
| not x | **!x** |
| x OR y | **x \| y** |
| x AND y | **x & y** |

# Obtaining summary statistics

| | |
|---|---|
| average/mean | **mean(x)** |
| median | **median(x)** |
| minimum | **min(x)** |
| maximum | **max(x)** |
| variance | **var(x)** |
| correlation | **cor(x)** |

# Exercise 1:
## *Numeric vector manipulation*

# Character vector manipulation

- b <- c("ok", "yes", "no", "ok")
- Select all the "ok" elements

b == "ok"

| ok | yes | no | ok |
|---|---|---|---|

[1] TRUE FALSE FALSE TRUE

**b[b == "ok"]**

| ok | ok |
|---|---|

# Exercise 2:
## *Character vector manipulation*

***Exercise 1***

# Vector manipulation

- Create vector **x** of 1000 random numbers.

  **rnorm()**

- Compute the mean, median, minimum and maximum of this vector.

  **mean(), median(), min(), max()**.

- Create **y**, vector containing the first 100 elements of **x**.

- Compute summary statistics of **y**

  **summary()**

# Input / Output

# File/directory path

- The path of a file/directory is its **location** in the file system.
- Your home directory is the one that hosts your personal folder:

**/nfs/users/[yourgroup]/[yourlogin]**

- Shortcut to your home directory**:** ~

# File/directory path

- Path of the current directory obtained with: **getwd**()

[1] "/nfs/users/bi/sbonnin"

- Move one directory up with: **setwd**("..")

You are now in: "/nfs/users/bi/"

- Go to another directory:

**setwd**("/nfs/users/bi/sbonnin/test")

# Read in a file into a vector

- **scan**("file.txt")

By default, scans "double" elements.

→can fail if the input contains characters

- **scan**("file.txt", what="character")

→ specify the type of data to scan!

- **scan**("/users/bi/sbonnin/Dir/file.txt", )

→ If file is not in the current directory:

# Write a vector into a file

- **write**(ourfile, file="ourfile.txt")
- **write**(ourfile, file="ourfile.txt", ncolumns=1)

→organizes the data in one column

# Exercise 3:
## *R script*

# R script

**Create** an empty file **ex2.R**

**Within that file:**

- Create a directory "exercise2" – **dir.create()**
- Go to directory "exercise2" – **setwd()**
- Read file **TO ADD** into tt object – **scan()**
- Add 10 to each element of the vector
- Write result into a file – **write()**
- Goes back to home directory – **setwd("~")**

**Execute** script **ex2.R** – source in RStudio

# Factors

- Factors are **categorical vectors**

- Important for:
  - plotting
  - statistical modeling

# Factors

- Like vectors but with **levels** that represent each category

```
e <- factor(c("high", "low", "medium", "low"))

e
[1] high   low    medium low
Levels: high low medium
```

# Factors

- Levels can be given **ordered**

```
e <- factor(e, levels=c("low", "medium", "high"),
ordered=TRUE)

e
[1] high   low    medium low
Levels: low < medium < high
```

# Factors

- Example of a character vector versus a factor

```
e <- factor(c("high", "low", "medium", "low"))
e2 <- c("high", "low", "medium", "low")

str(e)
 Factor w/ 3 levels "high","low","medium": 1 2 3 2
str(e)
 chr [1:4] "high" "low" "medium" "low"
```

# Outline

## *day 2 – May 26th*

- Data structures:
  - Matrix
  - Data Frames
- More on Input and Output

# Data Structures

Vectors

**Matrices**

**Data frames**

List

# Matrices

- A matrix is a vector of **2 dimensions**

- All columns in a matrix must have :
  - the same **type** (numeric, character, logical)
  - the same **length**

```
b <- matrix(c(1, 0, 34, 44, 12, 4),
            nrow=3,
            ncol=2)
```

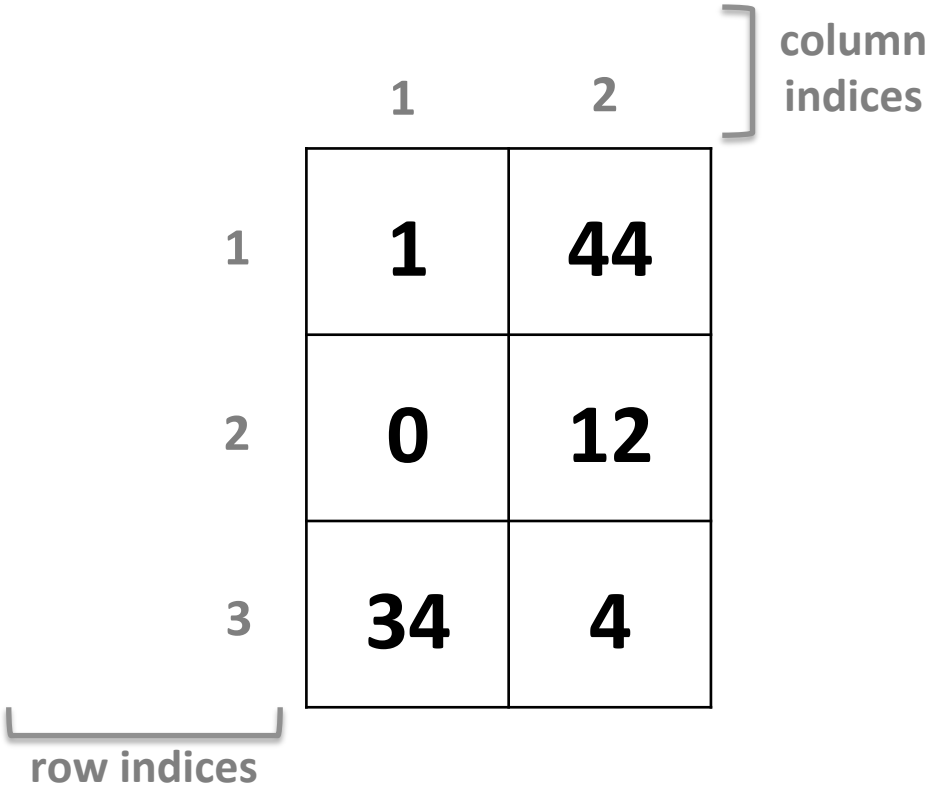
# Matrices

**b**

| | |
|---|---|
| **1** | **44** |
| **0** | **12** |
| **34** | **4** |

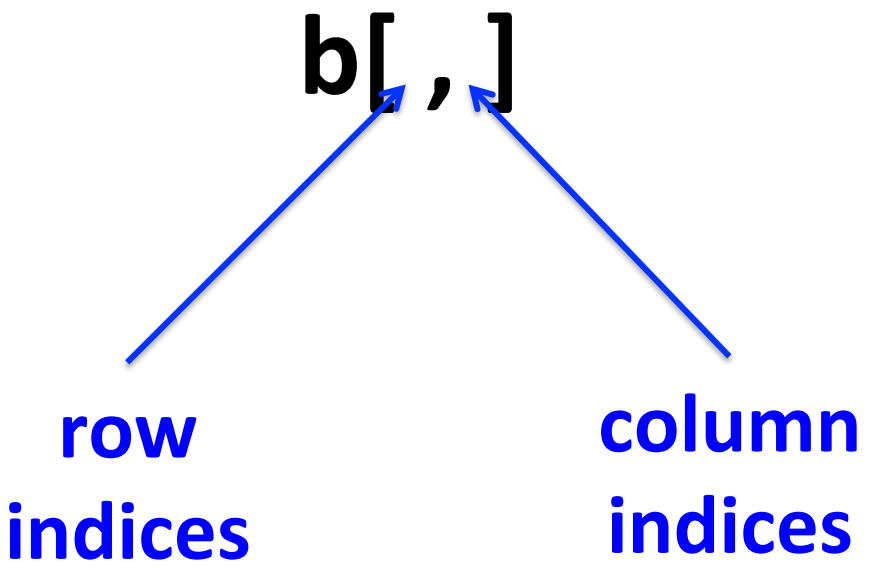
# Matrices

**b**



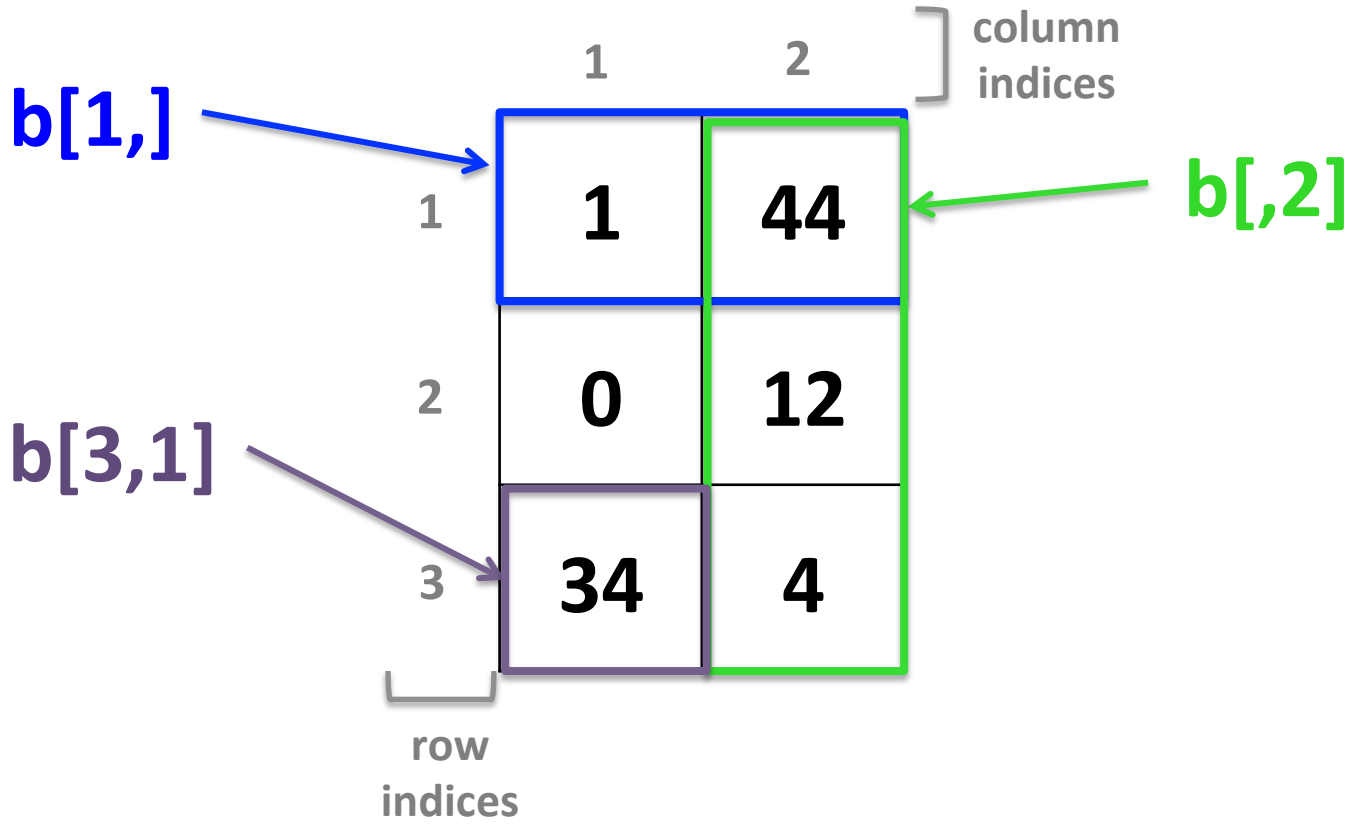
# Matrices

- Fetch rows, columns or single elements of a matrix:



# Matrices

**b**



# Manipulate matrices

- **add 1** to all elements of a matrix

```
b <- b + 1
```

- **multiply by 3** all elements of a matrix

```
b <- b * 3
```

- **subtract 2** to each element of the first row of a matrix

```
b[1,] <- b[1,] – 2
```

- **replace** elements that comply a condition:

```
b[b>3] <- 4
```

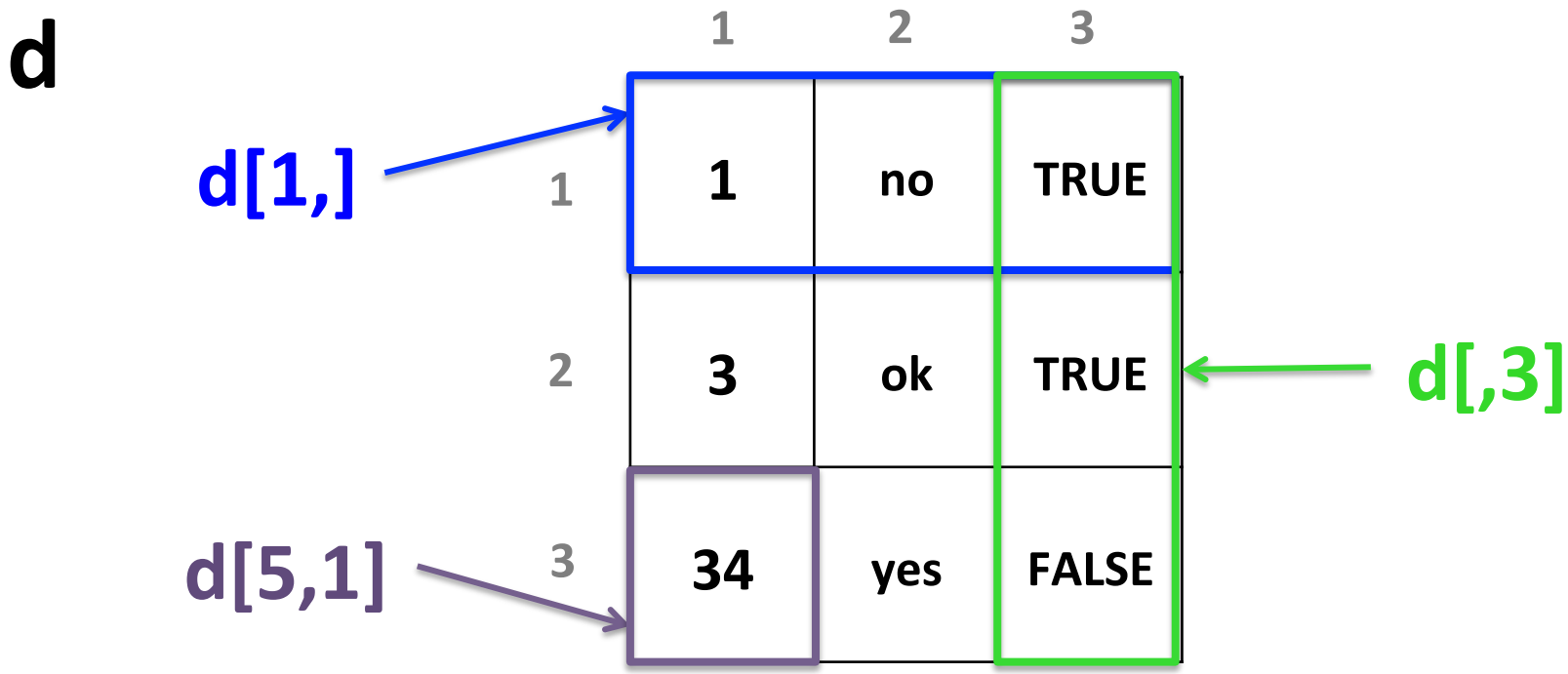
# Data frames

- Structures of 2 dimensions.
- More general than matrices.
- Different columns can have **different types** but must have the **same length.**

```
d <- data.frame(c(1, 3, 34),
                c("no", "ok", "yes"),
                c(TRUE, TRUE, FALSE))
```

# Data frames

- Fetch rows, columns or single elements of a data frame:



# Dimensions

- Get dimension of data frame or matrix

- **dim**(b)

  first element: number of rows

  second element: number of columns
- **nrow**(b)
- **ncol**(b)

# Manipulate 2-dimensional objects

- Same principle as for vectors… but 2-dimensional.
- Select rows of b if elements in **column 1 are superior to 0:**

| | |
|---|---|
| **1** | **44** |
| **0** | **12** |
| **34** | **4** |

**b[b[,1] > 0,]**

| | |
|---|---|
| **1** | **44** |
| **34** | **4** |

# Manipulate 2-dimensional objects

- Select columns of b if elements in **row 3 are inferior or equal to 4:**

| | |
|---|---|
| 1 | 44 |
| 0 | 12 |
| 34 | 4 |

**b[ , b[3 ,] <= 4 ]**

| |
|---|
| 44 |
| 12 |
| 4 |

# Manipulate 2-dimensional objects

- Select rows of d if elements in **column 3 are TRUE:**

| | | |
|---|---|---|
| **1** | **no** | **TRUE** |
| **3** | **ok** | **TRUE** |
| **34** | **yes** | **FALSE** |

**d[ d[, 3] == TRUE, ]**

| | | |
|---|---|---|
| **1** | **no** | **TRUE** |
| **3** | **ok** | **TRUE** |

# Some useful commands for matrices and data frames

- **cbind**: adds a column
  cbind(d, 1:3)

- **rbind**: adds a row
  rbind(d, 4:6)

- **rowSums/colSums**: processes the sum of rows/columns of a matrix
  rowSums(b); colSums(b)

# Exercise 4:
## *matrix manipulation*

*Exercise 3*

# Matrix and data frame manipulation

- Read in file "" into object "z"
  - What is the structure of z ? **str(), data.class()**
  - What are the dimensions of z? **nrow(), ncol(), dim(), str(**
- Convert **z** into a matrix **m. as.matrix()**
  - What is the structure of m ? **str(), data.class()**
- Create a new vector that **subtracts the elements of the first column of z from the elements of the second column of z**

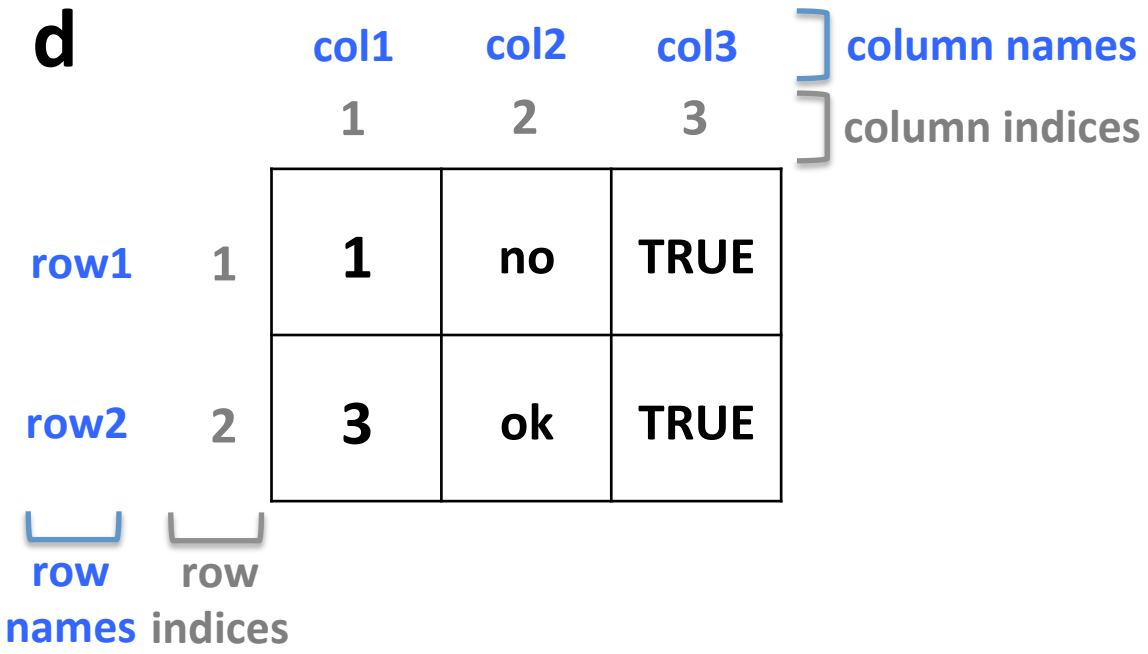
# Dimension names

- Column and / or row names can be added to matrices and data frames:

```
colnames(d) <- c("numb", "charac", "logic")
rownames(d) <- c("row1", "row2", "row3")
```

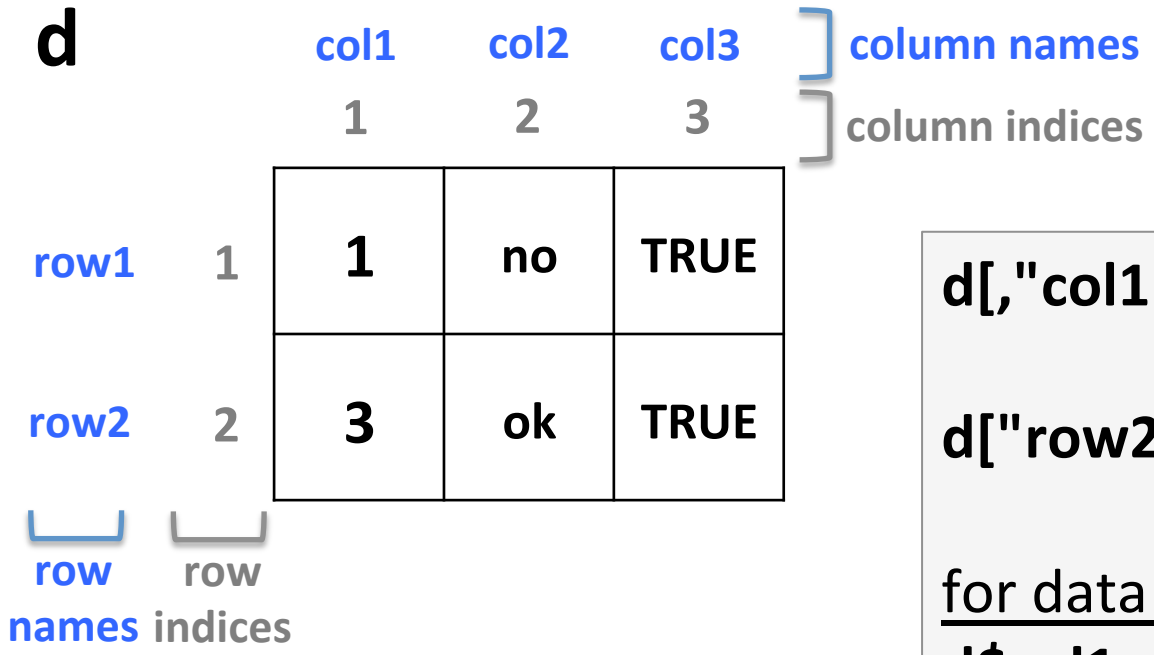
# Dimension names

- Instead of indices, you can use dimension names to subset a matrix or a data frame:



# Dimension names

- Instead of indices, you can use dimension names to subset a matrix or a data frame:




**d[,"col1"] is d[,1]**

**d["row2","col1"] == d[2,1]**

for data frames:

**d$col1 == d[,1] == d[,"col1]**

# the **apply** function

- Powerful tool to **apply a command** to all rows or all columns of a data frame or a matrix

**d**

| col2 | col3 |
|---|---|
| no | TRUE |
| ok | TRUE |

**apply(d, 2, table)**

```
$col2
no ok
 1  1

$col3
TRUE
   2
```

# Input / Output

- a <- **read.table**("file.txt",
  sep="\t", **# input column field separator**
  header=TRUE **# first row of file considered as header / column names vector**
  )

- **write.table**(a, "file.txt",
  sep="\t", **# output column field separator**
  col.names=TRUE, **# col names written in output file**
  row.names=TRUE, **# row names written in output file**
  quote=FALSE) **# quote around character elements not written in output file**

# Exercise 4:
## *Data frame manipulation*

*Exercise 4*

# R script with matrix and data frame

**Create** an empty file **ex4.R**

**Within that file:**

- Read in file **TO ADD** into object "z"
- What are the dimension names of z ? **dimnames()**
- Change column names of z to "plus3" and "times4" – **colnames()**
- **add 3** to each element of column 1
- **multiply by 4** each element of column 2

**Write** z into new file "" – **write.table()**

# Outline

## *day 3 – May 29th*

- Lists
- Library / packages
  - CRAN
  - Bioconductor
- (Writing functions in R)

# Lists

- Linear structures.

- A component of a list can be **any data structure:**

  → matrix, vector, data frame, list…

# Lists

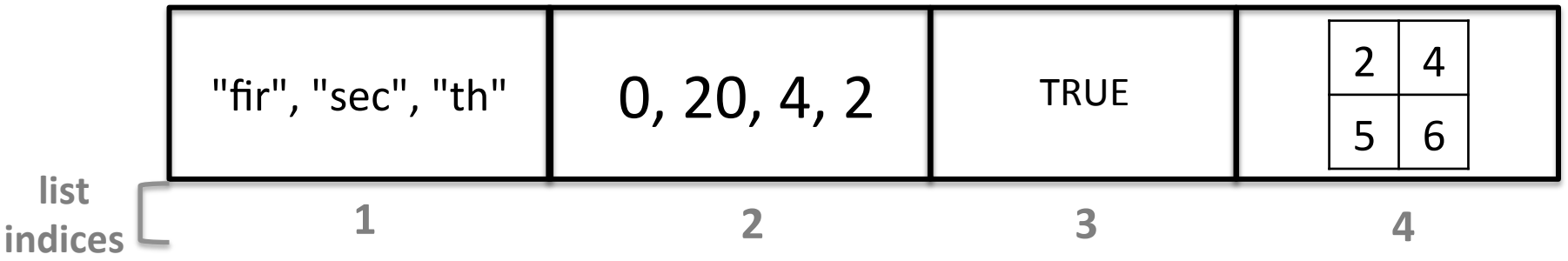
- Create a list:

```
e <- list()     # Empty list that can be filled later on

e <- list(c("fir", "sec", "th"),     # 1st element of the list
          c(0,20,4,2),               # 2d element of the list
          TRUE,                      # 3d element of the list
          matrix(c(2,5,4,6), nrow=2,         # 4th element of the list
ncol=2))
```

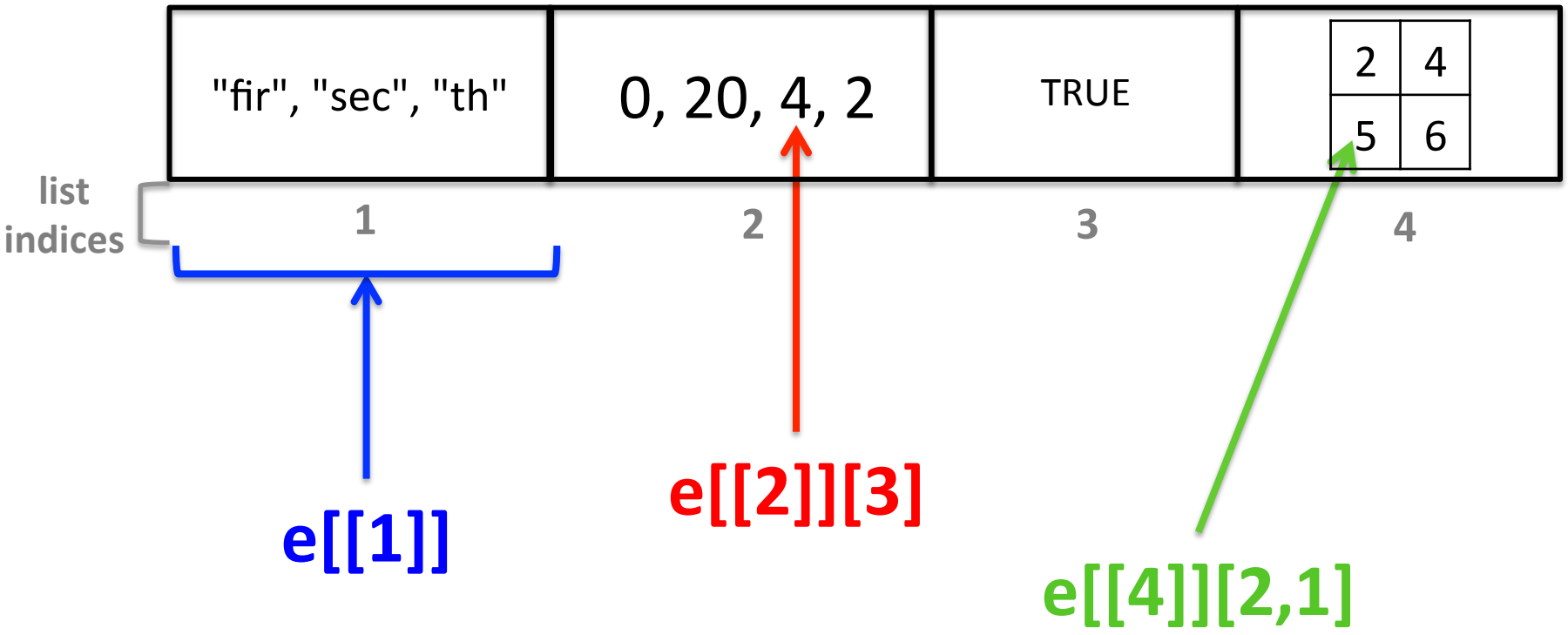
# Lists

- Accessing elements of a list **e**:



# Lists

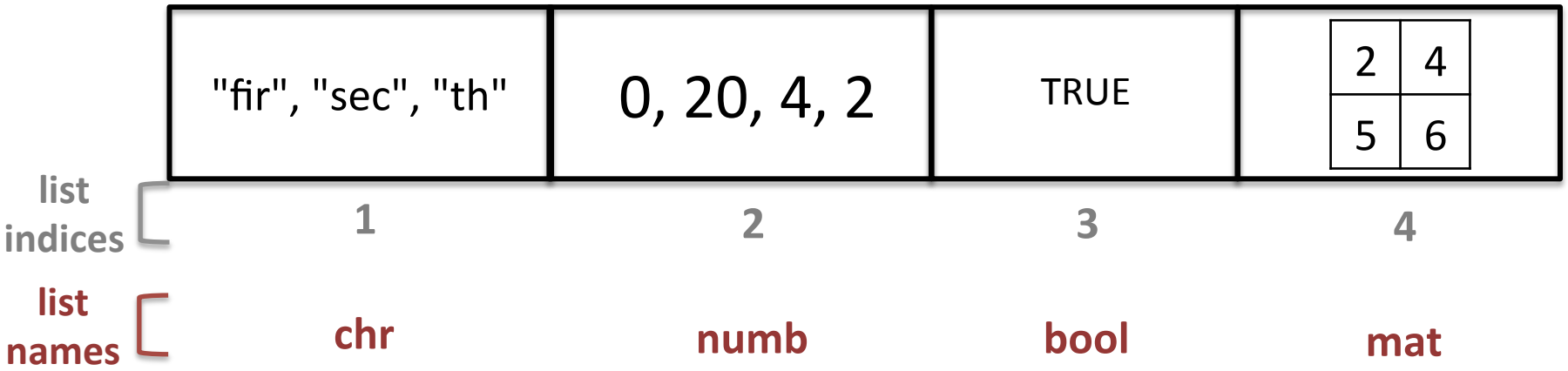
- Accessing elements of a list **e**:



# Lists

- Each element of a list has a name
- **names**(e)
- names(e) <- c("char", "numb", "bool", "mat")

# Lists




- e[["mat"]] == e$mat == e[[4]]

```
$mat
     [,1] [,2]
[1,]    2    4
[2,]    5    6
```

# Exercise 6:
## *List manipulation*

***Exercise 5***

# List manipulation

- Create:

  y <- list("a", "b", "c")
- Assign new names to y: "A" "B" "C" – **names()**
- Replace the third element of y with a numeric matrix of 2x2 dimensions

# Libraries / packages

# Library/packages

- **Packages** are collections of R functions, data, and compiled code in a well-defined format.

- The directory where packages are stored is called the **library**.

*Definitions from http://www.statmethods.net/interface/packages.html*

# R-base: standard packages

- **R-base**:

  about 25 standard packages supplied with R by default (example: base, stats, graphics).

# R-contrib: all other packages

- 2 main repositories:

  – CRAN: Comprehensive R Archive Network

  **10669 packages** available

  ***https://cran.r-project.org***

  – Bioconductor:

  **1383 packages** available

  ***https://www.bioconductor.org/***

# Find packages

- R packages in CRAN:

***https://cran.r-project.org/web/packages/***

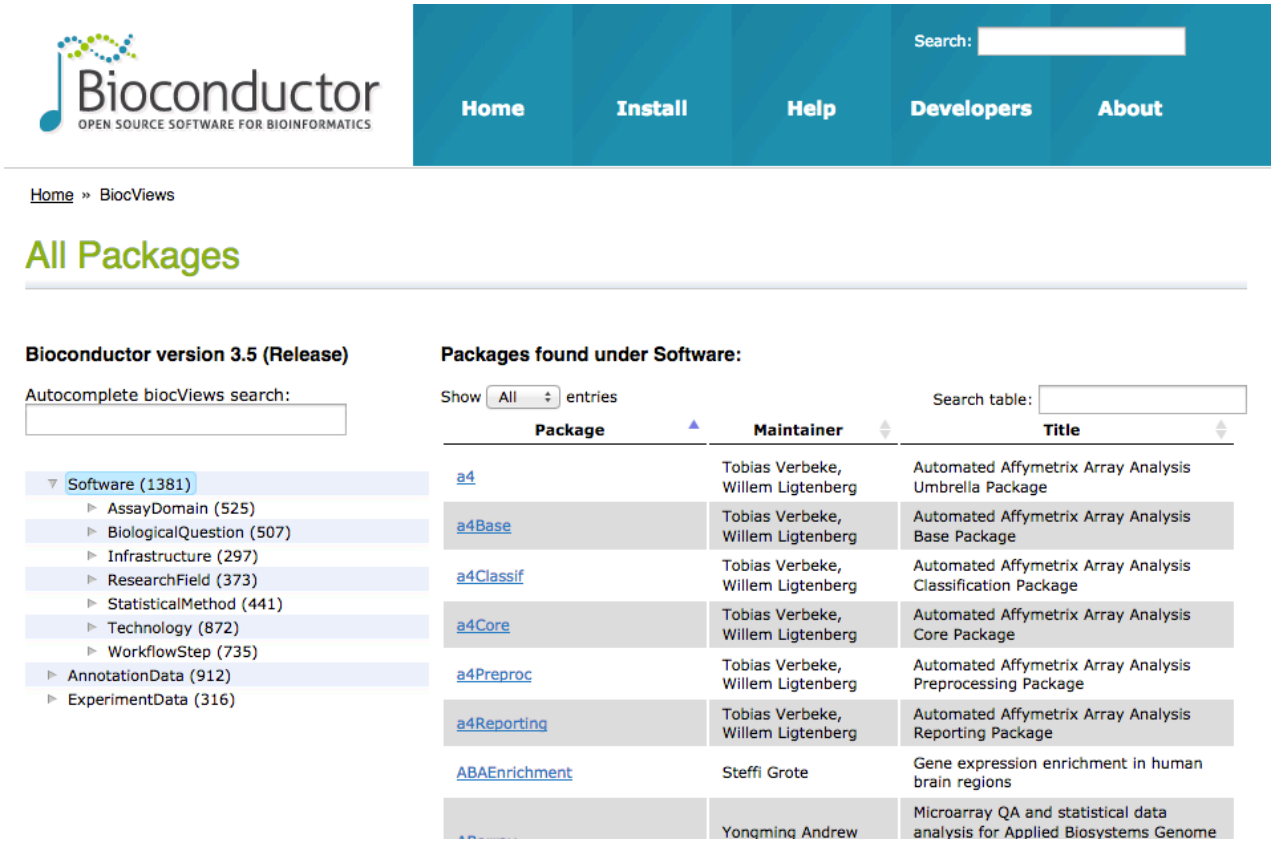
**Available CRAN Packages By Date of Publication**

| Date | Package | Title |
|---|---|---|
| 2017-05-21 | aSPU | Adaptive Sum of Powered Score Test |
| 2017-05-21 | d3r | 'd3.js' Utilities for R |
| 2017-05-21 | dynr | Dynamic Modeling in R |
| 2017-05-21 | edfReader | Reading EDF(+) and BDF(+) Files |
| 2017-05-21 | GlobalOptions | Generate Functions to Get or Set Global Options |
| 2017-05-21 | GWmodel | Geographically-Weighted Models |
| 2017-05-21 | IgorR | Read Binary Files Saved by 'Igor Pro' (Including 'Neuromatic' Data) |
| 2017-05-21 | imputeTS | Time Series Missing Value Imputation |
| 2017-05-21 | lagged | Classes and Methods for Lagged Objects |
| 2017-05-21 | mclust | Gaussian Mixture Modelling for Model-Based Clustering, Classification, and Density Estimation |
| 2017-05-21 | NPMOD | Non Parametric Module |
| 2017-05-21 | numGen | Number Series Generator |
| 2017-05-21 | penRvine | Flexible R-Vines Estimation Using Bivariate Penalized Splines |
| 2017-05-21 | PhenotypeSimulator | Flexible Phenotype Simulation from Different Genetic and Noise Models |
| 2017-05-21 | plfMA | A GUI to View, Design and Export Various Graphs of Data |
| 2017-05-21 | RANN | Fast Nearest Neighbour Search (Wraps ANN Library) Using L2 Metric |
| 2017-05-21 | regnet | Network-Based Regularization for Generalized Linear Models |
| 2017-05-21 | timereg | Flexible Regression Models for Survival Data |
| 2017-05-20 | AIG | Automatic Item Generator |

# Find packages

- R packages in Bioconductor:

***https://bioconductor.org/packages***



# Bioconductor

- R packages specialized in bioinformatics analysis
  - Supports most types of genomics and NGS data (e.g. limma, DESeq2, BayesPeak)
  - Specific data classes (e.g. Granges from GenomicRanges)
  - Integrates command line tools (e.g Rsamtools)
  - Annotation tools (e.g. biomaRt)

# Types of Bioconductor packages

- **Software**: set of functions

  e.g. limma (microarray data analysis)

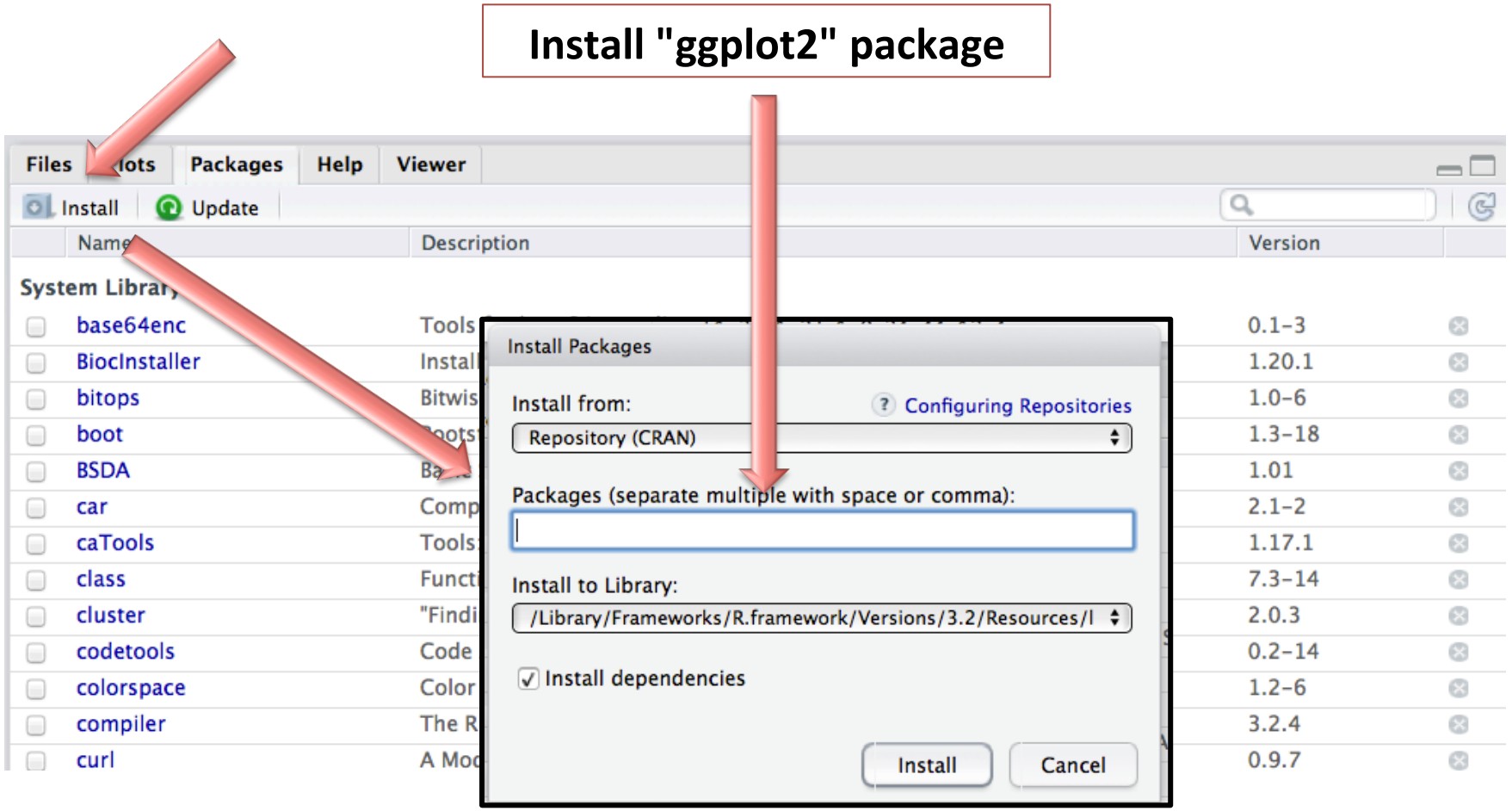
- **Annotation**: annotation of specific arrays, organisms, events, etc.

  e.g. BSgenome.Hsapiens.UCSC.hg38

- **Experiment**: data that can be loaded and used

  e.g. ALL (acute lymphoblastic leukemia dataset)

# Install a package with R studio

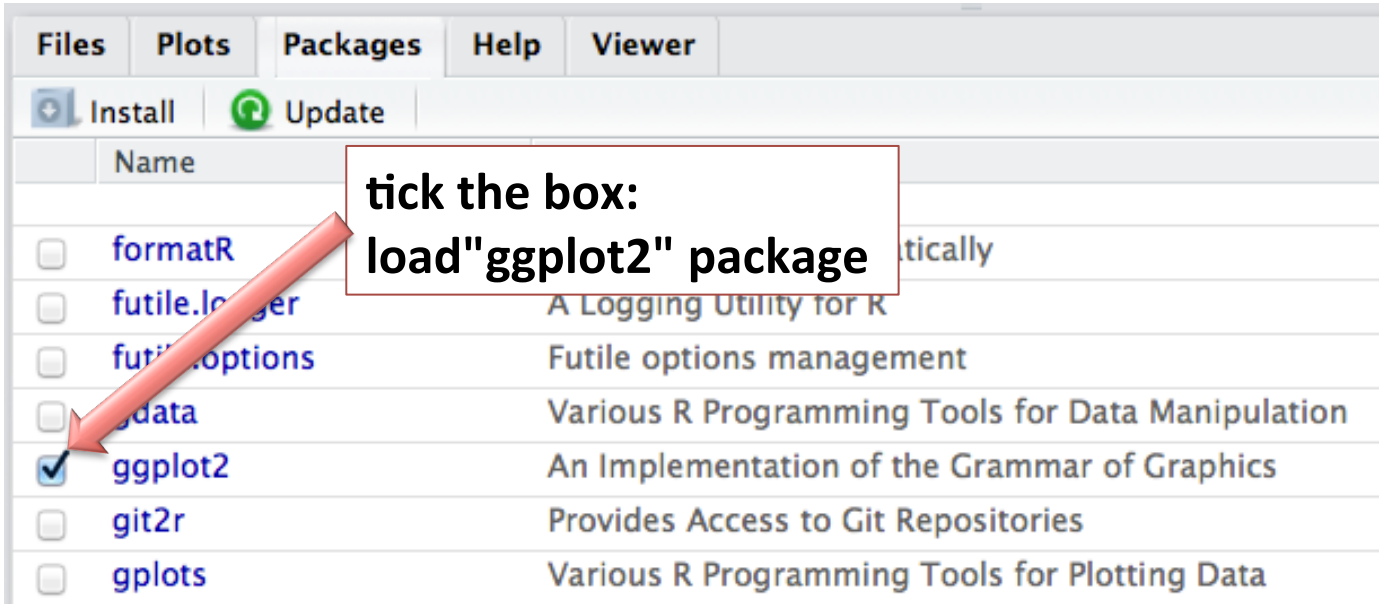


# Load a package

- **from console**

  **library(**ggplot2**)**

- **with RStudio**



# Listing functions from packages

- **ls(**"package:ggplot2"**)**

```
 [1] "%+%"                  "%+replace%"
 [3] "Coord"                "CoordCartesian"
 [5] "CoordFixed"             "CoordFlip"
 [7] "CoordMap"               "CoordPolar"
 [9] "CoordQuickmap"            "CoordTrans"
[11] "Geom"                  "GeomAbline"
[13] "GeomAnnotationMap"       "GeomArea"
[15] "GeomBar"               "GeomBlank"
```

…

# R Studio server at CRG

- You need to specify a writeable directory to install packages into!
- Follow these steps:

```
setwd("~") # go to home directory
dir.create("R_packages") # create directory where to store packages
.libPaths("~/R_packages/") # add path to library path
```

# Exercise 7:
# *Packages*

***Exercise 6***

# Packages

- Install and load packages **ggplot2** and **WriteXLS**
- ggplot2 contains a dataset called **diamonds**

  what are diamonds' dimensions?
- Store the first 200 rows of diamonds in **diams2**
- Write that data frame into "diamonds200.xls" – **WriteXLS**()

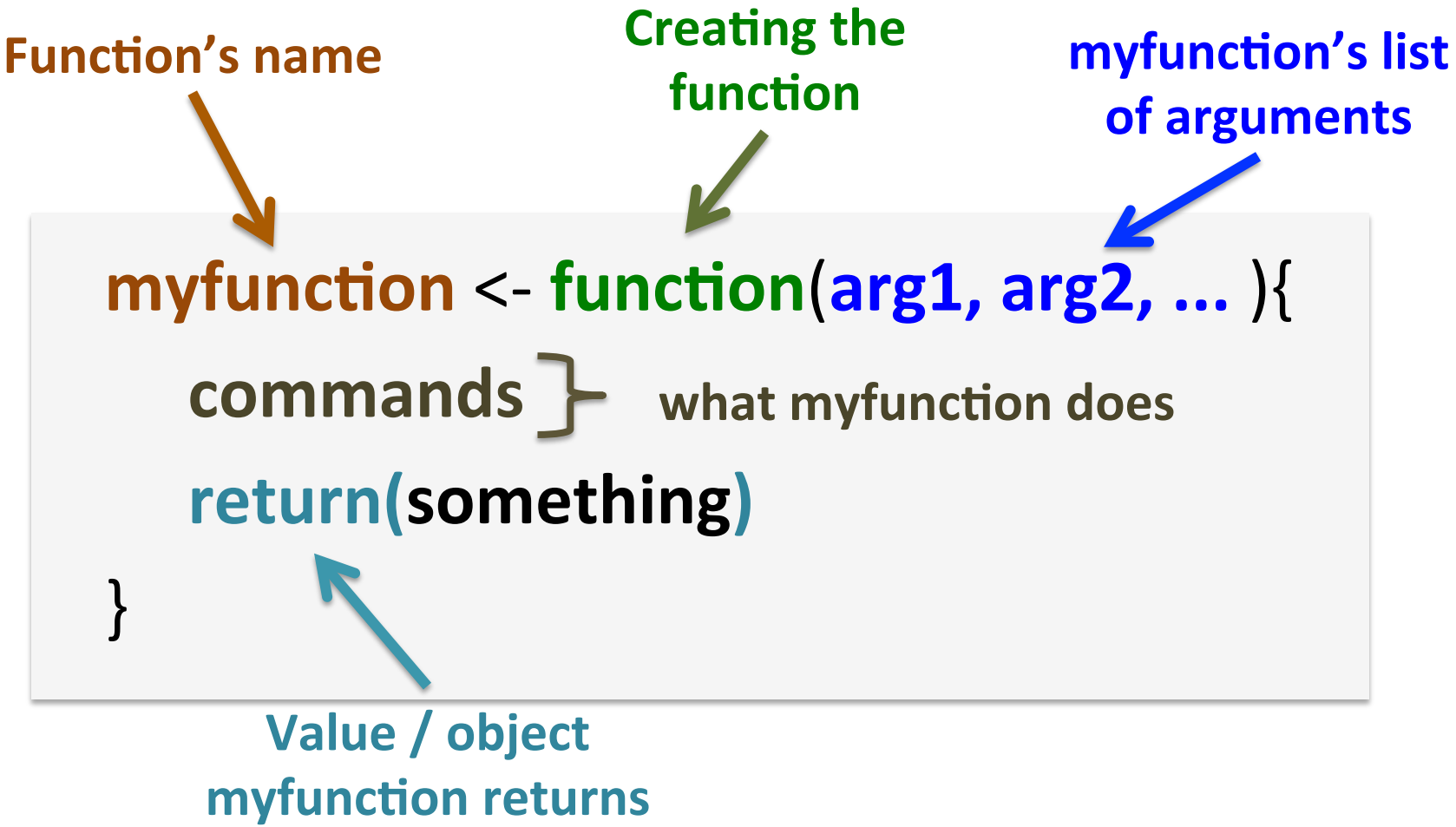
# Writing functions in R

# User-written functions

- Functions are pieces of code written **to carry out specified tasks** and be able to repeat them easily.

- R allows you to create your own functions.

# Functions' structure



# Objects in functions

```
myfunction <- function(arg1){
    a <- arg1
    return(a+1)
}
```

```
> myfunction(10)
[1] 11
> a
Error: object 'a' not found
```

```
> a <- 12
> myfunction(10)
[1] 11
> a
[1] 12
```

# Exercise optional 1: Functions

***Exercise 5***

# Functions

- Create an empty **myfunctions.R** file
- Write a function that outputs the **minimum** and a **maximum** of a numerical vector.
- Save myfunctions.R
- Source **myfunctions.R** with the **source** function.
- Test your function on a simple vector, for example:

**ve <- c(0, 3, -2, 1.5, 8, 4.3, -2.1)**

# Outline

## *day 4 – May 30th*

- Graphing in R:
  - basic graphing
  - ggplot2 package

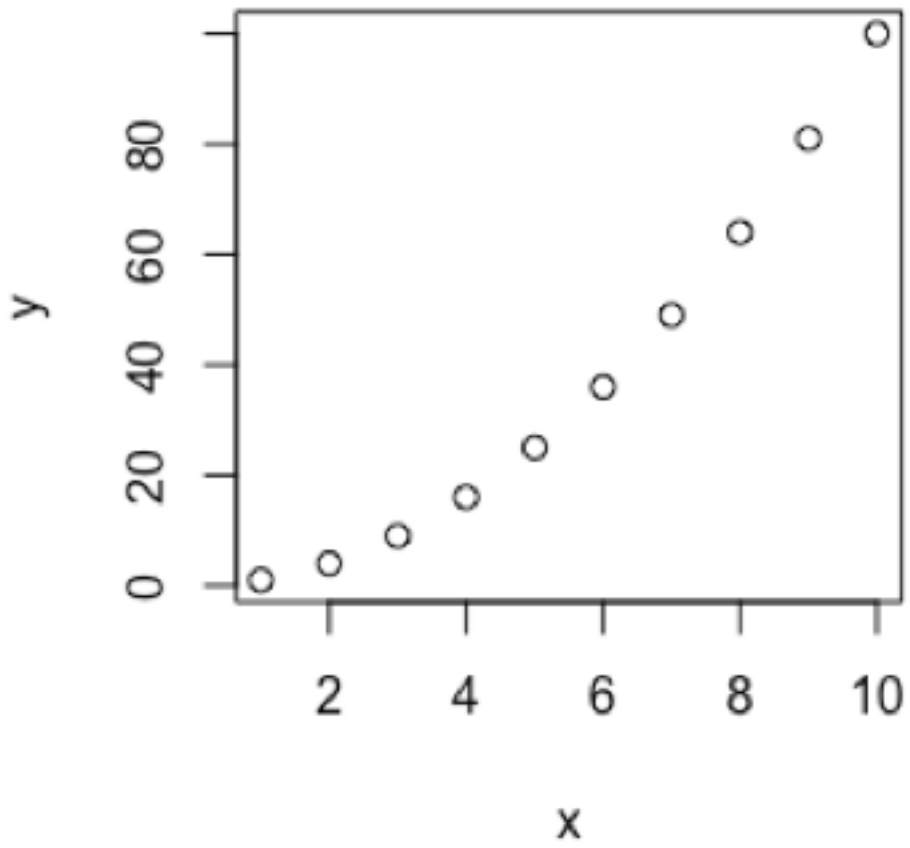
# Graphing in R:
## basic graphing

# Producing graphs

- R-base package graphics offers functions for producing many plots:
  - scatter plots – plot()
  - density plots – density()
  - histogram – hist()
  - boxplots – boxplot()
  - pie charts – pie()

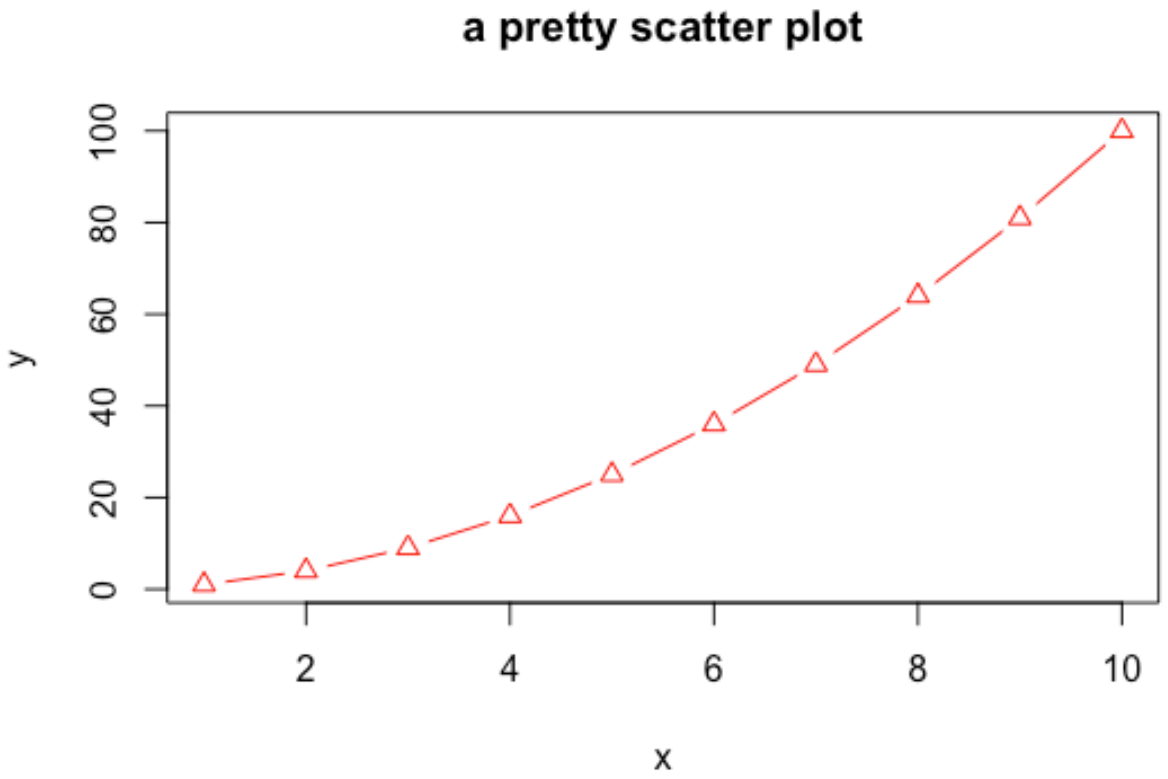
# Basic scatter plot

- x <- 1:10
- y <- x^2
- **plot(x, y)**



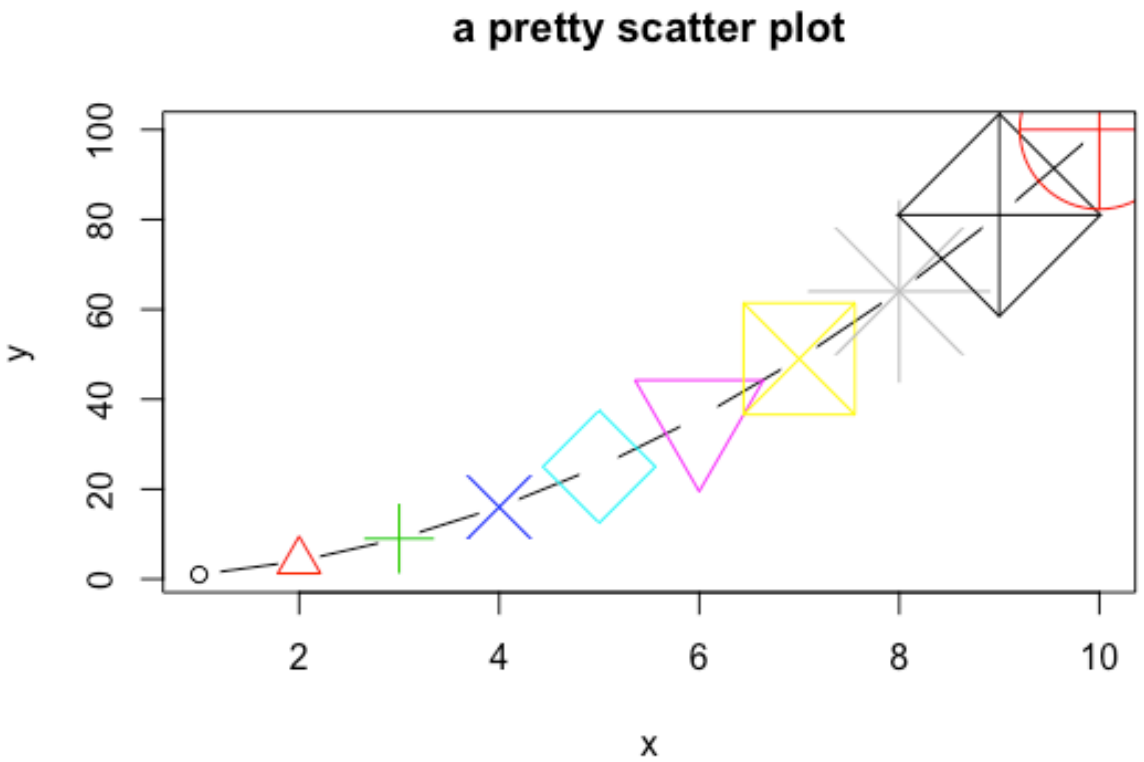
# Scatter plot

- **plot(x, y, col="red", pch=2, type="b", main="a pretty scatter plot")**



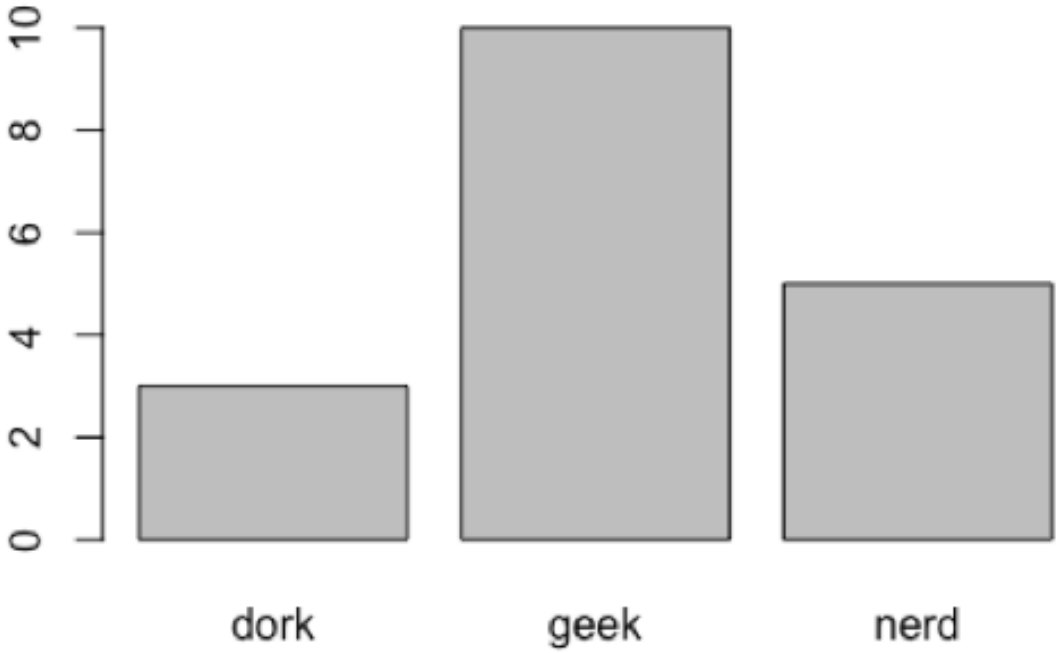
# Scatter plot

- **plot(x, y, col=1:10, pch=1:10, cex=1:10, type="b", main="a pretty scatter plot")**



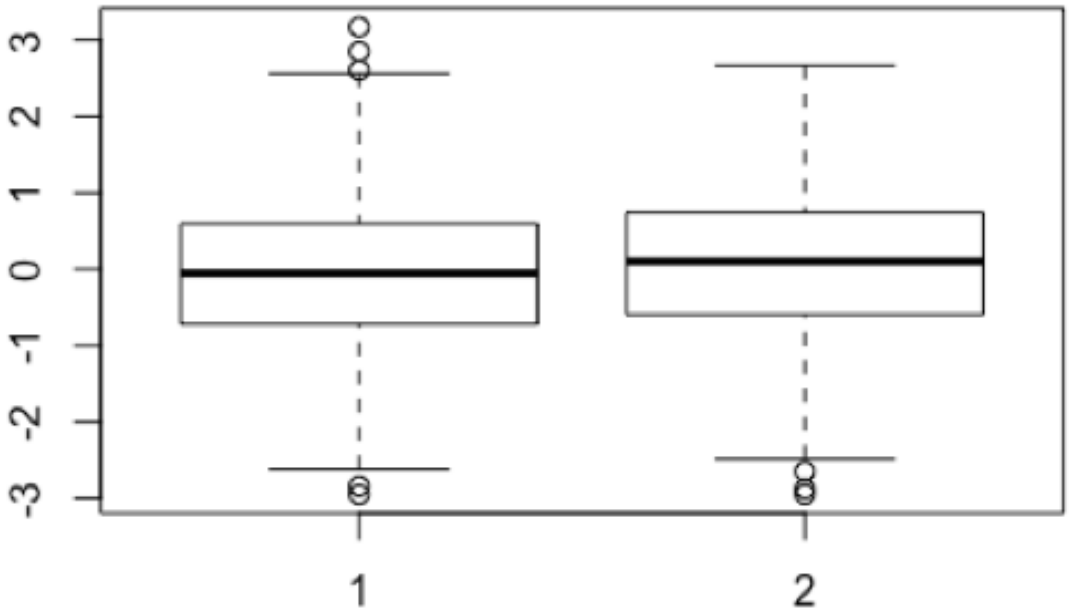
# Barplot

- x <- rep(c("geek", "nerd", "dork"), c(10,5,3))
- **barplot**(table(x))



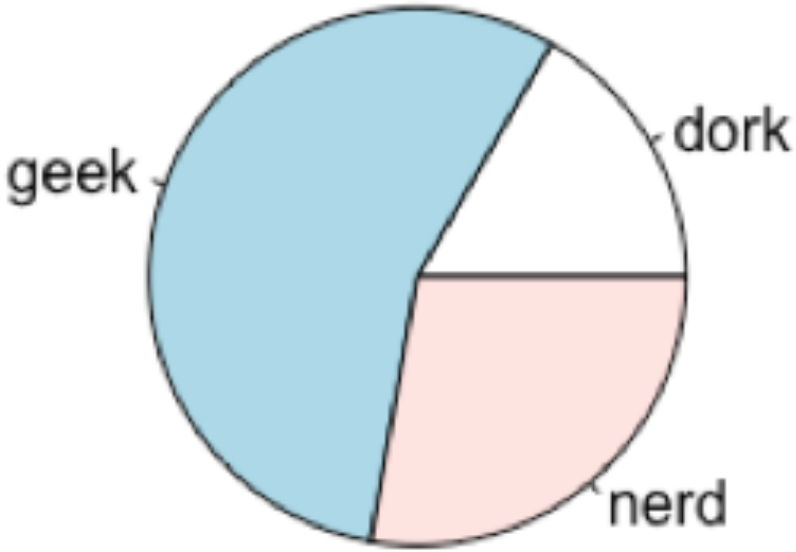
# Boxplots

- x <- matrix(rnorm(1000), ncol=2)



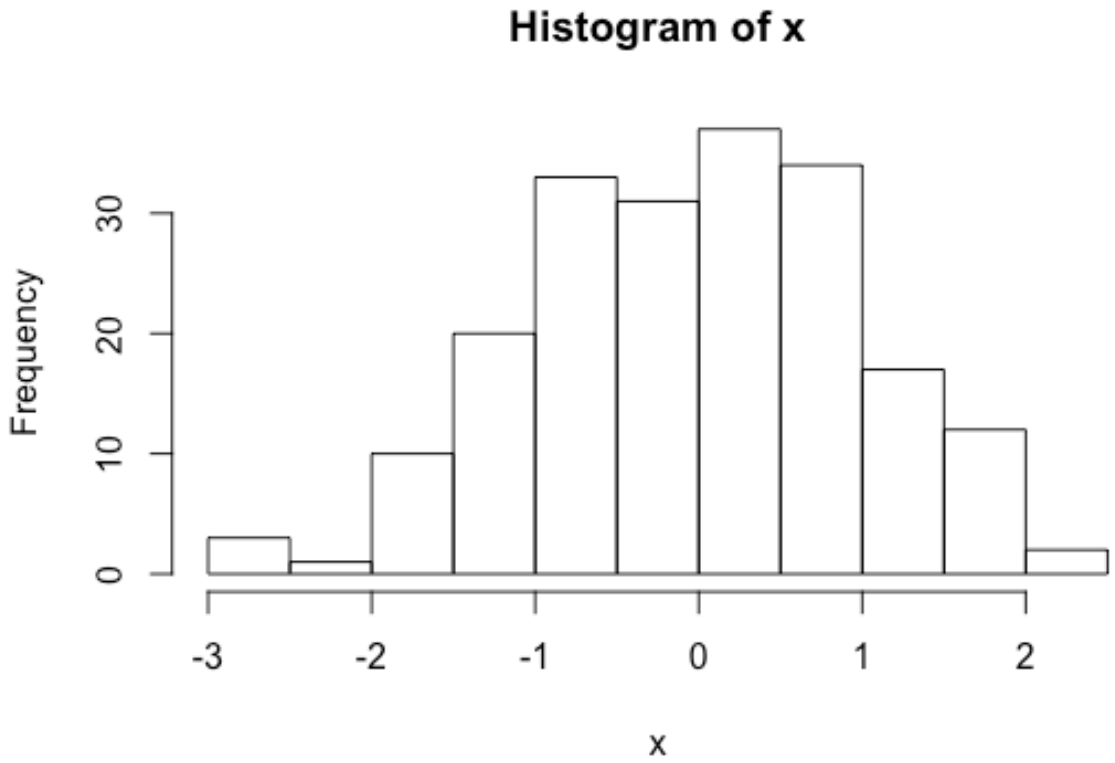
# Pie plot

- x <- rep(c("geek", "nerd", "dork"), c(10,5,3))
- **pie**(table(x))



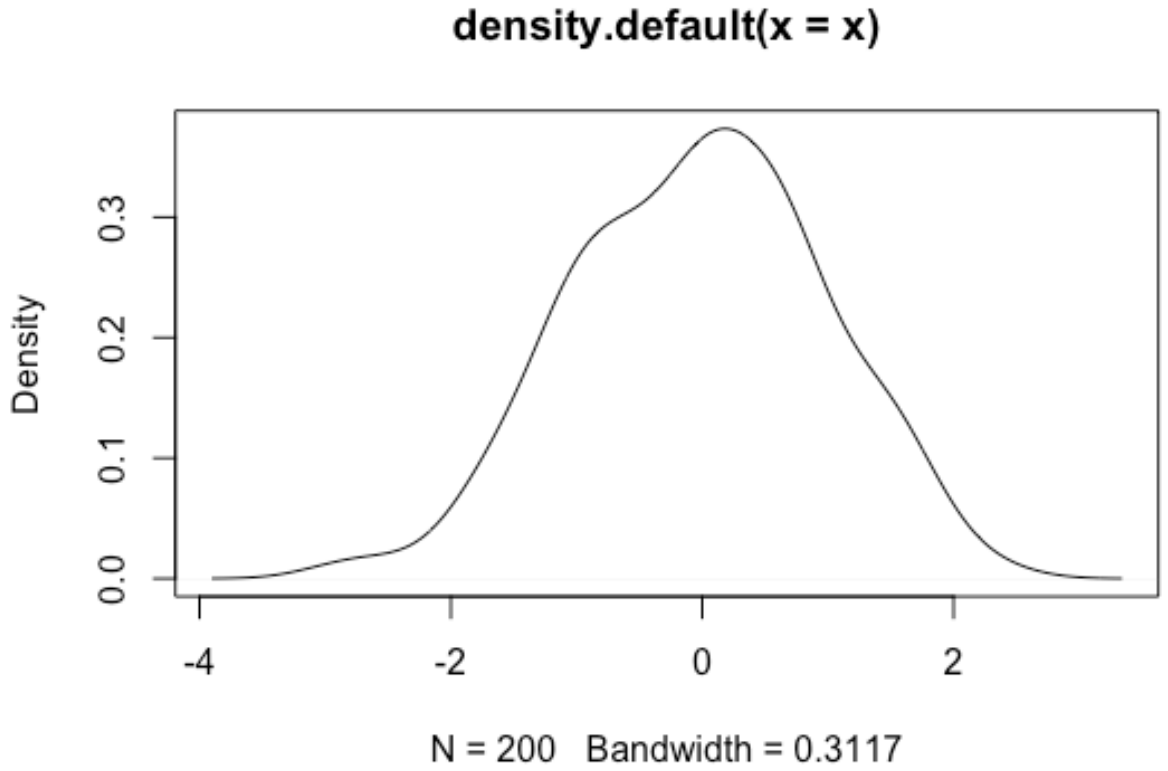
# Histogram

- x <- rnorm(200)
- **hist**(x)



# Density plot

- x <- rnorm(200)
- **plot(density**(x))



# Exporting graphs

- Graphs can be saved into many formats, including: **pdf, jpeg, bmp, tiff**.
- Open the file to save your plot in:

```
pdf("my_graph.pdf")
```

- Produce a graph:

```
plot(1:10)
```

- And close it:

```
dev.off()
```

# Exporting graphs

- From R Studio:

# Exercise 8:
# Basic plots

# Graphing in R:
## Introduction to ggplot2 package

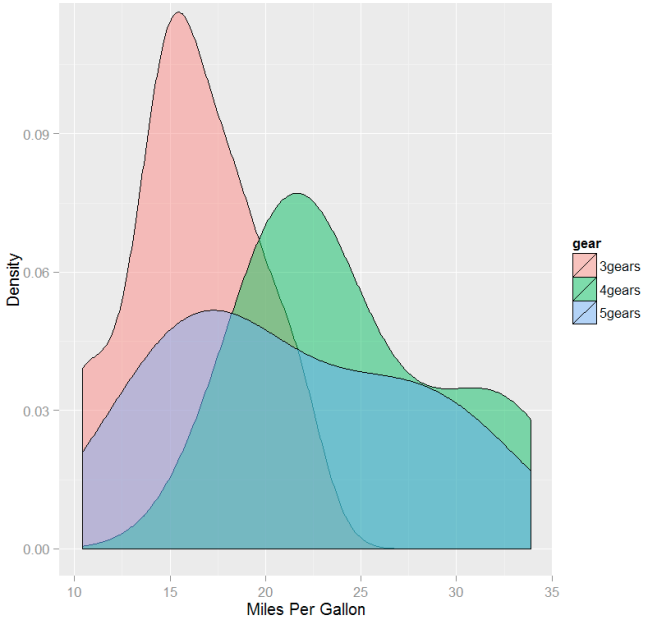
# ggplot2

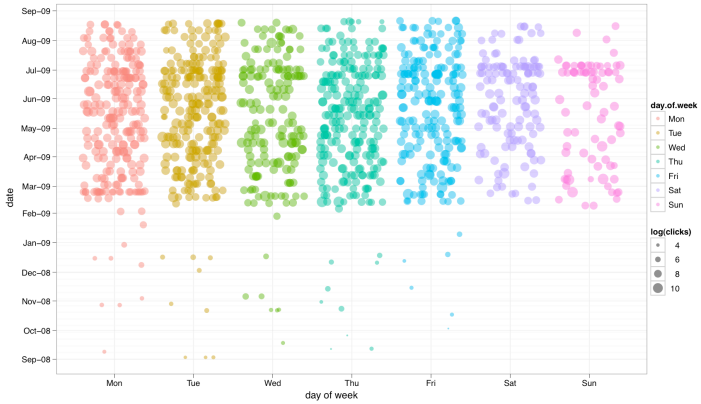
- Graphing package inspired by the Grammar of Graphics seminal work of Leland Wilkinson.

- A tool that enables to concisely describe the components of a graphic.

# ggplot2

- Flexible
- Customizable
- Pretty
- Well documented



# ggplot2

- Base layer to plot two variables:

**ggplot(dataframe, aes(x, y))**

- Add layer specifying what kind of plot you want.

  Example of a scatter plot:

**ggplot(dataframe, aes(x, y)) + geom_point()**

# Exercise 3:
## First *ggplot2* plot

***Exercise 3***

# ggplot2

- Create a data frame containing columns **x** and **y**, each a random generation of 100 normal distribution values – **rnorm**().

- Produce a scatter plot using variable **x** and **y** from the data frame.

- Save plot in a **jpeg** file.

# Dataset

- **diamonds** data set in ggplot2 package

- **dim**(diamonds); **colnames**(diamonds); **head**(diamonds)

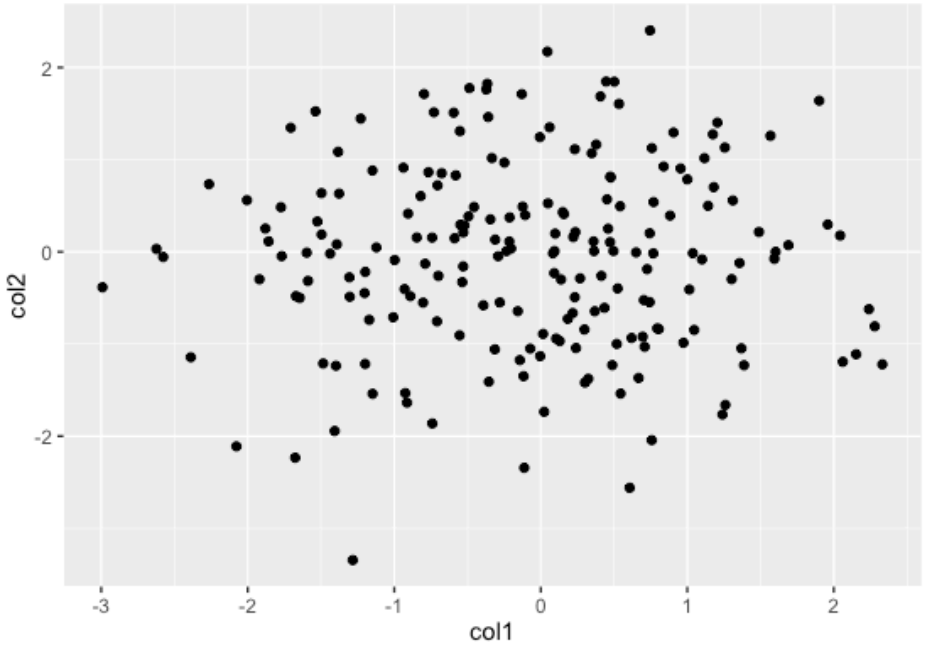
- subset diamonds:

```
diams2 <- diamonds[sample(rownames(diamonds), 1000), ]
```

# Scatter plots

- x <- data.frame(col1=rnorm(200), col2=rnorm(200))

- ggplot(x, aes(col1, col2)) + geom_point()



# Scatter plots

- Store the graph in an object:

```
p <- ggplot(x, aes(col1, col2)) + geom_point()
```

- add "layers" to p:
  - title with ggtitle("my title")
  - vertical line with geom_vline(xintercept=2)

# Scatter plots

- Color according to a factor:

```
x <- data.frame(col1=rnorm(200), col2=rnorm(200),
col3=rep(c("yes", "no"), c(80, 120)))

p <- ggplot(x, aes(col1, col2, color=col3)) + geom_point()
```

# Scatter plots

- p <- ggplot(x, aes(col1, col2, color=col3)) + geom_point()

- END

# Scatter plots

*A scatter plot is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data.*

*[Wikipedia]*

# Scatter plots

- p <- **ggplot**(diams, aes(carat, price, **color**=cut)) + **geom_point**()

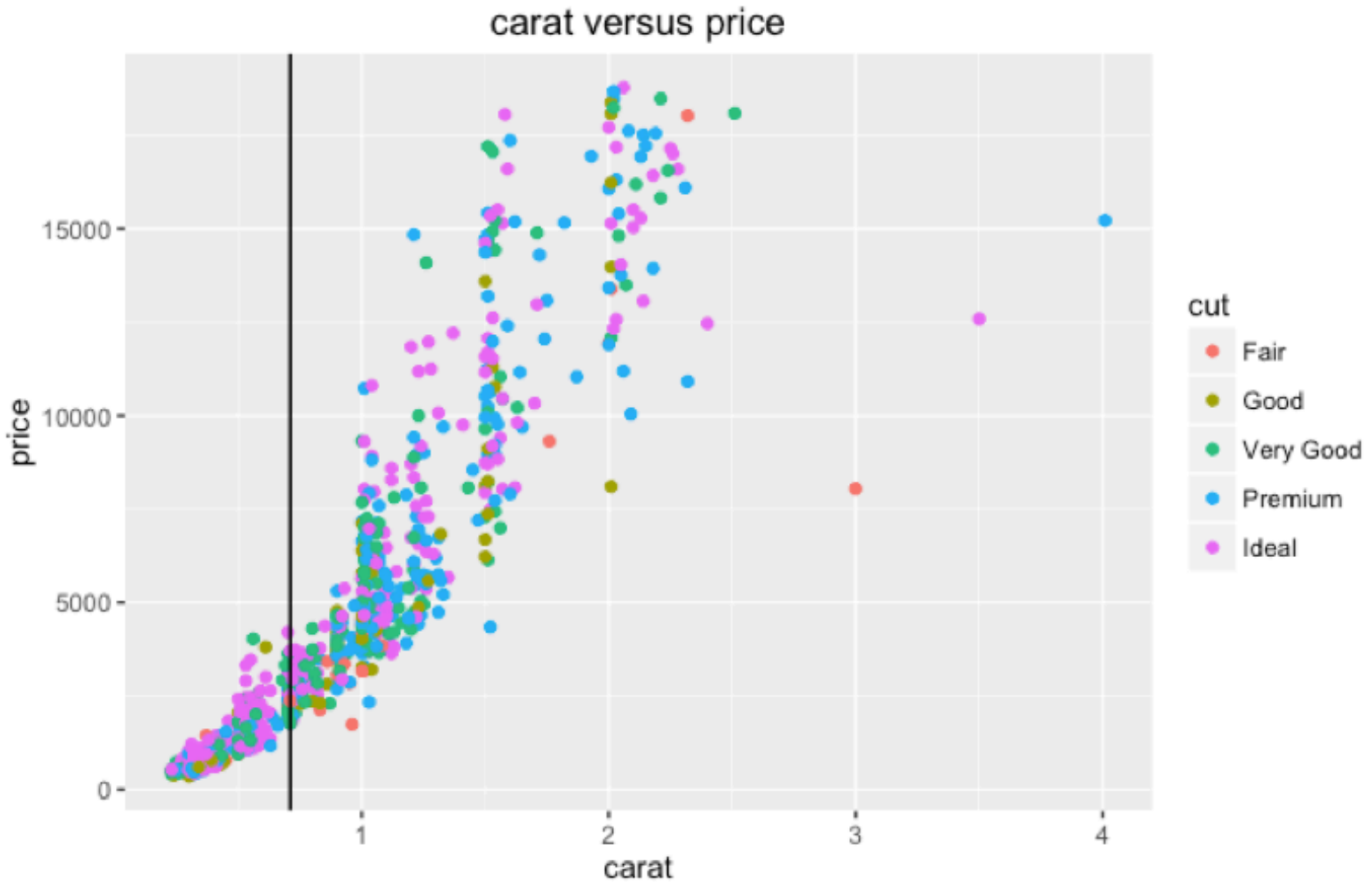
- add title with **ggtitle**

  p2 <- p + **ggtitle**("carat versus price")

- draw a vertical line at the median carat value with **geom_vline**

  p3 <- p2 + **geom_vline**(xintercept=median(diams2$carat))

# Scatter plots



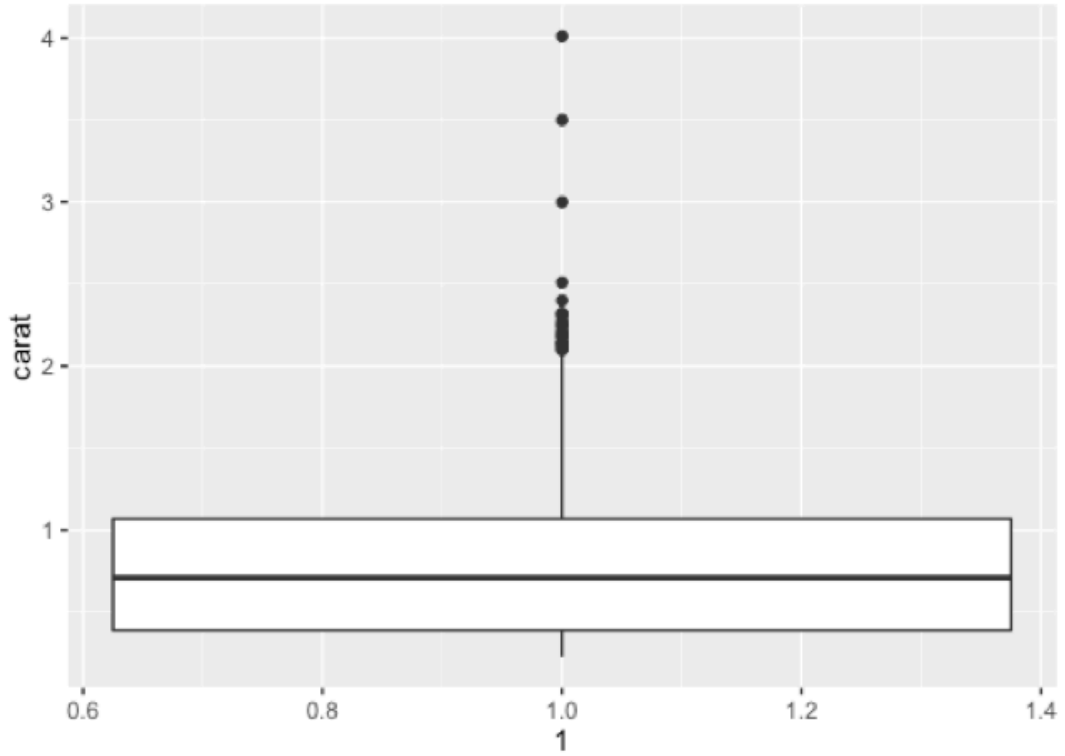
# Boxplots

*In descriptive statistics, a boxplot is a convenient way of graphically depicting groups of numerical data through their quartiles.*

*[Wikipedia*]

# Boxplots

- **ggplot**(diams2, aes(x=1, y=carat)) + **geom_boxplot**()

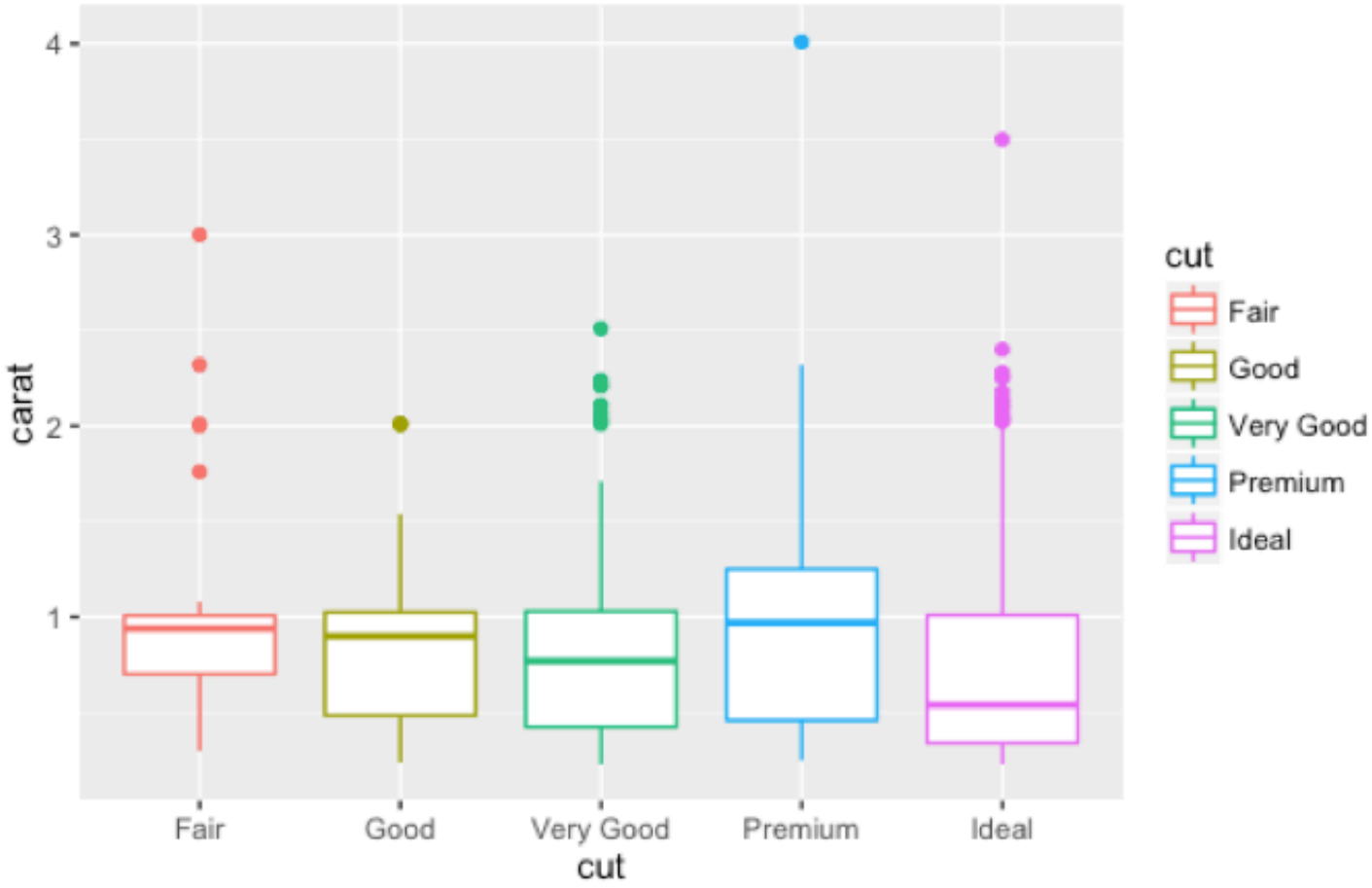


# Boxplots

- p <- **ggplot**(diams2, aes(x=cut, y=carat)) + **geom_boxplot**()

- add colors to the boxes lines with **color** in aes
  - p <- **ggplot**(diams2, aes(x=cut, y=carat, **color**=cut)) + **geom_boxplot**()

# Boxplots


4
3
2
1
carat
Fair
Good
Very Good
Premium
Ideal
cut
cut
Fair
Good
Very Good
Premium
Ideal

# Barplots

*A bar chart or bar graph is a chart that presents grouped data with rectangular bars with lengths proportional to the values that they represent. [Wikipedia]*

# Barplots

- p <- **ggplot**(diams2, aes(x=cut)) + **geom_bar**()

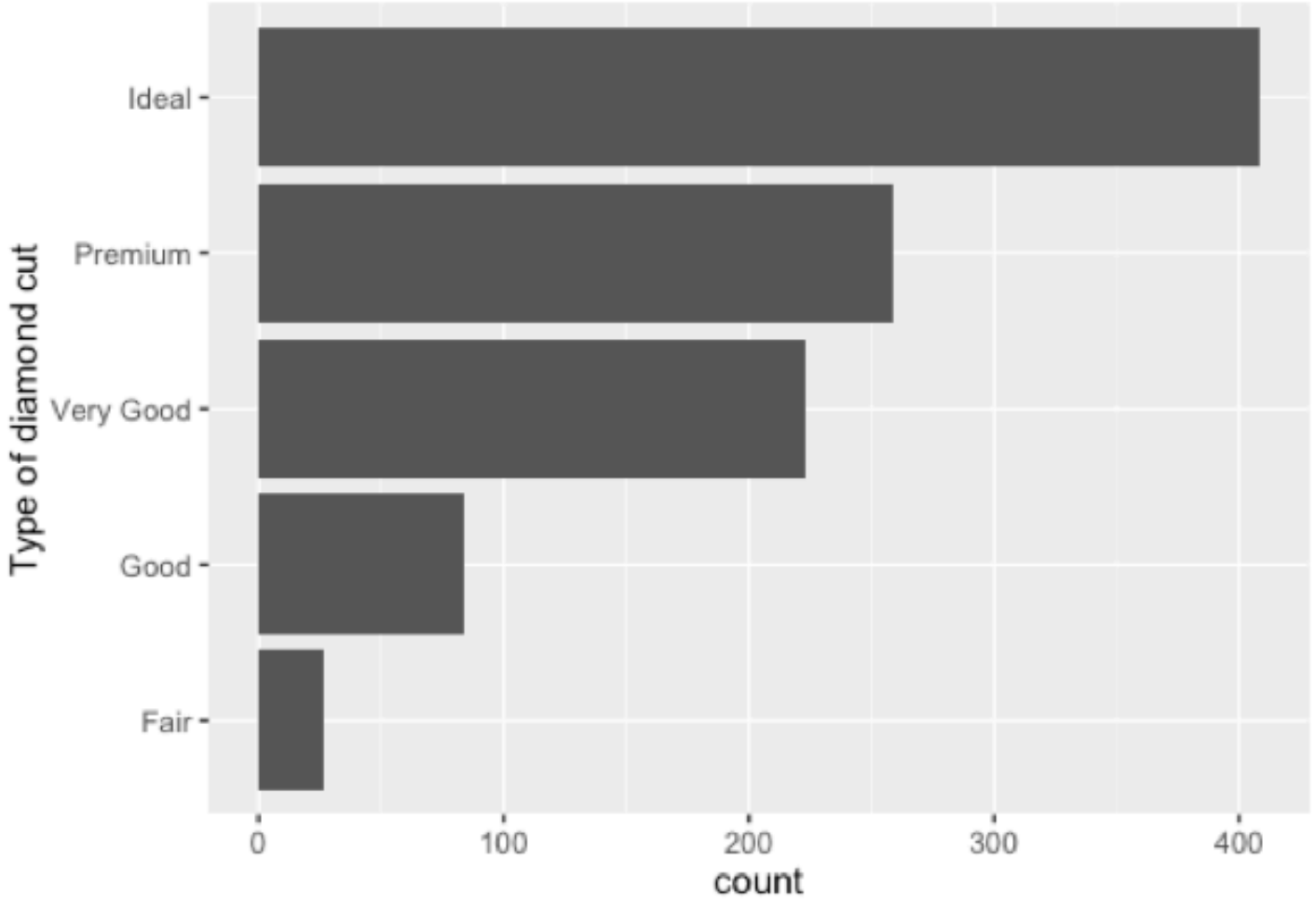
- Change x axis label with **scale_x_discrete**:

  p2 <- p + **scale_x_discrete**(name="Type of diamond cut")

- Swapping x and y axis with **coord_flip**():

  p3 <- p2 + **coord_flip**()

# Barplots


Ideal
Premium
Very Good
Good
Fair
Type of diamond cut
0
100
200
300
400
count

# Histograms

*A histogram is a plot that lets you discover, and show, the underlying frequency distribution of a set of continuous data.*

*[Wikipedia*]

# Histograms

- **ggplot**(diams2, aes(x=price)) + **geom_histogram**()

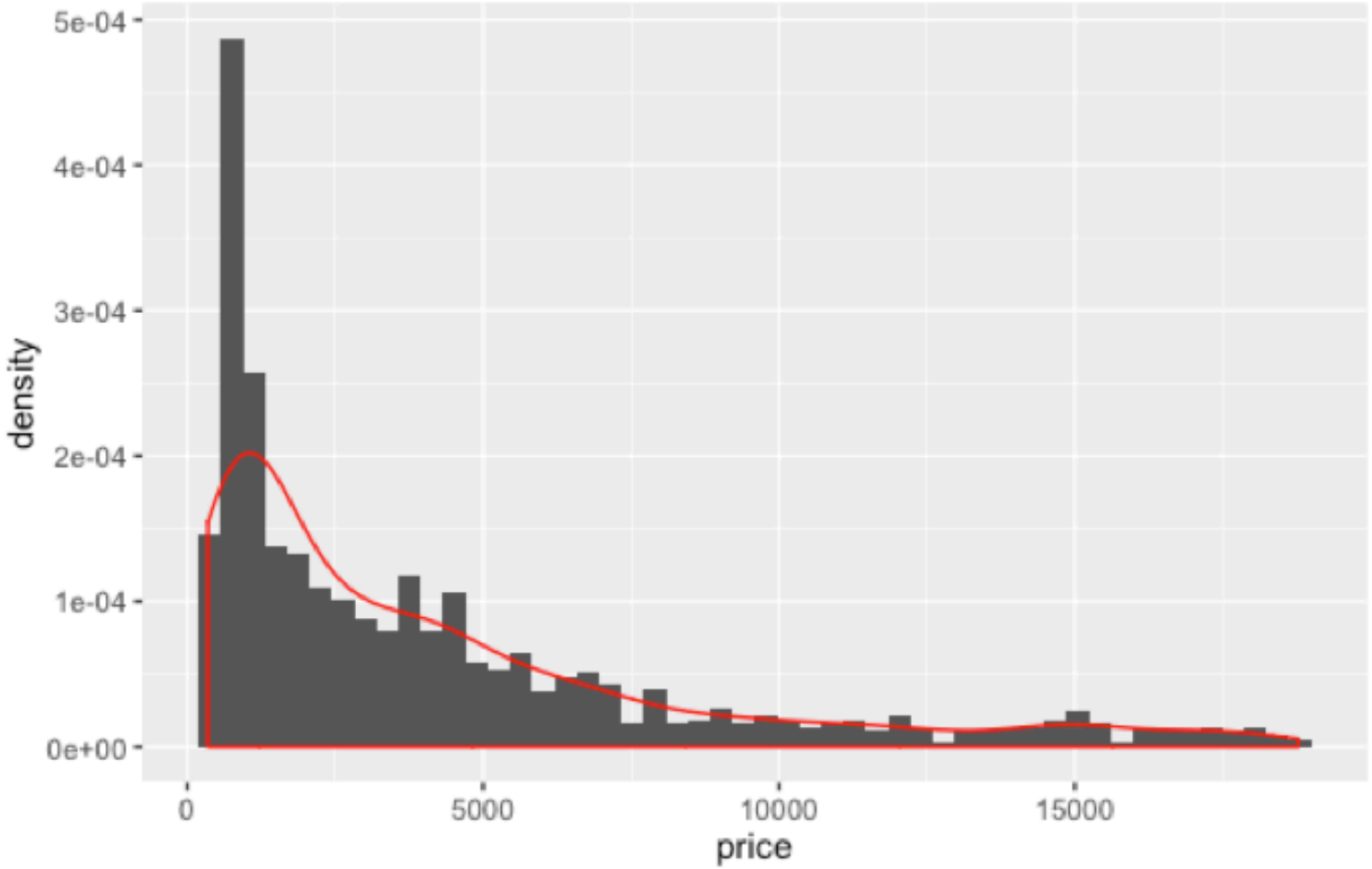
- Increase resolution with **bins** option:

```
ggplot(diams2, aes(x=price)) + geom_histogram(bins=50)
```

- Add a density curve:

```
ggplot(diams2, aes(x=price))
    + geom_histogram(aes(y=..density..), bins=50)
    + geom_density(color="red")
```

# Histograms


5e-04
4e-04
3e-04
2e-04
1e-04
0e+00
density
0
5000
10000
15000
price