Lecture 01

An R Introduction to Statistics

csci E63 Big Data Analytics

Zoran B. Djordjević

About this Course

- This is an IT (Information Technology) course.

- This course uses techniques of Statistics, Mathematics and Machine Learning.

- We will teach you how to install and use most important frameworks used for modern analysis of big data sets.

- We will use some sophisticated statistical and machine learning algorithms.

- You need to take separate courses in Statistics, and Machine Learning to become a Big Data Scientist.

- Mathematics is too big, just learn fractions.

Python, R, Java, Scala, ….

- Which language should you use in the analysis of big data?

- You can use any: C, C++, C#, Java, R, Python, Scala, Julia, Go, MatLab, …..

- Your life is a never ending continuous education.

- A lot depends on the environment, the company, where you work. Some have adopted Python, some would never part with R.

- One has an impression that many developers are migrating to Python. A lot of literature, many new APIs and frameworks.

- Python is not a brand new language but got a lot of wind from Big Data and Machine Learning community, recently.

- You can still do anything and everything in Java.

- Scala has a very compact syntax, similar to Python and can incorporate and run any Java class. That is quite powerful.

- Since we suspect that most of you are Python oriented, this lecture will give you a taste of R. All other lectures will be in Python or Java

Things R does and What R does not do

- data handling and storage: numeric, textual

- matrix algebra

- hash tables and regular expressions

- high-level data analytic and statistical functions

- classes ("Object Oriented")

- graphics

- programming language: loops, branching, subroutines

- is not a database, but connects to DBMSs

- has no graphical user interfaces, however it connects to Java, TclTk and it has **R Studio**.

- language interpreters are not fast. However, R could be extended by compiled C/C++ code

- no spreadsheet view of data, but connects to MS Excel

- no professional / commercial support

Statistical Packages

- Packaging: a crucial infrastructure to efficiently produce, load and keep consistent software libraries from (many) different sources / authors

- Most R packages deal with statistics and data analysis

- Many statistical researchers publish their state of the art methods as R packages.

- Comprehensive R Archive Network (CRAN) is a place where you can fetch those packages for free. You can get truly powerful tools at CRAN.

Where to get R and R Studio

- Download R for Windows, Mac-OS or Linux from

  http://cran.r-project.org/

- If you like command line interface, you do not need more than that.

- If you prefer an IDE, download *R Studio* from

  http://www.rstudio.com/

- You will keep on fetching packages (libraries) from the CRAN http://cran.r-project.org/ site.

- Run R installation first, than install R Studio. That is all.

- When we use R, we will use R Studio, except in rare circumstances.

- Eclipse plugins for R: http://www.walware.de/goto/statET

R Studio

R Studio

- The bottom left region is called Console. Console displays commands and results. You can type commands directly into the console.

- The top left region is the text editor. Open it with File > New File > R Script. Editor has syntax highlighting, parentheses matching and other features. You can open scripts (collections of commands) into text editor. You can highlight and execute individual commands or groups of commands from the text editor. You should get into the habit of always typing your commands in the editor.

- Top right region has command history and existing variables

- Bottom right region displays Help pages. Hit Help tab to get to the main help page.

- "An Introduction to R" and other manuals are decent reads.

R as a Calculator

```
> log2(32)

[1] 5

> sqrt(2)

[1] 1.414214

> seq(0, 5, length=6)

[1] 0 1 2 3 4 5

> plot(sin(seq(0, 2*pi,
length=100)))
```

- To quit R, type
```
> q()
```
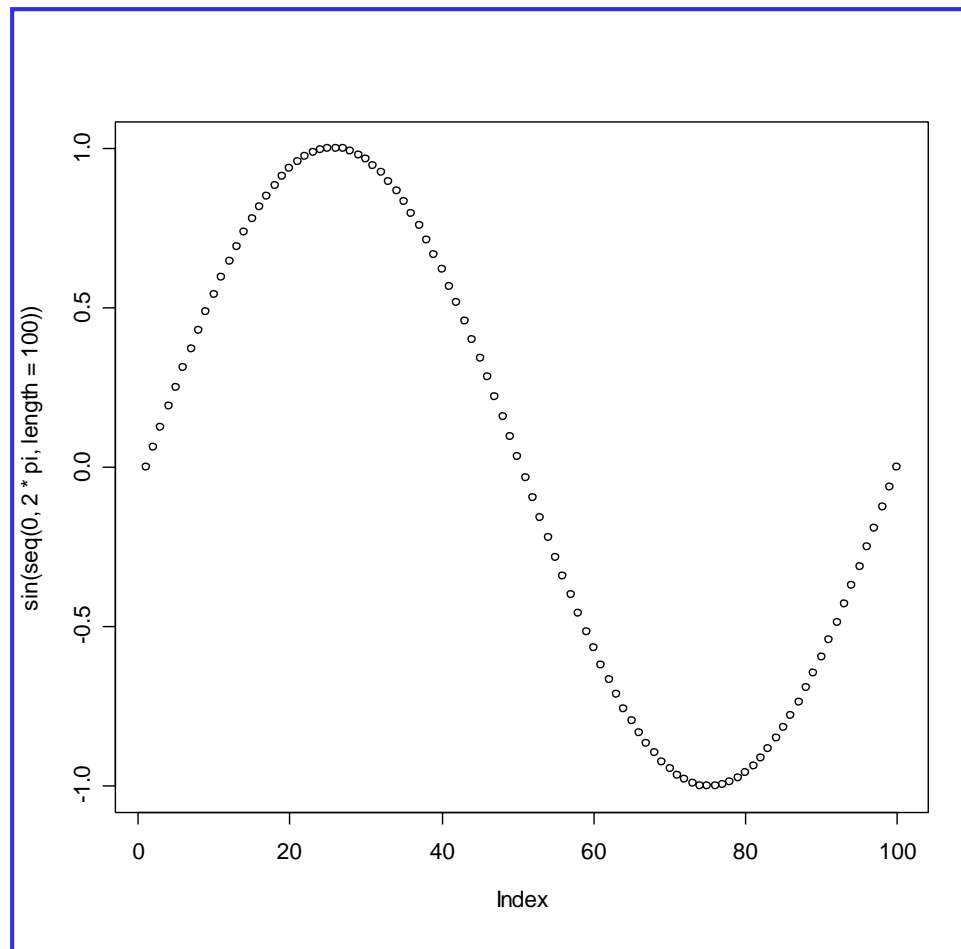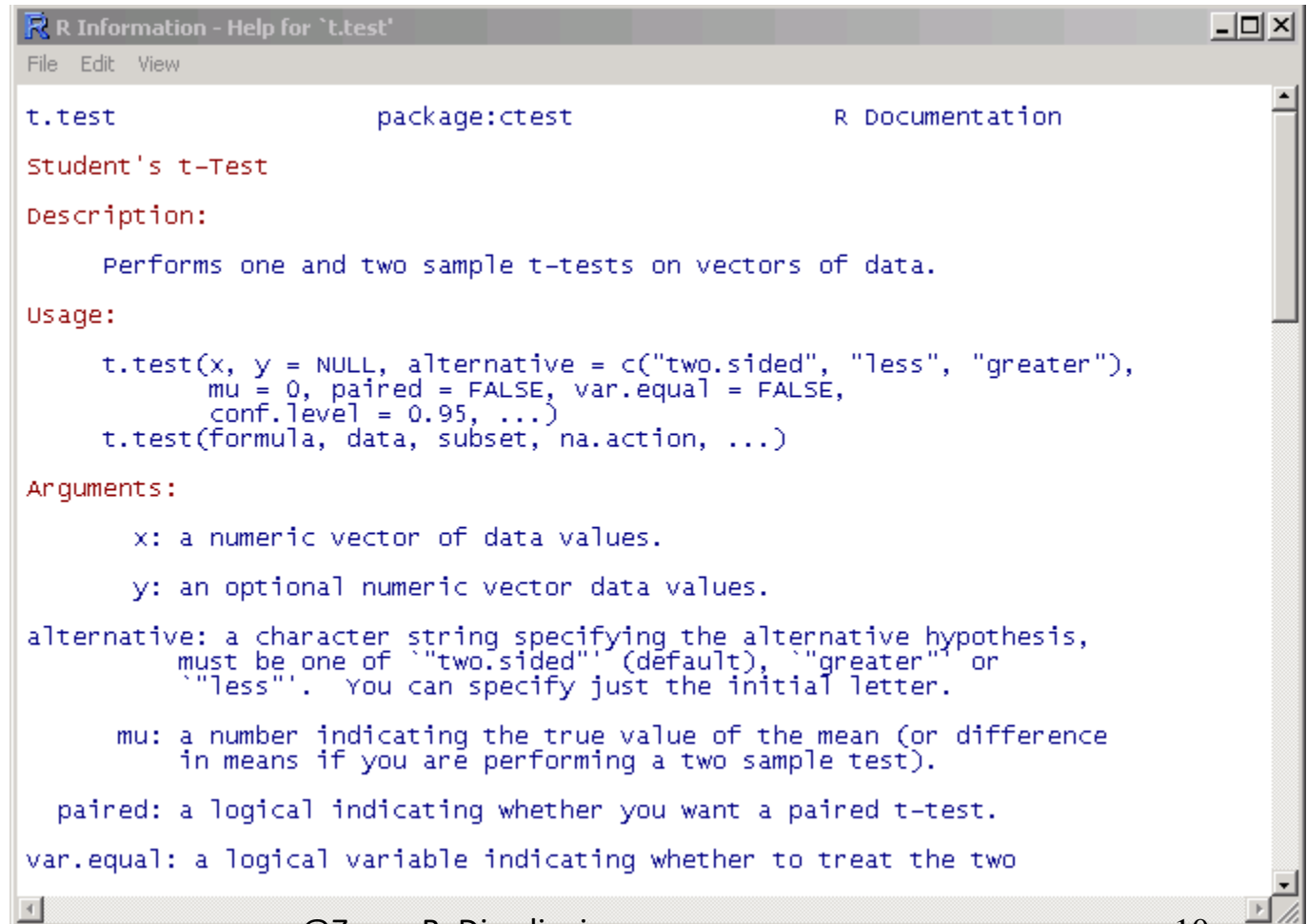
Getting Help

- Help provide details about a specific command whose name you know (its input arguments, options, algorithm, results, etc.):

```
>? t.test

or

>help(t.test)
```


R Information - Help for `t.test'

File   Edit   View

t.test   package:ctest   R Documentation

Student's t-Test

Description:

    Performs one and two sample t-tests on vectors of data.

Usage:

    t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
           mu = 0, paired = FALSE, var.equal = FALSE,
           conf.level = 0.95, ...)
    t.test(formula, data, subset, na.action, ...)

Arguments:

        x: a numeric vector of data values.

        y: an optional numeric vector data values.

alternative: a character string specifying the alternative hypothesis,
        must be one of `"two.sided"' (default), `"greater"' or
        `"less"'. You can specify just the initial letter.

       mu: a number indicating the true value of the mean (or difference
           in means if you are performing a two sample test).

   paired: a logical indicating whether you want a paired t-test.

var.equal: a logical variable indicating whether to treat the two

R Workspace

- When you close the R Studio or the R console window, the system asks if you want to save the workspace image.
- If you do, all objects in your current R session are saved in a binary file `.Rdata` located in the working directory of R.
- During your R session you can also explicitly save the workspace image to the current working directory

  ```
  > save.image()
  ```
- To check what the current working directory is, type

  ```
  > getwd()
  ```
- To change working directory type:

  ```
  > setwd("C:\\Zoran\\Classes\\R\\")
  ```
- To save to a specific file and specific location, type

  ```
  > save.image("C:\\Zoran\\Classes\\Rdatas\\1.RData")
  ```
- If you saved a workspace, the next time you start R, you can load it.

  ```
  > load("C:\\Zoran\\Classes\\R\\2.RData")
  ```
- All previously saved objects are available again. This allows you to switch between jobs and projects without ever loosing your work.

Objects

- Objects in R have names.
  - Names start with a letter (A-Z or a-z),
  - can contain letters, digits (0-9), and/or periods "."
  - case-sensitive `mydata` is different from `MyData`
  - do not use underscore "_"
- Types of objects: `vector, factor, array, matrix, data.frame, list, functions, classes`
- Objects have attributes
  - mode: numeric, character, complex, logical
  - length: number of elements in the object

Assignment and Variable Types

- R uses operators $->$ or $<-$ to assign values.
- Sign = is a substitute for $<-$
- Atomic variables could be numeric, character or logical

```
> a <- 49 -> z
> sqrt(a)
[1] 7
```
numeric

```
> a = "The dog ate my homework"
> sub("dog","cat",a)
[1] "The cat ate my homework"
```
character string

```
> a = (1+1==3)
> a
[1] FALSE
```
logical

Variables could be Missing Values

- Variables of each data type (numeric, character, logical) can also take the value NA: not available.
  - NA is not the same as 0
  - NA is not the same as ""
  - NA is not the same as FALSE

- Operations (calculations, comparisons) that involve NA may or may not produce NA:

```
> NA==1
[1] NA
> 1+NA
[1] NA
> max(c(NA, 4, 7))
[1] NA
> max(c(NA, 4, 7), na.rm=T)
[1] 7
```

```
> NA | TRUE
[1] TRUE
> NA & TRUE
[1] NA

na.rm is a logical
indicator, means remove
missing values
```

Predefined Functions

- R comes with some 30 packages. To see which ones are there, at the command prompt, type

  ```
  > library()
  ```

- R comes with a set of useful demos. To see which ones are there, at the command prompt, type

  ```
  > demo()
  ```

- R also comes with a number of datasets which are used in some demos and come quite handy for testing your own procedures. To see a list of provided datasets, type

  ```
  > data()
  ```

- To see all variables in the workspace, type

  ```
  > ls()
  ```

- To remove some of those variables, provide a comma separated list of variables to function `rm()`, e.g.

  ```
  > rm(d, fpdat, z)
  ```

Note on Loading Packages

- If you need a package, you usually type:

➢ `install.packages("package_name")      # fortunes`

- Then load the package

`> library(package_name)`

- And you are done. R goes to `cran.r-project.org`, finds the package, installs it for you and you are done.

- That did not work for me for the package `e1071`.

- I still had to go to `cran.r-project.org`, go to `Packages` and look for the packages I needed, e.g. `e1071`.

- You will have the option to download a ZIP or a tar file.

- Expand that archive and drop resulting directory in the subdirectory library of the installation directory of your R. In my case that directory is

- `C:\Program Files\R\R-3.4.1\library`

- To remove a package from memory, type:

➢ `detach(package:fortunes)   and then`

➢ `Remove.packages("fortunes")`

Functions and Operators

- **Functions** do things with data. Functions have
  - "Inputs": i.e. function arguments (0,1,2,…) and an
  - "Output": function result (exactly one)

We could **define functions, for example** `add()` :

```
> add = function(a,b)
 { result = a+b           // "+" in the left column,
      return(result) }   // indicates continuation of
                         // a command
> add(2,3)
> [1] 5
```

- **Operators** are short-cut writings for frequently used functions of one or two arguments.

Examples: `+  -  *  /  !  &  |  %%`

Vectors, Matrices and Arrays

- Vector: an ordered collection of data of the same type

```
> a = c(1,2,3)
> a*2           //  operation on a vector
[1] 2 4 6
```

- Function `c()`, constructs vectors. You may remember it as `concatenate()`, if you do not find that confusing, or `combine()` if combine sounds less confusing

- In R, a single number is the special case of a vector with 1 element.

- Other vector types: character strings, logical vectors

Vectors

```
> Mydata <- c(2,3.5,-0.2)   #Vector constructed with c() function
> Colors <-
    c("Red","Green","Red")   #Character vector

> x1 <- 25:30                #Vector constructed with Range operator :
> x1
[1] 25 26 27 28 29 30        #  Number sequences

> Colors[2]
[1] "Green"                   #Select One element with []
                             # Note, R counts from 1 not 0
> x1[3:5]
[1] 27 28 29                 # Selected several elements with range
                             # of index values
```

Operations on Vector Elements

```
> Mydata
[1] 2 3.5 -0.2
```
Print vector Mydata

```
> Mydata > 0
[1] TRUE TRUE FALSE
```
Logical test on the elements
produces a vector of logical values

Extract the positive elements using
logical vector as indexes

```
> Mydata[Mydata>0]
[1] 2 3.5
```

```
> Mydata[-c(1,3)]
[1] 3.5
```
Minus sign infront of an index or
indexes removes element(s)

Matrixes

- Matrix: Rectangular table of data of the same type

```
> m <- matrix(1:12, nrow=4, byrow = TRUE); m
      [,1] [,2] [,3]# byrow=T means that matrix
[1,]    1    2    3 # is filled by rows, otherwise
[2,]    4    5    6 # is filled by columns
[3,]    7    8    9
[4,]   10   11   12      ## vector y will be added to
> y <- -1:2             ## every column of m
> m.new <- m + y        ## m.new is a new matrix
> t(m.new)              ## transpose function t()
     [,1] [,2] [,3] [,4]
[1,]    0    4    8   12
[2,]    1    5    9   13
[3,]    2    6   10   14
> dim(m)
[1] 4 3
> dim(t(m.new))         ## dot in m.new is just part of the name
[1] 3 4
```

Fetching Data from Matrices

```
> x <- matrix(c(3, -1,2,0,-3, 6),nrow = 3,byrow = TRUE)

> x.mat[,2]              ##  2nd col
[1] -1 0 6

> x.mat[c(1,3),]        ##  1st and 3rd rows

      [,1] [,2]
[1,]     3   -1
[2,]    -3    6

> x.mat[-2,]            ##  Give us matrix, exclude 2nd row
      [,1] [,2]
[1,]     3   -1
[2,]    -3    6
```

Dealing with Matrices

```
> dim(x.mat)                    ## Dimension
[1] 3 2
> t(x.mat)                      ## Transpose

      [,1] [,2] [,3]
[1,]     3    2   -3
[2,]    -1    0    6


> x.mat %*% t(x.mat)            ## Matrix Multiplication


      [,1] [,2] [,3]
[1,]    10    6  -15
[2,]     6    4   -6
[3,]   -15   -6   45
```
•Quick quiz: `x <- c(1,2,3)` a vertical or horizontal vector
```
> solve(a)                      ## Inverse of a square matrix
> eigen()                       ## Eigenvectors and eigenvalues
```

Subsetting

- It is often necessary to extract a subset of a vector or a matrix

- R offers a couple of neat ways to do that

```
> x <- c("a", "b", "c", "d", "e", "f", "g", "h")
> x[1]
> x[3:5]
> x[-(3:5)]
> x[c(T, F, T, F, T, F, T, F)]
> x[x <= "d"]
> m[,2]
> m[3,]
```

- Test above expressions to see their effects

Lists

- **Vector:** an ordered collection of data of the same type.

```
> a = c(7,5,1)
> a[2]
[1] 5
```

- **List:** an ordered collection of data of arbitrary types. Lists have elements, each of which can contain any type of R object, i.e. the elements of a list do not have to be of the same type.
- List elements are accessed through different indexing operations.

```
> doe = list(name="john", age=28, married=F)
> doe$name
[1] "john "
> doe[3]
[1] FALSE
```

Lists

- A component of a list can be referred as `aa[[I]]` or `aa$times`. Here `aa` is the name of a list, `I` is the position of the component we are extracting and `times` is the name of a component of `aa`.

- The names of components may be abbreviated down to the minimum number of letters needed to identify them uniquely.

- `aa[[1]]` is the first component of aa, while `aa[1]` is the sublist consisting of the first component of `aa` only.

- There are functions whose return value is a List. We have seen some of them: `eigen, max, min, svd, …`

Lists are very flexible

- List can contain a numeric vector as one component and a character vector as the other. The following is a list with anonymous components

```
> my.list <- list(c(5,4,-1),c("X1","X2","X3"))
> my.list
[[1]]:
[1]   5   4 -1
[[2]]:
[1] "X1" "X2" "X3"
> my.list[[1]]
[1]   5   4 -1
```

- You can name components of your list, and access them by their names

```
> my.list <- list(c1=c(5,4,-1),c2=c("X1","X2","X3"))
> my.list$c2[2:3]
[1] "X2" "X3"
```

Rename Components, Convert to Vector

```
Empl <- list(employee="Anna", spouse="Fred", children=3,
    child.ages=c(4,7,9))
```

- You can change names of components of a list. Rather than `employee`, `spouse`, `childern` and `child.ages`, those names could, for example, be the first 4 letters (a,b,c,d).

- Rename component names by assigning those letters to the function `names(Empl)`, like

```
names(Empl) <- letters[1:4]
```
print new `Empl` to see the effect

- You can extend a list with new components

```
Empl <- c(Empl, service=8)
```

- You can concatenate lists

```
newList <- c(firstList, secondList)
```

- You can convert a list to a vector. Mixed types will be converted to character, giving a `character vector`

```
unlist(Empl).
```

Extracting a Slice of a List

- For example, the following variable `x` is a list containing copies of three vectors `n, s, b`, and a numeric value 3.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)
> x = list(n, s, b, 3) # x contains copies of n, s, b
```

- A slice is a copy of one or several components of a list.

- We retrieve a list slice with the single square bracket "`[]`" operator. The following is a slice containing the second member of `x`, which is a copy of `s`.

```
x[2]
[[1]]
[1] "aa" "bb" "cc" "dd" "ee"
```

- Please note that you cannot modify a slice

Referencing a Component of a List

- To reference a member of the list and modify it, we have to use the double square bracket `[[]]` operator. `x[[2]]` is the second member of `x`. `x[[2]]` is a copy of `s`, but is not a slice containing `s`

```
> x[[2]]
[1] "aa" "bb" "cc" "dd" "ee"
```

- We can modify the content of the referenced component directly.

```
> x[[2]][5] = "ff"
> x[[2]]
[1] "aa" "bb" "cc" "dd" "ff"
```

- If the referenced component is modified, the list `x` itself is modified

```
> x
[[1]]
 [1] 2 3 5
[[2]]
[1] "aa" "bb" "cc" "dd" "ff"
 [[3]] [1] TRUE FALSE TRUE FALSE FALSE
[[4]] [1] 3
```

Naming Rows and Columns of a Matrix

- Consider matrix **x.mat**:

```
> x.mat
     [,1] [,2]
[1,]    3   -1
[2,]    2    0
[3,]   -3    6
```

- You can name rows and columns of a matrix using a list with two components (names of rows and names o columns)

➢ **dimnames(x.mat) <-**
  list(c("R1","R2","R3"),c("C1","C2"))

```
> x.mat
   C1 C2
R1  3 -1
R2  2  0
R3 -3  6
```

Factors

- A factor is a vector object used to specify a set of discrete values (categories, enumerations) appearing as results of certain measurement, e.g.
  Gender {male, female}, Income {low, medium, high}, etc.
- For efficiency, factors are stored as numbers but have character labels for display. Efficiency is noticeable for large data sets.
- Consider a list of students in a class by gender:

 class <-c("male", "female", "female", "male", "male", "female")

- Apply function `factor()` and place the result in a new variable

```
> student.gender <- factor(class)
> student.gender
[1] male female female male male female
Levels: female male
> str(student.gender)    # str() displays structure of object
Factor w/ 2 levels "female","male": 2 1 1 2 2 1
> class(student.gender)    [1] "factor"
> mode(student.gender)     [1] "numeric"
> levels(student.gender)   [1] "female" "male"
> labels(student.gender)   [1] "1" "2" "3" "4" "5" "6"
```

Factors and `tapply()`

- Let us look at grades in the same class
  ```
  >class <- c("male","female","female","male","male","female")
  >grades <- c(4,3,4,3,3,4)
  ```
- To calculate sample mean grade for each gender, we can use function `tapply()`
  ```
  > grades.mean <- tapply(grades, student.gender, mean)
  > grades.mean
  > female male
  3.666667 3.333333
  ```
- Function `tapply()` is used to apply a function, here `mean()`, to each group of components of the first argument, here `grades`, defined by the levels of the second component, `student.gender`.
- `tapply()` also works when its second argument is not a factor, e.g., `tapply(grades, class, mean)`.
- This is true for many similar functions, since arguments are coerced to factors when necessary (using `as.factor()`).

`factor()`

- The function `factor()` is used to encode a vector as a factor (a set of 'categories' or 'enumerated types'). If argument `ordered` is TRUE, the factor levels are assumed to be ordered. **is.factor, is.ordered, as.factor** and **as.ordered** are the membership and coercion functions for these classes.

Usage

factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x))

x	a vector of data, usually taking a small number of distinct values
levels	an optional vector of the values that x might have taken. The default is the unique set of values taken by as.character(x), sorted into increasing order of x
labels	either an optional vector of labels for the levels (in the same order as levels after removing those in exclude), or a character string of length 1.
exclude	a vector of values to be excluded when forming the set of levels
ordered	logical flag to determine if the levels should be regarded as ordered

Data Frames

- Data Frame represents the typical data table with rows and columns, like a spreadsheet. Data frame is the central type in R.
- Data within each column (variable, component) have the same type (e.g. number, text, logical). Different columns may have different types.
- Both rows and columns have human readable names
- The components must be vectors (numeric, character, or logical), factors, numeric matrices, lists, or other data frames.
- Matrices, lists, and data frames provide as many variables to the new data frame as they have columns, elements, or variables, respectively.
- Numeric vectors, logicals and factors are included as is, and character vectors are coerced to be factors, whose levels are the unique values appearing in the vector.
- Vector structures appearing as variables of the data frame must all have the same length, and matrix structures must all have the same row size.

Making Data Frames

- Let us add vector **income** to the description of our student class

```
> income <-c(45000, 34500, 67000, 42000, 81000, 53000)
```

- We will construct a data frame `students` using components (variables): `student.gender`, `grades` and `income`

```
> students <- data.frame(gender=student.gender,
grade=grades, householdincome=income)
> students
```

	gender	grade	householdincome
1	male	4	45000
2	female	3	34500
3	female	4	67000
4	male	3	42000
5	male	3	81000
6	female	4	53000

Data Frames

- You have noticed that names of data frame components: `gender`, `grade` and `housholdincome` appear as names of columns or variables in the printout of the students data frame.

- We could display those "column" names using function `names()`

```
> names(students)
[1] "gender" "grade" "householdincome"
```

- We could fetch rownames using `rownames()` or `row.names()`

```
> row.names(students)
[1] "1" "2" "3" "4" "5" "6"
```

- We do not treat students as numbers. Let us change `rownames`

```
> row.names(students)<- c("John","Mary","Dianna","Bob","Mike","Joann")
```

- Now, when we ask for `rownames`, we get

```
>rownames(students)
[1] "John" "Mary" "Dianna" "Bob" "Mike" "Joann"
```

Function `labels()`

- Function `labels()` returns names of both columns and rows

```
> labels(students)
[[1]]
[1] "John" "Mary" "Dianna" "Bob" "Mike" "Joann"
[[2]]
[1] "gender" "grade" "householdincome"
```

Characterizing Data Frames

- Data frames respond to standard inquiry functions:

```
> class(students)
[1] "data.frame"
> is.data.frame(students)
[1] TRUE
> mode(students)
[1] "list"
> str(students)
'data.frame': 6 obs. of  3 variables:
 $ gender        :Factor w/ 2 levels "female","male":2 1 1 2 2 1
 $ grade         :num  4 3 4 3 3 4
 $ housholdincome:num  45000 34500 67000 42000 81000 53000
```

- Notice that function `str()` tells us the number of rows (observations) in the data frame and the number of variables (columns). Function `nrow()` does the same.

Importing a Data Frame

- Function `data()`, run without an argument, lists all data sets provided with the standard distribution of R. Those are all data frames.

- Let us load `mtcars` data set (Motor Trend Cars Road Test) by passing the data set name to `data()` function:

```
> data(mtcars)
```

- You can inspect the top of the data set with Unix like function `head()` and the bottom with function `tail()`

```
> tail(mtcars)
                mpg cyl  disp  hp drat    wt qsec vs am gear carb
Porsche 914-2   26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
Lotus Europa    30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Ford Pantera L  15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
Ferrari Dino    19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
Maserati Bora   15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
Volvo 142E      21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

Subsetting

- You can select a single column of a data frame using $ notation:

```
> head(mtcars$mpg)
[1] 21.0 21.0 22.8 21.4 18.7 18.1
```

- You can extract subsets of data frame data using bracket [] notation

```
> mtcars[1:5,3]                          # rows 1 to 5, column 3
[1] 160 160 108 258 360
> mtcars[1:5,"hp"]                       # rows 1 to 5 column "hp"
[1] 110 110  93 110 175
> mtcars[mtcars$mpg < 15,c("mpg","gear")] # rows where mpg <15
                        mpg gear          # coulmns MPG and GEAR
Duster 360             14.3     3
Cadillac Fleetwood     10.4     3
Lincoln Continental 10.4     3
> mtcars[c(1,2),]                        # rows 1 and 2, all columns
             mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4        21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
> mtcars[1,2]        # Treating data.form as a matrix
[1] 6               # element in the 1st row, 2nd column
> mtcars[3,1]
[1] 22.8            # element in the 3rd row, 1st column
```

Convert a Matrix into Data Frames

- Let us consider matrix `x.mat` with named rows and columns. We could verify the type (class) of the object

```
> x.mat
     C1 C2
R1  3 -1
R2  2 0
R3 -3 6
> class(x.mat)
[1] "matrix"
```

- The above matrix could be transformed into a data frame

```
> y <- data.frame(x.mat)
> class(y)
[1] "data.frame"
```

Program Branching

R is programming language and it has standard tools for branching and decision making:

```
if (logical expression) {
  statements
} else {
  alternative statements
}
```

else branch is optional

Loops

- When the same or similar tasks need to be performed multiple times; for all elements of a list; for all columns of an array, etc. R uses for loops

```
for(i in 1:10) {
    print(i*i)
}

i=1
while(i<=10) {
    print(i*i);
    i=i+sqrt(i)
}
```

User Defined Functions

- We have already seen that users could define functions. The general form of function definition is

```
name <- function(arguments) {
    expression
  }
```

- For example, function **larger** is defined as

```
larger <- function(x, y=9) {
  if(any(x < 0)) return(NA)
  y.is.bigger <- y > x
  x[y.is.bigger] <- y[y.is.bigger]
  x
}
```

- Note, $y=9$ provided the default value for paramter y

lapply, sapply, apply

- When the same or similar tasks need to be performed multiple times for all elements of a list or for all columns of an array. Easier and faster than "for" loops

  lapply(li, function)
- To each element of the list li, apply function function. The result is a list whose elements are the individual function results.

```
> li = list("klaus","martin","georg")
> lapply(li, toupper)
  [[1]]
  [1] "KLAUS"
  [[2]]
  [1] "MARTIN"
  [[3]]
  [1] "GEORG"
```

lapply, sapply, apply

sapply(li, function)

- Like apply, but tries to simplify the result, by converting it into a vector or array of appropriate size

```
> li = list("klaus","martin","georg")
> sapply(li, toupper)
[1] "KLAUS"   "MARTIN" "GEORG"

> fct = function(x) { return(c(x, x*x, x*x*x)) }
> sapply(1:5, fct)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    1    4    9   16   25
[3,]    1    8   27   64  125
```

apply

`apply( arr, margin, fct )`

- Applies the function **`fct`** along some dimensions of the array **`arr,`** according to **`margin,`** and returns a vector or array of the appropriate size; margine=1 indicates rows; margine=2, columns

```
> x
      [,1]  [,2]  [,3]
[1,]     5     7     0
[2,]     7     9     8
[3,]     4     6     7
[4,]     6     3     5

> apply(x, 1, sum)
[1] 12 24 17 14

> apply(x, 2, sum)
[1] 22 25 20
```

Hash Tables

- In vectors, lists, data frames and arrays, elements are stored one after another, and are accessed in that order by their offset (or: index), which is an integer number.

- Sometimes, consecutive integer numbers are not the "natural" way to access: e.g., gene names, oligo sequences

- E.g., if we want to look for a particular gene name in a long list or data frame with tens of thousands of genes, the linear search may be very slow.

- Solution: instead of list, use a hash table. It sorts, stores and accesses its elements in a way similar to a telephone book.

Hash Tables

- In R, a hash table is the same as a workspace for variables, which is the same as an environment.

```
> tab = new.env(hash=T)

> assign("cenp-e", list(cloneid=682777,
    description="putative kinetochore motor ..."), env=tab)

> assign("btk", list(cloneid=682638,
    fullname="Bruton agammaglobulinemia tyrosine kinase"), env=tab)

> ls(env=tab)
[1] "btk"      "cenp-e"

> get("btk", env=tab)
$cloneid
[1] 682638
$fullname
[1] "Bruton agammaglobulinemia tyrosine kinase"
```

Importing and exporting data

- There are many ways to get data into R and out of R.

- Most programs (e.g. Excel), as well as humans, know how to deal with rectangular tables in the form of tab-delimited text files.

```
> x = read.delim("filename.txt")
also: read.table, read.csv

> write.table(x, file="x.txt", sep="\t")
```
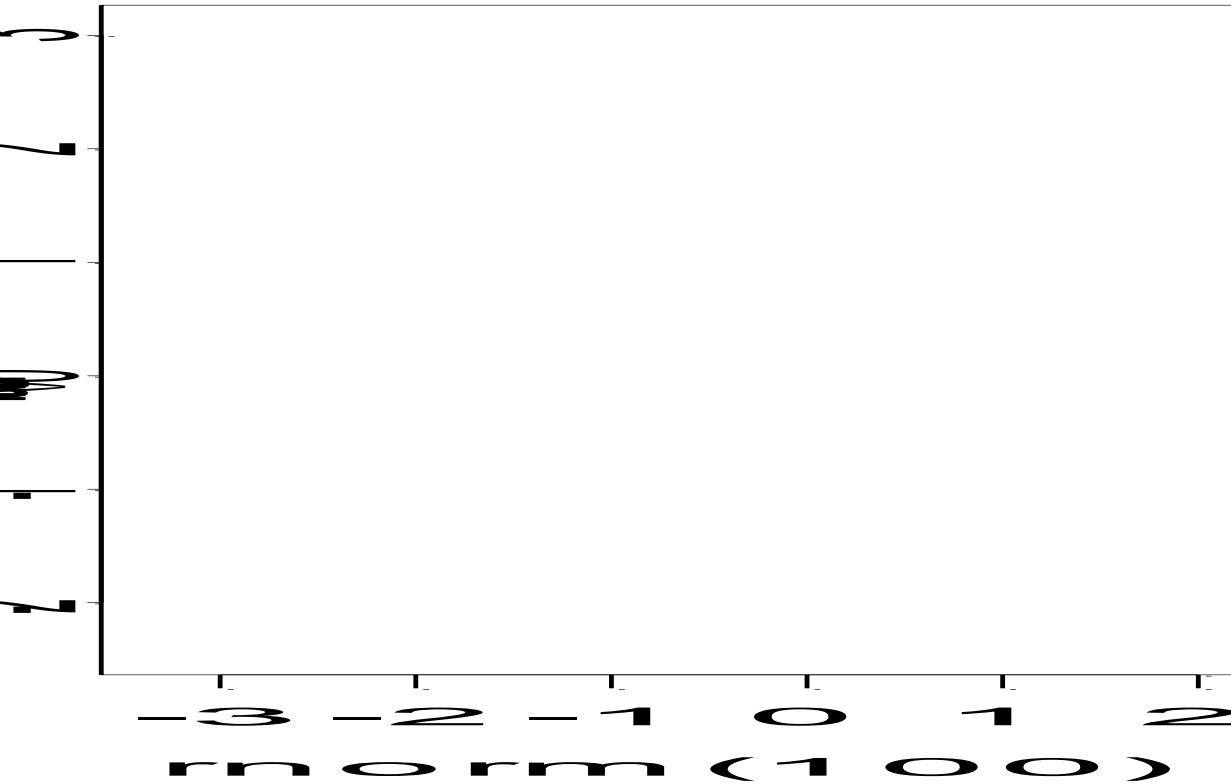
plot()

- If x and y are vectors, `plot(x,y)` produces a scatterplot of x against y.

- `plot(x)` produces a time series plot if x is a numeric vector or time series object.

- `plot(df), plot(~ expr), plot(y ~ expr),` where `df` is a data frame, `y` is any object, **expr** is a list of object names separated by `'+'` `(e.g. a + b + c).`

- The first two forms produce distributional plots of the variables in a data frame (first form) or of a number of named objects (second form). The third form plots y against every object named in `expr`.

Graphics with `plot()`

> `plot(rnorm(100),rnorm(100))`

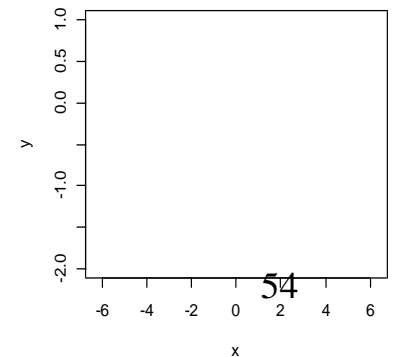Function `rnorm()`
Simulates a random
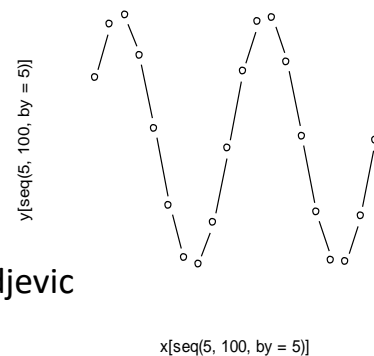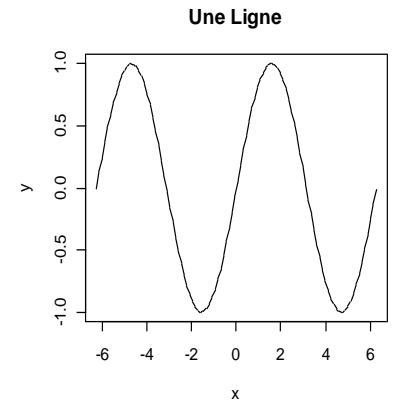normal distribution `.`

Help `?rnorm,`
`?runif,`
`?rexp,`
`?binom, …`

Graphics with `plot()`

```
x <- seq(-2*pi,2*pi,length=100)
y <- sin(x)
par(mfrow=c(2,2))
plot(x,y,xlab="x",ylab="Sin x")
plot(x,y,type= "l", main="A Line")

plot(x[seq(5,100,by=5)],
   y[seq(5,100,by=5)],
   type= "b",axes=F)

plot(x,y,type="n",
   ylim=c(-2,1)
par(mfrow=c(1,1))
```


@Zoran B. Djordjevic

54
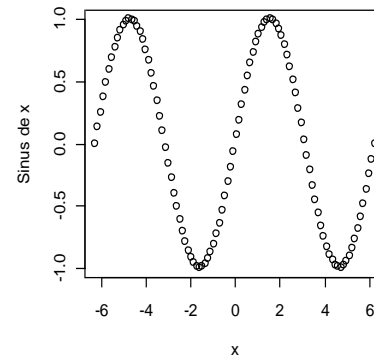
Graphical Parameters of **plot()**

type = "c": c =p (default), l, b,s,o,h,n.

pch="+" : character or numbers 1 – 18

lty=1 : numbers

lwd=2 : numbers

axes = "L": L= F, T

xlab ="string", ylab="string"

sub = "string", main ="string"

xlim = c(lo,hi), ylim= c(lo,hi)

And some more.

Graphical Parameters of **plot()**

```
x <- 1:10
 y <- 2*x + rnorm(10,0,1)
 plot(x,y,type="p") #Try l,b,s,o,h,n
     # axes=T, F
     #  xlab="age", ylab="weight"
     # sub="sub title", main="main title"
     # xlim=c(0,12), ylim=c(-1,12)
```

Other graphical functions

See also:

barplot()
image()
hist()
pairs()
persp()
piechart()
polygon()

library(modreg)
scatter.smooth()
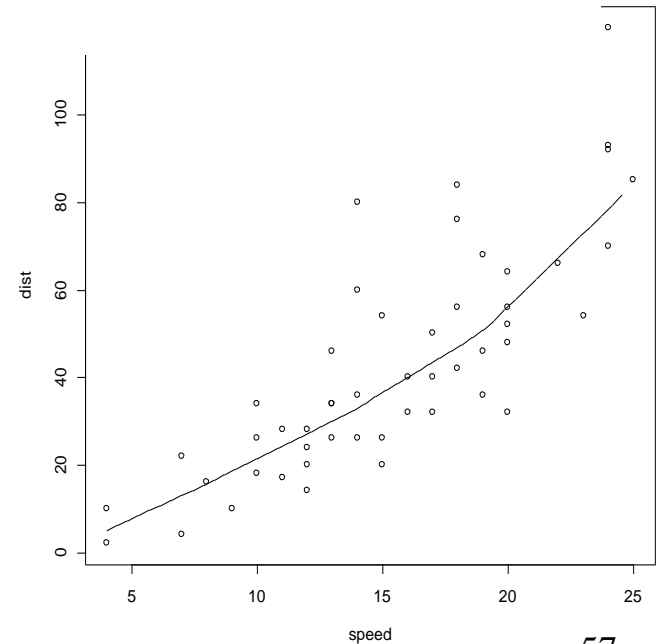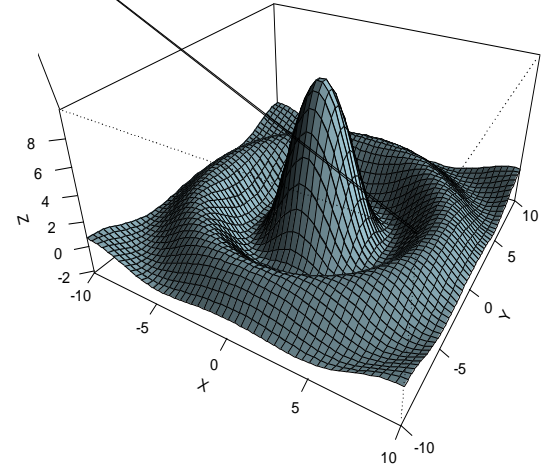
Plots for Multivariate Data

```
pairs(stack.x)
x <- 1:20/20
y <- 1:20/20
z <-
 outer(x,y,function(a,b){cos(10*a*b)/(1+
 a*b^2)})
contour(x,y,z)
persp(x,y,z)
image(x,y,z)
```

Histogram

- A *histogram* is a special kind of bar plot

- It allows you to visualize the *distribution* of values for a numerical variable

- When drawn with a *density scale:*

  - the *AREA* (NOT height) of each bar is the proportion of observations in the interval

  - the *TOTAL AREA* is 100% (or 1)

R: Histogram

- Type **?hist** to view the help file

  - Note some important arguments, esp **breaks**

- Simulate some data, make histograms varying the number of bars (also called 'bins' or 'cells'), e.g.

```
par(mfrow=c(2,2))# set up multiple plots
simdata <-rchisq(100,8)
hist(simdata)   # default number of bins
hist(simdata,breaks=2)   # etc,4,20
```

Histogram of simdata

@Zoran B. Djordjevic

61
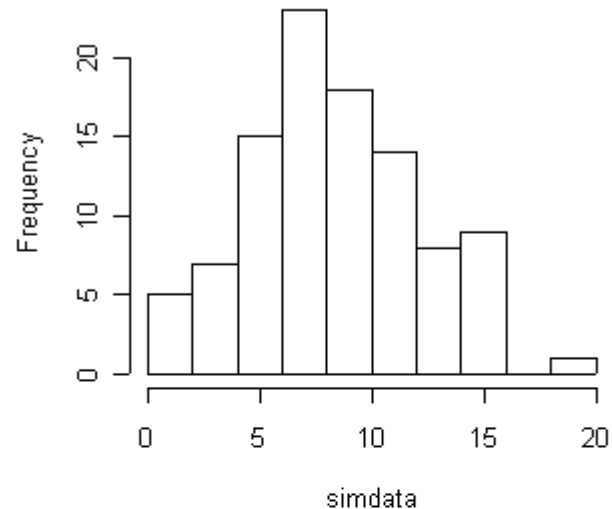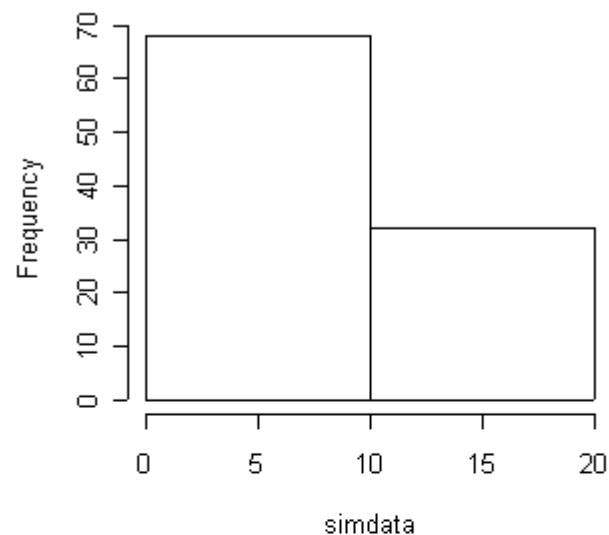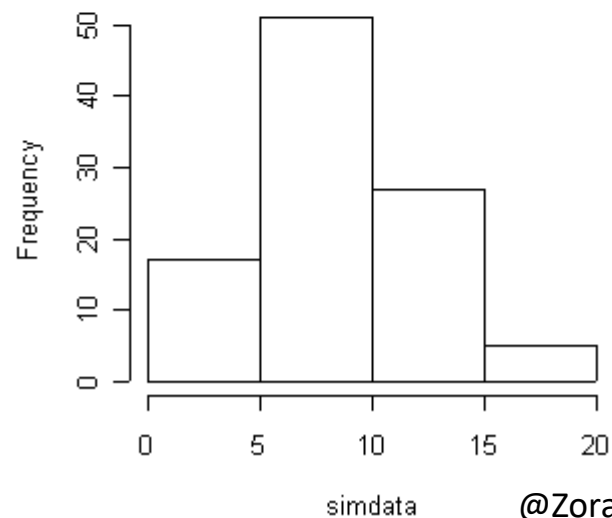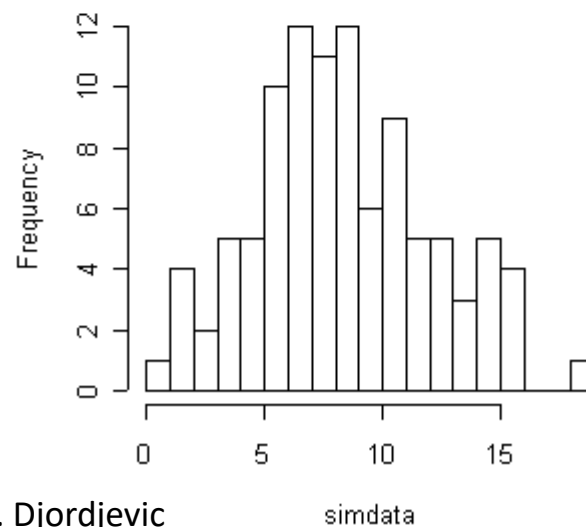
R: setting your own breakpoints

```
bps <- c(0,2,4,6,8,10,15,25)
hist(simdata,breaks=bps)
```


Histogram of simdata

62
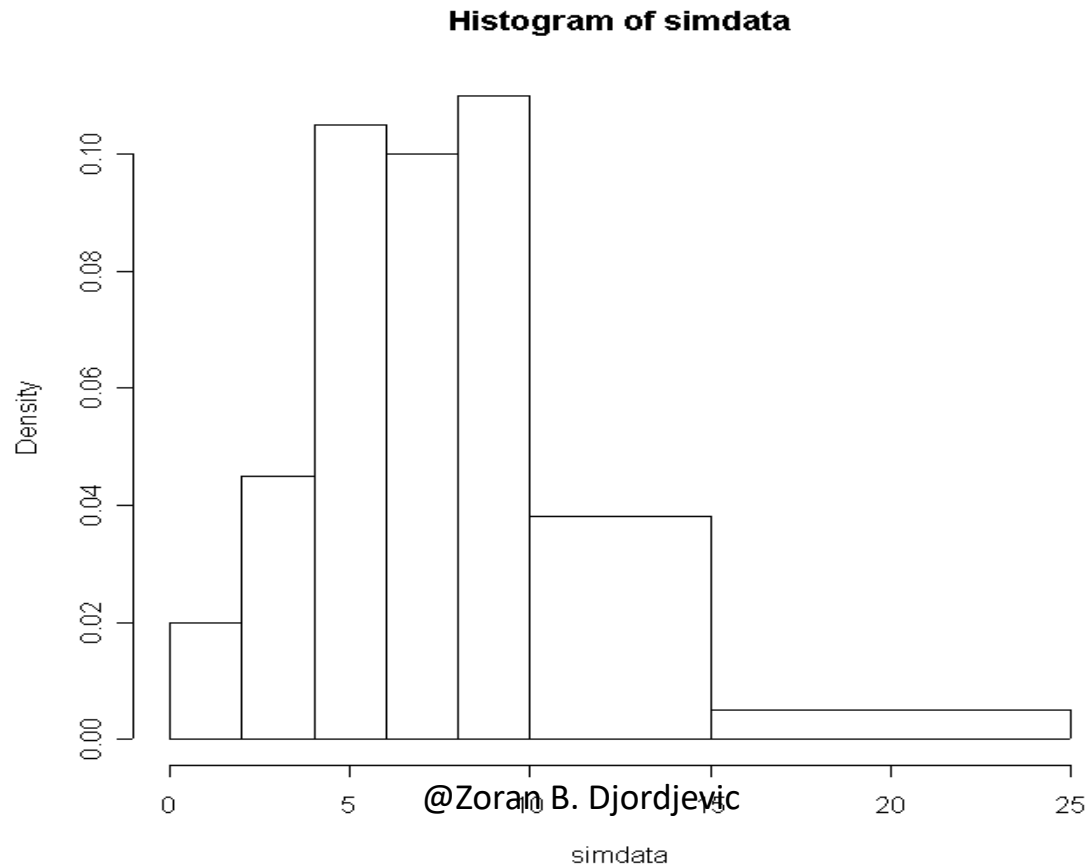
R: Scatterplot

- A scatterplot is a standard two-dimensional (X,Y) plot

- Used to examine the relationship between two (continuous) variables

- It is often useful to plot values for a single variable against the order or time the values were obtained

- Type **?plot** to view the help file

  - For now we will focus on simple plots, but R allows extensive user control for highly customized plots

- Simulate a bivariate data set:

```
z1 <- rnorm(50)
z2 <- rnorm(50)
rho <- .75            # (or any number between -1 and 1)
x2<- rho*z1+sqrt(1-rho^2)*z2
plot(z1,x2)
```

Scatterplot of X2 vs. Z1
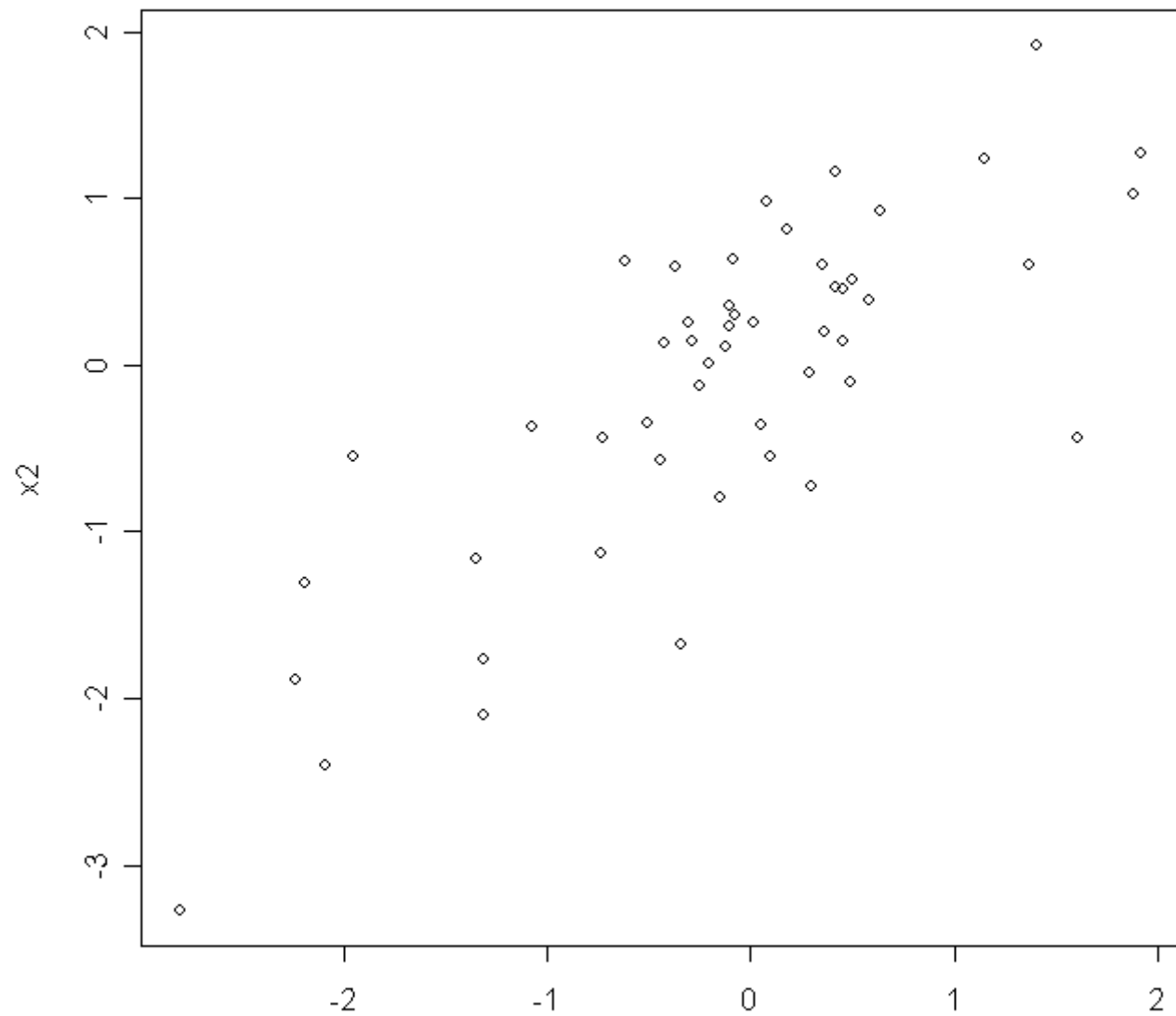
Elements of Statistics

Categorical (Qualitative) Data

- A data sample is called categorical, or qualitative, if its values belong to a collection of known, defined, non-overlapping classes. Common examples include student letter grades (A, B, C, D or F), and commercial bond ratings (AAA, AAB,...), human gender (M,F,N,TG)

- R built-in data frame named `painters` is a compilation of scores (grades) on a few classical painters. The data set belongs to the MASS package, and has to be pre-loaded into the R workspace prior to use.

```
> library(MASS)
> head(painters)
              Composition Drawing Colour Expression School
Da Udine               10       8     16          3      A
Da Vinci               15      16      4         14      A
Del Piombo              8      13     16          7      B
Del Sarto              12      16      9          8      A
Fr. Penni               0      15      8          0      D
Guilio Romano          15      16      4         14      A
```

Column `School` contains Categorical Data

- The last column, `School`, contains the information on school classification of the painters. The schools are named as A, B, ...,H, and the School variable is categorical.

```
> school = painters$School; school
 [1] A A A A A A A A A A B B B B B B C C C C C C D D D D D D D D D D E
E E
[36] E E E E F F F F G G G G G G H H H H
Levels: A B C D E F G H
> help(painters)
```

- This is a subjective assessment, on a 0 to 20 integer scale, of 54 classical painters. The painters were graded on four characteristics: `composition, drawing, colour` and `expression.`

- The school to which a painter belongs, is indicated by a factor level code: `"A"`: Renaissance; `"B"`: Mannerist; `"C"`: Seicento; `"D"`: Venetian; `"E"`: Lombard; `"F"`: Sixteenth Century; `"G"`: Seventeenth Century; `"H"`: French.

- `Composition, Drawing, Colour, and Expression` represent subjective measures of individual painters by an art critic, de Piles.

@Zoran B. Djordjevic

67

Frequency Distribution of Categorical Data

- In the data set `painters`, the frequency distribution of the School variable is a summary of the number of painters in each school.
- Frequency distribution is determined with R function `table()`

```
> school.freq = table(school)
> school.freq
school
A B C D E F G H
10 6 6 10 7 4 7 4
```

- To represent the results as a column, use function `cbind()`

```
> cbind(school.freq)
  school.freq
A          10
B           6
C           6
D          10
E           7
F           4
G           7
H           4
```

Relative Frequency Distribution of Categorical Data

- The relative frequency distribution is the proportion with which a particular category participates in the total population of all samples.
- The relationship between relative frequency and frequency is given by the ratio:

$$\textit{\textbf{Relative Frequency}} = \frac{\textit{\textbf{Frequency}}}{\textit{\textbf{Sample Size}}}$$

- We find the sample size of data set painters with R function `nrow()`. The relative frequency distribution is then determined :

```
> school.relfreq = school.freq / nrow(painters)
> school.relfreq
  school
          A          B          C          D          E          F
  0.636319 0.11111111 0.11111111 0.636319 0.12962963 0.07407407
          G          H
  0.12962963 0.07407407
```

- Please note, the sum over all relative frequencies is equal to 1

Making Relative Frequencies More Readable

- We can print with fewer digits by using function `options()`

```
> old = options(digits=1)
```

```
> old            # If you care to know what went into variable old
 $digits
[1] 7
```

- We apply function `cbind()` to print the result in column format.

```
> old = options(digits=1)
> cbind(school.relfreq)
  school.relfreq
A           0.19
B           0.11
C           0.11
D           0.19
E           0.13
F           0.07
G           0.13
H           0.07
> options(old)  # Restore old options
```

Bar Graph

- The bar graph of the School variable is a collection of vertical bars showing the number of painters in each school.
- We use function `barplot()` to produce the bar graph.

```
> barplot(school.freq)
```

To add color, we create a vector of colors and then add that vector to the `barplot()` as a color palette

```
> colors = c("red", "yellow", "green", "violet", "orange", "blue", "pink", "cyan")
> barplot(school.freq, col=colors)
```
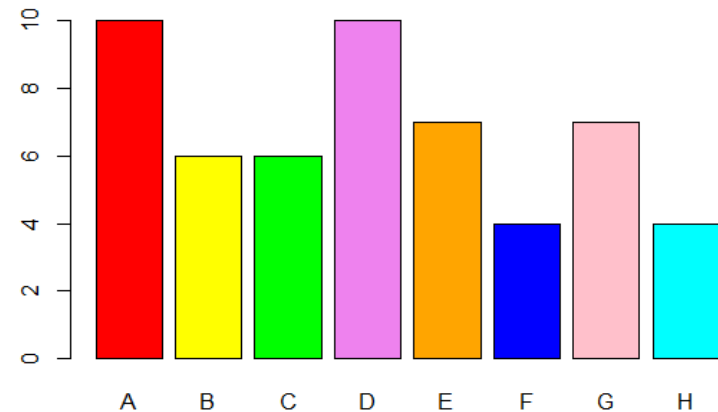
Category Statistics, `mean composition`

- Each school can be characterized by its various statistics, such as `means` of: `composition,` `drawing,` `coloring` **and** `expression.`
- Suppose we would like to know which school has the highest mean composition score.
- We would have to first find out the mean composition score of each school.
- **Let us** find the mean composition score of one school, e.g. school `C.` We do that in 3 steps:

1. Create a logical index vector for school C.

   ```
   c_school = school == "C"   # the logical index vector
   ```

2. Find the subset of painters for school C.

   ```
   c_painters = painters[c_school, ]   # child data set
   ```

3. Find the mean composition score of school C.

   ```
   mean(c_painters$Composition)     # mean composition
   [1] 13.16667                     # score of school C
   ```

mean composition score for all schools

- We could calculate mean composition score for all schools one by one or could use R function `tapply()`

```
> mean.scores = tapply(painters$Composition,painters$School,mean)
> mean.scores
     A        B        C        D        E        F        G        H
10.40000 12.16667 13.16667  9.10000 13.57143  7.25000 13.85714 14.0000
```

- Finally, we take an average of those values to get a mean over all schools.

```
> mean(mean.scores)
[1] 11.68899
```

- Function `tapply()` is used to apply a function, here `mean()`, to each group of components of the first argument, here `painters$Composition`, defined by the levels of the second component, here `painters$School`.

- Official Description of `tapply()` : "Applies a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors."

Quantitative Data

- Quantitative data, or continuous data, consists of numeric data that support arithmetic operations. This in contrast with categorical data, whose values belong to pre-defined classes with no arithmetic operation allowed.
- A built-in data frame `faithful` consists of a set of observations of the Old Faithful geyser in the USA Yellowstone National Park.

```
> head(faithful)
  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
4     2.283      62
5     4.533      85
```

- There are two observation variables in the data set. The first one, called `eruptions`, is the duration of the geyser eruptions. The second one, called `waiting`, is the length of waiting period until the next eruption. We want to find out whether there is a correlation between the two variables.

Frequency Distribution of Quantitative Data

- The frequency distribution of a quantitative variable can be presented as a summary of occurrences of data in a collection of non-overlapping categories.
- This means that we will break the range of values over which a variable of interest varies into a set of intervals (usually of equal duration) and then count how many times values in our sample fall in each of those intervals.
- In what follows we will find the frequency distribution of the **eruption durations** in `faithful` data set. We do it in several steps:

1. We first find the range of eruption durations.
2. Break the range into non-overlapping intervals.
3. Classify the eruption durations according to which interval they fall into.
4. "Compute the frequency of eruptions in each interval" or count the number of eruption durations in each interval.

@Zoran B. Djordjevic

Frequency Distribution of Quantitative Data

- We first find the range of eruption durations with the range function.
- Observed eruptions are between 1.6 and 5.1 minutes in duration .

```
duration = faithful$eruptions;
range(duration)
[1] 1.6 5.1
```

- Break the range into non-overlapping intervals by defining a sequence of equal distance break points.
- We come up with the interval `[1.5, 5.5]`.
- We set the break points to be the half-integer sequence `{ 1.5, 2.0, 2.5,..}`

```
> breaks = seq(1.5, 5.5, by=0.5);    # half-integer sequence
    breaks
[1] 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

- Next we classify the eruption durations according to the intervals.

Frequency Distribution of Quantitative Data

- We need to assign values from vector `duration` to the intervals delimited by sequence `breaks`. That is done by function `cut()`.

```
> duration.cut = cut(duration, breaks, right=FALSE)
```

- `cut()` accepts a vector that will be converted to a factor. In our case, `duration`.
- The second argument of `cut()` are labels for the factor levels of the resulting category. By default, labels are constructed as intervals of the form `"(a,b]"`. Values of `a-s` and b-s are taken from the supplied vector containing labels, here `breaks`.
- As the intervals are to be closed on the left, and open on the right, what is reverse from the default, we set `right= FALSE`.
- Frequency of eruptions in each interval is calculated with `table()`.

```
> duration.freq = table(duration.cut);
  duration.freq
duration.cut
[1.5,2) [2,2.5) [2.5,3) [3,3.5) [3.5,4) [4,4.5) [4.5,5) [5,5.5)
     51      41       5       7      30      73      61       4
```

Histogram

- We use function `cbind()` to print the result in the column format.

```
> cbind(duration.freq)
duration.freq
[1.5,2)  51
[2,2.5)  41
[2.5,3)  5
[3,3.5)  7
[3.5,4)  30
[4,4.5)  73
[4.5,5)  61
[5,5.5)  4
```

It appears that `hist()` function is an efficient mechanism for finding and displaying frequency distributions.

```
duration = faithful$eruptions

hist(duration, # apply the hist()
right=FALSE)    # intervals closed
                # on the left
```


Histogram of duration

@Zoran B. Djordjevic

78
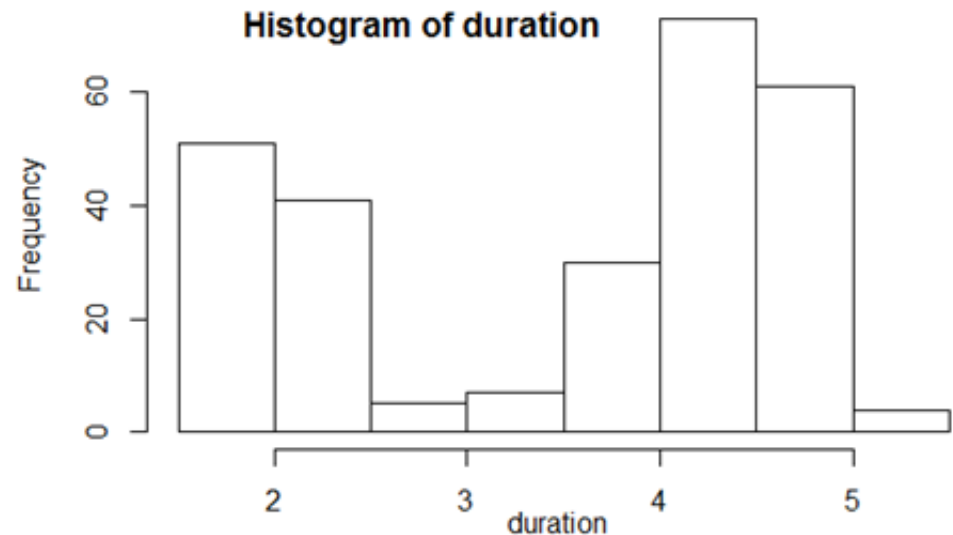
Relative Frequency Distribution Quantitative Data

- The relative frequency distribution of a data variable is the proportion of frequencies falling into a collection of non-overlapping categories (intervals)
- To find the relative frequency distribution of the eruption `durations`, we first find the frequency distribution of the eruption durations

```
> duration.freq = table(duration.cut)
```

- Next we divide the frequency distribution with the sample size established by `nrow()`. `nrow()` tells us how many measurements are in the whole `faithful` sample.

- The relative frequency distribution is then calculated as

```
> duration.relfreq = duration.freq / nrow(faithful);
  duration.relfreq
 duration.cut
 [1.5,2)   [2,2.5)   [2.5,3)   [3,3.5)   [3.5,4)   [4,4.5) [4.5,5) [5,5.5)
0.187500 0.150735 0.018382 0.025735 0.110294 0.268382 0.22426 0.01470
```

Relative Frequency Distribution Quantitative Data

- We can print with fewer digits and make **results** more readable by setting the digits option.

```
> old = options(digits=3)
```

- We then apply the `cbind()` function to print both the frequency distribution and relative frequency distribution in parallel columns.

```
> old = options(digits=1) ;
  cbind(duration.freq, duration.relfreq);
        duration.freq duration.relfreq
[1.5,2)            51             0.19
[2,2.5)            41             0.15
[2.5,3)             5             0.02
[3,3.5)             7             0.03
[3.5,4)            30             0.11
[4,4.5)            73             0.27
[4.5,5)            61             0.22
[5,5.5)             4             0.01
> options(old)    # restore the old option
```

@Zoran B. Djordjevic

Scatter Plot of Old Faithful Data

- A scatter plot pairs up values of two quantitative variables in a data set and display them as geometric points on a diagram.
- **W**e pair up the `eruptions` and `waiting` values in the same observation as `(x,y)` coordinates.
- We plot the points in the Cartesian plane.
- The eruption data value pairs with help of function `cbind()`

```
duration = faithful$eruptions;       # the eruption
  waiting = faithful$waiting;          # the waiting interval
  head(cbind(duration, waiting));
      duration waiting
 [1,]    3.600      79
 [2,]    1.800      54
 [3,]    3.333      74
 [4,]    2.283      62
 [5,]    4.533      85
 [6,]    2.883      55
```

Scatter Plot of Old Faithful Data

- We apply the `plot` function to compute the scatter plot of eruptions and waiting.

```
> plot(duration, waiting, xlab="Eruption duration",
  ylab="Time waited")
```
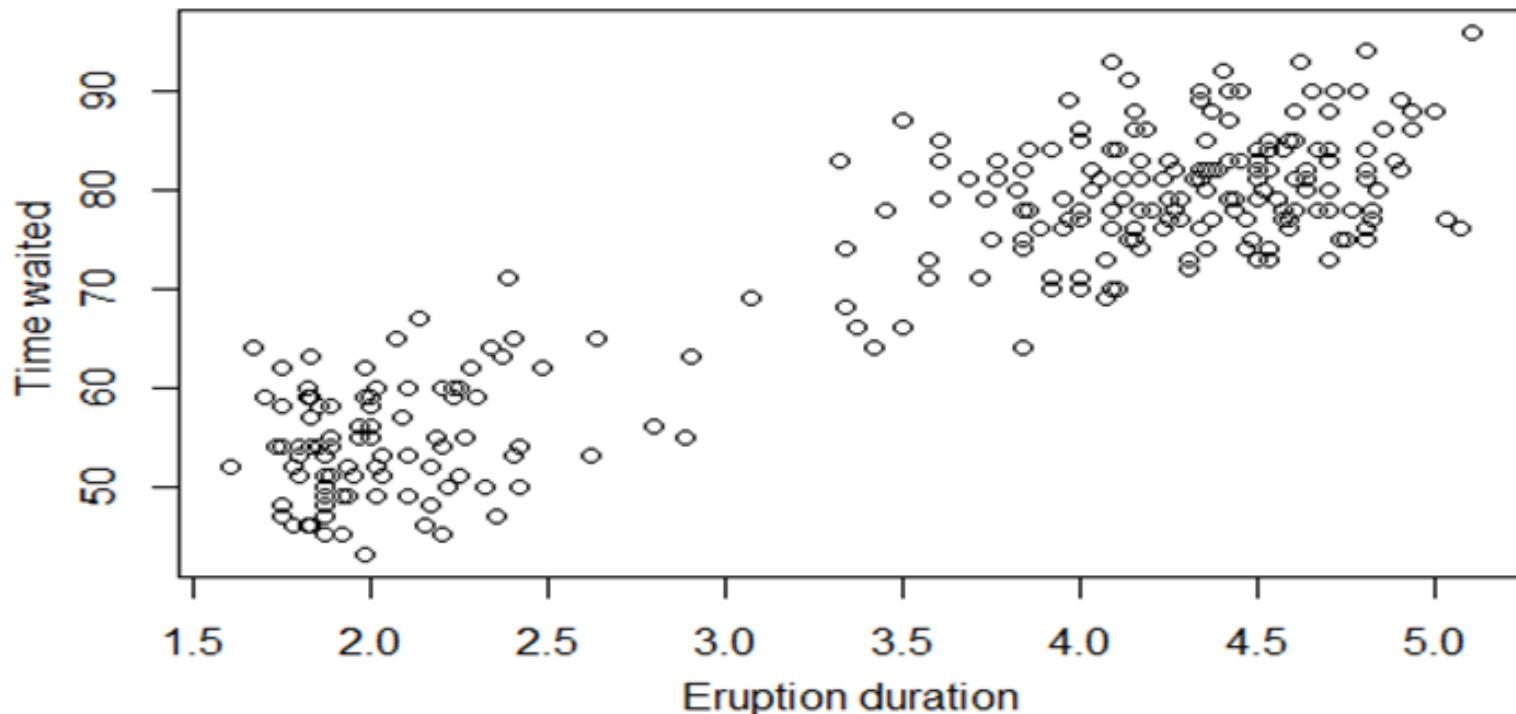

We do see "correlation" between variables. If you increase one, on average the other also increases

@Zoran B. Djordjevic

Scatter Plot of Old Faithful Data

- To establish the "best possible" linear relationship between two variable, we generate the so called linear model, or linear regression, using function `lm()`:

```
> model = lm(waiting ~ duration, data = faithful)
```

- and draw the trend line with function `abline()`:

```
> abline(model, col='red', lwd = 2)
```

- Parameter `lwd` **determines th** line width.

- We will discuss function `lm()` at length, later.

@Zoran B. Djordjevic
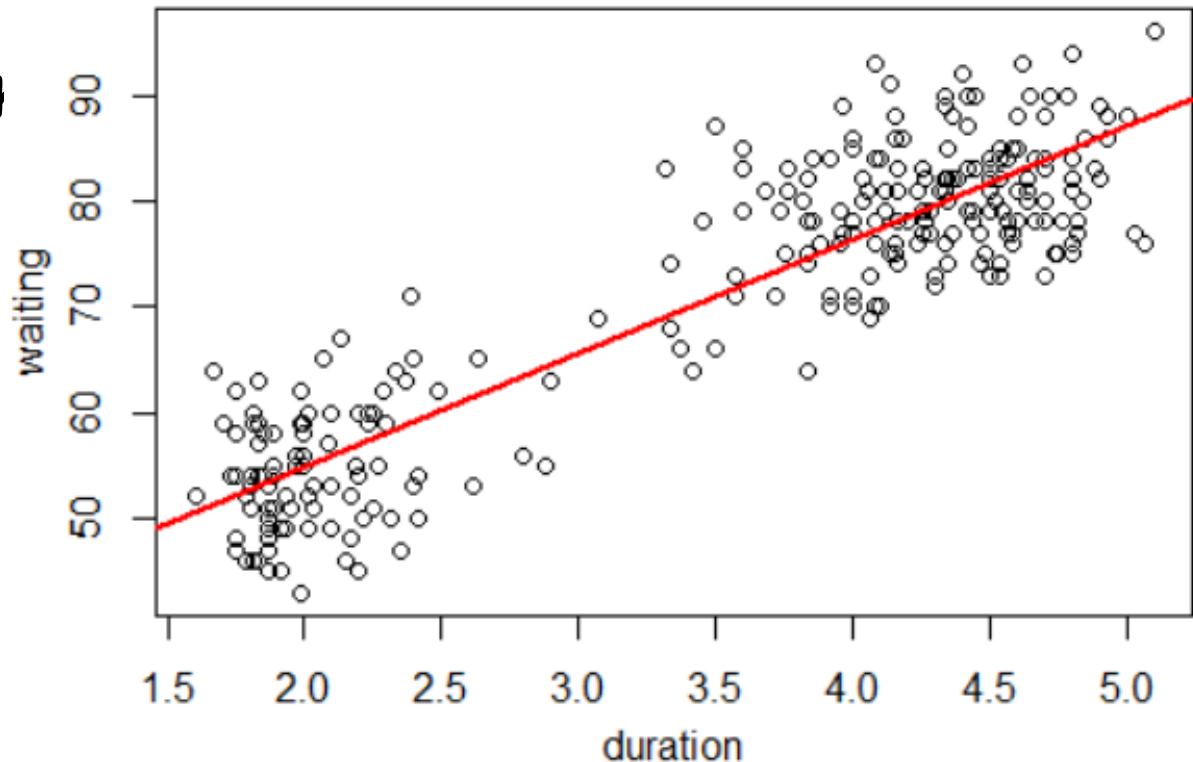
Cumulative Relative Freq. Distribution

- The cumulative relative frequency distribution of a quantitative variable is a summary of frequency proportions below a given level. Formally, cumulative relative frequency distribution is the integral of the relative frequency distribution from the beginning of the range to the observation point (interval, level).

$$Cumulative\ Rel.Freq.Distribution\ (\boldsymbol{l}) = \int_0^{\boldsymbol{l}} Rel.Freq.Distribution(\boldsymbol{i})\ d\boldsymbol{i}$$

- Find the frequency distribution of the eruption durations as follows:

```
> duration = faithful$eruptions ;
  breaks = seq(1.5, 5.5, by=0.5);
  duration.cut = cut(duration, breaks, right=FALSE);
  duration.freq = table(duration.cut)
```

- We then apply `cumsum()` function to compute the cumulative frequency distribution.

```
> duration.cumfreq = cumsum(duration.freq)
```

- The sample size of faithful is found with `nrow()`, **and** the cumulative relative frequency distribution is given by:

```
> duration.cumrelfreq = duration.cumfreq / nrow(faithful)
```

@Zoran B. Djordjevic

Cumulative Relative Freq. Distribution

- We then apply the `cbind()` function to print both the cumulative frequency distribution and relative cumulative frequency distribution in parallel columns.

```
> old = options(digits=2);
  cbind(duration.cumfreq, duration.cumrelfreq);
        duration.cumfreq duration.cumrelfreq
[1.5,2)               51               0.19
[2,2.5)               92               0.34
[2.5,3)               97               0.36
[3,3.5)              104               0.38
[3.5,4)              134               0.49
[4,4.5)              207               0.76
[4.5,5)              268               0.99
[5,5.5)              272               1.00
> options(old)
```

@Zoran B. Djordjevic

Cumulative Relative Frequency Distribution

- We could plot the cumulative relative frequency of durations of eruptions starting with zero element.

```
> cumrelfreq0 = c(0, duration.cumrelfreq);
  plot(breaks, cumrelfreq0,
main="Old Faithful Eruptions",      # main title
xlab="Duration minutes",
ylab="Cumulative eruption distribution");
  lines(breaks, cumrelfreq0);        # join the points
```

Old Faithful Eruptions
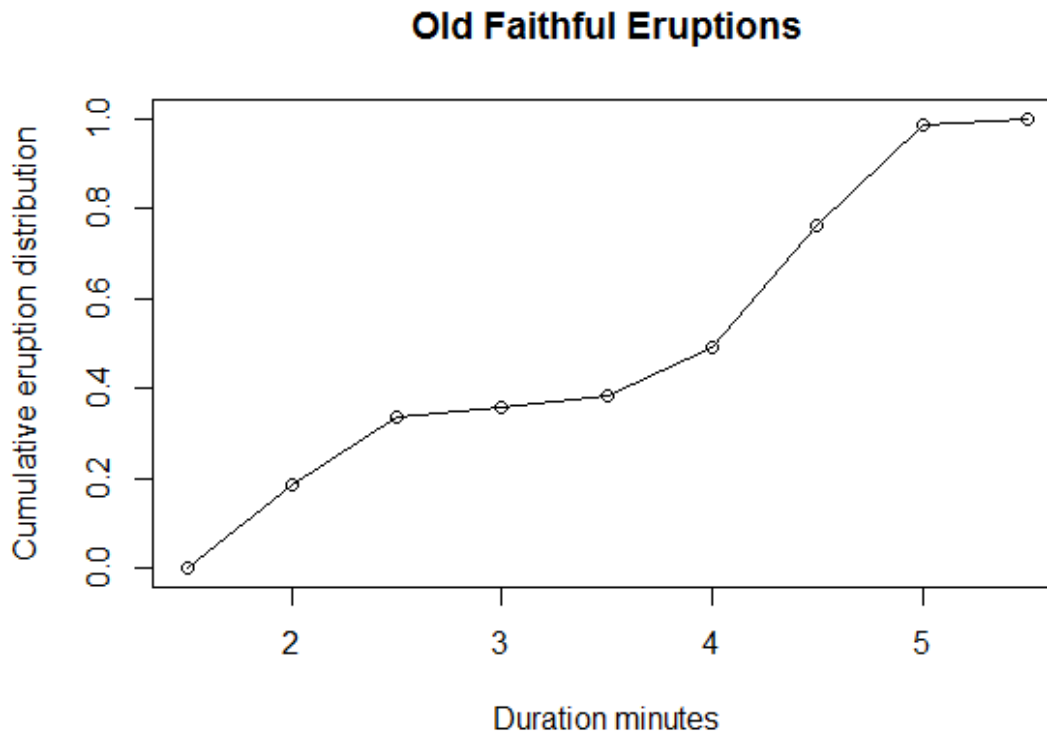
- Please note that cumulative distributions always range from 0 to 1

Probability Distributions

Probability Distribution

- The Histogram of the durations of Old Faithful Eruptions and the subsequent Cumulative Relative Frequency Distribution are telling us how particular events are distributed along a particular parameter axis or space.

- Such distributions are of great interest in probability and statistics and are usually studied under the term of **Probability Distributions**.

- For example, the collection of all possible outcomes of a sequence of coin tossing will turn out to be a distribution, known as the binomial distribution.

- The means of sufficiently large samples of a data population are known to resemble the normal distribution.

- The characteristics of these and other theoretical distributions are well understood. They can be used to make statistical inferences on data populations which they represent well.

Binomial Distribution

- The **binomial distribution** is a discrete probability distribution. It describes the outcome of n independent trials in an experiment. Each trial is assumed to have only two outcomes, either success or failure. If the probability of a successful trial is p, then the probability of having k successful outcomes in an experiment of n-independent trials is equal to .

$$f(k) = \binom{n}{k} p^k (1 - p)^{(n-k)}, \text{ where } k = 0, 1, 2, ..., n$$

Factor $\binom{n}{k} = \dfrac{n!}{k!(n-k)!}$ is referred to as the binomial coefficient

Practical Problem

- Suppose there are twelve multiple choice questions (trials) in an English class quiz. Each question has five possible answers, and only one of them is correct. Find the probability of having four or less correct answers if a student attempts to answer every question at random.

Binomial Problem, Solution

- If only one out of five possible answers is correct, the probability of answering a question correctly by random is *p=1/5=0.2*.

- Probability for answering incorrectly is *1-p = 4/5 = 0.8*.

- From *f(x)* we can find the probability of having exactly k=4 correct answers in 12 random attempts :

  $f(4,12) = \frac{12!}{4!8!} 0.2^4 0.8^8$. We could also use R function $choose(n,k)$, i.e. $choose(12,4)0.2^4 0.8^{12}$ or use R function `dbinom(k, size, prob)`

  `dbinom(4,size=12,prob=0.2);`   all giving the result
  `0.1328756`

 To find the probability of having four or less correct answers in 12 random attempts, we apply function dbinom() with *k* = 0*,...,*4.

  ```
  > dbinom(0, size=12, prob=0.2) +
  + dbinom(1, size=12, prob=0.2) +
  + dbinom(2, size=12, prob=0.2) +
  + dbinom(3, size=12, prob=0.2) +
  + dbinom(4, size=12, prob=0.2);
  [1] 0.9274
  ```

Mass Function vs. Cumulative Distribution

- The probability distribution is some times referred to as the probability mass function.

- On the right we see variation of the binomial distribution with k (number of successes) out of n = 20 and n = 40 trials.

- The bottom diagram presents cumulative binomial distribution, i.e. the probability that there were k or less successes in n = 20 and 40 trials.

- The cumulative binomial distribution is calculated as
```
> pbinom(4,12,0.2);
[1] 0.9274445
```


@Zoran B. [
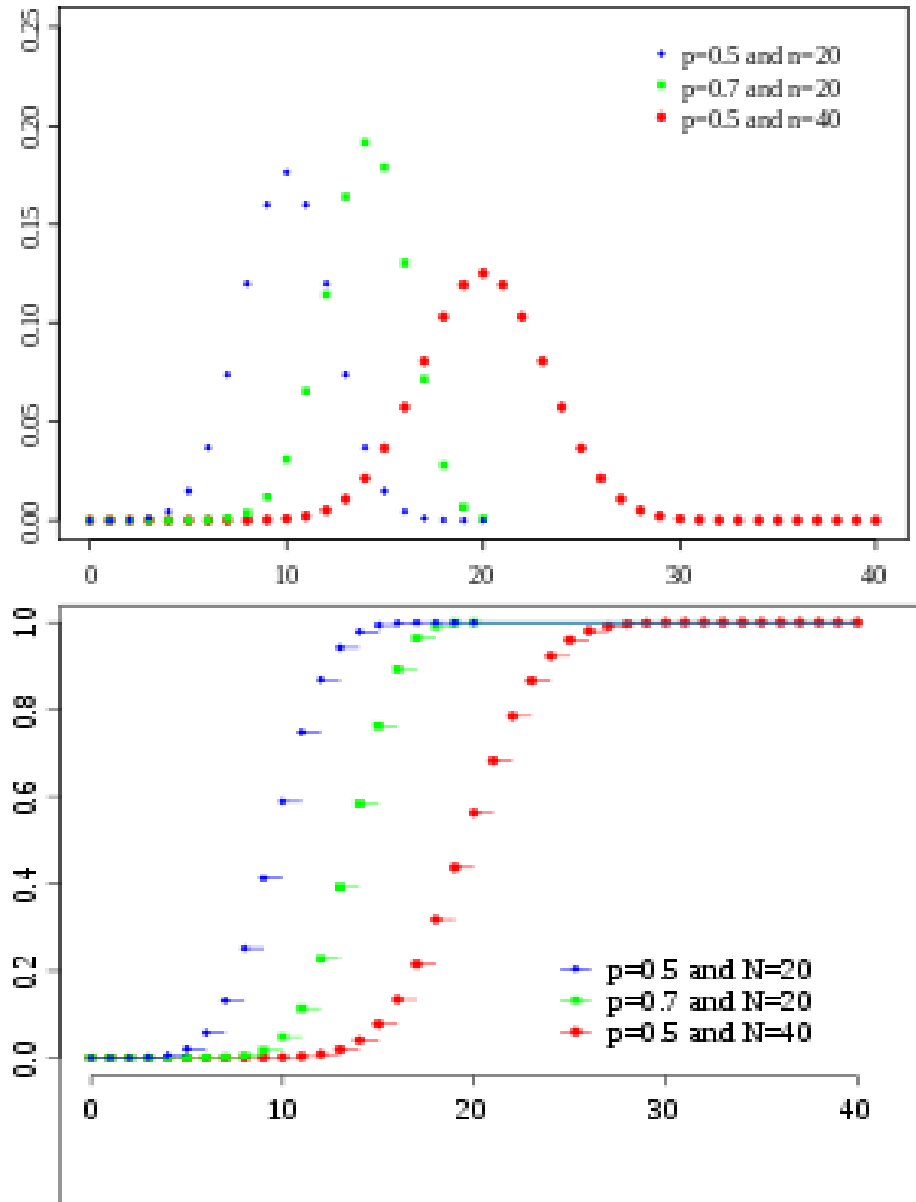
Continuous Uniform Distribution

- The **continuous uniform distribution** is the probability distribution of random number selection from the continuous interval between *a* and *b*. Its density function is defined by:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{when } a \leq x \leq b \\ 0 & \text{when } x < a \text{ or } x > b \end{cases}$$

- Below is a graph of continuous uniform distribution with *a*= 1,*b*= 3.
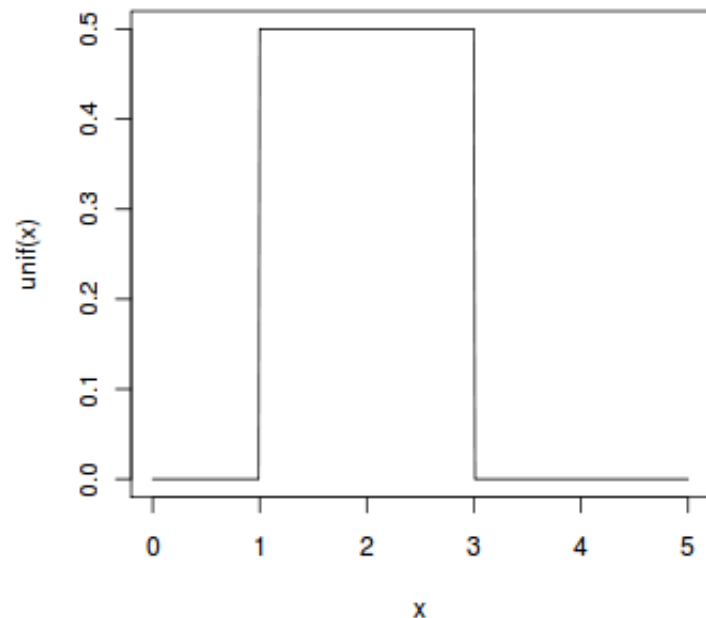
- A set of numbers uniformly distributed between 1 and 3 could be generated with a call to R function

```
> runif(10,min=1, max=3)
[1]  2.032381 1.792425 1.805124 1.733175
[5]  1.642199 1.830730 1.183520 1.251148
[9]  2.372529 2.625160
```

Statistical Measures

Statistical Measures

- If we know the probability (statistical) distribution of a process, i.e. a random variable, we could describe results of measurements involving that process (variable) most accurately.

- There are situations when we cannot rely on distribution functions:
  - We do not posses the full knowledge of the behavior of a random variable.
    - We posses no extensive data set illustrating the behavior nor
    - We have a simple (or complex) formula for the probability distribution
  - We need to transmit information about a process using a few numbers rather than an extended data set or a formula.

- There exists a set of standard descriptions or measures of statistical and probability distributions

Statistical Measures

- Mean
- Median
- Quartile
- Percentile
- Range
- Interquartile Range
- Box Plot
- Variance
- Standard Deviation
- Covariance
- Correlation Coefficient
- Central Moment
- Skewness
- Kurtosis

Mean

- The **mean** of an observation variable is a numerical measure of the central location of data values. It is the sum of its data values divided by data count. It corresponds to the center of gravity.
- Hence, for a data sample of size *n,* its **sample mean** is defined as:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Problem
- Find the mean eruption duration in the `faithful` data set.

Solution
- We apply the mean function to compute the values of `eruptions.`

➢ `duration = faithful$eruptions # the eruption durations`

➢ `mean(duration) # apply the mean function`
  `[1] 3.4878`

Median

- The **median** of an observation variable is the value at the middle when the data is sorted in ascending order. It is an ordinal measure of the central location of the data values.
- If you have 11 measurements and you order them from the lowest to the highest, the median is the 6th measurement in the ordered set

Problem

- Find the median of the eruption duration in the data set faithful.

Solution

- We apply the median function to compute the median value of eruptions.

```
> duration = faithful$eruptions; # eruption durations
  median(duration); # apply the median function
[1] 4
```

Quartile

- There are several **quartiles** of an observation variable. The **first quartile**, or **lower quartile**, is the value that cuts off the first 25% of the data when data is sorted in an ascending order. The **second quartile**, or **median**, is the value that cuts off the first 50%. The **third quartile**, or **upper quartile**, is the value that cuts off the first 75%.

Problem

- Find the quartiles of the eruption durations in the `faithul` data set.

Solution

- We apply the quantile function to compute the quartiles of eruptions.

```
> duration = faithful$eruptions; # the eruption durations
  quantile(duration) # apply the quantile function
0% 25% 50% 75% 100%
1.6000 2.1627 4.0000 4.4543 5.1000
```

Answer

- The first, second and third quartiles of the eruption duration are 2.1627, 4.0000 and 4.4543 minutes respectively.

Percentile

- The nth **percentile** of an observation variable is the value that cuts off the first *n*-percent of the data values when the data set is sorted in ascending order.

Problem

- Find the 32nd, 57th and 98th percentiles of the eruption durations in the data set faithful.

Solution

- We apply the `quantile` function to compute the percentiles of eruptions with the desired percentage ratios.

```
> duration = faithful$eruptions # eruption durations
> quantile(duration, c(.32, .57, .98));
32% 57% 98%
2.3952 4.1330 4.9330
```

Answer

- The 32nd, 57th and 98th percentiles of the eruption duration are 2.3952, 4.1330 and 4.9330 minutes respectively.

Range

- The **range** of an observation variable is the difference of its largest and smallest data values. It is a measure of how far apart the entire data spreads in value.

Problem

- Find the range of the eruption durations in the `faithful` data set.

Solution

- We apply the max and min function to compute the largest and smallest values of eruptions, then take the difference.

```
> duration = faithful$eruptions # eruption durations
> max(duration) − min(duration) # apply the max and min
                                # functions
[1] 3.5
```

Answer

- The range of the eruption duration is 3.5 minutes.

Interquartile Range

- The **interquartile range** of an observation variable is the difference of its upper and lower quartiles. It is a measure of how far apart the middle portion of data spreads in value.

Problem

- Find the interquartile range of eruption duration in the data set faithful.

Solution

- We apply the IQR function to compute the interquartile range of eruptions.

```
> duration = faithful$eruptions; # eruption durations
  IQR(duration)                  # apply the IQR function
[1] 2.2915
```

Answer

- The interquartile range of eruption duration is 2.2915 minutes.

Box Plot

- The **box plot** of an observation variable is a graphical representation based on its quartiles, as well as its smallest and largest values. It attempts to provide a visual shape of the data distribution.
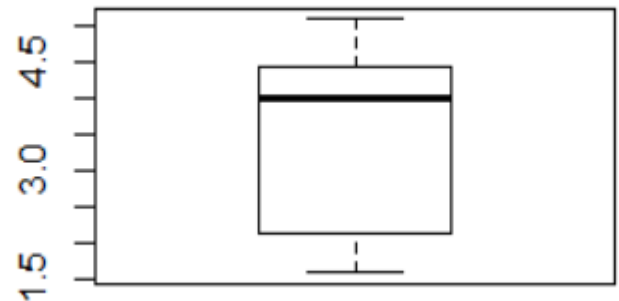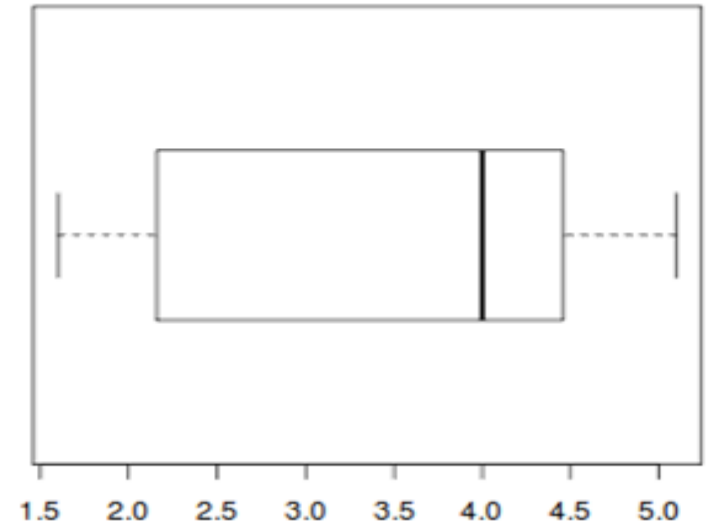
Problem

- Find the box plot of the eruption duration in the data set `faithful`.

Solution

- We apply the `boxplot()` function to produce the box plot of `eruptions`.

```
> duration = faithful$eruptions
# eruption durations
 boxplot(duration, horizontal=FALSE)  #
vertical box plot
```
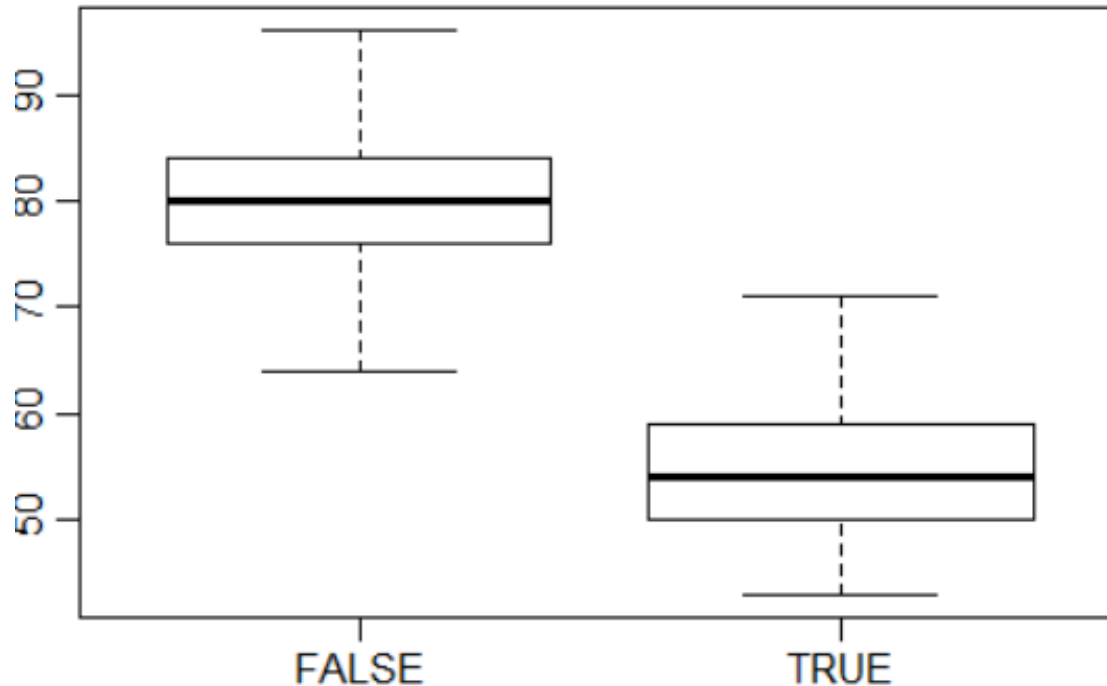
Old Faithful Measures

- Let us add a new column to the faithful dataset

```
> faithful$type <- duration < 3.1;
  type <- faithful$type
```

- Present waiting times measures for two different types:

```
> boxplot(waiting ~ type, data = faithful)
```

Variance

- The **variance** is a numerical measure of how the data values are dispersed around the mean. In particular, the **sample variance** is defined as:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

- Similarly, the **population variance** is defined in terms of the population mean μ and population size N as:

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{\mu})^2$$

Problem

- Find the variance of the eruption duration in `faithful` data set.

Solution

- We apply the `var()` function to compute the variance of eruptions.

```
> duration = faithful$eruptions;
  var(duration) # apply the var function
[1] 1.3027
```

Function $var()$

- Function $var(x)$, for variance, acts on sample vector x and calculates:

$$var(x) = sum((x - mean(x))\verb|^|2)/(length(x) - 1)$$

- or sample variance.

- If the argument to $var()$ is an n by p matrix the value is a p by p sample *Covariance Matrix* obtained by regarding the rows as independent p-variate (p-dimensional) sample vectors.

Standard Deviation

- The **standard deviation** of an observation variable is the square root of its variance.

Problem

- Find the standard deviation of the eruption duration in the `faithful` data set

Solution

- We apply the `sd()` function to compute the standard deviation of eruptions.

```
> duration = faithful$eruptions; # eruption durations
  sd(duration) # apply the sd function
[1] 1.1414
```

Covariance

- The **covariance** of two variables *x* and *y* in a data sample measures how or whether two variables are (linearly) related.
- A positive covariance indicates a positive linear relationship between the variables, and a negative covariance indicates the opposing relationship.
- The **sample covariance** is defined in terms of the sample means as:

$$s_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})$$

- Similarly, the **population covariance** is defined in terms of the population means μ_x, μ_y as:

$$\sigma_{xy} = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \mu_y)(x_i - \mu_x)$$

Problem

- Find the covariance of the eruption duration and waiting time in the data set faithful. Observe if there is any linear relationship between the two variables.

Solution

- We apply the `cov()` function to compute the covariance of eruptions and waiting.

```
> duration = faithful$eruptions;  # the eruption durations
  waiting = faithful$waiting;     # the waiting period
  cov(duration, waiting)          # apply the cov function
[1] 13.978
```

Eigen Vectors of Covariance Matrix

- Eigen vectors of covariance matrix or its normalized form provide important insight in the behavior of our data.
- The largest eigen vectors of that matrix point into directions of strongest variation of underlying variables.

@Zoran B. Djordjevic
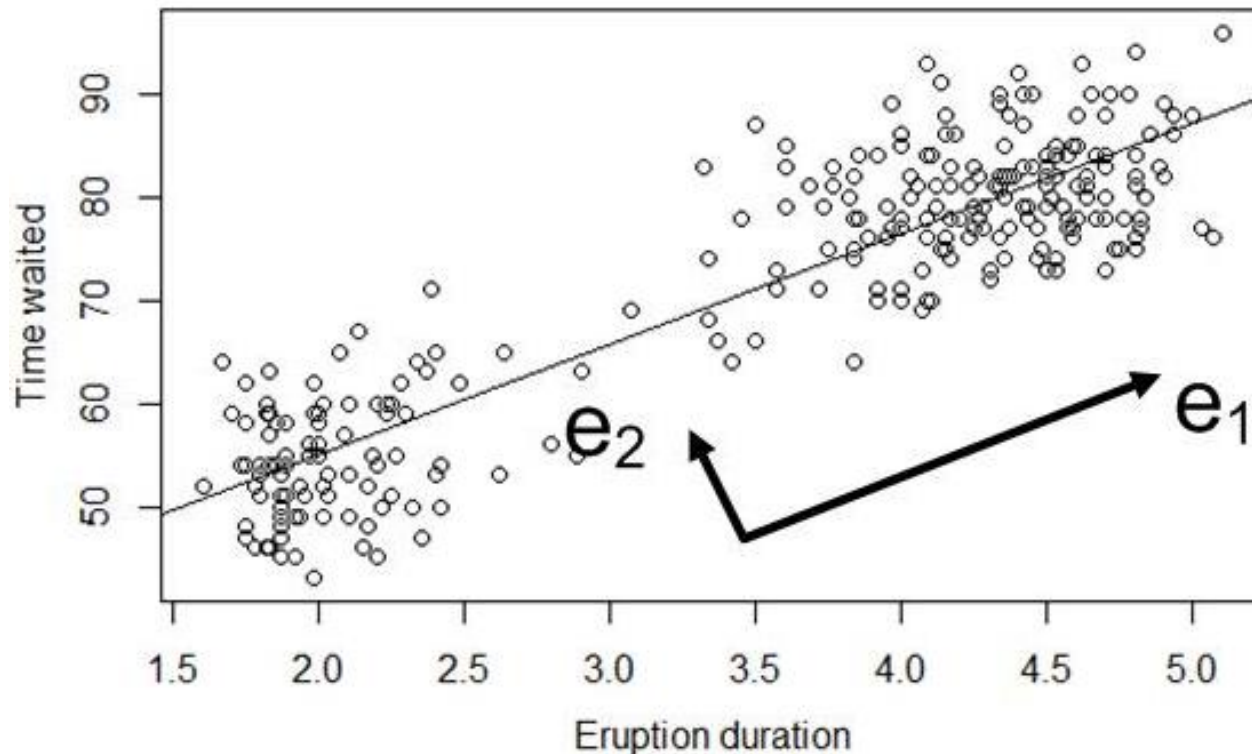
Correlation Coefficient

- The **correlation coefficient** of two variables in a data sample is their covariance divided by the product of their individual standard deviations . It is a normalized measurement of how the two are (linearly) related.

- Formally, the **sample correlation coefficient** is defined by the following formula, where s_x and s_y are the sample standard deviations, and s_{xy} is the sample covariance.

$$r_{xy} = \frac{s_{xy}}{s_x s_y}$$

- Similarly, the **population correlation coefficient** is defined as:

$$\rho = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

- Where σ_x and σ_y are the population standard deviations, and σ_{xy} is the population covariance.

- If the correlation coefficient is close to 1, it indicates that the variables are positively linearly related and the scatter plot falls almost along a straight line with a positive slope.

- Correlation coefficient of -1 indicates that the variables are negatively linearly related and the scatter plot almost falls along a straight line with negative slope.

- Correlation coefficient of 0 (zero) indicates a very weak linear relationship between the variables, or absence of a relationship between variables.

Correlation Coefficient

Problem

- Find the correlation coefficient of the eruption duration and waiting time in the `faithful` data set. Observe if there is any linear relationship between two variables.

Solution

- We apply the `cor()` function to compute the correlation coefficient of `eruptions` and `waiting`.

```
> duration = faithful$eruptions; # eruption durations
  waiting = faithful$waiting;    # waiting period
  cor(duration, waiting);        # apply the cor function
[1] 0.90081
```

- The correlation coefficient of the eruption duration and waiting time is 0.90081.

- The correlation coefficient is close to 1, and we can conclude that eruption duration and the waiting time are positively linearly correlated.

Central Moment

- The kth **central moment** (or moment about the mean) of a data population is:

$$\mu^2 = \frac{1}{N} \sum_{i=1}^{} (x_i - \mu)^k$$

- Similarly, the kth central moment of a data sample is:

$$m_k{}^2 = \frac{1}{N} \sum_{i=1}^{} (x_i - \bar{x})^k$$

- The second central moment of a sample population is its variance.

Problem

- Find the third central moment of eruption duration in the `faithful` data set

Solution

- We apply the function moment from the e1071 package.
- Package e1071 is not in the core R library, and has to be installed and loaded into the R workspace.

```
> library(e1071); # load e1071
  duration = faithful$eruptions; # eruption durations
  moment(duration, order=3, center=TRUE);
[1] -0.6149    #  The third central moment of eruption                      #
duration is -0.6149.
```

Skewness

- The **skewness** of a data population is defined by a specific ratio of μ_2 and μ_3 are the second and third central moments.

$$\gamma_1 = \mu_3 \Big/ \mu_2{}^{3/2}$$

- Intuitively, the skewness is a measure of symmetry.
- As a rule, negative skewness indicates that the mean of the data values is less than the median, and the data distribution is *left-skewed*. Positive skewness would indicates that the mean of the data values is larger than the median, and the data distribution is *right-skewed*.

Problem

- Find the skewness of eruption duration in the data set faithful.
- You will do it for your homework

Kurtosis

- The **kurtosis** of a univariate population is defined by the following formula, where μ_2 and μ_4 are the second and fourth central moments

$$\gamma_2 = {\mu_4}\Big/{\mu_2{}^2}$$

- Intuitively, the *kurtosis* is a measure of the "*peakedness*" of the data distribution. Negative kurtosis would indicates a *flat* data distribution, which is said to be **platykurtic**.

- Positive kurtosis would indicates a *peaked* distribution, which is said to be **leptokurtic**. Incidentally, the normal distribution has zero kurtosis, and is said to be **mesokurtic**.

Problem

- Find the kurtosis of eruption duration in the data set faithful

Solution

- You will do it for your homework

Normal and Other Distributions

Normal Distribution

- The **normal distribution** is defined by the following probability density function, where μ is the population mean and σ^2 is the variance.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\ e^{-(x-\mu)^2/2\sigma^2}$$

- If a random variable X follows the normal distribution, then we write:

$$X \sim N(\mu, \sigma^2)$$

- In particular, the normal distribution with $\mu = 0$ and $\sigma = 1$ is called the ***standard normal distribution***, and is denoted as $N(0,1)$.

- The graph on the next pages shows a standard normal distribution. The "normal" normal distribution looks very much the same. It is just shifted to point $x = \mu$ and expanded σ times.

Gaussian Distribution, Central Limit Theorem

- The normal distribution is also called Gaussian distribution.

- The normal distribution is used in many situations because of the fact that when you combine a large number of random variables, each with a fairly arbitrary distribution, to produce a new average variable (value), that resulting variable has a normal   distribution.

- That fact is described by the **Central Limit Theorem**
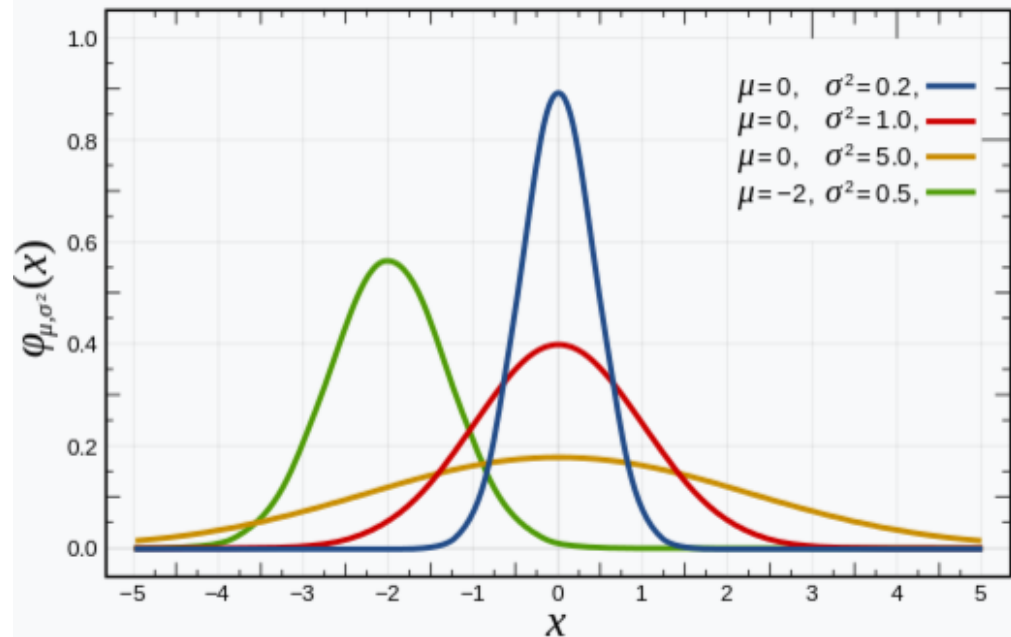
The red curve is the *standard normal distribution*

Standard Deviation σ
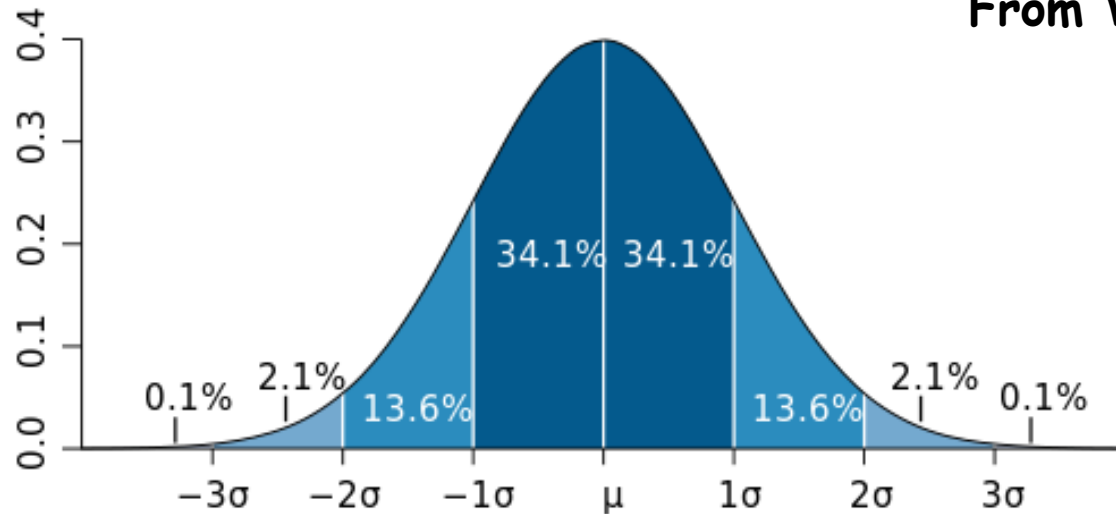
±1*σ	68.2 %
±2*σ	95.4 %
±3*σ	99.6 %

- Standard deviation provides some convenience reference points on the distribution curve. For example, more that 2/3 of all samples fall in the interval of width 2*σ around the mean.
- Similarly, 95.4% of all samples fall in the interval of width 4*σ around the mean.
- Now you know where the term 6σ is coming from.

@Zoran B. Djordjevic

Central Limit Theorem, Arbitrary Distributions

- Let $X_1, X_2, \ldots, X_N$ be a set of N independent random variables and each X_i has an arbitrary probability distribution $P(X_i)$ with mean μ_i and a finite variance $\sigma_i{}^2$ Then, the variable

$$X_{Norm} = \frac{1}{N} \left(\sum_{i=1}^{N} X_i - \sum_{i=1}^{N} \mu_i \right)$$

- i.e., the variation of the sum of variables X_i from the sum of their means, in the case when N (the number of random variables) is large, approaches a distribution function which is normally distributed. The mean of the resulting distribution is $\mu = 0$ and the variance is equal to $\sigma_X = {}^1/_N \sqrt{\sum_{i=1}^{N} \sigma_i^2}$

- Note that variances add as if random processes are vectors in an N-dimensional vector space.

CLT, Collection of Identical Distributions

- If all variables $\{X_i\}$ have the same probability distribution with identical variance σ_x, and mean μ_x then the average variable

$$X_{Norm} = \frac{1}{N}\left(\sum_{i=1}^{N} X_i\right)$$

- is normally distributed with $\mu_X = \mu_x$ and variance $\sigma_X = \sigma_x/\sqrt{N}$

[1] The **mean** of the population of random variable is always equal to the mean of the parent population from which the population samples were drawn.

[2] The **standard deviation** of the population of means is always equal to the standard deviation of the parent population divided by the square root of the sample size (N).

[3] Most importantly, **the distribution of means** will increasingly approximate a **normal distribution** as the size N of samples increases.

- The Central Limit Theorem explains the ubiquity of the famous bell-shaped "Normal distribution" (or "Gaussian distribution") in the all kinds of measurements.

Application of Normal Distribution

Problem

- Assume that the test scores of a college entrance exam fits a normal distribution. Furthermore, the mean test score is 72, and the standard deviation is 15.2. What is the percentage of students scoring 84 or more in the exam?

Solution

- We apply the function `pnorm()` of the normal distribution with mean 72 and standard deviation 15.2. Since we are looking for the percentage of students scoring higher than 84, we are interested in the *upper tail* of the normal distribution.

```
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)
[1] 0.21492
```

- The percentage of students scoring 84 or more in the college entrance exam is 21.5%.

> ?pnorm

- Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd. Usage:

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
Arguments
```

`x,q`   vector of quantiles.

`p`   vector of probabilities.

`n`   number of observations.

`mean`   vector of means.

`sd`   vector of standard deviations.

`log, log.p` if `TRUE`, probabilities `p` given as `log(p)`.

`lower.tail` if `TRUE`(default), probabilities are `P[X ≤ x]` otherwise, `P[X > x]`.

If `mean` or `sd` are not specified they assume the default values of `0` and `1`, respectively.

Exponential Distribution

- The **exponential distribution** describes the arrival time of a randomly recurring independent event sequence. If μ is the mean waiting time for the next event recurrence, its probability density function is give by:

$$f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

Graph to the right corresponds to the exponential distribution with $\mu = 1$.
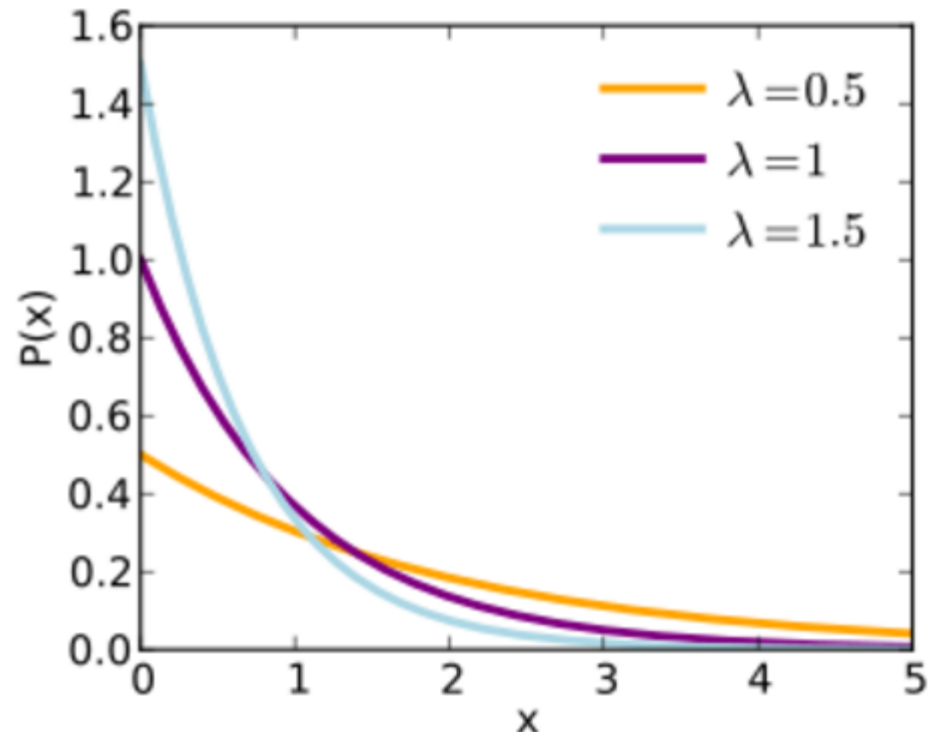
Exponential Distribution

Problem

- Suppose the mean checkout time of a supermarket cashier is three minutes. Find the probability of a customer checkout being completed by the cashier in less than two minutes.

Solution

- The checkout processing rate is equals to one divided by the mean checkout completion time. Hence the processing rate is 1/3 checkouts per minute. We then apply the function `pexp()` of the exponential distribution with rate=1/3.

```
> pexp(2, rate=1/3)
[1] 0.48658
```
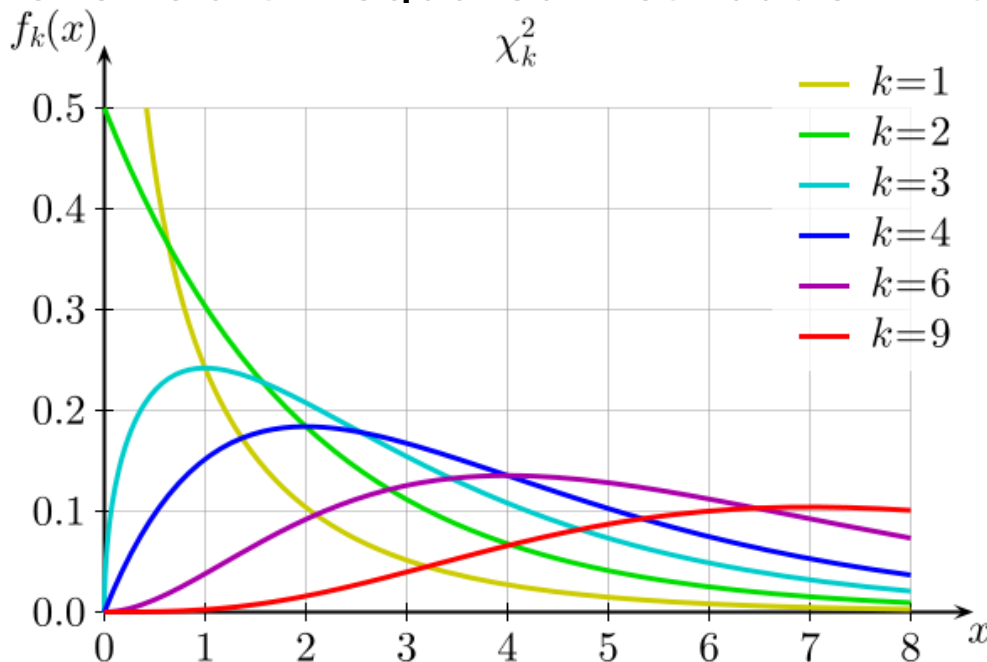
Answer

- The probability of finishing a checkout in under two minutes by the cashier is 48.7%

Chi-squared Distribution

- If $X_1, X_2, \ldots, X_m$ be a set of **m** independent random variables each having the standard normal distribution, then variable V, defined as the sum of squares of $\{X_i\}$ -s

$$V = X^2{}_1 + X_2{}^2 + \ldots + X^2{}_m$$

Follows a Chi-Squared Distribution with m-degrees of freedom.

The mean value of variable V is m and its variance is equal to $2m$