INTRODUCTION TO R

E. Fersini

INTRODUCTION

- Why R?

R is a **programming** and statistical **language**.

R is usefull for **machine learning**, data analysis and **visualization**.

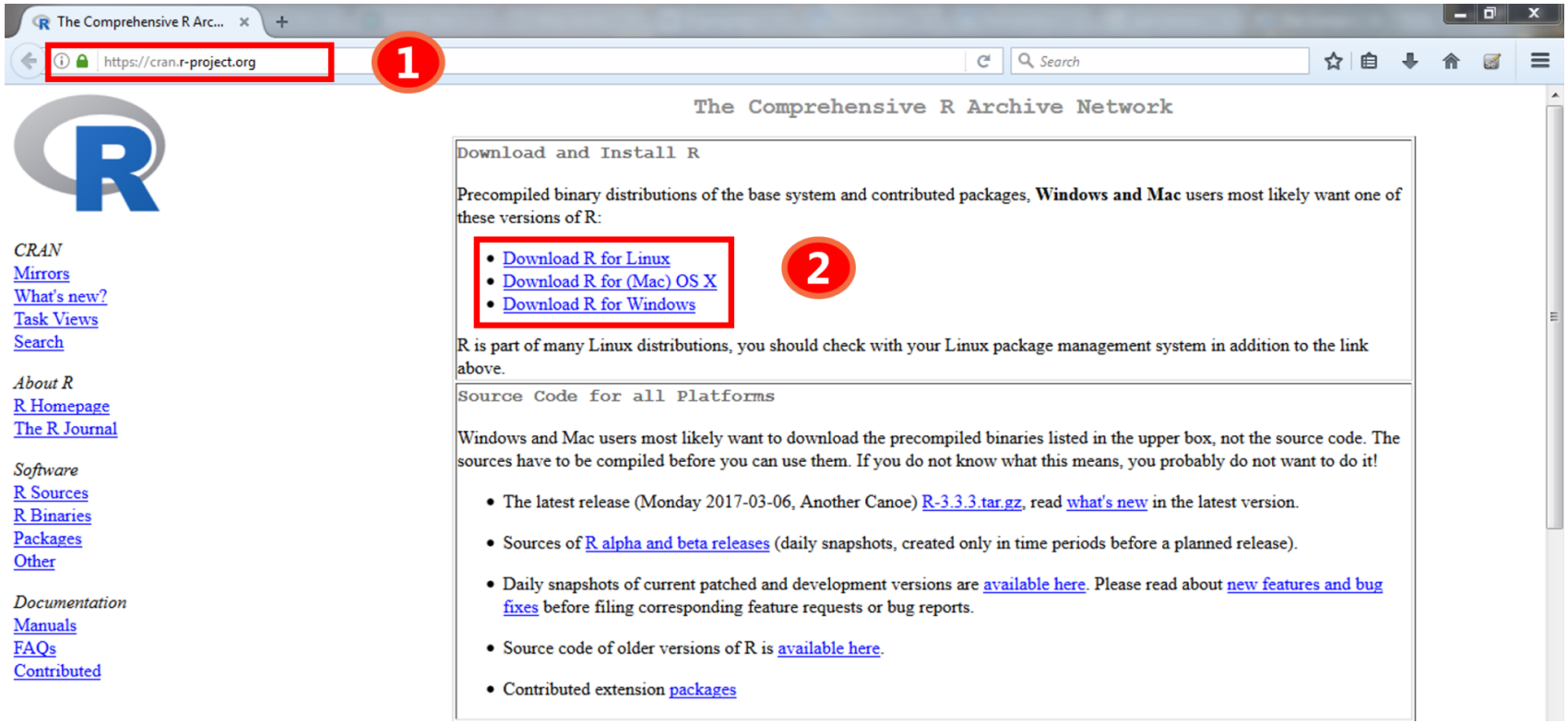
R is **simple** and easy to learn, read and write.

R is **open source**.

R has an IDE, **RStudio**.

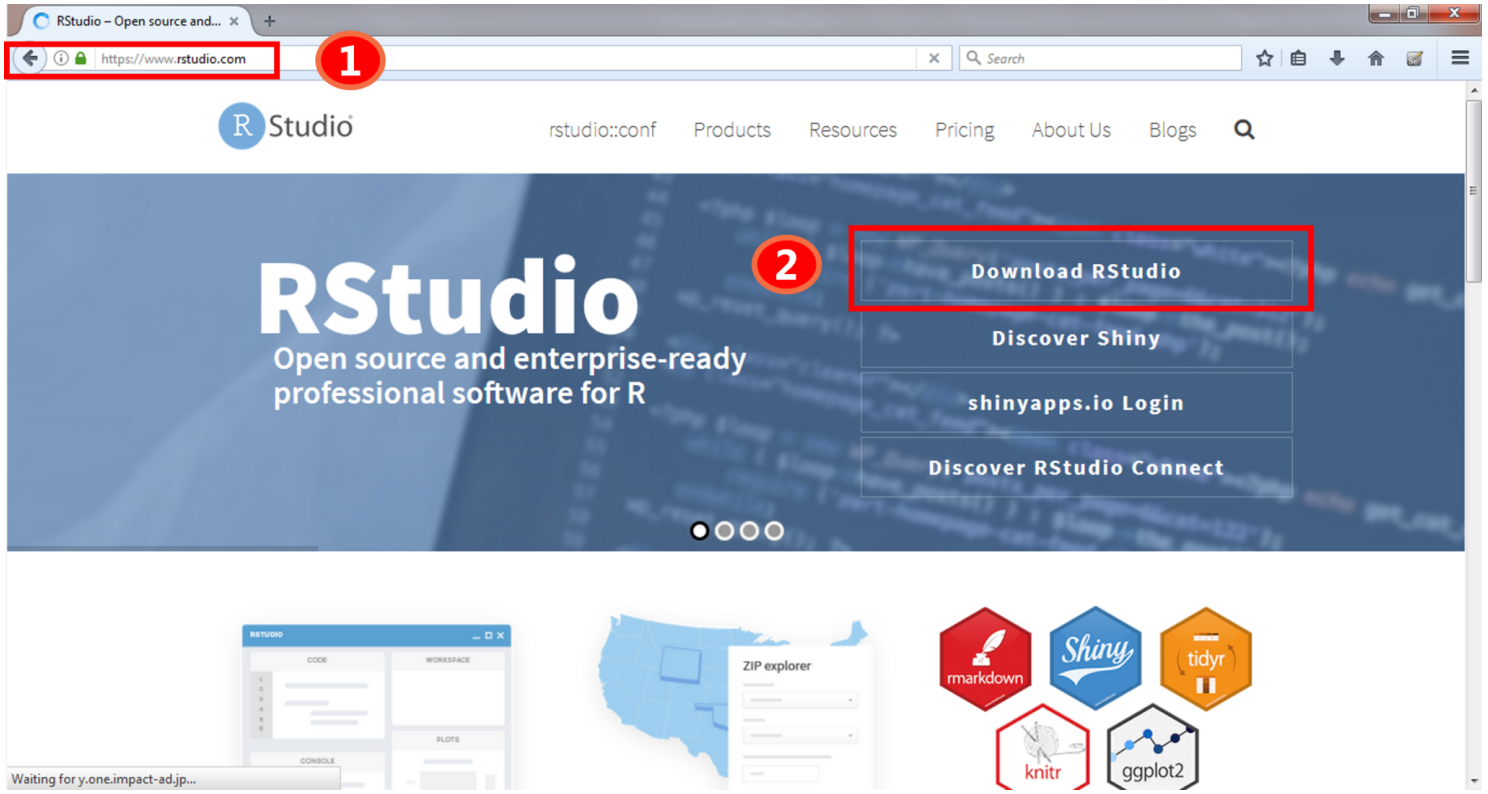
INSTALLING R

- If you want to install R on your system:
  - Step 1 : Go to the link- https://cran.r-project.org/
  - Step 2 : Download and install R on your system.

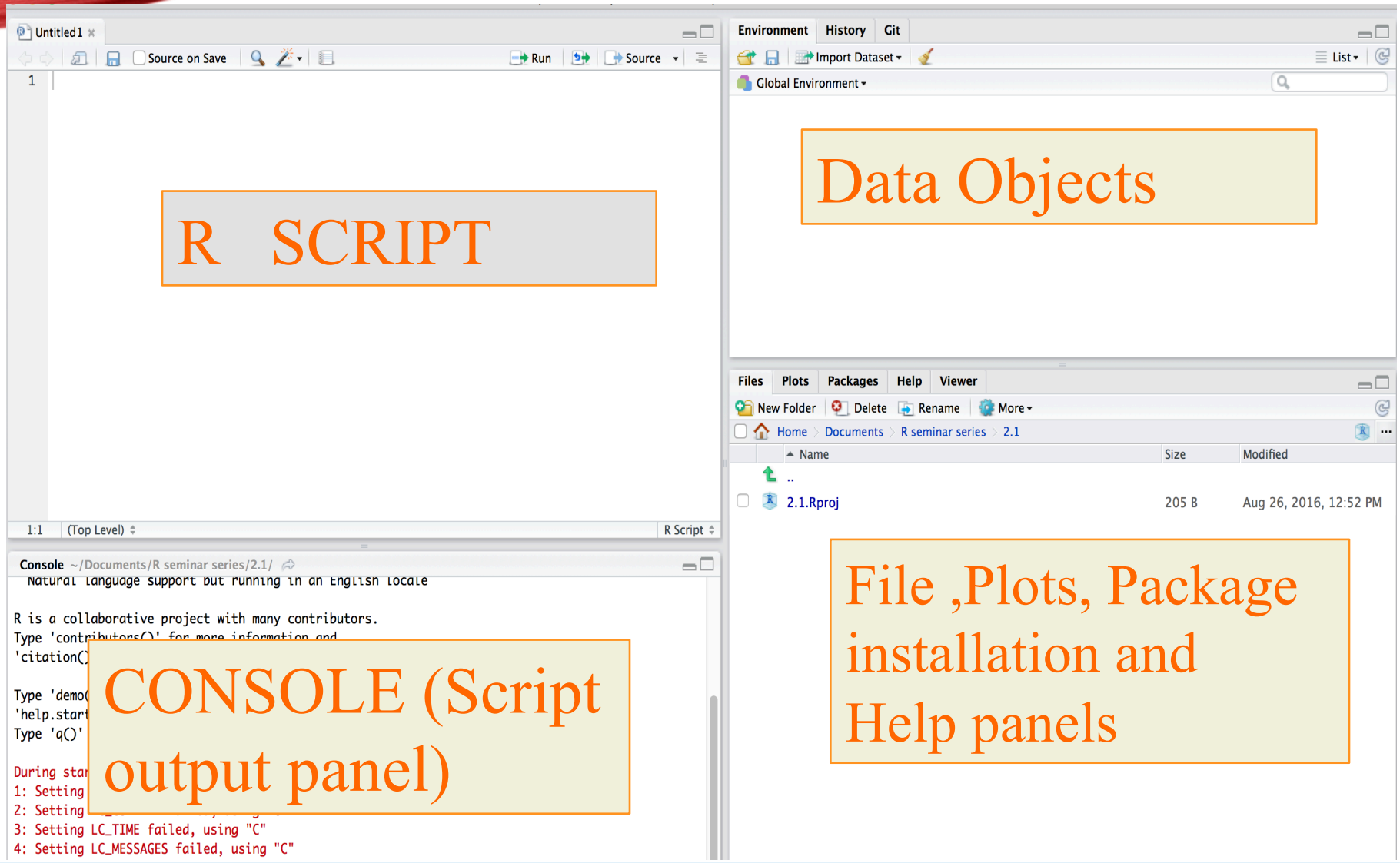


INSTALLING RStudio

- If you want to install RStudio on your system:
  - Step 1 : Go to the link https://www.rstudio.com/
  - Step 2 : Download and install RStudio on your system.

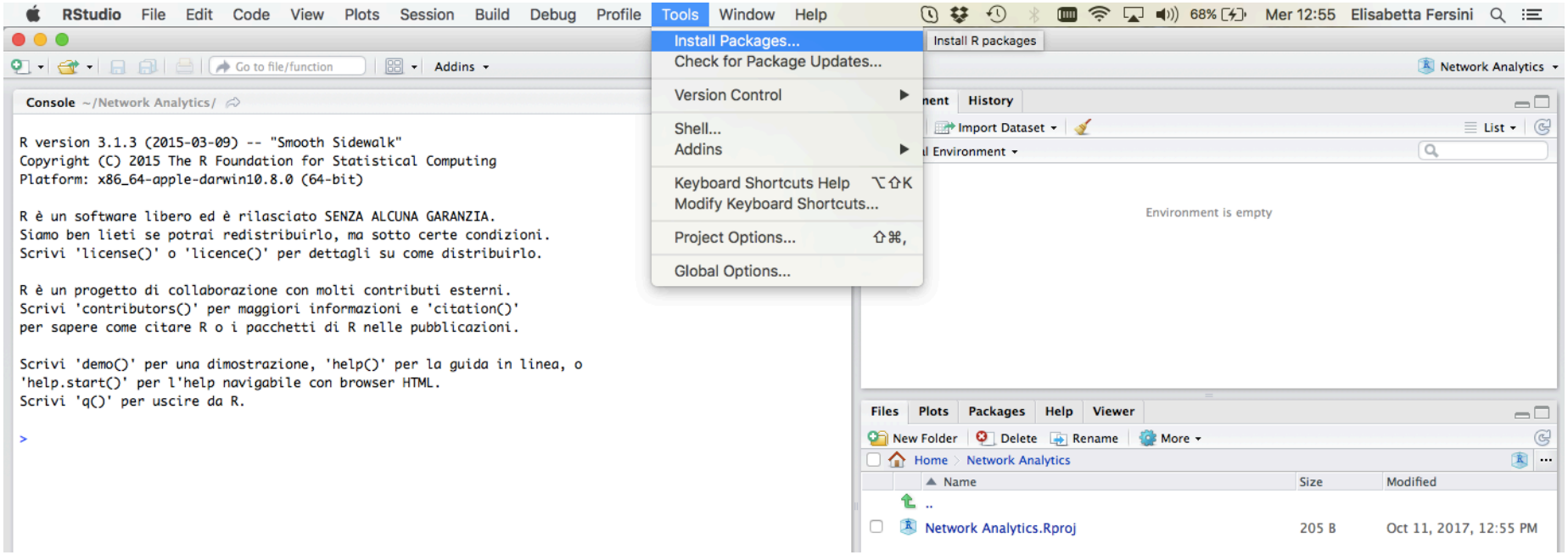


RStudio


Untitled1
Source on Save
Run
Source
1
1:1
(Top Level)
R Script
R SCRIPT
Environment
History
Git
Import Dataset
List
Global Environment
Data Objects
Files
Plots
Packages
Help
Viewer
New Folder
Delete
Rename
More
Home
Documents
R seminar series
2.1
Name
Size
Modified
..
2.1.Rproj
205 B
Aug 26, 2016, 12:52 PM
Console ~/Documents/R seminar series/2.1/
Natural language support but running in an English locale
R is a collaborative project with many contributors.
'citation()'
Type 'demo(
'help.start
Type 'q()'
During star
1: Setting
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
CONSOLE (Script output panel)
File ,Plots, Package installation and Help panels

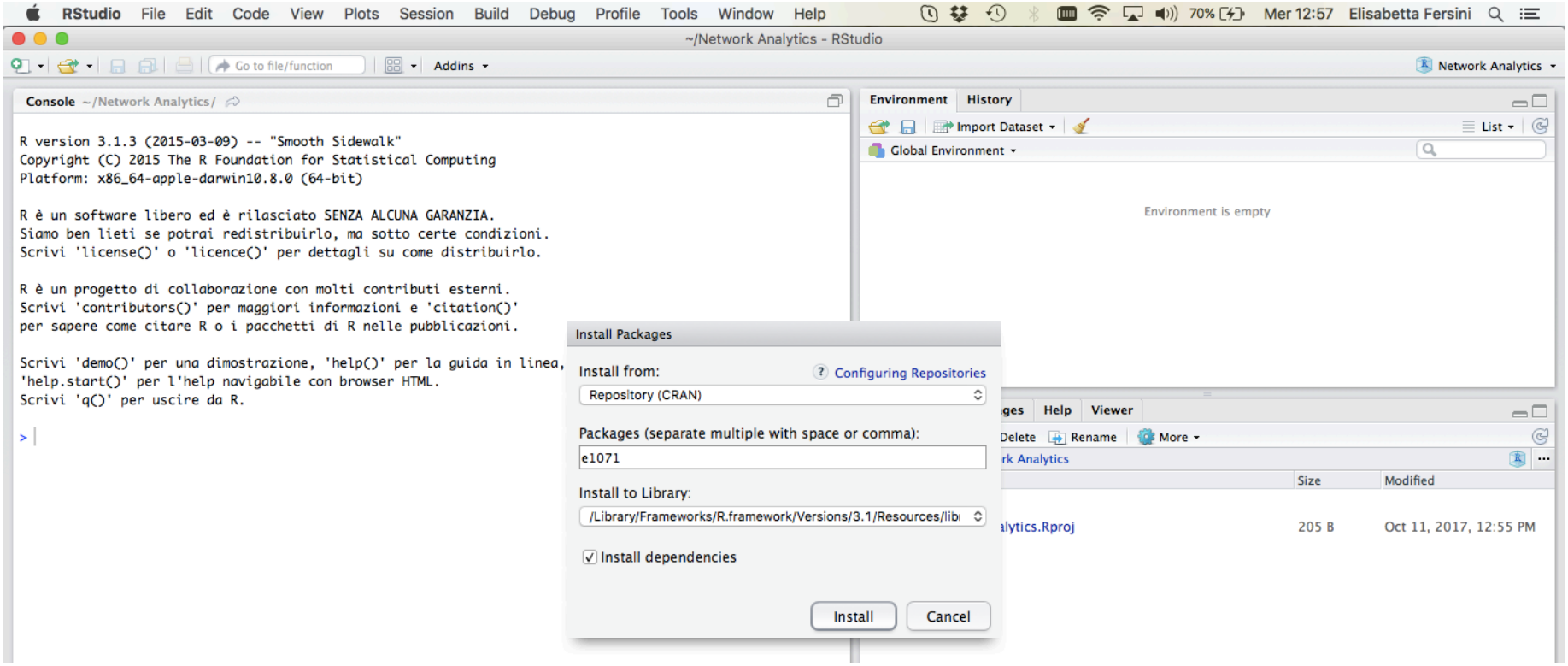
INSTALLING PACKAGES

- Any package that you want to use must be firstly installed:



INSTALLING PACKAGES

- Any package that you want to use must be firstly installed:



LOADING PACKAGES

- Once you have installed a package, you need to load it to use it:

```
library (e1071)
```

- If you would like to view the documentation of the package, you can use the help()function:

```
help(package ="e1071")
```

DATA OPERATORS IN R

- There are mainly 5 different types of operators:
  1. Arithmetic Operators: Perform arithmetic operations such as addition, subtraction, multiplication, division etc.
  2. Assignment Operators: Assignment operators are used to assign values.
     - Assignment Operators **=** **<-** **-**

```
x=5
x
[1] 5
```

```
x<-12
x
[1] 12
```

```
26 -x
x
[1] 26
```

DATA OPERATORS IN R

3. Relational Operator: It defines a relation between two entities

```
x=5
x<9
[1] TRUE
```

4. Logical Operator: Used often to verify conditions

```
2&3
[1] TRUE

2&!3
[1] FALSE
```

DATA OPERATORS IN R

5. Special Operators: These operators are used for specific purpose, not for logical computation. Examples:

   - Creating the series of numbers in sequence for a vector

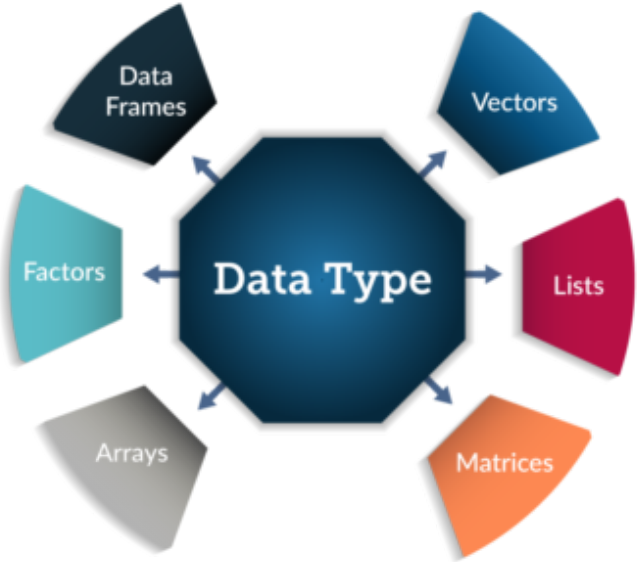
```
x=4:8
x
[1] 4 5 6 7 8
```

   - %in% is used to identify if an element belongs to a vector

```
x=4:8
y=6
y%in%x
[1] TRUE
```

DATA TYPES

- In R, we do not need to declare a variable as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable.
- There are mainly six data types present in R:



DATA TYPES

- **VECTOR**: It is a sequence of data elements of the same basic type

```
v=c(1,3,5,7)
```

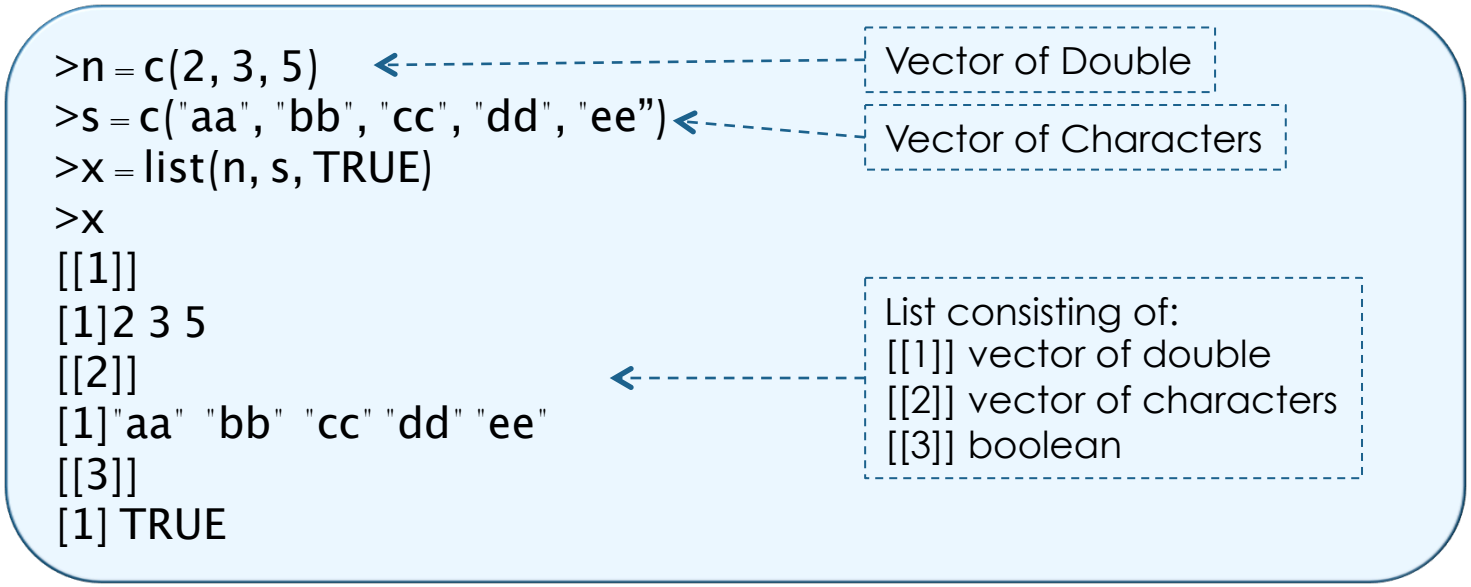
- There are 5 classes of vectors:

Remark: in many versions numerical vectors are treated as double precision real numbers. To explicitly create integers, add a L at the end

```
v1=c(1L, 3L, 5L, 7L)
```

DATA TYPES

- **LIST**: Lists are the R objects which contain elements of different types like – numbers, strings, vectors and another list inside it.



DATA TYPES

- **MATRIX**: Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. A Matrix is created using the matrix() function.
  - Example: matrix(data, nrow, ncol, byrow, dimnames) where,

```
Mat <- matrix(c(1:16), nrow=4, ncol= 4)
Mat
     [,1] [,2] [,3] [,4]
[1,]  1    5    9    13
[2,]  2    6    10   14
[3,]  3    7    11   15
[4,]  4    8    12   16
```

DATA TYPES

- **ARRAY**: Arrays are the R data objects which can store data in more than two dimensions. It takes vectors as input and uses the values in the *dim* parameter to create an array.

```
>vector1= c(5,9,3)

>vector2 = c(10,11,12,13,14,15)

result = array(c(vector1,vector2),dim= c(3,3,2))
```

```
, , 1
     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2
     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15
```

DATA TYPES

- **FACTORS**: Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in data analysis for statistical modeling.

```
c("East","West","East","North","North","East","West","West“,"East“)
>factor_data <- factor(data)
>factor_data

[1] East  West  East  North North East  West  West  East
Levels: East North West
```

DATA TYPES

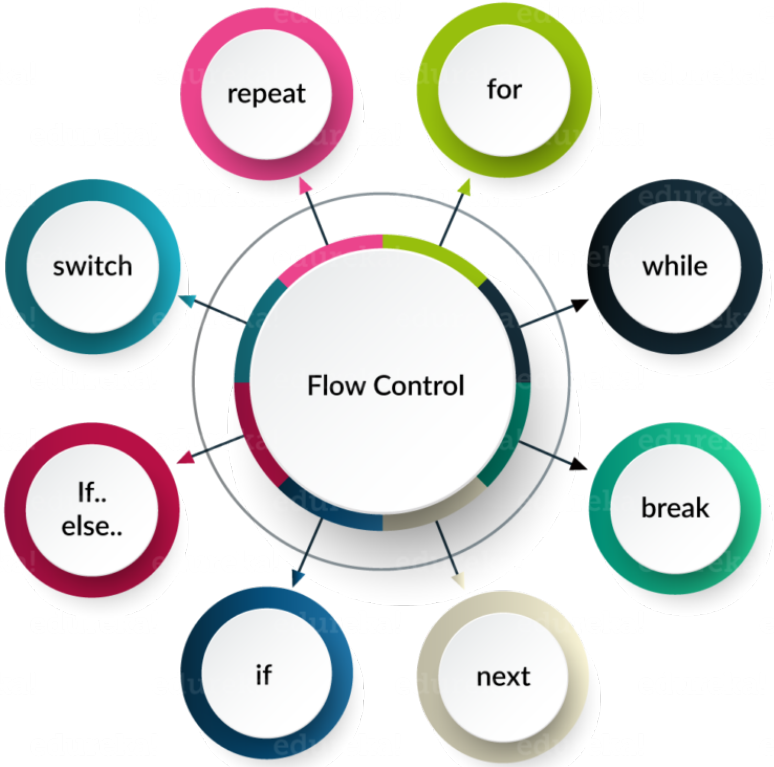
- **DATA FRAME**: A data frame is a table in which each column contains values of one variable and each row contains one set of values from each column.

```
>std_id = c (1:5)
>std_name = c("Rick","Dan","Michelle","Ryan","Gary")
>marks = c(623.3,515.2,611.0,729.0,843.25)
>std.data <- data.frame(std_id, std_name, marks)
>std.data
```

std_id	std	name	marks
1	1	Rick	623.30
2	2	Dan	515.20
3	3	Michelle	611.00
4	4	Ryan	729.00
5	5	Gary	843.25

FLOW CONTROL

- **Flow control statements** play an important role as they allow you to control the flow of execution of a script inside a function.



FLOW CONTROL

- **IF**, **ELSE** and **ELSEIF**: These control statement evaluate a single/ multiple condition(s).

```
x=5
if(x>0){
    print ("Positive number")
  }
```

```
x = -5
if(x>0){print ("Positive number")}
else {print ("Negative number")}
```

```
x = -5
if(x>0){print ("Positive number")}
else if (x<0) {print ("Negative number")}
else print("Zero")
```

FLOW CONTROL

- **SWITCH**: These control statements are basically used to compare a certain expression to a known value.

```
vtr <- c(150,200,250,300,350,400)
option <-"mean"
switch(option,
            "mean" = print(mean(vtr)),
            "mode" = print(mode((vtr))),
            "median" = print(median((vtr)))
)
```

FLOW CONTROL

- **REPEAT**: The repeat loop helps to execute the same set of code again and again until a stop condition is met at the end.
- **WHILE**: same loop, but the condition is checked at the beginning.

```
x=2
while(x<1000){
   x=x^2
   print(x)
}
```

FLOW CONTROL

- **FOR:** For loops are used when you need to execute a block of code several number of times.

```
vtr= c(7,19,25,65, 45)
 for( i in vtr){
   print(i)
 }
```

FLOW CONTROL

- **Break**: Break statements help to terminate program, loops and switch.

```
x =1:5
for (val in x){
   if (val == 3){
            break
  }
  print(val)
}
```

FLOW CONTROL

- **NEXT**: A next statement is used when you want to skip the current iteration of the loop without terminating it.

```
for(i in 1:15){
   if((i%%2)==0){
            next
   }
  print(i)
}
```

SOME USEFUL FUNCTIONS

- NUMERIC functions

Function	Description
abs(*x*)	absolute value
sqrt(*x*)	square root
ceiling(*x*)	ceiling(3.475) is 4
floor(*x*)	floor(3.475) is 3
trunc(*x*)	trunc(5.99) is 5
round(*x*, digits=*n*)	round(3.475, digits=2) is 3.48
signif(*x*, digits=*n*)	signif(3.475, digits=2) is 3.5
cos(*x*), sin(*x*), tan(*x*)	also acos(*x*), cosh(*x*), acosh(*x*), etc.
log(*x*)	natural logarithm
log10(*x*)	common logarithm
exp(*x*)	e^*x*

SOME USEFUL FUNCTIONS

- Functions on STRINGS

Function	Description
substr(*x*, **start=***n1*, **stop=***n2***)**	Extract or replace substrings in a character vector. x <- "abcdef" substr(x, 2, 4) is "bcd" substr(x, 2, 4) <- "22222" is "a222ef"
grep(*pattern*, *x* , **ignore.case**=FALSE, **fixed**=FALSE**)**	Search for *pattern* in *x*. If fixed =FALSE then *pattern* is a regular expression. If fixed=TRUE then *pattern* is a text string. Returns matching indices. grep("A", c("b","A","c"), fixed=TRUE) returns 2
sub(*pattern*, *replacement*, *x*, **ignore.case** =FALSE, **fixed**=FALSE**)**	Find *pattern* in *x* and replace with *replacement* text. If fixed=FALSE then *pattern* is a regular expression. If fixed = T then *pattern* is a text string. sub("\\s",".","Hello There") returns "Hello.There"
strsplit(*x*, *split***)**	Split the elements of character vector *x* at *split*. strsplit("abc", "") returns 3 element vector "a","b","c"
paste(..., sep="")	Concatenate strings after using *sep* string to seperate them. paste("x",1:3,sep="") returns c("x1","x2" "x3") paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3") paste("Today is", date())
toupper(*x***)**	Uppercase
tolower(*x***)**	Lowercase

SOME USEFUL FUNCTIONS

- STATISTICAL functions

Function	Description

dnorm(*x***)**	normal density function (by default m=0 sd=1)
	# plot standard normal curve
	x <- pretty(c(-3,3), 30)
	y <- dnorm(x)
	plot(x, y, type='l', xlab="Normal Deviate", ylab="Density", yaxs="i")

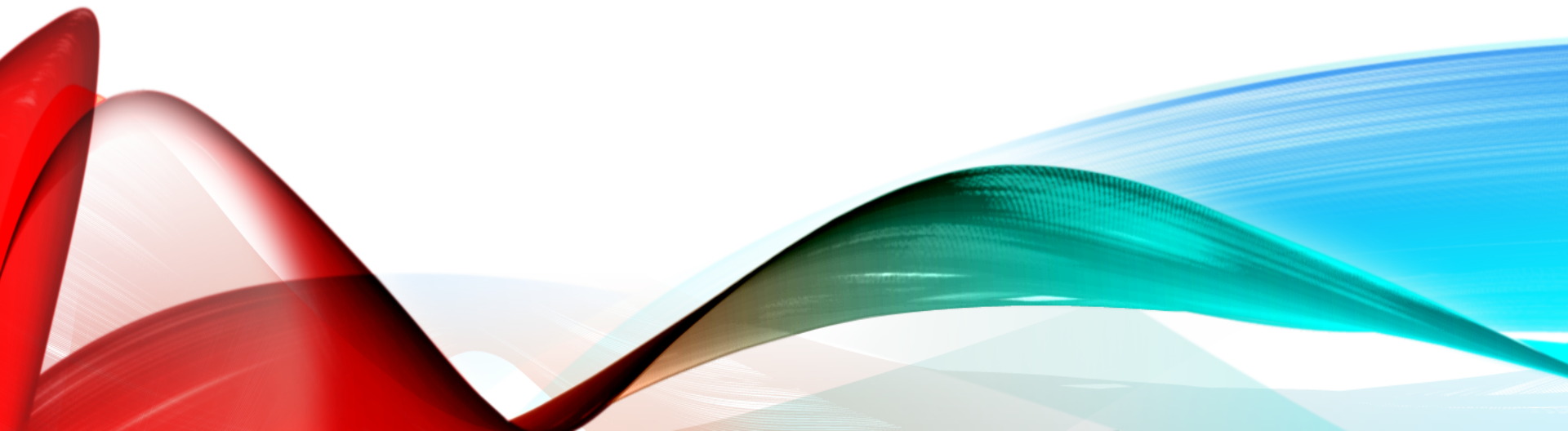
pnorm(*q***)**	cumulative normal probability for q (area under the normal curve to the right of q) pnorm(1.96) is 0.975
qnorm(*p***)**	normal quantile. value at the p percentile of normal distribution qnorm(.9) is 1.28 # 90th percentile
rnorm(*n***, m=0,sd=1)**	n random normal deviates with mean m and standard deviation sd. #50 random normal variates with mean=50, sd=10 x <- rnorm(50, m=50, sd=10)

dbinom(*x***, *size*, *prob*)**
pbinom(*q***, *size*, *prob*)**
qbinom(*p***, *size*, *prob*)**
rbinom(*n***, *size*, *prob*)**
	binomial distribution where size is the sample size
	and prob is the probability of a heads (pi)
	# prob of 0 to 5 heads of fair coin out of 10 flips
	dbinom(0:5, 10, .5)
	# prob of 5 or less heads of fair coin out of 10 flips
	pbinom(5, 10, .5)

dpois(*x***, *lamda*)**
ppois(*q***, *lamda*)**
qpois(*p***, *lamda*)**
rpois(*n***, *lamda*)**
	poisson distribution with m=std=lamda
	#probability of 0,1, or 2 events with lamda=4
	dpois(0:2, 4)
	# probability of at least 3 events with lamda=4
	1- ppois(2,4)

dunif(*x***, min=0, max=1)** **punif(***q***, min=0, max=1)** **qunif(***p***, min=0, max=1)** **runif(***n***, min=0, max=1)**	uniform distribution, follows the same pattern as the normal distribution above. #10 uniform random variates x <- runif(10)

LET'S START

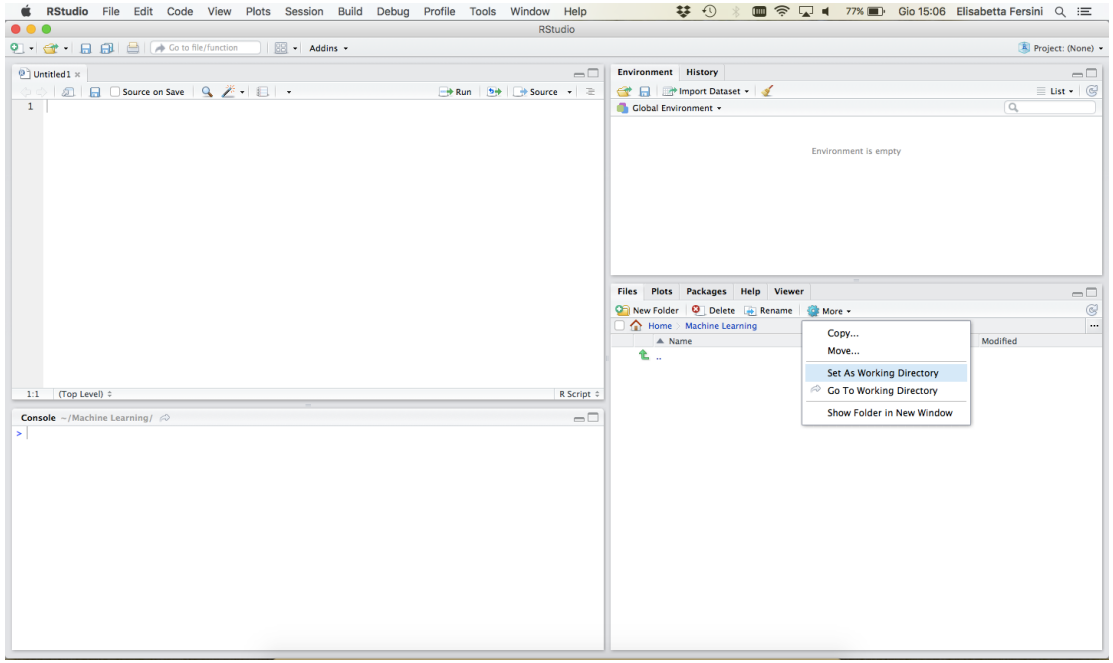


WORKING DIRECTORY

- Set your own working directory

```
setwd("~/Machine Learning")
```

- Alternatively



READING AND WRITING DATA

- Before starting to explore data, and learning from data, you must load the data into the R session.
- To view the built-in datasets of R:

```
data()
```

  - R will return a list of datasets in a dataset package, and the list comprises the name and description of each dataset.

READING AND WRITING DATA

- To load a specific dataset into an R session:

```
data(iris)
```

  - The dataset *iris* is now loaded into the data frame format, which is a common data structure in R to store a data table.

- To view the data type of *iris*, simply use the class() function:

```
class(iris)
[1] "data.frame"
```

READING AND WRITING DATA

- To store an object in a file use the save() function.

```
save(iris, file="myData.RData")
```

- To read a saved object into an R session, use the load() function.
-

```
load("myData.RData")
```

READING AND WRITING DATA

- In addition to using built-in datasets, R also provides the read.table() function to import data from text into a data frame.

```
>test.data = read.table(text = "1 2 3 4",
col.names=c("a","b"),row.names = c("first","second"))
```

- To export data to a text file, you can use the write.table() function.

```
write.table(test.data, file = "test.txt" , sep = " ")
```

READING AND WRITING DATA

- You can also export data to a comma-separated file by using the write.csv() function:

```
write.csv(test.data, file = "test.csv")
```

- With the read.csv() function, the csv le can be imported as a data frame.

```
csv.data = read.csv("test.csv", header = TRUE, row.names=1)
```

MANIPULATE DATA

- To select values, you may use a bracket notation that designates the indices of the dataset.
  - The first index is for the rows and the second for the columns:

```
iris[1,"Sepal.Length"]
```

- You can also select multiple columns using c():

```
Sepal.iris = iris[, c("Sepal.Length", "Sepal.Width")]
```

MANIPULATE DATA

- To subset data with the rows of given indices, you can specify the indices at the first index with the bracket notation.

```
Sepal.iris = iris[, c("Sepal.Length", "Sepal.Width")]
```

- To subset data with the rows of given indices, you can specify the indices at the rst index with the bracket notation.

```
Five.Sepal.iris = iris[1:5, c("Sepal.Length", "Sepal.Width")]
```

MANIPULATE DATA

- It is also possible to set conditions to filter the data.

```
setosa.data = iris[iris$Species=="setosa",1:5]
```

- The which() function returns the indexes of satisfied data.
  - The following example returns indices of the iris data containing species equal to *setosa*:

```
which(iris$Species=="setosa")
```

MANIPULATE DATA

- Besides using the bracket notation, R provides a subset() function that enables users to subset the data frame by observations with a logical statement.

```
setosa.data = subset(iris, Species =="setosa")
```

- Most of the time, you may want to apply a union or intersect a condition (AND, OR) while subsetting data.

```
example.data= subset(iris, Petal.Length <=1.4 & Petal.Width >= 0.2,
select=Species )
```

APPLYING BASIC STATISTICS

- For numeric values, you can perform descriptive statistics

```
mean(iris$Sepal.Length)
sd(iris$Sepal.Length)
var(iris$Sepal.Length)
min(iris$Sepal.Length)
max(iris$Sepal.Length)
median(iris$Sepal.Length)
range(iris$Sepal.Length)
quantile(iris$Sepal.Length)
```

APPLYING BASIC STATISTICS

- In order to obtain summary statistics on every numeric attribute of the data frame, one may use sapply.

```
sapply(iris[1:4], mean, na.rm=TRUE)
```

VISUALIZING DATA

- Calculate the frequency of species within the iris using the table command:

```
table.iris = table(iris$Species)
table.iris
```

- As the frequency in the table shows, each species represents 1/3 of the iris data. We can draw a simple pie chart to represent the distribution of species within the iris:

```
pie(table.iris)
```

VISUALIZING DATA

- The hist() function creates a frequency plot of sorts along the x-axis.

```
hist(iris$Sepal.Length)
```

- The boxplot() function allow you to convey a lot of information in one simple plot.

```
boxplot(Petal.Width ~ Species, data = iris)
```

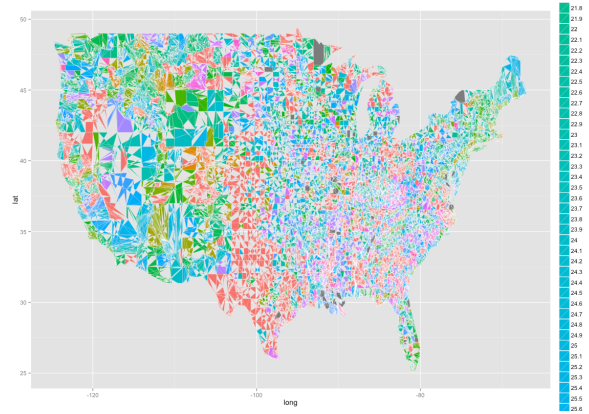
VISUALIZING DATA

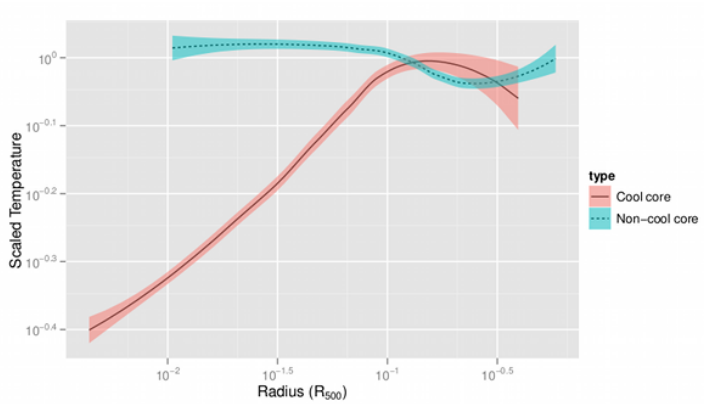
- A scatter plot is used when there are two variables to plot against one another.

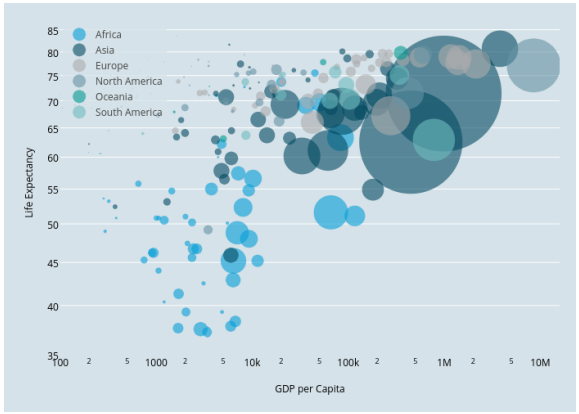
```
plot(x=iris$Petal.Length, y=iris$Petal.Width, col=iris$Species)
```

VISUALIZING DATA

- The ggplot2 package is another plotting system for R. It allows users to add, remove, or alter components in a plot with a higher abstraction.
- For those of you interested in the topic of ggplot2, you can refer to this site: http://ggplot2.org/
- Some good examples of "fancy" plots can be found at http://www.r-graph-gallery.com/portfolio/ggplot2-package/

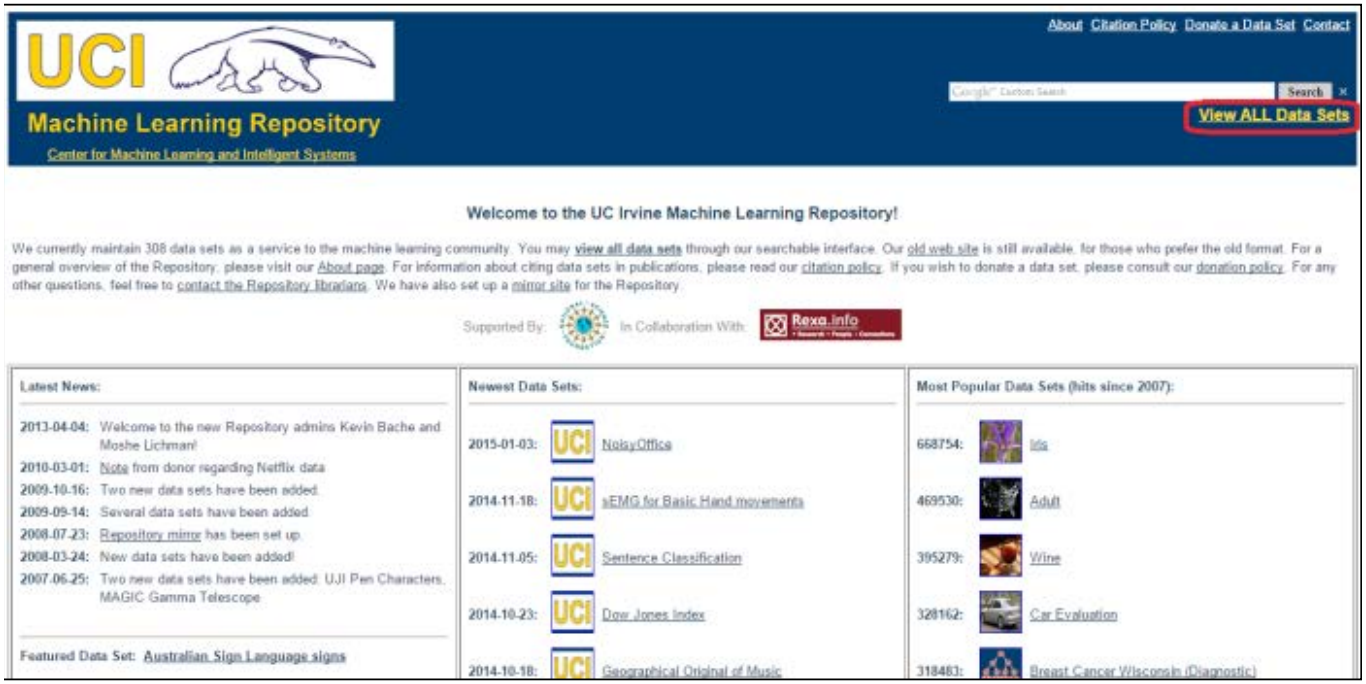






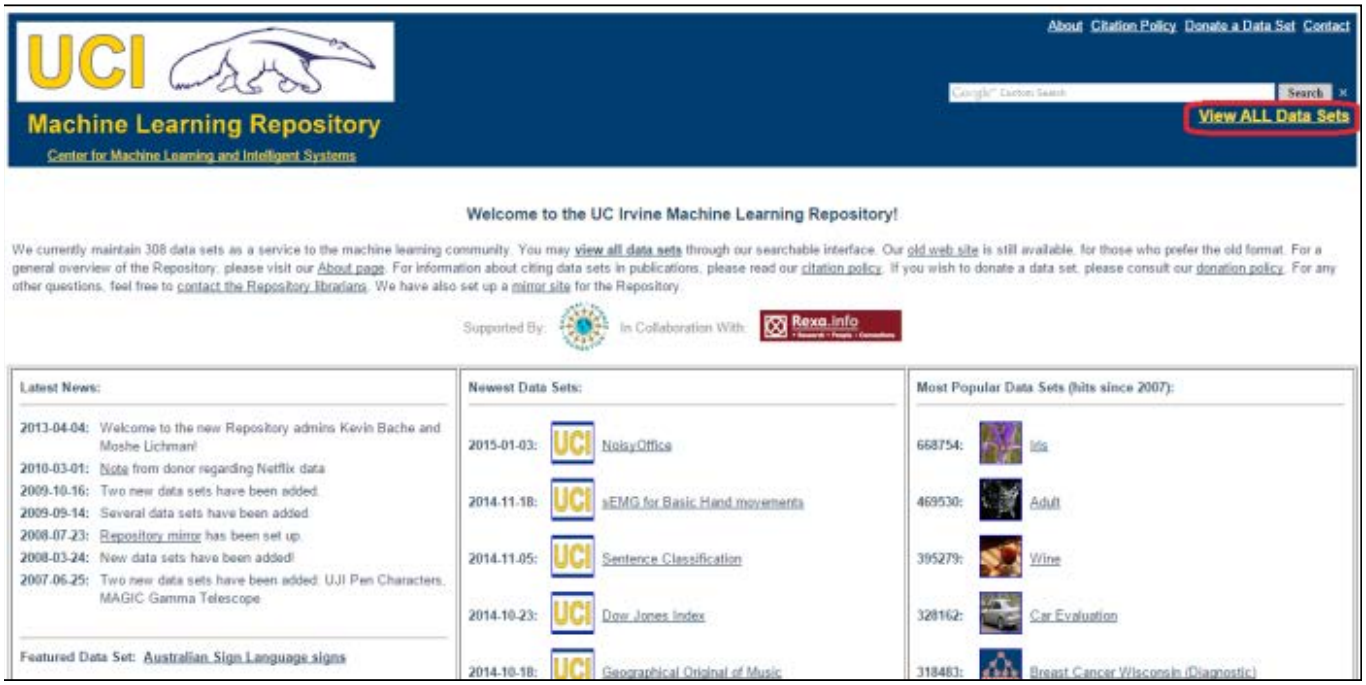
GETTING A DATASET FOR ML

- To obtain data from external data repositories:
  1. Access the UCI machine learning repository:
     - http://archive.ics.uci.edu/ml/



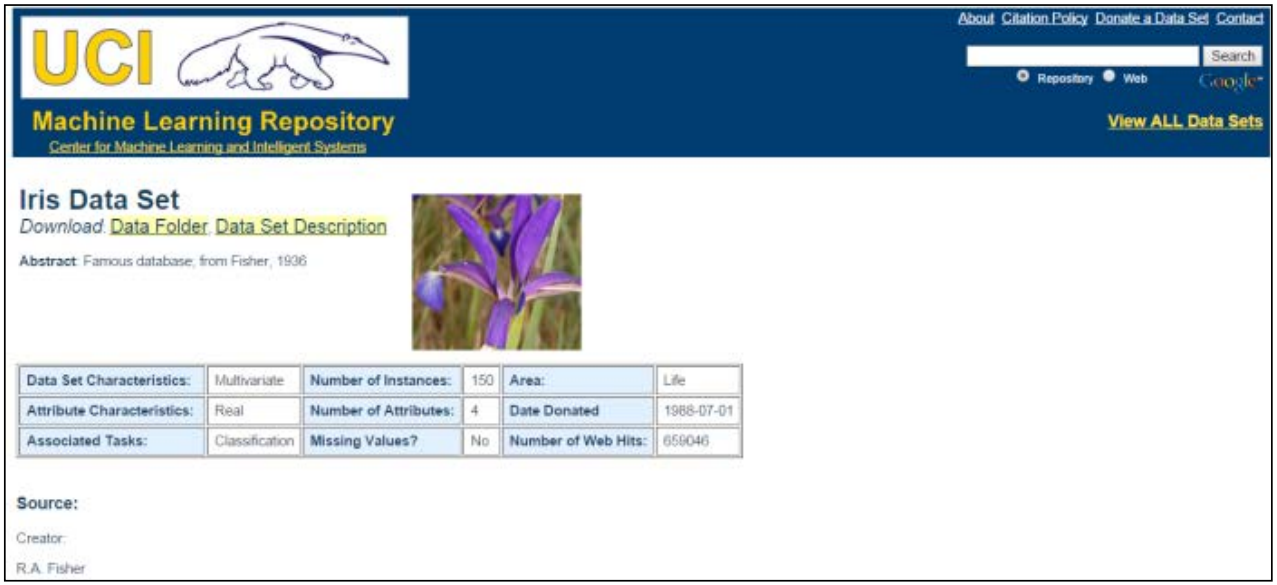
GETTING A DATASET FOR ML

2. Click on View ALL Data Sets. Here you will find a list of datasets containing field names, such as Name, Data Types, Default Task, Attribute Types, # Instances, # Attributes, and Year:
3. Search for iris dataset



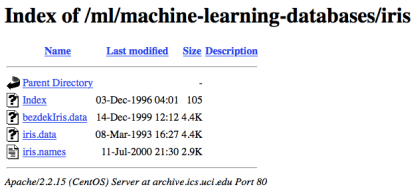
GETTING A DATASET FOR ML

4. Click on iris. This will display the data folder and the dataset description:



GETTING A DATASET FOR ML

5. Click on Data Folder, which will display a directory containing the iris dataset:



Index of /ml/machine-learning-databases/iris

Name	Last modified	Size	Description
Parent Directory		-	
Index	03-Dec-1996 04:01	105	
bezdekIris.data	14-Dec-1999 12:12	4.4K	
iris.data	08-Mar-1993 16:27	4.4K	
iris.names	11-Jul-2000 21:30	2.9K	

Apache/2.2.15 (CentOS) Server at archive.ics.uci.edu Port 80

6. You can tuse the read.csv() function to read the dataset:

```
iris.data = read.csv(url("http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"), header = FALSE,  col.names = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width","Species"))
```

OTHER DATASET FOR ML

- Kdnuggetsoffers a resourceful list of datasets for machine learning (and data mining).
  - http://www.kdnuggets.com/datasets/index.html

- Kaggle offers a resourceful list of datasets and competitions for machine learning (and data mining).
  - www.kaggle.com

A CLOSER LOOK AT THE DATASET

- Before starting to create a machine learning model, we should have a closer look at the dataset:
  - *Dimensions.*
  - *Feature types.*
  - *Look at the row data*
  - *Visualising the dataset*

SIZE OF THE DATASET

- We can have an idea of the size of the dataset by using the dim() function:
  - how many instances (rows).
  - how many attribute/features/coordinate (columns) the dataset.

```
dim(iris)
```

FEATURE TYPE OF THE DATASET

- We also need to know the type of the attribute of a given dataset
  - It could be useful to use a specific model or to make some operations on the instances/features
  - Recall that the main feature can be:
    - *Double*
    - *Integer*
    - *String*
    - *Factor*

```
sapply(iris, class)
```

Remark: ***class*** returns a character vector of all classes an object inherits from

LOOK AT THE ROW DATA

- To look at the row data, you can use the head() function:

```
head(iris)
```

- To know the nominal classes in the dataset, you can use the levels() function:
  - In the iris dataset, the classification problem is multinomial (3 levels)

```
levels(iris$Species)
```

Remark: the symbol **$** allows us to have a direct access to a given attribute

VISUALIZING THE DATASET

- We can understand better our dataset, by using two types of visulalization:
  - Univariate: to understand a single attribute
  - Multivaritate: to understand the relationships between attributes

VISUALIZING THE DATASET

- In many many machine learning problems it is useful to distinguish input attributes (input space) from target variable (output space)

```
x = iris[,1:4]
y = iris[,5]
```

- We can visualize the input attribute by **using univariate plots**, e.g. (single chart) box plot

```
par(mfrow=c(1,4))
for(i in 1:4) {
      boxplot(x[,i], main=names(iris)[i]) }
```

Remarks:
- The **par( )** function combine multiple plots into one overall graph.
- The option **mfrow=c(**nrows, ncols**)** create a matrix of nrows x ncols plots that are filled in by row.
- **mfcol=c(**nrows, ncols**)** fills in the matrix by columns.

VISUALIZING THE DATASET

- We can have a more clear idea by plotting the distribution of each feature of the input space

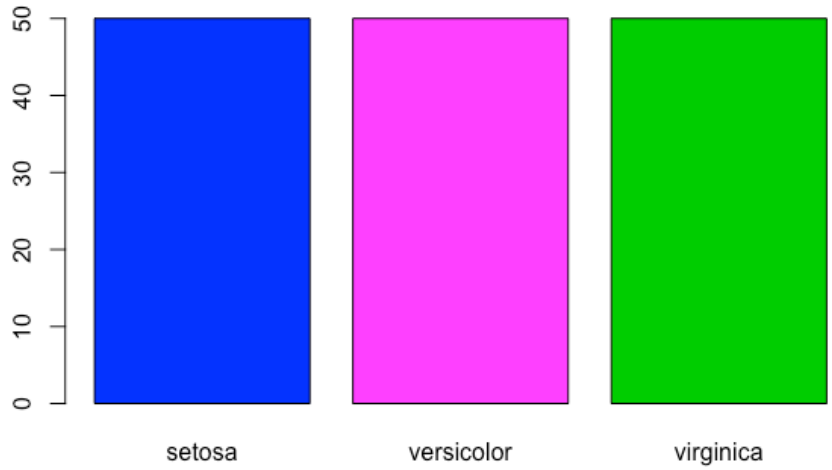
```
library(caret)
featurePlot(x, y, plot="density", scales=list(x=list(relation="free"),
y=list(relation="free")), auto.key=list(columns=3))
```

Remarks: Since the distributions are different in terms of scale, the options **scales=list(x=list(relation="free"), y=list(relation="free"))** automatically makes the chart scale free

VISUALIZING THE DATASET

- We can also visualize our target variable distribution, by using the plot() function:

```
plot(y,col=c(4,6,3))
```



VISUALIZING THE DATASET

- We can also try to understand if some features are "related" to some other features by using a **multivariate plot**
- Inizialize the caret package and plot the scatterplot of all couples of input attribtutes

```
featurePlot(x=x, y=y, plot="pairs", auto.key=list(columns=3))
```