Basic Operations in R

abs (x)	Returns the absolute value of x
as.data.frame (x)	Converts x (generally a matrix) into a data frame
as.Date (*V*, "*FM*")	Tells to R that the variable *V* contents dates in the format *FM*.
as.character (*V*)	Converts the variable *V* into characters
as.factor (*V*)	Converts the variable *V* in a factor variable
as.matrix (x)	Converts x (usually a table or vector) into a matrix
as.numeric (*L*)	Converts the logical expression *L* into numbers (0 or 1)
as.numeric (*V*)	Converts the variable *V* in a numerical variable
as.vector (x)	Converts x into a vector (x is usually a matrix)
c (a, b, ... , n)	Creates a vector containing the elements (numbers) a, b, ..., n
c ("a", "b", ... , "n")	Creates a vector containing the elements (characters) "a", "b", ..., "n"
cbind (F_1, F_2, ... F_n)	Combines the data frames F_1, F_2, ... F_n into only one data frame, taking into account that F_1, F_2, ... F_n have the same observations in different variables
colMeans (*M*)	Gets the mean from all the elements contained in every column from matrix *M*
colnames (*F*)	Gets the names of the columns contained in the data frame *F* (*F* can also be a matrix)
colSums (*M*)	Adds up all the elements contained in every column from matrix *M*
complete (mice(*F*))	Completes the missing values of the data frame F based on the values in the other variables. Requires loading the "mice" package. A split seed could also be useful.
cor (*F*)	Calculates the correlation between all the variables contained in the data frame *F*
cor (*F*[c("V_1", " V_2", ..., " V_n ")])	Calculates the correlation between the variables V_1, V_2, ..., V_n, all contained in the data frame *F.*
cor (V_1, V_2)	Calculates the correlation between the variables V_1 and V_2
coredata (*F*)	Extracts the core data from the data frame *F*
coredata (*V*)	Extracts the core data from the variable *V*
data.frame (V_1, V_2, ..., V_n)	Creates a data frame combining the vectors V_1, V_2, ..., V_n
diag (*M*)	Gets the diagonal elements from a table / matrix
dim (x)	Gets the dimension from the object x
dim (x) = c (nr, nc)	Changes the dimension of the object x, resulting in an object with nr rows and nc columns
F [, !(names(*F*) %in% *X*)]	Removes all the variables whose names are contained in the vector *X* from frame *F*
F [c("V_1", "V_2", ..., "V_n")]	Shows only the selected variables V_1, V_2, ..., V_n of the data frame *F*
F [n,]	Gets the element n from the data frame *F*
F [,n]	Gets the column n from the data frame *F*
F [sample(nrow(*F*), n),]	Takes a random sample of n elements from the data frame *F*
F$*Vr*	Accesses to the variable *Vr* contained in the data frame *F*
getwd ()	Gets the current working directory
grepl("W", *V*, fixed=TRUE)	If the word W appears in the variable *V*, returns "TRUE". If not, returns "FALSE"
head (x)	Shows the six first elements of x
image (*M*, axes=FALSE)	Plots an image from the matrix *M,* using default colours
is.na (*V*)	Returns a vector of TRUE/FALSE for if the variable *V* is missing.
jitter (x)	Adds or subracts a small amount of random noise to x
length (x)	Provides the length of the element x
ls ()	Shows a list with all the variables created in the current R session
M$residuals	Gets the residuals from the model *M*
$M_1 * M_2$	Multiplicates M_1 and M_2, element by element (escalar product)
match ("Name", *V*)	Searches where the word "Name" is in the variable *V*

matrix (*V*, byrow=TRUE, nrow=n)	Creates a matrix from the vector *V,* dividing its elements into n rows
mean (*V*)	Calculates the mean of the variable *V*
min (*V*)	Returns the minimum value of the variable *V*
months (*V*)	Extracts the months from the variable *V*, which has a date format
names (*F*)	Gets the names of all the variables contained in the data frame *F*
na.omit (*F*)	Removes observations with any missing value (or "NA" value) from the data frame *F*
na.rm = TRUE	Additional argument that removes the "NA" values from *V*. It can be used with funcions like mean, sd, max, min, tapply, etc.
nchar (x)	Counts the number of characters contained in x
ncol (*F*)	Gets the number of variables (or columns) contained in the data frame *F*
nrow (*F*)	Gets the number of observations (or rows) contained in the data frame *F*
paste0 ("A", V_1, V_2, "B", ...)	Pastes the text A with the text contained in the variables V_1 and V_2, and with the text B, without leaving spaces between any items (Output: AV_1V_2B)
paste ("A", V_1, V_2, "B", ...)	Does the same as paste0, but leaving one space between the items (Output: A V_1 V_2 B).
rbind (F_1, F_2, ... F_n)	Combines the data frames F_1, F_2, ... F_n into only one data frame, taking into account that F_1, F_2, ... F_n have different observations with the same variables
read.csv ("File.csv")	Reads the .csv file "File" into R
relevel (*V*, "Name")	Sets the value "Name" as the reference of the unordered factor variable *V*
rm (*V*)	Removes the variable *V*
rowMeans (*M*)	Gets the mean from all the elements contained in every row from matrix *M*
rowSums (*M*)	Adds up all the elements contained in every row from matrix *M*
sd (*V*)	Calculates the standard desviation of the variable *V*
seq (a, b, c)	Creates a vector with a sequence of numbers from a to b with increments of c
sign (n)	Returns 1 if n is a positive number, -1 if n is a negative number, and 0 if n is zero.
sort (table(*V*))	Sorts a table which counts the number of observations in the variable *V* by the number of observations in that variable
sort (*V*)[k]	Gets the value of the element k contained in the variable *V*, sorted from the element wi with the least value (k = 1) to the element with the highest value (k = n), where n is the total number of observations contained in the data set.
sqrt (x)	Returns the square root of x
str (*F*)	Gets the structure from the data frame *F*
subset (*F*, *C*)	Gets a subset from the data frame *F* with the elements satisfying condition *C*
sum(diag(table(*T*)))/nrow(*F*)	If *T* is a classification matrix from a logistic / classification tree model, and *F* is a data frame corresponding, this command gets directly the accuracy
sum (x)	Gets the sum from all the elements contained in x (x could be a vector or a table/matrix)
summary (*F*)	Gets the summary from the data frame *F*
summary (*M*)	Gets the summary from the model *M*
summary (*V*)	Gets the summary from the variable *V*
table (*V*)	Counts the number of observations in each category in the variable *V*
table (V_1, V_2, ..., V_n)	Counts the number of observations in each category in the variables V_1, V_2, ..., V_n
tail (x)	Shows the six last elements of x
tapply (V_1, V_2, *O*)	Splits the data by the variable V_2, and applies the operation *O* to the variable V_1
V = NULL	Removes the variable *V*
V [n]	Gets the element n from the vector *V*
weekdays (*V*)	Extracts the weekdays from the variable *V*, which has a date format
which (V == "Text")	Finds the element in a data frame whose variable *V* is exactly "Text"
which.max (*V*)	Finds the element in a data frame whose variable *V* has the maximum value
which.min (*V*)	Finds the element in a data frame whose variable *V* has the minimum value

write.csv (*F*, "File.csv")	Saves the data frame *F* in the .csv file called "File.csv"
%in%	In operator
&	And operator
?function	Opens the R help and shows the features of the function
|	Or operator

Linear Regressions in R

Creating a one variable linear regression from the data frame *F*, with *VD* as the dependient variable and *VI* as the independient variable	lm (*VD* ~ *VI*, data=*F*)
Creating a multivariable linear regression from the data frame *F*, with *VD* as the dependient variable and VI_1, VI_2, ..., VI_n as independient variables	lm (*VD* ~ VI_1 + VI_2 + ... + VI_n, data=*F*)
Building a linear model from the data frame *F* to predict the dependient variable, *VD*, using all the other variables in the data frame.	lm (*VD* ~ ., data=*F*)
Getting the summary from the model *M*	summary (*M*)
Computing the SSE of the model *M*	sum ((*M*$residuals)^2)
Making predictions using the model *M* in the training set *F*	predict (*M*)
Making predictions using the model *M* in the new data frame or testing set *TestF*	predict (*M*, newdata=*TestF*)
Computing the SST, where *F* is the training set, *TestF* is the name of the testing set and *V* is the dependient variable we are trying to predict	sum ((*TestF*$*V* - mean(*F*$*V*))^2)
Computing the out-of-sample SSE of the model *M,* where *TestF* is the name of the testing set, *V* is the dependient variable and P is the vector where the predictions are stored	sum ((*TestF*$*V* - *P*)^2)
Computing the out-of-sample R^2 of the model *M,* taking into account that SSE refers to out-of-sample SSE	1 - SSE/SST
Creating automatically a model with a good compromise of model simplicity and R^2 from the model *M*, which has been built previously	step (*M*)

Splitting Randomly a dataset in R (Continuous problems)

Sets a seed with the number n. If we split randomly the same data set two or more with the same seed n, the same splitting will be got all the times	set.seed (n)
Splits the data frame *F* in two data sets, leaving a proportion of p data in the first data set, and a proportion of (1-p) data in the second set.	sample (1:nrow(*F*), size=p*nrow(*F*))
Creates the first dataset (from the data set *F*), given the split is called *S*	*F*[*S*,]
Creates the second dataset (from the data set *F*)), given the split is called *S*	*F*[-*S*,]

Splitting Randomly a dataset in R (All problems)

Loads the library caTools	library (caTools)
Sets a seed with the number n. If we split randomly the same data set two or more with the same seed n, the same splitting will be got all the times	set.seed (n)
Splits the outcome variable *V* in two data sets, leaving a proportion of p data in the first data set, and a proportion of (1-p) data in the second set, making sure that the outcome variable is well-balanced in each set	sample.split (*V*, SplitRatio = p)
Creates the first dataset (from the data set *F*), given the split is called *S*	subset (*F*, *S* == TRUE)
Creates the second dataset (from the data set *F*)), given the split is called *S*	subset (*F*, *S* == TRUE)

Creating Logistic Regressions in R

Creating a one variable logistic regression from the data frame *F*, with *VD* as the dependient variable and *VI* as the independient variable	glm (*VD* ~ *VI*, data=*F,* family=binomial)
Creating a multivariable logistic regression from the data frame *F*, with *VD* as the dependient variable and VI_1, VI_2, ..., VI_n as independient variables	glm (*VD* ~ VI_1 + VI_2 + ... + VI_n, data=*F,* family=binomial)
Creating a logistic model from the data frame *F* to predict the dependient variable, *VD*, using all the other variables in the data frame except V_1, V_2, ..., V_n.	glm (*VD* ~ ., data=*F,* family=binomial)
Creating a logistic model from the data frame *F* to predict the dependient variable, *VD*, using all the other variables in the data frame.	glm (*VD* ~ . -V_1 -V_2 -... - V_n, data=*F,* family=binomial)
Getting the summary from the model *M*	summary (*M*)
Making predictions of probabilites using the model *M* in the training set *F*	predict (*M,* type="response")
Making predictions using the model *M* in the new data frame or testing set *TestF*	predict (*M*, type="response", newdata=*TestF*)

Computing ROC and AUC in R

Loads the library "ROCR"	library (ROCR)
Creates a prediction taking into account the predictions (*P*) made by the model and the actual values of the variable *V*	prediction (*P*, *V*)
Does the same than the previous command, but with classification trees. The [,2] means that the prediction function has to take the second column	prediction (*P*[,2], *V*)
Plots the ROC curve. PR is the output of the prediction function. The other arguments are optional, and are used to colour the plot and to add values of thresolds to it.	plot (performance(*PR*, "tpr", "fpr") , colorize = TRUE, print.cutoffs.at = seq(0,1,0.1), text.adj = c(-0.2,1.7))
Computes the AUC from the prediction made in the previous step, called *P2*	AUC = as.numeric(performance(*P2*, "auc")@y.values)

Cross Validation Method in R (CART)

Loads the library "caret"	library (caret)
Loads the library "e1071"	library (e1071)
Defines the number of folds that we want, in this case, n	trainControl (method="cv", number = n)
Defines the cp parameters from a to b in increments of c	expand.grid (.cp=seq(a, b, c))
Gets the optimal value of the cp parameter. F is the name of the training set, *C* is the output of the trainControl funcion and *G* is the output of the expand.grid function	train (*VD* ~ VI_1 + VI_2 + ... + VI_n, data=*F,* method="rpart", trControl=*C*, tuneGrid=*G*)

CART Models in R

Loads the library "rpart"	library (rpart)
Loads the library "rpart.plot"	library (rpart.plot)
Creates a classification tree from the data frame *F*, with *VD* as the dependient variable and VI_1, VI_2, ..., VI_n as independient variables. If no additional arguments are added, R will built the tree by default. Some additional arguments that can be added are the following:	rpart (*VD* ~ VI_1 + VI_2 + ... + VI_n, data=*F,* method="class")
Sets that in each split must be a minimum of n points	minbucket = n
Sets the cp (complexity parameter) value equal to k	cp = k
Sets the matrix *M* as a penalty matrix	parms=list(loss=*M*)
Creates a regression tree from the data frame *F*, with *VD* as the dependient variable and VI_1, VI_2, ..., VI_n as independient variables, with a complexity parameter of n. If no additional arguments are added, R will built the tree by default. Some additional arguments that can be added are the following:	rpart (*VD* ~ VI_1 + VI_2 + ... + VI_n, data=*F*)
Sets the cp (complexity parameter) value equal to k	cp = k
Plots the tree *T,* with n significant digits in each split	prp (*T,* digits = n)
Loads the library "randomForest"	library (randomForest)
Converts the variable we are trying to predict, *V,* into a factor.	as.factor (*V*)
Creates a random forest from the data frame *F*, with *VD* as the dependient variable and VI_1, VI_2, ..., VI_n as independient variables, with a minimum of n points in every split and a number m of trees.	randomForest (*VD* ~ VI_1 + VI_2 + ... + VI_n, data=*F,* method="class", nodesize=n, ntrees=m)
Makes predictions in unseen data *TestF*, in classification problems, based on the tree model *T* and using a thresold of 0.5. If the argument "newdata" is removed, the predictions will be made in the training set	predict (*T*, newdata=*TestF,* type="class")
In classification problems, gives the probabilities of every of the possible outputs happening If the argument "newdata" is removed, the predictions will be made in the training set	predict (*T*, newdata=*TestF*)
Makes predictions in unseen data *TestF*, in regression problems, based on the tree model *T* If the argument "newdata" is removed, the predictions will be made in the training set	predict (*T*, newdata=*TestF*)
Makes predictions in unseen data *TestF*, in classification problems, based on the random forest model *R.* If the argument "newdata" is removed, the predictions will be made in the training set. Adding the argument type="prob" is also optional. If it's not added, the output will be a prediction; if it's added, the output will be the probability of something happening.	predict (*R*, newdata=*TestF*, type="prob")

Interpreting Random Forests

Plots a chart for each variable measuring the number of times that variable was selected for splitting. *M* is the name of the model, *VU* is the output from the first line, and *VUS* is the output from the second line	varUsed (*M*, count = TRUE) sort (*VU*, decreasing = FALSE, indexreturn = TRUE) dotchart (*VUS*$x, names(M$forest$xlevels[*VUS*$ix]))
Plots the reduction in impurity of each variable in the model *M*	varImpPlot (*M*)

Text Analytics in R

Sets the language to default (English)	Sys.setlocale ("LC_ALL", "C")
Reads the .csv file "File" into R, keeping the text in a way that can be analyzed	read.csv ("File.csv", stringsAsFactors=FALSE)
Loads the library "tm"	library (tm)
Loads the library "SnowballC"	library (SnowballC)
Converts the text contained in the variable *V* into a corpus (a corpus is a collection of documents)	Corpus (VectorSource (*V*))
Gets the text contained in the element n of the corpus *C*	*C*[[n]]$content
Changes all the text contained in the corpus *C* to lowercase.	tm_map (*C*, content_transformer(tolower))
Removes all the puntuaction sings contained in the corpus *C*	tm_map (*C*, removePunctuation)
Gets a list with the english stopwords.	stopwords ("english")
Removes all english stopwords from the corpus *C*	tm_map (*C*, removeWords, stopwords("english"))
Removes all english stopwords and the words "W_1", "W_2", ... , "W_n" from the corpus *C*	tm_map (*C*, removeWords, c("W_1", "W_2", ... , "W_n", stopwords("english")))
Stems all the words from the corpus *C* (stemming a word is getting the root from it)	tm_map (*C*, stemDocument)
Creates a document term matrix counting the number of times that every word contained in the corpus *C* appears on it	DocumentTermMatrix (*C*)
Shows the matrix created with the previous command, called *M*	inspect (*M*)
Finds the terms appearing at least n times in the matrix *M* (generated from the corpus *C*)	findFreqTerms (*M*, lowfreq=n)
Removes the terms appearing in the matrix *M* in a proportion lower than (1 - p)	removeSparseTerms (*M*, p)
Converts the matrix *M* into a standard matrix, and then, into a data frame	as.data.frame (as.matrix(*M*))
If the data frame F got from the previous step has any variable with its name starting with a number, fixes it	make.names (colnames(*F*))

Hierarchical clustering in R

Gets the euclidean distance in every variable contained in the data frame (or vector, but in this case there is only one variable) *F.* (Be careful! In large data sets this command could collapse the computer)	dist (*F*, method = "euclidean")
Computes the hierarchical clustering inputting the distances *D*, where *D* is the output got from the previous command	hclust (*D*, method = "ward.D")
Plots the dendrogram of the clustering *C*, where *C* is the output of the hclust function	plot (*C*)
In the graph plotted with the previous command, shows in "col" colour the result of dividing the data set by n clusters	rect.hclust (*C*, k=n, border="col")
Divides the observations into n different clusters.	cutree (*C*, k=n)
Runs the second argument (colMeans) on each element of the first argument. The result is finding the centroids of every cluster contained in the group of clusters *G* (*G* is the output of the cutree function) from every variable from variable a to variable b (a and b are numbers) contained in the data frame *F*	lapply (split (*F*, *G*), colMeans)
Gets in which cluster of all the clusters contained in the group *G* is the observation n	*G* [n]
Gets all the observations of the variable *V* contained in the cluster *c*	*c*$*V*

K-means clustering in R

Loads the library "flexclust"	library (flexclust)
Computes the K-means clustering inputting the vector or data frame *V*, dividing the data into k clusters and doing no more than n iterations. The output of this command is *K*	kmeans (*V*, centers = k, iter.max = n)
Gets in what cluster is placed every observation from the vector/data frame *V*	*K*$clusters
Gets the centroids of the clusters	*K*$centers
Gets the size of each cluster	*K*$size
Converts the information got from the clustering information, *K* to an object of the class KCCA. *V* is the vector or data frame used in the kmeans function. This step is needed before using the predict function, and may take a long time. The output of this step will be *K.kcca*	as.kcca (*K*, *V*)
Using *K.kcca*, predicts in which cluster will be located every observation from the new data frame/vector, called *NV*	predict (*K.kcca*, newdata = *NV*)

Normalising data

Loads the library "caret"	library (caret)
Gets the mean and the standard deviation from the object x. Then, normalises the object y with the features of the object x. If we want to normalise an object with its own features, then the object y would be the same as the object x	predict (preProcess (x), y)

Plots & Data Visualization in R

abline (h=a)	In a plot, draws an horizontal line in y=a.
abline (v=a)	In a plot, draws a vertical line in x=a.
abline (v=as.Date(c("Dt")))	In a plot whose x-axis are dates, draws a vertical line in the point where date = "Dt"
boxplot ($V_1 \sim V_2$)	Creates a boxplot with the variable V_1 sorted by the variable V_2. V_2 is not always needed
hist (V)	Creates an histogram of the variable V
lines (x, y)	Adds a line to the line plot created previously with the variables x and y
points(x[C], y[C], col="col", pch=19)	In an existing plot, colours with the colour "col" the subset of points fulfilling the condition C
plot (x, y)	Creates a basic plot with x as the independient variable, and y as the dependient one
plot (x[a:b], y[a:b])	Creates a basic plot, but only with the observations ranged between a and b

---

The following section refers to additional arguments that can be added in most of the types of plots listed above:

breaks = N	In an histogram, divides the entire range of it in N pieces. The width of every piece is calculated by $Width = \frac{max(V) - min(V)}{N}$
col = "col"	Changes the colour of the plot to the colour "col". Examples: red, blue, green, yellow...
lwd = w	Changes the width of a line from default (w = 1) into w.
main = "Title of the plot"	Names the plot with the name "Title of the plot"
type = "l"	In a x-y plot, draws the output as a line
xlab = "Axis x name"	Labels the x-axis with the name "Axis x name"
xlab = ""	Labels the x-axis as default
xlim = c (a, b)	Limits the values on the x-axis from a to b.
ylab = "Axis y name"	Labels the y-axis with the name "Axis y name"
ylab = ""	Labels the y-axis as default

Other functions in R

image (*M*, axes = FALSE, col = grey(seq(0, 1, length = 256)))	Plots an image from the matrix *M,* using the gray colour convention
image (*M*, axes=FALSE, col = rainbow(k))	Plots an image from the matrix *M,* using k different colours from the rainbow palette
lag (zoo(*V*, -n, na.pad = TRUE)	Lags the variable *V* a number of n observations, adding NA values to the first n observations (needs loading the library "zoo" previously.
merge (*F1*, *F2*, by.x="*NameV1*", by.y="*NameV2*", all.x=TRUE)	Joins two data frames, *F1* and *F2,* matching the variable called *NameV1* from *F1* data frame with the variable called *Name2* from *F2* data frame, keeping all rows from the data frame *F1*, even if some of the rows from *V1* doesn't match any row in *F2*.
read.csv ("File.csv", header = FALSE)	Reads the .csv file "File" into R, but telling R that the data frame hasn't got a header (or variable name) row
read.table ("File.csv", header = FALSE, sep = "S", quote = "\"")	Reads the file "File.txt", but telling R that the data frame hasn't got a header (or variable name) row and making sure that the text is read in properly