R programming: syntax

For strengths and weaknesses of each algo & their syntax consult The Book.

Packages are highlighted in grey.

A) Software management

Command example	Operation explanation
library(RWeka)	Installs the RWeka package and all its dependencies
?install.packages	Help file for install package function
library(RODBC)	Loads a package into R (RODBC used for importing data from SQL databases - ODBC is a standard protocol for connecting to databases regardless of OS or Database Management System, aka DBMS)
save(x, y, z, file = “mydata.RData”)	Saves objects x, y, z regardless of whether they are vectors, factors, lists or data frames into a file of given name.
load(“mydata.RData”)	Recreates the x, y, z data structures
save.image()	Saves current session to a file called .RData (R will look for this file automatically next time you start R)
pt_data <- read.csv("/path/to/data.csv", stringsAsFactors = FALSE)	Reads Comma Seperated Value (CSV) files into an R object.
mydata <- read.csv("mydata.csv", stringsAsFactors = FALSE, header = FALSE)	By default R assumes that CSV files include headers as the first row of the file thus header = FALSE must be used for headless CSV files.
write.csv(pt_data, file = "pt_data.csv")	Used to create a CSV file from an R object

mydb <- odbcConnect("my_dsn")	DSN = data source name (required for using RODBC to import data from Open Database Connectivity Structured Query Language, ODB SQL, databases)
mydb <- odbcConnect("my_dsn", uid = "my_username" pwd = "my_password")	If a password is needed
sqlQuery()	Function used to query SQL databases
> patient_query <- "select * from patient_data where alive = 1" > patient_data <- sqlQuery(channel = mydb, query = patient_query, stringsAsFactors = FALSE)	Typical method for using SQL in R Resulting patient_data variable will be a data frame containing all rows selected using the SQL query stored in patient_query
odbcClose(mydb)	Closes the mydb connection (automatically done when R session is ended)

B) Data structures

object_name	Prints the information stored in an R object
remove(object_name)	Removes an R object
c()	Combine function which creates a vector
CVector_name <- c("John")	Writes a character vector
NVector_name <- c(9.81)	Writes a numeric vector
IVector_name <- c(12, 13)	Writes an integer vector (two entries)
LVector_name <- c(TRUE, FALSE)	Writes a logical vector (two entries)
NULL	Special vector type used in machine learning used to indicate absence of a value
NA	Special vector type used in machine learning used to indicate missing value (used for uninitialized values in vectors)
& | !	AND, OR, NOT logical operators
%>%	Pipe operator
Vector_name[2]	Prints 2nd val of vector in form: [1] 13
Vector_name[1:4]	Prints elements of vector from 1st to 4th in the form: [1] 12, 13, NA, NA
Vector_name[-2]	Prints all elements of the vector except 2nd element, in usual format.
Vector_name[c(TRUE, FALSE)]	Prints vector according to logical vector specified
factor()	Used for storing nominal values (small, medium, large), takes up less memory than c()
gender <- factor(c("MALE", "FEMALE", "MALE"))	Creating a factor of 3 genders &storing this in the 'gender' var in the form: [1] MALE FEMALE MALE Levels: FEMALE MALE
levels	Keyword for manually adding levels to factors
blood <- factor(c("O", "AB", "A"), levels = c("A", "B", "AB", "O"))	Creating a factor with 3 blood types and adding a level that did not appear in the data before writing to the var 'blood' in the form: [1] O AB A Levels: A B AB O
list()	List function which creates a list a fast way of assigning/displaying data of an object
subject1 <- list(fullname = subject_name[1], temperature = temperature[1], flu_status = flu_status[1], gender = gender[1], blood = blood[1])	Stores the following information for the 'subject1' object: $fullname [1] "John Doe" $temperature [1] 98.1 $flu_status [1] FALSE $gender [1] MALE Levels: FEMALE MALE $blood [1] O Levels: A B AB O

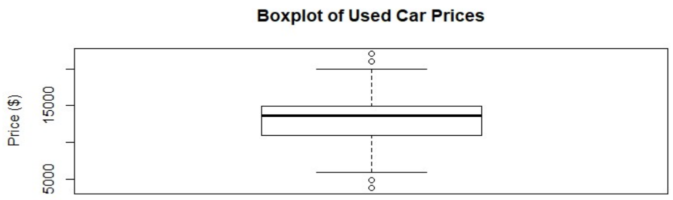
subject1[2]	Since values are labelled with the names specified in the list command this prints the values of the second feature of the subject1 object i.e.: $temperature [1] 98.1
subject1$temperature	An easier way of accessing subject1’s temperature feature. Also ensures that if you add or remove values from the list that you do not accidently retrieve the wrong list item
subject1[c(“temperature”, “flu_status”)]	Accessing several items in a list by specifying a vector of names (note lists can be used to build datasets but this is better done with a specialised data structure: the data frame = a list of vectors)
Data.frame()	Using data vectors previously created, this function combines them into a data frame (columns are features/attributes and rows are examples)
pt_data <- data.frame(subject_name, temperature, flu_status, gender, blood, stringsAsFactors = FALSE)	An example of a dataframe. stringsAsFactors = FALSE required to prevent R from automatically converting every character vector to a factor, to output: subject_name temp flu_status gender blood 1 John Doe 98.1 FALSE MALE O 2 Jane Doe 98.6 FALSE FEMALE AB 3 Steve Graves 101.4 TRUE MALE A
pt_data$subject_name	Extracts the subject names from the data frame above to output: [1] "John Doe" "Jane Doe" "Steve Graves"
pt_data[c("temperature", "flu_status")]	Similarly to lists, you can extract several features using a vector of names, to output: temperature flu_status 1 98.1 FALSE 2 98.6 FALSE 3 101.4 TRUE
pt_data[1, 2]	Extracts data from first row, second column to output: [1] 98.1
pt_data[c(1, 3), c(2, 4)]	Extracts more than one row and clumn of data from a data frame to output: temperature gender 1 98.1 MALE 3 101.4 MALE

pt_data[, 1]	Extracts all of one column where left blank
pt_data[c(1, 3), c("temperature", "gender")]	Since all methods may be used for data tables this is equivalent to: pt_data[-2, c(-1, -3, -5)]
matrix()	Function used to create a matrix
m <- matrix(c('a', 'b', 'c', 'd'), nrow = 2)	Matrix creation results in the following: [,1] [,2] [1,] "a" "c" [2,] "b" "d"
m <- matrix(c('a', 'b', 'c', 'd'), ncol = 2)	Equivalent matrix creation (specify either nrow or ncol)
m <- matrix(c('a', 'b', 'c', 'd', 'e', 'f'), nrow = 2)	Note that column-major order is implemented this columns are filled first. Output: [,1] [,2] [,3] [1,] "a" "c" "e" [2,] "b" "d" "f"
m[2, 3]	Extraction of data within a matrix. Can also be done with arrays (outside scope of sheet) Will return values f and m, like for data structures.
View(function)	Used to view a function

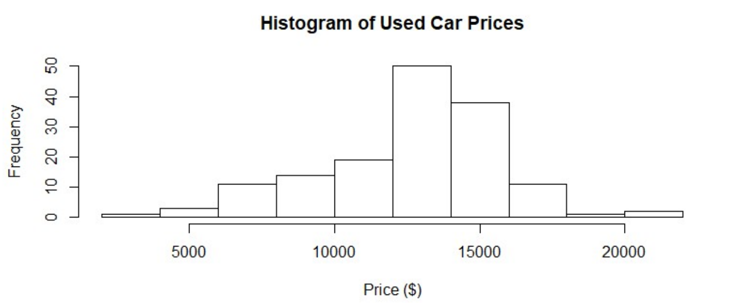
C) Univariate statistics

str(data_frame_name)	Useful exploring of data frame files. Output:
	'data.frame': 150 obs. of 6 variables:
	 $ year : int 2011 2011 2011 2011 2012 2010 2011 2010 2011 2010 ...
	 $ model : chr "SEL" "SEL" "SEL" "SEL" ...
	 $ price : int 21992 20995 19995 17809 17500 17495 17000 16995 16995 16995 ...
	 $ mileage : int 7413 10926 7351 11613 8367 25125 27393 21026 32655 36116 ...
	 $ color : chr "Yellow" "Gray" "Silver" "Gray" ...
	 $ transmission: chr "AUTO" "AUTO" "AUTO" "AUTO" ...

summary(usedcars$year)	Useful for investigating numeric variables (displays several common summary statistics). Output: Min. 1st Qu. Median Mean 3rd Qu. Max. 2000 2008 2009 2009 2010 2012
mean()	Function used to find the mean of data
median()	Function used to find the median of data
mode()	WRONG. Look at the table output for the category with the greatest number of values.
range()	Returns minimum and maximum values of data
IQR()	Used to find the inter-quartile range of data
quantile()	Identifies quantiles for a set of values
quantile(usedcars$price, probs = c(0.01, 0.99))	Returns arbitrary quantiles such as 1st and 99th percentiles. Output: 1% 99% 5428.69 20505.00
quantile(usedcars$price, seq(from = 0, to = 1, by = 0.20))	Returns: 0% 20% 40% 60% 80% 100% 800.0 10759.4 12993.8 13992.0 14999.0 21992.0
main, xlab, ylab	Parameters used to label title & axis of plots
boxplot(usedcars$price, main="Boxplot of Used Car Prices", ylab="Price ($)")	

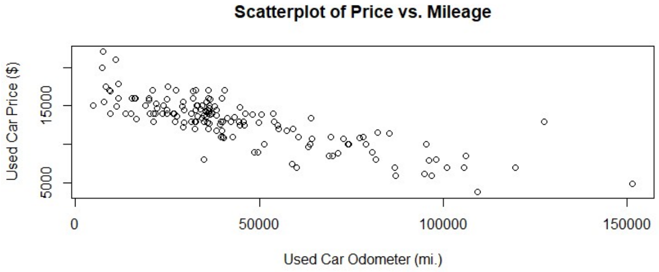


hist(usedcars$price, main = "Histogram of Used Car Prices", xlab = "Price ($)")	



var()	Outputs the variance of a dataset
sd()	Outputs the standard deviation of a dataset
table(usedcars$year)	Explores categorical variables by showing frequency of occurance of a dataset in a table. Output: 2000 2001 2002 2003 2004 2005 2006 2007 2008 $ 3 1 1 1 3 2 6 11 14 $
model_table <- table(usedcars$model) prop.table(model_table)	Builts a proportional table. Output: 2000 2001 2002 $ 0.020000000 0.006666667 0.006666667 $
> color_table <- table(usedcars$color) > color_pct <- prop.table(color_table) * 100 > round(color_pct, digits = 1)	More clean way of showing proportional tables. Output: Black Blue Gold Gray Green Red $ 23.3 11.3 0.7 10.7 3.3 16.7 $

D) Multivariate statistics

<table>
<tr><td>plot()</td><td>Plot function requires two inputs: y&x</td></tr>
<tr><td>plot(x =
usedcars$mileage, y =
usedcars$price,
main = "Scatterplot of
Price vs. Mileage",
xlab = "Used Car Odometer
(mi.)",
ylab = "Used Car Price
($)")</td><td>Scatterplot command

</td></tr>
<tr><td>usedcars$conservative <-
usedcars$color %in%
c("Black", "Gray",
"Silver", "White")</td><td>Splits the colours of used cars into conservative and non conservative categories. %in% returns TRUE or FALSE for each value in the vector on the LHS of the operator depending on whether the value is found in the vector on the RHS</td></tr>
<tr><td>CrossTable(x =
usedcars$model, y =
usedcars$conservative)</td><td>Function from the gmodels package to look at cross-tabulation. Output:
Cell Contents
|-------------------------|
| N |
| Chi-square contribution |
| N / Row Total |
| N / Col Total |
| N / Table Total |
-------------------------	 Total Observations in Table: 150 	usedcars$conservative usedcars$model	FALSE	TRUE	Row Total	 ---------------	-----------	-----------	-----------	 SE	27	51	78
0.009	0.004	
0.346	0.654	0.520
0.529	0.515	
0.180	0.340		 ---------------	-----------	-----------	-----------	 SEL	7	16	23
0.086	0.044	
0.304	0.696	0.153
0.137	0.162	
0.047	0.107		 ---------------	-----------	-----------	-----------	 SES	17	32	49
0.007	0.004	
0.347	0.653	0.327
0.333	0.323	
0.113	0.213		 ---------------	-----------	-----------	-----------	 Column Total	51	99	150
0.340	0.660		 ---------------	-----------	-----------	-----------	</td></tr>
</table>

`CrossTable(x = usedcars$model, y = usedcars$conservative, chisq = TRUE)`	This includes Chi Squarred test (probability that cell counts are due to chance alone: if low then likely that two vars are associated)
	Output:
	`Statistics for All Table Factors`
	
	`Pearson's Chi-squared test`
	`------------------------------------------------------------`
	`Chi^2 =  0.1539564     d.f. =  2     p =  0.92591`
	
	Since probability is nearly 93% it is highly likely that variations are due to chance alone and not a true association between model and colour.

1. k-Nearest-Neighbour

wbcd <- wbcd[-1]	Used to remove ID variables as a model that includes an ID will most likely suffer from overfitting.
wbcd$diagnosis <- factor(wbcd$diagnosis, levels = c("B", "M"), labels = c("Benign", "Malignant"))	Feature must be coded as a factor so we need to recode the diagnosis variable.

normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x)))
}
	Creates the normalize function. Now normalize() can be used as a regular function.

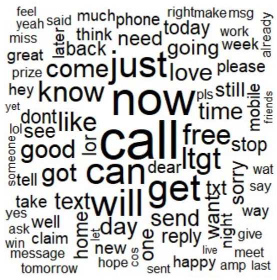
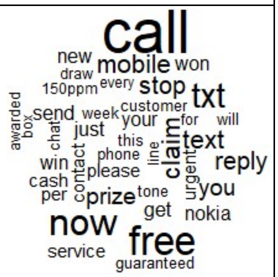
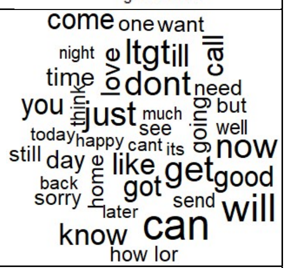
wbcd_n <- as.data.frame(lapply(wbcd[2:31], normalize))	This command applies the normalize() function to columns 2 through 31 in the wbcd data frame, converts the resulting list to a data frame, and assigns it the name wbcd_n.
wbcd_train <- wbcd_n[1:469,] wbcd_test <- wbcd_n[470:569,]	Splits the full dataset into two datasets (1-469 for training and the remaining 100 for testing)
wbcd_train_labels <- wbcd[1:469, 1]	Extracts the diagnosis label from the previous data (for use later)
library(class)	Useful package for k-NN
wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test, cl = wbcd_train_labels, k=21)	Runs a basic kNN algo to the data. Trains and tests on the relevant inputs.
CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)	Method of checking how well the predicted values match up with known values:

```
                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        77 |         0 |        77 |
                 |     1.000 |     0.000 |     0.770 |
                 |     0.975 |     0.000 |           |
                 |     0.770 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         2 |        21 |        23 |
                 |     0.087 |     0.913 |     0.230 |
                 |     0.025 |     1.000 |           |
                 |     0.020 |     0.210 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        79 |        21 |       100 |
                 |     0.790 |     0.210 |           |
-----------------|-----------|-----------|-----------|
```

wbcd_z <- as.data.frame(scale(wbcd[-1]))	Z-normalises the data using the built in scale function

2. Naïve Bayes (Classification for probabilistic learning)

library(tm)	A useful text mining package for NB
library(wordcloud)	A useful text frequency visualising tool
library(e1071)	A nice package from Vienna University with ML (Naïve Baines & others)
library(gmodels)	Useful for evaluating model perf (crosstable function)
sms_corpus <- Corpus(VectorSource(sms_raw$text))	Two functions used here. First Corpus creates an R object to store text documents with a parameter specifying the format of text documents to be loaded. Here we use VectorSource() which tells corpus to use the messages in the vector sms_train$text.
print(sms_corpus)	Used to show information about a corpus
inspect(sms_corpus[1:3])	Used to view the first, second and third SMS messages
corpus_clean <- tm_map(sms_corpus, tolower) corpus_clean <- tm_map(corpus_clean, removeNumbers)	tm_map() is used to transform or map a corpus - here converts all to lowercase & removes numbers
corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())	Removes useless stop words and replaces them by a space
corpus_clean <- tm_map(corpus_clean, removePunctuation)	Removes all punctuation and replaces them by a space
corpus_clean <- tm_map(corpus_clean, stripWhitespace)	Removes all excess space thus seperates words by a single space
sms_dtm <- DocumentTermMatrix(corpus_clean)	This will tokenize the corpus and return the sparse matrix with the name sms_dtm.
sms_raw_train <- sms_raw[1:4169,] sms_raw_test <- sms_raw[4170:5559,]	Splits the raw data frame for training and testing
sms_dtm_train <- sms_dtm[1:4169,] sms_dtm_test <- sms_dtm[4170:5559,]	Splits the Document Term Matrix
sms_corpus_train <- corpus_clean[1:4169] sms_corpus_test <- corpus_clean[4170:5559]	Splits the Corpus
prop.table(table(sms_raw_train$type)) prop.table(table(sms_raw_test$type))	Checks the proportion of spam in the training and test frames are similar (ie subsets are representative)

`wordcloud(sms_corpus_train, min.freq = 40, random.order = FALSE)`	Visual representation of word frequency within a tm corpus object. Set min.freq to 10% of number of documents in the corpus. 
`spam <- subset(sms_raw_train, type == "spam")`	Creates a subset of SMS messages of spam type
`ham <- subset(sms_raw_train, type == "ham")`	Creates a subset of SMS messages of ham type
`wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))`	Creates a visual representation with the 40 most used words. 
`wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))`	Creates a visual representation with the 40 most used words. 
`findFreqTerms(sms_dtm_train, 5)`	Finds all terms that are mentionned at least 5 times
`sms_freq_words <- findFreqTerms(sms_dtm_train, 5)`	Saves frequent words into an object for later use
`sms_dtm_freq_train <- sms_dtm_train[ , sms_freq_words]`	Prepares training dataset
`sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]`	Prepares testing dataset

`convert_counts <- function(x) {`
`x <- ifelse(x > 0, "Yes", "No")`
`}`
	Custom function that converts counts to Yes/No strings.

`apply()`	Works like lapply() but instead takes 3 inputs: the dataset to apply the function to, either a row (MARGIN=1) or a column (MARGIN=2) and the function to be applied.
`sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)`	Converts sms_dtm_freq_train to Yes/No
`sms_test <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)`	Converts sms_dtm_freq_test to Yes/No

`sms_train_labels <- sms_raw[1:4169, ]$type` `sms_test_labels <- sms_raw[4170:5559, ]$type`	Creates the par of vectors with labels for each of the rows in the training and testing matrices.

`sms_classifier <- naiveBayes(sms_train, sms_train_labels)`
 	
`sms_test_pred <- predict(sms_classifier, sms_test)`
 	
`CrossTable(sms_test_pred, sms_test_labels, prop.chisq = FALSE, prop.t = FALSE, dnn = c('predicted', 'actual'))`
	Runs the naiveBayes algorithm to train the machine
	
	Makes a prediction based on training
	
	Evaluates model performance

`sms_classifier2 <- naiveBayes(sms_train, sms_train_labels, laplace = 1)`
 	
`sms_test_pred2 <- predict(sms_classifier2, sms_test)`
 	
`CrossTable(sms_test_pred2, sms_test_labels, prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE, dnn = c('predicted', 'actual'))`
	Better method of using naiveBayes by adding a Laplace estimator to avoid every message with the word ringtone being interpretted as spam.
	
	Model evaluation

3. Decision Trees

set.seed(123) train_sample <- sample(1000, 900)	Use to pick 900 random samples from 1000 ordered sample data
credit_train <- credit[train_sample,] credit_test <- credit[- train_sample,]	Splits dataset into training and testing objects
library(C50)	Useful divide & conquer algo package
credit_train$default<- as.factor(credit_train$default)	Used to convert non-factor stuff to factor
credit_model <- C5.0(credit_train[- 17], credit_train$default)	Turning an R object into a C5.0 decision tree with the following info: Call: C5.0.default(x = credit_train[-17], y = credit_train$default) Classification Tree Number of samples: 900 Number of predictors: 20 Tree size: 54 Non-standard options: attempt to gr oup attributes
summary(credit_model)	Contains the following: checking_balance in {> 200 DM,unknown}: 1 (412/50) checking_balance in {< 0 DM,1 - 200 DM}: :...other_debtors = guarantor: :...months_loan_duration > 36: 2 (4/1) … *= If checking account balance is unknown or greater than 200 DM, then classify as "not likely to default."* *Otherwise, if the checking account balance is less than zero DM or between* *one and 200 DM.* *And …*

	<pre>Evaluation on training data (900 cases)
	:
	
	            Decision Tree
	          ---------------- 	
	          Size Errors
	
	            54 135(15.0%) <<
	
	           (a) (b) <-classified a
	s
	          ---- ----
	           589 44 (a): class 1
	            91 176 (b): class 2</pre>
	Output indicates an error rate of 15%. 44 false positives, 91 false negatives.

`credit_pred <- predict(credit_model, credit_test)`	Predicts the future decisions based on model (used to evaluate model performance).

`CrossTable(credit_test$default, credit_pred, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted default'))`	<pre> Cell Contents
	|-------------------------|
	| N |
	| N / Table Total |
	|-------------------------|
	
	Total Observations in Table: 100
	
	               | predicted default
	actual default | 1 | 2 | Row Total |
	---------------|-----------|-----------|-----------|
	             1 | 60 | 7 | 67 |
	               | 0.600 | 0.070 | |
	---------------|-----------|-----------|-----------|
	             2 | 19 | 14 | 33 |
	               | 0.190 | 0.140 | |
	---------------|-----------|-----------|-----------|
	  Column Total | 79 | 21 | 100 |
	---------------|-----------|-----------|-----------|</pre>
	Correctly predicted 59 no default and 14 default out of 100 thus 73% accuracy.

`credit_boost10 <- C5.0(credit_train[-17], credit_train$default, trials = 10)`	Boosting model performance by using multiple weak performaning learners with different strengths together. (always start with 10 trials)

`> summary(credit_boost10)`
`> credit_boost_pred10 <- predict(credit_boost10, credit_test)`
`> CrossTable(credit_test$default, credit_boost_pred10, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted default'))`
	Used to show algorithm improved its performance by having only 29 mistakes on 900 training examples however its performance only went up to 76%.
	
	The lack of an even greater improvement may be a function of our relatively small training dataset, or it may just be a very difficult problem to solve.

> matrix_dimensions <- list(c("no", "yes"), c("no", "yes")) > names(matrix_dimensions) <- c("predicted", "actual")	Creates a cost matrix. Predicted and actual values take two values yes or no so 2x2 matrix composed of two vectors each with two values. Also important to name matrix dimensions to avoid later confusion. matrix_dimensions now holds: $predicted [1] "no" "yes" $actual [1] "no" "yes"

error_cost <- matrix(c(0, 1, 4, 0), nrow = 2, dimnames = matrix_dimensions)	Creates the following matrix :
	        actual
	predicted no yes
	      no 0 4
	      yes 1 0

> credit_cost <- C5.0(credit_train[-17], credit_train$default, costs = error_cost) > credit_cost_pred <- predict(credit_cost, credit_test) > CrossTable(credit_test$default, credit_cost_pred, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual default', 'predicted default'))	Same algorithm call but with a cost function added as a parameter. This artifically filters out damaging false positives. Does not necessarily lead to better performance, this is a trade off.
mushrooms <- read.csv("mushrooms.csv", stringsAsFactors = TRUE)	Import the data into mushrooms object
mushrooms$veil_type <- NULL	Since veil type is always the same value for all samples it cannot be used for prediction thus it must be dropped as shown.
mushroom_1R <- OneR(type ~ ., data = mushrooms)	Creates rules using 1R algo
> mushroom_1R > summary(mushroom_1R)	Reveals accuracy of ~99%
mushroom_JRip <- JRip(type ~ ., data = mushrooms)	Creates rules using RIPPER algo
> mushroom_JRip	Following rules: JRIP rules: =========== (odor = f) => type=p (2160.0/0.0) (gill_size = n) and (gill_color = b) => type=p (1152.0/0.0) (gill_size = n) and (odor = p) => type=p (256.0/0.0) (odor = c) => type=p (192.0/0.0) (spore_print_color = r) => type=p (72.0/0.0) (stalk_surface_below_ring = y) and (stalk_surface_above_ring = k) => type=p (68.0/0.0) (habitat = l) and (cap_color = w) => type=p (8.0/0.0) (stalk_color_above_ring = y) => type=p (8.0/0.0) => type=e (4208.0/0.0) Number of Rules : 9

4. Regression Methods (forecasting numeric data)

`a <- y_bar - b*x_bar`	Calculation for linear regression via Ordinary Least Squares (OLS) a?

`b <- cov(launch$temperature,`
`launch$distress_ct) /`
`var(launch$temperature)`
	Calculation for linear regression OLS b.

`a <- mean(launch$distress_ct) - b *`
`mean(launch$temperature)`
`> a`
	Estimation for linear regression OLS a.

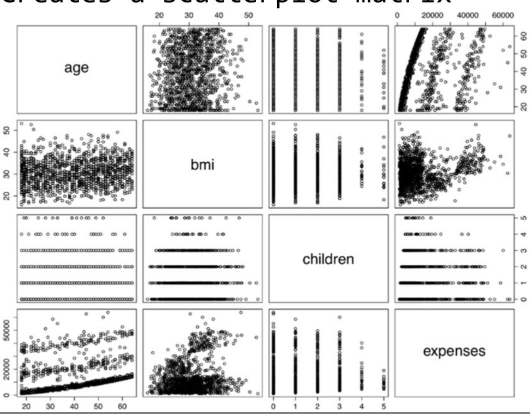
`r <- cov(launch$temperature,`
`launch$distress_ct) /`
`(sd(launch$temperature) *`
`sd(launch$distress_ct))`
	Explicit calculation in R for Pearson’s correlation

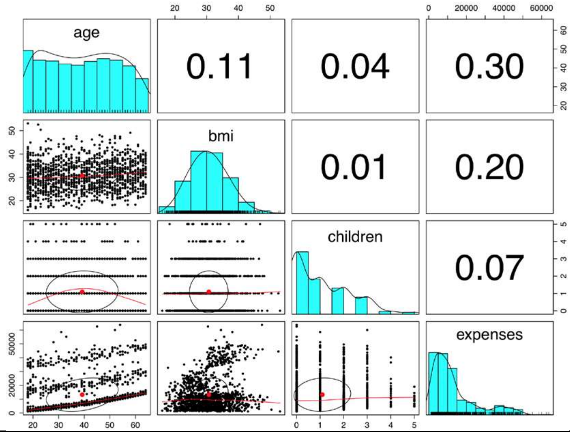
`cor(launch$temperature,` `launch$distress_ct)`	Simplest way to calculate Pearson’s correlation

`reg <- function(y, x) {`
`x <- as.matrix(x)`
`x <- cbind(Intercept = 1, x)`
`b<-solve(t(x) %*% x) %*% t(x) %*% y`
`colnames(b) <- "estimate"`
`print(b)`
`}`
	Simple multivariable linear regression function creation.
	- as.matrix() function is used to convert the data frame into matrix form
	- cbind() function is used to bind an additional column onto the x matrix
	- Intercept = 1 instructs R to name the new column Intercept and to fill the column with repeating 1 values.
	- solve() takes the inverse of a matrix
	- t() is used to transpose a matrix
	- %*% multiplies two matrices

`reg(y = launch$distress_ct, x =` `launch[2])`	Should do the univariate simple regression for us as detailed above.
`summary(insurance$charges)` `hist(insurance$charges)`	Check for normality

`cor(insurance[c("age", "bmi",`
`"children", "expenses")])`
	Check for independancy/corrolation
	`age bmi children charges`
	`age 1.0000000 0.1092719 0.04246900 0.29900819`
	`bmi 0.1092719 1.0000000 0.01275890 0.19834097`
	`children 0.0424690 0.0127589 1.00000000 0.06799823`
	`charges 0.2990082 0.1983410 0.06799823 1.00000000`

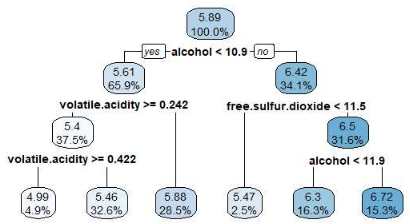
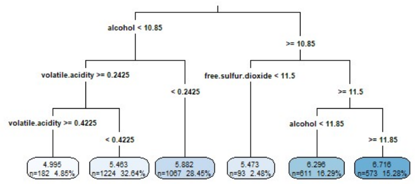
`pairs(insurance[c("age", "bmi",` `"children", "charges")])`	Creates a scatterplot matrix 

<table>
<tr><th>library(psych)</th><th>Useful R package for SCPLOM</th></tr>
<tr><td>pairs.panels(insurance[c("age", "bmi", "children", "expenses")])</td><td>Produces an enhanced scatterplot matrix (SCPLOM) with corrolation matrix, sample distribution, scatterplot, correlation ellipse and loess curve.

</td></tr>
<tr><td>ins_model <- lm(charges ~ age + children + bmi + sex + smoker + region, data = insurance)</td><td>Fits a linear regression model relating six independent variables to the medical charges. Result:
Coefficients:
(Intercept) age children bmi
-11938.5 256.9 475.5 339.2
sexmale smokeryes regionnorthwest regionsoutheast
-131.3 23848.5 -353.0 -1035.0
regionsouthwest
-960.1
Intercept is predicted values when independent variables are 0. Often ignored as has no real world meaning.
The beta coefficients indicate the estimated increase in expenses for an increase of
one in each of the features, assuming all other values are held constant.
Notice dummy coding was used to create dummy variables for categorical features.</td></tr>
<tr><td>summary(ins_model)</td><td>Used to evaluate model performance
Residuals:
Min 1Q Median 3Q Max
-11304.9 -2848.1 -982.1 1393.9 29992.8

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -11938.5 987.8 -12.086 < 2e-16 ***
age 256.9 11.9 21.587 < 2e-16 ***
children 475.5 137.8 3.451 0.000577 ***
bmi 339.2 28.6 11.860 < 2e-16 ***
sexmale -131.3 332.9 -0.394 0.693348
smokeryes 23848.5 413.1 57.723 < 2e-16 ***
regionnorthwest -353.0 476.3 -0.741 0.458769
regionsoutheast -1035.0 478.7 -2.162 0.030782 *
regionsouthwest -960.0 477.9 -2.009 0.044765 *

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

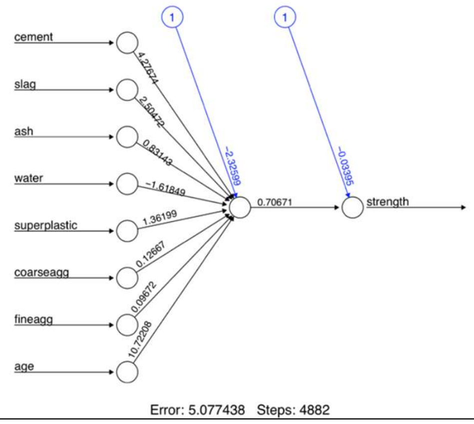
Residual standard error: 6062 on 1329 degrees of freedom
Multiple R-squared: 0.7509, Adjusted R-squared: 0.7494
F-statistic: 500.8 on 8 and 1329 DF, p-value: < 2.2e-16</td></tr>
<tr><td>insurance$age2 <- insurance$age^2</td><td>Adding non-linear age to the model</td></tr>
<tr><td>insurance$bmi30 <- ifelse(insurance$bmi >= 30, 1, 0)</td><td>Adding a threshold bmi to the model</td></tr>
<tr><td>charges ~ bmi*smoker</td><td>Adding interaction to the model</td></tr>
</table>

`ins_model2 <- lm(expenses ~ age + age2 + children + bmi + sex + bmi30*smoker + region, data = insurance)`	
`sdr_a <- sd(tee) - (length(at1) / length(tee) * sd(at1) + length(at2) / length(tee) * sd(at2))`	Calculating Standard Deviation Reduction for numeric decision trees
`wine_train <- wine[1:3750, ]` `wine_test <- wine[3751:4898, ]`	Dividing the dataset
`install.packages("rpart")`	Useful package for regression trees as described by the CART team
`m.rpart <- rpart(quality ~ ., data = wine_train)`	Sets the 'quality' as the outcome variable and allows all other columns in wine_train to be used as predictors
`library(rpart.plot)`	Useful library for visualising decision trees
`rpart.plot(m.rpart, digits = 3)`	
`rpart.plot(m.rpart, digits = 4, fallen.leaves = TRUE, type = 3, extra = 101)`	

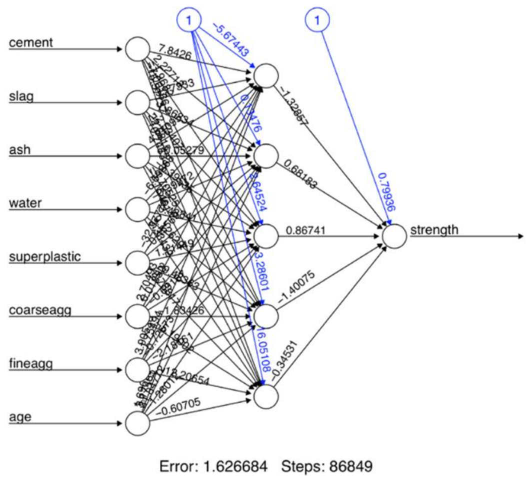
`p.rpart <- predict(m.rpart, wine_test)`
`summary(p.rpart)`
`summary(wine_test$quality)`
`cor(p.rpart, wine_test$quality)`
`MAE(p.rpart, wine_test$quality)`
	Evaluates model performance.

`MAE <- function(actual, predicted) { mean(abs(actual - predicted)) }`	Useful function for estimating mean absolute error
`m.m5p <- M5P(quality ~ ., data = wine_train)`	Improving the decision tree by using the M5Prime algorithm

5. Neural Networks & Support Vector Machines (Black Box Algos)

`concrete_norm <- as.data.frame(lapply(concrete, normalize))`	Normalise all the concrete data
`library(neuralnet)`	Simple yet quite powerful NN package (also could use nnet or RSNNS)
`concrete_model <- neuralnet(strength ~ cement + slag + ash + water + superplastic + coarseagg + fineagg + age, data = concrete_train)`	Trains the algorithm
`plot(concrete_model)`	Used to visualise network topology 

`model_results <- compute(concrete_model, concrete_test[1:8])`
 	
`predicted_strength <- model_results$net.result`
 	
`cor(predicted_strength, concrete_test$strength)`
	Evaluating model performance
	
	Slightly different than predict() as it returns two components, $neurons and $net.result
	
	Corrolation shows how connected the two numeric vectors are.

`concrete_model2 <- neuralnet(strength ~ cement + slag + ash + water + superplastic + coarseagg + fineagg + age, data = concrete_train, hidden = 5)`	Adding 5 hidden nodes to the NN 

<table>
<tr><td><pre>model_results2 <-
compute(concrete_model2,
concrete_test[1:8])

predicted_strength2 <-
model_results2$net.result

cor(predicted_strength2,
concrete_test$strength)</pre></td><td>Evalutating new model performance.</td></tr>
</table>

<table>
<tr><td><pre>letters_train <- letters[1:16000,]
letters_test <-
letters[16001:20000,]</pre></td><td>Split the testing and training data as usual</td></tr>
<tr><td><pre>library(e1071)
library(klaR)
library(kernlab)</pre></td><td>Recommended libraries for SVM algos (kernlab used below)</td></tr>
<tr><td><pre>letter_classifier <- ksvm(letter ~
., data = letters_train, kernel =
"vanilladot")</pre></td><td>Training the machine using linear kernal SVM</td></tr>
<tr><td><pre>letter_predictions <-
predict(letter_classifier,
letters_test)</pre></td><td>Using the trained machine to make predictions</td></tr>
<tr><td><pre>table(letter_predictions,
letters_test$letter)</pre></td><td><pre>letter_predictions   A   B   C   D   E   F   G   H   I
                 A 144   0   0   0   0   0   0   0   0
                 B   0 121   0   5   2   0   1   2   0
                 C   0   0 120   0   4   0 10   2   2
                 D   2   2   0 156   0   1   3 10   4
                 E   0   0   5   0 127   3   1   1   0
                 F   0   0   0   0   0 138   2   2   6
                 G   1   1   2   1   9   2 123   2   0
                 H   0   0   0   1   0   1   0 102   0
                 I   0   1   0   0   0   1   0   0 141</pre></td></tr>
<tr><td><pre>> agreement <- letter_predictions
== letters_test$letter
> table(agreement)
> prop.table(table(agreement))</pre></td><td>Shows tables of false and true predictions both numerically and as a percentage</td></tr>
<tr><td><pre>letter_classifier_rbf <-
ksvm(letter ~ ., data =
letters_train, kernel = "rbfdot")</pre></td><td>Training the machine using Gaussian RBF kernel SVM</td></tr>
<tr><td><pre>> letter_predictions_rbf <-
predict(letter_classifier_rbf,
letters_test)</pre></td><td>Using the trained machine to make predictions</td></tr>
<tr><td><pre>> agreement_rbf <-
letter_predictions_rbf ==
letters_test$letter
> table(agreement_rbf)
> prop.table(table(agreement_rbf))</pre></td><td>Evaluating model performance</td></tr>
</table>

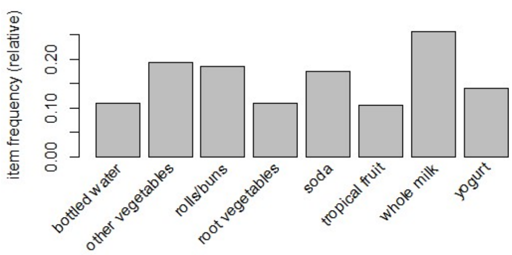
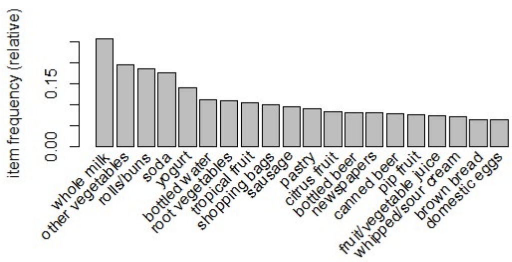
6. Apriori (Association Rules - Market Basket Analysis)

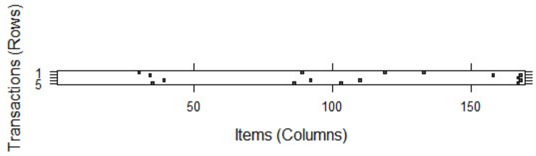
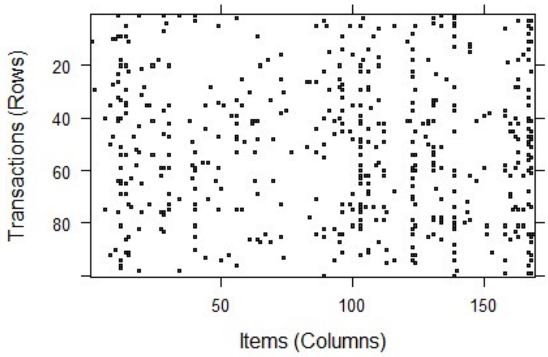
<table>
<tr><th>library(arules)</th><th>Package to make a sparse matrix from lists (import .csv not useful)</th></tr>
<tr><td>groceries <-
read.transactions("groceries.csv",
sep = ",")</td><td>Making a sparse matrix from a .csv</td></tr>
<tr><td>summary(groceries)</td><td><pre>
transactions as itemMatrix in sparse format with
 9835 rows (elements/itemsets/transactions) and
 169 columns (items) and a density of 0.02609146

most frequent items:
      whole milk other vegetables   rolls/buns   soda
            2513   1903   1809   1715
          yogurt   (Other)
            1372   34055

element (itemset/transaction) length distribution:
sizes
   1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16
2159 1643 1299 1005 855 645 545 438 350 246 182 117   78   77   55   46
  17   18   19   20   21   22   23   24   26   27   28   29   32
  29   14   14   9   11   4   6   1   1   1   1   3   1

   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
  1.000   2.000   3.000   4.409   6.000 32.000

includes extended item information - examples:
            labels
1 abrasive cleaner
2 artif. sweetener
3   baby cosmetics
</pre></td></tr>
<tr><td>inspect(groceries[1:5])</td><td>items
[1] {citrus fruit, margarine, ready soups, semi-finished bread}
[2] {coffee, tropical fruit, yogurt}
[3] {whole milk}
[4] {cream cheese, meat spreads, pip fruit, yogurt}
[5] {condensed milk, long life bakery product,other vegetables, whole milk}</td></tr>
<tr><td>itemFrequency(groceries[, 1:3])</td><td>Shows support level for the first three items in the grocery data (ordered alphabetically).</td></tr>
<tr><td>itemFrequencyPlot(groceries,
support = 0.1)</td><td>Histogram showing all items with a minimum support of 0.1 (as percent)

</td></tr>
<tr><td>itemFrequencyPlot(groceries, topN =
20)</td><td>Shows sorted histogram by decreasing support of top 20 items

</td></tr>
</table>

<table>
<tr><td><code>image(groceries[1:5])</code></td><td>Shows the entire sparse matrix for first 5 transactions/itemsets and 169 possible items.

</td></tr>
<tr><td><code>image(sample(groceries, 100))</code></td><td>Shows a sample of 100 random transactions

</td></tr>
<tr><td><code>apriori(groceries)</code></td><td>Runs the Apriori algorithm on the data.
Default support = 0.1, confidence = 0.8.</td></tr>
<tr><td><code>groceryrules <- apriori(groceries, parameter = list(support = 0.006, confidence = 0.25, minlen = 2))</code></td><td>Full command restricting support level, confidence and minimum length (avoid single items that are always bought). Stores rules found in object. Summary:
<pre>
rule length distribution (lhs + rhs):sizes
  2   3   4
150 297 16

   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
  2.000   2.000   3.000   2.711   3.000   4.000

summary of quality measures:
    support   confidence   lift
 Min.   :0.006101   Min.   :0.2500   Min.   :0.9932
 1st Qu.:0.007117   1st Qu.:0.2971   1st Qu.:1.6229
 Median :0.008744   Median :0.3554   Median :1.9332
 Mean   :0.011539   Mean   :0.3786   Mean   :2.0351
 3rd Qu.:0.012303   3rd Qu.:0.4495   3rd Qu.:2.3565
 Max.   :0.074835   Max.   :0.6600   Max.   :3.9565

mining info:
      data ntransactions support confidence
 groceries   9835   0.006   0.25
</pre></td></tr>
<tr><td><code>inspect(groceryrules[1:3])</code></td><td>Shows the first 3 rules.
<pre>
    lhs   rhs   support   confidence lift
[1] {pot plants} => {whole milk}   0.006914082 0.4000000 1.565460
[2] {pasta}   => {whole milk}   0.006100661 0.4054054 1.586614
[3] {herbs}   => {root vegetables} 0.007015760 0.4312500 3.956477
</pre></td></tr>
<tr><td><code>inspect(sort(groceryrules, by = "lift")[1:5])</code></td><td>Useful to reorder the rules found to find the best five rules according to the lift statistic (number of times more likely to purchase X given Y &VV).
<pre>
    lhs   rhs   support confidence   lift
[1] {herbs}   => {root vegetables}   0.007015760 0.4312500 3.956477
[2] {berries}   => {whipped/sour cream} 0.009049314 0.2721713 3.796886
[3] {other vegetables,
     tropical fruit,
     whole milk}   => {root vegetables}   0.007015760 0.4107143 3.768074
[4] {beef,
     other vegetables} => {root vegetables}   0.007930859 0.4020619 3.688692
[5] {other vegetables,
     tropical fruit}   => {pip fruit}   0.009456024 0.2634561 3.482649
</pre></td></tr>
</table>

<table>
<tr><td>berryrules <- subset(groceryrules, items %in% "berries")</td><td>Subset of rules for transactions including a specific product.

Items - keyword for items in rules

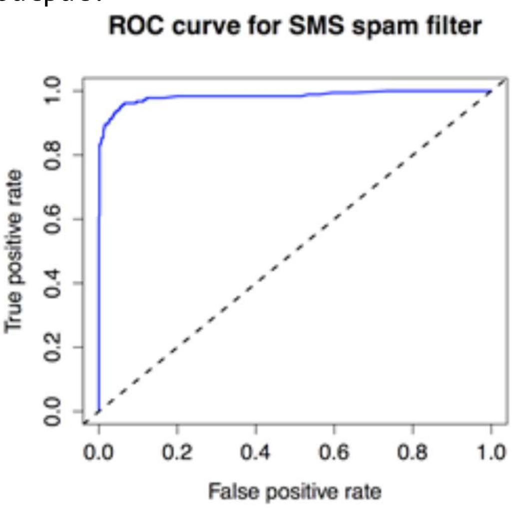
Subset is very powerful. Can also use partial matching (%pin%) and complete matching (%ain%). Can also be limitted by support, confidence or lift. Can be used with R’s logical operators (& | !)</td></tr>
<tr><td>inspect(berryrules)</td><td>Provides the rules found:<pre>   lhs   rhs   support   confidence lift
[1] {berries} => {whipped/sour cream} 0.009049314 0.2721713 3.796886
[2] {berries} => {yogurt}   0.010574479 0.3180428 2.279848
[3] {berries} => {other vegetables}   0.010269446 0.3088685 1.596280
[4] {berries} => {whole milk}   0.011794611 0.3547401 1.388328</pre></td></tr>
<tr><td>write(groceryrules, file = "groceryrules.csv", sep = ",", quote = TRUE, row.names = FALSE)</td><td>Publish rules found in a csv file.</td></tr>
<tr><td>groceryrules_df <- as(groceryrules, "data.frame")</td><td>Creates a data frame with the rules in the factor format, and numeric vectors for support, confidence, and lift</td></tr>
<tr><td>str(groceryrules_df)</td><td><pre>'data.frame':   463 obs. of 4 variables:
 $ rules   : Factor w/ 463 levels "{baking powder} => {other vegetables}",..: 3
40 302 207 206 208 341 402 21 139 140 ...
 $ support   : num 0.00691 0.0061 0.00702 0.00773 0.00773 ...
 $ confidence: num 0.4 0.405 0.431 0.475 0.475 ...
 $ lift   : num 1.57 1.59 3.96 2.45 1.86 ...</pre></td></tr>
</table>

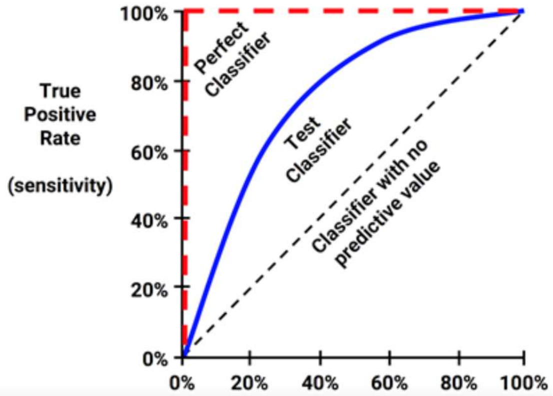
7. K-means (Clustering - Finding Groups of Data)

table(teens$gender, useNA = "ifany")	Check a dataset for missing data (factors)
summary(teens$age)	Check a dataset for missing data (numerical)
teens$age <- ifelse(teens$age >= 13 & teens$age < 20, teens$age, NA)	Omits those that lied about their age
teens$female <- ifelse(teens$gender == "F" & !is.na(teens$gender), 1, 0)	Dummy coding - replaces all those that are female with 0 else with 1. is.na() returns TRUE if the gender is equal to NA.
mean(teens$age, na.rm = TRUE) aggregate(data = teens, age ~ gradyear, mean, na.rm = TRUE)	Imputation - find the mean age of students in one class / each class and guess as to the true value of missing data
ave_age <- ave(teens$age, teens$gradyear, FUN = function(x) mean(x, na.rm = TRUE)) teens$age <- ifelse(is.na(teens$age), ave_age, teens$age)	A better way of returning data in the right format for imputation
library(stats)	Default very powerful package containing a kmeans algo
interests <- teens[5:40]	Makes a data frame containing only the features regarding interests (all but the top 4)
interests_z <- as.data.frame(lapply(interests, scale))	Z-standardises the interests data frame (since lapply() returns a matrix)
set.seed(2345)	Used to get the same as in example
teen_clusters <- kmeans(interests_z, 5)	Applies the kmeans algo and stores into an R object
teen_clusters$size	Returns the size of each cluster
teen_clusters$centers	Returns the coordinates of the 5 cluster centroids for the 36 interests
teens$cluster <- teen_clusters$cluster	Feedback the custer data into the teens object for evalutating model performance
teens[1:5, c("cluster", "gender", "age", "friends")] aggregate(data = teens, age ~ cluster, mean)	Shows what clusters the first 5 teens belong to Shows the average ages of each cluster (quite consistent here)
aggregate(data = teens, female ~ cluster, mean)	Shows the average gender of each cluster (highly predictive here) - particularly interesting as gender was not fed into the algo
aggregate(data = teens, friends ~ cluster, mean)	Shows how many friends each cluster typically has. Princess cluster has more friends even if not an input.

E) Evaluating model performance

`CrossTable(sms_results$actual_type, sms_results$predict_type)`	Evaluation using a confusion matrix
`library(caret)`	Classification and Regression Training package
`confusionMatrix(sms_results$predict_type, sms_results$actual_type, positive = "spam")`	Evaluation using caret
`library(vcd)`	Package for estimating Kappa using the Kappa() function
`Kappa(table(sms_results$actual_type, sms_results$predict_type))`	Outputs (use unweighted): value ASE Unweighted 0.8825203 0.01949315 Weighted 0.8825203 0.01949315
`library(irr)`	Package for estimating Kappa using the kappa2() function
`kappa2(sms_results[1:2])`	Calculates kappa from the vectors of predicted and actual values stored in a data frame
`sensitivity(sms_results$predict_type, sms_results$actual_type, positive = "spam")`	Caret function that calculates sensitivity
`specificity(sms_results$predict_type, sms_results$actual_type, negative = "ham")`	Caret function that calculates specificity
`posPredValue(sms_results$predict_type, sms_results$actual_type, positive = "spam")`	Caret function that calculates precision
`sensitivity(sms_results$predict_type, sms_results$actual_type, positive = "spam")`	Caret function that calculates recall (same as sensitivity)
`f <- (2 * prec * rec) / (prec + rec)`	F-measure function
`library(ROCR)`	Package for drawing Receiver Operating Characteristic curve (sensitivity/specificity plot)

`perf <- performance(pred, measure = "tpr", x.measure = "fpr")`
 	
`plot(perf, main = "ROC curve for SMS spam filter", col = "blue", lwd = 3)`
 	
`abline(a = 0, b = 1, lwd = 2, lty = 2)`
	Output:
	

	Ideally: 

`perf.auc <- performance(pred, measure = "auc")`
 	
`unlist(perf.auc@y.values)`
	Use both ROC and AUC to visualise model performance

`random_ids <- order(runif(1000))`	Create a random ID vector

`credit_train <- credit[random_ids[1:500], ]`
 	
`credit_validate <- credit[random_ids[501:750], ]`
 	
`credit_test <- credit[random_ids[751:1000], ]`
	Holdout sampling for evalutating model performance (use carefully if you only have a very small class aa it may be omitted - consider using stratified random sampling)
	
	After holdout is over, retrain final model on entire dataset

`in_train <- createDataPartition(credit$default, p = 0.75, list = FALSE)`
 	
`credit_train <- credit[in_train, ]`
 	
`credit_test <- credit[-in_train, ]`
	Stratified random sampling using caret package

`folds <- createFolds(credit$default, k = 10)`	Creating folds for 10-fold cross validation performance evalutation (industry standard)

`cv_results <- lapply(folds, function(x) {`
`credit_train <- credit[-x, ]`
`credit_test <- credit[x, ]`
`credit_model <- C5.0(default ~ ., data = credit_train)`
`credit_pred <- predict(credit_model, credit_test)`
`credit_actual <- credit_test$default`
`kappa <- kappa2(data.frame(credit_actual, credit_pred))$value`
`return(kappa)`
`})`
	Function to undertake 10-fold cross validation performance evalutation (industry standard).
	
	Can use boostrapping over cross-validation for very small datasets.

F) Improving model performance

Model	Learning Task	Method name	Parameters
k-Nearest Neighbors	Classification	knn	k
Naive Bayes	Classification	nb	fL, usekernel
Decision Trees	Classification	C5.0	model, trials, winnow
OneR Rule Learner	Classification	OneR	None
RIPPER Rule Learner	Classification	JRip	NumOpt
Linear Regression	Regression	lm	None
Regression Trees	Regression	rpart	cp
Model Trees	Regression	M5	pruned, smoothed, rules
Neural Networks	Dual use	nnet	size, decay
Support Vector Machines (Linear Kernel)	Dual use	svmLinear	C
Support Vector Machines (Radial Basis Kernel)	Dual use	svmRadial	C, sigma
Random Forests	Dual use	rf	mtry

Caret summary of methods for previous algorithms used. This can be queried by prompting:

```
modelLookup("C5.0")
```

```
m <- train(default ~ ., data = credit, method = "C5.0")
```

Model improvement using caret

```
1  1000 samples
     16 predictor
      2 classes: 'no', 'yes'

2  No pre-processing
   Resampling: Bootstrapped (25 reps)

   Summary of sample sizes: 1000, 1000, 1000, 1000, 1000, 1000, ...

3  Resampling results across tuning parameters:

     model  winnow  trials  Accuracy   Kappa      Accuracy SD  Kappa SD
     rules  FALSE    1      0.6847204  0.2578421  0.02558775   0.05622302
     rules  FALSE   10      0.7112829  0.3094601  0.02087257   0.04585890
     rules  FALSE   20      0.7221976  0.3260145  0.01977334   0.04512083
     rules   TRUE    1      0.6888432  0.2549192  0.02683844   0.05695277
     rules   TRUE   10      0.7113716  0.3038075  0.01947701   0.04484956
     rules   TRUE   20      0.7233222  0.3266866  0.01843672   0.03714053
     tree   FALSE    1      0.6769653  0.2285102  0.03027647   0.07001131
     tree   FALSE   10      0.7222552  0.2880662  0.02061900   0.05601918
     tree   FALSE   20      0.7297858  0.3067404  0.02007556   0.05616826
     tree    TRUE    1      0.6771020  0.2219533  0.02703456   0.05955907
     tree    TRUE   10      0.7173312  0.2777136  0.01700633   0.04358591
     tree    TRUE   20      0.7285714  0.3058474  0.01497973   0.04145128

4  Accuracy was used to select the optimal model using  the largest value.
   The final values used for the model were trials = 20, model = tree
   and winnow = FALSE.
```

```
p <- predict(m, credit)
```

Use improved model to make prediction

```
table(p, credit$default)

head(predict(m, credit))

head(predict(m, credit, type = "prob"))
```

Evaluate performance of new improved model (less accurate than the output of the boostrap since test was done not using new data)

```
ctrl <- trainControl(method = "cv", number = 10, selectionFunction = "oneSE")
```

Creates a control object that uses 10-fold cross validation and the oneSE selection function

```
grid <- expand.grid(.model = "tree", .trials = c(1, 5, 10, 15, 20, 25, 30, 35), .winnow = "FALSE")
```

Creates the grid of parameters to optimise

```
m <- train(default ~ ., data = credit, method = "C5.0", metric = "Kappa", trControl = ctrl, tuneGrid = grid)
```

Results in the following object:

```
1000 samples
  16 predictor
   2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)

Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...

Resampling results across tuning parameters:

  trials  Accuracy  Kappa      Accuracy SD  Kappa SD
   1      0.724     0.3124461  0.02547330   0.05897140
   5      0.713     0.2921760  0.02110819   0.06018851
  10      0.719     0.2947271  0.03107339   0.06719720
  15      0.721     0.3009258  0.01969207   0.05105480
  20      0.717     0.2929875  0.02790858   0.07912362
  25      0.728     0.3150336  0.03224903   0.09367152
  30      0.729     0.3104144  0.02766867   0.08069045
  35      0.741     0.3389908  0.03142893   0.09352673

Tuning parameter 'model' was held constant at a value of tree
Tuning parameter 'winnow' was held constant at a value of FALSE
Kappa was used to select the optimal model using  the one SE rule.
The final values used for the model were trials = 1, model = tree
and winnow = FALSE.
```

library(ipred)	Useful R package for boostrap aggregating (bagging)
mybag <- bagging(default ~ ., data = credit, nbagg = 25) credit_pred <- predict(mybag, credit) table(credit_pred, credit$default)	Use the improved model to make a prediction
ctrl <- trainControl(method = "cv", number = 10) train(default ~ ., data = credit, method = "treebag", trControl = ctrl)	Evalutating future model performance of improved model
bagctrl <- bagControl(fit = svmBag$fit, predict = svmBag$pred, aggregate = svmBag$aggregate)	First, creates a bagging control object
svmbag <- train(default ~ ., data = credit, "bag", trControl = ctrl, bagControl = bagctrl)	Trains the improved bagged model, outputs: Bagged Model 1000 samples 16 predictors 2 classes: 'no', 'yes' No pre-processing Resampling: Cross-Validation (10 fold) Summary of sample sizes: 900, 900, 900, 900, 900, 900, ... Resampling results Accuracy Kappa Accuracy SD Kappa SD 0.728 0.2929505 0.04442222 0.1318101 Tuning parameter 'vars' was held constant at a value of 35
library(adabag)	Useful R package for adaptive boosting
m_adaboost <- boosting(default ~ ., data = credit) p_adaboost <- predict(m_adaboost, credit) head(p_adaboost$class) p_adaboost$confusion	Using the AdaBoost.M1 algorithm to make an adaptive boosted learner Departing from convention, rather than returning a vector of predictions, this returns an object with information about the model. The predictions are stored in a sub-object called class. Confusion matrix in subobject called confusion. (based on training data)
adaboost_cv <- boosting.cv(default ~ ., data = credit) adaboost_cv$confusion	More suitable performance evaluation

Kappa(adaboost_cv$confusion)	Finds kappa statistic using the vcd package
library(randomForest)	Most reliable R package, caret compliant, for random forest algos
rf <- randomForest(default ~ ., data = credit)	Training the rf learner
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)	Sets the control
grid_rf <- expand.grid(.mtry = c(2, 4, 8, 16))	Sets the tuning grid
m_rf <- train(default ~ ., data = credit, method = "rf", metric = "Kappa", trControl = ctrl, tuneGrid = grid_rf)	Use the kappa metric to select the best model

G) Specialised ML topics

library(rio)	Most reliable R IO package
credit <- import("credit.csv")	Import .csv file
export(credit, "credit.xlsx")	Export .xlsx file
convert("credit.csv", "credit.dta")	Converts from .csv to .dta

library(RODBC)	For access to DBMSes in R
my_db <- odbcConnect("my_dsn") *or* my_db <- odbcConnect("my_dsn", uid = "my_username", pwd = "my_password")	Access DB
my_query <- "select * from my_table where my_value = 1" results_df <- sqlQuery(channel = my_db, query = sql_query, stringsAsFactors = FALSE)	Query DBs (typical SQL)
odbcClose(my_db)	Close the DB access

mydata <- read.csv("http://www.mysite.com/mydata.csv")	Read .csv from website
mytext <- readLines("http://www.mysite.com/myfile.txt")	Read .txt from website
download.file("http://www.mysite.com/myfile.zip", "myfile.zip")	Download any file for reading

library(RCurl)	For access to web source
packt_page <- ("https://www.packtpub.com/")	Save webpage's source html
str(packt_page, nchar.max=200)	Access the first 200 chars of a file

library(httr)	For access to web source (better)
packt_page <- GET("https://www.packtpub.com")	Save webpage's source html + site properties (useful in web API JSON)
str(packt_page, max.level = 1)	Read webpage site properties
str(content(packt_page, type="text"), nchar.max=200)	Read webpage html
map_search <- GET("https://maps.googleapis.com/maps/api/geocode/json", query = list(address = "Eiffel Tower"))	Save the JSON output of an API request into an R object
content(map_search)	Access the content of the resulting JSON
content(map_search)$results[[1]]$formatted_address	Access more specific content of the resulting JSON

library(rvest)	Web scraping package
packt_page <- html("https://www.packtpub.com")	Save webpage source html + site properties (calls GET())
html_node(packt_page, "title")	Scrapes the content between <title> and </title> tags
html_node(packt_page, "title") %>% html_text()	Scrapes the content between <title> and </title> tags and converts to text
ml_packages <- html_nodes(cran_ml, "a")	Webscrapes all the a objects from the page into an R vector

library(libxml2)	XML reading package
Similar to html, refer to documentation	

library(rjson)	JSON conversions from web APIs
ml_book <- list(book_title = "Machine Learning with R", author = "Brett Lantz") toJSON(ml_book)	Converts R object to JSON

ml_book_json <- "{
\"title\": \"Machine Learning with
R\",
\"author\": \"Brett Lantz\",
\"publisher\": {
\"name\": \"Packt Publishing\",
\"url\":
\"https://www.packtpub.com\"
},
\"topics\": [\"R\", \"machine
learning\", \"data mining\"],
\"MSRP\": 54.99
}"
 	
ml_book_r <- fromJSON(ml_book_json)
	Converts JSON string into R object

library(network)	Bioinformatics package for specialised network data structure
library(sna)	Bioinformatics package for social network analysis
library(igraph)	Bioinformatics package for visualising network data
Refer to documentation	

library(dplyr)	Generalising tabular data structures
Refer to documentation	

library(data.table)	Making data frames faster
Refer to documentation	

library(ffdf)	Making data frames larger (disk-based)
Refer to documentation	

library(bigmemory)	Making big matrices
Refer to documentation	

library(biglm)	Building bigger regression models
Refer to documentation	

library(bigrf)	Building bigger random forests
Refer to documentation	

library(parallel)	Parallel computing
library(multicore)	Multicore CPU usage
library(snow)	Distributed parallel computing
library(RHIPE)	Cloud computing
Library(gputools)	CUDA/GPU computing
Refer to documentation	