R Convention Guide

The rules presented in this document guide the .R code in this repository

“Programs must be written for people to read, and only incidentally for machines to execute.”

-H. Abelson, Structure and Interpretation of Computer Programs

Introduction

- What are **coding conventions**?
  - “a set of **guidelines** for a **specific programming language** that controls issues such as :
    - how to name variables and functions
    - how the source code is indented/laid out
    - how and when to add comments
    - where (inside the source code) to declare variables and libraries
    - how to write statements
    - how files containing source code are organized inside a directory
    - when to use white space
    - which programming practices and principles to use

    …etc.”

Introduction

- In essence it tells you which rules to follow to write your script (program)
  - So, I can’t give my own names to variables? You can, but names must follow rules. These rules are specified by coding conventions.
- Why use coding conventions and style?
  - improves the **readability** of their source code and make software **maintenance easier**.
  - Makes **maintenance** and **evolution** of the script easier
  - Coding conventions especially important when **code** is **shared** between **team members**.
  - Scripts and programs almost **never fully supported by its original author**; others take over and evolve it.

Introduction

- Example of the role of coding conventions
  - Both R scripts (programs) attempt to do exactly the same thing in the same way. Both execute and correctly calculate the desired result. One is not like the other.

Bad coding conventions

Don't ✗

```
d<-c(22,26,21,25)
x<-mean(d)
```

Difficult to see what data each variable holds and its role in the program. Difficult to understand, by humans, what this simple program is calculating and why. Difficult to read and understand because of the way variables are named: this is due to the absence of coding conventions/principles. This script will be difficult to modify and evolve.

Good coding conventions

Do ✓

```
studentAges <- c(22, 26, 21, 25)
averageStudentAge <- mean(studentAges)
```

Easy to see what data each variable holds and its role in the program. Easy to understand, by humans, what this simple program is calculating. Easy to read and understand simply because of the way the variables were named: this is due to adhering to coding conventions/principles. This script will be easy to modify and evolve.

Introduction

- Simple (more complicated) example on the role of coding conventions
  - Both scripts (programs) attempt to do exactly the same thing in the same way. Both execute and correctly calculate the desired result. (Hint: Compare readability!)

Don't

```
md <- read.csv("foodConsumption.csv", header=TRUE)
x <- sum(md[,"FoodExpenditure"]) / nrow(md)
y <- sqrt(sum((md[,"FoodExpenditure"] - x)^2) / (nrow(md)-1) )
```

Do

```
householdConsumption <- read.csv("ConsumptionData.csv", header=TRUE)

# We'll calculate manually the sample standard deviation of FoodExpenditure.
# This is done for testing purposes only.
averageFoodConsumption <- sum(householdConsumption[ , "FoodExpenditure"]) / nrow(householdConsumption)
sampleStandardDeviation <- sqrt(sum((householdConsumption[, "FoodExpenditure"] - averageFoodConsumption)^2) / (nrow(householdConsumption)-1) )
```

Important notice!

The coding conventions presented here are only suggestions.

Use any coding convention that is useful for you and helps you in your project (No need to use all of them).

The important thing is: ADOPT ANY CODING CONVENTION and STYLE. AND ONCE ADOPTED, USE IT CONSISTENTLY.

STICK TO IT!

Table of Contents

Naming conventions

Naming conventions

- Naming conventions refer to guidelines/rules related to how files containing R code, variables, functions etc in the source code should be named.

File names in the same project containing R source code files should be meaningful and have the exact same extension (either .r or .R – not mixed!)*

- Examples

Don't ✗

```
Calculate_Average.r
Dickey_fuller_test.R

avrg.r
DFTfubar.r
test.R
DT.R
```

Do ✓

```
Calculate_Average.r
Dickey_fuller_test.r

CalculateAverage.R
Dickey_fuller_test.R
testDataNormality.R
DecisionTree.R
```

* The term "project" is used to refer to a set files containing R source code that are written in the context of the problem solution. In the simplest case, a project consists of a single file containing R code.

Define and use naming conventions (i.e. naming rules) for files, functions and variables and stick to them

- **Adopt** and **follow consistently** conventions for the following:
  - File names containing R source code: e.g. underscores and ending in either .r or .R
  - User defined functions: e.g. big camel case (=first letter capital). For function names, use a description of the value(s) returned by the function
  - Variables: e.g. small camel case (=first letter small)
  - Constants: e.g. small camel case with first letter k, all letters capital

Do

```
Dickey_fuller_test.R

GradientDescentCoefficients <- function(dependent, independent, alpha){
…
}

averageStudentAge <- 12.3
kDataFileName <- “bankData.csv”
MINIMUM_NUMBER_OF_CLUSTERS <- 5
```

Names of files, variables, functions etc should be meaningful

- Meaningful? names should indicate what data they hold (for variables), what calculation is done (for functions) and what their role is in the program.
  - Not always easy or possible

Don't ✗

```
sa <- c(31, 22, 25)
myData <- read_csv('customers.csv', …)

DoSomethingWithNumbers <- function(n1, n2){
…
}
```

Do ✓

```
studentAges <- c(31, 22, 25)
bankCustomerData <- read_csv('customers.csv', …)

GretestCommonDivisor <- function(n1, n2){
…
}
```

Code

R starts counting from 1. Not 0

- When counting or indexing, R starts from 1, not 0. This is in contrast to the majority of programming languages like Python, C, Python etc

Don't ✗

```
# Household income of first row in data frame
householdDataFrame[0, "Income"]
```

Do ✓

```
# Household of first row in data frame
householdDataFrame[1, "Income"]
```

For variable assignment use <- not =

- Two operators are supported in R: **<-** and **=**
  - For variable assignment, always use **<-**
  - Use **=** only for default parameters in functions and named parameter passing

Don't ✕

```
name = "Tony Montana"
b = 42

someFunction <- function(p1<-42, p2<-""){
…
}
someFunction(p1<- -9, p2<- "Jim")
```

Do ✓

```
name <- "Tony Montana"
b <- 42
someFunction <- function(p1=42, p2=""){
…
}
someFunction(p1=-9, p2="Jim")
```

Inside .R files, lines should be less than 80 characters long. If a line is more than 80 characters long, add line breaks (at allowed positions e.g. commas, assignments, operators etc) to make lines smaller. Align lines to make code easier to read.

Don't ✗

```
grCensus2011[ which( (grCensus2011[,1]==5) & (grCensus2011[,42] == max(grCensus2011[ which(grCensus2011[,1]==5),42], na.rm=TRUE)) ), 3]
```

Do ✓

```
grCensus2011[ which( (grCensus2011[, 1] == 5)
                     &
                     (grCensus2011[, 42] == max(grCensus2011[ which(grCensus2011[, 1]==5), 42],
                                                na.rm=TRUE)
                     )
                   ),
              3]
```

Define functions when critical code repeats in a script. This allows reusing code and make modifications easier/saver

- Functions are a way to reuse code within .R scripts without repeating code that can be error prone. Functions facilitate also code changes.

Don't ✗

```
greekBanks <- read_csv('banks.csv', header=TRUE)
# Remove banks located in Patras
greekBanks <- greekBanks[ -which(greekBanks[ , "location"] == "Patras"), ]
# Remove banks that are investmemnt banks
greekBanks <- greekBanks[ -which(greekBanks[ , "type"] == "Investment"), ]
```

This does an exact match for the values "Patras" and "investment". If criteria change (e.g. contain "Patras" or "investment") **all** row removal lines must change. If one is overlooked, this may cause serious problems.

Do ✓

```
ExcludeBanks <- function(bankDataFrame, column, excludedValue){
            return( bankDataFrame[ -which(bankDataFrame[, column] == excludedValue), ] )
}

greekBanks <- read_csv('banks.csv', header=TRUE)
greeBanks <- ExcludeBanks(greeBanks, 'location', 'Patras')
greeBanks <- ExcludeBanks(greeBanks, 'type', 'Investment')
```

Since row removal based on criteria are repeated, a function is defined that removes lines meeting criteria. If the removal criteria change, only ONE removal line must be changed (inside the function) making the modification saver.

Calling the function for removing rows based on criteria.

Functions should do only one thing and do it well

Don't ✗

```
AddVectors <- function(v1, v2){
     icecreamSales <- read_csv('d.csv')
     average <- mean( icecreamSales[, "sales"])
     print(average)
     return(v1 + v2)
}
```

This function does more than just adding vectors. Does things irrelevant to adding vectors..

Do ✓

```
AddVectors <- function(v1, v2){
     return(v1 + v2)
}

AverageIcecreamSales<-function(icecreamCSVFile){
     iceSales <- read_csv(icecreamCSVFile)
     return(mean(iceSales[,"Sales"])
}
```

Returns from functions should always be explicit

Don't ✗

```
AddVectors <- function(v1, v2){
        v1 + v2 # Value will be returned from function
}
```

Do ✓

```
AddVectors <- function(v1, v2){
     return(v1 + v2) # Explicit return
}
```

An opening curly bracket { should not be in a separate line and should always be followed by a new line

Don't ✗

```
DotProduct <- function(v1, v2)
{
    return(v1 %*% v2)
}

If (a>b){print("a greater than b")}
```

Do ✓

```
DotProduct <- function(v1, v2){
     return(v1 %*% v2)
}

if (a > b){
    print("a greater than b")
}
```

Place spaces around all infix operators such as <- , ==, +, -, =, >, <, <=, >=, etc.

Don't ✗

```
if (x+3>0){
    print("Greater than 0")
}

result<-v1%*%v2
b<--42
```

Do ✓

```
if (x + 3 > 0){
    print("Greater than 0")
}

result <- v1 %*% v2
b <- -42
```

Don’t place spaces in parenthesized expressions

- Example

Don’t ✗

```
if ( mustStop ){
    print( “Stopping…” )
}

result<- ( v1%*%v2 ) - 6
ExcludeBanks( greeBanks, ‘type’, ‘Investment’ )
```

Do ✓

```
if (mustStop){
    print(“Stopping…”)
}

result <- (v1 %*% v2) - 6
ExcludeBanks(greekBanks, ‘type’, ‘Investment’)
```

Always indent your code. Adopt and use consistent indentation (e.g. two/three spaces for new blocks). Never use tabs or mix tabs and spaces.

Don't ✗

```
while (i < 6){
       print(data[i, 3])

           # increase by one
           i = i + 1
}



SomeFunction <- function(v1=-1,
                    v2="Joe",
                              v3=NULL) {
     print("Starting execution of SomeFunction")
}
```

Do ✓

```
while (i < 6){
   print(data[i, 3])

   # Move to next row
   i = i + 1
}

# When parameter list is large
SomeFunction <- function(v1=-1,
                              v2="Joe",
                              v3=NULL){
   print("Starting execution of SomeFunction")
}
```

Start every script with setting up properly the R execution environment.

- Start your script by explicitly:
  - Cleaning workspace environment using rm(list=ls()) from previous runs
  - Setting the script's working directory with setwd() if needed.

Don't ✗

```
################################################
# Application starts from here.
# Loads and preprocesses data. Then K-means
# is executed.
#
#  v0.3 abc@upatras.gr – Dec 2019
################################################


library(dplyr)
rm(list=ls())
library(ggplot2)
library(rpart)
setwd("C:\\users\Alan\\RProjects")
```

Do ✓

```
################################################
# Application starts from here.
# Loads and preprocesses data. Then K-means
# is executed.
#
#  v0.3 abc@upatras.gr – Dec 2019
################################################

# Cleanup environment/workspace and setup
rm(list=ls())
setwd("C:\\users\Alan\\RProjects")

library(dplyr)
library(ggplot2)
library(rpart)
```

Try to avoid loops in R. Use vectorized calculations whenever possible

- Prefer indexing, apply(), lapply(), sapply(), tapply(), filter, subset etc…

Don't ✗

```
# Print rows where Income is < than 800
for (i in 1:nrow(incomeDataFrame)){
     if (incomeDataFrame[,'income'] < 800)
          print(incomeDataFrame[i,])
}

# create vector initialized with numbers from 1 up to 15
x <- c()
for (i in 1:15){
     x[length(x) + 1] <- i
}
```

Do ✓

```
print(incomeDataFrame[incomeDataFrame[,'Income'] < 800, ])

x<-1:15
```

Comments

Comments should begin with # and a single space immediately after.

Don't

```
while (i < 6){
   print(data[i, 3])

   #Move to next row
   i = i + 1
}
```

Do

```
while (i < 6){
   print(data[i, 3])

   # Move to next row
   i = i + 1
}
```

Comments should say why something is done, not what is being done.

All can see this. The comment just repeats and describes the code.

Don't ✗

```
# Read data from the csv file into a Data Frame
greekBanks <- read_csv('banks.csv', header=TRUE)

# Exclude banks that have type equal to Investment
greeBanks <- ExcludeBanks(greeBanks, 'type', 'Investment')
```

Do ✓

```
# Load banks. These need to be preprocessed first
# before executing kmeans
greekBanks <- read_csv('banks.csv', header=TRUE)

# Exclude investment banks because they have many NAs.
# This breaks the analysis.
greeBanks <- ExcludeBanks(greeBanks, 'type', 'Investment')
```

90% of all code comments:

Start every .R file with a comment saying what it contains, who wrote it, its version, when it was created and how it fits into the larger program

Don't ✗

```
library(dplyr)
library(ggplot2)
library(rpart)
```

Do ✓

```
##################################################
# This file contains functions for calculating
# descriptive statics such as central tendency
# and measures of variability for various kinds
# of data (nominal, ordinal, ratio, interval).
#
# If a script needs to calculate descriptive
# statistics, it needs to include this R file
# using source().
#
#                  v0.1/nmark@upatras.gr/Jan 2022
##################################################

library(dplyr)
library(ggplot2)
library(rpart)
```

Use comments to further group/organize your code based on their role inside a .R source file

Don't ✗

```
greekBanks <- read_csv('bankdata.csv', header=T)

greekBanks <- na.omit(greekBanks)
greekBanks <- ExcludeBanks(greeBanks, 'type', 'Investment')
greekBanks['bankSize'] <- (greekBanks['bankSize'] – min(greekBanks['bankSize']) /
             (max(greekBanks['bankSize']) - min(greekBanks['bankSize'])


decisionTreeFraudModel <- rpart(Fraud ~ ., data = greekBanksTraining,
                                method = "class",
                                parms = list(split = 'information'))
```

Do ✓

```
greekBanks <- read_csv('bankdata.csv', header=T)

#############################################################################
# Preprocessing
#############################################################################
greekBanks <- na.omit(greekBanks)
greekBanks <- ExcludeBanks(greeBanks, 'type', 'Investment')
greekBanks['bankSize'] <- (greekBanks['bankSize'] – min(greekBanks['bankSize']) /
                          (max(greekBanks['bankSize']) - min(greekBanks['bankSize'])

#############################################################################
# Training of decision tree fraud model with entropy as splitting criterion
#############################################################################
decisionTreeFraudModel <- rpart(Fraud ~ ., data = greekBanksTraining,
                                method = "class",
                                parms = list(split = 'information'))
```

Comment before you write/finalize your code

Don't ✗

```
# Incomplete
RemoveFamiliesWithHighIncome(familiesDF, mxIncome){

}
```

Do ✓

```
#' INCOMPLETE-Removes households with high income.
#'
#  @details
#' Will remove outliers in terms of income. Since
#' OLS is used to estimate coefficients, outliers
#' will introduce a bias. Hence, get rid of them.
#' TODO: Is this really needed? Can we use another
#' method of estimating coefficients? Also, which method
#' to use to define outliers? Use fixed mxIncome or z-values?
#' @param familiesDF data frame with household income
#' @param mxIncome rows with higher income than mxIncome will be removed
RemoveFamiliesWithHighIncome(familiesDF, mxIncome){
}
```

Add comments for R functions. Use the roxygen2 conventions (see references section)

Don't ✗

```
# Removes rows from data frame that have high income
RemoveFamiliesWithHighIncome <- function(familiesDF, maxIncome){
   return(familiesDF[ familiesDF$Income < maxIncome], )
}
```

Do ✓

```
#' Removes households with higher income than a threshold.
#'
#  @details
#' Will remove outliers in terms of income. Since
#' OLS is used to estimate coefficients, outliers
#' will introduce a bias. Hence, get rid of them.
#' TODO: Is this really needed? Can we use another
#' method of estimating coefficients? Also, which method
#' to use to define outliers?
#' @param familiesDF data frame with household income
#' @param minIncome rows with higher or equal to income than minIncome are removed
RemoveFamiliesWithHighIncome <- function(familiesDF, maxIncome){
   return(familiesDF[familiesDF$Income < maxIncome, ])
}
```

File organization

Separate function/variable/constant definitions based on their purpose in different files. Use source() to include .R files where you need these functions/variables/constants

File: descriptive_statistics.R

Do ✓

```
##############################################################################################
#
# This file contains functions and constants for calculating descriptive statics such as
# central tendency and measures of variability for various kinds of data
# (nominal, ordinal, ratio, interval). If the program needs to calculate means,
# medians, modes, stdev, variances, minimum- maximum values, kurtosis, skewness it does it
# by calling a function from this file.
#
# v1.2 -  up123456@ac.upatras.gr – Nov 2022
#
##############################################################################################
```

File: main_file.R

Do ✓

```
# Include some necessary functions/variables
source('descriptive_statistics.R')
```

Doing this will make all functions, variables and constants defined in file descriptive_statistics.R available for use in file main_file.R . Allows functions/variables/constants defined in descriptive_statistics.R to be **reused across projects**.

Files containing R source code should be smaller than 3000 lines. If .R files are larger, break them up into smaller ones and use source() to include them where needed. Breakup is semantically guided.

File: main.R

Don't

```
################################################
#                                              #
#  Using the K-means  algorithm                #
#  to cluster iris species                     #
#                                              #
#  Will be using the iris dataset              #
#                                              #
################################################


# We want the irir dataset - it's available in R
data(iris)

# Initialize random number generator. We need this because we'll be be using the nstart
# parameter in the kmeans() function that randomly initializes the process
set.seed(20)


# Now the idea is to cluster the iris dataset - which contains variables about their sepal/petal length and width
# as well as their species - based on the following attributes: Petal.length and Petal.width .
# We set number of clusters K=3 as there are 3 species of iris: setosa, versicolor and virginica.
# The idea is to see if flowers of the same species will be put in the same cluster with the K-means algorithm
# so that we can check how good our clustering is.

# Before executing the K-means algorithm, we have to normalize the variables that will be used for
# clustering since K-means uses Euclidean distance which is sensible to big values. We use min-max normalization.

# Define our min-max normalization function
norm <- function(x){ return( (x-min(x)) / (max(x)-min(x)) ) }

# Apply min-max normalization to the clustering attributes
iris$Petal.Length <- norm(iris$Petal.Length)
iris$Petal.Width <- norm(iris$Petal.Width)

# Petal.Length and Petal.Width are now normalized.
# Now do the clustering based on Petal.Length and Petal.Width.
# We use K=3 (centers parameter) since we have 3 species and in order to see how K-means performs.
# Parameters:
# iris[, 3:4]: specifies data that will be given as input to K-means for clustering (3->Petal.Length, 4->Petal.Width)
# centers: Number of clusters to build (here 3)
# iter.max: tells us how many interations K-means will make
# nstart: tells us how many random samples will be tested as starts. Be best will be choosen.
irisCluster <- kmeans(iris[, 3:4], centers=3, nstart = 20, iter.max=20)

# A look at the results.


…
# MANY LINES OR R CODE HERE…
…


# iris[, 3:4]: specifies data that will be given as input to K-means for clustering (3->Petal.Length, 4->Petal.Width)
# centers: Number of clusters to build (here 3)
# iter.max: tells us how many interations K-means will make
# nstart: tells us how many random samples will be tested as starts. Be best will be choosen.
irisCluster <- kmeans(iris[, 3:4], centers=3, nstart = 20, iter.max=20)

# A look at the results.


# Define our min-max normalization function
norm <- function(x){ return( (x-min(x)) / (max(x)-min(x)) ) }

# Apply min-max normalization to the clustering attributes
iris$Petal.Length <- norm(iris$Petal.Length)
iris$Petal.Width <- norm(iris$Petal.Width)

# Parameters:
# iris[, 3:4]: specifies data that will be given as input to K-means for clustering (3->Petal.Length, 4->Petal.Width)
# centers: Number of clusters to build (here 3)
# iter.max: tells us how many interations K-means will make
# nstart: tells us how many random samples will be tested as starts. Be best will be choosen.
irisCluster <- kmeans(iris[, 3:4], centers=3, nstart = 20, iter.max=20)


# MANY LINES OR R CODE HERE…
# MANY LINES OR R CODE HERE…
# MANY LINES OR R CODE HERE…
# MANY LINES OR R CODE HERE…
# MANY LINES OR R CODE HERE…
# MANY LINES OR R CODE HERE…
# MANY LINES OR R CODE HERE…
```

Breakup single file into many smaller ones.

File: descriptive_statistics.R

Do

```
################################################################################
#
# This file contains functions and constants for calculating descriptive statics such as
# central tendency and measures of variability for various kinds of data
# (nominal, ordinal, ratio, interval). If the program needs to calculate means,
# medians, modes, stdev, variances, minimum- maximum values, kurtosis, skewness it does it
# by calling a function from this file.
#
# v1.2 -  up123456@ac.upatras.gr – Nov 2022
#
################################################################################
```

File: model.R

Do

```
################################################################################
#
# This file contains functions and constants for calculating descriptive statics such as
# central tendency and measures of variability for various kinds of data
# (nominal, ordinal, ratio, interval). If the program needs to calculate means,
# medians, modes, stdev, variances, minimum- maximum values, kurtosis, skewness it does it
# by calling a function from this file.
#
# v1.2 -  up123456@ac.upatras.gr – Nov 2022
#
################################################################################
```

File: main.R (execution starts from this file)

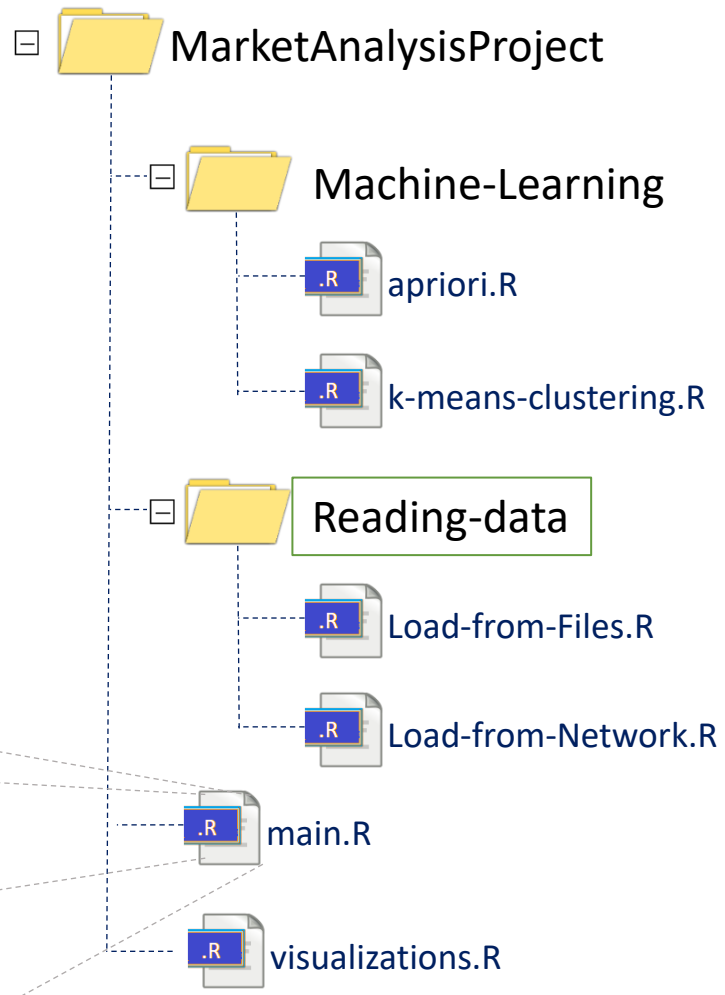
Do

```
library(rpart)
source('descriptive_statistics.R')
source('model.R')

rm(list=ls())
```

Put all source files of your project/application in the same folder/directory. In large projects (in terms of number of files), use relative folders to organize and access the project's source files.

Do

```
#####################################################################
#  Application starts from here
#  The data is loaded and k-means clustering is executed.
#  Clusters are visualized.
#
#                            v0.8.3/gbabis@upatras.gr/Dec 2019
#####################################################################
library( dplyr )

source( 'Reading-data/Load-from-Files.R')
source( 'Machine-Learning/k-means-clustering.R')
source( 'visualizations.R')
…
```



File organization of a large R project with many files on disk.

Dependencies*

*Dependencies: the libraries, modules, external files, constants, functions etc a R script requires and depends on to calculate properly the results.

Never reinvent the wheel. Prefer using existing R libraries doing a job; don't write your own function (DIY). Write your own function only if libraries does not work as desired.

Don't ✗

```
#' Calculates vif to check for multicollinearity
#'
#' @details does this by estimating a linear regression model…
#' @param data a data frame with variables
#' @param variables vector with variable names to check for collinearity
calculateVarianceInflationFactor <- function(data, variables){
   …

}
```

Do ✓

```
# Offers function vif which calculates multicollinearity
# scores for variables in a data frame
library(car)

# Calculate multicollinearity scores for all variables
vif(data)
```

Prefer existing libraries because:

1) Have been tested and used
2) Are maintained
3) Bugs are fixed
4) Increases your productivity

Make dependencies from other libraries and/or source files explicit. Put them at the beginning of R source files.

Don't ✗

```
library(dplyr)

greekBanks <- read_csv('bankdata.csv', header=T)

greekBanks <- na.omit(greekBanks)
greekBanks <- ExcludeBanks(greeBanks, 'type', 'Investment')
greekBanks['bankSize'] <- (greekBanks['bankSize'] – min(greekBanks['bankSize']) /
             (max(greekBanks['bankSize']) - min(greekBanks['bankSize'])

library(rpart)
source( 'visualizations.R')
decisionTreeFraudModel <- rpart(Fraud ~ ., data = greekBanksTraining,
                                method = "class",
                                parms = list(split = 'information'))
```

Do ✓

```
library(dplyr)
library(rpart)

source('visualizations.R')


greekBanks <- read_csv('bankdata.csv', header=T)

greekBanks <- na.omit(greekBanks)
greekBanks <- ExcludeBanks(greeBanks, 'type', 'Investment')
greekBanks['bankSize'] <- (greekBanks['bankSize'] – min(greekBanks['bankSize']) /
                          (max(greekBanks['bankSize']) - min(greekBanks['bankSize'])


decisionTreeFraudModel <- rpart(Fraud ~ ., data = greekBanksTraining,
                                method = "class",
                                parms = list(split = 'information'))
```

When making dependencies explicit in source files, report setup code first, then external libraries, then sourced files, then project scoped global constants, then function definitions.

Don't ✗

```
################################################################################
#  Application starts from here
#  The data is loaded and k-means clustering is executed. Clusters are visualized.
#
#  v2.3 - mmt@upatras.gr – Dec 2019
################################################################################
library(dplyr)
source( 'Reading-data/Load-from-Files.R')
library(rpart)
source( 'Machine-Learning/apriori.R')
source( 'visualizations.R')
kDefaultAlphaValue <- 0.00623
library(ggplot2)
setwd("\Users\jim\postgrad\Exercise2\R")
DisplayMainMenu <- function(prompt=">>", history=42){
}
```

Do ✓

```
################################################################################
#  Application starts from here
#  The data is loaded and k-means clustering is executed. Clusters are visualized.
#
#  v2.3 - mmt@upatras.gr – Dec 2019
################################################################################
rm(list=ls())
setwd("C:\\users\\student\\postgrad\\")

library(dplyr)
library(rpart)
library(ggplot2)

source( 'Reading-data/Load-from-Files.R')
source( 'Machine-Learning/apriori.R')
source( 'visualizations.R')

kDefaultAlphaValue <- 0.00623
DisplayMainMenu <- function(prompt=">>", history=42){
}
```

Environment setup code

External libraries

Sourced files

Constants

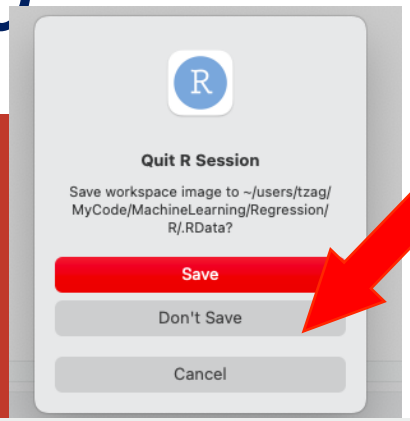
Function definitions

If setup code depends on external libraries or sourced files, setup code may repeat after external libraries and sourced files.

Always start executing scripts from a clean environment. Never save .RData file even when asked

Don't ✗

```
library(dplyr)

greekBanks <- read_csv('bankdata.csv', header=T)

greekBanks <- na.omit(greekBanks)
greekBanks <- ExcludeBanks(greeBanks, 'type', 'Investment')
greekBanks['bankSize'] <- (greekBanks['bankSize'] - min(greekBanks['bankSize']) /
             (max(greekBanks['bankSize']) - min(greekBanks['bankSize'])

library(rpart)
source( 'visualizations.R')
decisionTreeFraudModel <- rpart(Fraud ~ ., data = greekBanksTraining,
                                method = "class",
                                parms = list(split = 'information'))
```



Don't save .RData when asked. Don't. Never ever.

Do ✓

```
# Close any graphic device left open
graphics.off()

# Cleanup the environment
# all=TRUE in ls() is optional
rm(list = ls())

# Clear console
cat("\014")


library(dplyr)
library(rpart)

source('visualizations.R')

decisionTreeFraudModel <- rpart(Fraud ~ ., data = greekBanksTraining, method='class')
```

Clearing environment

General guidelines

Always get it right before you make it faster.

Don’t test your own code.

In coding/scripting, quality cannot be retrofitted (quality in terms of readability, maintainability, correctness, reliability, testability, safety etc)

Start properly.

Don’t be a d*ck

Never forget

Use any coding convention that is useful for you (no need to use all of them).

BUT ONCE ADOPTED, USE IT CONSISTENTLY!

References

References

- **R related:**
  - R best practices
    - https://www.datanovia.com/en/blog/r-coding-style-best-practices/
    - https://r-guru.com/best-practices-checklist
    - https://swcarpentry.github.io/r-novice-inflammation/06-best-practices-R.html
  - Google's R Style Guide
    - https://google.github.io/styleguide/Rguide.html
    - https://web.stanford.edu/class/cs109l/unrestricted/resources/google-style.html
  - R Amazon AWS
    - https://rstudio-pubs-static.s3.amazonaws.com/390511_286f47c578694d3dbd35b6a71f3af4d6.html
  - <- vs = in R
    - https://stackoverflow.com/questions/2271575/whats-the-difference-between-and-in-r
  - roxygen2
    - https://cran.r-project.org/web/packages/roxygen2/vignettes/rd.html

References

- **General**
  - When to use coding conventions
    - https://svitla.com/blog/why-where-and-when-to-use-coding-conventions
  - Thomas, D., Hunt, A.: *"The Pragmatic Programmer: From Journey to Mastery" - 20th Anniversary Edition*, Addison-Wesley Professional, ISBN-10 : 0135957052 , 2019
  - Martin, R. C.: "Clean Code: A Handbook of Agile Software Craftsmanship", ISBN-10: 0132350882, Pearson; 1st edition, 2008
  - *"Coding Standards A Complete Guide - 2021 Edition"*, The Art of Service - Coding Standards Publishing, ISBN-10 : 1867435020

References

- **General**
  - Stallman, R.: *“GNU Coding Standards”*, Samurai Media Limited, ISBN-10 : 9888381415, 2015
  - McConnell, S.: *“Code Complete: A Practical Handbook of Software Construction”*, Microsoft Press; 2nd edition, ISBN-10 : 0735619670, 2004.
  - Davis, A.: *“201 Principles of Software Development”*, ISBN-10 : 0070158401, McGraw-Hill, 1995)
  - Hungarian Notation: https://en.wikipedia.org/wiki/Hungarian_notation
  - K&R Style: https://gist.github.com/jesseschalken/0f47a2b5a738ced9c845

Appendix

- dsds

Don’t

```
d<-c(22,26,21,25)
x<-mean(d)
```

Do

```
studentAge <- c(22, 26, 21, 25)
averageStudentAge <- mean(studentAge)
```

Lorem Ipsum

Lorem ipsum dolor sit amet, nibh est. A magna maecenas, quam magna nec quis, lorem nunc. Suspendisse viverra sodales mauris, cras pharetra proin egestas arcu.

02

Lorem Ipsum

Lorem ipsum dolor sit amet, nibh est. A magna maecenas, quam magna nec quis, lorem nunc. Suspendisse viverra sodales mauris, cras pharetra proin egestas arcu.

01

Lorem Ipsum

Lorem ipsum dolor sit amet, nibh est. A magna maecenas, quam magna nec quis, lorem nunc. Suspendisse viverra sodales mauris, cras pharetra proin egestas arcu.

02

Lorem Ipsum

Lorem ipsum dolor sit amet, nibh est. A magna maecenas, quam magna nec quis, lorem nunc. Suspendisse viverra sodales mauris, cras pharetra proin egestas arcu.

03

Lorem Ipsum

Lorem ipsum dolor sit amet, nibh est. A magna maecenas, quam magna nec quis, lorem nunc. Suspendisse viverra sodales mauris, cras pharetra proin egestas arcu.

04

Lorem Ipsum

Lorem ipsum dolor sit amet, nibh est. A magna maecenas, quam magna nec quis, lorem nunc. Suspendisse viverra sodales mauris, cras pharetra proin egestas arcu.

02

Lorem Ipsum

Lorem ipsum dolor sit amet, nibh est. A magna maecenas, quam magna nec quis, lorem nunc. Suspendisse viverra sodales mauris, cras pharetra proin egestas arcu.

03

- dsds

Do

```
studentAge <- c(22, 26, 21, 25)
averageStudentAge <- mean(studentAge)
```

Don’t

```
d<-c(22,26,21,25)
x<-mean(d)
```