Applied Time Series

Oxana Malakhovskaya, NRU HSE

September 7, 2019

Factors

- A factor is a statistical data type used to store categorical variables, i.e. the variables that can have a limited number of values (like gender, for example).
- R works with ordered and unordered factors.
- We create factors in R by using the `factor()` function .

```
student_nationalities <- c("Russian","Polish", "French",
                           "American", "Russian",
                           "Italian", "French")
nationalities <- factor(student_nationalities)
nationalities
```

```
## [1] Russian  Polish   French   American Russian  Italian
## Levels: American French Italian Polish Russian
```

Factors

- To create a vector of ordered factors we must use `ordered` and `levels` options as arguments of the `factor` function.

```
running_speed <- c("Low","Fast","Medium","Fast",
                   "Low","Medium", "Fast")
speed_levels <- factor(running_speed, ordered = TRUE,
                   levels = c("Low", "Medium", "Fast"))
speed_levels
```

```
## [1] Low    Fast   Medium Fast   Low    Medium Fast
## Levels: Low < Medium < Fast
```

```
speed_levels[1] > speed_levels[4]
```

```
## [1] FALSE
```

Lists

- A list in R is a general form of vector that gathers differernt objects under one name in an ordered way.
- These objects are calles components and they can be matrices, vectors, data frames, even other lists, etc. No interdependence among the list components is necessary.
- We can create a list with the `list()` function.

```
vect1 <- 8:-1
matr1 <- matrix(1:4, nrow = 2)
df1 <- people[1:2,]
new_list <- list(vect1, matr1, df1)
new_list
```

Lists: an example

```
## [[1]]
##  [1]  8  7  6  5  4  3  2  1  0 -1
##
## [[2]]
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## [[3]]
##      name gender height weight single
## 1  Ivanov   male    175   83.5   TRUE
## 2 Petrova female    164   58.5   TRUE
```

Lists: components naming

- We can easily name the elements of the list when we create it.

```
new_list <- list(new_vector = vect1, new_matrix = matr1,
                 new_data_frame = df1)
```

- If we want to name the components of the list after the list was created, we can use the `names()` command as we do with vectors.

```
new_list <- list(vect1, matr1, df1)
names(new_list) <- c("new_vector","new_matrix",
                     "new_data_frame")
```

Lists: components naming(2)

```
new_list
```

```
## $new_vector
##  [1]  8  7  6  5  4  3  2  1  0 -1
##
## $new_matrix
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## $new_data_frame
##      name gender height weight single
## 1  Ivanov   male    175   83.5   TRUE
## 2 Petrova female    164   58.5   TRUE
```

Selecting a component from a list

- To select a component from a list using an index, we can use double square brackets. All three ways are equivalent:

```
new_list[[1]]
```

```
##  [1]  8  7  6  5  4  3  2  1  0 -1
```

```
new_list[["new_vector"]]
```

```
##  [1]  8  7  6  5  4  3  2  1  0 -1
```

```
new_list$new_vector
```

```
##  [1]  8  7  6  5  4  3  2  1  0 -1
```

Selecting an element from a list

- To select a particular element from the list we use squared brackets exactly as we do with vectors, matrices and data frames.

```
new_list[[1]][7]
```

```
## [1] 2
```

```
new_list[["new_matrix"]][2,2]
```

```
## [1] 4
```

```
new_list$new_data_frame[2,"name"]
```

```
## [1] Petrova
## Levels: Ivanov Petrova Vasechkin
```

Loops

Loops are used for repeated calculations. The R syntax allows different ways to do a loop. Here are two of them.

- If we know the exact number of iterations to be done, we use the `for()` function.

```
for (i in 1:2) {
  A <- diag(i)
  print(A)
}
```

```
##      [,1]
## [1,]    1
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Loops(2)

- If we want the iterations to continue until a condition is met, we use the `while()` function

```
z <- 0
while (z < 4) {
  ran <- abs(rnorm(1))
  z <- z + ran
  print(c(ran, z))
}
```

```
## [1] 0.5797943 0.5797943
## [1] 0.9099121 1.4897064
## [1] 1.206160 2.695866
## [1] 1.073483 3.769350
## [1] 1.101732 4.871082
```

Conditions

- For instructing the program to do a set of operations only if a certain condition is satisfied, we make use of the function `if()` (and `else` if necessary). Several conditions can be combined with `&&` (and) and `||` (or).

```
a <- rnorm(1)
a
```

```
## [1] -1.410146
```

```
if (a > 0) {
  print('positive')
} else {
  print('negative')
}
```

```
## [1] "negative"
```

Functions

- Functions are objects in R that when they are called, execute a certain sequence of operations. The functions are also stored in the workspace. Writing a function is a good way to expand what R can do.
- R allows the user to create his or her own functions in addition to those that already exist in the baseline software.
- A function can be defined with the code of the following form: `name <- function(arg_1, arg_2,...) expression`

```
power2 <- function(x) {
  y <- x^2
  return(y)
}
z <- power2(4)
z
```

```
## [1] 16
```

Packages: general information

- R comes with several standard packages but not all of them are automatically attached.
- Besides them, there are many contributed packages, written by people all over the world.
- Packages contain functions, the documentation explaining how to use them, and sample data.
- We can download packages from CRAN and github. Probably, there is a package written by one of the hundreds volunteers that happens to be useful for your task in hand. Install the package and attach it to your current session.

Packages: installing

- To install a package we go `Tools` $\rightarrow$ `Install Packages`. Alternatively, we can type `install.packages("packagename")` in the command prompt.

```
install.packages("tseries")
```

- Packages are sometimes interdependent. If we install one package intentionally, several others may be installed automatically.

Packages: attaching

- After downloading a package we have to attach it with the `library()` command (the require() function is also possible but not recommended). The name of the package can be written with or without quotes when the library function is called.

```
library(tseries)
```

- To see a list of attached packages we use the `search()` function with no arguments.
- Functions in different packages may happen to have the same names. To avoid confusion we use the following sintax to apply a function from a particular package: `packagename :: functionname`

```
stats :: lag
```

Datasets

- The package `datasets` contains preloaded data bases. To see a list of them we use the `data()` function with no arguments.
- Other packages can contain datasets as well. To access them we use the same function but with arguments telling R the name of the base and the name of the package.

```
data(NelPlo, package = "tseries")
```

- If a package is attached, the datasets contained in this package become available to the user.
- To see a list of datasets in a package we use the data function with an only argument telling R the name of the package.

```
data(package = "tseries")
```

Editing data

- To edit data in a separate spreadsheet-like window we use the `edit(objectname)` command.

```
people2 <-  edit(people)
```

- By clicking the object in the workspace we can just view it but this does not allow to edit it. This is equivalent to the command `View(objectname)`.

```
View(people)
```

Our first packages

- Install `tidyverse` package. This is a collection of R packages designed for data analyses. The packages included in the `tidyverse` set are:

```
* `ggplot2` -  to get a nice visualization of data
* `dplyr` - to manipulate data quickly and efficiently
* `readr` - to read rectangular data (like csv).
*  and three other packages
```

- Each package from this collection may also require other packages which are also installed automatically.
- Do not forget to attach the `tidyverse` package.

```
library(tidyverse)
```

Working with time series

The algorighm of working with times series models.

1. Importing data that we will do with `readr` or `readxl` packages.
2. Data visualisation that we will do with `ggplot2` package (if necessary).
3. Data manipulation that we will do with `dplyr` package treating series as data frames.
4. Transformation data frames into special time series (ts) format if necessary (some packages work only with ts data).
5. Estimation and evaluation of models with special packages developed specially for a certain class of models.

Entering data from keyboard

- If a dataset is small we just enter it with `c()` or 'data.frame()' commands and then edit them if necessary. We also can create an empty data frame and then edit it.

```
dataset <- data.frame()
dataset <- edit(dataset)
```

Importing data from an external source

- If a dataset is big we can import the data by clicking 'Import Dataset' in the Environment menu and choosing the source. The reading commands are from packages that are attached automatically.
- The same reading commands may be typed by the user as well. Do not forget to attach the respective package first.

```
library(readr) # not necessary if the `tidyverse` package
               #is already attached.
data1 <- read_csv("mydata.csv")
```

- There are also importing commands that do not require any packages to be attached (the 'utils' package where they belong to is attached automatically).

```
data2 <- read.csv("mydata.csv", stringsAsFactors = FALSE)
```