ACT-11302 Calculo Actuarial III

Sesion 00 - Introduccion a R

Juan Carlos Martinez-Ovando

Departamento Academico de Actuaria y Seguros

Preambulo

En este curso usaremos R como herramienta analitica de computo, y RStudio como editor y visualizador a datos y analisis.
En estas notas revisaremos algunas de las funcionalidades mas general de R.
Empezaremos con la instalacion de paquetes o librerias uqe utlizaremos a lo largo del curso.
Desde RStudio estas librerias pueden instalarse desde el repositorio de R en el ITAM empleando las siguientes instrucciones:

Paquetes

```
options("repos"="http://cran.itam.mx")

upackages = c("actuar","repmis", "tidyr",
              "lubridate", "dplyr", "ggplot2", "googledrive")

install.packages(upackages)
```

Paquetes

Breve descripcion y uso:

- `repmis` - Importacion de datos
- `tidyr` - Manipulacion de datos
- `lubridate` - Manejo de datos temporales
- `dplyr` - Manipulacion y agregacion de datos
- `ggplot2` - Visualizacion de datos

Importacion de datos

Cargamos los datos desde un repositorio en GitHUb (en este caso, el repositorio de datos de nuestro curso). Para esto, empleamos en RStudio el paquete `repmis`, `curl`, APIs, entre otros.
Los datos del curso estan almacenados en carpetas de Google Drive, veremos como descarcarlos directamente desde tales repositorios.
Por ejemplo, para leer los datos de reclamos de seguros de viviendas en Dinamarca[1], empleamos:

[1] Referencia: McNeil et al (2007) *Estimating the Tails of Loss Severity Distributions using Extreme Value Theory1997*

Codigo 1

```
library("googledrive")
temp <- tempfile(fileext = ".zip")
dl <- download.file(
        as_id("1AiZda_1-2nwrxI8fLD0Y6e5rTg7aocv0"),
        path = temp,
        overwrite = TRUE)
output <- unzip(temp, exdir = tempdir())
datos <- read.csv(out[14], sep = ",")
```

Codigo 2

```
library(repmis)

source_data("https://github.com/jcmartinezovando/est25134_2017a/blob/ma
```

Comentario

Noten que debemos especificar el path de los datos desde el repositorio en GitHub; este lo pueden encontrar seleccionando la opcion `Copy path` que se despliega al seleccionar cada archivo da datos. Empleando `repmis` es necesario que el path de datos sea `blob` Noten tambien la ultima instruccion para especificar que los datos son libres de formato. Tambien podemos cargar los datos en R empleando las siguientes opciones.

Otra forma

```
rm(list=ls())
githubURL <- "https://github.com/jcmartinezovando/est25134_2017a/raw/ma

# For Windows
load(url(githubURL))

# If trouble, try this in Linux or iOS
download.file(githubURL,"DanishInsuranceMultivariate_Data")
load("DanishInsuranceMultivariate_Data")
ls()
```

Otros comentarios

Noten que en este caso, el path de datos debe ser del tipo `raw`. Es la instruccion analoga a la ultima que empleamos en `repmis`.
La ultima opcion es quizas la mas conveniente, pues es autonoma de otras librerias a las preestablecidas en R. Sin embargo, en ciertos contextos `repmis` puede ser una muy buena opcion.

Operaciones con vectores

Los vectores en R se definen, en principio, como secuencias de datos. Las entradas de los vectores se acceden empleando []

```
a <- c(5, 2, 4.1, 7, 9.2)
a
```

```
## [1] 5.0 2.0 4.1 7.0 9.2
```

```
a[2:4]
```

```
## [1] 2.0 4.1 7.0
```

Operaciones

Como en otros lenguajes de programacion, las operaciones con vectores se realizan entrada por entrada. En particular, debemos cuidar que el pŕoducto de vectores, como se muestra a continuacion, los componentes sean compatibles en longitud.

```
b <- a + 10
b
```

```
## [1] 15.0 12.0 14.1 17.0 19.2
```

```
c <- sqrt(a)
c
```

```
## [1] 2.236068 1.414214 2.024846 2.645751 3.033150
```

```
a + c
```

```
## [1]  7.236068  3.414214  6.124846  9.645751 12.233150
```

```
10*a
```

```
## [1] 50 20 41 70 92
```

Operaciones

Se pueden definir arreglos vectoriales de objetos particulares, como textos:

```
vector_texto <- c('manzana','manzana','pera','platano','fresa')
vector_texto
```

```
## [1] "manzana" "manzana" "pera"    "platano" "fresa"
```

Solo que en este caso, varias operaciones (particularmente aritmeticas) no son aplicables.

Data-frames

Los `data.frames` son arreglos de vectores en R cuyos vectores entrada (columna) pueden tener atributos particulares. A lo largo del curso estaremos trabajando con `data.frames` todo el tiempo.
Para definir *data-frames* podemos usar instrucciones de `R-base` o del paquete `dplyr`. A continuacion un ejemplo:

```
# R-base
tabla1 <- data.frame(n = 1:5, valor = a, fruta = vector_texto)
tabla1
```

```
##   n valor   fruta
## 1 1   5.0 manzana
## 2 2   2.0 manzana
## 3 3   4.1    pera
## 4 4   7.0 platano
## 5 5   9.2   fresa
```

```
# dplyr
tabla2 <- data_frame(n = 1:5, valor = a, fruta = vector_texto)
```

Consultas

Consultas a entradas renglon o columna de los data frames pueden relaizar analogamente en `R-base` o `dplyr`. Sin embargo, en el segundo se cuanta con mas atributos de utilidad que explotaremos mas adelante.

```
# R-base
tabla1$valor
```

```
## [1] 5.0 2.0 4.1 7.0 9.2
```

```
tabla1$valor[3]
```

```
## [1] 4.1
```

```
# dplyr
tabla2$valor
```

```
## [1] 5.0 2.0 4.1 7.0 9.2
```

```
tabla2$valor[3]
```

```
## [1] 4.1
```

Funciones

Las funciones en R son reglas de asociacion de objetos. Dos ejemplos.

- Funcion convencional

```
ejemplo_fun <- function(x){
  y <- x + 1
  y^2 # este ultimo valor es el que regresa la funcion
}

ejemplo_fun(2)
```

```
## [1] 9
```

```
(2+1)^2
```

```
## [1] 9
```

Funcion con funciones anonimas como argumento

```
procentaje_agr <- function(x, mult = 100, FUN = round, ...){
 porcentaje <- FUN(x * mult, ...)
 paste(porcentaje, "%", sep = "")
}
utilidades <- c(2100, 1430, 3580, 5230)
utilidades_relativas <- function(x) round(x / sum(x) * 100)
procentaje_agr(utilidades,
        FUN = function(x) round(x / sum(x) * 100) )
```

```
## [1] "17%" "12%" "29%" "42%"
```

Pipes

En `dply` existen varias funciones que son utiles para la manipulacion de `data-frames`. Entre ellas:

1. `select` - extraer columnas
2. `filter` - extraer renglones
3. `mutate` - crear variables
4. `group_by` - agrupar por columnas
5. `summarise` - resumen de contenido y caracteristicas.

En particular, cuanta con la funcion `%>%` (`pipe`), que definen secuencias de funciones aplicadas a uno o varios objetos. Por ejemplo:

```
9 %>% sqrt
```

```
## [1] 3
```

Mas sobre pipes

Esta funcion se define como la secuencia de crear el objeto 9 y posteriormente aplicarle la funcion `sqrt`. Esto es equivalente a la siguiente instruccion en `R-base`:

```
sqrt(9)
```

```
## [1] 3
```

Table of Contents