

Introducción al software estadístico

Módulo V

Nicolás Schmidt

`nschmidt@cienciassociales.edu.uy`

Departamento de Ciencia Política
Facultad de Ciencias Sociales
Universidad de la República

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Estructura de la presentación

1 Marco de Datos

■ Estructura

- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Estructura de un `data.frame`

Un marco de datos es la estructura más frecuente en el análisis de datos. En un sentido estricto un marco de datos (`data.frame`) es una lista con vectores de igual longitud con nombres únicos.

Esto hace que los marcos de datos compartan propiedades con las listas y con las matrices.

- Un marco de datos comparte atributos con las matrices: `names()` y `rownames()`. [`names()` es igual a `colnames()`]
- El `length()` de un marco de datos es la longitud de la lista de vectores. Esto es igual a `ncol()`

Estructura de un data.frame

Como la lista es una estructura de datos heterogénea y un marcos de datos es una lista esto significa que puede contener datos de distinto tipo. El tipo de dato que contiene cada variable se puede consultar de distintas maneras:

Ejemplo: `str()` y `sapply()`

```
str(iris)

## 'data.frame': 150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...

sapply(iris, class)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##      "numeric"      "numeric"      "numeric"      "numeric"      "factor"
```

Estructura de la presentación

1 Marco de Datos

- Estructura
- **Atributos**
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Atributos de un data.frame

Un data.frame tiene tres atributos:

<code>names</code>	Nombre de las variables (columnas)
<code>row.names</code>	Nombre de las filas
<code>class</code>	La clase. Si se aplica <code>typeof</code> a un <code>data.frame</code> el resultado es 'list'.

Ejemplo:

```
attributes(data.frame(a = 1:5, b = 11:15))  
  
## $names  
## [1] "a" "b"  
##  
## $class  
## [1] "data.frame"  
##  
## $row.names  
## [1] 1 2 3 4 5
```


Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Creación de un data.frame()

```
data.frame(..., row.names = NULL, check.rows = FALSE,  
           check.names = TRUE, fix.empty.names = TRUE,  
           stringsAsFactors = default.stringsAsFactors())
```

...	Valores o nombres = valores.
row.names	Vector de caracteres que contiene el nombre de las filas o nombre de la columna que se va a usar como nombre de filas o el número de la columna.
check.rows	Si es TRUE verifica la consistencia de la longitud y los nombres de las filas.
check.names	Por defecto es TRUE. Chequea que no existan espacios en los nombres de las variables. Si hay los anula con un '.'.
fix.empty.names	Por defecto es TRUE. Si una o más variables (columnas) no tienen nombre les pone uno por defecto.
stringsAsFactors	Por defecto es TRUE. Convierte los vectores de caracteres a factores.

Ejemplo: usando el argumento row.names

```
df1 <- data.frame(a = 30:33, b = 40:43, c = letters[1:4]); df1
```

```
##      a  b c  
## 1 30 40 a  
## 2 31 41 b  
## 3 32 42 c  
## 4 33 43 d
```

```
rownames(df1)
```

```
## [1] "1" "2" "3" "4"
```

```
df2 <- data.frame(a = 30:33, b = 40:43, c = letters[1:4], row.names = 3); df2
```

```
##      a  b  
## a 30 40  
## b 31 41  
## c 32 42  
## d 33 43
```

```
rownames(df2)
```

```
## [1] "a" "b" "c" "d"
```

Ejemplo: usando el argumento row.names

```
df3 <- data.frame(a = 30:33, b = 40:43, c = letters[1:4], row.names = "b"); df3
```

```
##      a c  
## 40 30 a  
## 41 31 b  
## 42 32 c  
## 43 33 d
```

```
rownames(df3)
```

```
## [1] "40" "41" "42" "43"
```

```
df4 <- data.frame(a = 30:33, b = 40:43, c = letters[1:4],  
                  row.names = paste("Fila", 1:4)); df4
```

```
##      a  b c  
## Fila 1 30 40 a  
## Fila 2 31 41 b  
## Fila 3 32 42 c  
## Fila 4 33 43 d
```

```
rownames(df4)
```

```
## [1] "Fila 1" "Fila 2" "Fila 3" "Fila 4"
```

Ejemplo: usando el argumento stringsAsFactors

```
df <- data.frame(a = 30:33, b = letters[1:4]); df

##      a b
## 1 30 a
## 2 31 b
## 3 32 c
## 4 33 d

sapply(df, class)

##           a           b
## "integer" "factor"

df <- data.frame(a = 30:33, b = letters[1:4], stringsAsFactors = FALSE); df

##      a b
## 1 30 a
## 2 31 b
## 3 32 c
## 4 33 d

sapply(df, class)

##           a           b
## "integer" "character"
```

Ejemplo: usando el argumento `check.names` y `fix.empty.names`

```
data.frame("a b" = 30:33, 40:43, check.names = TRUE)
```

```
##      a.b X40.43
## 1    30      40
## 2    31      41
## 3    32      42
## 4    33      43
```

```
data.frame("a b" = 30:33, 40:43, check.names = FALSE)
```

```
##      a b 40:43
## 1    30    40
## 2    31    41
## 3    32    42
## 4    33    43
```

```
data.frame("a b" = 30:33, 40:43, check.names = TRUE, fix.empty.names = FALSE)
```

```
##      a.b
## 1    30 40
## 2    31 41
## 3    32 42
## 4    33 43
```

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- **Cargar**
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Funciones para cargar un data.frame

Lo más frecuente es que los marcos de datos no se construyan en R sino que se importen. El asunto con la importación de datos es que según la extensión del archivo será la función y las especificaciones que se necesiten.

Archivo	Función	paquete
txt	<code>read.table()</code>	utils, readr
csv	<code>read.csv()</code>	utils, readr
excel	<code>read.xlsx()</code>	xlsx
	<code>read_excel()</code>	readxl
ods	<code>read_ods()</code>	readODS
Stata	<code>read.dta()</code>	foreign
	<code>read.dta13()</code>	readstata13
	<code>read_dta()</code>	haven
spss	<code>read.spss()</code>	foreign
	<code>read_spss()</code>	haven

Detalles a tener en cuenta al cargar datos

Para que R pueda cargar los datos que se le solicitan es necesario que se especifique la ruta hacia el archivo que se desea leer. El argumento que comúnmente se llama `file` en todas las funciones de carga de datos admiten una ruta completa o el nombre del archivo con la extensión siempre que esté activo el `setwd()` o estemos trabajando en un proyecto de RStudio.

Ejemplo: flujo frecuente de carga de datos

```
setwd(ruta_donde_está_el_archivo)

library(foreign)

base <- read.dta("nombre_del_archivo.extension_delarchivo"...)
```

Argumentos relevantes de las funciones de carga de datos

Las funciones que aparecen en la tabla no tienen los mismos argumentos, pero es frecuente encontrar estos argumentos:


Argumento	Descripción
<code>header = TRUE</code>	Si los datos tiene variables con nombre las mantiene
<code>stringsAsFactors</code>	Si es FALSE no modifica las variables de tipo character a factor
<code>as.is = TRUE</code>	Si es TRUE mantiene las cadenas de caracteres como cadenas.
<code>sep = " "</code>	Separador entre datos
<code>dec = " "</code>	Especifica el separador de decimales

Cargar datos de manera interactiva

Dos opciones para cargar datos sin tener que especificar la ruta manualmente son:

- `file.choose()`. Esta función abre una ventana interactiva para seleccionar el archivo que se desea abrir.

```
# Ejemplo  
datos <- read.dta13(file.choose())
```

- Opción  Import Dataset en RStudio.

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- **Indexación**
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

'[]', '[,]', '[[', '\$'

Hay múltiples maneras de acceder a los datos de un `data.frame`. Las distintas formas tiene un impacto distinto sobre la estructura que se indexa.

Argumento	Descripción
[]	Devuelve un <code>data.frame</code> .
[,]	Devuelve un vector.
[[]]	Devuelve un vector.
\$	Devuelve un vector.

‘[]’, ‘[,]’, ‘[[’, ‘\$’

Ejemplo:

```
head(iris[2], 10)
```

```
##      Sepal.Width
## 1           3.5
## 2           3.0
## 3           3.2
## 4           3.1
## 5           3.6
## 6           3.9
## 7           3.4
## 8           3.4
## 9           2.9
## 10          3.1
```

```
class(iris[2])
```

```
## [1] "data.frame"
```

‘[]’, ‘[,]’, ‘[[’, ‘\$’

Ejemplo:

```
iris[[2]]
```

```
##      [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9
##     [18] 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2
##     [35] 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2
##     [52] 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7
##     [69] 2.2 2.5 3.2 2.8 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0
##     [86] 3.4 3.1 2.3 3.0 2.5 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7
##    [103] 3.0 2.9 3.0 3.0 2.5 2.9 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6
##    [120] 2.2 3.2 2.8 2.8 2.7 3.3 3.2 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0
##    [137] 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2 3.3 3.0 2.5 3.0 3.4 3.0
```

```
class(iris[[2]])
```

```
## [1] "numeric"
```

‘[]’, ‘[,]’, ‘[[’, ‘\$’

Ejemplo:

```
iris[,2]
```

```
##      [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9
##     [18] 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2
##     [35] 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2
##     [52] 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7
##     [69] 2.2 2.5 3.2 2.8 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0
##     [86] 3.4 3.1 2.3 3.0 2.5 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7
##    [103] 3.0 2.9 3.0 3.0 2.5 2.9 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6
##    [120] 2.2 3.2 2.8 2.8 2.7 3.3 3.2 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0
##    [137] 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2 3.3 3.0 2.5 3.0 3.4 3.0
```

```
class(iris[,2])
```

```
## [1] "numeric"
```


‘[]’, ‘[,]’, ‘[[’, ‘\$’

Ejemplo:

```
iris$Sepal.Width
```

```
##      [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9
##     [18] 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2
##     [35] 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2
##     [52] 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7
##     [69] 2.2 2.5 3.2 2.8 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0
##     [86] 3.4 3.1 2.3 3.0 2.5 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7
##    [103] 3.0 2.9 3.0 3.0 2.5 2.9 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6
##    [120] 2.2 3.2 2.8 2.8 2.7 3.3 3.2 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0
##    [137] 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2 3.3 3.0 2.5 3.0 3.4 3.0
```

```
class(iris$Sepal.Width)
```

```
## [1] "numeric"
```

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Creación de variables

Hay múltiples maneras de crear nuevas variables:

Ejemplo:

```
datos <- iris
datos[,6] <- 1
datos[[7]] <- 2
datos[8] <- 3
datos$variable <- 4
str(datos)

## 'data.frame': 150 obs. of  9 variables:
## $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
## $ V6           : num  1 1 1 1 1 1 1 1 1 1 ...
## $ V7           : num  2 2 2 2 2 2 2 2 2 2 ...
## $ V8           : num  3 3 3 3 3 3 3 3 3 3 ...
## $ variable     : num  4 4 4 4 4 4 4 4 4 4 ...
```

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Unir bases de datos

Una manera simple es usar las funciones `cbind()` y `rbind()` pero tienen el problema de que los datos deben estar igualmente ordenados. La alternativa es usar `merge()`.

```
merge(x, y, by = intersect(names(x), names(y)), by.x =  
by, by.y = by, all = FALSE, all.x = all, all.y = all,  
sort = TRUE, suffixes = c(".x", ".y"), no.dups = TRUE,  
incomparables = NULL, ...)
```

`x, y`
`by, by.x, by.y`
`all, all.x, all.y`
`sort`
`suffixes`

`no.dups`

Datos que se forzarán a uno.
Indicaciones de las columnas utilizadas para la fusión.
Agrega filas cuando no son coincidente y computa un NA.
Por defecto es TRUE. Ordena los datos por columnas.
Vector de caracteres de longitud 2 para identificar
nombres repetidos.
Argumento adicional a `suffixes`.

Creao dos data.frame

```
df1 <- data.frame(  
  Nombre = c("Pedro", "Jose", "Juan", "Martin", "Julio"),  
  Compra1 = sample(500:1000, 5, replace = TRUE),  
  Compra2 = sample(500:1000, 5, replace = TRUE))  
df2 <- data.frame(  
  Nombre = c("Pedro", "Jose", "Maria", "Martin", "Juliana"),  
  Compra1 = sample(500:1000, 5, replace = TRUE),  
  Compra2 = sample(500:1000, 5, replace = TRUE))
```

df1

##	Nombre	Compra1	Compra2
## 1	Pedro	953	818
## 2	Jose	806	932
## 3	Juan	786	716
## 4	Martin	766	656
## 5	Julio	525	837

df2

##	Nombre	Compra1	Compra2
## 1	Pedro	641	619
## 2	Jose	736	931
## 3	Maria	599	665
## 4	Martin	737	899
## 5	Juliana	686	685

merge()

```
merge(df1, df2, by = "Nombre")
```

```
##   Nombre Compra1.x Compra2.x Compra1.y Compra2.y
## 1   Jose        806        932        736        931
## 2  Martin        766        656        737        899
## 3  Pedro        953        818        641        619
```

```
merge(df1, df2, by = "Nombre", all.x = TRUE)
```

```
##   Nombre Compra1.x Compra2.x Compra1.y Compra2.y
## 1   Jose        806        932        736        931
## 2   Juan        786        716         NA         NA
## 3  Julio        525        837         NA         NA
## 4  Martin        766        656        737        899
## 5  Pedro        953        818        641        619
```

```
merge(df1, df2, by = "Nombre", all.y = TRUE)
```

```
##   Nombre Compra1.x Compra2.x Compra1.y Compra2.y
## 1   Jose        806        932        736        931
## 2  Martin        766        656        737        899
## 3  Pedro        953        818        641        619
## 4  Juliana         NA         NA        686        685
## 5   Maria         NA         NA        599        665
```

merge()

```
merge(df1, df2, by = "Nombre", all = TRUE)
```

##	Nombre	Compra1.x	Compra2.x	Compra1.y	Compra2.y
## 1	Jose	806	932	736	931
## 2	Juan	786	716	NA	NA
## 3	Julio	525	837	NA	NA
## 4	Martin	766	656	737	899
## 5	Pedro	953	818	641	619
## 6	Juliana	NA	NA	686	685
## 7	Maria	NA	NA	599	665

```
merge(df1, df2, by = "Nombre", all = TRUE, suffixes = c("_T1", "_T2"))
```

##	Nombre	Compra1_T1	Compra2_T1	Compra1_T2	Compra2_T2
## 1	Jose	806	932	736	931
## 2	Juan	786	716	NA	NA
## 3	Julio	525	837	NA	NA
## 4	Martin	766	656	737	899
## 5	Pedro	953	818	641	619
## 6	Juliana	NA	NA	686	685
## 7	Maria	NA	NA	599	665

merge()

```
df1$Apellido = c("Lopez", "Perez", "Gonzalez", "Rodriguez", "Rato")
df2$Apellido = c("Pallas", "Perez", "Torres", "Rodriguez", "Garcia")
df1 <- df1[,c(1,4,2,3)]; df1 # cambio el orden de las columnas: la 4 va al lugar 2

##   Nombre  Apellido Compra1 Compra2
## 1  Pedro    Lopez     953     818
## 2   Jose    Perez     806     932
## 3   Juan  Gonzalez     786     716
## 4 Martin Rodriguez     766     656
## 5  Julio     Rato     525     837

df2 <- df2[,c(1,4,2,3)]; df2 # cambio el orden de las columnas: la 4 va al lugar 2

##   Nombre  Apellido Compra1 Compra2
## 1  Pedro    Pallas     641     619
## 2   Jose    Perez     736     931
## 3  Maria    Torres     599     665
## 4 Martin Rodriguez     737     899
## 5 Juliana   Garcia     686     685
```

merge()

```
merge(df1, df2, by = "Nombre")
```

```
##   Nombre Apellido.x Compra1.x Compra2.x Apellido.y Compra1.y Compra2.y
## 1   Jose      Perez      806      932      Perez      736      931
## 2 Martin Rodriguez      766      656 Rodriguez      737      899
## 3  Pedro      Lopez      953      818      Pallas      641      619
```

```
merge(df1, df2, by.x = c("Nombre", "Apellido"), by.y = c("Nombre", "Apellido"))
```

```
##   Nombre Apellido Compra1.x Compra2.x Compra1.y Compra2.y
## 1   Jose      Perez      806      932      736      931
## 2 Martin Rodriguez      766      656      737      899
```

```
merge(df1, df2, by.x = c("Nombre", "Apellido"), by.y = c("Nombre", "Apellido"),
      suffixes = c("_T1", "_T2"))
```

```
##   Nombre Apellido Compra1_T1 Compra2_T1 Compra1_T2 Compra2_T2
## 1   Jose      Perez      806      932      736      931
## 2 Martin Rodriguez      766      656      737      899
```

Uniendo por todas las variables

`merge()`

```
df1$tiempo <- 2017  
df2$tiempo <- 2018  
merge(df1, df2, all = TRUE)
```

##	Nombre	Apellido	Compra1	Compra2	tiempo
## 1	Jose	Perez	736	931	2018
## 2	Jose	Perez	806	932	2017
## 3	Juan	Gonzalez	786	716	2017
## 4	Julio	Rato	525	837	2017
## 5	Martin	Rodrigues	737	899	2018
## 6	Martin	Rodrigues	766	656	2017
## 7	Pedro	Lopez	953	818	2017
## 8	Pedro	Pallas	641	619	2018
## 9	Juliana	Garcia	686	685	2018
## 10	Maria	Torres	599	665	2018

merge()

```
d <- merge(df1[sample(1:4)], df2[sample(1:4)], all = TRUE); d
```

	Compra2	Apellido	Compra1	Nombre
## 1	619	Pallas	641	Pedro
## 2	656	Rodrigues	766	Martin
## 3	665	Torres	599	Maria
## 4	685	Garcia	686	Juliana
## 5	716	Gonzalez	786	Juan
## 6	818	Lopez	953	Pedro
## 7	837	Rato	525	Julio
## 8	899	Rodrigues	737	Martin
## 9	931	Perez	736	Jose
## 10	932	Perez	806	Jose

```
d[match(c("Nombre", "Apellido", "Compra1", "Compra2"), names(d))]
```

	Nombre	Apellido	Compra1	Compra2
## 1	Pedro	Pallas	641	619
## 2	Martin	Rodrigues	766	656
## 3	Maria	Torres	599	665
## 4	Juliana	Garcia	686	685
## 5	Juan	Gonzalez	786	716
## 6	Pedro	Lopez	953	818
## 7	Julio	Rato	525	837
## 8	Martin	Rodrigues	737	899
## 9	Jose	Perez	736	931
## 10	Jose	Perez	806	932

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

■ Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Identificar NA

Usamos la base de datos iris del paquete datasets que ya viene con la instalación de R y modificamos aleatoriamente valores por NA.

Para ver todas las bases de datos que hay en ese paquete:

```
ls("package:datasets")
```

Modificamos datos:

```
iris2 <- iris[,-5]
for(i in 1:ncol(iris2)){
  set.seed(2018*i)
  losNA <- sample(1:150, 10*i)
  iris2[losNA,i] <- NA
}
summary(iris2)
```

##	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
##	Min. :4.300	Min. :2.000	Min. :1.000	Min. :0.100
##	1st Qu.:5.100	1st Qu.:2.800	1st Qu.:1.500	1st Qu.:0.400
##	Median :5.800	Median :3.000	Median :4.150	Median :1.400
##	Mean :5.847	Mean :3.065	Mean :3.626	Mean :1.265
##	3rd Qu.:6.400	3rd Qu.:3.400	3rd Qu.:5.025	3rd Qu.:1.900
##	Max. :7.900	Max. :4.400	Max. :6.900	Max. :2.500
##	NA's :10	NA's :20	NA's :30	NA's :40

Identificar NA

Hay 4 funciones que son particularmente útiles para identificar valores NA

- `is.na()`
- `complete.cases()`
- `anyNA()`
- `na.omit()`

El valor NA es de tipo `logical`. Esto significa que identificar estos valores implica usar operadores lógicos.

Ejemplo: is.na() para toda la base

```
iNA <- is.na(iris2)
```

```
head(iNA)
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]          FALSE          FALSE          FALSE          TRUE
## [2,]          FALSE           TRUE          FALSE         FALSE
## [3,]          FALSE          FALSE          FALSE          TRUE
## [4,]          FALSE          FALSE          FALSE         FALSE
## [5,]          FALSE          FALSE          FALSE          TRUE
## [6,]          FALSE          FALSE          FALSE         FALSE
```

```
tail(iNA)
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [145,]          FALSE          FALSE          TRUE          FALSE
## [146,]          FALSE          FALSE          FALSE         FALSE
## [147,]          FALSE          FALSE          TRUE          FALSE
## [148,]          FALSE          FALSE          TRUE          FALSE
## [149,]           TRUE          FALSE          FALSE         FALSE
## [150,]          FALSE          FALSE          TRUE          FALSE
```

```
sum(iNA)
```

```
## [1] 100
```

Ejemplo: is.na() por columnas

```
iris2[is.na(iris2[,1]),]
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
##  9              NA          2.9          1.4          NA
## 19              NA          3.8          1.7          NA
## 30              NA          3.2          1.6          NA
## 44              NA          3.5           NA          NA
## 51              NA           NA          4.7          1.4
## 70              NA          2.5          3.9          NA
## 78              NA          3.0          5.0          1.7
## 88              NA          2.3           NA          NA
## 137             NA          3.4           NA          2.4
## 149             NA          3.4          5.4          2.3
```

```
iris2[is.na(iris2[,1]) & is.na(iris2[,2]) ,]
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## 51              NA          NA          4.7          1.4
```

```
colSums(is.na(iris2))
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width
##           10           20           30           40
```

Ejemplo: `is.na()` por filas

```
rowSums(is.na(iris2))
```

```
##      [1] 1 1 1 0 1 0 1 1 2 0 0 1 0 1 0 1 0 0 2 0 1 0 0 0 0 1 1 0 1 2 1 0 1 0 1
##     [36] 0 0 0 0 0 0 0 0 3 3 0 0 1 0 1 1 2 0 1 1 0 1 1 1 1 0 0 1 0 0 0 0 1 1 1 2
##     [71] 0 0 1 0 1 1 1 1 2 0 2 0 1 1 1 1 1 3 0 0 2 0 1 0 0 0 1 1 0 0 0 0 0 2 1
##    [106] 1 2 0 1 0 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 2 0 1 2 0 0 1
##    [141] 0 2 1 3 1 0 1 1 1 1
```

```
iris2[rowSums(is.na(iris2)) > 2, ]
```

##	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
## 43	4.4	NA	NA	NA
## 44	NA	3.5	NA	NA
## 88	NA	2.3	NA	NA
## 144	6.8	NA	NA	NA

Ejemplo: complete.cases()

```
sum(complete.cases(iris2[,1]))

## [1] 140

# cantidad de casos completos por columna
apply(iris2, 2, function(x){sum(complete.cases(x))})

## Sepal.Length Sepal.Width Petal.Length Petal.Width
##           140           130           120           110

#porcentaje de casos completos
apply(iris2, 2, function(x){sum(complete.cases(x))/length(x)})

## Sepal.Length Sepal.Width Petal.Length Petal.Width
##    0.9333333    0.8666667    0.8000000    0.7333333

which(apply(iris2, 1, function(x){sum(complete.cases(x))/length(x)}) == .50)

## [1]  9 19 30 51 70 79 81 91 104 107 134 137 142

mitad <- which(apply(iris2, 1, function(x){sum(complete.cases(x))/length(x)})==.50)
which(apply(iris2[-mitad,], 1, function(x){sum(complete.cases(x))/length(x)})==.50)

## named integer(0)
```

⇒ `anyNA` es una función genérica: `anyNA(x)` es una alternativa más rápida a `any(is.na(x))`.

Ejemplo: `anyNA()`

```
apply(iris2, 2, anyNA)

## Sepal.Length Sepal.Width Petal.Length Petal.Width
##           TRUE           TRUE           TRUE           TRUE
```

⇒ `na.omit()` elimina todos los valores NA

Ejemplo: `na.omit()`

```
iris.limpia <- na.omit(iris2)
colSums(is.na(iris.limpia))

## Sepal.Length Sepal.Width Petal.Length Petal.Width
##           0           0           0           0

dim(iris.limpia)

## [1] 71 4
```

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

■ Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

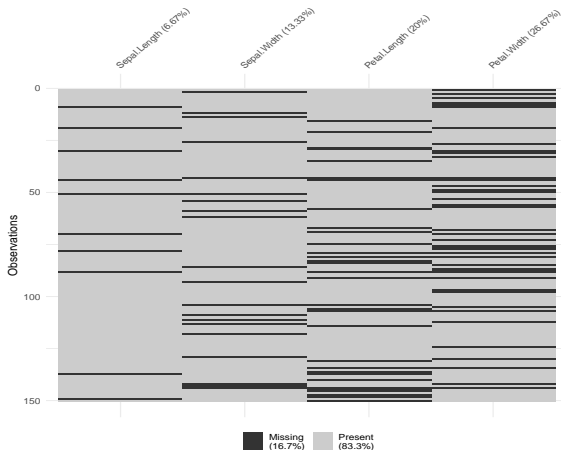
7 Funciones de estadística descriptiva básicas

8 with() y attach()

Visualizar valores NA

Hay varios paquetes que permiten visualizar valores NA. Uno de ellos es `visdat`:

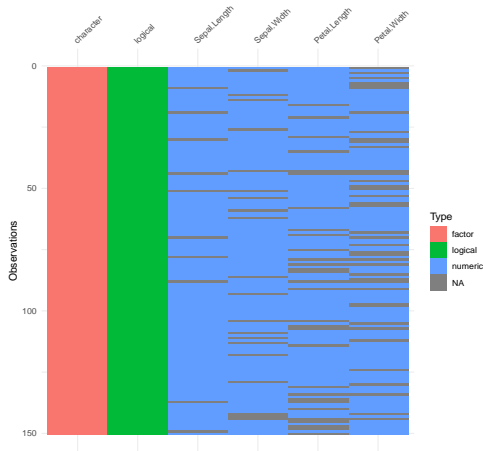
```
visdat::vis_miss(iris2)
```



Visualizar valores NA

Tambien es posible visualizar por tipo de dato:

```
visdat::vis_dat(cbind(iris2, character = "a", logical = TRUE))
```



Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

■ Creación de Tablas

- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Creación de Tablas

Una tabla comúnmente es una matriz (o un array para ser más específicos) que es de la clase 'table'. Al ser una matriz comparte las propiedades (`dim()` y `dimnames()`) de esa estructura de datos y se accede a los datos de la misma manera. Las tablas reportan utilidad ya que nos proporcionan información resumida sobre los datos.

- En el caso de una variable es una tabla de frecuencia. La estructura es similar a un vector. El acceso a los datos es igual que con vectores.
- El caso de dos o más variables se denomina tabla de contingencia. La estructura de estas tablas es un array y se accede haciendo referencia a las dimensiones.

Creación de Tablas

Ejemplo:

```
tab1 <- table(mtcars$cyl, mtcars$carb); tab1
```

```
##  
##      1 2 3 4 6 8  
##    4 5 6 0 0 0  
##    6 2 0 0 4 1  
##    8 0 4 3 6 1
```

```
class(tab1)
```

```
## [1] "table"
```

```
is.array(tab1)
```

```
## [1] TRUE
```

```
is.matrix(tab1)
```

```
## [1] TRUE
```

Creación de Tablas

Ejemplo:

```
tab1 <- table(iris[,1] > mean(iris[,1]), iris[,5])  
rownames(tab1) <- c("Menor a la media", "Mayor a la media")  
addmargins(tab1, 2)
```

```
##  
##          setosa versicolor virginica Sum  
## Menor a la media      50         24         6  80  
## Mayor a la media       0         26        44  70
```

Creación de Tablas

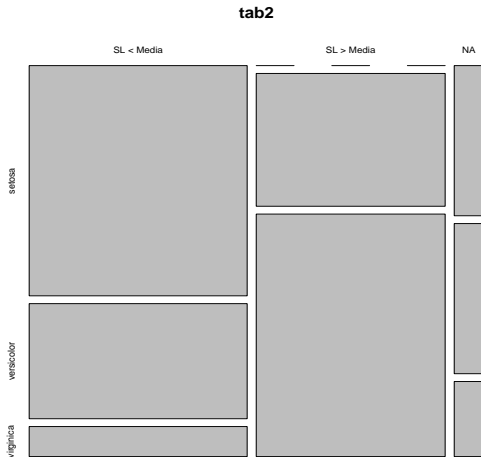
Ejemplo:

```
iris2 <- iris
for(i in 1:4){
  set.seed(2018*i)
  losNA <- sample(1:150, 10*i)
  iris2[losNA,i] <- NA
}
tab2 <- table(iris2[,1] > mean(iris2[,1], na.rm=TRUE), iris2[,5], useNA = "ifany")
dimnames(tab2) = list(c("SL < Media", "SL > Media", "NA"), levels(iris2$Species))
addmargins(prop.table(tab2, 2), 1)
```

##	setosa	versicolor	virginica
## SL < Media	0.92	0.46	0.12
## SL > Media	0.00	0.46	0.84
## NA	0.08	0.08	0.04
## Sum	1.00	1.00	1.00

Visualización gráfica de una tabla

```
plot(tab2)
```



Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas

- **Modificar y operar con tablas**

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Atributos de una tabla

Ejemplo:

```
tab3 <- table(iris[,5]); tab3

##
##      setosa versicolor  virginica
##      50         50         50

attributes(tab3)

## $dim
## [1] 3
##
## $dimnames
## $dimnames[[1]]
## [1] "setosa"      "versicolor" "virginica"
##
##
## $class
## [1] "table"

tab3[1]

## setosa
##      50
```

Modificando Tabla

Ejemplo:

```
tab3

##
##      setosa versicolor  virginica
##      50          50          50

dimnames(tab3)[[1]] <- paste(dimnames(tab3)[[1]], "iris", sep = "_")
tab3 <- as.matrix(tab3)
colnames(tab3) <- "Frecuencia"
tab3 <- as.table(tab3)
tab3

##              Frecuencia
## setosa_iris           50
## versicolor_iris       50
## virginica_iris        50

class(tab3)

## [1] "table"
```

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Guardando datos desde R

En R hay múltiples maneras de guardar datos de una sesión de trabajo. Lo más frecuente son las funciones `write..` o `save....` Pero también hay funciones para guardar resultados de operaciones en R como `sink` o `capture.output`.

Ejemplo con `sink()`

```
sink("salidas.txt")      # inicio documento en el que se van a guardar resultados

modelo <- lm(iris[,1] ~ iris[,2], data = iris)
summary(modelo)

sink()                   # cierro el documnto
```

Exportar bases de datos

Archivo	Función	paquete
txt	<code>write.table()</code>	utils
csv	<code>write.table()</code>	utils
	<code>write.csv()</code>	readr
excel	<code>write.xlsx()</code>	xlsx
ods	<code>write_ods()</code>	readODS
Stata	<code>write.dta()</code>	foreign
	<code>save.dta13()</code>	readstata13
	<code>write_dta()</code>	haven
spss	<code>write.foreign()</code>	foreign
	<code>write.sav()</code>	haven

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 `by()`

6 `aggregate()`

7 Funciones de estadística descriptiva básicas

8 `with()` y `attach()`

by()

Esta función permite aplicar una función a un marco de datos dividido por factores.

Ejemplo:

```
by(iris[,1], iris[,5], mean)

## iris[, 5]: setosa
## [1] 5.006
## -----
## iris[, 5]: versicolor
## [1] 5.936
## -----
## iris[, 5]: virginica
## [1] 6.588

mod <- with(iris, by(iris, Species,
                     function(x)lm(Sepal.Length ~ Petal.Length, data = x)))
sapply(mod, coef)

##               setosa versicolor virginica
## (Intercept)  4.2131682   2.407523  1.0596591
## Petal.Length 0.5422926   0.828281  0.9957386
```

```
by(iris[,1:4], iris[,5], summary)
```

```
## iris[, 5]: setosa
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
## Min.	:4.300	Min. :2.300	Min. :1.000	Min. :0.100
## 1st Qu.:	4.800	1st Qu.:3.200	1st Qu.:1.400	1st Qu.:0.200
## Median :	5.000	Median :3.400	Median :1.500	Median :0.200
## Mean :	5.006	Mean :3.428	Mean :1.462	Mean :0.246
## 3rd Qu.:	5.200	3rd Qu.:3.675	3rd Qu.:1.575	3rd Qu.:0.300
## Max. :	5.800	Max. :4.400	Max. :1.900	Max. :0.600

```
## -----
```

```
## iris[, 5]: versicolor
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
## Min.	:4.900	Min. :2.000	Min. :3.00	Min. :1.000
## 1st Qu.:	5.600	1st Qu.:2.525	1st Qu.:4.00	1st Qu.:1.200
## Median :	5.900	Median :2.800	Median :4.35	Median :1.300
## Mean :	5.936	Mean :2.770	Mean :4.26	Mean :1.326
## 3rd Qu.:	6.300	3rd Qu.:3.000	3rd Qu.:4.60	3rd Qu.:1.500
## Max. :	7.000	Max. :3.400	Max. :5.10	Max. :1.800

```
## -----
```

```
## iris[, 5]: virginica
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
## Min.	:4.900	Min. :2.200	Min. :4.500	Min. :1.400
## 1st Qu.:	6.225	1st Qu.:2.800	1st Qu.:5.100	1st Qu.:1.800
## Median :	6.500	Median :3.000	Median :5.550	Median :2.000
## Mean :	6.588	Mean :2.974	Mean :5.552	Mean :2.026
## 3rd Qu.:	6.900	3rd Qu.:3.175	3rd Qu.:5.875	3rd Qu.:2.300
## Max. :	7.900	Max. :3.800	Max. :6.900	Max. :2.500

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

aggregate()

Esta función permite aplicar funciones a conjuntos de datos:

Ejemplo:

```
aggregate(iris[,1:4], list(iris[,5]), FUN = mean)
```

##	Group.1	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
## 1	setosa	5.006	3.428	1.462	0.246
## 2	versicolor	5.936	2.770	4.260	1.326
## 3	virginica	6.588	2.974	5.552	2.026

```
aggregate(iris[,1:4], list(iris[,5]), FUN = min)
```

##	Group.1	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
## 1	setosa	4.3	2.3	1.0	0.1
## 2	versicolor	4.9	2.0	3.0	1.0
## 3	virginica	4.9	2.2	4.5	1.4

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

Funciones de estadística descriptiva básicas

Función	Descripción
<code>mean()</code>	Media
<code>median()</code>	Mediana
<code>var()</code>	Varianza
<code>sd()</code>	Desvío estándar
<code>cor(x,y)</code>	Correlación
<code>cov(x,y)</code>	Covarianza
<code>range()</code>	Rango
<code>quantile()</code>	Cuantiles (0, 25, 50, 75, 100)
<code>fivenum()</code>	Cuantiles (0, 25, 50, 75, 100)
<code>IQR()</code>	Rango intercuartílico
<code>summary()</code>	Resumen estadístico

Estructura de la presentación

1 Marco de Datos

- Estructura
- Atributos
- Creación
- Cargar
- Indexación
- Creación de variables
- Unir bases de datos

2 NA

- Identificar

- Visualizar

3 Tablas

- Creación de Tablas
- Modificar y operar con tablas

4 Exportar marcos de datos o tablas

5 by()

6 aggregate()

7 Funciones de estadística descriptiva básicas

8 with() y attach()

with()

Ejemplo:

```
table(cars$speed[cars$dist > 80], cars$dist[cars$dist > 80])
```

```
##  
##      84 85 92 93 120  
## 18  1  0  0  0  0  
## 24  0  0  1  1  1  
## 25  0  1  0  0  0
```

```
with(cars, table(speed[dist > 80], dist[dist > 80]))
```

```
##  
##      84 85 92 93 120  
## 18  1  0  0  0  0  
## 24  0  0  1  1  1  
## 25  0  1  0  0  0
```

attach()

Ejemplo:

```
names(cars)

## [1] "speed" "dist"

speed

## Error in eval(expr, envir, enclos): objeto 'speed' no encontrado

cars$speed

## [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14
## [24] 15 15 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 22 23 24
## [47] 24 24 24 25

search()

## [1] ".GlobalEnv"          "package:bindrcpp"    "package:knitr"
## [4] "package:stats"       "package:graphics"   "package:grDevices"
## [7] "package:utils"       "package:datasets"   "package:methods"
## [10] "Autoloads"           "package:base"
```

attach()

```
attach(cars)
search()

## [1] ".GlobalEnv"          "cars"                  "package:bindrcpp"
## [4] "package:knitr"        "package:stats"         "package:graphics"
## [7] "package:grDevices"    "package:utils"         "package:datasets"
## [10] "package:methods"     "Autoloads"             "package:base"

speed

## [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14
## [24] 15 15 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 22 23 24
## [47] 24 24 24 25

detach(cars)
search()

## [1] ".GlobalEnv"          "package:bindrcpp"      "package:knitr"
## [4] "package:stats"        "package:graphics"      "package:grDevices"
## [7] "package:utils"        "package:datasets"      "package:methods"
## [10] "Autoloads"            "package:base"
```