FACULTAD DE MATEMÁTICAS
PONTIFICIA UNIVERSIDAD
CATÓLICA DE CHILE

EYP1113 - Probabilidad y Estadística

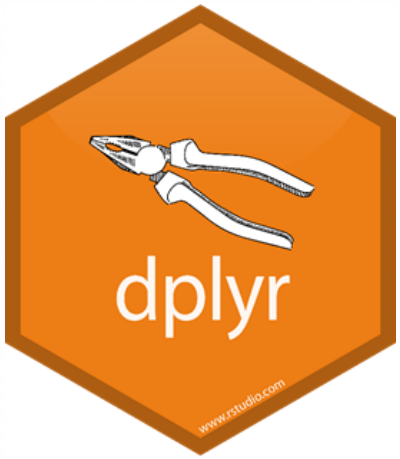
Laboratorio 06

Pilar Tello Hernández
pitello@uc.cl

Facultad de Matemáticas
Departamento de Estadística
Pontificia Universidad Católica de Chile

Segundo Semestre 2021

dplyr



El paquete `dplyr` es principalmente un conjunto de funciones diseñadas para permitir la manipulación de macros de datos de una manera intuitiva y fácil de usar. Sirve para transformar conjuntos de datos existentes en un formato más adecuado para algún tipo particular de análisis o visualización de datos.

Instalando `dplyr`

Para instalar este paquete entonces hay que ejecutar:

```
install.packages("dplyr")
```

Luego para cargar el paquete debemos ejecutar:

```
library(dplyr)
```

Para poder ir ejemplificando utilizaremos la base de datos `RentasMunich.dta`, que ya conocemos. Luego ejecutaremos:

```
library(rio)
data <- import("RentasMunich.dta")
```

RentasMunich.dta

La data contiene 3082 observaciones correspondientes a rentas de apartamento en Munich durante 1999.
Sus variables son:

- rent = valor renta
- rentsqm = renta por metro cuadrado
- area = tamaño/superficie en metros cuadrados
- yearc = año de construcción
- location = localización: 1 Promedio; 2 Bueno; 3 Alta
- bath = tipo de baño: 0 Estándar; 1 Premium
- kitchen = tipo de cocina: 0 Estándar; 1 Premium
- cheating = presencia de calefacción: 0 No; 1 Sí
- district = distrito de Munich donde el apartamento está localizado.

filter

La función `filter` permite seleccionar un subconjunto de filas de los datos aplicando ciertos filtros. Lo primero que se le entrega a la función es la base de datos y luego podemos ir filtrando en base a los criterios que se requieran. Las variables se llaman directamente por su nombre, puesto que la función lee de la base de datos. En nuestra base un ejemplo es el siguiente:

```
filter(data, bano=="Premium", cocina=="Premium",
localizacion=="Alta")
```

slice

La función `slice` permite seleccionar un subconjunto de filas de los datos según la posición de las filas. Lo primero que se le entrega a la función es la base de datos y luego la posición de las filas que se quieren conservar. En nuestra base un ejemplo es el siguiente:

```
slice(data,1:20)
```

arrange

La función arrange permite ordenar el conjunto de datos según el orden de sus columnas. Lo primero que se le entrega a la función es la base de datos y luego las columnas que se quieren ordenar. Por defecto el orden es alfabético y de menor a mayor. En nuestra base un ejemplo es el siguiente:

```
head(arrange(data, cocina), n=10)
head(arrange(data, cocina, bano), n=10)
head(arrange(data, cocina, bano, localizacion), n=10)
head(arrange(data, cocina, bano, localizacion, calefaccion),
n=10)
```

Se puede indicar orden descendente con la función desc(). Un ejemplo:

```
head(arrange(data, desc(cocina)), n=10)
head(arrange(data, desc(cocina), desc(bano)), n=10)
head(arrange(data, desc(cocina), desc(bano),
desc(localizacion)), n=10)
head(arrange(data, desc(cocina), desc(bano), desc(localizacion),
desc(calefaccion)), n=10)
```

select

La función `select` permite obtener un subconjunto de datos según sus columnas. Lo primero que se le entrega a la función es la base de datos y luego las columnas que se quieren mantener. En nuestra base un ejemplo es el siguiente:

```
head(select(data, cocina, bano), n=10)
```

rename

La función `rename` permite renombrar columnas. Lo primero que se le entrega a la función es la base de datos y luego el nuevo nombre de las columnas que se quieren renombrar con su nombre antiguo. En nuestra base un ejemplo es el siguiente, renombraremos la variable `rent` por `renta`:

```
head(rename(data, renta=rent), n=10)
```

distinct

La función `distinct` permite obtener valores únicos de una variable. Se le entrega directamente el vector de datos para el cual estamos buscando los valores únicos. En nuestra base un ejemplo en conjunto con `select` es el siguiente:

```
distinct(select(data,localizacion))
```

mutate

La función `mutate` permite agregar nuevas columnas que son funciones de las columnas existentes. Lo primero que se le entrega a la función es la base de datos y luego el nombre de la nueva columna a agregar y su definición. En nuestra base un ejemplo es el siguiente:

```
head(mutate(data, rentam2=rent/area), n=10)
```

transmute

La función `transmute` es muy similar a la función `mutate`, pero quedándonos sólo con las nuevas columnas que creamos. Lo primero que se le entrega a la función es la base de datos y luego el nombre de la nueva columna a agregar y su definición. En nuestra base un ejemplo es el siguiente:

```
head(transmute(data, rentam2=rent/area), n=10)
```

summarise

La función summarise permite obtener resúmenes de datos de nuestra base de datos. Lo primero que se le entrega a la función es la base de datos y luego el nombre de la medida a resumir junto a su definición. En nuestra base un ejemplo es el siguiente:

```
summarise(data,min_renta=min(rent),
media_renta=mean(rent),
max_renta=max(rent),
sd_renta=sd(rent))
```

sample_n y sample_frac

La función `sample_n` permite obtener un número de filas aleatorias de nuestra base de datos y la función `sample_frac` un porcentaje de filas aleatorias. Lo primero que se le entrega a las funciones es la base de datos y luego el número de filas que se quieren obtener o el porcentaje entre 0 y 1. Por defecto la muestra obtenida es sin reemplazo pero en el caso en que se quisiera una muestra con reemplazo debemos agregar `replace=TRUE` a los argumentos. En nuestra base un ejemplo es el siguiente:

```
sample_n(data, size=10)
sample_frac(data, size=0.003)
```

Operador Pipe

El operador "Pipe" %>% permite realizar múltiples funciones y operaciones dentro de la base de datos. Esto permite resumir líneas de código y ocupar menos espacio en la memoria, porque para aplicar cada función recién aprendida debemos ir escribiéndola en alguna parte. Lo primero que debemos entregar es la base de datos y luego separadas por %>% las operaciones a realizar. Un ejemplo con nuestra base de datos:

```
data2 <- data %>% filter(rent>600) %>%
sample_n(size=10) %>% arrange(desc(rent))
data2
```

group_by

La función group_by permite agrupar nuestra base de datos en conjunto al operador %>%. Con esto se eligen las variables por las que queremos agrupar y es muy útil para obtener estadísticas de resumen en función de una variable o más. Un ejemplo con nuestra base de datos:

```
resumen <- data %>% group_by(bano, cocina) %>%
summarise(n=n(), minimo=min(rent), media=mean(rent),
maximo=max(rent), sd=sd(rent))
resumen
```

Funciones útiles en R

Dos funciones muy útiles en R son `which()` y el operador `%in%`.

La función `which()` sirve para obtener los índices de las filas de una base de datos que cumplan alguna condición dada.

El operador `%in%` indica si un valor o los componentes de un vector se encuentran dentro de los valores de otro vector. Esto nos retorna un valor o vector con valores booleanos para cada componente de éste.

Un ejemplo con nuestra base de datos:

```
which(data$bano=="Premium" & data$cocina=="Premium" &
data$localizacion=="Alta")
head(data$localizacion %in% c("Bueno","Alta"),n=10)
```