Manipulación de datos en R

Programación para el análisis de datos

Departamento de Ciencias Sociales, UCU - Martín Opertti

Repaso explorar datos

Data de la nba

- Vamos a trabajar con un dataframe que contiene los resultados de todos los partidos jugados por equipos de la NBA en las últimas temporadas. Por un detalle de qué es cada variable, ver el siguiente enlace
- Cada observación (fila) es un partido
- Las variables incluyen nombre del equipo local y visitante, los puntos que anotó cada equipo y otros datos del partido como las asistencias y los rebotes que obtuvo cada equipo.

Data de la nba

Ahora importemos y exploremos el dataframe como hicimos la clase anterior:

```
# Importo desde .csv
nba_data <- read_csv("data/nba_data.csv") %>%
  janitor::clean_names()
```

```
glimpse(nba_data)
```

```
## Rows: 23,520
## Columns: 23
## $ game_date_est    <date> 2020-12-19, 2020-12-19, 2020-12-19, 2020-12-18, 2020~
## $ game_id          <dbl> 12000047, 12000048, 12000049, 12000039, 12000040, 120~
## $ game_status_text <chr> "Final", "Final", "Final", "Final", "Final", "Final",~
## $ home_team_id     <dbl> 1610612753, 1610612764, 1610612763, 1610612754, 16106~
## $ visitor_team_id  <dbl> 1610612766, 1610612765, 1610612737, 1610612755, 16106~
## $ season           <dbl> 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,~
## $ team_id_home     <dbl> 1610612753, 1610612764, 1610612763, 1610612754, 16106~
## $ pts_home         <dbl> 120, 99, 116, 107, 105, 119, 89, 127, 103, 129, 113, ~
## $ fg_pct_home      <dbl> 0.433, 0.427, 0.400, 0.371, 0.380, 0.513, 0.348, 0.51~
## $ ft_pct_home      <dbl> 0.792, 0.625, 0.744, 0.692, 0.737, 0.788, 0.810, 0.61~
## $ fg3_pct_home     <dbl> 0.425, 0.295, 0.396, 0.262, 0.356, 0.517, 0.178, 0.36~
## $ ast_home         <dbl> 23, 24, 21, 19, 27, 27, 18, 25, 21, 30, 26, 26, 18, 2~
## $ reb_home         <dbl> 50, 45, 43, 45, 37, 41, 48, 51, 51, 53, 46, 53, 42, 4~
## $ team_id_away     <dbl> 1610612766, 1610612765, 1610612737, 1610612755, 16106~
## $ pts_away         <dbl> 117, 96, 117, 113, 117, 83, 113, 113, 105, 96, 114, 1~
## $ fg_pct_away      <dbl> 0.444, 0.402, 0.422, 0.533, 0.534, 0.395, 0.432, 0.42~
## $ ft_pct_away      <dbl> 0.864, 0.647, 0.837, 0.629, 0.741, 0.611, 0.778, 0.90~
## $ fg3_pct_away     <dbl> 0.439, 0.326, 0.297, 0.355, 0.514, 0.387, 0.457, 0.25~
## $ ast_away         <dbl> 21, 18, 24, 23, 30, 20, 26, 21, 27, 17, 25, 32, 20, 1~
## $ reb_away         <dbl> 52, 51, 47, 48, 51, 35, 53, 44, 56, 42, 33, 43, 42, 4~
## $ home_team_wins   <dbl> 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,~
## $ home_team        <chr> "Orlando Magic", "Washington Wizards", "Memphis Grizz~
## $ visitor_team     <chr> "Charlotte Hornets", "Detroit Pistons", "Atlanta Hawk~
```

Transformar datos

Transformar datos con dplyr

El paquete dplyr contiene funciones muy útiles para la transformación de dataframes (tibbles). Todas las funciones tienen en común que su primer argumento es un dataframe y que devuelven un dataframe. Algunas de las funciones que vamos a ver:

- `filter()`: filtrar observaciones en base a valores
- `select()`: filtrar variables
- `rename()`: renombrar variables
- `arrange()`: ordena los valores según variables
- `distinct()`: extraer valores únicos
- `summarise()`: colapsa valores según alguna fórmula (sumar, número de casos, media, etc.)
- `group_by()`: define grupos de valores utilizar las otras funciones

Filtrar

Una de las tareas más comunes en el análisis de datos es filtrar observaciones en base a condiciones. Existen muchas maneras de filtrar datos en R, la función `filter()` de dplyr es una de las más sencillas de utilizar. El primer argumento es el dataframe y el segundo la condición por la que queremos filtrar.

```
# Tenemos datos de muchas temporadas:
table(nba_data$season)
```

```
##
## 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
## 1385 1362 1432 1419 1411 1425 1424 1422 1104 1420 1427 1418 1416 1405 1382 1378
## 2019 2020
## 1241   49
```

```
# Filtremos para quedarnos con la temporada 2018 solamente
nba_data_19 <- filter(nba_data, season == 2019)
table(nba_data_19$season)
```

```
##
## 2019
## 1241
```

Filtrar

Utilizando operadores lógicos podemos filtrar de formas más complejas:

```
# Todas las temporadas menos la 2020
nba_data_03_19 <- filter(nba_data, season != 2020)
table(nba_data_03_19$season)
```

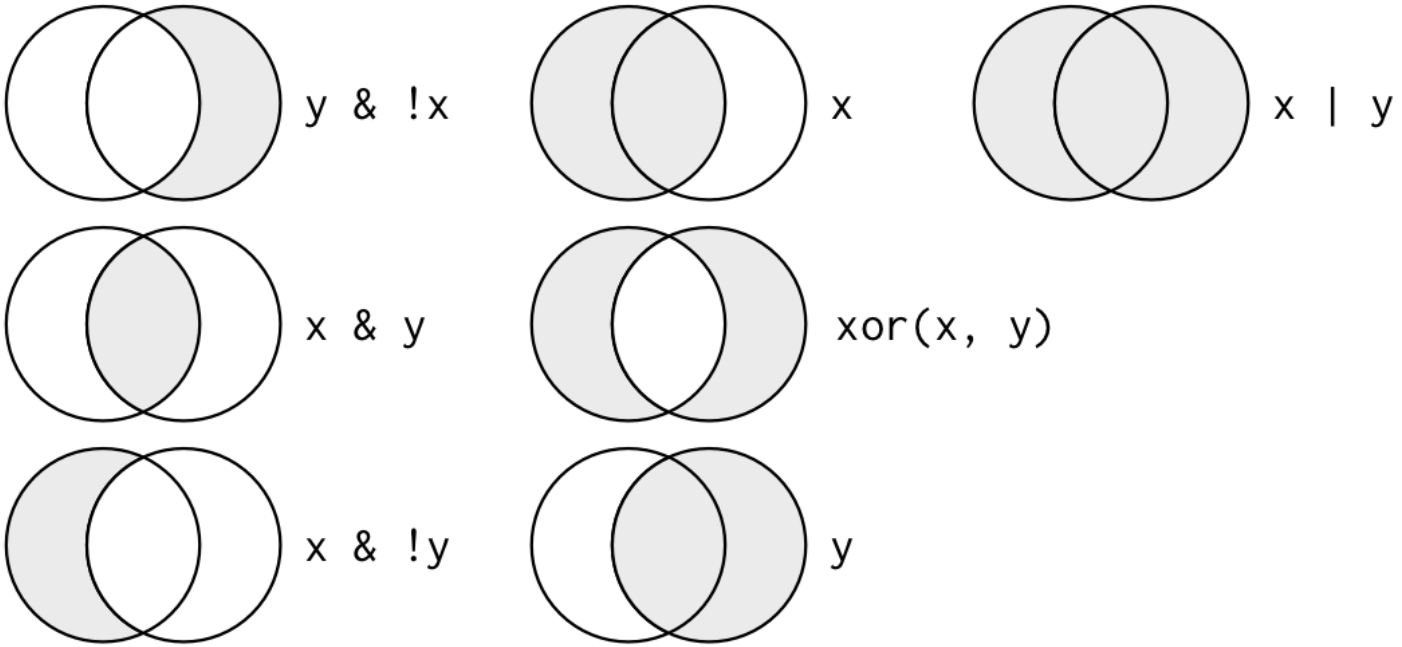
```
##
## 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
## 1385 1362 1432 1419 1411 1425 1424 1422 1104 1420 1427 1418 1416 1405 1382 1378
## 2019
## 1241
```

```
# Solo las temporadas 2005, 2010, 2012 y 2017
temporadas <- c(2005, 2010, 2012, 2017)
nba_data_temp <- filter(nba_data, season %in% temporadas)

table(nba_data_temp$season)
```

```
##
## 2005 2010 2012 2017
## 1432 1422 1420 1382
```

Filtrar (operadores lógicos)



Fuente: Grolemund, G., & Wickham, H. (2018).

Filtrar (NA)

También podemos usar las funciones que identifican datos perdidos:

```
# No tenemos datos de rebotes para algunos partidos...

# Para extraer los casos con datos perdidos en la variable reb_home
data_incompleta <- filter(nba_data, is.na(reb_home))
dim(data_incompleta)
```

```
## [1] 99 23
```

```
# Para extraer con los casos que tienen datos en reb_home
data_completa_reb <- filter(nba_data, !is.na(reb_home))
dim(data_completa_reb)
```

```
## [1] 23421    23
```

Datos duplicados

En ocasiones queremos chequear los valores únicos de una variable. La función `distinct()` de dplyr nos devuelve (por defecto) los valores únicos de una variable

```
print(data)
```

```
##   id color letra
## 1  1  Azul     A
## 2  2  Azul     B
## 3  2  Azul     B
## 4  2  Azul     C
## 5  3  Rojo     D
## 6  4 Verde     E
```

```
# Filtro para obtener datos no duplicados en la variable de id únicamente
# Por defecto devuelve solo los valores de la variable especificada
distinct(data, id)
```

```
##   id
## 1  1
## 2  2
## 3  3
## 4  4
```

Datos duplicados

También podemos usar `distinct()` con el argumento `.keep_all = TRUE` para obtener los casos únicos de una variable pero manteniendo el resto del daaframe

```
# Filtro para obtener datos no duplicados en la variable de id únicamente, con
# el argumento .keep_all = TRUE para mantener el resto de las variables
distinct(data, id, .keep_all = TRUE)
```

```
##   id color letra
## 1  1  Azul     A
## 2  2  Azul     B
## 3  3  Rojo     D
## 4  4 Verde     E
```

```
# Filtro para obtener datos no duplicados
# (excluye las filas idénticas en todas las variables)
distinct(data, .keep_all = TRUE)
```

```
##   id color letra
## 1  1  Azul     A
## 2  2  Azul     B
## 3  2  Azul     C
## 4  3  Rojo     D
## 5  4 Verde     E
```

Seleccionar variables

Con `select()` podemos seleccionar las variables (columnas) que queremos mantener en un dataframe. Podemos nombrarlas, seleccionar cuáles queremos eliminar y referirnos por su orden:

```
# Selccionar un conjunto de variables
select(nba_data, pts_home, pts_away)

# Selccionar todas las variables menos las especificadas
select(nba_data, -pts_home)

# Seleccionar un rango de variables según orden
select(nba_data, game_date_est:visitor_team_id)
select(nba_data, 1:10) # Orden numérico
```

Seleccionar variables

El paquete dplyr también contiene un conjunto de helpers para seleccionar variables de forma efectiva por su posición o patrones de texto:

- `starts_with()`: variables que empiezan con término
- `ends_with()`: variables que terminan con término
- `contains()`: variables que contienen cierto término

```
# Se utilizan dentro del select

# Por ejemplo, seleccionemos todas las variables que terminen en home
data_ej <- select(nba_data, ends_with("home"))
colnames(data_ej)
```

```
## [1] "team_id_home" "pts_home"     "fg_pct_home"  "ft_pct_home"  "fg3_pct_hom
## [6] "ast_home"     "reb_home"
```

Repasemos

Supongamos que queremos realizar varias de las operaciones que hemos visto a un dataframe. Por ejemplo, supongamos que queremos un dataframe que solo incluya partidos de los Chicago Bulls, sin datos perdidos, y que simplemente contenga la fecha, el nombre y los puntos anotados de los dos equipos.

```
data_bulls <- filter(nba_data,
                     home_team == "Chicago Bulls" | visitor_team == "Chicago Bulls")
data_bulls <- drop_na(data_bulls)
data_bulls <- select(data_bulls,
                     game_date_est, home_team, visitor_team, pts_home, pts_away)
print(data_bulls)
```

```
## # A tibble: 1,563 x 5
##    game_date_est home_team              visitor_team        pts_home pts_away
##    <date>        <chr>                  <chr>                  <dbl>    <dbl>
##  1 2020-12-18    Oklahoma City Thunder  Chicago Bulls            103      105
##  2 2020-12-16    Oklahoma City Thunder  Chicago Bulls            103      124
##  3 2020-12-13    Chicago Bulls          Houston Rockets          104       91
##  4 2020-12-11    Chicago Bulls          Houston Rockets          104      125
##  5 2020-03-10    Chicago Bulls          Cleveland Cavaliers      108      103
##  6 2020-03-08    Brooklyn Nets          Chicago Bulls            110      107
##  7 2020-03-06    Chicago Bulls          Indiana Pacers           102      108
##  8 2020-03-04    Minnesota Timberwolves Chicago Bulls            115      108
##  9 2020-03-02    Chicago Bulls          Dallas Mavericks         109      107
## 10 2020-02-29    New York Knicks        Chicago Bulls            125      115
## # ... with 1,553 more rows
```

Pipeline %>%

Un enfoque más sencillo es utilizar el pipeline. Como vimos, la mayoría de las funciones de dplyr que se aplican a un dataframe tienen como primer argumento el dataframe al que le queremos aplicar la función. Con el pipeline especificamos el dataframe solamente una vez al principio, y luego todas las funciones que vamos utilizando no necesitan especificación. De esta forma nos enfocamos en la transformación y no en el objeto.

```
data_bulls_pip <- nba_data %>%
  filter(home_team == "Chicago Bulls" | visitor_team == "Chicago Bulls") %>%
  drop_na() %>%
  select(game_date_est, home_team, visitor_team, pts_home, pts_away)
print(data_bulls)
```

```
## # A tibble: 1,563 x 5
##    game_date_est home_team              visitor_team        pts_home pts_away
##    <date>        <chr>                  <chr>                  <dbl>    <dbl>
##  1 2020-12-18    Oklahoma City Thunder  Chicago Bulls            103      105
##  2 2020-12-16    Oklahoma City Thunder  Chicago Bulls            103      124
##  3 2020-12-13    Chicago Bulls          Houston Rockets          104       91
##  4 2020-12-11    Chicago Bulls          Houston Rockets          104      125
##  5 2020-03-10    Chicago Bulls          Cleveland Cavaliers      108      103
##  6 2020-03-08    Brooklyn Nets          Chicago Bulls            110      107
##  7 2020-03-06    Chicago Bulls          Indiana Pacers           102      108
##  8 2020-03-04    Minnesota Timberwolves Chicago Bulls            115      108
##  9 2020-03-02    Chicago Bulls          Dallas Mavericks         109      107
## 10 2020-02-29    New York Knicks        Chicago Bulls            125      115
## # ... with 1,553 more rows
```

Pipeline %>%

- Una de las ventajas del Tidyverse es la facilidad con la que se puede leer e interpretar el código. Un elemento fundamental para esto es el pipeline (`%>%`). Es muy útil para expresar una secuencia de muchas operaciones.
- Habíamos visto varias formas de realizar esto: sobrescribir el mismo objeto, con objetos intermedios o anidando funciones.
- El pipeline del paquete magrittr hace más fácil modificar operaciones puntuales dentro de conjunto de operaciones, hace que sea más fácil leer (evitando leer de adentro hacia afuera) entre otras ventajas.
- Es recomendable evitar usar el pipeline cuando queremos trabajar más de un objeto a la vez
- `x %>% f == f(x)`
- Se puede leer como un "y entonces"

Ordenar datos

Para ordenar datos (numérica o alfabéticamente) podemos usar la función `arrange()`. Por defecto, `arrange()` ordena ascendentemente, con `desc()` podemos cambiar eso.

```
# Con el pipeline, seleccionemos algunas variables y luego ordenemos
nba_data %>%
  filter(home_team == "Chicago Bulls" |
           visitor_team == "Chicago Bulls") %>%
  select(game_date_est, home_team, visitor_team, pts_home, pts_away) %>%
  arrange(pts_home)
```

```
## # A tibble: 1,570 x 5
##    game_date_est home_team            visitor_team    pts_home pts_away
##    <date>        <chr>                <chr>              <dbl>    <dbl>
##  1 2012-03-19    Orlando Magic        Chicago Bulls         59       85
##  2 2003-10-08    Indiana Pacers       Chicago Bulls         62       58
##  3 2012-02-10    Charlotte Hornets    Chicago Bulls         64       95
##  4 2013-05-13    Chicago Bulls        Miami Heat            65       88
##  5 2006-10-31    Miami Heat           Chicago Bulls         66      108
##  6 2003-11-03    Chicago Bulls        Houston Rockets       66       98
##  7 2015-04-30    Milwaukee Bucks      Chicago Bulls         66      120
##  8 2013-02-21    Chicago Bulls        Miami Heat            67       86
##  9 2012-02-08    New Orleans Pelicans Chicago Bulls         67       90
## 10 2006-10-19    San Antonio Spurs    Chicago Bulls         67       99
## # ... with 1,560 more rows
```

Ordenar datos

```
#Solo por pts_home ascendente
nba_data %>%
  arrange(pts_home)

# Solo por pts_home descendente
nba_data %>%
  arrange(desc(pts_home))

# Primero por equipo local (alfabeticamente) y después por puntos
nba_data %>%
  arrange(home_team, pts_home)
```

Renombrar variables con rename()

Con la función `rename()` podemos renombrar las variables de un dataframe.

```
nba_data_2 <- nba_data %>%
  rename(season_year = season)

colnames(nba_data_2)
```

```
##  [1] "game_date_est"    "game_id"          "game_status_text" "home_team_id
##  [5] "visitor_team_id"  "season_year"      "team_id_home"     "pts_home"
##  [9] "fg_pct_home"      "ft_pct_home"      "fg3_pct_home"     "ast_home"
## [13] "reb_home"         "team_id_away"     "pts_away"         "fg_pct_away"
## [17] "ft_pct_away"      "fg3_pct_away"     "ast_away"         "reb_away"
## [21] "home_team_wins"   "home_team"        "visitor_team"
```

Extraer valores únicos con distinct() y pull()

La función `distinct()` filtra las observaciones unicas (puede especificarse una variable sola, ver argumento `.keep_all`), la función `pull()` toma una columna de un dataframe y la extrae como vector. La combinación de ambas funciones es muy útil cuando queremos extraer por ejemplo valores únicos de una variable.

```
# Solo con distinct() devuelve un dataframe
nba_data_2 <- nba_data %>%
  distinct(home_team)
nba_data_2
```

```
## # A tibble: 30 x 1
##    home_team
##    <chr>
##  1 Orlando Magic
##  2 Washington Wizards
##  3 Memphis Grizzlies
##  4 Indiana Pacers
##  5 Toronto Raptors
##  6 New York Knicks
##  7 Boston Celtics
##  8 New Orleans Pelicans
##  9 Oklahoma City Thunder
## 10 Denver Nuggets
## # ... with 20 more rows
```

Extraer valores únicos con distinct() y pull()

```
# Extraemos los distintos valores como vector
nba_data %>%
  distinct(home_team) %>%
  pull()
```

```
##  [1] "Orlando Magic"          "Washington Wizards"     "Memphis Grizzlies"
##  [4] "Indiana Pacers"         "Toronto Raptors"        "New York Knicks"
##  [7] "Boston Celtics"         "New Orleans Pelicans"   "Oklahoma City Thunder"
## [10] "Denver Nuggets"         "Phoenix Suns"           "Houston Rockets"
## [13] "Dallas Mavericks"       "Sacramento Kings"       "Los Angeles Clippers"
## [16] "Philadelphia 76ers"     "Cleveland Cavaliers"    "Charlotte Hornets"
## [19] "Miami Heat"             "Milwaukee Bucks"        "Minnesota Timberwolves"
## [22] "Utah Jazz"              "Atlanta Hawks"          "Brooklyn Nets"
## [25] "Detroit Pistons"        "Chicago Bulls"          "Los Angeles Lakers"
## [28] "Portland Trail Blazers" "San Antonio Spurs"      "Golden State Warriors"
```

Otras funciones de dplyr muy útiles

- `slice_min()` y `slice_max()`: filtrar n observaciones de mayor o menor valor según variable. En general, la familia de funciones slice permite filtrar observaciones en función de su posición.
- `count()` contar observaciones por grupo
- `rowise()` para realizar operaciones por fila
- `relocate()` cambiar el orden de columnas

Ejercicio

Utilizando la base nba_data, crear un dataframe que contenga 3 variables: puntos, asistencias y rebotes (con esos nombres) de todos los partidos de local de los Dallas Mavericks en la temporada 2011, ordenar de forma descendentepor la cantidad de puntos y extraer los 5 partidos con más puntos. Usar el pipeline

Resumenes y tablas

Resumir datos

Resumir datos o crear tablas descriptivas es una de las partes fundamentales del análisis de datos. Para ello utilizaremos la función `summarise()`, muchas veces en conjunto con `group_by()`.

Esencialmente `summarise()` resume un dataframe en una fila según una estadística especificada. Por ejemplo, calculando la media de una variable

```
nba_data %>%
  drop_na() %>%
  summarise(media = mean(pts_home))
```

```
## # A tibble: 1 x 1
##   media
##   <dbl>
## 1  102.
```

```
# Por ahora no hay mucha diferencia con
mean(nba_data$pts_home, na.rm = TRUE)
```

```
## [1] 102.2834
```

Resumir datos

Hasta ahora `summarise()` no nos es de gran utilidad, la utilidad de `summarise()` es su uso conjunto con `group_by()`, para estimar diferentes estadísticas según grupos específicos.

Cuando utilizamos `group_by()` en un pipeline cambiamos la unidad de análisis desde todo el dataframe a niveles de una variable. Retomando el ejemplo, podemos ver el promedio de puntos por temporada:

```
nba_data %>%
  group_by(season) %>%
  summarise(media = mean(pts_home, na.rm = T)) %>%
  head()
```

```
## # A tibble: 6 x 2
##   season media
##    <dbl> <dbl>
## 1   2003  94.9
## 2   2004  98.6
## 3   2005  98.4
## 4   2006  99.8
## 5   2007 101.
## 6   2008 101.
```

Resumir datos

Algunas de las operaciones más utilizadas para resumir datos:

- `mean()`: media
- `median()`: mediana
- `sd()`: desvío estandar
- `sum()`: suma
- `n()`: número de observaciones
- `n_distinct()`: número de valores únicos
- `min()` y `max()`: mínimo y máximo

Resumir datos

Siempre debemos especificar los grupos con `group_by()` antes de utilizar `summarise()`. Si queremos seguir realizando operaciones luego de `summarise()`, y no queremos que estas operaciones estén agrupadas utilizamos `ungroup()`.

```
nba_data %>%
  drop_na() %>%
  group_by(season) %>%
  summarise(media = mean(pts_home)) %>%
  ungroup() %>%
  filter(season > 2010)
```

```
## # A tibble: 10 x 2
##    season media
##     <dbl> <dbl>
##  1   2011  97.4
##  2   2012  99.4
##  3   2013 102.
##  4   2014 101.
##  5   2015 104.
##  6   2016 107.
##  7   2017 107.
##  8   2018 112.
##  9   2019 112.
## 10   2020 110.
```

Resumir datos

Para obtener una tabla de frecuencias utilizando este enfoque utilizamos dentro de `summarise()` la función `n()`. También podemos utilizar el equivalente (y más corto) `count()`

```
nba_data %>%
  group_by(season) %>%
  summarise(temporadas = n()) # No lleva argumento
```

```
## # A tibble: 18 x 2
##    season temporadas
##     <dbl>      <int>
##  1   2003       1385
##  2   2004       1362
##  3   2005       1432
##  4   2006       1419
##  5   2007       1411
##  6   2008       1425
##  7   2009       1424
##  8   2010       1422
##  9   2011       1104
## 10   2012       1420
## 11   2013       1427
## 12   2014       1418
## 13   2015       1416
## 14   2016       1405
## 15   2017       1382
## 16   2018       1378
## 17   2019       1241
## 18   2020         49
```

Resumir datos

Para obtener lo mismo que con `summarise()` y `n()` podemos utilizar el equivalente (y más corto) `count()`

```
count(nba_data, season)
```

```
## # A tibble: 18 x 2
##    season     n
##     <dbl> <int>
##  1   2003  1385
##  2   2004  1362
##  3   2005  1432
##  4   2006  1419
##  5   2007  1411
##  6   2008  1425
##  7   2009  1424
##  8   2010  1422
##  9   2011  1104
## 10   2012  1420
## 11   2013  1427
## 12   2014  1418
## 13   2015  1416
## 14   2016  1405
## 15   2017  1382
## 16   2018  1378
## 17   2019  1241
## 18   2020    49
```

Resumir datos

Si queremos calcular la proporción o el porcentaje debemos transformar la frecuencia en porcentaje creando una nueva variable con `mutate()`

```
nba_data %>%
  group_by(season) %>%
  summarise(temporadas = n()) %>%
  mutate(porcentaje = temporadas / sum(temporadas))
```

```
## # A tibble: 18 x 3
##    season temporadas porcentaje
##     <dbl>      <int>      <dbl>
##  1   2003       1385    0.0589
##  2   2004       1362    0.0579
##  3   2005       1432    0.0609
##  4   2006       1419    0.0603
##  5   2007       1411    0.0600
##  6   2008       1425    0.0606
##  7   2009       1424    0.0605
##  8   2010       1422    0.0605
##  9   2011       1104    0.0469
## 10   2012       1420    0.0604
## 11   2013       1427    0.0607
## 12   2014       1418    0.0603
## 13   2015       1416    0.0602
## 14   2016       1405    0.0597
## 15   2017       1382    0.0588
## 16   2018       1378    0.0586
## 17   2019       1241    0.0528
## 18   2020         49    0.00208
```

Resumir datos

Para emprolijar la tabla:

```
nba_data %>%
  group_by(season) %>%
  summarise(temporadas = n()) %>%
  mutate(porcentaje = round((temporadas / sum(temporadas))*100, digits = 1))
```

```
## # A tibble: 18 x 3
##    season temporadas porcentaje
##     <dbl>      <int>      <dbl>
##  1   2003       1385        5.9
##  2   2004       1362        5.8
##  3   2005       1432        6.1
##  4   2006       1419        6
##  5   2007       1411        6
##  6   2008       1425        6.1
##  7   2009       1424        6.1
##  8   2010       1422        6
##  9   2011       1104        4.7
## 10   2012       1420        6
## 11   2013       1427        6.1
## 12   2014       1418        6
## 13   2015       1416        6
## 14   2016       1405        6
## 15   2017       1382        5.9
## 16   2018       1378        5.9
## 17   2019       1241        5.3
## 18   2020         49        0.2
```

Resumir datos

Podemos utilizar más de una variable dentro de `group_by()`. Por ejemplo, calculemos la media de puntos de los Bulls y Knicks (cuando juegan de local) en cada temporada:

```
nba_data %>%
  drop_na() %>%
  filter(home_team == "Chicago Bulls" | home_team == "New York Knicks") %>%
  group_by(season, home_team) %>%
  summarise(media = mean(pts_home))
```

```
## `summarise()` has grouped output by 'season'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 36 x 3
## # Groups:   season [18]
##    season home_team       media
##     <dbl> <chr>           <dbl>
##  1   2003 Chicago Bulls    88.9
##  2   2003 New York Knicks  93.1
##  3   2004 Chicago Bulls    96.2
##  4   2004 New York Knicks  98.7
##  5   2005 Chicago Bulls    98.3
##  6   2005 New York Knicks  98.0
##  7   2006 Chicago Bulls    98.8
##  8   2006 New York Knicks 101.
##  9   2007 Chicago Bulls    98.6
## 10   2007 New York Knicks  97
## # ... with 26 more rows
```

Resumir datos

Una de las grandes ventajas de `summarise()` es que podemos resumir muy fácilmente varias estadísticas en un solo dataframe.

```
nba_data %>%
  filter(season > 2015) %>%
  group_by(season) %>%
  summarise(media_pts_home = mean(pts_home),
            suma_pts_home = sum(pts_home),
            max_pts_home = max(pts_home),
            partidos = n())
```

```
## # A tibble: 5 x 5
##   season media_pts_home suma_pts_home max_pts_home partidos
##    <dbl>          <dbl>         <dbl>        <dbl>    <int>
## 1   2016           107.        150217          149     1405
## 2   2017           107.        148178          149     1382
## 3   2018           112.        154752          161     1378
## 4   2019           112.        139333          158     1241
## 5   2020           110.          5372          131       49
```

Resumir datos de multiples variables

Con `summarise()` y `across()` podemos especificar un tipo de resumen para un conjunto de variables. Por ejemplo, calcular la media para el equipo local de todas las variables que terminan con "pct_home".

```
nba_data %>%
  group_by(home_team) %>%
  summarise(across(ends_with("pct_home"), ~ mean(.x, na.rm = TRUE)))
```

```
## # A tibble: 30 x 4
##    home_team             fg_pct_home ft_pct_home fg3_pct_home
##    <chr>                       <dbl>       <dbl>        <dbl>
##  1 Atlanta Hawks               0.458       0.761        0.349
##  2 Boston Celtics              0.463       0.773        0.355
##  3 Brooklyn Nets               0.447       0.757        0.344
##  4 Charlotte Hornets           0.443       0.752        0.347
##  5 Chicago Bulls               0.442       0.759        0.356
##  6 Cleveland Cavaliers         0.456       0.744        0.358
##  7 Dallas Mavericks            0.463       0.787        0.364
##  8 Denver Nuggets              0.469       0.749        0.350
##  9 Detroit Pistons             0.453       0.734        0.345
## 10 Golden State Warriors       0.474       0.766        0.374
## # ... with 20 more rows
```

Resumir datos de multiples variables

Con `across()` (ver across) es posible especificar conjuntos de variables utilizando helpers como start_with() o ends_with(), con vectores de variables, utilizando operadores o condicionales

```
nba_data %>%
  group_by(home_team) %>%
  summarise(across(c("pts_home", "ast_home"), ~ mean(.x, na.rm = TRUE)))
```

```
## # A tibble: 30 x 3
##    home_team              pts_home ast_home
##    <chr>                     <dbl>    <dbl>
##  1 Atlanta Hawks             101.      23.6
##  2 Boston Celtics            101.      23.5
##  3 Brooklyn Nets              99.1     22.4
##  4 Charlotte Hornets          99.3     22.0
##  5 Chicago Bulls              99.1     22.9
##  6 Cleveland Cavaliers       101.      22.6
##  7 Dallas Mavericks          104.      22.1
##  8 Denver Nuggets            108.      25.3
##  9 Detroit Pistons            99.0     22.5
## 10 Golden State Warriors     109.      25.8
## # ... with 20 more rows
```

Resumir datos

Con `where()` podemos establecer condiciones, como por ejemplo resumir todas las variables numéricas

```
nba_data %>%
  group_by(home_team) %>%
  summarise(across(where(is.numeric), ~ mean(.x, na.rm = TRUE)))
```

```
## # A tibble: 30 x 20
##    home_team   game_id home_team_id visitor_team_id season team_id_home pts_home
##    <chr>         <dbl>        <dbl>            <dbl>  <dbl>        <dbl>    <dbl>
##  1 Atlanta Ha~  2.17e7   1610612737     1610612752.  2011.   1610612737    101.
##  2 Boston Cel~  2.27e7   1610612738     1610612752.  2011.   1610612738    101.
##  3 Brooklyn N~  2.13e7   1610612751     1610612751.  2011.   1610612751     99.1
##  4 Charlotte ~  2.06e7   1610612766     1610612751.  2011.   1610612766     99.3
##  5 Chicago Bu~  2.13e7   1610612741     1610612752.  2011.   1610612741     99.1
##  6 Cleveland ~  2.22e7   1610612739     1610612752.  2011.   1610612739    101.
##  7 Dallas Mav~  2.17e7   1610612742     1610612752.  2011.   1610612742    104.
##  8 Denver Nug~  2.18e7   1610612743     1610612752.  2011.   1610612743    108.
##  9 Detroit Pi~  2.17e7   1610612765     1610612751.  2010.   1610612765     99.0
## 10 Golden Sta~  2.23e7   1610612744     1610612752.  2011.   1610612744    109.
## # ... with 20 more rows, and 13 more variables: fg_pct_home <dbl>,
## #   ft_pct_home <dbl>, fg3_pct_home <dbl>, ast_home <dbl>, reb_home <dbl>,
## #   team_id_away <dbl>, pts_away <dbl>, fg_pct_away <dbl>, ft_pct_away <dbl>,
## #   fg3_pct_away <dbl>, ast_away <dbl>, reb_away <dbl>, home_team_wins <dbl>
```

Ejercicio

Con los datos de los partidos de la base nba_data:

1) Crear una tabla con la cantidad de partidos de local que hay en la base para cada equipo y ordenar de mayor a menor según cantidad de partidos

2) Utilizando solamente los datos de la temporada 2016, crear una tabla resumen con la cantidad de partidos ganados de local por cada equipo

3) Crear una tabla resumen con la cantidad de partidos ganados de visitante por cada equipo para cada temporada de la que tenemos datos

4) Utilizando solamente los datos de la temporada 2016, calcular el promedio de puntos anotados por el equipo local y por el equipo visitante según si ganaron o no ese partido