

# Introduction to R Programming

## Data Wrangling

Pedro Fonseca

07 Maio 2020

# The `subset()` function

- ▶ `subset()` is a generic function that returns subsets of vectors, matrices or data frames which meet logical conditions
- ▶ It is useful shortcut for subsetting data frames
- ▶ `subset()` is meant for interactive data exploration

## The subset() function

```
df <- data.frame(  
  name = c("Yu", "Matt", "Jane", "Tim", "Dave", "Marie"),  
  inc = c(6, 1, 2, NA, 5, 9),  
  gender = factor(c("F", "M", "F", "M", "M", "F")),  
  state = factor(c("AZ", "KS", NA, "CA", "FL", "MA")))
```

df

##	name	inc	gender	state
## 1	Yu	6	F	AZ
## 2	Matt	1	M	KS
## 3	Jane	2	F	<NA>
## 4	Tim	NA	M	CA
## 5	Dave	5	M	FL
## 6	Marie	9	F	MA

## The subset() function

```
subset(df, inc > 4)
```

```
##      name inc gender state
## 1    Yu    6      F    AZ
## 5  Dave    5      M    FL
## 6 Marie    9      F    MA
```

```
subset(df, name == "Marie")
```

```
##      name inc gender state
## 6 Marie    9      F    MA
```

```
subset(df, inc > 4 & name != "Marie")
```

```
##      name inc gender state
## 1    Yu    6      F    AZ
## 5  Dave    5      M    FL
```

## The subset() function

```
subset(df, inc > 4 & name != "Marie" & gender == "F")
```

```
##   name inc gender state  
## 1  Yu   6      F    AZ
```

```
subset(df, inc >= 2 & state %in% c("MA", "FL", "CA"))
```

```
##   name inc gender state  
## 5  Dave   5      M    FL  
## 6 Marie   9      F    MA
```

## The subset() function

You can use the select argument to choose columns:

```
subset(df, inc == 5, select = c(state, name))
```

```
##    state name  
## 5    FL Dave
```

```
subset(df, inc == 5, select = c(1, 4))
```

```
##    name state  
## 5 Dave    FL
```

```
subset(df, inc == 5, select = inc:state)
```

```
##    inc gender state  
## 5    5      M    FL
```

## The subset() function

And also to drop columns:

```
subset(df, inc == 5, select = -c(state, name))
```

```
##   inc gender  
## 5    5      M
```

```
subset(df, inc == 5, select = -c(state, name, gender))
```

```
##   inc  
## 5    5
```

## The subset() function

You can use subset() to filter out missing data with respect to specific variables:

```
subset(df, !is.na(state), select = c(name, inc))
```

```
##      name inc
## 1     Yu   6
## 2    Matt   1
## 4     Tim  NA
## 5    Dave   5
## 6  Marie   9
```



## The subset() function

```
subset(df, !is.na(inc) & !is.na(state),  
       select = c(name, inc, state))
```

```
##      name inc state  
## 1    Yu    6    AZ  
## 2  Matt    1    KS  
## 5  Dave    5    FL  
## 6 Marie    9    MA
```

# The `subset()` function

`subset()`:

- ▶ Also works with vectors, matrices and lists.
- ▶ Doesn't drop dimensions (by default).

In the logical expressions that indicate which rows to keep, missing values are taken as `FALSE`.

## Modifying columns with transform()

transform() can be used to modify the columns of a data frame:

```
transform(df, state = paste0(state, "-US"))
```

```
##      name inc gender state
## 1    Yu    6      F AZ-US
## 2  Matt    1      M KS-US
## 3  Jane    2      F NA-US
## 4   Tim   NA      M CA-US
## 5  Dave    5      M FL-US
## 6 Marie    9      F MA-US
```

## Modifying columns with transform()

Let's change how the levels of the gender factor are displayed:

```
transform(df, gender = factor(  
  gender, labels = c("Female", "Male")))
```

```
##      name inc gender state  
## 1    Yu    6 Female    AZ  
## 2  Matt    1   Male    KS  
## 3  Jane    2 Female  <NA>  
## 4   Tim   NA   Male    CA  
## 5  Dave    5   Male    FL  
## 6 Marie    9 Female    MA
```

## Modifying columns with transform()

Now let's express inc in euros:

```
transform(df, inc = ifelse(is.na(inc), NA,  
                             paste0(inc * 1000, "€")  
                             )  
          )
```

##	name	inc	gender	state
## 1	Yu	6000€	F	AZ
## 2	Matt	1000€	M	KS
## 3	Jane	2000€	F	<NA>
## 4	Tim	<NA>	M	CA
## 5	Dave	5000€	M	FL
## 6	Marie	9000€	F	MA

## Create columns with transform()

Transform() can also be used to create new variables.

Let's create a variable with income in the logarithmic scale:

```
transform(df, logInc = log(inc))
```

##	name	inc	gender	state	logInc
## 1	Yu	6	F	AZ	1.7917595
## 2	Matt	1	M	KS	0.0000000
## 3	Jane	2	F	<NA>	0.6931472
## 4	Tim	NA	M	CA	NA
## 5	Dave	5	M	FL	1.6094379
## 6	Marie	9	F	MA	2.1972246

## Create columns with transform()

Now lets standardize the income column:

```
standardize <- function(x){  
  z <- (x-mean(x, na.rm = TRUE))/sd(x, na.rm = TRUE)  
  round(z, 2)  
}
```

```
transform(df, norm_inc = standardize(inc))
```

##	name	inc	gender	state	norm_inc
## 1	Yu	6	F	AZ	0.44
## 2	Matt	1	M	KS	-1.12
## 3	Jane	2	F	<NA>	-0.81
## 4	Tim	NA	M	CA	NA
## 5	Dave	5	M	FL	0.12
## 6	Marie	9	F	MA	1.37

## Relational models

- ▶ Sometimes our tables are related to other tables.
- ▶ It is often necessary to complement one table with information from another table, or to cross information between tables.
- ▶ We usually join tables by using one or more variables that are present in both tables as a key to match rows from one table to the other.



## A simple relational model

```
set.seed(1)

Sales <- data.frame(
  Product = sample(c("Toaster", "Radio", "TV"),
                  size = 7, replace = TRUE),
  CustomerID = c(rep("1_2019", 2),
                 paste(2:3, "2019", sep = "_"),
                 paste(1:3, "2020", sep = "_")))

Sales$Price <- round(ifelse(
  Sales$Product == "TV", rnorm(1, 400, 20),
  ifelse(Sales$Product == "Toaster",
        rnorm(1, 40, 2), rnorm(1, 35, 2))))
```

## A simple relational model

```
set.seed(1)

Clients <- data.frame(
  CustomerID = c(paste(2:4, "2019", sep = "_"),
                 paste(1:2, "2020", sep = "_")),
  State = sample(c("CA", "AZ", "IL", "MA"),
                 size = 5, replace = TRUE))
```

## A simple relational model

Table 1: Sales

Product	CustomerID	Price
Toaster	1_2019	38
TV	1_2019	407
Toaster	2_2019	38
Radio	3_2019	36
Toaster	1_2020	38
TV	2_2020	407
TV	3_2020	407

Table 2: Clients

CustomerID	State
2_2019	CA
3_2019	MA
4_2019	IL
1_2020	CA
2_2020	AZ

## Joining tables

- ▶ `CustomerID` is present in both tables and uniquely identifies each row of the `Clients` table. We can therefore use it as a key to match rows from one table to another.
- ▶ In R this can be done with the `merge()` function.

## Inner join

The inner join returns only rows that have matching values in both tables:

```
merge(x = Sales, y = Clients,  
      by = "CustomerID")
```

##	CustomerID	Product	Price	State
## 1	1_2020	Toaster	38	CA
## 2	2_2019	Toaster	38	CA
## 3	2_2020	TV	407	AZ
## 4	3_2019	Radio	36	MA

## Natural join

A natural join is an inner join where the joining attributes are defined as having equal names, so they need not be stated explicitly:

```
merge(x = Sales, y = Clients)
```

##	CustomerID	Product	Price	State
## 1	1_2020	Toaster	38	CA
## 2	2_2019	Toaster	38	CA
## 3	2_2020	TV	407	AZ
## 4	3_2019	Radio	36	MA

## Left join

To includes all the rows of x and only those from y that match use  
`all.x = TRUE`:

```
merge(x = Sales, y = Clients,  
      by = "CustomerID",  
      all.x = TRUE)
```

##	CustomerID	Product	Price	State
## 1	1_2019	Toaster	38	<NA>
## 2	1_2019	TV	407	<NA>
## 3	1_2020	Toaster	38	CA
## 4	2_2019	Toaster	38	CA
## 5	2_2020	TV	407	AZ
## 6	3_2019	Radio	36	MA
## 7	3_2020	TV	407	<NA>

## Right join

To include all the rows of y and only those from x that match use `all.y = TRUE`:

```
merge(x = Sales, y = Clients,  
      by = "CustomerID",  
      all.y = TRUE)
```

##	CustomerID	Product	Price	State
## 1	1_2020	Toaster	38	CA
## 2	2_2019	Toaster	38	CA
## 3	2_2020	TV	407	AZ
## 4	3_2019	Radio	36	MA
## 5	4_2019	<NA>	NA	IL



## Full outer join

To keep all rows from both tables use `all = TRUE`.

```
merge(x = Sales, y = Clients,  
      by = "CustomerID",  
      all = TRUE)
```

##	CustomerID	Product	Price	State
## 1	1_2019	Toaster	38	<NA>
## 2	1_2019	TV	407	<NA>
## 3	1_2020	Toaster	38	CA
## 4	2_2019	Toaster	38	CA
## 5	2_2020	TV	407	AZ
## 6	3_2019	Radio	36	MA
## 7	3_2020	TV	407	<NA>
## 8	4_2019	<NA>	NA	IL

## Cross Join

Cartesian product of the two tables. The output has  $\text{nrow}(x) * \text{nrow}(y)$  rows and  $\text{ncol}(x) + \text{ncol}(y)$  columns.

```
merge(x = Sales, y = Clients,  
      by = NULL)
```

## Joining tables

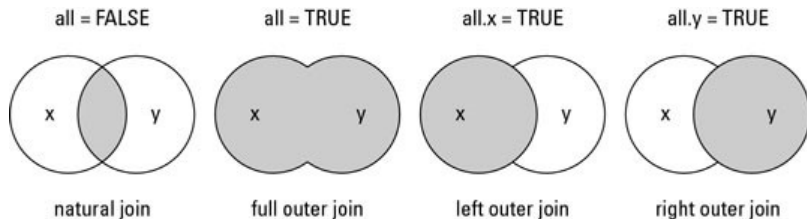


Figure 1: Join operations with Merge()

## Joining tables

- ▶ If the merging key is a combination of more than one column, you can provide a vector to `by`.
- ▶ If the columns used as key have different names in different tables, we need to use `by.x` and `by.y` instead of `by`.
- ▶ If no `by` argument is provided, the tables are merged on the columns with names they both have.
- ▶ `all.x`, `all.y`, and `all` are set to `FALSE` by default. This is why the default join is the natural join.

## The sqldf package

- ▶ You can run SQL queries in R using the sqldf package.
- ▶ SQL queries must be provided to the sqldf() function as strings.

```
library(sqldf)
```

## Inner join with sqldf

```
sqldf("SELECT CustomerID, Product, Price, State  
      FROM Sales  
      JOIN Clients  
      USING(CustomerID)  
      ORDER BY CustomerID")
```

##	CustomerID	Product	Price	State
## 1	1_2020	Toaster	38	CA
## 2	2_2019	Toaster	38	CA
## 3	2_2020	TV	407	AZ
## 4	3_2019	Radio	36	MA

## Left join with sqldf

```
sqldf("SELECT CustomerID, Product, Price, State  
      FROM Sales  
      LEFT JOIN Clients  
      USING(CustomerID)  
      ORDER BY CustomerID")
```

##	CustomerID	Product	Price	State
## 1	1_2019	Toaster	38	<NA>
## 2	1_2019	TV	407	<NA>
## 3	1_2020	Toaster	38	CA
## 4	2_2019	Toaster	38	CA
## 5	2_2020	TV	407	AZ
## 6	3_2019	Radio	36	MA
## 7	3_2020	TV	407	<NA>

## Cross join with sqldf

```
sqldf("SELECT *  
      FROM Sales  
      CROSS JOIN Clients  
      ORDER BY CustomerID")
```



## Subgroup summaries with `aggregate()`

The `aggregate()` function can be used to compute subgroup summary statistics.

## Subgroup summaries with aggregate()

```
my_df <- data.frame(  
  age = c(22, 36, 21, 39, 33, 45, 34, 59),  
  smoker = factor(c("no", "yes", "no", "no", "yes",  
                    "no", "yes", "yes")),  
  child = factor(c("no", "yes", "no", "no", "yes",  
                   "yes", "no", "yes")),  
  income = c(0.8, 1.8, 1.6, 1.5, 2.3, 1.4, 1.8, 1.5),  
  stringsAsFactors = FALSE)
```

## Subgroup summaries with aggregate()

```
my_df
```

##	age	smoker	child	income
## 1	22	no	no	0.8
## 2	36	yes	yes	1.8
## 3	21	no	no	1.6
## 4	39	no	no	1.5
## 5	33	yes	yes	2.3
## 6	45	no	yes	1.4
## 7	34	yes	no	1.8
## 8	59	yes	yes	1.5

## Subgroup summaries with aggregate()

On average, do people with children earn more than people without children?

```
aggregate(formula = income ~ child,  
           data = my_df,  
           FUN = mean)
```

```
##   child income  
## 1    no  1.425  
## 2   yes  1.750
```

## Subgroup summaries with aggregate()

```
aggregate(  
  x = my_df["income"],  
  by = list(child = my_df$child),  
  FUN = mean)
```

```
##   child income  
## 1    no  1.425  
## 2   yes  1.750
```

## Subgroup summaries with aggregate()

On average, do people who smoke earn more than people who don't?

```
aggregate(income ~ smoker, my_df, mean)
```

```
##   smoker income  
## 1      no  1.325  
## 2     yes  1.850
```

## Subgroup summaries with aggregate()

```
aggregate(  
  my_df["income"],  
  list(smoker = my_df$smoker),  
  mean)
```

```
##   smoker income  
## 1      no  1.325  
## 2     yes  1.850
```

## Subgroup summaries with aggregate()

Is the median income higher for smokers or non-smokers?

```
aggregate(income ~ smoker, my_df, median)
```

```
##   smoker income  
## 1     no   1.45  
## 2    yes   1.80
```



## Subgroup summaries with aggregate()

```
aggregate(  
  my_df["income"],  
  list(smoker = my_df$smoker),  
  median)
```

```
##   smoker income  
## 1      no   1.45  
## 2     yes   1.80
```

## Subgroup summaries with aggregate()

What is the lowest income for someone with children? And without?

```
aggregate(income ~ child, my_df, min)
```

```
##   child income  
## 1    no    0.8  
## 2   yes    1.4
```

## Subgroup summaries with aggregate()

```
aggregate(  
  my_df["income"],  
  list(child = my_df$child),  
  min)
```

```
##   child income  
## 1    no    0.8  
## 2   yes    1.4
```

## Subgroup summaries with aggregate()

Is the average age of people with children higher than that of people without children?

```
aggregate(age ~ child, my_df, mean)
```

```
##   child   age  
## 1    no 29.00  
## 2   yes 43.25
```

## Subgroup summaries with aggregate()

```
aggregate(  
  my_df["age"],  
  list(child = my_df$child),  
  mean)
```

```
##   child   age  
## 1    no 29.00  
## 2   yes 43.25
```

## Subgroup summaries with aggregate()

Is the median age of smokers higher than that of non-smokers?

```
aggregate(age ~ smoker, my_df, median)
```

```
##   smoker  age  
## 1     no 30.5  
## 2    yes 35.0
```

## Subgroup summaries with aggregate()

```
aggregate(  
  my_df["age"],  
  list(smoker = my_df$smoker),  
  median)
```

```
##   smoker  age  
## 1     no 30.5  
## 2    yes 35.0
```

## Subgroup summaries with aggregate()

Compare the age of the younger person with children with the age of the younger person without children:

```
aggregate(age ~ child, my_df, min)
```

```
##   child age  
## 1    no  21  
## 2   yes  33
```



## Subgroup summaries with aggregate()

```
aggregate(  
  my_df["age"],  
  list(child = my_df$child),  
  min)
```

```
##   child age  
## 1    no  21  
## 2   yes  33
```

## Subgroup summaries with aggregate()

What is the age of the older smoker?

```
subset(  
  aggregate(age ~ smoker, my_df, max),  
  smoker == "yes",  
  select = "age"  
)
```

```
##    age
```

```
## 2   59
```

## Subgroup summaries with aggregate()

```
subset(  
  aggregate(  
    my_df["age"],  
    list(smoker = my_df$smoker),  
    max),  
  smoker == "yes",  
  select = "age")
```

```
##    age
```

```
## 2   59
```

## Subgroup summaries with aggregate()

We can divide our subgroups further into more subgroups:

```
aggregate(income ~ smoker + child, my_df, mean)
```

##	smoker	child	income
## 1	no	no	1.300000
## 2	yes	no	1.800000
## 3	no	yes	1.400000
## 4	yes	yes	1.866667

## Subgroup summaries with aggregate()

```
aggregate(  
  my_df["income"],  
  list(smoker = my_df$smoker,  
        child = my_df$child),  
  mean)
```

```
##   smoker child   income  
## 1     no    no 1.300000  
## 2    yes    no 1.800000  
## 3     no   yes 1.400000  
## 4    yes   yes 1.866667
```

## Subgroup summaries with aggregate()

On average, do parents who smoke earn more than parents who don't smoke?

```
subset(  
  aggregate(income ~ smoker + child, my_df, mean),  
  child == "yes",  
  select = c(smoker, income)  
)
```

```
##   smoker   income  
## 3      no 1.400000  
## 4     yes 1.866667
```

## Subgroup summaries with aggregate()

```
subset(  
  aggregate(  
    my_df["income"],  
    list(smoker = my_df$smoker,  
         child = my_df$child),  
    mean),  
  child == "yes",  
  select = c(smoker, income)  
)
```

```
##   smoker   income  
## 3      no 1.400000  
## 4     yes 1.866667
```

## Subgroup summaries with aggregate()

Is the median age of parents who smoke higher than that of parents who don't smoke?

```
subset(  
  aggregate(age ~ smoker + child, my_df, median),  
  child == "yes",  
  select = c(smoker, age)  
)
```

```
##   smoker age  
## 3      no  45  
## 4     yes  36
```



## Subgroup summaries with aggregate()

```
subset(  
  aggregate(  
    my_df["age"],  
    list(smoker = my_df$smoker,  
         child = my_df$child),  
    median),  
  child == "yes",  
  select = c(smoker, age)  
)
```

```
##   smoker age  
## 3      no  45  
## 4     yes  36
```

## Subgroup summaries with `sqldf()`

On average, do people with children earn more than people without children?

```
sqldf(  
  "SELECT child, AVG(income) as income  
  FROM my_df  
  GROUP BY child"  
)
```

```
##   child income  
## 1    no  1.425  
## 2   yes  1.750
```

## Subgroup summaries with `sqldf()`

On average, do people who smoke earn more than people who don't?

```
sqldf(  
  "SELECT smoker, AVG(income) as income  
  FROM my_df  
  GROUP BY smoker"  
)
```

##	smoker	income
## 1	no	1.325
## 2	yes	1.850

## Subgroup summaries with sqldf()

What is the lowest income for someone with children? And without?

```
sqldf(  
  "SELECT child, min(income) as income  
  FROM my_df  
  GROUP BY child"  
)
```

##	child	income
## 1	no	0.8
## 2	yes	1.4

## Subgroup summaries with sqldf()

Is the average age of people with children higher than that of people without children?

```
sqldf(  
  "SELECT child, AVG(age) as age  
  FROM my_df  
  GROUP BY child"  
)
```

```
##   child   age  
## 1    no 29.00  
## 2   yes 43.25
```

## Subgroup summaries with sqldf()

Compare the age of the younger person with children with the age of the younger person without children:

```
sqldf(  
  "SELECT child, min(age) as age  
  FROM my_df  
  GROUP BY child"  
)
```

```
##   child age  
## 1    no  21  
## 2   yes  33
```

## Subgroup summaries with sqldf()

What is the age of the older smoker?

```
sqldf(  
  "SELECT max(age) as age  
  FROM my_df  
  GROUP BY smoker  
  HAVING smoker = 'yes'  
  "  
)
```

```
##    age
```

```
## 1   59
```

## Subgroup summaries with aggregate()

We can divide our subgroups further into more subgroups:

```
sqldf(  
  "SELECT smoker, AVG(income) as income  
  FROM my_df  
  GROUP BY child, smoker  
  "  
)
```

##	smoker	income
## 1	no	1.300000
## 2	yes	1.800000
## 3	no	1.400000
## 4	yes	1.866667



## Subgroup summaries with sqldf()

On average, do parents who smoke earn more than parents who don't smoke?

```
sqldf(  
  "SELECT smoker, AVG(income) as income  
  FROM my_df  
  GROUP BY child, smoker  
  HAVING child = 'yes'  
  "  
)
```

##	smoker	income
## 1	no	1.400000
## 2	yes	1.866667