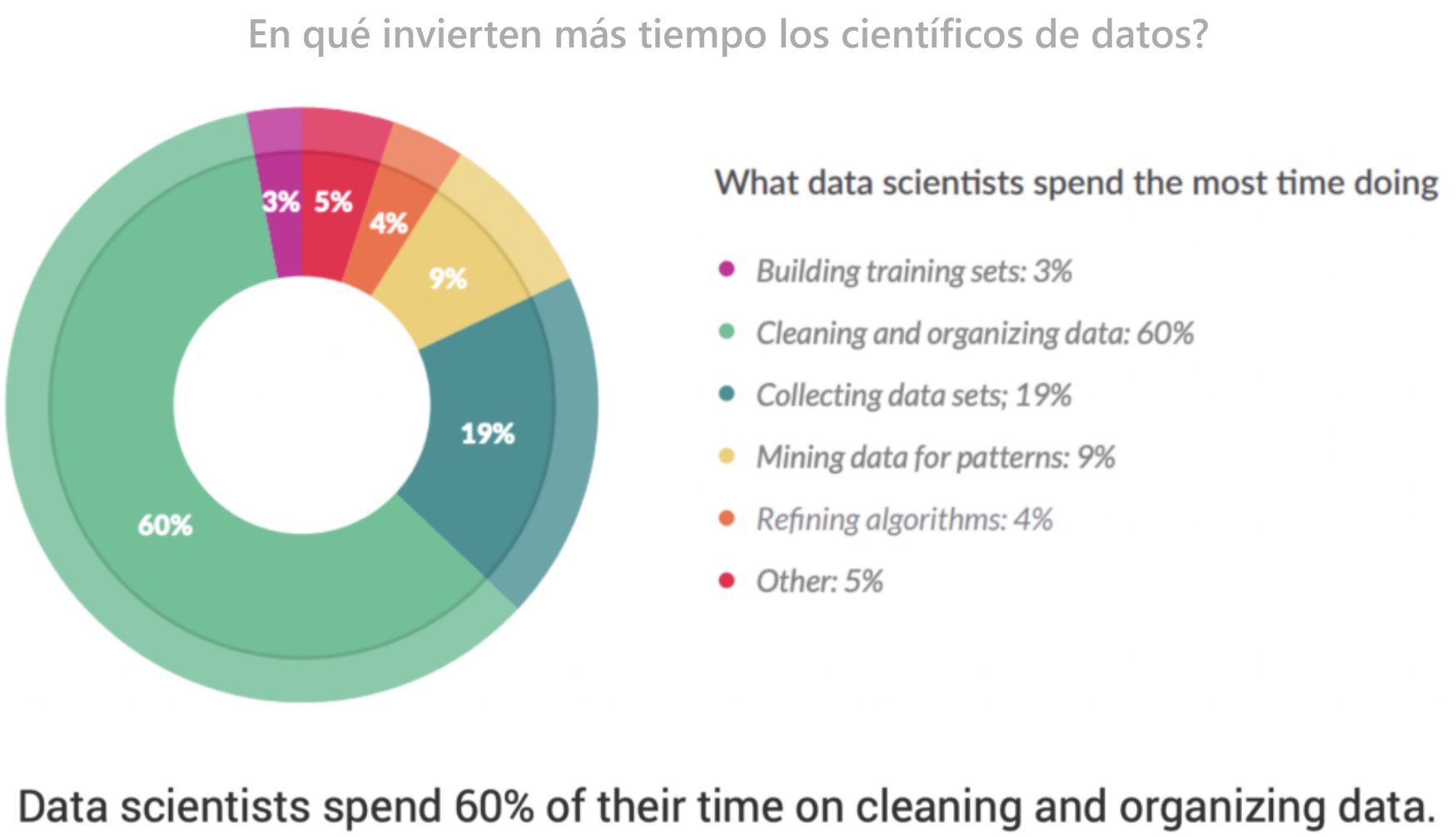
# Limpieza de datos con R


## Qué es limpieza de datos?

El DataCleaning o limpieza de datos, es **uno de los aspectos más importantes** en la ciencia de datos, a tal punto que muchos investigadores aseguran que en este procedimiento puede emplearse regularmente desde el $50\%$ hasta el $80\%$ del tiempo total de la investigación, siendo el tiempo restante invertido en los procesos de recolección, análisis y entrega de resultados.




Fuente: https://www.datavisor.com/blog/defeat-fraud-comprehensive-ai-powered-solution/

El objetivo principal de la limpieza de datos, es el **asegurar la calidad de la información** que se usará en los análisis, además de minimizar el riesgo en la toma de decisiones, en base a información perdida, poco precisa, duplicada, contradictoria, errónea o incompleta, ya qué ésta podría influir significativamente en los resultados estadísticos y conclusiones.

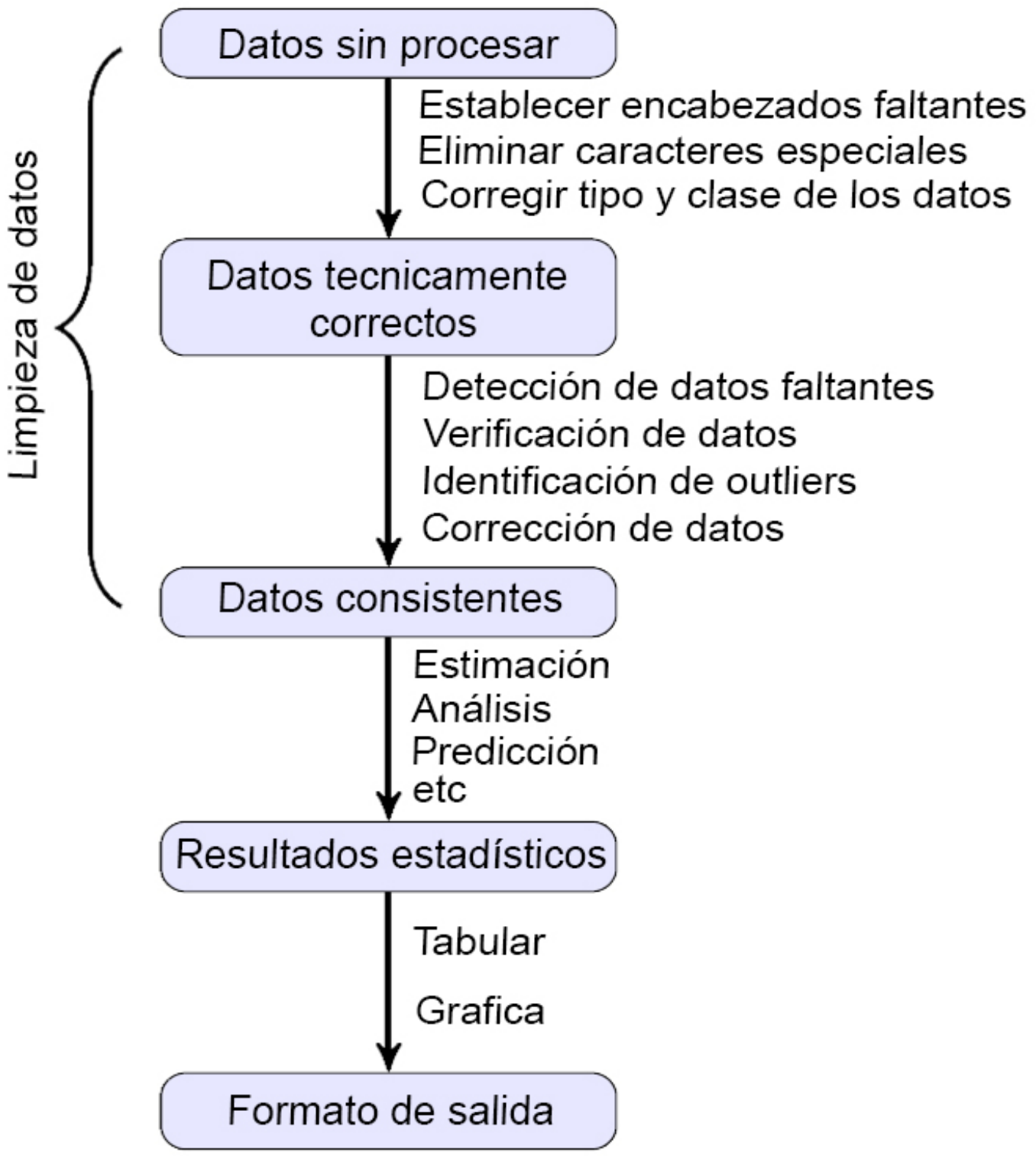
Los procedimientos de limpieza de la información, se realizarse **para mantener los datos, indicadores y reportes lo más precisos, consistentes y confiables de como sea posible**, ya que en muchas situaciones, es frecuente encontrar que en el proceso de recolección y almacenamiento de la información pueden surgir

diversos problemas, que terminan generando datos erróneos, ya sea por la falta de capacitación, falta de honestidad o errores involuntarios de los responsables de levantar y registrar la información, entre otros problemas.

Es por ello, que realizar un buen pre-procesamiento de la información permite **tomar mejores decisiones** y asegurar que los resultados obtenidos se asemejen a la realidad del objeto de estudio.

De Jonge & Van Der Loo (2013, p. 7) presentan un esquema que resume los **cinco pasos** que deben seguirse para realizar un **análisis estadístico adecuado** de la información, y señala en cuales de éstos es donde realiza el proceso de limpieza de datos.

Análisis estadístico en 5 pasos De Jonge & Van Der Loo (2013, p. 7)




## Lista de librerías

Con el fin de realizar el proceso de la limpieza de datos en `R`, se van a emplear los siguientes paquetes

```
library(readxl)  # Permite leer archivos en formato xlsx
library(janitor)  # Librería especializada en la limpieza de datos
library(dplyr)  # Librería especializada para la manipulación de datos
library(magrittr)  # Librería que probee una serie de operadores 'pipe'
library(editrules)  # Permite crear codiciones para detectar inconsistencias.
library(forcats)  # Librería para especializada en variables tipo factor.
library(Hmisc)  # Posee funciones que permite realizar imputaciones.
library(VIM)  # Posee funciones que permite realizar imputaciones.
```

## Operador pipe de continuidad %>%

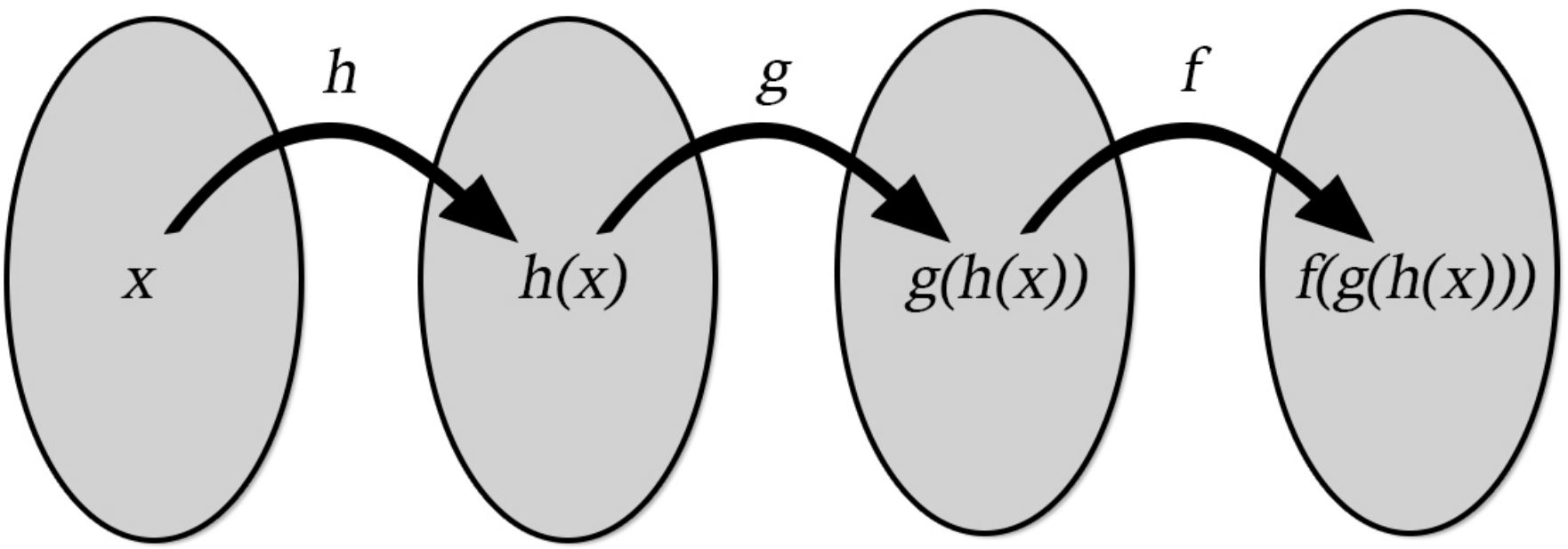
Este operador es una función de adición, el cual va tomando los elementos de izquierda a derecha y los va ejecutando en orden. Suponga a un conjunto de datos $x$, una función $f()$, una función $g()$ y una función $h()$.

Suponga además, que se desea realizar la siguiente operación

$$f(g(h(x)))$$

entonces, es posible realizar el siguiente procedimiento, en donde se presenta su equivalencia de mediante el operador .

Ilustración Operador pipe de continuidad




$$f(g(h(x))) \quad = \quad x \ \%>\% \ h() \ \%>\% \ g() \ \%>\% \ f()$$

## Ejemplo en R

Suponga que $x$ es un vector compuesto por $c(0.3, 3, 0.9)$, sea una función $f() = exp()$, una función $g() = abs()$ y una función $h() = log()$. Entonces si se desea calcular la función $f(g(h(x)))$ se tendrá que

```
### Definimos a una variable X
x <- c(0.3, 3, 0.9)

### Se realiza el cálculo de forma convencional
exp(abs(log(x)))
```

```
[1] 3.333333 3.000000 1.111111
```

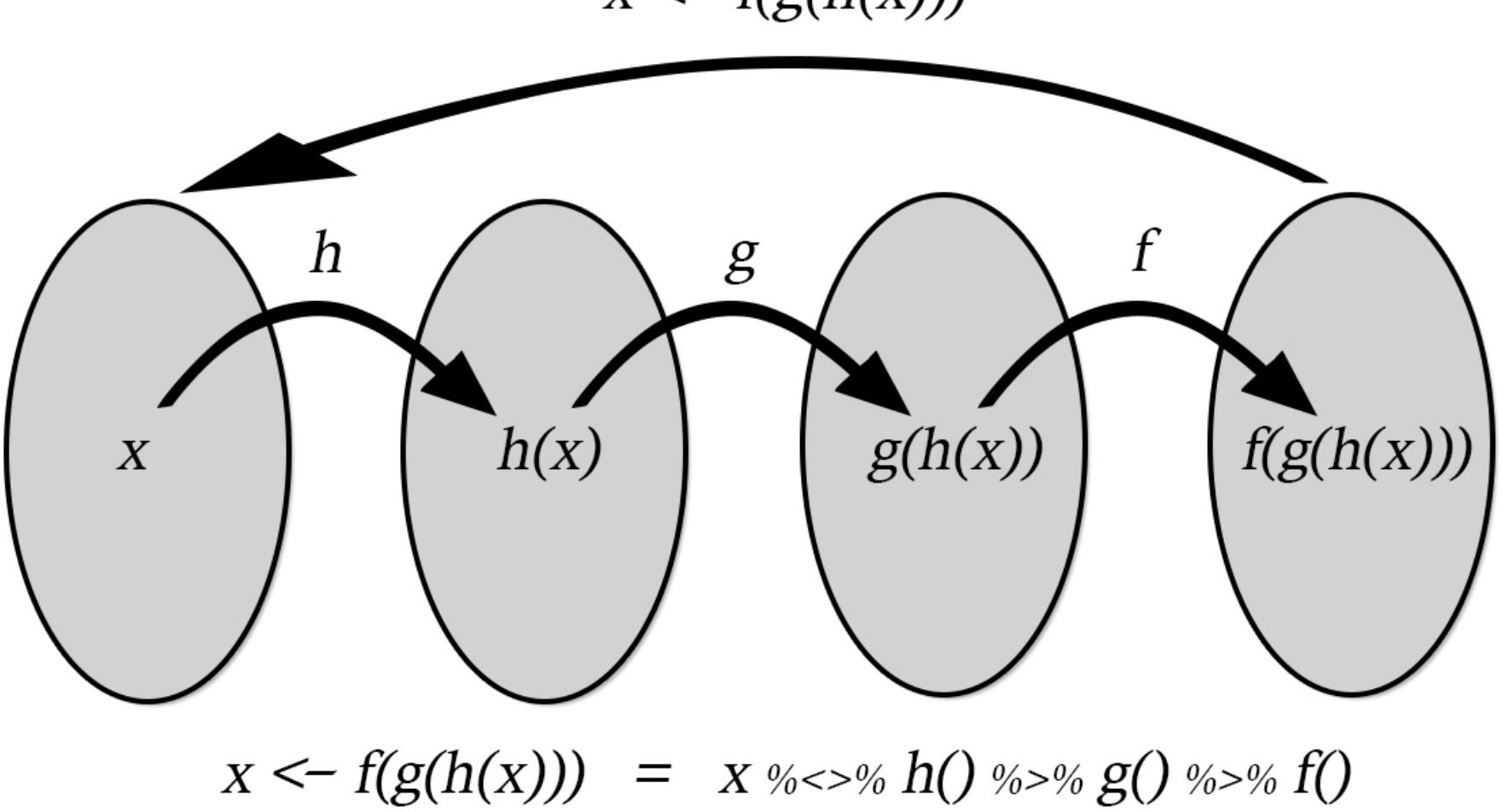
```
### Se realiza el cálculo mediante el empleo de pipe
x %>% log() %>% abs() %>% exp()
```

```
[1] 3.333333 3.000000 1.111111
```

## Operador pipe de asignación compuesta %<>%

Este operador funciona de forma similar al pipe de continuidad, pero con la diferencia, de que luego de realizar las operaciones éste reemplaza la variable $x$ original, por la secuencia de funciones que se le aplican a la misma variable $x$.

Ilustración operador pipe de asignación compuesta




## Ejemplo en R

Suponga al igual que en el ejemplo anterior, que $x$ es un vector compuesto por $c(0.3, 3, 0.9)$, función $f() = exp()$, una función $g() = abs()$ y una función $h() = log()$. Entonces si se desea calcular la función $f(g(h(x)))$ y almacenarla en $x$.

```
### Se presenta la variable X sin modificar
x
```

```
[1] 0.3 3.0 0.9
```

```
### Se realiza y almacena el cálculo mediante el empleo de pipe
x %<>% log() %>% abs() %>% exp()

### Se presenta el variable X modificada
x
```

```
[1] 3.333333 3.000000 1.111111
```

## Datos sin procesar a Datos técnicamente correctos

Los archivos de datos sin procesar, son generalmente bases de datos que pueden **carecer de encabezados**, contener codificación de **caracteres especiales** en los nombres, tener **errores en la clasificación** del tipos de datos, poseer **ordenes incorrecto** en dentro de sus categorías, entre otros.

Estos problemas puede generar inconvenientes al momento de realizar la lectura de los datos, manipulación de la información, o de realizarse análisis con estos datos, podrían arrojar resultados que no son consistentes con la realidad. Por ello se hace necesario usar alguna herramienta que permita **corregir dichos problemas**.

Con el fin de presenta los métodos para solucionar dichos problemas, suponga la siguiente base de datos, conformada por 10 columnas y 11 filas.

| Fecha | $ID | # telefono | [Estrato] | MUNICIPIO | | Deuda-vivienda? | Costo_vivienda? | % pagado | muni |
|---|---|---|---|---|---|---|---|---|---|
| 15/02/2013 | 1035869 | 3124751231 | 2 | La Estrella | NA | No | NaN | NaN | La Estrella |
| 01/05/2013 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 7 Mar 14 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 | La Estrella |
| encuestado el 12 01 13 | 3935563 | 7005931 | 4 | Hola! :D | 630700 | NA | -62708000 | 0.18 | Hola! :D |
| 01 05 2013 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 15 02 2013 | 2222985 | 8660538 | 10 | Sabaneta | Na | Si | 3.838e+09 | NA | Sabaneta |
| 23/09/2014 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 | Caldas |
| 2/06/2014 | 1022146 | 3979642 | 2 | Caldas | 2204800 | No responde | NA | 0.05 | Caldas |
| encuestado el 15/12/15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | | Itagüi |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 6 Feb 13 | 32321157 | 39522101 | 3 | La Estrella | | Sí | 169451000 | 0.22 | La Estrella |

Las variables anteriores son:

- **`Fecha:`** Hace referencia a la fecha en la cual se recolectó la información.
- **`ID:`** Hace referencia a un número de identificación de la persona.
- **`# de telefono:`** Hace referencia al número de la persona.
- **`Estrato:`** Hace referencia al estrato socioeconómico de la vivienda.
- **`MUNICIPIO:`** Hace referencia al municipio del Valle de Aburrá en donde se encuentra radicada la persona.
- **`:`** Hace referencia a los ingresos por concepto laboral.
- **`Deuda-vivienda?:`** Hace referencia a las personas que poseen o no deudas de vivienda.
- **`Costo vivienda?:`** Hace referencia al costo total de la vivienda.
- **`% pagado:`** Hace referencia al porcentaje del costo total pagado de la vivienda.
- **`muni:`** Variable duplicada, hace referencia al municipio en donde se encuentra radicada la persona.

Los datos anteriores, pueden ser descargados desde el siguiente Link.

### Lectura de datos y pre-visualización de encabezados

Para realizar el proceso de tratamiento de encabezados, `R` **posee diferentes librerías y funciones que permiten corregir estos errores**, entre ellas, la librería `dplyr` la cual provee una serie de funciones para la manipulación de bases de datos, la librería `lubridate` para tratar fechas, la librería `janitor` la cual posee una serie de funciones que sirven para el proceso de limpieza de datos, entre otras.

Para iniciar su empleo, procedemos a cargar la base de datos

### Cargar base de datos en línea

En este caso habrán dos formas de cargar la base de datos en R, la primera es importando los datos de internet en un archivo temporal y luego cargando la base de datos desde el archivo temporal.

```
temp = tempfile(fileext = ".xlsx")  # Crea archivo temporal
URL <-
"https://github.com/jiperezga/jiperezga.github.io/raw/master/Dataset/EjemploLimpieza.xls
  # URL base de datos
download.file(URL, destfile = temp, mode = "wb")  # Descarga archivo en el archivo
temporal creado
datos <- read_xlsx(temp)  # Carga base de datos desde archivo temporal
```

Una forma alternativa, será descargar la base de datos en una carpeta del pc, y posteriormente realizando la carga de la base de datos, buscando el fichero en donde se encuentra guardada.

```
datos <- read_xlsx(file.choose())  # Carga base de datos buscando fichero local
```

Una vez realizada la carga de la base de datos, echamos un vistazo al nombre de los encabezados de cada columna. Para ello empleamos la función `names()`.

```
datos %>% names()  # Muestra nombre de la base de datos
```

```
 [1] "Fecha"           "$ID"             "# telefono"
 [4] "[Estrato]"       "MUNICIPIO"       "...6"
 [7] "Deuda-vivienda?" "Costo_vivienda?" "% pagado"
[10] "muni"
```

En la salida anterior, se observa que dentro de los nombres de la base de datos tenemos, encabezados faltantes, los cuales se identifican por tener tres puntos, seguidos del número de la columna (Por ejemplo …6). También tenemos caracteres especiales tales como $, #, [, ], %, ?, uso de espacios, guiones “-”, y abuso de letras en mayúsculas.

### Establecer encabezados faltantes

Para corregir los problemas encontrados en los encabezados, iniciamos con el **establecer o renombrar** aquellas variables que poseen nombres faltantes, y para ello empleamos la función `rename` de la librería `dplyr`.

```
datos %<>% rename(ingresos = "...6")  # renombra columna en la base de datos
datos %>% names()
```

```
 [1] "Fecha"           "$ID"             "# telefono"
 [4] "[Estrato]"       "MUNICIPIO"       "ingresos"
 [7] "Deuda-vivienda?" "Costo_vivienda?" "% pagado"
[10] "muni"
```

### Eliminar caracteres especiales

Para **corregir aquellos caracteres especiales, espacios, guiones y letras mayúsculas**, se emplea la función `clean_names()` de la librería `janitor`, la cual permite:

- Analizar las mayúsculas y minúsculas y separadores a un formato consistente.
- Maneja caracteres y espacios especiales.
- Agrega números a nombres duplicados.
- Convierte “%” en “percent” y “#” en “number” para conservar el significado.

```
datos %<>% clean_names()  # limpia los nombres nombre de la base de datos
datos %>% names()
```

```
 [1] "fecha"           "id"              "number_telefono"
 [4] "estrato"         "municipio"       "ingresos"
 [7] "deuda_vivienda"  "costo_vivienda"  "percent_pagado"
[10] "muni"
```

## Corregir tipo y clase de los datos

El tipo y clase de los datos juega un papel importante en el tratamiento de datos, debido a que **su adecuada especificación será la responsable de que el análisis aplicado a la variable sea el adecuado**.

La razón de ésto se debe a que podrían haber dentro de la base de datos, **variables numéricas tratadas como factores, factores tratadas como numéricas, variables ordinales tratadas como si fueran nominales, etc**. Lo cual haría que pueda aplicarse análisis a las variables de forma indebida, por ejemplo, realizar un análisis numérico a una variable que realmente es tipo factor, como es al número de teléfono.

Para observa las clases que poseen los datos, podemos emplear la función `str()`, o la función `glimpse()` de la librería `dplyr`.

```
datos %>% glimpse()  # muestra las clases que poseen los datos
```

```
Observations: 11
Variables: 10
$ fecha           <chr> "15/02/2013", "01/05/2013", "7 Mar 14", "encue...
$ id              <chr> "1035869", "1035857", "1007306", "3935563", "1...
$ number_telefono <chr> "3124751231", "1192334", "3434589", "7005931",...
$ estrato         <chr> "2", "3", "3", "4", "3", "10", "2", "2", "1", ...
$ municipio       <chr> "La Estrella", "Bello", "La Estrella", "Hola! ...
$ ingresos        <chr> "NA", "3114800", "3083500", "630700", "3114800...
$ deuda_vivienda  <chr> "No", "Sí", "Sí", "NA", "Sí", "Si", "Sí", "No ...
$ costo_vivienda  <chr> "NaN", "728753400", "405147000", "-62708000", ...
$ percent_pagado  <chr> "NaN", "0.28999999999999998", "0.31", "0.18", ...
$ muni            <chr> "La Estrella", "Bello", "La Estrella", "Hola! ...
```

De la salida anterior, vemos que solo las variables `municipio`, `deuda_de_vivienda` y `muni` poseen una categoría adecuada, aunque esta variable también podría ser recodificada a tipo factor.

La variable `fecha` aparece como tipo carácter (character) cuando debería ser tipo fecha (date). Por su parte, las variables `id`, number_de_telefono</tt> y `estrato` es clasificada como tipo numérica (double) pero éstas representan una cualidad, y por tanto, dichas variables deberían ser de tipo carácter (character).

Similarmente, las variables `ingresos`, `costo_vivienda` y `percent_pagado` aparecen en la salida como de tipo carácter (character) cuando realmente éstas variables son de tipo numérica (double). Lo anterior es causado debido a que entre la información contenida dentro de las variables numéricas, se encuentran valores tales como `NA` y `NaN` los cuales son tomados como valores especiales por el R y otros valores tales como `No responde` y `Na`.

**Para convertir una variable de un clase a otra**, existen una serie de funciones básicas en R que permiten hacer dicho procedimiento, estas funciones son

- **as.numeric():** Convierte una variable a tipo numérico (double).
- **as.logical():** Convierte una variable a tipo lógico.
- **as.integer():** Convierte una variable a tipo entero.
- **as.factor():** Convierte una variable a tipo factor.
- **as.character():** Convierte una variable a tipo carácter (character).
- **as.ordered():** Convierte una variable a tipo factor asumiendo un orden o jerarquía entre los niveles.

Cada una de estas funciones toma un objeto R e intenta convertirlo a la clase especificada detrás del “as.”, en donde, aquellos valores que no se pueden convertir al tipo especificado, R los convertirá por defecto a un NA.

Para corregir la clase de las variables, usaremos la función `mutate()` de la librería `dplyr`, de la forma

```
### Transformar variables a tipo factor
datos %<>% mutate_at(vars(id, number_telefono, estrato, municipio, deuda_vivienda,
    muni), as.factor) %>% glimpse()
```

```
Observations: 11
Variables: 10
$ fecha           <chr> "15/02/2013", "01/05/2013", "7 Mar 14", "encue...
$ id              <fct> 1035869, 1035857, 1007306, 3935563, 1035857, 2...
$ number_telefono <fct> 3124751231, 1192334, 3434589, 7005931, 1192334...
$ estrato         <fct> 2, 3, 3, 4, 3, 10, 2, 2, 1, NA, 3
$ municipio       <fct> La Estrella, Bello, La Estrella, Hola! :D, Bel...
$ ingresos        <chr> "NA", "3114800", "3083500", "630700", "3114800...
$ deuda_vivienda  <fct> No, Sí, Sí, NA, Sí, Si, Sí, No responde, Sí, N...
$ costo_vivienda  <chr> "NaN", "728753400", "405147000", "-62708000", ...
$ percent_pagado  <chr> "NaN", "0.28999999999999998", "0.31", "0.18", ...
$ muni            <fct> La Estrella, Bello, La Estrella, Hola! :D, Bel...
```

```
### Transformar variables a tipo numérico (double)
datos %<>% mutate_at(vars(ingresos, costo_vivienda, percent_pagado), as.numeric) %>%
    glimpse()
```

```
Observations: 11
Variables: 10
$ fecha           <chr> "15/02/2013", "01/05/2013", "7 Mar 14", "encue...
$ id              <fct> 1035869, 1035857, 1007306, 3935563, 1035857, 2...
$ number_telefono <fct> 3124751231, 1192334, 3434589, 7005931, 1192334...
$ estrato         <fct> 2, 3, 3, 4, 3, 10, 2, 2, 1, NA, 3
$ municipio       <fct> La Estrella, Bello, La Estrella, Hola! :D, Bel...
$ ingresos        <dbl> NA, 3114800, 3083500, 630700, 3114800, NA, 335...
$ deuda_vivienda  <fct> No, Sí, Sí, NA, Sí, Si, Sí, No responde, Sí, N...
$ costo_vivienda  <dbl> NaN, 728753400, 405147000, -62708000, 72875340...
$ percent_pagado  <dbl> NaN, 0.29, 0.31, 0.18, 0.29, NA, 0.75, 0.05, N...
$ muni            <fct> La Estrella, Bello, La Estrella, Hola! :D, Bel...
```

Ahora, para **transformar la variable ‘fecha’ a tipo fecha (date)**, se pueden emplear las funciones `dmy()` (día-mes-año), `mdy()` (mes-día-año), `myd()` (mes-año-día), `ymd()` (año-mes-día), `ydm()` (año-día-mes) o `dym()` (día-año-mes) de la librería `lubridate`, dependiendo del orden de como se encuentre registrada la fecha.

La ventaja que tiene usar estas variable sobre la función base `as.Date` del R, pues dichas funciones transforma las fechas almacenadas en vectores numéricos y de caracteres en objetos tipo (date), aunque éstas contengan caracteres adicionales a las de fecha.

Dado que al observar las entradas que hay en la variable fecha son del tipo (día-mes-año), se emplea la función `dmy()` de la librería `lubridate`.

```
### Transformar variables a tipo fecha (date)
datos %<>% mutate_at(vars(fecha), dmy) %>% glimpse()
```

```
Observations: 11
Variables: 10
$ fecha           <date> 2013-02-15, 2013-05-01, 2014-03-07, 2013-01-1...
$ id              <fct> 1035869, 1035857, 1007306, 3935563, 1035857, 2...
$ number_telefono <fct> 3124751231, 1192334, 3434589, 7005931, 1192334...
$ estrato         <fct> 2, 3, 3, 4, 3, 10, 2, 2, 1, NA, 3
$ municipio       <fct> La Estrella, Bello, La Estrella, Hola! :D, Bel...
$ ingresos        <dbl> NA, 3114800, 3083500, 630700, 3114800, NA, 335...
$ deuda_vivienda  <fct> No, Sí, Sí, NA, Sí, Si, Sí, No responde, Sí, N...
$ costo_vivienda  <dbl> NaN, 728753400, 405147000, -62708000, 72875340...
$ percent_pagado  <dbl> NaN, 0.29, 0.31, 0.18, 0.29, NA, 0.75, 0.05, N...
$ muni            <fct> La Estrella, Bello, La Estrella, Hola! :D, Bel...
```

De la salida anterior, se aprecia que ya todas las variables se encuentran debidamente codificadas, logrando así, junto con los procedimientos anteriores **obtener un conjunto de datos es técnicamente correcto**. Dichos datos quedan estructurados de la forma

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado | muni |
|---|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | NA | No | NaN | NaN | La Estrella |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 | La Estrella |
| 2013-01-12 | 3935563 | 7005931 | 4 | Hola! :D | 630700 | NA | -62708000 | 0.18 | Hola! :D |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2013-02-15 | 2222985 | 8660538 | 10 | Sabaneta | NA | Si | 3.838e+09 | NA | Sabaneta |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 | Caldas |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | No responde | NA | 0.05 | Caldas |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | NA | Itagüi |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | NA | Sí | 169451000 | 0.22 | La Estrella |

## Datos técnicamente correctos a datos consistentes

Los datos consistentes **son la etapa en la que el conjunto de observaciones están listos** para la realización de inferencia estadística. Éstos datos son los que asumen como punto de partida en la mayoría de las teorías estadísticas, aunque **en la práctica no siempre se emplean éstos**, si no que usan datos sin procesar o técnicamente correctos.

Aunque es posible emplear datos sin procesar o técnicamente correcto para realizar inferencia estadística, el evadir el paso de llevar los datos a su forma consistente, puede tener una seria influencia en los resultados estadísticos, puesto que **la detección, verificación, corrección e imputación de datos, puede brindar información valiosa** en los análisis realizados o en la interpretación de los mismos.

**Detección de datos faltantes**

La detección de datos faltantes radica en localizar para cada variable, si se encuentran casillas vacías, valores especiales, tales como `Na, NaN` o `NULL`.

- **`NA` (Not Available):** Es un carácter especial para indicar valores perdidos. Éstos pueden ser detectados en `R` mediante la función `is.na()`.
- **`NaN` (Not a number):** Es un carácter especial para datos de clase numérica, para indicar un valor asociado a un cálculo cuyo resultado es desconocido, el cual seguramente no es un número. Este puede obtenerse mediante operaciones tales como $0/0$, $Inf/Inf$, $Inf - Inf$. Éstos pueden ser detectados en `R` mediante la función `is.nan()`, aunque también son detectados por la función `is.na()`.
- **`NULL`:** Es un carácter especial para indicar valores indefinidos o indicar la no existencia de valor dentro de la base de datos o de una entrada de la misma. Éstos pueden ser detectados en `R` mediante la función `is.null()`.

Para detectar aquellas filas que se encuentren faltantes dentro de una base de datos, podemos usar una combinación entre las funciones `apply()`, `which()`, tal como se hizo en la salida anterior, adicionando las funciones `is.na()` e `is.null()`, dentro de la función `which()`.

```
## Detectar NA, NaN y NULL dentro de la base de datos Se agrega 'NA' y 'Na' y
## 'NaN' debido a que hay entradas que poseen estas categorías como niveles
## del factor
faltantes <- apply(X = datos, MARGIN = 2, FUN = function(x) which(is.na(x) |
    is.null(x) | x == "Na" | x == "NA" | x == "NaN"))  # se pueden agregar más
caracteres
faltantes
```

```
$fecha
[1] 10

$id
[1] 10

$number_telefono
[1] 10

$estrato
[1] 10

$municipio
[1] 10

$ingresos
[1]  1  6 10 11

$deuda_vivienda
[1]  4 10

$costo_vivienda
[1]  1  8 10

$percent_pagado
[1]  1  6  9 10

$muni
[1] 10
```

De la salida anterior se observa que en todas las variables aparece con valores faltantes la fila número 10. Adicionalmente, se aprecian para las variables `municipio`, `ingresos`, `deuda_vivienda`, `costo_vivienda`, `percent_pagado` y `muni`, celdas similares son encontradas en el proceso de verificación de datos.

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado | muni |
|---|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | NA | No | NaN | NaN | La Estrella |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 | La Estrella |
| 2013-01-12 | 3935563 | 7005931 | 4 | Hola! :D | 630700 | NA | -62708000 | 0.18 | Hola! :D |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2013-02-15 | 2222985 | 8660538 | 10 | Sabaneta | NA | Si | 3.838e+09 | NA | Sabaneta |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 | Caldas |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | No responde | NA | 0.05 | Caldas |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | NA | Itagüi |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | NA | Sí | 169451000 | 0.22 | La Estrella |

## Verificación de datos

El objetivo de esta fase, consta en observar si hay algún tipo de **restricciones en los datos que está siendo violadas**, tales como inconsistencias en variables que está limitada a valores no negativos, variables que deben encontrarse en un rango determinado, variables de respuesta cerrada que poseen un registro por fuera del rango de respuestas, problemas debidos al acento (tildes, virgulillas, etc), entre otras.

Por tanto, se hace necesario **localizar por columna**, los valores que presenten inconsistencias, y que sepamos poseen restricciones en sus respuestas. En la base de datos que poseemos, tenemos las siguientes restricciones.

- `estrato:` Es una variable categórica, sus valores deben encontrarse entre 1 y 6.
- `municipio:` Es una variable categórica, sus valores están restringidos a los municipios del Valle de Aburrá.
- `ingreso:` Es una variable numérica, no puede contener valores negativos.
- `deuda_vivienda:` Es una variable categórica, sus respuestas deberían ser Sí o No.
- `costo_vivienda:` Es una variable numérica, no puede contener valores negativos.
- `percent_pagado:` Es una variable numérica, sus valores deben encontrarse entre 0 y 1.
- `muni:` Es una variable categórica, sus valores están restringidos a los municipios del Valle de Aburrá.

Para realizar tal proceso de verificación se debe realizar la localización de aquellos valores que presenten inconsistencias, en donde, pueden emplearse las funciones `editset()` y `violatedEdits()` de la librería `editrules`

```
## Creamos conjunto de condiciones que deben cumplirse condiciones para
## variables categóricas
CondC <- editset(c("estrato %in% 1:6", "municipio %in% c('Barbosa', 'Girardota',
'Copacabana', 'Bello', 'Medellín', 'Envigado', 'Itagüi', 'Sabaneta', 'La Estrella',
'Caldas')",
    "deuda_vivienda %in% c('Sí', 'No')", "muni %in% c('Barbosa', 'Girardota',
'Copacabana', 'Bello', 'Medellín', 'Envigado', 'Itagüi', 'Sabaneta', 'La Estrella',
'Caldas')"))
CondC
```

```
Data model:
dat1 : deuda_vivienda %in% c('No', 'Sí')
dat2 : estrato %in% c('1', '2', '3', '4', '5', '6')
dat3 : muni %in% c('Barbosa', 'Bello', 'Caldas', 'Copacabana', 'Envigado', 'Girardota',
'Itagüi', 'La Estrella', 'Medellín', 'Sabaneta')
dat4 : municipio %in% c('Barbosa', 'Bello', 'Caldas', 'Copacabana', 'Envigado',
'Girardota', 'Itagüi', 'La Estrella', 'Medellín', 'Sabaneta')

Edit set:
NULL :
```

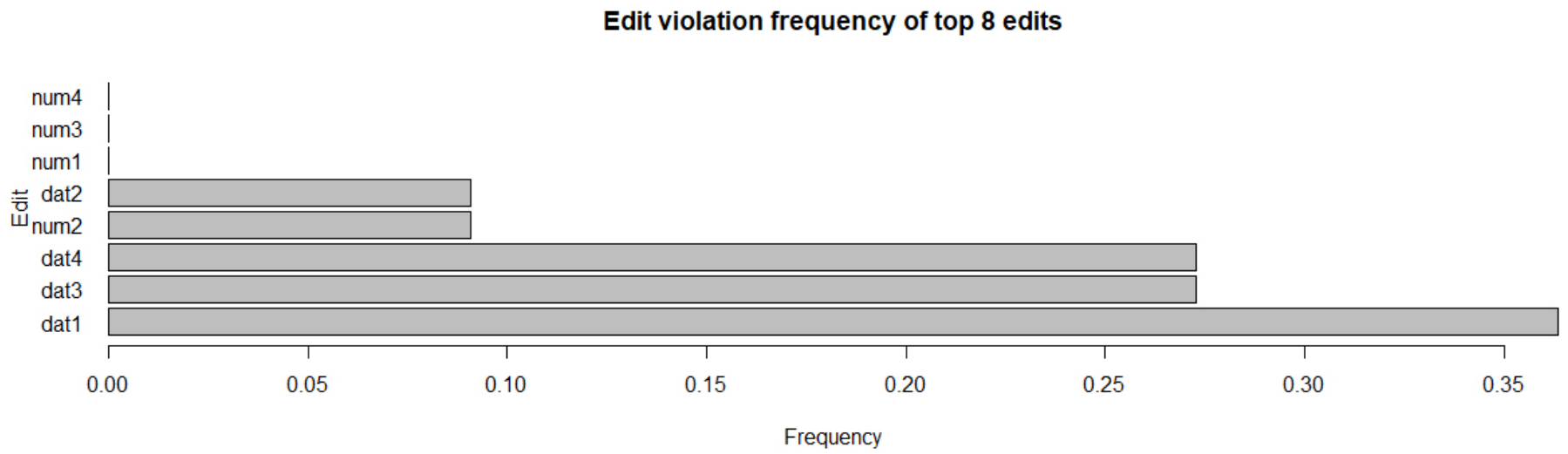
```
# Condiciones para variables númericas
CondN <- editset(c("ingresos > 0", "costo_vivienda > 0", "percent_pagado >= 0",
    "percent_pagado <= 1"))
CondN
```

```
Edit set:
num1 : 0 < ingresos
num2 : 0 < costo_vivienda
num3 : 0 <= percent_pagado
num4 : percent_pagado <= 1
```

```
# Presentamos donde ocurren dichas violaciones
Errores <- violatedEdits(c(CondC, CondN), datos)
Errores
```

```
      edit
record  num1  num2  num3  num4  dat1  dat2  dat3  dat4
    1     NA    NA    NA    NA FALSE FALSE FALSE FALSE
    2  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
    3  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
    4  FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE
    5  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
    6     NA FALSE    NA    NA  TRUE  TRUE FALSE FALSE
    7  FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
    8  FALSE    NA FALSE FALSE  TRUE FALSE FALSE FALSE
    9  FALSE FALSE    NA    NA FALSE FALSE FALSE FALSE
    10    NA    NA    NA    NA  TRUE  TRUE  TRUE  TRUE
    11    NA FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
# Presentamos de forma gráfica donde se cometen errores
plot(Errores)
```





En la imagen anterior se presentan dos gráficos. El gráfico superior muestra **cuales son las condiciones que más se están violando**, en donde se aprecia que la condición `dat1`, es la que tiene una mayor porcentaje de errores, con un poco más del $35\%$ de las violaciones de la base de datos. Cabe anotar que en el caso de variables categóricas, **los `NA` son contados como una violación**. En el gráfico inferior muestra **cuantas filas contienen `NA`, `NaN` o violaciones a las condiciones establecidas**, y señala cuantas filas no poseen ningún tipo de violación.

Para localizar la fila en donde se realizan violaciones en cada variable, podemos emplear la función `localizeErrors()` de la librería `editrules` seguida de una combinación entre las funciones `apply()` y `which()`, de la forma

```
# Localiza y muestra por variable las violaciones
loc <- localizeErrors(c(CondC, CondN), datos)$adapt
apply(X = loc, MARGIN = 2, FUN = function(x) which(x == TRUE))
```

```
$fecha
named integer(0)

$id
named integer(0)

$number_telefono
named integer(0)

$estrato
 6 10
 6 10

$municipio
 4 10
 4 10

$ingresos
 1  6 10 11
 1  6 10 11

$deuda_vivienda
 4  6  8 10
 4  6  8 10

$costo_vivienda
 1  4  8 10
 1  4  8 10

$percent_pagado
 1  6  9 10
 1  6  9 10

$muni
 4 10
 4 10
```

De la salida anterior, observamos que hay violaciones en la entrada $5$ y $10$ de la variable `estrato`, en las entradas $4, 9$ y $10$ de la variable `municipio`, etc, etc, etc. Es de anotar que todos los valores faltantes también son detectados por el proceso de verificación de datos.

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado | muni |
|---|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | NA | No | NaN | NaN | La Estrella |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 | La Estrella |
| 2013-01-12 | 3935563 | 7005931 | 4 | Hola! :D | 630700 | NA | -62708000 | 0.18 | Hola! :D |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2013-02-15 | 2222985 | 8660538 | 10 | Sabaneta | NA | Si | 3.838e+09 | NA | Sabaneta |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 | Caldas |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | No responde | NA | 0.05 | Caldas |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | NA | Itagüi |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | NA | Sí | 169451000 | 0.22 | La Estrella |

## Identificación de outlier

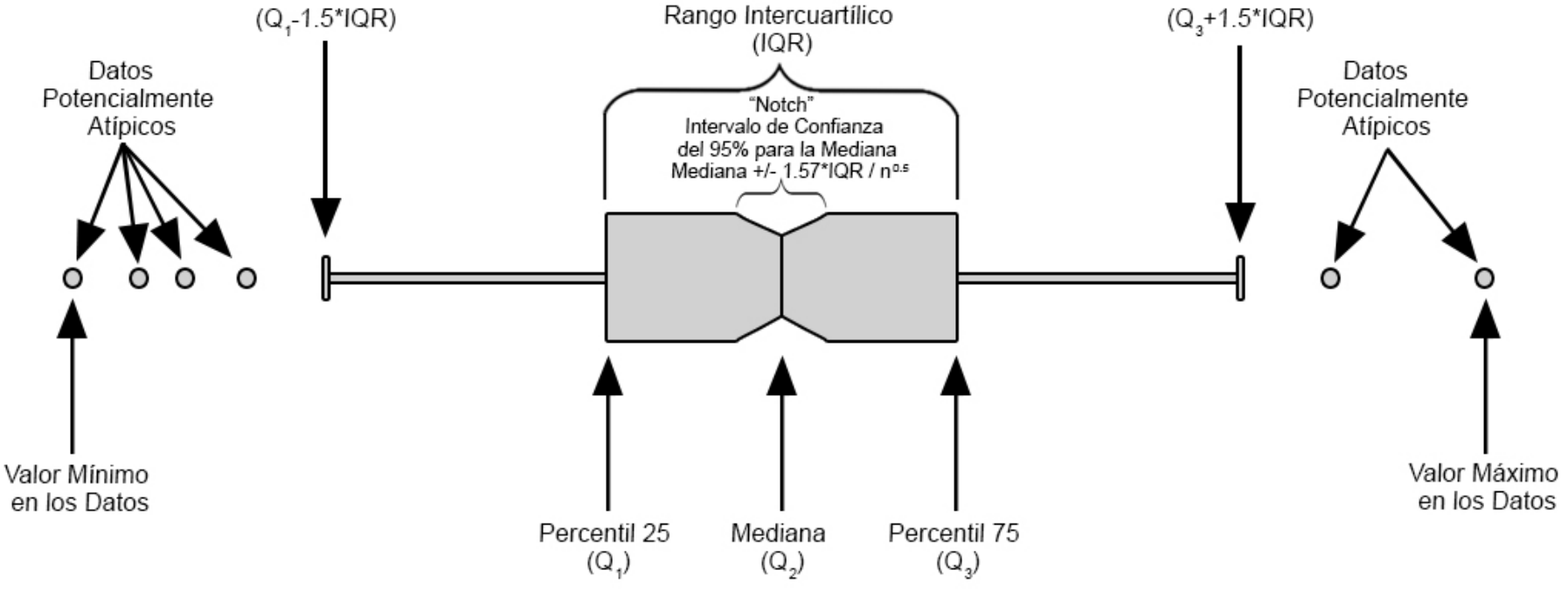
Para definir un outlier o dato atípico existe una gran cantidad de definiciones, pero en general puede entenderse como observaciones que pueden parecer anormales o extremas en un conjunto de datos, los cuales **pueden afectar significativamente la inferencia estadística** realizada.

Similarmente, existen diferentes metodologías para determinar datos atípicos, en donde, **la mayoría dependen de los estimadores insesgados asociados a la distribución de probabilidades** que presenta mejor ajuste al conjunto de observaciones.

En el caso en el cual no se conoce las distribuciones de probabilidad, pero se observa que **los datos son más o menos unimodales y distribuidos simétricamente**, puede emplearse el método de caja y bigotes de Tukey, McGill, & Larsen (1978) para detección de datos atípicos.

Éste método emplea estadísticos de orden no paramétricos, para calcular un gráfico de caja y bigotes, con el fin de visualizar los datos atípicos que se encuentren por encima y por debajo de $1.5$ veces el rango intercuartílico.

Gráfico de caja y bigotes




En `R` dicho gráfico puede realizarse mediante la función `boxplot()`, mientras que, los valores atípicos o la fila asociada dichos valores atípicos pueden ser calculados con la siguiente función.

### Creación de función Outliers

```
# Función para extraer valores atípicos de variables numéricas
Outliers <- function(data, row = TRUE) {
    if (!require(magrittr))
        install.packages("magrittr")
    if (!require(dplyr))
        install.packages("dplyr")
    require(magrittr)
    require(dplyr)
    vars <- data %>% select_if(is.numeric) %>% names()
    if (row != TRUE & row != FALSE)
        stop("row argument must be equal to TRUE or FALSE")
    aux <- function(data, vars) {
        Outlier <- data %>% select(vars) %>% boxplot(plot = FALSE) %$% out
        if (row == TRUE) {
            Outlier <- suppressWarnings(which(eval(parse(text = paste("data$",
                vars))) == Outlier))
        }
        return(vars = Outlier)
    }
    sapply(vars, aux, data = data)
}
```

```
## Empleo de la función Outliers Muestra fila en la que se encuentra el
## Outlier
datos %>% Outliers(row = FALSE)
```

```
$ingresos
[1] 630700

$costo_vivienda
[1] 3838000000

$percent_pagado
[1] 0.75 0.05
```

de la salida anterior, se aprecia que la observación atípica de la variable `ingresos` corresponde a un salario de $630700$ pesos, mientras que, para la variable `costo_vivienda` corresponde a un costo de vivienda de $3838$ millones de pesos, y para la variable `percent_pagado` lo porcentajes de $0.05$ y $0.75$.

Es de anotar que de estos datos, solo el valor atípico de `costo_vivienda` es el único que parece tener un valor excesivo, pues se cree que se agregó un $0$ de más.

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado | muni |
|---|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | NA | No | NaN | NaN | La Estrella |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 | La Estrella |
| 2013-01-12 | 3935563 | 7005931 | 4 | Hola! :D | 630700 | NA | -62708000 | 0.18 | Hola! :D |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2013-02-15 | 2222985 | 8660538 | 10 | Sabaneta | NA | Si | 3.838e+09 | NA | Sabaneta |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 | Caldas |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | No responde | NA | 0.05 | Caldas |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | NA | Itagüi |
| NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | NA | Sí | 169451000 | 0.22 | La Estrella |

## Corrección de datos

La métodos de corrección tienen por objetivo tratar revisar aquellas observaciones en las que se encontraron inconsistencias, removiendo filas o columnas vacías, eliminando filas o columnas duplicadas, corrección deductiva e imputando datos.

## Remover filas o columnas vacías

Para asegurarnos que las celdas faltantes detectadas con “NA”, “Na”, “NaN”, “999”, etc, en la subsección Detección de datos faltantes, se transformen a `NA`, pueden emplearse las funciones `na_if()` o `convert_to_NA()` de las librerías `dplyr` y `janitor`, respectivamente.

`na_if()` **se emplea cuando no existan variables de tipo `date`** en la base de datos, mientras que, `convert_to_NA()` **se emplea cuando existan variables tipo `date`** en la base de datos.

```
# Reemplaza valores 'NA', 'NaN' y 'Na' dentro de variables tipo factor
datos %<>% convert_to_NA("NA") %>% convert_to_NA("NaN")  # Se pueden agregar más
caracteres
```

Como puede ocurrir que al eliminar los “NA” de las variables tipo `factor`, **puede quedar categorías sin uso** dentro de las variables factor, se procede a eliminar dichos niveles mediante la función `droplevels()` de la base de `R`.

```
# Elimina niveles sin uso dentro de variables tipo factor
datos %<>% droplevels()
```

Finalmente, para **eliminar aquellas filas o columnas poseen solo valores `NA`** dentro de una base de datos, es posible usar la función `remove_empty()` de la librería `janitor`, la cual verifica todas las filas y todas las columnas, para observar en cuales de ellas contienen solo valores `NA`.

```
# Elimina filas y columnas que poseen solo valores NA
datos %<>% remove_empty()
```

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado | muni |
|---|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | NA | No | NA | NA | La Estrella |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 | La Estrella |
| 2013-01-12 | 3935563 | 7005931 | 4 | Hola! :D | 630700 | NA | -62708000 | 0.18 | Hola! :D |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 | Bello |
| 2013-02-15 | 2222985 | 8660538 | 10 | Sabaneta | NA | Si | 3.838e+09 | NA | Sabaneta |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 | Caldas |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | No responde | NA | 0.05 | Caldas |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | NA | Itagüi |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | NA | Sí | 169451000 | 0.22 | La Estrella |

## Eliminar duplicados

Esta fase tiene como objetivo eliminar aquellas filas y columnas que poseen registros duplicados, es decir, filas o columnas que son exactamente iguales. **Para eliminar las filas duplicadas**, es posible emplear la función `distinct()` de la librería `dplyr`, mientras que, **para eliminar las columnas duplicadas**, es posible emplear la función `select_if()` de la librería `dplyr`, en combinación con las funciones `duplicated()` y `as.list()` de la librería base de `R`.

```
# Elimina filas que poseen registros duplicados
datos %<>% distinct()

# Elimina colúmnas que poseen registros duplicados
datos %<>% select_if(!duplicated(as.list(.)))
```

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado |
|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | NA | No | NA | NA |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 |
| 2013-01-12 | 3935563 | 7005931 | 4 | Hola! :D | 630700 | NA | -62708000 | 0.18 |
| 2013-02-15 | 2222985 | 8660538 | 10 | Sabaneta | NA | Si | 3.838e+09 | NA |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | No responde | NA | 0.05 |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | NA |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | NA | Sí | 169451000 | 0.22 |

## Corrección deductiva

Esta fase es la más complicada del todo del procedimiento, debido a que es necesario **revisar detalladamente** la información obtenida en las fases de Verificación de datos y Identificación de Outliers, ya que **es necesario revisar las casillas en las que se encontraron violaciones** y tratar de transformar dichos valores no válidos, basados en información de valores válidos.

Para realizar tal procedimiento, podemos usar las funciones `mutate` de la librería `dplyr`, en combinación con las funciones `if_else()` de la librería `dplyr`, `fct_recode()` de la librería `forcats` y `droplevels()` de la base de `R`, para establecer condiciones que permitan modificar las otras variables.

```
## Corrección deductiva Estrato 10 no existe, así que se reemplaza valor por
## un NA (se escribe NULL)
datos %<>% mutate(estrato = fct_recode(estrato, `NULL` = "10")) %>% droplevels()

# El municipio 'Hola! :D' no existe, así que se reemplaza dicho valor por NA
datos %<>% mutate(municipio = fct_recode(municipio, `NULL` = "Hola! :D")) %>%
    droplevels()

# Se recodifica el nivel Si sin tilde de la variable deuda_vivienda por el
# nivel Sí con tilde
datos %<>% mutate(deuda_vivienda = fct_recode(deuda_vivienda, Sí = "Si")) %>%
    droplevels()

# La variable costo_vivienda no puede ser negativa
datos %<>% mutate(costo_vivienda = if_else(costo_vivienda < 0, abs(costo_vivienda),
    costo_vivienda))

# Se cree que se agregó un cero extra en el costo de la vivienda que aparece
# por un valor de 3838 millones de pesos, así que se le elimina un cero.
datos %<>% mutate(costo_vivienda = if_else(costo_vivienda == "3838000000",
costo_vivienda/10,
    costo_vivienda))

# Puede asumirse, que deuda_vivienda es igual a 'Sí' cuando se reporta valor
# en costo_vivienda o percent_pagado.
datos %<>% mutate(deuda_vivienda = if_else(costo_vivienda > 0 | percent_pagado >
    0, "Sí", "No", "No"))
```

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado |
|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | NA | No | NA | NA |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 |
| 2013-01-12 | 3935563 | 7005931 | 4 | NA | 630700 | Sí | 62708000 | 0.18 |
| 2013-02-15 | 2222985 | 8660538 | NA | Sabaneta | NA | Sí | 383800000 | NA |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | Sí | NA | 0.05 |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | NA |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | NA | Sí | 169451000 | 0.22 |

## Imputación de datos

La imputación es un procedimiento que **consta en estimar o derivar valores para campos que poseen datos faltantes*, en donde, dichas acciones tendrán una fuerte influencia en los resultados de un análisis estadístico. Tales procedimientos poseen una gran cantidad de metodologías diferentes, las cuales se escapan de esta presentación, y por tanto nos centraremos en unos cuantos.

De Jonge & Van Der Loo (2013, p. 46) presentan una tabla en donde presenta las diferentes librerías de R, que ofrecen alguna función que permita la realización de imputaciones, enumera éstas con una serie de métodos de imputación basados en modelos populares, y señalan que en **De Waal, Pannekoek, & Scholtus (2011, cap. 7) se realiza una revisión general de varios métodos de imputación**.

### Librerías que permites métodos de imputación en R De Jonge & Van Der Loo (2013, p. 46)

| Package | Numeric | | | Hot deck | | | | Longitudinal | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | mean | ratio | reg. | rand | seq | pmm | kNN | int | lo/no | LS |
| Amelia | · | · | · | · | · | · | · | · | · | · |
| BaBoon | · | · | · | · | · | ✓ | · | · | · | · |
| cat | · | · | · | ✓ | · | · | · | · | · | · |
| deducorrect | · | · | · | · | · | · | · | · | · | · |
| e1071 | ✓ | · | · | · | · | · | · | · | · | · |
| ForImp | ✓ | · | · | · | · | · | ✓‡ | · | · | · |
| Hmisc | ✓ | · | · | ✓ | · | ✓ | · | · | · | · |
| imputation | ✓ | · | ✓† | · | · | · | ✓ | · | · | · |
| impute | · | · | · | · | · | · | ✓ | · | · | · |
| mi | · | · | ✓* | ✓ | · | ✓ | · | · | · | · |
| mice | ✓ | · | ✓* | ✓ | · | ✓ | · | · | · | · |
| mix | · | · | · | · | · | · | · | · | · | · |
| norm | · | · | · | · | · | · | · | · | · | · |
| robCompositions | · | · | ✓* | ✓ | · | · | ✓ | · | · | · |
| rrcovNA | · | · | · | · | · | · | · | · | · | · |
| StatMatch | · | · | · | ✓ | · | · | ✓ | · | · | · |
| VIM | · | · | ✓* | ✓ | · | · | ✓ | · | · | · |
| yaImpute | · | · | · | · | · | · | ✓ | · | · | · |
| zoo | ✓ | · | · | · | · | · | · | ✓ | ✓ | · |

*Methods are ultimately based on some form of regression, but are more involved than simple linear regression.

†Uses a non-informative auxiliary variable (row number).

‡Uses nearest neighbor as part of a more involved imputation scheme.

Para ilustrar lo métodos de imputación, se presentan el método de imputación numérica, método de imputación Hot deck, y método de imputación kNN.

## Imputación numérica

Como su nombre lo indica, este método de imputación se realiza para variables numéricas y **lo que haces es emplear medidas de tendencias central, tal como la media, mediana, media recortada o moda, para imputar los valores faltantes**. Dichas medidas de tendencia central, pueden consultarse en el siguiente link Medidas de tendencia central.

Para la realización de esta imputación pueden emplearse las funciones `mutate()`, `if_else()` de la librería `dplyr`, junto con la función, `is.na()`, de la librería base de `R`, y la función `impute()` de la librería `Hmisc`. También será necesario seleccionar la medida de tendencia central de interés, entre ellas se tiene, `mean()` para la media o media recortada, `median()` para la mediana, `Moda()` (ver función en Medidas de tendencia central.) para moda.

```
# Se crea nueva base de datos para mostrar diferentes métodos de imputación
NumImp <- datos

# Se imputan los ingresos con la mediana
NumImp %<>% mutate(ingresos = impute(ingresos, median))

# Se imputa costo_vivienda (restringido a deuda_vivienda) con la media
NumImp %<>% mutate(costo_vivienda = if_else(is.na(costo_vivienda) & deuda_vivienda ==
    "Sí", mean(costo_vivienda, na.rm = TRUE), costo_vivienda))

# Se imputa percent_pagado (restringido a deuda_vivienda) con la media
# recortada al 20%
NumImp %<>% mutate(percent_pagado = if_else(is.na(percent_pagado) & deuda_vivienda ==
    "Sí", mean(percent_pagado, na.rm = TRUE, trim = 0.2), percent_pagado))
```

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado |
|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | 3099150 | No | NA | NA |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 |
| 2013-01-12 | 3935563 | 7005931 | 4 | NA | 630700 | Sí | 62708000 | 0.18 |
| 2013-02-15 | 2222985 | 8660538 | NA | Sabaneta | 3099150 | Sí | 383800000 | 0.25 |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | Sí | 847222301 | 0.05 |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | 0.25 |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | 3099150 | Sí | 169451000 | 0.22 |

## Imputación Hot Deck

Este método de imputación se realiza para tanto para variables numéricas como categóricas, y **lo que haces es emplear registros que se encuentren en la base de datos, para imputar los valores faltantes**, pero se recomienda emplear cuando hay varios registros, debido a que si son pocos, es posible que no hayan registros lo suficientemente similares, para realizar la imputación de forma adecuada.

Entre los métodos de imputación Hot Deck, se tiene que el más simple es el **método aleatorio**, el cual consta de elegir un valor de forma uniforme y aleatoriamente de la misma base de datos y reemplaza con éste el valor faltante. Éste método puede ser implementado mediante la función `impute()` de la librería `Hmisc`. Dado que el método de imputación se realiza de forma aleatoria **se recomienda establecer una semilla `set.seed()` antes de aplicar el método**, para poder replicar los resultados.

Entonces, para la realización de esta imputación, pueden emplearse las funciones `mutate_at()`, `mutate()`, `if_else()` de la librería `dplyr`, junto con las funciones, `as.numeric()`, de la librería base de `R`, y la función `impute()` de la librería `Hmisc`.

```
# Se crea nueva base de datos para mostrar diferentes métodos de imputación
HotImp <- datos

# Se establece una semilla cualquiera
set.seed(1844)

# Se imputan todas las variables excepto, costo_vivienda y percent_pagado
# debido a que tienen una restricción
HotImp %<>% mutate_at(vars(-c(costo_vivienda, percent_pagado)), ~impute(., "random"))

# Se imputa costo_vivienda (restringido a deuda_vivienda) con la media
HotImp %<>% mutate(costo_vivienda = if_else(deuda_vivienda == "Sí",
as.numeric(impute(costo_vivienda,
    "random")), costo_vivienda))

# Se imputa percent_pagado (restringido a deuda_vivienda) con la media
# recortada al 20%
HotImp %<>% mutate(percent_pagado = if_else(deuda_vivienda == "Sí",
as.numeric(impute(percent_pagado,
    "random")), percent_pagado))
```

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado |
|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | 3356600 | No | NA | NA |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 |
| 2013-01-12 | 3935563 | 7005931 | 4 | Itagüi | 630700 | Sí | 62708000 | 0.18 |
| 2013-02-15 | 2222985 | 8660538 | 2 | Sabaneta | 3356600 | Sí | 383800000 | 0.31 |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | Sí | 3838000000 | 0.05 |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | 0.29 |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | 3083500 | Sí | 169451000 | 0.22 |

## Imputación kNN

El método de imputación del $k$-ésimo vecino más cercano ($k$ Nearest Neighbor), también puede ser aplicado a variables numéricas y variables factor, y consta de la aplicación de una función de distancia $d(i,j)$ que calcula una medida de disimilitud (poca semejanza) entre los registros, y **selecciona como el valor a reemplazar el dato faltante, a aquel valor que que tenga una distancia más pequeña**, es decir, a su vecino más cercano.

Para el cálculo del método de imputación kNN, es posible emplear la función kNN librería `VIM`, la cual emplea la función de distancia de Gower (1971, p. 859), para definir cuál es el vecino más cercano. Dado que dicha función no permite el uso de restricciones, debe realizarse en dos pasos, el primero paso es imputar toda la base de datos, y el segundo paso es recuperar desde la base original aquellas variables que poseen restricciones.

Para realizar el procedimiento de imputación de emplean la funciones `mutate()`, `if_else()` de la librería `dplyr` y la función kNN de la librería `VIM`.

```
# Se crea una nueva base de datos realizando la imputación total
knnImp <- kNN(datos, imp_var = FALSE)

## Se recuperan los valores dadas las restricciones establecidas Se recupera
## costo_vivienda
knnImp %<>% mutate(costo_vivienda = if_else(deuda_vivienda == "No",
datos$costo_vivienda,
    costo_vivienda))

# Se recupera percent_pagado
knnImp %<>% mutate(percent_pagado = if_else(deuda_vivienda == "No",
datos$percent_pagado,
    percent_pagado))
```

| fecha | id | number_telefono | estrato | municipio | ingresos | deuda_vivienda | costo_vivienda | percent_pagado |
|---|---|---|---|---|---|---|---|---|
| 2013-02-15 | 1035869 | 3124751231 | 2 | La Estrella | 3114800 | No | NA | NA |
| 2013-05-01 | 1035857 | 1192334 | 3 | Bello | 3114800 | Sí | 728753400 | 0.29 |
| 2014-03-07 | 1007306 | 3434589 | 3 | La Estrella | 3083500 | Sí | 405147000 | 0.31 |
| 2013-01-12 | 3935563 | 7005931 | 4 | La Estrella | 630700 | Sí | 62708000 | 0.18 |
| 2013-02-15 | 2222985 | 8660538 | 2 | Sabaneta | 3114800 | Sí | 383800000 | 0.29 |
| 2014-09-23 | 3488986 | 5625266 | 2 | Caldas | 3356600 | Sí | 186855000 | 0.75 |
| 2014-06-02 | 1022146 | 3979642 | 2 | Caldas | 2204800 | Sí | 186855000 | 0.05 |
| 2015-12-15 | 39359318 | 2304725 | 1 | Itagüi | 3428900 | Sí | 539641705 | 0.29 |
| 2013-02-06 | 32321157 | 39522101 | 3 | La Estrella | 3114800 | Sí | 169451000 | 0.22 |

# Muchas gracias! :D

## Ejercicio

Suponga la siguiente base de datos compuesta por 10 variables 30 entradas, que tiene por objetivo caracterizar a las personas que asisten a una congregación Cristiana. Dada la poca experiencia de los encuestadores, la base de datos posee una gran cantidad de errores los cuales, se espera en lo posible puedan ser corregidos.

La base de datos contiene las siguientes variables.

- **fecha- encuesta:** Fecha en la que se realizó la encuesta.
- **$ID:** Número de identificación con el cual aparece registrada la persona en la congregación.
- **tipo-documento:** Representa el tipo de documento que posee la persona. Los niveles son:
  - Cédula de ciudadanía
  - Tarjeta de identidad
  - Tarjeta de extranjería
  - Registro civil

- **# cédula:** Hace referencia el número de cédula de la persona encuestada.
- **“Estrato”:** Estrato socioeconómico de la persona encuestada.
- **COmúna:** Hace referencia a la comúna de la ciudad de Medellín donde vive la persona. Los niveles son:
  - Aranjuez
  - Belén
  - Buenos Aires
  - Castilla
  - Doce de octubre
  - Guayabal
  - La América
  - La Candelaria
  - Laureles
  - Manrique
  - Poblado
  - Popular
  - Robledo
  - San Javier
  - Santa Cruz
  - Villa Hermosa

- **:** Número de hijos que posee la persona encuestada.
- **Asist_Prom:** Hace referencia al número promedio de veces al mes que asiste la persona a la congregación.
- **‘Tiemp_Prom’:** Tiempo promedio que pasa la persona semanalmente en la congregación.
- **Prom*Caridad:** Hace referencia a la cantidad promedio en pesos que aporta la persona mensualmente a la congregación como caridad.

La base de datos puede encontrarse en el siguiente Link.

Con esta base realice el siguiente procedimiento.

1. Revise la base de datos de forma visual y trate de listar los posibles valores que ésta posea.
2. Limpie los encabezados y establezca aquellos encabezados faltantes.
3. Corregir el tipo y clase de las variables. Revise las variables numéricas y corrija en posible aquellos valores que no son numéricos antes transformar la variables.
4. Encuente los valores faltantes.
    - NA
    - NaN
    - NULL
    - Na
    - nA
    - 999
5. Verifique que los datos posean las categorías.
6. De las variables numéricas, identifique aquellos valores atípicos, y trate de establecer cuales de ellos pueden ser corregidos y bajo qué condiciones.
7. Realice las correcciones que considere necesarias.
8. Emplee el método kNN de imputación y presente la base de datos definitiva.

### Solución R

```
### Se cargan librerías
library(readxl)  # Permite leer archivos en formato xlsx
library(janitor)  # Librería especializada en la limpieza de datos
library(dplyr)  # Librería especializada para la manipulación de datos
library(magrittr)  # Librería que probee una serie de operadores 'pipe'
library(editrules)  # Permite crear codiciones para detectar inconsistencias.
library(forcats)  # Librería para especializada en variables tipo factor.
library(lubridate)  # Librería que permite analizar y manipular fechas.
library(Hmisc)  # Posee funciones que permite realizar imputaciones.
library(VIM)  # Posee funciones que permite realizar imputaciones.

### Se hace lectura de los datos
temp = tempfile(fileext = ".xlsx")  # Crea archivo temporal
URL <-
"https://github.com/jiperezga/jiperezga.github.io/raw/master/Dataset/EjercicioLimpieza.x
  # URL base de datos
download.file(URL, destfile = temp, mode = "wb")  # Descarga archivo en el archivo
temporal creado
datos <- read_xlsx(temp)  # Carga base de datos desde archivo temporal

datos %<>% rename(numero_hijos = "...7")  # renombra columna en la base de datos
datos %<>% clean_names()  # limpia los nombres nombre de la base de datos
datos %<>% rename(comuna = "c_omuna")  # renombra columna c_omuna en la base de datos

### Transformar variables a tipo factor
datos %<>% mutate_at(vars(id, tipo_documento, number_cedula, estrato, c_omuna),
    as.factor)

### Revisar variables numéricas para detectar valores especiales
datos %>% select(numero_hijos) %>% table()  # Se detectan valores 'dos y uno'
datos %>% select(asist_prom) %>% table()  # Se detectan valores 'Ninguna'
datos %>% select(tiemp_prom) %>% table()  # No se detecta ningun valor no numerico
datos %>% select(prom_caridad) %>% table()  # No se detecta ningun valor no numerico

### Se corrigen valores especiales en numero_hijos y asist_prom, antes de
### transformar
datos %<>% mutate_at(vars(numero_hijos), ~replace(., . == "dos", "2"))  # Se reemplaza
el 2 en variable numero_hijos
datos %<>% mutate_at(vars(numero_hijos), ~replace(., . == "uno", "1"))  # Se reemplaza
el 1 en variable numero_hijos
datos %<>% mutate_at(vars(asist_prom), ~replace(., . == "ninguna", "0"))  # Se
reemplaza el 1 en variable asist_prom

### Transformar variables a tipo numérico (double)
datos %<>% mutate_at(vars(numero_hijos, asist_prom, tiemp_prom, prom_caridad),
    as.numeric)

### Transformar variables a tipo fecha (date) (se encuentra mes-día-año)
datos %<>% mutate_at(vars(fecha_encuesta), mdy)


## Creamos conjunto de condiciones que deben cumplirse condiciones para
## variables categóricas
CondC <- editset(c("tipo_documento %in% c('Cédula de ciudadanía', 'Tarjeta de
identidad', 'Tarjeta de extranjería', 'Registro civil')",
    "estrato %in% 1:6", "comuna %in% c('Popular', 'Santa Cruz', 'Manrique',
'Aranjuez', 'Castilla', 'Doce de octubre', 'Robledo', 'Villa Hermosa', 'Buenos Aires',
'La Candelaria',
                  'Laureles', 'La América', 'San Javier', 'Poblado', 'Guayabal',
'Belén')"))
CondC
```

```r
# Condiciones para variables númericas
CondN <- editset(c("numero_hijos >= 0", "asist_prom >= 0", "tiemp_prom >= 0",
    "prom_caridad >= 0"))
CondN

# Presentamos donde ocurren dichas violaciones
Errores <- violatedEdits(c(CondC, CondN), datos)
Errores

# Localiza y muestra por variable las violaciones
loc <- localizeErrors(c(CondC, CondN), datos)$adapt
apply(X = loc, MARGIN = 2, FUN = function(x) which(x == TRUE))


## Detectar NA, NaN, NULL..... dentro de la base de datos estas categorías
## como niveles del factor
faltantes <- apply(X = datos, MARGIN = 2, FUN = function(x) which(is.na(x) |
    is.null(x) | x == "Na" | x == "NA" | x == "NaN" | x == "999" | x == "nA"))  # se
pueden agregar más caracteres
faltantes
```

```r
# Función para extraer valores atípicos de variables numéricas
Outliers <- function(data, row = TRUE) {
    if (!require(magrittr))
        install.packages("magrittr")
    if (!require(dplyr))
        install.packages("dplyr")
    require(magrittr)
    require(dplyr)
    vars <- data %>% select_if(is.numeric) %>% names()
    if (row != TRUE & row != FALSE)
        stop("row argument must be equal to TRUE or FALSE")
    aux <- function(data, vars) {
        Outlier <- data %>% select(vars) %>% boxplot(plot = FALSE) %$% out
        if (row == TRUE) {
            Outlier <- suppressWarnings(which(eval(parse(text = paste0("data$",
                vars))) == Outlier))
        }
        return(vars = Outlier)
    }
    sapply(vars, aux, data = data)
}
```

```r
## Empleo de la función Outliers Muestra fila en la que se encuentra el
## Outlier
datos %>% Outliers(row = FALSE)

# Reemplaza valores 'NA', 'NaN' y 'Na' dentro de variables tipo factor
datos %<>% convert_to_NA("NA") %>% convert_to_NA("NaN") %>% convert_to_NA("Na") %>%
    convert_to_NA("999") %>% convert_to_NA("NULL") %>% convert_to_NA("nA") %>%
    convert_to_NA("Na")  # Se pueden agregar más caracteres

# Elimina niveles sin uso dentro de variables tipo factor
datos %<>% droplevels()

# Elimina filas y columnas que poseen solo valores NA
datos %<>% remove_empty()

# Elimina filas que poseen registros duplicados
datos %<>% distinct()
datos
```

```
# Elimina colúmnas que poseen registros duplicados
datos %<>% select_if(!duplicated(as.list(.)))

## Corrección deductiva tipo_documento 'Cedula' se reemplaza por 'Cédula de
## ciudadanía'
datos %<>% mutate(tipo_documento = fct_recode(tipo_documento, `Cédula de ciudadanía` =
"Cedula")) %>%
    droplevels()

# estrato '0' no existe, así que se reemplaza por NA
datos %>% mutate(estrato = na_if(estrato, "0")) %>% droplevels()

# estrato 'uno' se reemplaza por 1 y 'dos se reemplaza por 2
datos %<>% mutate(estrato = fct_recode(estrato, `1` = "uno", `2` = "dos")) %>%
    droplevels()

# comuna 'No sabe' se reemplaza por 'NA' para imputarlo
datos %<>% mutate(comuna = fct_recode(comuna, `NULL` = "No sabe")) %>% droplevels()

# comuna 'San Pedro', 'Guatapé' y se reemplaza por 'Fuera de Medellín'
datos %<>% mutate(comuna = fct_recode(comuna, `Fuera de Medellín` = "San Pedro",
    `Fuera de Medellín` = "Guatapé")) %>% droplevels()

# tiemp_prom se cambian valores negativos a positivos
datos %<>% mutate(tiemp_prom = if_else(tiemp_prom < 0, abs(tiemp_prom), tiemp_prom))

# Se realiza imputación kNN de toda la base de datos, dado que no hay
# restricciones aparentes
datos %>% kNN(imp_var = FALSE)
```

## Referencias


De Jonge, E., & Van Der Loo, M. (2013). *An introduction to data cleaning with r*. Statistics Netherlands Heerlen.

De Waal, T., Pannekoek, J., & Scholtus, S. (2011). *Handbook of statistical data editing and imputation* (Vol. 563). John Wiley & Sons.

Gower, J. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, 857–871.

Tukey, J., McGill, R., & Larsen, W. (1978). Variations of box plots. *The American Statistician*, *32*(1), 12–16.