



Introducción al uso de "R" en emergencias de salud

Módulo 2:

Tratamiento de datos y características relacionadas con las personas

Clase 8: Gramática tidyverse para el análisis de datos:

Preparación de bases de datos para análisis de situación de salud (Parte 2)



```
modulo_2 <- ("Aula 8")
```

```
> Crear y transformar columnas: mutate()
```

```
> Separar y unir columnas: separate() y unite()
```

```
> (Re) categorizar variables: case_when()
```

```
> Filtrar líneas: filter()
```

FICHA_TECNICA

Introducción al uso de "R" en emergencias de salud

Módulo 2: Tratamiento de datos y características relacionadas con las personas

Clase 8: Gramática tidyverse para el análisis de datos:

Preparación de bases de datos para análisis de situación de salud (Parte 2)

ORGANIZACIÓN PANAMERICANA DE LA SALUD/ORGANIZACIÓN MUNDIAL DE LA SALUD (OPS/OMS)

Coordinación General

Dra. Socorro Gross Galiano – Representante de la OPS/OMS – Brasília/Brasil

Dra. Maria Almiron – Advisor, Detection, Verification and Risk Assessment/Health Emergency Information & Risk Assessment (HIM)/ PAHO Health Emergencies Department (PHE) – Washington/United States

Coordinación Ejecutiva

Unidad Técnica de Vigilancia, Preparación y Respuesta a Emergencias y Desastres/ OPS/OMS – Brasília/Brasil

Walter Massa Ramalho

Juan Cortez-Escalante

Laís de Almeida Relvas-Brandt

Mábia Milhomem Bastos

Amanda Coutinho de Souza

Health Emergency Information & Risk Assessment (HIM)/ PAHO Health Emergencies Department (PHE) – Washington/United States

Cristian Hertlein

Wildo Navegantes de Araújo

Equipo Técnico

Unidad Técnica de Vigilancia, Preparación y Respuesta a Emergencias y Desastres/ OPS/OMS

Contenido

Laís de Almeida Relvas-Brandt

Revisión

Flávia Reis de Andrade

Amanda Coutinho de Souza

Laís de Almeida Relvas-Brandt

Equipo Pedagógico

Mônica Diniz Durães – Colaboradora – Consultora Nacional de Capacidades Humanas para la Salud/ OPS/OMS

Creación Digital

Pedro Augusto Jorge de Queiroz

Flávia Reis de Andrade

> 1

> Para seguir la clase

Para seguir esta clase, abra el proyecto del curso haciendo doble clic en el archivo `r_opas.Rproj`, dentro de la carpeta `r_opas` que descargó con el material del curso. Cuando se inicie RStudio, abra el script `r_opas_mod2_clase8_tidyverse2.R`, ubicado en la carpeta `r_opas/scripts`. O cree un script en blanco para que pueda ingresar los códigos, como aprendió en la clase anterior.

Luego ejecute la función `pacman::p_load()` para instalar y/o cargar los paquetes que se utilizarán en la clase. Son ellos: paquete `pacman` para instalar y cargar todas las demás bibliotecas que se necesitarán en esta clase: `rio` (para la importación y exportación de datos), `tidyverse` (para el manejo y tratamiento de datos), `lubridate` (para el manejo de variables de tipo fecha) y `abjutils` (limpieza de texto). Los paquetes `lubridate` y `abjutils` se presentarán en esta clase, y los otros paquetes utilizados están incluidos en el `tidyverse`.

```
# ORGANIZACIÓN PANAMERICANA DE LA SALUD (OPS)
# UNIDAD TÉCNICA DE VIGILANCIA, PREPARACIÓN Y RESPUESTA A EMERGENCIAS Y DESASTRES
# CURSO PRINCIPIANTE DE R APLICADO A EMERGENCIAS DE SALUD
# MÓDULO 2 - TRATAMIENTO DE DATOS Y CARACTERÍSTICAS RELACIONADAS CON LAS PERSONAS
# CLASE 8 - PREPARACIÓN DE LAS BASES DE DATOS PARA ANÁLISIS DE SITUACIÓN DE SALUD 2

# 1. Para seguir la clase #####
# Instalar/Cargar paquetes
pacman::p_load(rio,          # Importación y exportación de datos
               tidyverse,    # Manejo y tratamiento de datos
               lubridate,    # Manejo de variables de tipo fecha
               abjutils)     # Limpieza de texto
```

Ahora que usted tiene los paquetes activados, importe los datos del curso con la función `rio::import()`. Vamos a trabajar con los archivos en `.rds` que se exportaron al final de la clase anterior (clase 7). Si usted ha seguido el curso hasta aquí, los archivos estarán disponibles en la carpeta `r_opas/dados`. Si usted va a estudiar la clase 8 por separado, es importante que sepa que trabajaremos con datos que ya se han trabajado

en clases anteriores. Para seguir esta clase, transfiera los archivos abajo de la carpeta r_opas/backup a la carpeta r_opas/dados. Dichos archivos se refieren a los siguientes conjuntos de datos originales:

- Notificaciones de síndrome gripal realizadas por Maranhão en 2022. Fuente: [eSUS Notifica](#); 17/06/2022.
- Notificaciones de síndrome respiratorio agudo grave (SRAG) realizadas por Maranhão (estado brasileño) entre 2020 y 2022. Fuente: Sistema de Vigilancia Epidemiológica de la Influenza ([SIVEP-Gripe](#)); 17/06/2022.
- Estimaciones de población de los municipios de Maranhão en 2020 y 2021. Fuente: Ministerio de Salud; obtenidas a partir del tabulador de datos [TabNet/DataSUS](#); 17/06/2022.

1. Para seguir la clase

```
# Importar Notificaciones de síndrome gripal en el eSUS
# realizadas por Maranhão, 2022 (Descarga: 17/06/2022)
sg <- import("datos/sg_clase8.rds")
```

```
# Importar Notificaciones de síndrome respiratorio agudo grave en el SIVEP-Gripe
# realizadas por Maranhão, 2020-2022 (Descarga: 17/06/2022)
srag <- import("datos/srag_clase8.rds")
```

```
# Importar estimaciones de población municipales,
# Maranhão, 2020-2021 (Fuente: Ministerio de salud)
pop_21 <- import("datos/pop_21_clase8.rds")
pop_20 <- import("datos/pop_20_clase8.rds")
```

> 2

> Crear y transformar columnas: mutate()

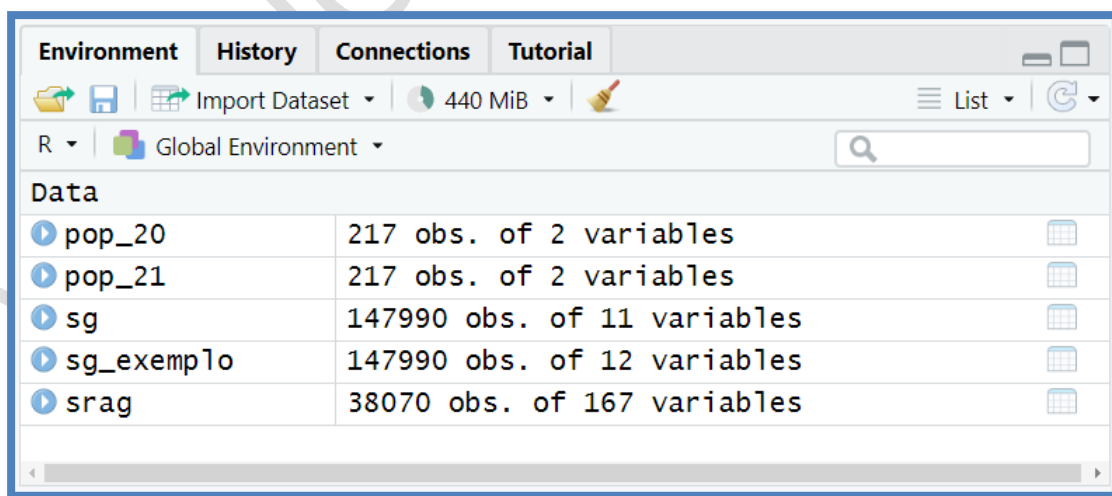
En esta clase seguiremos conociendo funciones útiles para el tratamiento de datos en la rutina de los servicios de vigilancia en salud. Nuestro primer aprendizaje será crear y transformar variables (columnas) en nuestras bases de datos usando la función

`dplyr::mutate()`. El paquete `dplyr` está incluido en el `tidyverse`. Comenzaremos con un ejemplo simple: crear una variable como una copia de otra variable que ya existe en la base de datos.

En el siguiente ejemplo, usamos el signo de asignación (`<-`) para crear el `data.frame` `sg_exemplo` a partir de los datos del otro `data.frame` (`sg`). Con el uso del `pipe` (`%>%`), entregamos el conjunto de datos a la función `mutate()`. En su configuración, creamos la variable `datanotificacao` y le asignamos los datos de la variable `DATANOTIFICACAO`. Recuerde que, en la clase anterior, estandarizamos el uso de letras minúsculas para variables transformadas y letras mayúsculas para variables originales (sin transformación) a lo largo de este curso. Ejecute el código a continuación.

```
# 2. Crear y transformar columnas: mutate() ####  
# Crear una columna con base en otra variable de la base  
sg_exemplo <- sg %>%  
  mutate(datanotificacao = DATANOTIFICACAO)
```

Observe en el panel *environment* (Figura 1) que el nuevo objeto `sg_exemplo` tiene el mismo número de registros que el objeto `sg`, pero con una variable más (10 variables en lugar de 9). Intente abrir la vista previa del `data.frame` `sg_exemplo` y compare el contenido de las dos variables.



Environment		History	Connections	Tutorial
R Global Environment				
Data				
pop_20	217 obs. of 2 variables			
pop_21	217 obs. of 2 variables			
sg	147990 obs. of 11 variables			
sg_exemplo	147990 obs. of 12 variables			
srag	38070 obs. of 167 variables			

Figura 1: Panel *environment* con los objetos.

También tenga en cuenta el resultado de la función `dplyr::glimpse()` para el nuevo objeto `sg_exemplo` que fue creado. Compare la variable `datanotificacao` con su original `DATANOTIFICACAO`. Observe que ambas variables son de tipo `character` (`<chr>`), además, los tres primeros registros de la base son iguales (`"2022/01/01"`, `"2022/03/23"`, `"2022/01/04"`). Luego usted puede borrar este objeto que creamos como ejemplo con la función `rm()` del paquete `base`.

2. Crear y transformar columnas: `mutate()`

```
glimpse(sg_exemplo)
```

```
Rows: 147,990
```

```
Columns: 10
```

```
$ DATANOTIFICACAO    <chr> "2022/01/01", "2022/03/23", "2022/01/04", "20...
$ SEXO               <chr> "Feminino", "Masculino", "Masculino", "Femini...
$ IDADE              <int> 10, 34, 36, 31, 24, 21, 59, 29, 41, 31, 27, 5...
$ DATAINICIOSINTOMAS <chr> "2022/01/01", "2022/01/01", "2022/01/01", "20...
$ EVOLUCAOCASO       <chr> "", "", "Ignorado", "", "", "", "", "", "", ""...
$ CLASSIFICACAOFINAL <chr> "", "", "Descartado", "", "", "", "", "", "", ""...
$ MUNICIPIOIBGE      <int> 2103307, 2107506, 2103000, 2104800, 2100600, ...
$ RESULTADOTESTE     <chr> "", "", "", "", "", "", "", "", ""
$ SINTOMAS           <chr> "Coriza, Dor de Cabeça, Tosse, Dor de Gargant...
$ datanotificacao    <chr> "2022/01/01", "2022/03/23", "2022/01/04", "20...
```

Eliminar el objeto

```
rm(sg_exemplo)
```

Ahora exploraremos la configuración de la función `mutate()` con ejemplos en las bases del curso. Su primer argumento es el nombre de la variable que se creará o transformará. Luego, con un signo de igual (`=`), definimos cómo se configurará esta variable. Por ejemplo, podemos crear una variable y asignarle una expresión matemática o un conjunto de funciones anidadas (una dentro de otra). En el siguiente código, creamos las variables `datanotificacao` y `datainiciosintomas` y les asignamos los valores de sus variables originales transformadas a fecha. Para esta transformación usamos la función `lubridate::as_date()`.

```
# 2. Crear y transformar columnas: mutate() ####
# La función mutate permite múltiples argumentos
sg_exemplo <- sg %>%
  mutate(datanotificacao = as_date(DATANOTIFICACAO),
         datainiciosintomas = as_date(DATAINICIOSINTOMAS))

glimpse(sg_exemplo)
Rows: 147,990
Columns: 11
$ DATANOTIFICACAO    <chr> "2022/01/01", "2022/03/23", "2022/01/04", "2...
$ SEXO               <chr> "Feminino", "Masculino", "Masculino", "Femin...
$ IDADE              <int> 10, 34, 36, 31, 24, 21, 59, 29, 41, 31, 27, ...
$ DATAINICIOSINTOMAS <chr> "2022/01/01", "2022/01/01", "2022/01/01", "2...
$ EVOLUCAOCASO       <chr> "", "", "Ignorado", "", "", "", "", "", "", ...
$ CLASSIFICACAOFINAL <chr> "", "", "Descartado", "", "", "", "", "", "", ...
$ MUNICIPIOIBGE      <int> 2103307, 2107506, 2103000, 2104800, 2100600,...
$ RESULTADOTESTE     <chr> "", "", "", "", "", "", "", "", "", "", "", ...
$ SINTOMAS           <chr> "Coriza, Dor de Cabeça, Tosse, Dor de Gargan...
$ datanotificacao    <date> 2022-01-01, 2022-03-23, 2022-01-04, 2022-01...
$ datainiciosintomas <date> 2022-01-01, 2022-01-01, 2022-01-01, 2022-01...
```

Vea en el resultado de la función `glimpse()` que se crearon variables de clase fecha (`<date>`) y que, aunque los datos son los mismos, ahora vemos en las variables creadas el formato estándar AAAA-MM-DD (año-mes-día), con guion (-) en lugar de barra oblicua (/) y sin las comillas. También tenga en cuenta que la función `mutate()` permite múltiples argumentos, por lo que se puede hacer numerosas transformaciones al mismo tiempo.

La función `lubridate::as_date()` supone que los datos están en el formato estándar del idioma inglés (año-mes-día), que también es el formato estándar para R. En el código anterior, no tuvimos problemas, porque los datos de las variables originales `DATANOTIFICACAO` y `DATAINICIOSINTOMAS` estaban en ese formato estándar. Pero nuestros datos no siempre se almacenan así. En los siguientes temas, presentaremos

algunas funciones útiles para transformar variables según su clase, comenzando por las variables de tipo fecha.

2.1 - Transformaciones de variables de tipo fecha

Existen diferentes formas de almacenar variables de tipo fecha en R. Por ejemplo, usted puede querer extraer una parte de una fecha, como solo el año, el mes o el día. O, si es importante para la vigilancia de la enfermedad o agravio a la salud que está describiendo (como en el caso de violencias y accidentes), usted puede almacenar la fecha y la hora en la misma celda (investigue sobre las clases `POSIXt`, `POSIXct` y/o `POSIXlt`). También le puede interesar calcular la semana epidemiológica de inicio de los síntomas de los casos que va a analizar, entre otros ejemplos.

Además, hay varias formas de manipular variables de tipo fecha en R. Sugerimos el paquete `lubridate`, ya que sus funciones son versátiles y resuelven la mayoría de las transformaciones que solemos realizar para trabajar con fechas de eventos epidemiológicos. El paquete también está incluido en el `tidyverse`, pero no se activa automáticamente y debe especificarse para cargarse, lo que ya hicimos al comienzo de la clase.

Para transformar correctamente variables de tipo fecha en R, es necesario identificar el estándar con el que están registrados los datos en la base de datos que se analizará. Esto suele generar un poco de confusión entre los(las) usuarios(as) principiantes de R, ya que suele ser más intuitivo pensar al revés: al configurar la variable, imaginamos cómo queremos presentarla y no cómo se presentan los datos sin tratamiento. ¡Así que esté atento(a)!

Una vez identificado el estándar, usted debe elegir la función de transformación correspondiente. Por ejemplo, si los datos están en el formato estándar `DD-MM-AAAA` (día-mes-año), debe utilizar la función `lubridate::dmy()`. Los tres caracteres en el nombre de la función son una referencia a *day-month-year*, que en español significa día-mes-año.

En el siguiente código, creamos las variables `dt_notific`, `dt_sin_pri`, `dt_nasc` y `dt_interna` y les asignamos la función `dmy()` para transformar los datos originales

de cada variable en clase fecha (<date>). Elegimos la función `dmy()` porque los datos originales están en el formato DD-MM-AAAA.

Luego ejecutamos la función `glimpse()` con variables seleccionadas para comprobar el resultado. Compare las nuevas variables (en minúsculas) con las variables originales correspondientes (en mayúsculas) en el resultado de la función `glimpse()`. Observe que la función `dmy()` hizo las mismas transformaciones que la función `as_date()`, que vimos arriba: las variables se crearon con clase fecha (<date>), la vista ahora está en el formato estándar AAAA-MM-DD (año-mes-día), con guion (-) y sin las comillas.

```
# 2. Crear y transformar columnas: mutate() ####
# 2.1 - Transformaciones de variables de tipo fecha
# Día-Mes-Año (Day-Month-Year)
srag <- srag %>%
  mutate(dt_notific = dmy(DT_NOTIFIC),
         dt_sin_pri = dmy(DT_SIN_PRI),
         dt_nasc = dmy(DT_NASC),
         dt_interna = dmy(DT_INTERNA))

# glimpse() con variables seleccionadas para observar el resultado
srag %>%
  select(DT_NOTIFIC, dt_notific,
         DT_SIN_PRI, dt_sin_pri,
         DT_NASC, dt_nasc) %>%
  glimpse()
Rows: 38,070
Columns: 6
$ DT_NOTIFIC <chr> "01/05/2020", "23/06/2020", "10/01/2021", "30/04/2020", ...
$ dt_notific <date> 2020-05-01, 2020-06-23, 2021-01-10, 2020-04-30, 2020-07...
$ DT_SIN_PRI <chr> "29/04/2020", "21/06/2020", "10/12/2020", "20/04/2020", ...
$ dt_sin_pri <date> 2020-04-29, 2020-06-21, 2020-12-10, 2020-04-20, 2020-07...
$ DT_NASC <chr> "04/01/1947", "05/03/1966", "23/09/1963", "07/11/1964", ...
$ dt_nasc <date> 1947-01-04, 1966-03-05, 1963-09-23, 1964-11-07, 1950-06...
```

Hay otras funciones de clasificación de fechas en el paquete `lubridate` con esa misma estructura de tres caracteres. Por ejemplo, en el siguiente código presentamos las funciones `mdy()` para cuando los datos originales estén en el formato estándar mes-día-año, `ydm()` para año-día-mes y `ymd()` para año-mes-día. Es interesante notar que las funciones del paquete `lubridate` comprenden varios formatos diferentes y que el resultado de todas las funciones a continuación es el mismo para R: la fecha 2020-03-11.

```
# 2. Crear y transformar columnas: mutate() ####
# 2.1 - Transformaciones de variables de tipo fecha
# Mes-Día-Año (Month-Day-Year)
mdy("11/03/2020")
[1] "2020-03-11"

mdy("Mar 11 20")
[1] "2020-03-11"

# Año-Día-Mes (Year-Day-Month)
ydm("2020 11 03")
[1] "2020-03-11"

ydm("2020 11 Março")
[1] "2020-03-11"

# Año-Mes-Día (Year-Month-Day)
ymd("20200311")
[1] "2020-03-11"
```

Una vez que sus variables estén configuradas correctamente para la clase fecha, hay numerosas funciones que usted puede usar para su manejo. En el siguiente código, presentamos las funciones `month()`, para extraer solo el mes, y `year()`, para extraer sólo el año de una fecha. Crearemos las variables `mes_notific` en el objeto `sg` (mes de notificación de los casos de SG) y `ano_notific` en el objeto `srag` (año de notificación de los casos de SRAG). Entonces usaremos `count()`, que usted aprendió

en la última clase, para ver una tabla exploratoria de las nuevas variables que acabamos de crear. También puede utilizar la función `day()` para extraer sólo el día de una fecha.

```
# 2. Crear y transformar columnas: mutate() ####  
# 2.1 - Transformaciones de variables de tipo fecha  
# Solo el mes  
sg <- sg %>%  
  mutate(mes_notific      = month(datanotificacao)) %>%  
  mutate(mes_ini_sint     = month(datainiciosintomas))  
  
sg %>%  
  count(mes_notific)  
# A tibble: 9 x 2  
#   mes_notific     n  
#   <dbl> <dbl>  
1         1 85123  
2         2 38877  
3         3 12136  
4         4  5238  
5         5  3814  
6         6  2797  
7        11     1  
8        12     3  
9        NA     1  
  
# Solo el año  
sg <- sg %>%  
  mutate(ano_ini_sint     = year(datainiciosintomas))  
  
sg %>%  
  count(ano_ini_sint)  
# A tibble: 3 x 2  
#   ano_ini_sint     n  
#   <dbl> <dbl>  
1      2020 14819  
2      2021 20040  
3      2022  3211
```

En el análisis de datos para la vigilancia de eventos en salud, también es muy común trabajar con fechas transformadas a semanas epidemiológicas (SE). El calendario de

semanas epidemiológicas es un consenso internacional sobre un período estándar para agrupar eventos epidemiológicos. Los 365 días del año se agrupan en 52 o 53 SE, de domingo a sábado, para facilitar la comparabilidad de la situación epidemiológica en diferentes localidades o períodos del año. Por convención, la primera SE del año es la que contiene el mayor número de días de enero y la última SE es la que contiene el mayor número de días de diciembre.

La principal consecuencia práctica del calendario epidemiológico es que el año natural y el año epidemiológico pueden no coincidir para las fechas de finales de diciembre o principios de enero. En el siguiente ejemplo, presentamos la función `epiyear()` y `epiweek()` del paquete `lubridate`, destinadas a calcular el año y la semana epidemiológica, respectivamente. Podemos ver que ambas fechas 31/12/2021 y 01/01/2022 muestran como resultado la semana epidemiológica 52 y el año epidemiológico de 2021, aunque son de años naturales diferentes.

```
# 2. Crear y transformar columnas: mutate() ####  
# 2.1 - Transformaciones de variables de tipo fecha  
# Año epidemiológico  
epiyear("2022/01/01")  
[1] 2021  
  
epiyear("2021/12/31")  
[1] 2021  
  
# Semana epidemiológica  
epiweek("2021/12/31")  
[1] 52  
  
epiweek("2022/01/01")  
[1] 52
```

Para calcular el año o la semana epidemiológica de una fecha en su base de datos, informe como argumento de las funciones `epiyear()` y `epiweek()` el conjunto de datos que se utilizará como referencia (variable de clase fecha). Observe y ejecute los

códigos a continuación. En estos ejemplos, no vamos a crear o transformar variables en el objeto `sg`, ya que nuestro interés es solo realizar una tabla exploratoria de los años y meses de notificación de los casos.

```
# 2. Crear y transformar columnas: mutate() ####  
# 2.1 - Transformaciones de variables de tipo fecha
```

```
# Semana epidemiológica
```

```
sg %>%
```

```
  mutate(se_notific = epiweek(datanotificacao))%>%
```

```
  count(se_notific)
```

	se_notific	n
1	1	8663
2	2	17899
3	3	21689
4	4	30644
5	5	20029
6	6	10075
7	7	8388
8	8	6292
9	9	1779
10	10	3766
11	11	2714
12	12	2394
13	13	2258
14	14	1831
15	15	1195
16	16	901
17	17	760
18	18	765
19	19	896
20	20	626
21	21	1070
22	22	1108
23	23	1305
24	24	841
25	46	1
26	48	1

```

27      51      2
28      52     97
29      NA      1

# Año epidemiológico
srag %>%
  count(epiyear(dt_notific))
  epiyear(dt_notific)    n
1                2020 14835
2                2021 20041
3                2022  3194

```

Entre las funciones útiles para trabajar con fechas, también presentamos las funciones `today()`, para la fecha de hoy, `date()` y `now()`, para fecha y hora de ahora. Pruebe ejecutar el siguiente código para ver que el resultado se construirá con base en los datos del sistema. Para utilizar estas funciones, es importante fijarse en la fecha o la hora de su computadora, para que el resultado corresponda al momento exacto en que está trabajando.

```

# 2. Crear y transformar columnas: mutate() ####
# 2.1 - Transformaciones de variables de tipo fecha

# Fecha de hoy
today()
[1] "2022-07-28"
epiweek(today())
[1] 30

# Fecha y hora de ahora
date()
[1] "Thu Jul 28 08:23:10 2022"

now()
[1] "2022-07-28 08:23:24 -03"

```

Otro aspecto importante que se debe destacar es el hecho de que R almacena internamente las fechas como números. Por mucho que veamos las variables fecha en el formato AAAA-MM-DD, internamente R cuenta el número de días desde su fecha de "origen" (1 de enero de 1970). Esta estrategia permite que R trate las fechas como variables continuas y, así, realice operaciones especiales, como el cálculo de la distancia entre las fechas.

En el siguiente código, creamos la variable `delay_not` como la diferencia entre la fecha de notificación y la fecha de inicio de los síntomas de los casos notificados de SG. Al principio, hicimos una simple sustracción con el operador `"-"`. Luego hicimos un `count()` de la nueva variable creada y presentamos aquí en el material solo una parte del resultado (usted podrá observar más líneas en su panel *console*).

Tenga en cuenta que la fecha de notificación es posterior a la fecha de inicio de los síntomas, por lo que está en primer lugar en la sustracción. La presencia de resultados negativos, aunque son la minoría, indica la presencia de inconsistencia en los registros de notificación, con fecha de inicio de los síntomas posterior a la de notificación. También observe que en nuestra tabla exploratoria vemos el término `"days"` ("días", en español) al lado de cada categoría de la nueva variable `delay_not`. Y que si ejecutamos la función `class()`, la variable `delay_not` fue creada con clase `difftime`.

```
# 2. Crear y transformar columnas: mutate() ####
# 2.1 - Transformaciones de variables de tipo fecha
# Fechas y Operadores aritméticos
sg <- sg %>%
  mutate(delay_not = datanotificacao - datainiciosintomas)

# Tabulación y clase de la variable delay_not para observar el resultado
sg %>%
  count(delay_not)
  se_notific      n
17      -5 days    3
18      -3 days    2
19      -2 days    2
20      -1 days    2
```

```

21      0 days 10463
22      1 days  7313
23      2 days 13453
24      3 days 20012
25      4 days 17086
26      5 days 12679

```

```

class(sg$delay_not)
[1] "difftime"

```

Para almacenar conteos de tiempo en variables numéricas, será necesario utilizar otras funciones de apoyo. En el siguiente código, sobrescribiremos la variable `delay_not` con la función `mutate()` para calcular la diferencia entre la fecha de notificación y la fecha de inicio de los síntomas de los casos notificados de SG, como hicimos anteriormente. Presentamos la función `lubridate::time_length()` para este cálculo. En el primer argumento de la función `time_length()`, informamos la función `lubridate::interval()` con las fechas en orden de calendario y en el segundo argumento la unidad de tiempo en días ("days"). Tenga en cuenta que llegamos al mismo resultado del `count()` que hicimos arriba, cuando construimos la variable como una sustracción. Pero esta vez tenemos una variable numérica.

```

# 2. Crear y transformar columnas: mutate() ####
# 2.1 - Transformaciones de variables de tipo fecha
# Conteo de tiempo (numérico, en días)
sg <- sg %>%
  mutate(delay_not =
    time_length(interval(datainiciosintomas,
                        datanotificacao),
                "days"))

# Tabulación y clase de la variable delay_not para observar el resultado
sg %>%
  count(delay_not)
17      -5      3

```



```

18      -3      2
19      -2      2
20      -1      2
21       0 10463
22       1  7313
23       2 13453
24       3 20012
25       4 17086
26       5 12679
class(sg$delay_not)
[1] "numeric"

```

Tenga cuidado de informar el orden correcto de las variables fecha para la función `interval()`, porque en ella declaramos primero la fecha más antigua (menor, para R). Si en sus resultados usted encuentra un número discrepante de resultados negativos, posiblemente lo informó en el orden inverso. En el siguiente código, observe que la función `interval()` comprende otras unidades de tiempo, tales como "year" para año, "month" para mes, "week" para semanas y "second" para segundos. Si es necesario, utilice la función `round()` para redondear este conteo.

```

# 2. Crear y transformar columnas: mutate() ####
# 2.1 - Transformaciones de variables de tipo fecha
# Conteo de tiempo (numérico, con meses y redondeo)
sg %>%
  mutate(delay_not =
           time_length(interval(datainiciosintomas,
                                datanotificacao),
                        "months"),
           delay_not = round(delay_not, 0)) %>%

count(delay_not)
  delay_not      n
1      -24      1
2      -12      1
3       -8      1

```

```

4      -2      2
5      -1     11
6       0 119874
7       1  17542
8       2   7393
9       3   2066
10      4    945
11      5    153
12     NA      1

```

```

class(sg$delay_not)
[1] "numeric"

```

Las funciones del paquete `lubridate` funcionan bien con variables de tipo carácter y numéricas. Al realizar transformaciones de variables de tipo fecha, recuerde siempre verificar la clase de la variable original que será manejada. En caso de que usted encuentre un error, considere primero transformar la variable a carácter y luego hacer la transformación con la función deseada. Para conocer más funcionalidades del paquete `lubridate`, acceda a *cheat sheet* (hoja de consultas) del paquete en <https://github.com/rstudio/cheatsheets/blob/main/lubridate.pdf>.

Y para terminar su estudio de las transformaciones de variables de clase `date`, es importante que usted sepa que R tiene una codificación específica para su presentación. El método `strptime` está incluido en el paquete base y se ha reproducido en numerosas funciones fuera de él. Él reúne funciones y códigos útiles de formateo de fechas, a través de una conversión entre fechas y textos.

Por ejemplo, en el siguiente código usamos la función `format()` con dos argumentos: el primero es la fecha que será formateada y el segundo es una codificación propia del método `strptime`, en que `%d` significa día en dígitos, `%m` significa mes en dígitos, y `%Y` significa año, también en dígitos. El código se informó entre comillas y con barras entre las tres partes. Como resultado, encontramos el output `[1] "11/03/2020"`. En el segundo ejemplo, informamos el formato `"%d %B %Y"` y obtuvimos el output `[1] "11 março 2020"`, porque no pusimos barras y el código `%B` significa mes por extenso.

```
# 2. Crear y transformar columnas: mutate() ####  
# 2.1 - Transformaciones de variables de tipo fecha  
format(as_date("2020-03-11"), "%d/%m/%Y")  
[1] "11/03/2020"  
  
format(as_date("2020-03-11"), "%d %B %Y")  
[1] "11 março 2020"
```

Para conocer la lista completa de códigos del método `strptime`, acceda a su documentación ejecutando el comando `?strptime`. Usted encontrará una extensa lista de posibles formatos, ¡pero no tiene que memorizarla! Siempre podrá consultar la documentación de las funciones para formatos específicos como estos. Además, por ahora es suficiente que entienda el método.

2.2 - Transformación de variables numéricas

En el análisis de datos epidemiológicos, a menudo manejamos dos tipos de variables numéricas: las discretas (números enteros) para conteos generales, como el número de casos o días de hospitalización, y las continuas (número con decimales), generalmente para valores calculados, como la tasa de ataque o letalidad de una enfermedad o agravio a la salud. Como vimos en el módulo anterior, en R decimos que tales variables tienen las clases `integer` y `numeric`, respectivamente.

Podemos usar las funciones `as.integer()` y `as.numeric()` del paquete `base` para transformar variables en `integer` y `numeric`. Recuerde que la clase `numeric` acepta todos los números reales (con o sin valores decimales), por lo que muchas veces se suele priorizarla. Tales variables de clase `numeric` también se llaman `double`, que significa “doble” en español. Por ello, aparecen en el resultado de la función `glimpse()` como `<dbl>`, precisamente por la característica de que contienen ambos tipos de variables.

Siga en el código de abajo la transformación de la variable referente a la edad de los casos de SRAG para `integer`. A continuación, solicitamos un `glimpse()` de la base con algunas variables seleccionadas: `NU_EDAD_N` (variable edad original), `edad`

(variable transformada, que acabamos de crear) y `mes_sin_pri` (variable del mes de inicio de los síntomas creada anteriormente, con la función `month()`). Incluimos esta última solo para mostrar que, aunque el mes es un número entero, el dato se almacenó en una variable de tipo `numeric`.

```
# 2. Crear y transformar columnas: mutate() ####
# 2.2 - Transformación de variables numéricas
# Números enteros
srag <- srag %>%
  mutate(idade = as.integer(NU_IDADE_N))

# glimpse() con variables seleccionadas para observar el resultado
srag %>%
  select(NU_IDADE_N, idade, mes_sin_pri) %>%
  glimpse()

Rows: 38,070
Columns: 3
$ NU_IDADE_N  <chr> "73", "54", "57", "55", "70", "84", "51", "11", ...
$ idade      <int> 73, 54, 57, 55, 70, 84, 51, 11, 33, 93, 35, 61, ...
$ mes_sin_pri <dbl> 4, 6, 12, 4, 7, 12, 6, 7, 3, 6, 5, 6, 5, 6, 4, ...
```

2.3 - Transformaciones de variables de tipo factor

Vimos en el módulo anterior que, para R, los factores son las variables ordinales. También vimos la función `factor()` del paquete base, que es capaz de crear factores y recodificar sus etiquetas. En el código de abajo, rescatamos la configuración de la función `factor()` para construir la variable `sexo` en la base de SRAG, con base en la variable original `CS_SEX0`. Observe el `count()` de las dos variables y vea que la disposición de las categorías ha cambiado al orden informado en el argumento `levels =`, mientras que las etiquetas pasaron a expresar las palabras completas, como en el argumento `labels =`.

```
# 2. Crear y transformar columnas: mutate() ####
2.3 - Transformación de variables de tipo factor
# Crear factor y cambiar las etiquetas
srag <- srag %>%
  mutate(sexo = factor(CS_SEXO,
                        levels = c("F", "M", "I"),
                        labels = c("Feminino", "Masculino", "Ignorado")))

srag %>%
  count(CS_SEXO)
  CS_SEXO      n
1      F 16799
2      I    19
3      M 21252

srag %>%
  count(sexo)
      sexo      n
1 Feminino 16799
2 Masculino 21252
3 Ignorado   19
```

Entre los principales paquetes de tidyverse, usted también puede encontrar el paquete forcats, que reúne funciones específicas para manejar los factores. En el siguiente código, presentamos la función `forcats::fct_infreq()`, que crea factores en el orden inverso de la frecuencia de los datos. Note que usamos la función anidada a la función `count()` para hacer una tabla de frecuencia exploratoria de la función `sexo` que acabamos de crear, pero sin transformación en la base de datos. Compare el resultado de la función `count()` con el código anterior y observe que ahora los casos de SRAG se presentaron en orden descendente del número de casos en cada sexo.

```
# 2. Crear y transformar columnas: mutate() ####
2.3 - Transformación de variables de tipo factor
# Factores en orden inverso
srag %>%
```

```
count(fct_infreq(sexo))
fct_infreq(sexo)      n
1      Masculino 21252
2      Femenino 16799
3      Ignorado   19
```

De hecho, en análisis descriptivos de bases de datos epidemiológicas, el principal uso de variables con la clase `factor` es para la visualización de datos. En general, la función `factor()` del paquete `base` será suficiente para las transformaciones que usted deberá realizar. Para conocer otras funciones del paquete `forcats`, usted también puede acceder a *cheat sheet* del paquete en: <https://github.com/rstudio/cheatsheets/blob/main/factors.pdf>.

2.4 - Transformaciones de variables de tipo texto

Otro tipo de variables común en análisis de datos epidemiológicos son aquellas que almacenan cadenas de caracteres o textos (`<chr>`). Hay algunas situaciones que pueden generar variables de esta clase. Por ejemplo, si el cuestionario de investigación epidemiológica o formulario de notificación que originó la base de datos contiene preguntas abiertas en las que el(la) encuestado(a) puede escribir libremente, si la variable es categórica, si la variable almacena números y letras (dígitos alfanuméricos).

Como vimos en el módulo anterior, en R llamamos dichas variables de *strings*, que son de clase `character` (`<chr>`). Y para trabajar con esta clase de variables, sugerimos usar el paquete `stringr`. El paquete está incluido en `tidyverse` y ya se activó automáticamente con él, por lo que no fue necesario especificarlo en la sección inicial de esta clase, como hicimos con el paquete `lubridate`.

Usted ya sabe que para crear valores almacenando caracteres en R debemos usar comillas. Ahora vamos a conocer la función `as.character()` del paquete `base` para transformar los datos de nuestros *data.frames* en caracteres. En el siguiente código, creamos el objeto `caractere` y almacenamos en él el número 12345. El objeto fue creado como un número, como lo confirma la función `class()`. Luego sobrescribimos

este objeto asignando la función de reclasificación as.character() y confirmamos la transformación con la función class().

```
# 2. Crear y transformar columnas: mutate() ####
```

2.4 - Transformación de variables de tipo texto

```
# Transformar en carácter
```

```
caractere <- 12345
```

```
class(caractere)
```

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

```
caractere <- as.character(caractere)
```

```
class(caractere)
```

```
[1] "character"
```

Ahora conoceremos algunas de las funciones del paquete stringr. En el siguiente código, creamos el objeto texto para algunos ejemplos y almacenamos en él la palabra "Epidemiología". Luego presentamos la función stringr::str_sub(), que extrae "partes" de una cadena de caracteres, para recortar partes de la palabra "Epidemiología" con base en la posición de los caracteres.

Para configurar la función str_sub(), el argumento string = indica el texto que se quiere recortar, el argumento start = indica la posición del carácter donde comenzará y el argumento end = indica la posición del carácter donde terminará. Si no se especifica el nombre de los argumentos, la función considerará el orden (string = , start = , end =). Por esta razón, en el siguiente código, ambas cadenas de caracteres llegaron al resultado [1] "Epi".

```
# 2. Crear y transformar columnas: mutate() ####
```

2.4 - Transformación de variables de tipo texto

```
# Recortar textos según la posición
```

```
texto <- "Epidemiología"
```

```
class(texto)
```

```
[1] "character"
```

```
str_sub(string = "Epidemiologia",
        end = 3,
        start = 1)
```

```
[1] "Epi"
```

```
str_sub("Epidemiologia", 1, 3)
```

```
[1] "Epi"
```

Podemos usar la función `str_sub()` para extraer el código y el nombre del municipio en las bases de estimaciones de población. Observe las funciones a continuación. Primero usamos la función `glimpse` para comprobar la estructura general de la base. Luego creamos las variables `cod_ibge` y `nome_municipio` con la función `mutate()`, asignándoles la función `str_sub()`.

Creamos la variable `cod_ibge` con la intención de extraer los primeros seis dígitos de la variable original `MUNICIPIO` (solo el código), luego delimitamos el principio y el final con los argumentos `start =` y `end =` con sus nombres ocultos (solo informamos la posición de caracteres). A su vez, el nombre del municipio tiene un número de caracteres diferente en cada registro. Por ejemplo, “Açailândia” tiene 10 caracteres, mientras que “Alfonso Cunha” tiene 12. Por esta razón, solo delimitamos la posición del carácter de inicio con `start = 8`.

```
# 2. Crear y transformar columnas: mutate() ####
```

```
2.4 - Transformación de variables de tipo texto
```

```
glimpse(pop_20)
```

```
Rows: 217
```

```
Columns: 2
```

```
$ MUNICIPIO <chr> "210005 Açailândia", "210010 Afonso Cunha", "210015 Água Doce,...
```

```
$ POP_2020 <dbl> 113121, 6578, 12652, 22112, 26757, 8189, 27858, 31943, 11212,...
```

```
# Separar código y nombre del municipio
```

```
pop_20 <- pop_20 %>%
```

```
  mutate(cod_ibge = str_sub(MUNICIPIO,1,6)) %>%
```

```
  mutate(nome_municipio = str_sub(MUNICIPIO, start=8))
```



```
pop_21 <- pop_21 %>%
  mutate(cod_ibge = str_sub(MUNICIPIO,1,6)) %>%
  mutate(nome_municipio = str_sub(MUNICIPIO,8))
```

Cuando ejecutamos nuevamente la función `glimpse()`, podemos ver que ambas variables fueron creadas como queríamos, sin embargo, ambas fueron creadas como caracteres. De hecho, el producto de la función `str_sub()` siempre será de clase carácter. Usted puede mantener los números como clase `character` en R. No está necesariamente incorrecto y es posible que no tenga problemas para realizar sus análisis. En nuestro ejemplo, sin embargo, transformaremos la variable código del municipio en clase `integer`, como en el código siguiente.

2. Crear y transformar columnas: `mutate()`

2.4 - Transformación de variables de tipo texto

```
glimpse(pop_20)
Rows: 217
Columns: 4
$ MUNICIPIO      <chr> "210005 Açailândia", "210010 Afonso Cunha", "210015 Ág...
$ POP_2020       <dbl> 113121, 6578, 12652, 22112, 26757, 8189, 27858, 31943,...
$ cod_ibge       <chr> "210005", "210010", "210015", "210020", "210030", "210...
$ nome_municipio <chr> "Açailândia", "Afonso Cunha", "Água Doce do Maranhão",...
```

Atención a los números en `str_sub()`

```
pop_20 <- pop_20 %>%
  mutate(cod_ibge = as.integer(cod_ibge))
```

También podemos usar el signo de menos en los argumentos de la función `str_sub()` para indicar que el conteo se realizará en orden inverso. En el siguiente ejemplo, recortamos el objeto texto contando los caracteres hacia atrás. Extrajimos el término "logia", porque contamos la letra "I" como quinto término y la letra "a" como el primero. Cabe señalar que, para la función `str_sub()`, los espacios se consideran

caracteres y deben incluirse en el conteo. En el ejemplo, extrajimos el término “emergências sanitarias” informando -22 para el argumento start =. Tenga en cuenta que hay 21 caracteres en el término y que el conteo consideró el espacio entre "emergências" y "sanitárias".

2. Crear y transformar columnas: mutate()

2.4 - Transformación de variables de tipo texto

Posição em ordem inversa

```
str_sub(texto, -5, -1)
```

```
[1] "logia"
```

Espacios son caracteres

```
str_sub("Vigilância epidemiológica em emergências sanitárias", -22)
```

```
[1] "emergências sanitárias"
```

Las funciones del paquete `stringr` tienen una gramática similar, todas comienzan con el prefijo "str_". Si usted ingresa este prefijo en el panel editor o en el panel consola, podrá ver una lista de funciones del paquete `stringr` que puede explorar (Figura 2).

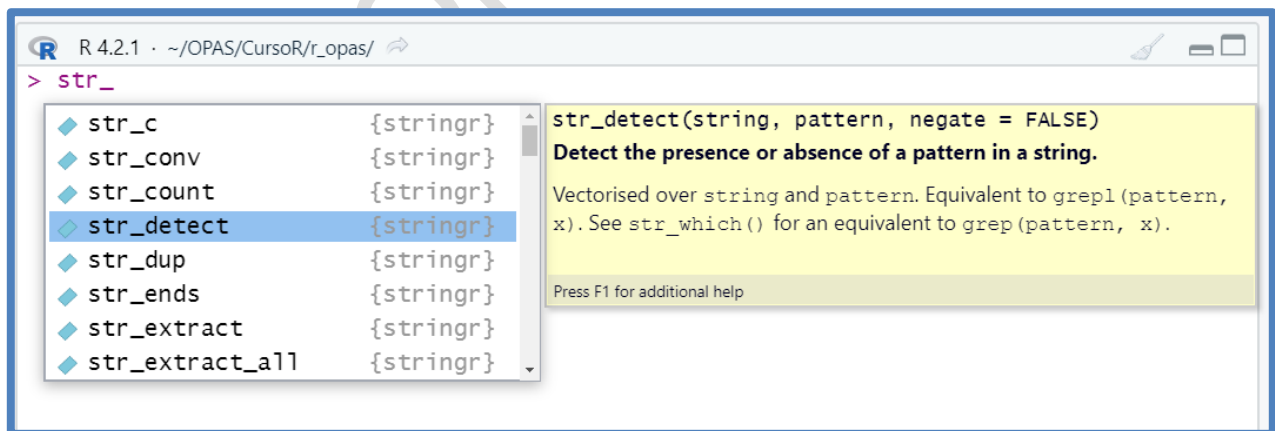


Figura 2: Lista de funciones del paquete `stringr` comenzando con el prefijo "str_".

Otra función útil del paquete `stringr` para el tratamiento de datos en las bases de los servicios de vigilancia epidemiológica es la función `str_detect()`, destinada a encontrar un estándar dentro de un texto. Esta función puede ser útil para identificar expresiones en variables que se originaron en preguntas abiertas o que concatenan respuestas de diferentes preguntas en el cuestionario.

Como ejemplo, intentaremos identificar la palabra “tosse” en la variable `SINTOMAS` de la base de Síndromes Gripales. Note que el *output* (resultado) del código fue `FALSE` para todos los registros en el primer comando. En el segundo, usamos la primera letra de la palabra en mayúsculas y obtuvimos el resultado `TRUE` para algunos registros. Con este ejemplo, resaltamos la importancia de estandarizar los textos al trabajar con `strings`, como veremos a continuación.

2. Crear y transformar columnas: `mutate()`

2.4 - Transformación de variables de tipo texto

Encontrar partes de texto

```
str_detect(sg$SINTOMAS,  
           pattern = "tosse")
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
str_detect(sg$SINTOMAS,  
           pattern = "Tosse")
```

```
[1] TRUE FALSE TRUE FALSE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
[13] FALSE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
[25] TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
```

En el siguiente código, presentamos la función `str_starts()`, destinada a identificar estándares en el inicio de `strings`. Primero identificamos qué registros `SRAG` tenían contenido en la variable `CO_MUN_RES`, referente al código del municipio de residencia, comenzando por “21”, código del estado de Maranhão. El resultado fue `TRUE` para la mayoría de los registros. Luego usamos la función `str_starts()` anidada a la función `table()`. Contabilizamos que 321 de los 38.070 casos de `SRAG` notificados por

Maranhão son de personas que viven fuera del estado y que la tos fue un síntoma registrado en 98.384 de los 147.990 casos de SG registrados por el estado.

```
# 2. Crear y transformar columnas: mutate() ####
2.4 - Transformación de variables de tipo texto
# Identificar caracteres al principio de un texto
str_starts(srag$CO_MUN_RES, "21")
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[13] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[25] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

# Tabular en cuántos casos de srag el municipio de residencia comienza con "21"
table(str_starts(srag$CO_MUN_RES, "21"))
FALSE TRUE
 321 37747

# Tabular cuántas veces aparece la palabra diarrea en la variable signos y
síntomas
table(str_detect(sg$SINTOMAS,
                  pattern = "Tosse"))
FALSE TRUE
49606 98384
```

Algunas funciones del paquete stringr pueden ser de interés para preparar strings antes de buscar estándares. En el siguiente código, creamos la variable síntomas como una transformación de la variable original SINTOMAS. Luego ejecutamos un glimpse() con solo estas dos variables. Al usar la función str_to_upper(), vea que todas las palabras cambiaron a mayúsculas.

```
# 2. Crear y transformar columnas: mutate() ####
2.4 - Transformación de variables de tipo texto
# Cambiar un texto a mayúsculas
sg <- sg %>%
  mutate(sintomas = str_to_upper(SINTOMAS))

glimpse(sg %>%
```

```
select(SINTOMAS, sintomas))
Rows: 147,990
Columns: 2
$ SINTOMAS <chr> "Coriza, Dor de Cabeça, Tosse, Dor de Garganta", "Dor de Cab...
$ sintomas <chr> "CORIZA, DOR DE CABEÇA, TOSSE, DOR DE GARGANTA", "DOR DE CAB..."
```

Usted puede usar la función `str_to_lower()` para transformar todo el texto a minúsculas, `str_trim()` para eliminar todos los espacios excesivos al principio y al final del texto, `str_squish()` para eliminar espacios excesivos entre las palabras, al principio y al final, `str_replace()` para sustituir estándares, entre muchas otras funciones.

Si es necesario, busque paquetes en Internet para realizar transformaciones adicionales. Por ejemplo, el paquete `abjutils` fue creado en Brasil por la Asociación Brasileña de Jurimetría para manejar los procesos brasileños y reúne funciones útiles para la limpieza de textos, como la función `rm_accent()`, utilizada para eliminar los acentos del texto o los separadores del código (como "-" o ".").

```
# 2. Crear y transformar columnas: mutate() ####
2.4 - Transformación de variables de tipo texto
# Convierte en minúsculas
str_to_lower(" vigilância epidemiológica em emergências sanitárias")
[1] " vigilância epidemiológica em emergências sanitárias"

# Elimina espacios al principio y al final
str_trim(" vigilância epidemiológica em emergências sanitárias ",
  side = "both")
[1] "vigilância epidemiológica em emergências sanitárias"

# Elimina espacios excesivos
str_squish(" vigilância epidemiológica em emergências sanitárias ")
[1] "vigilância epidemiológica em emergências sanitárias"

# Elimina acentos
abjutils::rm_accent("vigilância epidemiológica em emergências sanitárias")
```

Para manejar `strings` es importante que usted conozca la existencia de REGEX, que es una abreviatura en inglés para *regular expressions*. Es una estandarización de expresiones regulares (en español). Con REGEX, podemos buscar, validar y cambiar cualquier estándar de caracteres en textos.

En el siguiente código, primero solicitamos a R que sustituya por un espacio (" ") todos los caracteres que no sean letras en la palabra expresión "Epidemi010gi@" (escrita con cero y @ en lugar de las letras "o" y "a". Para ello utilizamos la función `str_replace_all()` y la expresión REGEX "`^[[:alpha:]]`". Observe el resultado y las sustituciones que se hicieron.

Luego contamos el número de casos notificados de SG que tenían Secreción Nasal o Tos entre los síntomas con la función `table()`. Para esto, anidamos la función `str_detect()` y usamos la expresión REGEX "`Coriza|Tosse`".

```
# 2. Crear y transformar columnas: mutate() ####
```

2.4 - Transformación de variables de tipo texto

```
# Elimina caracteres que no son letras y los sustituye por espacios.
```

```
str_replace_all("Epidemi010gi@", "^[[:alpha:]]", " ")
```

```
1] "Epidemi 1 gi "
```

```
# Tabular cuántos registros tuvieron Secreción Nasal o Tos
```

```
table(str_detect(sg$SINTOMAS,  
                pattern = "Coriza|Tosse"))
```

```
FALSE TRUE
```

```
35871 112119
```

Si necesita trabajar con textos dinámicos, puede ser interesante explorar las funcionalidades de la función `str_glue()`. Ella puede ser especialmente útil cuando manejamos conjuntos de datos que se actualizan con frecuencia. Por ejemplo, observé el *output* del comando a continuación: "147990 casos de SG com início dos sintomas até

16 jun 2022". Para ejecutar este resultado, informamos la expresión entre comillas y con dos tramos dinámicos, que se declaran entre corchetes `{}`. En el primer tramo, la función `nrow()` muestra un conteo del número de líneas del *data.frame* `sg`, mientras que en el segundo la función `max()` muestra la fecha máxima de inicio de los síntomas de los casos notificados. Y la función `format()` configura la fecha, como vimos en la transformación de fechas arriba. También usamos REGEX en la expresión: el tramo `\n`, lo que significa una instrucción para que el texto continúe en una nueva línea.

```
# 2. Crear y transformar columnas: mutate() ####
```

2.4 - Transformación de variables de tipo texto

```
# Strings dinámicos con str_glue()
```

```
str_glue("{nrow(sg)} casos de SG com início dos sintomas \n até  
{format(max(sg$datainiciosintomas), '%d %b %Y')}").")
```

```
147990 casos de SG com início dos sintomas
```

```
até 16 jun 2022.
```

Es importante decir que trabajar con el manejo de textos a través de expresiones regulares puede resultar un poco desafiante. Al mismo tiempo, también es importante señalar que usted no tiene que memorizar las expresiones, ya que siempre podrá consultarlas al programar. Por ejemplo, acceda a *cheat sheet* (hoja de consultas) del paquete `stringr` y explore más funciones y expresiones regulares que pueden ser útiles para su rutina: <https://github.com/rstudio/cheatsheets/blob/main/strings.pdf>. Usted también puede encontrar ejemplos en la documentación de REGEX (escriba y ejecute `?regex` en el panel *console*).

```
> 3
```

```
> Separar y unir variables
```

Usted también puede querer separar una variable texto en diferentes columnas en su *data.frame* con base en un estándar. Por ejemplo, en el siguiente código, separaremos los diferentes síntomas registrados como texto en una sola variable de la base `sg` en

nueve columnas diferentes. Para ello utilizaremos la función `separate()` del paquete `tidyr`, que está incluido en `tidyverse`. Informamos al argumento `sep` = que el estándar es una coma, por lo que con cada coma el texto se desplazará a una nueva columna. Informamos al argumento `into` = un vector con el nombre de las nueve nuevas columnas.

Luego ejecutamos la función `head()` con algunas columnas seleccionadas para observar la transformación. Observe que las columnas se completaron en el orden en que aparecieron los síntomas entre las comas para cada registro, de acuerdo con el estándar informado e independientemente del síntoma. También tenga en cuenta que no todos los casos tienen registrados los nueve síntomas. Cuando esto sucede, las variables se registran automáticamente como NA, como en el caso del tercer síntoma (síntoma 3) del cuarto registro. Por defecto, la función `separate()` excluye la variable que fue “separada”. Usted puede evitar esta configuración informando el argumento `remove` = FALSE.

3. Separar y unir variables

Separar textos, con base en un estándar

```
sg <- sg %>%
  separate(SINTOMAS,
           sep = ",",
           into = c("sintoma1", "sintoma2", "sintoma3", "sintoma4",
                   "sintoma5", "sintoma6", "sintoma7", "sintoma8",
                   "sintoma9"))
```

Primeros registros (head) para comprobar transformación

```
sg %>%
  select(sintomas, sintoma1, sintoma3) %>%
  head()
```

	sintomas	sintoma1	sintoma3
1	CORIZA, DOR DE CABEÇA, TOSSE, DOR DE GARGANTA	CORIZA	TOSSE
2	DOR DE CABEÇA, DOR DE GARGANTA	DOR DE CABEÇA	<NA>
3	CORIZA, TOSSE, DOR DE GARGANTA	CORIZA	DOR DE GARGANTA
4	FEBRE, OUTROS	FEBRE	<NA>
5	CORIZA, DOR DE CABEÇA, TOSSE, FEBRE, DISPNEIA, OUTROS	CORIZA	TOSSE
6	DOR DE CABEÇA, FEBRE, DISPNEIA, DISTÚRBIOS GUSTATIVOS	DOR DE CABEÇA	DISPNEIA

Ahora usaremos la función `unite()` del paquete `tidyr` para hacer lo contrario. Nuestra intención será unir textos separados en diferentes columnas en una sola variable. Informamos al argumento `col` = el nombre de la columna que se creará e informamos un vector con el nombre de las columnas que se unirán. Informamos al argumento `sep` = qué separador se colocará entre los textos que se “pegarán”. El argumento `remove` = `TRUE` informa que se deben eliminar las columnas que fueron “pegadas”, dejando solo la nueva columna. Y el argumento `na.rm` = `T` ignora los valores NA. Si no informamos nada, por defecto los textos se unirán con `elsep` = `"_"` (*underline*).

```
# 3. Separar y unir variables ####  
# Unir textos en una sola columna  
sg_exemplo <- sg_exemplo %>%  
  unite(  
    col = "sintomas",  
    c("sintoma1", "sintoma2", "sintoma3",  
      "sintoma4", "sintoma5", "sintoma6",  
      "sintoma7", "sintoma8", "sintoma9"),  
    sep = ", ",  
    remove = TRUE,  
    na.rm = T  
  )  
  
# view() para comprobar transformación  
sg %>%  
  select(sintomas, SINTOMAS) %>%  
  view()  
  
class(sg$SINTOMAS)  
[1] "character"
```

En el código anterior, observe que la variable `SINTOMAS`, que fue recreada con la función `unite()`, asumió automáticamente la clase `character`. Cabe señalar que la función `unite()` también es capaz de recibir variables de otras clases para unir las. A su

vez, para separar variables con la función `separate()`, puede ser necesario convertir primero la variable en `character` y luego realizar la separación. También señalamos que las funciones `separate()` y `unite()` crearon variables independientemente de la función `mutate()`, por lo que tienen una sección destacada para ellas en esta clase.

> 4

> (Re) categorizar variables: `case_when()`

Ahora que usted conoce transformaciones generales para diferentes clases de variables, pasaremos al manejo de los datos y comenzaremos a crear variables con declaraciones lógicas. Tales transformaciones son habituales en la preparación de datos epidemiológicos para el análisis de la situación de salud. Por ejemplo, podemos pensar en la construcción de la variable grupo de edad para hacer una pirámide de edad, que se realiza con base en la edad. O una variable de confirmación de un caso notificado, que se construiría con base en las variables de resultado en laboratorio y presencia de signos y síntomas.

Con el comando `dp1yr::case_when()` dentro de la función `mutate()`, usted podrá realizar la mayoría de estas transformaciones, desde recategorizaciones simples hasta complejas. La configuración de la función `case_when()` tiene dos tramos separados por una "tilde" `~`. Los criterios lógicos están a la izquierda, los valores asignados a cada criterio lógico están a la derecha y cada par de criterios / valor asignado están separados por comas. Finalmente, informamos al argumento `TRUE` qué se debe hacer si no se cumple ninguno de los criterios declarados.

Por ejemplo, en el siguiente código, vemos que la variable `UTI` está codificada en la base original. Luego creamos la variable `uti` (en minúsculas) y asignamos la etiqueta correspondiente a cada código numérico, según el diccionario de datos del sistema SIVEP-Gripe, que es la fuente de los datos. En la nueva variable, las hospitalizaciones se dividirán en las categorías "Unidade de terapia intensiva", "Outras internações" e "Ignorado". Tenga en cuenta que la categoría "Ignorado", en la nueva variable `uti`, abarca tanto los registros con el código 9 ("Ignorado") como los registros sin contenido (NA) de la variable `UTI` (original).

```
# 4. (Re) categorizar variables: case_when() ####
# Frecuencia de la variable UTI
srag %>%
  count(UTI)
  UTI      n
1      1 10585
2      2 14749
3      9  1825
4     NA 10911

# Crear variable uti con labels
srag <- srag %>%
  mutate(uti = case_when(UTI == 1 ~ "Unidade de terapia intensiva",
                          UTI == 2 ~ "Outras internações",
                          TRUE    ~ "Ignorado"))

# Frecuencia de la nueva variable uti
srag %>%
  count(uti)
          uti      n
1          Ignorado 12736
2      Outras internações 14749
3  Unidade de terapia intensiva 10585
```

Vea el código de la función `case_when()` en la construcción de la variable `uti`, arriba. A la izquierda de la tilde (`~`), usamos el operador igual (`==`) para establecer los criterios lógicos, que son los códigos de la variable `UTI` original. No usamos comillas, porque la variable `UTI` es numérica. A la derecha de la tilde (`~`), usamos comillas para asignar diferentes categorías a la nueva variable `uti`.

Para ejercitar el uso de la función `case_when()`, haremos una transformación con más detalle. En el siguiente código, observamos las diferentes categorías de la variable `CLASSIFICACAOFINAL` en la base original de SG, que ya está etiquetada (*label*) en lugar de codificada. Nuestra intención será crear dos variables a partir de ella: `classificacaofinal` para registrar si el caso fue “Confirmado”, “Descartado” o si aún se desconoce la clasificación final (“ignorado”) y `criterio_class` para registrar

si el caso fue confirmado por criterio "Clínico Epidemiológico", "Clínico Imagem", "Laboratorial" o "Clínico".

4. (Re) categorizar variables: case_when()

Frecuencia de la variable CLASSIFICACAOFINAL

sg %>%

count(CLASSIFICACAOFINAL)

	CLASSIFICACAOFINAL	n
1		122456
2	Confirmado Clínico Epidemiológico	1756
3	Confirmado Clínico Imagem	10
4	Confirmado en Laboratorio	9674
5	Confirmado por Critério Clínico	1109
6	Descartado	12225
7	Síndrome Gripal Não Especificada	760

¡No se asuste por el tamaño del código a continuación! Observemos la secuencia de líneas y verá que no son más que instrucciones que usted probablemente ya está acostumbrado(a) a hacer. Sobrescribimos el objeto `sg` y después del pipe (`%>%`) incluimos la función `mutate()` para crear la variable `classificacaofinal`. Asignamos a ella la función `case_when()` para instruir las condiciones que se deben considerar al crear las categorías de la nueva variable.

Para la categoría "Confirmado", usamos el operador `o` (`|`) que aprendimos en la clase 4 (módulo 1) y consideramos cuatro respuestas diferentes de la variable original. En otras palabras, si el contenido de la variable `CLASSIFICACAOFINAL` es "Confirmado Clínico Epidemiológico" o "Confirmado Clínico Imagem" o "Confirmado Laboratorial" o "Confirmado por Critério Clínico", la nueva variable `classificacaofinal` recibirá el valor "Confirmado".

Por otro lado, si el contenido de la variable original `CLASSIFICACAOFINAL` es "Descartado", se mantendrá exactamente el valor de la variable original. En esta configuración, observe que después de la tilde ("`~`") escribimos el nombre de la variable original en lugar de "Descartado", que llevaría al mismo resultado. Podemos leer esta

línea como: si el contenido de la variable original CLASSIFICACAOFINAL es "Descartado", mantenga el mismo contenido en la nueva variable classificacaofinal. Por fin, TRUE ~ "Ignorado" significa que todos los demás casos deben recibir el valor "ignorado".

4. (Re) categorizar variables: case_when()

Crear variable classificacaofinal y criterio_class

```
sg <- sg %>%
  mutate(classificacaofinal =
    case_when(CLASSIFICACAOFINAL == "Confirmado Clínico Epidemiológico" |
              CLASSIFICACAOFINAL == "Confirmado Clínico Imagem" |
              CLASSIFICACAOFINAL == "Confirmado Laboratorial"|
              CLASSIFICACAOFINAL == "Confirmado por Critério Clínico"
              ~ "Confirmado",
              CLASSIFICACAOFINAL == "Descartado" ~ CLASSIFICACAOFINAL,
              TRUE ~ "Ignorado")) %>%
  mutate(criterio_class =
    case_when(CLASSIFICACAOFINAL == "Confirmado Clínico Epidemiológico"
              ~ "Clínico Epidemiológico",
              CLASSIFICACAOFINAL == "Confirmado Clínico Imagem"
              ~ "Clínico Imagem",
              CLASSIFICACAOFINAL == "Confirmado Laboratorial"
              ~ "Laboratorial",
              CLASSIFICACAOFINAL == "Confirmado por Critério Clínico"
              ~ "Clínico",
              TRUE ~ ""))
```

Al final de la transformación mencionada anteriormente, incluimos un *pipe* y uno nuevo `mutate()` para crear la variable `criterio_class`. Para ella, recategorizamos los valores de los casos confirmados en la variable original CLASSIFICACAOFINAL con la intención de especificar el criterio de confirmación del caso. En esta variable dejamos en blanco el contenido de todos los registros que no fueron confirmados en la variable original CLASSIFICACAOFINAL (TRUE ~ ""). Si usted recupera la frecuencia de la variable original, podrá ver que este grupo incluirá los registros "Descartado",

“Síndrome Gripal No Especificada” y también las que no tenían contenido en la variable original.

El código anterior abarca muchos detalles y puede parecer un poco desafiante para el alumno principiante en R. Es importante estar atento(a) a comas, comillas, paréntesis, argumentos, reglas de conjuntos en matemáticas y, principalmente, ¡a los criterios epidemiológicos! De hecho, debemos tener cuidado de no cometer errores al configurar los argumentos lógicos de la función `case_when()`, lo que puede generar conteos incorrectos. Pero, con la práctica, el uso de la función seguramente se convertirá en un hábito para usted en la etapa de limpieza y tratamiento de datos.

Para apoyar este proceso, le sugerimos que siga los consejos para organizar su código, presentados en la clase 5 (módulo 1). Por ejemplo, si compara el código de creación de la variable `uti`, al principio de la sección, con el código de creación de las variables `classificacaofinal` y `criterio_class`, notará que mantuvimos una condición por línea en el primer ejemplo, mientras que en los últimos usamos más líneas.

Esta es una estrategia para mejorar la legibilidad del código sin tener que usar la barra de desplazamiento para ver el código completo en el panel editor. Por ejemplo, podríamos escribir `CLASSIFICACAOFINAL == "Confirmado Clínico Epidemiológico" ~ "Clínico Epidemiológico"` en la misma línea, o incluso toda la función `mutate()`, pero quizás no sea la forma más legible para documentar la transformación. En las transformaciones anteriores, también resaltamos el hecho de que la función `case_when()` comprende diferentes clases de variables, operadores y funciones que vimos anteriormente.

En el siguiente código, usted puede ver las frecuencias de las nuevas variables creadas y verificar si obtuvo los conteos que esperaba.

```
# 4. (Re) categorizar variables: case_when() ####

# Frecuencia de las nuevas variables classificacaofinal y criterio_class
sg %>%
  count(classificacaofinal)
  classificacaofinal      n
1      Confirmado 12549
2      Descartado 12225
3      Ignorado 123216
```

```
sg %>%
  count(criterio_class)

  criterio_class      n
1                135441
2             Clínico   1109
3 Clínico Epidemiológico 1756
4             Clínico Imagem    10
5             Laboratorial  9674
```

En transformaciones más simples, con solo dos condiciones, usted puede usar la función `if_else()` del paquete `dplyr` (`tidyverse`), que tiene una sintaxis resumida para obtener el mismo resultado de la función `case_when()` sin tener que informar el argumento `TRUE` =. En el siguiente código, creamos la variable `tos` dicotómica (con solo dos respuestas) usando la función `mutate()` y le asignamos la función `if_else()`.

La función `if_else()` sólo tendrá tres argumentos. El primero es el criterio que se considerará. En nuestro ejemplo, usamos la función `str_detect()` para saber en qué registros identificamos el término “TOSSE” en la variable `síntomas`. El segundo y tercer registros se refieren a los valores que se incluirán en la nueva variable `tosse` si se incluye el criterio o no, respectivamente. Podemos leer el código a continuación de la siguiente manera: si se detecta el término “TOSSE” en la variable `síntomas`, se asignará el valor “Sim” a la nueva variable `tosse`, de lo contrario, “Não”. Luego observe la frecuencia de la variable de `tos` creada.

```
# 4. (Re) categorizar variables: case_when() ####

# ifelse() para transformaciones con dos condiciones
sg <- sg %>%
  mutate(tosse = ifelse(str_detect(sintomas, "TOSSE"),
                        "Sim", "Não"))

# Frecuencia de la nueva variable tos
sg %>%
  count(tosse)

tosse      n
```

```
1 Não 49606
2 Sim 98384
```

```
> 5
```

```
> Filtrar líneas: filter()
```

La última transformación de datos que haremos en esta clase será filtrar líneas según algún criterio. Esta transformación se utilizará especialmente cuando nuestra intención sea aplicar un criterio para realizar alguna acción con los datos. Por ejemplo, en el siguiente código hicimos el conteo del criterio de clasificación de los casos confirmados de SG. Para ello utilizamos la función `filter()` del paquete `dplyr`, que se incluye en el `tidyverse`. En la función `filter()`, informamos el criterio `classificacaofinal == "Confirmado"` de modo que antes de contar el criterio de clasificación, la base `sg` se filtre solo para los casos confirmados.

```
# 5. Filtrar líneas: filter() ####
# Filtrar casos confirmados
sg %>%
  filter(classificacaofinal == "Confirmado") %>%
  count(criterio_class)

  criterio_class      n
1             Clínico 1109
2 Clínico Epidemiológico 1756
3             Clínico Imagem    10
4             Laboratorial 9674
```

Es muy común que los usuarios principiantes del lenguaje R utilicen los términos “seleccionar” y “filtrar” indistintamente, para referirse a la elección de las columnas o líneas que estarán en la base de datos que se analizará. Tenga cuidado de no confundirse, porque en R usamos las funciones `select()` para seleccionar variables (columnas) y la función `filter()` para filtrar registros (líneas).

Vea un último ejemplo en el código a continuación. Hicimos un conteo del número de niños y adolescentes que fueron notificados con SRAG. Note que ahora usamos una variable `integer` (`edad`) y el operador `<` (menor) para filtrar los casos notificados menores de 18 años, ya que la variable `filter()` comprende diferentes clases de variables. Al ejecutar el código, encontramos que 2.800 de los 38.070 casos de SRAG ocurrieron en niños y adolescentes.

```
# 5. Filtrar líneas: filter() ####  
# Filtrar niños y adolescentes  
srag %>%  
  filter(idade < 18) %>%  
  count()  
  n  
1 2800
```

Ahora que hemos terminado, recuerde guardar su *script* para mantener sus anotaciones bien documentadas. Luego, pase a los ejercicios donde practicaremos las transformaciones que vimos en esta clase. ¡Hasta pronto!

> 6

> Referencias

Batra, Neale, et al. The Epidemiologist R Handbook. 2021. DOI:10.5281/zenodo.475264610.528 Disponible en: <https://epirhandbook.com/en/>.

Silva, Henrique Alvarenga da. Manual Básico da Linguagem R: Introdução à análise de dados com a linguagem R e o RStudio para área da saúde. [recurso electrónico] / Henrique Alvarenga da Silva - 1ª edición – São João del Rei: Editora do Autor, 2018. ISBN: 978-65-900132-0-0 (e-book).

Wickham, H., & Grolemund, G. R for data science: import, tidy, transform, visualize, and model data. O'Reilly Media, Inc. 2016. ISBN: 9781491910399. Disponible en: <https://r4ds.had.co.nz/>