Lesson Notes | Working with Strings in R

Introduction
Learning Objectives
Packages
Defining Strings
String Formatting in R with {stringr}
    Changing Case
    Handling Whitespace
    Text Padding
    Text Wrapping
    Applying String Formatting to a Dataset
Splitting Strings with `str_split()` and `separate()`
    Using `str_split()`
    Using `separate()`
    Separating Special Characters
Combining Strings with `paste()`
Subsetting strings with `str_sub`
Wrap up
Answer Key

Introduction

Proficiency in string manipulation is a vital skill for data scientists. Tasks like cleaning messy data and formatting outputs rely heavily on the ability to parse, combine, and modify character strings. This lesson focuses on techniques for working with strings in R, utilizing functions from the {stringr} package in the tidyverse. Let's dive in!

Learning Objectives

- Understand the concept of strings and rules for defining them in R
- Use escapes to include special characters like quotes within strings
- Employ {stringr} functions to format strings:
    - Change case with `str_to_lower()`, `str_to_upper()`, `str_to_title()`
    - Trim whitespace with `str_trim()` and `str_squish()`
    - Pad strings to equal width with `str_pad()`
    - Wrap text to a certain width using `str_wrap()`
- Split strings into parts using `str_split()` and `separate()`
- Combine strings together with `paste()` and `paste0()`
- Extract substrings from strings using `str_sub()`

Packages

```
# Loading required packages
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse, here, janitor)
```

Defining Strings

There are fundamental rules for defining character strings in R.

Strings can be enclosed in either single or double quotes. However, the type of quotation mark used at the start must match the one used at the end. For example:

```
string_1 <- "Hello" # Using double quotes
string_2 <- 'Hello' # Using single quotes
```

You cannot normally include double quotes inside a string that starts and ends with double quotes. The same applies to single quotes inside a string that starts and ends with single quotes. For example:

```
will_not_work <- "Double quotes " inside double quotes"
will_not_work <- 'Single quotes ' inside double quotes'
```

But you can include single quotes inside a string that starts and ends with double quotes, and vice versa:

```
single_inside_double <- "Single quotes ' inside double quotes"
```

Alternatively, you can use the escape character \ to include a literal single or double quote inside a string:

```
single_quote <- 'Single quotes \' inside double quotes'
double_quote <- "Double quotes \" inside double quotes"
```

To display these strings as they would appear in output, such as on a plot, use `cat()`:

```
cat('Single quotes \' inside double quotes')
```

```
## Single quotes ' inside double quotes
```

```
cat("Double quotes \" inside double quotes")
```

```
## Double quotes " inside double quotes
```

`cat()` prints its arguments without additional formatting.

SIDE NOTE

Since \ is the escape character, you must use \\ to include a literal backslash in a string:

```
backslash <- "This is a backslash: \\"
cat(backslash)
```

```
## This is a backslash: \
```

PRACTICE (in RMD)

Q: Error Spotting in String Definitions

Below are attempts to define character strings in R, with two out of five lines containing an error. Identify and correct these errors.

```
ex_a <- 'She said, "Hello!" to him.'
ex_b <- "She said \"Let's go to the moon\""
ex_c <- "They've been "best friends" for years."
ex_d <- 'Jane\\'s diary'
ex_e <- "It's a sunny day!
```

String Formatting in R with {stringr}

The {stringr} package in R provides useful functions for formatting strings for analysis and visualization. This includes case changes, whitespace handling, length standardization, and text wrapping.

Changing Case

Converting case is often needed to standardize strings or prepare them for display. The {stringr} package provides several case-changing functions:

- `str_to_upper()` converts strings to uppercase.

```r
str_to_upper("hello world")
```

```
## [1] "HELLO WORLD"
```

- `str_to_lower()` converts strings to lowercase.

```r
str_to_lower("Goodbye")
```

```
## [1] "goodbye"
```

- `str_to_title()` capitalizes the first letter of each word. Ideal for titling names, subjects, etc.

```r
str_to_title("string manipulation")
```

```
## [1] "String Manipulation"
```

Handling Whitespace

Managing whitespace makes strings neat and uniform. The {stringr} package provides two main functions for this:

- `str_trim()` removes whitespace at the start and end.

```r
str_trim(" trimmed ")
```

```
## [1] "trimmed"
```

- `str_squish()` removes whitespace at the start and end, *and* reduces multiple internal spaces to one.

```
        str_squish("   too   much    space   ")
```

```
## [1] "too much space"
```

```
        # notice the difference with str_trim
        str_trim("   too   much    space   ")
```

```
## [1] "too   much    space"
```

Text Padding

`str_pad()` pads strings to a fixed width. For example, we can pad the number 7 to force it to have 3 characters:

```
        str_pad("7", width = 3, pad = "0") # Pad left to length 3 with 0
```

```
## [1] "007"
```

The first argument is the string to pad. `width` sets the final string width and `pad` specifies the padding character.

`side` controls whether padding is added on the left or right. The `side` argument defaults to “left”, so padding will be added on the left side if not specified. Specifying `side = "right"` pads on the right side instead:

```
        str_pad("7", width = 4, side = "right", pad = "_") # Pad right to
length 4 with _
```

```
## [1] "7___"
```

Or we can pad on both sides:

```
        str_pad("7", width = 5, side = "both", pad = "_") # Pad both sides
to length 5 with _
```

```
## [1] "__7__"
```

Text Wrapping

Text wrapping helps fit strings into confined spaces like plot titles. The `str_wrap()` function wraps text to a set width.

For example, to wrap text at 10 characters we can write:

```
        example_string <- "String Manipulation with str_wrap can enhance
readability in plots."
        wrapped_to_10 <- str_wrap(example_string, width = 10)
        wrapped_to_10
```

```
## [1] "String\nManipulation\nwith\nstr_wrap\ncan\nenhance\nreadability\nin
plots."
```

The output may appear confusing. The `\n` indicates a line break, and to view the modified properly, we need to use the `cat()` function, which is a special version of `print()`:

```
        cat(wrapped_to_10)
```

```
## String
## Manipulation
## with
## str_wrap
## can
## enhance
## readability
## in plots.
```

Notice that the function maintains whole words, so it won't split longer words like "manipulation".

Setting the width to 1 essentially splits the string into individual words:

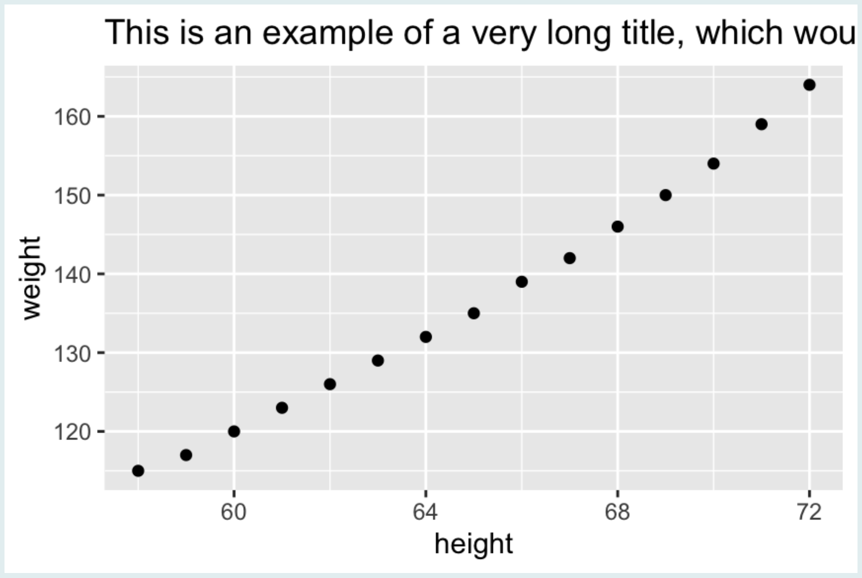
```
        cat(str_wrap(example_string, width = 1))
```

```
## String
## Manipulation
## with
## str_wrap
## can
## enhance
## readability
## in
## plots.
```

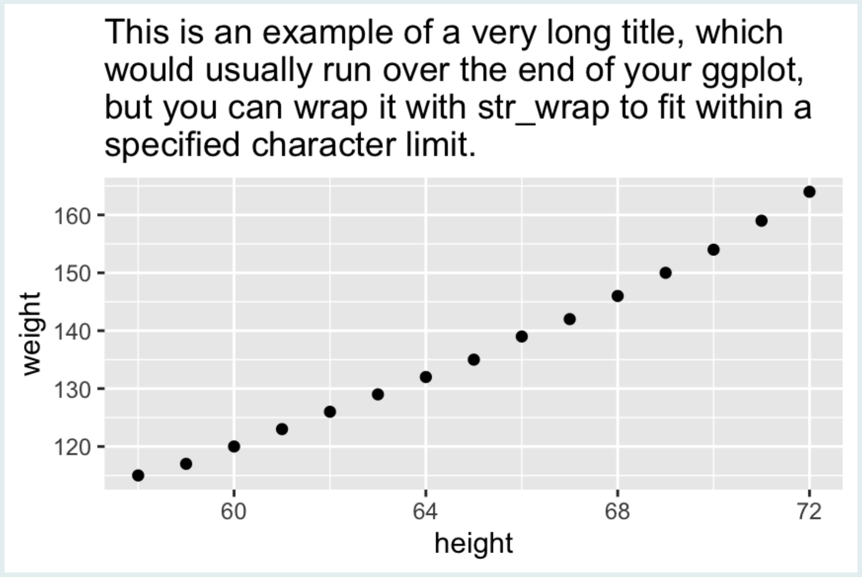
`str_wrap()` is particularly useful in plotting with ggplot2. For example, wrapping a long title to prevent it from spilling over the plot:

```
        long_title <- "This is an example of a very long title, which would
usually run over the end of your ggplot, but you can wrap it with str_wrap
to fit within a specified character limit."

        # Without wrapping
        ggplot(women, aes(height, weight)) +
          geom_point() +
          labs(title = long_title)
```



```
        # With wrapping at 80 characters
        ggplot(women, aes(height, weight)) +
          geom_point() +
          labs(title = str_wrap(long_title, width = 50))
```



So `str_wrap()` keeps titles neatly within the plot!

Q: Cleaning Patient Name Data

PRACTICE (in RMD)

A dataset contains patient names with inconsistent formatting and extra white spaces. Use the {stringr} package to standardize this information:

```
        patient_names <- c("  john doe", "ANNA SMITH   ",
"Emily Davis")
        # 1. Trim white spaces from each name.
        # 2. Convert each name to title case for consistency.
```

Q: Standardizing Drug Codes

PRACTICE (in RMD)

The following (fictional) drug codes are inconsistently formatted. Standardize them by padding with zeros to ensure all codes are 8 characters long:

```
        drug_codes <- c("12345", "678", "91011")
        # Pad each code with zeros on the left to a fixed
width of 8 characters.
```

Q: Wrapping Medical Instructions

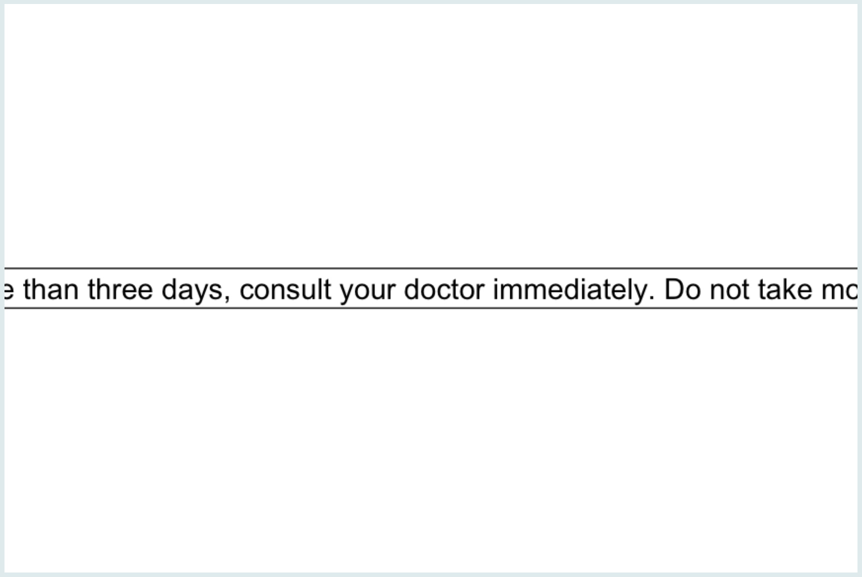
Use `str_wrap()` to format the following for better readability:

```
        instructions <- "Take two tablets daily after meals.
If symptoms persist for more than three days, consult your
doctor immediately. Do not take more than the recommended
dose. Keep out of reach of children."

        ggplot(data.frame(x = 1, y = 1), aes(x, y, label =
instructions)) +
          geom_label() +
          theme_void()
```

PRACTICE

(in RMD)



```
        # Now, wrap the instructions to a width of 50
characters then plot again.
```

Applying String Formatting to a Dataset

Now let's apply the {stringr} package's string formatting functions to clean and standardize a dataset. Our focus is on a dataset from a study on HIV care and treatment services in Zambézia Province, Mozambique, available here. The original dataset had various formatting inconsistencies, but we've added additional mistakes for educational purposes.

First, we load the dataset and examine specific variables for potential issues.

```
    # Load the dataset
    hiv_dat_messy_1 <-
openxlsx::read.xlsx(here("data/hiv_dat_messy_1.xlsx")) %>%
      as_tibble()

    # These four variables contain formatting inconsistencies:
    hiv_dat_messy_1 %>%
      select(district, health_unit, education, regimen)
```

```
## # A tibble: 1,413 × 4
##    district health_unit               education  regimen
##    <chr>    <chr>                     <chr>      <chr>
##  1 "Rural"  District Hospital Maganj… MISSING    AZT+3TC+NVP
##  2 "Rural"  District Hospital Maganj… secondary  TDF+3TC+EFV
##  3 "Urban"  24th  Of  July  Health  … MISSING    tdf+3tc+efv
##  4 "Urban"  24th  Of  July  Health  … MISSING    TDF+3TC+EFV
##  5 " Urban" 24th  Of  July  Health  … University tdf+3tc+efv
##  6 "Urban"  24th Of July Health Faci… Technical  AZT+3TC+NVP
##  7 "Rural"  District Hospital Maganj… Technical  TDF+3TC+EFV
##  8 "Urban"  24th Of July Health Faci… Technical  azt+3tc+nvp
##  9 "Urban"  24th Of July Health Faci… Technical  AZT+3TC+NVP
## 10 "Urban"  24th Of July Health Faci… Technical  TDF+3TC+EFV
## # ℹ 1,403 more rows
```

Using the `tabyl` function, we can identify and count unique values, revealing the inconsistencies:

```
    # Counting unique values
    hiv_dat_messy_1 %>% tabyl(health_unit)
```

```
##                              health_unit   n    percent
##         24th  Of  July  Health  Facility 239 0.16914367
##             24th Of July Health Facility 249 0.17622081
##   District  Hospital  Maganja  Da  Costa 342 0.24203822
##       District Hospital Maganja Da Costa 336 0.23779193
##                  Nante  Health  Facility 119 0.08421798
##                    Nante Health Facility 128 0.09058740
```

```
    hiv_dat_messy_1 %>% tabyl(education)
```

```
##   education   n     percent
##     MISSING 776 0.549186129
##        None 128 0.090587403
##     Primary 178 0.125973107
##   Secondary  82 0.058032555
##   Technical  17 0.012031139
##  University   4 0.002830856
##     primary 157 0.111111111
##   secondary  71 0.050247700
```

```
hiv_dat_messy_1 %>% tabyl(regimen)
```

```
##       regimen   n      percent valid_percent
##   AZT+3TC+EFV  24 0.0169851380  0.0179910045
##   AZT+3TC+NVP 229 0.1620665251  0.1716641679
##   D4T+3TC+ABC   1 0.0007077141  0.0007496252
##   D4T+3TC+EFV   2 0.0014154282  0.0014992504
##   D4T+3TC+NVP  16 0.0113234253  0.0119940030
##         OTHER   1 0.0007077141  0.0007496252
##   TDF+3TC+EFV 404 0.2859164897  0.3028485757
##   TDF+3TC+NVP   3 0.0021231423  0.0022488756
##   azt+3tc+efv  16 0.0113234253  0.0119940030
##   azt+3tc+nvp 231 0.1634819533  0.1731634183
##   d4t+3tc+efv   9 0.0063694268  0.0067466267
##   d4t+3tc+nvp  18 0.0127388535  0.0134932534
##   d4t+4tc+nvp   1 0.0007077141  0.0007496252
##  d4t6+3tc+nvp   2 0.0014154282  0.0014992504
##         other   2 0.0014154282  0.0014992504
##   tdf+3tc+efv 374 0.2646850672  0.2803598201
##   tdf+3tc+nvp   1 0.0007077141  0.0007496252
##          <NA>  79 0.0559094126            NA
```

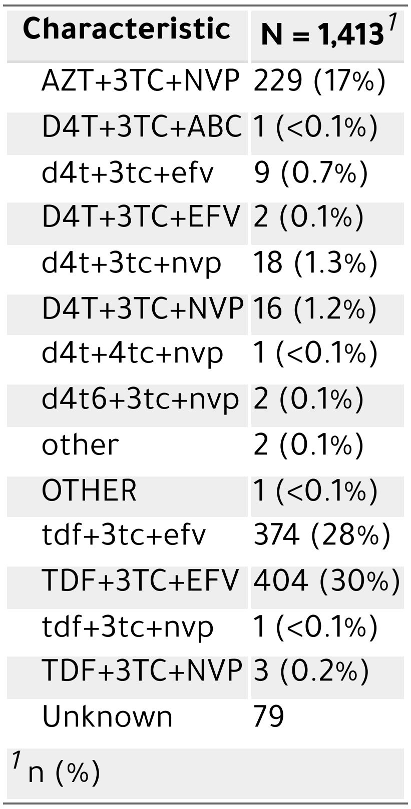
```
hiv_dat_messy_1 %>% tabyl(district)
```

```
##  district   n    percent
##     Rural 234 0.16560510
##     Urban 118 0.08351026
##     Rural 691 0.48903043
##     Urban 370 0.26185421
```

Another useful function for visualizing these issues is `tbl_summary` from the {gtsummary} package:

```
hiv_dat_messy_1 %>%
  select(district, health_unit, education, regimen) %>%
  tbl_summary()
```

Characteristic	**N = 1,413**[1]
district	
Rural	234 (17%)
Urban	118 (8.4%)
Rural	691 (49%)
Urban	370 (26%)
health_unit	
24th Of July Health Facility	239 (17%)
24th Of July Health Facility	249 (18%)
District Hospital Maganja Da Costa	342 (24%)
District Hospital Maganja Da Costa	336 (24%)
Nante Health Facility	119 (8.4%)
Nante Health Facility	128 (9.1%)
education	
MISSING	776 (55%)
None	128 (9.1%)
primary	157 (11%)
Primary	178 (13%)
secondary	71 (5.0%)
Secondary	82 (5.8%)
Technical	17 (1.2%)
University	4 (0.3%)
regimen	
azt+3tc+efv	16 (1.2%)
AZT+3TC+EFV	24 (1.8%)



Characteristic	N = 1,413[1]
AZT+3TC+NVP	229 (17%)
D4T+3TC+ABC	1 (<0.1%)
d4t+3tc+efv	9 (0.7%)
D4T+3TC+EFV	2 (0.1%)
d4t+3tc+nvp	18 (1.3%)
D4T+3TC+NVP	16 (1.2%)
d4t+4tc+nvp	1 (<0.1%)
d4t6+3tc+nvp	2 (0.1%)
other	2 (0.1%)
OTHER	1 (<0.1%)
tdf+3tc+efv	374 (28%)
TDF+3TC+EFV	404 (30%)
tdf+3tc+nvp	1 (<0.1%)
TDF+3TC+NVP	3 (0.2%)
Unknown	79

[1] n (%)

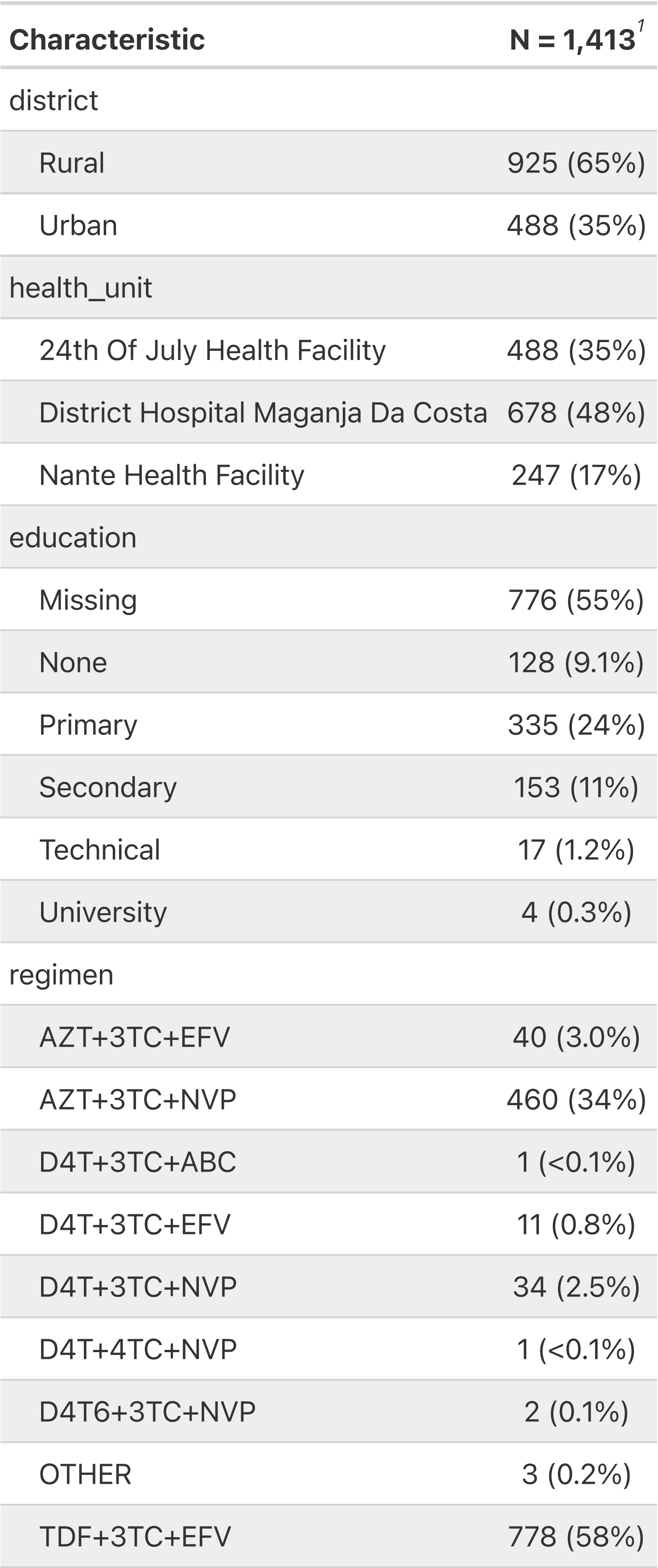
The output clearly shows inconsistencies in casing, spacing, and format, so we need to standardize them.

Next, we address these issues systematically:

```
        hiv_dat_clean_1 <- hiv_dat_messy_1 %>%
          mutate(
            district = str_to_title(str_trim(district)), # Standardizing district names
            health_unit = str_squish(health_unit),       # Removing extra spaces
            education = str_to_title(education),         # Standardizing education levels
            regimen = str_to_upper(regimen)              # Consistency in regimen column
          )
```

And we can verify the effectiveness of these changes by rerunning the `tbl_summary()` function:

```
        hiv_dat_clean_1 %>%
          select(district, health_unit, education, regimen) %>%
          tbl_summary()
```



Characteristic	N = 1,413[1]
district	
Rural	925 (65%)
Urban	488 (35%)
health_unit	
24th Of July Health Facility	488 (35%)
District Hospital Maganja Da Costa	678 (48%)
Nante Health Facility	247 (17%)
education	
Missing	776 (55%)
None	128 (9.1%)
Primary	335 (24%)
Secondary	153 (11%)
Technical	17 (1.2%)
University	4 (0.3%)
regimen	
AZT+3TC+EFV	40 (3.0%)
AZT+3TC+NVP	460 (34%)
D4T+3TC+ABC	1 (<0.1%)
D4T+3TC+EFV	11 (0.8%)
D4T+3TC+NVP	34 (2.5%)
D4T+4TC+NVP	1 (<0.1%)
D4T6+3TC+NVP	2 (0.1%)
OTHER	3 (0.2%)
TDF+3TC+EFV	778 (58%)

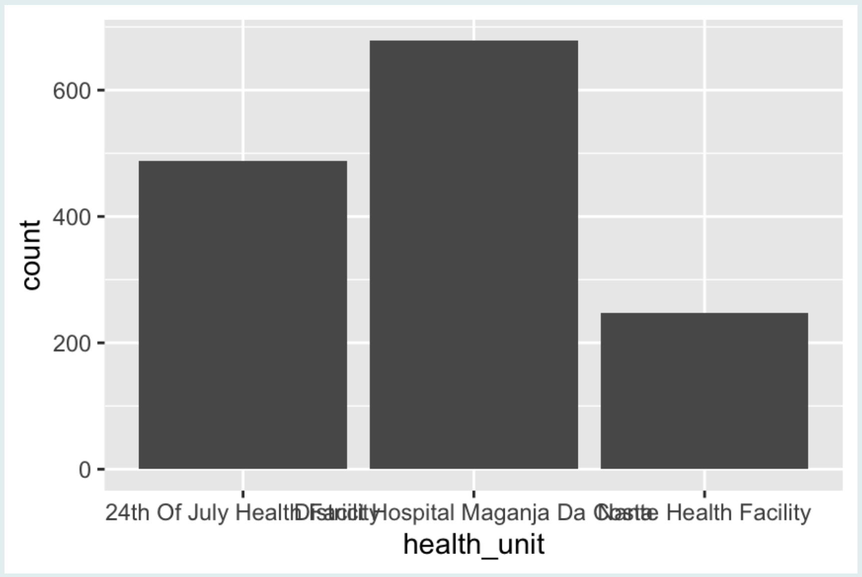
Characteristic	N = 1,413[1]
Unknown	79

[1] n (%)

Great!

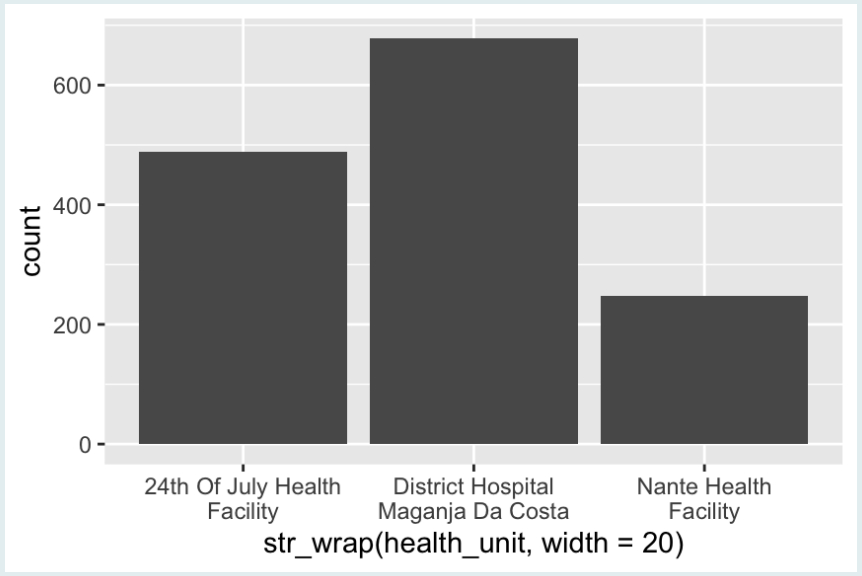
Finally, let's attempt to plot counts of the `health_unit` variable. For the plot style below, we encounter an issue with lengthy labels:

```
ggplot(hiv_dat_clean_1, aes(x = health_unit)) +
  geom_bar()
```



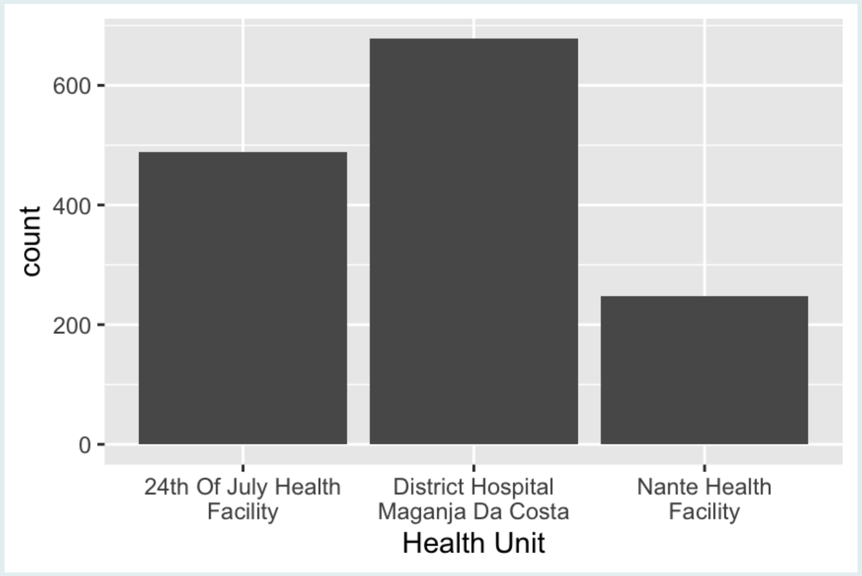
To resolve this, we can adjust the labels using `str_wrap()`:

```
hiv_dat_clean_1 %>%
  ggplot(aes(x = str_wrap(health_unit, width = 20))) +
  geom_bar()
```



Much cleaner, though we should probably fix the axis title:

```
hiv_dat_clean_1 %>%
  ggplot(aes(x = str_wrap(health_unit, width = 20))) +
  geom_bar() +
  labs(x = "Health Unit")
```



Now try your hand on similar cleaning operations in the practice questions below.

PRACTICE

(in RMD)

Q: Formatting a Tuberculosis Dataset

In this exercise, you will clean a dataset, `lima_messy`, originating from a tuberculosis (TB) treatment adherence study in Lima, Peru. More details about the study and the dataset are available here.

Begin by importing the dataset:

```
          lima_messy_1 <-
openxlsx::read.xlsx(here("data/lima_messy_1.xlsx")) %>%
            as_tibble()
          lima_messy_1
```

PRACTICE

(in RMD)

```
## # A tibble: 1,293 × 18
##    id      age          sex   marital_status
##    <chr>   <chr>        <chr> <chr>
##  1 pe-1008 38 and older M     Single
##  2 lm-1009 38 and older M     Married   /   cohabitating
##  3 pe-1010 27 to 37     m     Married   /  cohabitating
##  4 lm-1011 27 to 37     m     Married   /  cohabitating
##  5 pe-1012 38 and older m     Married  /  cohabitating
##  6 lm-1013 27 to 37     M     Single
##  7 pe-1014 27 To 37     m     Married   / cohabitating
##  8 lm-1015 22 To 26     m     Single
##  9 pe-1016 27 to 37     m     Single
## 10 lm-1017 22 to 26     m     Single
## # ℹ 1,283 more rows
## # ℹ 14 more variables: poverty_level <chr>, …
```

Your task is to clean the `marital_status`, `sex`, and `age` variables in `lima_messy`. Following the cleaning process, generate a summary table using the `tbl_summary()` function. Aim for your output to align with this structure:

Characteristic	**N = 1,293**
marital_status	
Divorced / Separated	93 (7.2%)
Married / Cohabitating	486 (38%)
Single	677 (52%)
Widowed	37 (2.9%)
sex	
F	503 (39%)
M	790 (61%)

Characteristic	N = 1,293
21 and younger	338 (26%)
22 to 26	345 (27%)
27 to 37	303 (23%)
38 and older	307 (24%)

PRACTICE (in RMD)

Implement the cleaning and summarize:

```
# Create a new object for cleaned data
lima_clean <- lima_messy %>%
  mutate(
    # Clean marital_status

    # Clean sex

    # Clean age

  )

# Check cleaning
lima_clean %>%
  select(marital_status, sex, age) %>%
  tbl_summary()
```

PRACTICE (in RMD)

Q: Wrapping Axis Labels in a Plot

Using the cleaned dataset `lima_clean` from the previous task, create a bar plot to display the count of participants by `marital_status`. Then wrap the axis labels on the x-axis to a maximum of 15 characters per line for readability.

```
# Create your bar plot with wrapped text here:
```

Splitting Strings with `str_split()` and `separate()`

Splitting strings is common task in data manipulation. The tidyverse offers efficient functions for this task, notably `stringr::str_split()` and `tidyr::separate()`.

Using str_split()

The str_split() function is useful for dividing strings into parts. For example:

```
example_string <- "split-this-string"
str_split(example_string, pattern = "-")
```

```
## [[1]]
## [1] "split"  "this"   "string"
```

This code splits example_string at each hyphen.

However, applying str_split() directly to a dataframe can be more complex.

Let's try it with the IRS dataset from Malawi as a case study. You should already be familiar with this dataset from a previous lesson. It is available here. For now, we'll focus on the start_date_long column:

```
irs <- read_csv(here("data/Illovo_data.csv"))
irs_dates_1 <- irs %>% select(village, start_date_long)
irs_dates_1
```

```
## # A tibble: 112 × 2
##    village            start_date_long
##    <chr>              <chr>
##  1 Mess               April 07 2014
##  2 Nkombedzi          April 22 2014
##  3 B Compound         May 13 2014
##  4 D Compound         May 13 2014
##  5 Post Office        May 13 2014
##  6 Mangulenje         May 15 2014
##  7 Mangulenje Senior May 27 2014
##  8 Old School         May 27 2014
##  9 Mwanza             May 28 2014
## 10 Alumenda           June 18 2014
## # ℹ 102 more rows
```

Suppose we want to split the start_date_long variable to extract the day, month, and year. We can write:

```
irs_dates_1 %>%
  mutate(start_date_parts = str_split(start_date_long, " "))
```

```
## # A tibble: 112 × 3
##    village            start_date_long start_date_parts
##    <chr>              <chr>            <list>
##  1 Mess               April 07 2014    <chr [3]>
##  2 Nkombedzi          April 22 2014    <chr [3]>
##  3 B Compound         May 13 2014      <chr [3]>
##  4 D Compound         May 13 2014      <chr [3]>
##  5 Post Office        May 13 2014      <chr [3]>
##  6 Mangulenje         May 15 2014      <chr [3]>
##  7 Mangulenje Senior May 27 2014      <chr [3]>
##  8 Old School         May 27 2014      <chr [3]>
##  9 Mwanza             May 28 2014      <chr [3]>
## 10 Alumenda           June 18 2014     <chr [3]>
## # ℹ 102 more rows
```

This results in a list column, which can be difficult to work with. To make it more readable, we can use `unnest_wider()`:

```
        irs_dates_1 %>%
          mutate(start_date_parts = str_split(start_date_long, " ")) %>%
          unnest_wider(start_date_parts, names_sep = "_")
```

```
## # A tibble: 112 × 5
##    village           start_date_long start_date_parts_1
##    <chr>             <chr>           <chr>
##  1 Mess              April 07 2014   April
##  2 Nkombedzi         April 22 2014   April
##  3 B Compound        May 13 2014     May
##  4 D Compound        May 13 2014     May
##  5 Post Office       May 13 2014     May
##  6 Mangulenje        May 15 2014     May
##  7 Mangulenje Senior May 27 2014     May
##  8 Old School        May 27 2014     May
##  9 Mwanza            May 28 2014     May
## 10 Alumenda          June 18 2014    June
## # ℹ 102 more rows
## # ℹ 2 more variables: start_date_parts_2 <chr>, …
```

It works! Our date parts are now split. However, this approach is quite cumbersome. A better solution for splitting components is the `separate()` function.

Using `separate()`

Let's try the same task using `separate()`:

```
        irs_dates_1 %>%
          separate(start_date_long, into = c("month", "day", "year"), sep =
" ")
```

```
##    <chr>              <chr> <chr> <chr>
##  1 Mess               April 07    2014
##  2 Nkombedzi          April 22    2014
##  3 B Compound         May   13    2014
##  4 D Compound         May   13    2014
##  5 Post Office        May   13    2014
##  6 Mangulenje         May   15    2014
##  7 Mangulenje Senior May   27    2014
##  8 Old School         May   27    2014
##  9 Mwanza             May   28    2014
## 10 Alumenda           June  18    2014
## # ℹ 102 more rows
```

Much more straightforward!

This function requires specifying:

- The column to be split.
- `into` - Names of the new columns.
- `sep` - The separator character.

To retain the original column, use `remove = FALSE`:

```
          irs_dates_1 %>%
            separate(start_date_long, into = c("month", "day", "year"), sep =
" ", remove = FALSE)
```

```
## # A tibble: 112 × 5
##    village           start_date_long month day   year
##    <chr>             <chr>           <chr> <chr> <chr>
##  1 Mess              April 07 2014   April 07    2014
##  2 Nkombedzi         April 22 2014   April 22    2014
##  3 B Compound        May 13 2014     May   13    2014
##  4 D Compound        May 13 2014     May   13    2014
##  5 Post Office       May 13 2014     May   13    2014
##  6 Mangulenje        May 15 2014     May   15    2014
##  7 Mangulenje Senior May 27 2014     May   27    2014
##  8 Old School        May 27 2014     May   27    2014
##  9 Mwanza            May 28 2014     May   28    2014
## 10 Alumenda          June 18 2014    June  18    2014
## # ℹ 102 more rows
```

SIDE NOTE

Alternatively, the `lubridate` package offers functions to extract date components:

```
          irs_dates_1 %>%
            mutate(start_date_long = mdy(start_date_long)) %>%
```

SIDE NOTE

```
                   month = month(start_date_long, label =
TRUE),               year = year(start_date_long))
```

```
## # A tibble: 112 × 5
##    village            start_date_long   day month  year
##    <chr>              <date>          <int> <ord> <dbl>
##  1 Mess               2014-04-07          7 Apr    2014
##  2 Nkombedzi          2014-04-22         22 Apr    2014
##  3 B Compound         2014-05-13         13 May    2014
##  4 D Compound         2014-05-13         13 May    2014
##  5 Post Office        2014-05-13         13 May    2014
##  6 Mangulenje         2014-05-15         15 May    2014
##  7 Mangulenje Senior  2014-05-27         27 May    2014
##  8 Old School         2014-05-27         27 May    2014
##  9 Mwanza             2014-05-28         28 May    2014
## 10 Alumenda           2014-06-18         18 Jun    2014
## # ℹ 102 more rows
```

When some rows lack all the necessary parts, `separate()` will issue a warning. Let's demonstrate this by artificially removing all instances of the word "April" from our dates:

```
          irs_dates_with_problem <-
            irs_dates_1 %>%
            mutate(start_date_missing = str_replace(start_date_long, "April ",
  ""))
          irs_dates_with_problem
```

```
## # A tibble: 112 × 3
##    village           start_date_long start_date_missing
##    <chr>             <chr>           <chr>
##  1 Mess              April 07 2014   07 2014
##  2 Nkombedzi         April 22 2014   22 2014
##  3 B Compound        May 13 2014     May 13 2014
##  4 D Compound        May 13 2014     May 13 2014
##  5 Post Office       May 13 2014     May 13 2014
##  6 Mangulenje        May 15 2014     May 15 2014
##  7 Mangulenje Senior May 27 2014     May 27 2014
##  8 Old School        May 27 2014     May 27 2014
##  9 Mwanza            May 28 2014     May 28 2014
## 10 Alumenda          June 18 2014    June 18 2014
## # ℹ 102 more rows
```

Now, let's try to split the date parts:

```
          irs_dates_with_problem %>%
            separate(start_date_missing, into = c("month", "day", "year"), sep
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 3 rows [1,
2, 12].
```

```
## # A tibble: 112 × 5
##    village           start_date_long month day   year 
##    <chr>             <chr>           <chr> <chr> <chr>
##  1 Mess              April 07 2014   07    2014  <NA> 
##  2 Nkombedzi        April 22 2014   22    2014  <NA> 
##  3 B Compound        May 13 2014     May   13    2014 
##  4 D Compound        May 13 2014     May   13    2014 
##  5 Post Office       May 13 2014     May   13    2014 
##  6 Mangulenje        May 15 2014     May   15    2014 
##  7 Mangulenje Senior May 27 2014     May   27    2014 
##  8 Old School        May 27 2014     May   27    2014 
##  9 Mwanza            May 28 2014     May   28    2014 
## 10 Alumenda          June 18 2014    June  18    2014 
## # ℹ 102 more rows
```

As you can see, rows missing parts will produce warnings. Handle such warnings carefully, as they can lead to inaccurate data. In this case, we now have the day and month information for those rows in the wrong columns.

PRACTICE

(in RMD)

Q: Splitting Age Range Strings

Consider the esoph_ca dataset, from the {medicaldata} package, which involves a case-control study of esophageal cancer in France.

```
medicaldata::esoph_ca %>% as_tibble()
```

```
## # A tibble: 88 × 5
##    agegp alcgp     tobgp    ncases ncontrols
##    <ord> <ord>     <ord>     <dbl>     <dbl>
##  1 25-34 0-39g/day 0-9g/day      0        40
##  2 25-34 0-39g/day 10-19         0        10
##  3 25-34 0-39g/day 20-29         0         6
##  4 25-34 0-39g/day 30+           0         5
##  5 25-34 40-79     0-9g/day      0        27
##  6 25-34 40-79     10-19         0         7
##  7 25-34 40-79     20-29         0         4
##  8 25-34 40-79     30+           0         7
##  9 25-34 80-119    0-9g/day      0         2
## 10 25-34 80-119    10-19          0          1
## # ℹ 78 more rows
```

PRACTICE

(in RMD)

Split the age ranges in the agegp column into two separate columns: agegp_lower and agegp_upper.

After using the separate() function, the "75+" age group will require special handling. Use readr::parse_number()or another method to convert the lower age limit ("75+") to a number.

```
medicaldata::esoph_ca %>%
  separate(________) %>%
  # convert 75+ to a number
  mutate(________)
```

Separating Special Characters

To use the separate() function on special characters like the period (.), we need to escape them with a double backslash (\\).

Consider the scenario where dates are formatted with periods:

```
        irs_with_period <- irs_dates_1 %>%
          mutate(start_date_long = format(lubridate::mdy(start_date_long),
"%d.%m.%Y"))
        irs_with_period
```

```
## # A tibble: 112 × 2
##    village            start_date_long
##    <chr>              <chr>
##  1 Mess               07.04.2014
##  2 Nkombedzi          22.04.2014
##  3 B Compound         13.05.2014
##  4 D Compound         13.05.2014
##  5 Post Office        13.05.2014
##  6 Mangulenje         15.05.2014
##  7 Mangulenje Senior  27.05.2014
##  8 Old School         27.05.2014
##  9 Mwanza             28.05.2014
## 10 Alumenda           18.06.2014
## # ℹ 102 more rows
```

Attempting to separate this date format directly with sep = "." will not work:

```
          irs_with_period %>%
            separate(start_date_long, into = c("day", "month", "year"), sep =
".")
```

```
## # A tibble: 112 × 4
##    village            day   month year
##    <chr>              <chr> <chr> <chr>
##  1 Mess               ""    ""    ""
##  2 Nkombedzi          ""    ""    ""
##  3 B Compound         ""    ""    ""
##  4 D Compound         ""    ""    ""
##  5 Post Office        ""    ""    ""
##  6 Mangulenje         ""    ""    ""
##  7 Mangulenje Senior  ""    ""    ""
##  8 Old School         ""    ""    ""
##  9 Mwanza             ""    ""    ""
## 10 Alumenda           ""    ""    ""
## # ℹ 102 more rows
```

This doesn't work as intended because, in regex (regular expressions), the period is a special character. We'll learn more about these in due course. The correct approach is to escape the period uses a double backslash (\):

```
          irs_with_period %>%
            separate(start_date_long, into = c("day", "month", "year"), sep =
"\\.")
```

```
## # A tibble: 112 × 4
##    village            day   month year
##    <chr>              <chr> <chr> <chr>
##  1 Mess               07    04    2014
##  2 Nkombedzi          22    04    2014
##  3 B Compound         13    05    2014
##  4 D Compound         13    05    2014
##  5 Post Office        13    05    2014
##  6 Mangulenje         15    05    2014
##  7 Mangulenje Senior  27    05    2014
##  8 Old School         27    05    2014
##  9 Mwanza             28    05    2014
## 10 Alumenda           18    06    2014
## # ℹ 102 more rows
```

Now, the function understands to split the string at each literal period.

Similarly, when using other special characters like +, ∗, or ?, we also need to precede them with a double backslash (\) in the `sep` argument.

SIDE NOTE

SIDE NOTE

What is a Special Character?

In regular expressions, which help find patterns in text, special characters have specific roles. For example, a period (.) is a wildcard that can represent any character. So, in a search, "do.t" could match "dolt," "dost," or "doct" Similarly, the plus sign (+) is used to indicate one or more occurrences of the preceding character. For example, "ho+se" would match "hose" or "hooose" but not "hse." When we need to use these characters in their ordinary roles, we use a double backslash (\\) before them, like "\\." or "\\+." More on these special characters will be covered in a future lesson.

PRACTICE

(in RMD)

Q: Separating Special Characters

Your next task involves the `hiv_dat_clean_1` dataset. Focus on the `regimen` column, which lists drug regimens separated by a + sign. Your goal is to split this column into three new columns: `drug_1`, `drug_2`, and `drug_3` using the `separate()` function. Pay close attention to how you handle the + separator. Here's the column:

```
hiv_dat_clean_1 %>%
  select(regimen)
```

```
## # A tibble: 1,413 × 1
##    regimen
##    <chr>
##  1 AZT+3TC+NVP
##  2 TDF+3TC+EFV
##  3 TDF+3TC+EFV
##  4 TDF+3TC+EFV
##  5 TDF+3TC+EFV
##  6 AZT+3TC+NVP
##  7 TDF+3TC+EFV
##  8 AZT+3TC+NVP
##  9 AZT+3TC+NVP
## 10 TDF+3TC+EFV
## # ℹ 1,403 more rows
```

Combining Strings with paste()

The `paste()` function in R concatenates or joins together character strings. This allows you to combine multiple strings into a single string.

To combine two simple strings:

```
        string1 <- "Hello"
        string2 <- "World"
        paste(string1, string2)
```

```
## [1] "Hello World"
```

The default separator is a space, so this returns "Hello World".

Let's demonstrate how to use this on a dataset, with the IRS date data. First, we'll separate the start date into individual columns:

```
        irs_dates_separated <- # store for later use
          irs_dates_1 %>%
          separate(start_date_long, into = c("month", "day", "year"), sep =
" ", remove = FALSE)
        irs_dates_separated
```

```
## # A tibble: 112 × 5
##    village            start_date_long month day   year 
##    <chr>              <chr>           <chr> <chr> <chr>
##  1 Mess               April 07 2014   April 07    2014 
##  2 Nkombedzi          April 22 2014   April 22    2014 
##  3 B Compound         May 13 2014     May   13    2014 
##  4 D Compound         May 13 2014     May   13    2014 
##  5 Post Office        May 13 2014     May   13    2014 
##  6 Mangulenje         May 15 2014     May   15    2014 
##  7 Mangulenje Senior May 27 2014     May   27    2014 
##  8 Old School         May 27 2014     May   27    2014 
##  9 Mwanza             May 28 2014     May   28    2014 
## 10 Alumenda           June 18 2014    June  18    2014 
## # ℹ 102 more rows
```

Then we can recombine day, month and year with `paste()`:

```
        irs_dates_separated %>%
          select(day, month, year) %>%
          mutate(start_date_long_2 = paste(day, month, year))
```

```
## # A tibble: 112 × 4
##    day   month year  start_date_long_2
##    <chr> <chr> <chr> <chr>
##  1 07    April 2014  07 April 2014
##  2 22    April 2014  22 April 2014
##  3 13    May   2014  13 May 2014
##  4 13    May   2014  13 May 2014
##  5 13    May   2014  13 May 2014
##  6 15    May   2014  15 May 2014
##  7 27    May   2014  27 May 2014
##  8 27    May   2014  27 May 2014
##  9 28    May   2014  28 May 2014
## 10 18    June  2014  18 June 2014
## # ℹ 102 more rows
```

The sep argument specifies the separator between elements. For a different separator, like a hyphen, we can write:

```
irs_dates_separated %>%
  mutate(start_date_long_2 = paste(day, month, year, sep = "-"))
```

```
## # A tibble: 112 × 6
##    village            start_date_long month day   year
##    <chr>              <chr>           <chr> <chr> <chr>
##  1 Mess               April 07 2014   April 07    2014
##  2 Nkombedzi          April 22 2014   April 22    2014
##  3 B Compound         May 13 2014     May   13    2014
##  4 D Compound         May 13 2014     May   13    2014
##  5 Post Office        May 13 2014     May   13    2014
##  6 Mangulenje         May 15 2014     May   15    2014
##  7 Mangulenje Senior  May 27 2014     May   27    2014
##  8 Old School         May 27 2014     May   27    2014
##  9 Mwanza             May 28 2014     May   28    2014
## 10 Alumenda           June 18 2014    June  18    2014
## # ℹ 102 more rows
## # ℹ 1 more variable: start_date_long_2 <chr>
```

To concatenate without spaces, we can set sep = "":

```
irs_dates_separated %>%
  select(day, month, year) %>%
  mutate(start_date_long_2 = paste(day, month, year, sep = ""))
```

```
## # A tibble: 112 × 4
##    day   month year  start_date_long_2
##    <chr> <chr> <chr> <chr>
##  1 07    April 2014  07April2014
##  2 22    April 2014  22April2014
##  3 13    May   2014  13May2014
##  4 13    May   2014  13May2014
##  5 13    May   2014  13May2014
##  6 15    May   2014  15May2014
##  7 27    May   2014  27May2014
##  8 27    May   2014  27May2014
##  9 28    May   2014  28May2014
## 10 18    June  2014  18June2014
## # ℹ 102 more rows
```

Or we can use the `paste0()` function, which is equivalent to `paste(..., sep = "")`:

```
irs_dates_separated %>%
  select(day, month, year) %>%
  mutate(start_date_long_2 = paste0(day, month, year))
```

```
## # A tibble: 112 × 4
##    day   month year  start_date_long_2
##    <chr> <chr> <chr> <chr>
##  1 07    April 2014  07April2014
##  2 22    April 2014  22April2014
##  3 13    May   2014  13May2014
##  4 13    May   2014  13May2014
##  5 13    May   2014  13May2014
##  6 15    May   2014  15May2014
##  7 27    May   2014  27May2014
##  8 27    May   2014  27May2014
##  9 28    May   2014  28May2014
## 10 18    June  2014  18June2014
## # ℹ 102 more rows
```

Let's try to combine `paste()` with some other string functions to solve a realistic data problem. Consider the ID column in the `hiv_dat_messy_1` dataset:

```
hiv_dat_messy_1 %>%
  select(patient_id)
```

```
## # A tibble: 1,413 × 1
##    patient_id
##    <chr>
##  1 pd-10037
##  2 pd-10537
##  3 pd-5489
##  4 id-5523
##  5 pd-4942
##  6 pd-4742
##  7 pd-10879
##  8 id-2885
##  9 pd-4861
## 10 pd-5180
## # ℹ 1,403 more rows
```

Imagine we wanted to standardize these IDs to have the same number of characters. This is often a requirement for IDs (think about phone numbers, for instance).

To implement this, we can use `separate()` to split the IDs into parts, then use `paste()` to recombine them into a standardized format.

```
        hiv_dat_messy_1 %>%
          select(patient_id) %>% # for visibility
          separate(patient_id, into = c("prefix", "patient_num"), sep = "-",
remove = F) %>%
          mutate(patient_num = str_pad(patient_num, width = 5, side =
"left", pad = "0")) %>%
          mutate(patient_id_padded = paste(prefix, patient_num, sep = "-"))
```

```
## # A tibble: 1,413 × 4
##    patient_id prefix patient_num patient_id_padded
##    <chr>      <chr>  <chr>       <chr>
##  1 pd-10037   pd     10037       pd-10037
##  2 pd-10537   pd     10537       pd-10537
##  3 pd-5489    pd     05489       pd-05489
##  4 id-5523    id     05523       id-05523
##  5 pd-4942    pd     04942       pd-04942
##  6 pd-4742    pd     04742       pd-04742
##  7 pd-10879   pd     10879       pd-10879
##  8 id-2885    id     02885       id-02885
##  9 pd-4861    pd     04861       pd-04861
## 10 pd-5180    pd     05180       pd-05180
## # ℹ 1,403 more rows
```

In this example, `patient_id` is split into a prefix and a number. The number is then padded with zeros to ensure consistent length, and finally, the two parts are concatenated back together using `paste()` with a hyphen as the separator. This process standardizes the format of patient IDs.

Great work!

PRACTICE

(in RMD)

Q: Standardizing IDs in the `lima_messy_1` Dataset

In the `lima_messy_1` dataset, the IDs are not zero-padded, making them hard to sort.

For example, the ID pe−998 is at the top of the list after sorting in descending order, which is not what we want.

```
lima_messy_1 %>%
  select(id) %>%
  arrange(desc(id)) # sort in descending order
(highest IDs should be at the top)
```

```
## # A tibble: 1,293 × 1
##    id
##    <chr>
##  1 pe−998
##  2 pe−996
##  3 pe−951
##  4 pe−900
##  5 pe−2347
##  6 pe−2337
##  7 pe−2335
##  8 pe−2333
##  9 pe−2331
## 10 pe−2329
## # ℹ 1,283 more rows
```

PRACTICE (in RMD)

Try to fix this using a similar procedure to the one used for `hiv_dat_messy_1`.

Your Task:

- Separate the ID into parts.
- Pad the numeric part for standardization.
- Recombine the parts using `paste()`.
- Resort the IDs in descending order. The highest ID should end in 2347

```
lima_messy_1 %>%
  ______________
```

PRACTICE (in RMD)

PRACTICE

(in RMD)

Q: Creating Summary Statements

Create a column containing summary statements combining `village`, `start_date_default`, and `coverage_p` from the `irs` dataset. The statement should describe the spray coverage for each village.

Desired Output: "For village X, the spray coverage was Y% on Z date."

Your Task: - Select the necessary columns from the `irs` dataset. - Use `paste()` to create the summary statement.

```
irs %>%
  select(village, start_date_default, coverage_p) %>%

  ______________________
```

REMINDER

As we go through this lesson, remember that RStudio's auto-complete can help you find functions in the stringr package.

Just type `str_` and a list of stringr functions will pop up. All stringr functions start with `str_`.

So instead of trying to memorize them all, you can use auto-complete as a reference when needed.

Subsetting strings with `str_sub`

`str_sub` allows you to extract parts of a string based on character positions. The basic syntax is `str_sub(string, start, end)`.

Example: Extracting the first 2 characters from patient IDs:

```
patient_ids <- c("ID12345-abc", "ID67890-def")
str_sub(patient_ids, 1, 2) # Returns "ID", "ID"
```

```
## [1] "ID" "ID"
```

Or the first 5:

```
        str_sub(patient_ids, 1, 5) # Returns "ID123", "ID678"
```

```
## [1] "ID123" "ID678"
```

Negative values count backward from the end of the string. This is useful for extracting suffixes.

For example, to get the last 4 characters of patient IDs.

```
        str_sub(patient_ids, -4, -1) # Returns "-abc", "-def"
```

```
## [1] "-abc" "-def"
```

Be sure to pause and understand what happened above.

When indices are outside the string length, `str_sub` handles it gracefully without errors:

```
        str_sub(patient_ids, 1, 30) # Safely returns the full string when
the range exceeds the string length
```

```
## [1] "ID12345-abc" "ID67890-def"
```

In a data frame, we can use `str_sub` within `mutate()`. For example, below we extract the year and month from the `start_date_default` column and create a new column called `year_month`:

```
        irs %>%
          select(start_date_default) %>%
          mutate(year_month = str_sub(start_date_default, start = 1, end =
7))
```

```
## # A tibble: 112 × 2
##    start_date_default year_month
##    <date>             <chr>
##  1 2014-04-07         2014-04
##  2 2014-04-22         2014-04
##  3 2014-05-13         2014-05
##  4 2014-05-13         2014-05
##  5 2014-05-13         2014-05
##  6 2014-05-15         2014-05
##  7 2014-05-27         2014-05
##   8 2014-05-27         2014-05
##   9 2014-05-28         2014-05
##  10 2014-06-18         2014-06
## # ℹ 102 more rows
```

PRACTICE

(in RMD)

Q: Extracting ID Substrings

Use `str_sub()` to isolate just the numeric part of the `patient_id` column in the `hiv_dat_messy_1` dataset.

```
hiv_dat_messy_1 %>%
  select(patient_id) %>%
  # your code here:
  ______________
```

Wrap up

Congratulations on reaching the end of this lesson! You've learned about strings in R and various functions to manipulate them effectively.

The table below gives a quick recap of the key functions we covered. Remember, you don't need to memorize all these functions. Knowing they exist and how to look them up (like using Google) is more than enough for practical applications.

Function	Description	Example	Example Output
`str_to_upper()`	Convert characters to uppercase	`str_to_upper("hiv")`	“HIV”
`str_to_lower()`	Convert characters to lowercase	`str_to_lower("HIV")`	“hiv”
`str_to_title()`	Convert first character of each word to uppercase	`str_to_title("hiv awareness")`	“Hiv Awareness”
`str_trim()`	Remove whitespace from start & end	`str_trim(" hiv ")`	“hiv”
`str_squish()`	Remove whitespace from start & end and reduce internal spaces	`str_squish(" hiv cases ")`	“hiv cases”
`str_pad()`	Pad a string to a fixed width	`str_pad("45", width = 5)`	“00045”
`str_wrap()`	Wrap a string to a given width (for formatting output)	`str_wrap("HIV awareness", width = 5)`	“HIV”

str_split()	Split elements of a character vector	str_split("Hello-World", "-")	c(“Hello”, “World”)
paste()	Concatenate vectors after converting to character	paste("Hello", "World")	“Hello World”
str_sub()	Extract and replace substrings from a character vector	str_sub("HelloWorld", 1, 4)	“Hell”

separate()	Separate a character column into multiple columns	separate(tibble(a = "Hello-World"), a,
		into = c("b", "c"),
		sep = "-")
			|b |c |
			|Hello
			|World |

Note that while these functions cover common tasks such as string standardization, splitting and joining strings, this introduction only scratches the surface of what's possible with the {stringr} package. If you work with a lot of raw text data, you may want to do further exploring on the stringr website.

Answer Key

Q: Error Spotting in String Definitions

1. **ex_a**: Correct.
2. **ex_b**: Correct.
3. **ex_c**: Error. Corrected version: `ex_c <- "They've been \"best friends\" for years."`
4. **ex_d**: Error. Corrected version: `ex_d <- 'Jane\'s diary'`
5. **ex_e**: Error. Close quote missing. Corrected version: `ex_e <- "It's a sunny day!"`

Q: Cleaning Patient Name Data

```
patient_names <- c("  john doe", "ANNA SMITH   ", "Emily Davis")

patient_names <- str_trim(patient_names) # Trim white spaces
patient_names <- str_to_title(patient_names) # Convert to title case
```

Q: Standardizing Drug Codes

```
        drug_codes <- c("12345", "678", "91011")

        # Pad each code with zeros on the left to a fixed width of 8
characters.
        drug_codes_padded <- str_pad(drug_codes, 8, pad = "0")
```

Q: Wrapping Medical Instructions

```
        instructions <- "Take two tablets daily after meals. If symptoms
persist for more than three days, consult your doctor immediately. Do not
take more than the recommended dose. Keep out of reach of children."

        # Wrap instructions
        wrapped_instructions <- str_wrap(instructions, width = 50)

        ggplot(data.frame(x = 1, y = 1), aes(x, y, label =
wrapped_instructions)) +
          geom_label() +
          theme_void()
```

Q: Formatting a Tuberculosis Dataset

The steps to clean the `lima_messy` dataset would involve:

```
        lima_clean <- lima_messy %>%
          mutate(
            marital_status = str_squish(str_to_title(marital_status)), #
Clean and standardize marital_status
            sex = str_squish(str_to_upper(sex)),                        #
Clean and standardize sex
            age = str_squish(str_to_lower(age))                         #
Clean and standardize age
          )


        lima_clean %>%
          select(marital_status, sex, age) %>%
          tbl_summary()
```

Then, use the `tbl_summary()` function to create the summary table.

Q: Wrapping Axis Labels in a Plot

```
    # Assuming lima_clean is already created and contains marital_status
    ggplot(lima_clean, aes(x = str_wrap(marital_status, width = 15))) +
      geom_bar() +
      labs(x = "Marital Status")
```

Q: Splitting Age Range Strings

```
    esoph_ca %>%
      select(agegp) %>% # for illustration
      separate(agegp, into = c("agegp_lower", "agegp_upper"), sep = "-")
%>%

      mutate(agegp_lower = readr::parse_number(agegp_lower))
```

Q: Creating Summary Statements

```
    irs %>%
      select(village, start_date_default, coverage_p) %>%
      mutate(summary_statement = paste0("For village ", village, ", the
spray coverage was ", coverage_p, "% on ", start_date_default))
```

Q: Extracting ID Substrings

```
    hiv_dat_messy_1 %>%
      select(patient_id) %>%
      mutate(numeric_part = str_sub(patient_id, 4))
```

Contributors

The following team members contributed to this lesson:

CAMILLE BEATRICE VALERA

Project Manager and Scientific Collaborator, The GRAPH Network

KENE DAVID NWOSU

Data analyst, the GRAPH Network

Passionate about world improvement