

R Packages : dplyer + tidyr

Statistics, Visualization and More Using “R”

Fairuz Maliha Easty

Laiba Gilliani

Leonhard Hecht

R Package: dplyr – Introduction/Overview:

- Packages (libraries) are one of the most important features of R
- One of these Packages is dplyr (spoken: d-p-l-y-r **or** “D-plier” (diːˌplaɪ.ər))
- dplyr is part of the collection tidyverse, a package for data science (as well as tidyr, ggplot2, purr, readr and some more)
- dplyr is powerful and a widely used package for data manipulation
- Installation: `install.packages(“dplyr”)` or `install.packages(“tidyverse”)` once
- Usage with `library(dplyr)` or `library(tidyverse)`
- The current version is 1.2.0 released on 2th of February 2026

R Package: dplyr – Key features:

- **filter()**: To select rows based on conditions.
- **slice()**: To select rows.
- **arrange()**: To sort data by one or more columns.
- **select()**: To choose or rename columns.
- **mutate()**: To create new columns or modify existing ones
- **rename()**: To rename columns.
- Studio cheat sheet: <https://rstudio.github.io/cheatsheets/data-transformation.pdf>
- **help(verb)** or **?verb**

R Package: dplyr – Advantages:

- High performance also with large datasets (optimized in C++)
- Consistent syntax (readable verbs)
- Clear structure
- Good cooperation with other tidyverse packages

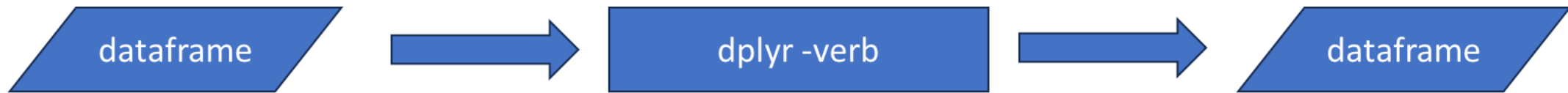
Specialty:

- Works with dataframes (input as well as output is a dataframe)
- Run in a pipe (next slide)

R Package: dplyr – Pipe:

Short form for pipeline:

- Founded for the Operating System Unix by Douglas Mcllroy 1972
- Pipes are a method to run several dplyr verbs in a chain



- In a pipe:



- The last verb in a pipe can also be a not dplyr command like head or so
- dplyr-verbs doesn't change the dataframe, but the result can be saved in a new dataframe

R Package: dplyr – Generic Syntax:

- "Subprograms" in dplyr are called verbs; the Data are subjects
- `verb(dataframe, options)`
- In pipe: `dataframe |> verb(options) |> verb(options) ...`
- Instead of `|>`, the previous version `%>%` is also sometimes used
- For the next Example we use dataframe 'airquality' which is part of the Basis-R package

```
> slice(airquality, 1:10)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
9     8      19 20.1   61     5   9
10   NA     194  8.6   69     5  10
```

R Package: dplyr – Example:

- We have the standard dataset `airquality` (New York, May-1-1973 to Sep-30-1973)
We are interested in renewable Energy with Solarpower and Windpower.
So we want to know, if there are dark-doldrum (not sunny and not windy) in that time.
- For that we want to have the wind not in mph but in km/h as column `Wind_km`.
(Hint: $\text{km/h} = \text{mph} * 1,60934$)
- `airquality |> mutate(Wind_km = Wind * 1.60934)` # new converted colomn Wind in km/h
- The Name of column `Solar.R` should be in the same format `Solar_lang`
- `airquality |> rename(Solar_lang=Solar.R)` # same format für Solar_unit

R Package: dplyr – Example assemble:

- Filter all, that are less sunny (lowest 10% <33) and less windy (lowest 10% <33.3)
- `airquality |> filter(Solar_lang < 33, Wind_km < 33.3)` # all rows with Solar_lang < 30 and Wind <33.3
- Only 10 results (lines)
- `airquality |> slice(1:10)` # result should max have 10 lines
- Sorted by Month and Day
- `arrange(Month, Day)` # Sorted by Month and than Day
- And only the interesting columns without Ozone and Temp in a dataframe
- `airquality |> select(-Ozone, -Temp)` # select all without column Ozone and Temp

R Package: dplyr – Exercise: (look at the snippet)

- You want to have:
- Only all days with **temperature > 56 and wind > 14**
- the **temperature not in Fahrenheit** but in °C and the wind additional not in mph but in km/h as column Wind_km.
(Hint: °C = ((°F-32) * 5) / 9 and km/h = mph * 1,60934)
- All lines ordered by **Wind_km in descending** order and **Temp in standard ascending** order
- **only Line 1 to 30**
- the name of column Solar.R should be Solar_Rate
- and the first 10 rows of all columns without Ozone in a dataframe called airq_new

Hint: If you get the message: „Error: object ‚Wind‘ not found“ you forgot to load the library dplyr or tidyverse, so another command called „filter“ was used!

R Package: dplyr – More verbs:

- **summarise()** (or `summarize()`): To summarize data (e.g., mean, sum, ...).
- **group_by()**: To group data for aggregation.
- **n()**: To count the rows in a grouped dataframe
- **count()**: To group and count rows
- **join()**: To combine tables
(e.g., `inner_join()`, `left_join()`, `right_join()`, `full_join()`).
- **across()**: To
- **filter_out()**: The **new complement** to `filter()` for **excluding** rows
(introduced in version 1.2.0).
- **and more**

R Package: dplyr – summarise() and grouped_by():

summarise(): identical with summarize():

- summarise() is used to sum up the values of a column by e.g. mean(), sd(), max() ...
- For example: `aq_meantemp <- airquality |> summarise(mean(Temp))`

grouped_by():

- You can not see the effect directly at the output

`airquality |> group_by(Month) # looks nearly like airquality`

- but with: (attention you need to add `na.rm=TRUE`, otherwise you get only NA.)

`airquality |> group_by(Month) |> summarise(mean(Ozone, na.rm=TRUE))`

```
> head((aq_mO_gM), 3)
# A tibble: 3 × 2
  Month `mean(Ozone, na.rm = TRUE)`
  <int> <dbl>
1     5 23.6
2     6 29.4
3     7 59.1
```

R Package: dplyr – n() and count():

n():

- To count the rows in a grouped dataframe you can use n() (nrow doesn't work with group_by())
- `airquality |> group_by(Month) |> summarise(n())`
Returns the number of rows per month

count():

- Count() is a combination of group_by and n()
- `airquality |> count(Month)`
Also Returns the number of rows per month

R Package: dplyr – Join-Versions:

To combine two Dataframes based on a key-column

```
df_output <- df_left |> join_function(df_right, by = "key-column")
```

- `df_left <- data_frame(nl = c("A","B","C"), nbl = c(1,2,3), n2l = c("a","b","c"))`
- `df_right <- data_frame(nr = c("A","C","D"), nbr = c(11,33,44), n2r = c("aa","cc","dd"))`

There are different kinds of join:

- **inner_join():** only keeps rows which are in both inputs
- `df_left |> inner_join(df_right, by=join_by("nl"=="nr"))`
- **left_join():** keeps all rows from the left df and adds suitable rows from the right
not suitable rows are filled with NA
- `df_left |> left_join(df_right, by=join_by("nl"=="nr"))`

R Package: dplyr – Join-Syntax:

- **right_join():** keeps all rows from the right df and adds suitable rows from the left
- `df_left |> right_join(df_right, by=join_by("nl"=="nr"))`
- **full_join():** keeps all rows of both inputs
- `df_left |> full_join(df_right, by=join_by("nl"=="nr"))`
- **semi_join():** keeps only rows of the left input, which have an accordance in the right
- `df_left |> semi_join(df_right, by=join_by("nl"=="nr"))`
- **anti_join():** keeps only rows of the left, which have no accordance in the right
- `df_left |> anti_join(df_right, by=join_by("nl"=="nr"))`

R Package: dplyr – Verbs with suffixes:

- All main verbs of dplyr have suffixes: (`_if`, `_all`, `_at`)
- `_if`: Executes the verb, if the condition is met
- `_all`: Executes the verb for every column
- `_at`: Executes the verb for the given columns

Example

- `airquality |> summarise_if(is.integer, max, na.rm=TRUE)`
Returns the Max-value of all integer-columns
- `airquality |> summarise_all(list(min,max),na.rm=TRUE)`
Returns a List of all Min and Max-values of all columns
- `airquality |> summarise_at(.vars = vars(Temp, Wind),.funs = min, na.rm=TRUE)`
Returns the minimum values of Temp and Wind

R Package: dplyr - further informations

This has only been a short overview.

There are a lot more options you can find by
`help(verb)` or `?verb`

or in the RStudio cheat sheet :

<https://rstudio.github.io/cheatsheets/data-transformation.pdf>

Tidyverse: Tidy

Tidyverse

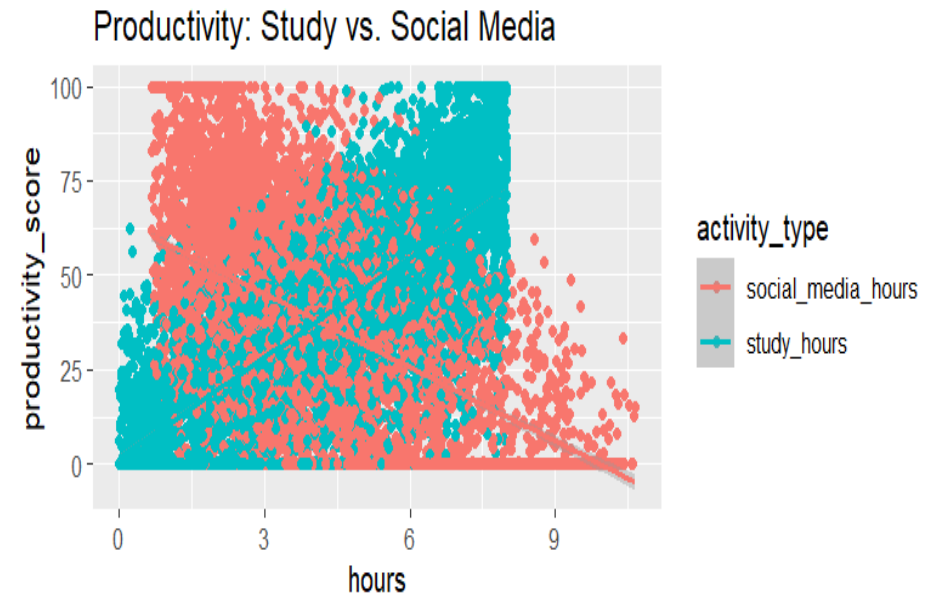
- A collection of the Packages
- Why is it so popular?
 - Readability
 - The "Tidy" Data format

Tidyverse

```
social_media_data %>%  
  # 1. CLEANING (tidyr)  
  # pivot the hours into one 'Activity' column  
  pivot_longer(  
    cols = c(social_media_hours, study_hours),  
    names_to = "activity_type",  
    values_to = "hours"  
  ) %>%  
  |  
  # 2. ANALYSIS (dplyr)  
  # focus_score higher than 7  
  filter(focus_score > 7) %>%  
  
  # 3. PLOTTING (ggplot2)  
  # Hours vs Productivity, colored by activity type  
  ggplot(aes(x = hours, y = productivity_score, color = activity_type)) +  
  geom_point() +  
  geom_smooth(method = "lm") +  
  labs(title = "Productivity: Study vs. Social Media")
```

Tidyverse

```
social_media_data %>%  
  # 1. CLEANING (tidyr)  
  # pivot the hours into one 'Activity' column  
  pivot_longer(  
    cols = c(social_media_hours, study_hours),  
    names_to = "activity_type",  
    values_to = "hours"  
  ) %>%  
  |  
  # 2. ANALYSIS (dplyr)  
  # focus_score higher than 7  
  filter(focus_score > 7) %>%  
  
  # 3. PLOTTING (ggplot2)  
  # Hours vs Productivity, colored by activity type  
  ggplot(aes(x = hours, y = productivity_score, color = activity_type)) +  
  geom_point() +  
  geom_smooth(method = "lm") +  
  labs(title = "Productivity: Study vs. Social Media")
```



Tidyverse

- `library(tidyverse)` loads 9 core packages

- ```
4
5 |
6
7 # load the library
8 library(tidyverse)
9
10
11
```

# Tidyverse

- `library(tidyverse)` loads 9 core packages

```
4
5 |
6
7 # load the library
8 library(tidyverse)
9
10
11
```

```
. library(tidyverse)
- Attaching core tidyverse packages ————— tidyverse 2.0.0 —
✓ dplyr 1.2.0 ✓ readr 2.2.0
✓ forcats 1.0.1 ✓ stringr 1.6.0
✓ ggplot2 4.0.2 ✓ tibble 3.3.1
✓ lubridate 1.9.5 ✓ tidyr 1.3.2
✓ purrr 1.0.1
```

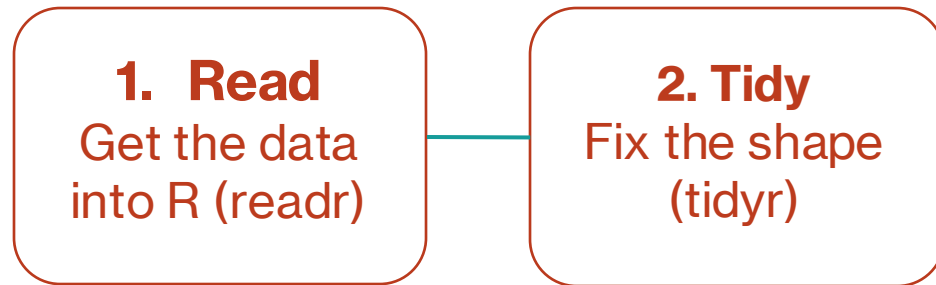
# The Tidy Workflow

# The Tidy Workflow

## **1. Read**

Get the data  
into R (readr)

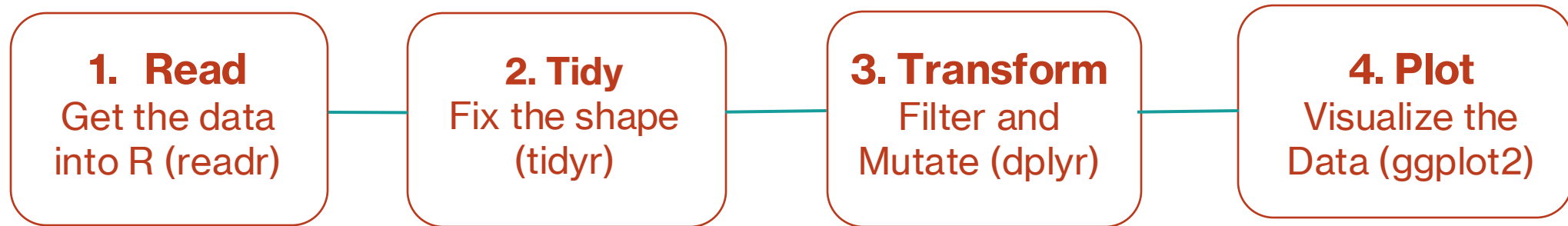
# The Tidy Workflow



# The Tidy Workflow



# The Tidy Workflow



**tidyr**

# The tidy Data!

Rules of a tidy data frame:

variables are columns,

observations are rows, and

values are cells.

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1999 | 2745   | 199857071  |
| Afghanistan | 2000 | 2666   | 200593360  |
| Brazil      | 1999 | 37737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 210766 | 128042583  |

variables

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1999 | 2745   | 199857071  |
| Afghanistan | 2000 | 2666   | 200593360  |
| Brazil      | 1999 | 37737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 210766 | 128042583  |

observations

| country     | year | cases | population |
|-------------|------|-------|------------|
| Afghanistan | 99   | 75    | 98071      |
| Afghanistan | 00   | 66    | 59360      |
| Brazil      | 99   | 737   | 006362     |
| Brazil      | 00   | 0488  | 504898     |
| China       | 99   | 2258  | 2915272    |
| China       | 00   | 0766  | 28042583   |

values

# tidyr: functions

- `pivot_longer()`
- `Pivot_wider()`
- `expand()`
- `complete()`
- Tibbles
- `unite()`
- `separate_wider_delim()`
- `separate_longer_delim()`
- `drop_na()`
- `fill()`
- `replace_na()`
- Nested Data
- `nest()`
- `unnest()`

# Tidy Format using as\_tibble()

```
> print(df)
```

```
t
1 0 · df <- as_tibble(social_media_data)
2 1 · print(df)
3 2 · A tibble: 6,000 × 9
4 3 age daily_screen_time social_media_hours study_hours sleep_hours notifications_per_day
5 4 <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
6 5 1 21 5.95 2.81 2.61 6.99 283
7 6 2 34 3.82 2.33 0.28 7.47 266
8 7 3 29 3.57 1.64 5.21 6.34 137
9 8 4 25 10.3 4.37 4.28 4.49 247
10 9 5 22 2.42 1.6 3.67 6.34 28
11 10 6 35 6.19 2.19 7.26 4.41 218
12 11 7 21 3.68 1.23 1.34 7.41 296
13 12 8 33 10.8 5.65 0.08 7.95 148
14 13 9 37 7.57 5.54 0.59 8.81 81
15 14 · 10 25 4.31 2.73 1.79 4.74 21
16 15 · i 5,990 more rows
· i 3 more variables: focus_score <dbl>, addiction_level <chr>, productivity_score <dbl>
· i Use `print(n = ...)` to see more rows
·
```

# Reshape Data with pivot\_longer()

|                    | C    | D    |
|--------------------|------|------|
| social_media_hours | 2.81 | 2.61 |
|                    | 2.33 | 0.28 |
|                    | 1.64 | 5.21 |
|                    | 4.37 | 4.28 |
|                    | 1.6  | 3.67 |
|                    | 2.19 | 7.26 |
|                    | 1.23 | 1.34 |
|                    | 5.65 | 0.08 |
|                    | 5.54 | 0.59 |
|                    | 2.73 | 1.79 |
|                    | 3.89 | 0.27 |
|                    | 5.95 | 4.02 |
|                    | 6.01 | 2.51 |
|                    | 3.62 | 1.44 |
|                    | 3.62 | 6    |
|                    | 7.98 | 1.64 |

```
>
> # PIVOT_LONGER: Move columns into rows
> df_long <- df %>%
+ pivot_longer(
+ cols = c(social_media_hours, study_hours),
+ names_to = "activity_type",
+ values_to = "hours_spent"
+)
>
> # View the first 6 rows
> df_long %>%
+ select(age, activity_type, hours_spent) %>%
+ head(6)
A tibble: 6 x 3
 age activity_type hours_spent
<dbl> <chr> <dbl>
1 21 social_media_hours 2.81
2 21 study_hours 2.61
3 34 social_media_hours 2.33
4 34 study_hours 0.28
5 29 social_media_hours 1.64
6 29 study_hours 5.21
>
```

# Reshape Data with pivot\_wider

```
> # PIVOT_WIDER: (summary tables)
> df_wide <- df_long %>%
+ pivot_wider(
+ names_from = activity_type,
+ values_from = hours_spent
+)
> # View the first few rows of the "widened" data
> df_wide %>%
+ select(age, social_media_hours, study_hours) %>%
+ head(6)
A tibble: 6 × 3
 age social_media_hours study_hours
 <dbl> <dbl> <dbl>
1 21 2.81 2.61
2 34 2.33 0.28
3 29 1.64 5.21
4 25 4.37 4.28
5 22 1.6 3.67
6 35 2.19 7.26
> |
```

# Expand Tables

- `expand()`
- `complete()`

# Expand Tables with expand()

```
i Use `print(n = ...)` to see more rows
> # Show every possible combination of Age and Addiction Level in our data
> df_grid <- df %>%
+ expand(age, addiction_level)
>
> print(df_grid)
A tibble: 104 x 2
 age addiction_level
 <dbl> <chr>
1 15 High
2 15 Low
3 15 Medium
4 15 NA
5 16 High
6 16 Low
7 16 Medium
8 16 NA
9 17 High
10 17 Low
i 94 more rows
i Use `print(n = ...)` to see more rows
>
```

# Expand Tables with complete()

```
>
>
> # Fill in missing combinations and set their productivity_score to 0
> df_completed <- df %>%
+ complete(age, addiction_level, fill = list(productivity_score = 0))
>
>
> # Find the newly added rows by looking for the ones where daily_screen_time is suddenly NA
> newly_added_rows <- df_completed %>%
+ filter(is.na(daily_screen_time))
>
> # Show the first few missing gaps that were just filled
> print(newly_added_rows)
A tibble: 120 x 9
 age addiction_level daily_screen_time social_media_hours study_hours sleep_hours notifications_per_day focus_score
 <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 15 High NA 6.72 3.71 7.26 239 100
2 15 Medium NA NA 2.76 8.5 144 100
3 15 Medium NA 3.06 4.07 4.17 106 96.0
4 15 Medium NA 3.79 1.09 4.72 237 89.4
5 15 Medium NA 3.03 0.42 8.49 65 100
6 15 NA NA 3.21 4.45 8.83 37 100
7 16 High NA 5.49 3.22 4.74 86 97.7
8 16 High NA 5.84 1.93 6.43 123 92.8
9 16 Low NA 1.68 7.42 4.4 94 100
10 16 Medium NA 3.06 2.02 7.73 289 95.9
i 110 more rows
i 1 more variable: productivity_score <dbl>
i Use `print(n = ...)` to see more rows
> |
```

# Split Cells

- unite()
- separate\_wider\_delim ()
- separate\_longer\_delim ()

# Split Cells with unite()

```
>
> # Fill in missing combinations and set their productivity_score to 0
> df_completed <- df %>%
+ complete(age, addiction_level, fill = list(productivity_score = 0))
>
>
> # Find the newly added rows by looking for the ones where daily_screen_time is suddenly NA
> newly_added_rows <- df_completed %>%
+ filter(is.na(daily_screen_time))
>
> # Show the first few missing gaps that were just filled
> print(newly_added_rows)
A tibble: 120 × 9
 age addiction_level daily_screen_time social_media_hours study_hours sleep_hours notifications_per_day focus_score
 <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 15 High NA 6.72 3.71 7.26 239 100
2 15 Medium NA NA NA 2.76 8.5 144 100
3 15 Medium NA 3.06 4.07 4.17 106 96.0
4 15 Medium NA 3.79 1.09 4.72 237 89.4
5 15 Medium NA 3.03 0.42 8.49 65 100
6 15 NA NA 3.21 4.45 8.83 37 100
7 16 High NA 5.49 3.22 4.74 86 97.7
8 16 High NA 5.84 1.93 6.43 123 92.8
9 16 Low NA 1.68 7.42 4.4 94 100
10 16 Medium NA 3.06 2.02 7.73 289 95.9
i 110 more rows
i 1 more variable: productivity_score <dbl>
i Use `print(n = ...)` to see more rows
~ |
```

# Split Cells with `separate_wider_delim()`

```
> # Split age_addiction back into two separate columns
> df_separated_wide <- df_united %>%
+ separate_wider_delim(
+ cols = age_addiction,
+ delim = "-",
+ names = c("age", "addiction_status")
+)
>
> # the result
> df_separated_wide %>% select(age, addiction_status) %>% head(3)
A tibble: 3 × 2
 age addiction_status
<chr> <chr>
1 21 Medium
2 34 Medium
3 29 Low
> |
```

# Split Cells with `separate_longer_delim()`

```
4 medium
> # Split age_addiction so age is one row and addiction is the next
> df_separated_long <- df_united %>%
+ separate_longer_delim(
+ cols = age_addiction,
+ delim = "-"
+)
>
> # Show the result
> df_separated_long %>% select(age_addiction) %>% head(6)
A tibble: 6 × 1
 age_addiction
 <chr>
1 21
2 Medium
3 34
4 Medium
5 29
6 Low
> |
```

# Handle Missing Values

- `drop_na()`
- `fill()`
- `replace_na()`

# Handle Missing Data with drop\_na()

```
rows dropped: 120
> # DROP_NA: Remove rows where a specific column is missing
> df_clean <- df %>% drop_na(productivity_score)
>
> # Compare the counts
> cat("Original rows:", nrow(df), "\n")
Original rows: 6000
> cat("Cleaned rows:", nrow(df_clean), "\n")
Cleaned rows: 5880
> cat("Rows dropped:", nrow(df) - nrow(df_clean))
Rows dropped: 120
> |
```

# Handle Missing Data with fill()

```
> # fill: Carry a value down (Useful for time-series data)
> df_filled <- df %>% fill(daily_screen_time, .direction = "down")
>
> # Show the change (compare original to filled for a specific slice)
> # We look for where the original had an NA
> target_rows <- which(is.na(df$daily_screen_time))[1:3]
>
> # Show the row before the NA and the filled NAs
> df_filled[c(target_rows[1]-1, target_rows),] %>%
+ select(age, daily_screen_time)
A tibble: 4 × 2
 age daily_screen_time
 <dbl> <dbl>
1 34 7.73
2 39 7.73
3 16 3.6
4 24 10.7
> |
```

# Handle Missing Data with `replace_na()`

```
+ 24 10.7
> # REPLACE_NA: Swap NAs for a specific value (like 0)
> df_replaced <- df %>%
+ mutate(focus_score = replace_na(focus_score, 0))
>
> # Filter for the rows that used to be NA
> # use the original df to find the 'problem' rows, then view them in the new df
> problem_indices <- which(is.na(df$focus_score))[1:5]
>
> df_replaced[problem_indices,] %>%
+ select(age, focus_score)
A tibble: 5 × 2
 age focus_score
 <dbl> <dbl>
1 16 0
2 34 0
3 17 0
4 29 0
5 29 0
> |
```

# Working with nested data

- nest ()
- Unnest ()

# Create Nested Data with nest()

```
> # View the result
> # Nest all columns EXCEPT addiction_level
> df_nested <- df %>%
+ nest(user_data = -addiction_level)
>
> # View the result
> print(df_nested)
A tibble: 4 × 2
 addiction_level user_data
 <chr> <list>
1 Medium <tibble [3,064 × 8]>
2 Low <tibble [959 × 8]>
3 High <tibble [1,857 × 8]>
4 NA <tibble [120 × 8]>
>
```

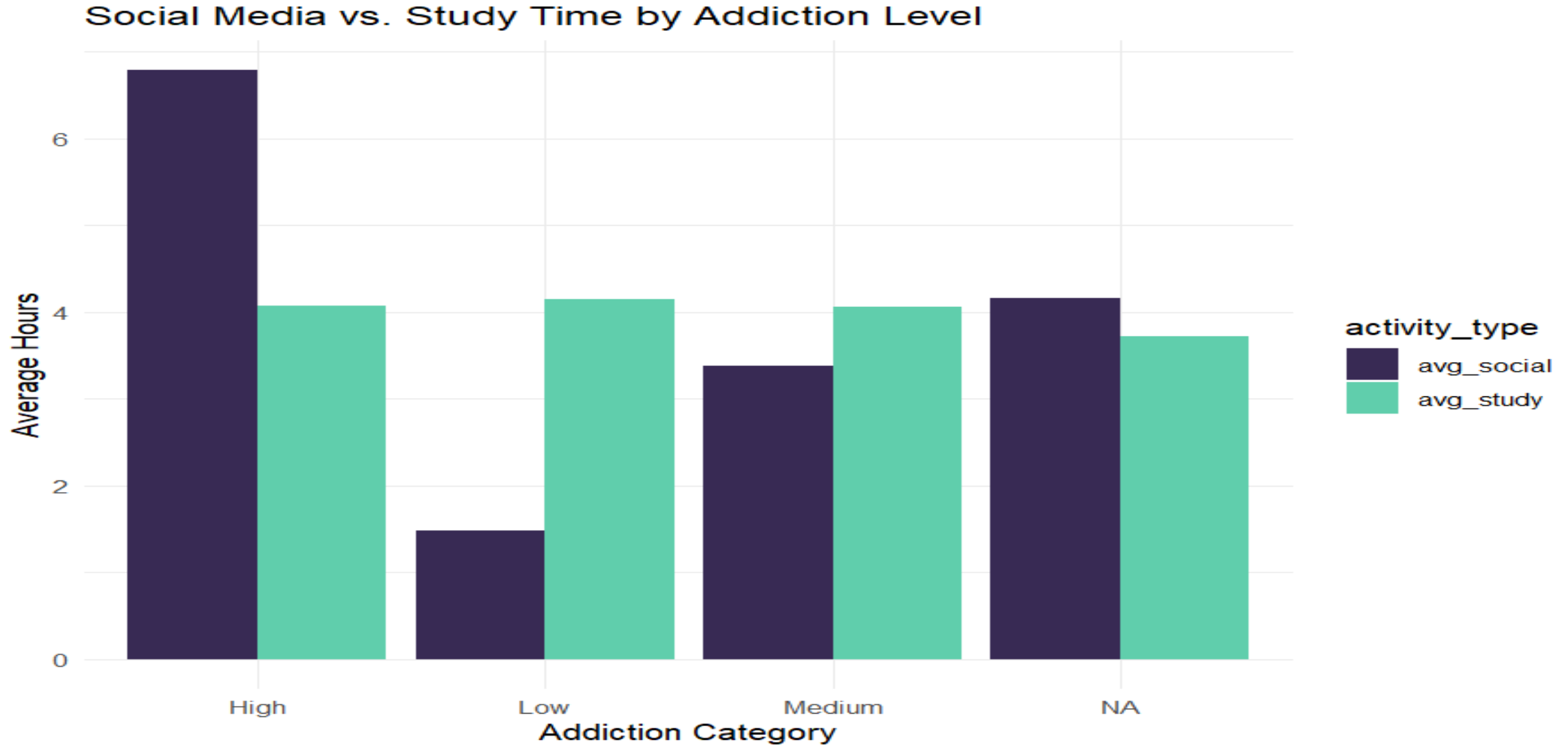
# Working with Nested Data with unnest()

```
- -
> # converting the mini-tables back into a flat table
> df_unpacked <- df_nested %>%
+ unnest(user_data)
>
> # Check the row count
> nrow(df_unpacked)
[1] 6000
>
> # View the result
> print(df_unpacked)
A tibble: 6,000 × 9
 addiction_level age daily_screen_time social_media_hours study_hours sleep_hours notifications_per_day focus_score
 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Medium 21 5.95 2.81 2.61 6.99 283 100
2 Medium 34 3.82 2.33 0.28 7.47 266 93.6
3 Medium 25 10.3 4.37 4.28 4.49 247 94.7
4 Medium 35 6.19 2.19 7.26 4.41 218 100
5 Medium 25 4.31 2.73 1.79 4.74 21 100
6 Medium 18 4.2 3.62 1.44 7.14 102 100
7 Medium 22 10.9 3.62 6 4.73 24 100
8 Medium 35 8.3 3.41 4.09 8.32 160 100
9 Medium 16 6.03 2.88 7.47 4.9 112 100
10 Medium 26 7.31 2.56 4 4.3 128 100
i 5,990 more rows
i 1 more variable: productivity_score <dbl>
i Use `print(n = ...)` to see more rows
~ |
```

# Exercise

Create a bar chart that compares the **Average Social Media Hours** vs. **Average Study Hours** for each **Addiction Level**.

# The End Result



# Snippet

```
252
253 # summary|
254 ex <- df %>%
255 group_by(.....) %>%
256 summarise(
257 avg_social = mean(....., na.rm = TRUE),
258 avg_study = mean(....., na.rm = TRUE)
259)
260
261 print(ex)
262
263
264 # The solution
265 ex_sol <- ex %>%
266 pivot_longer(
267 cols = c(.....,),
268 names_to = "activity_type",
269 values_to = "mean_hours"
270)
271
272 print(ex_sol)
273
274 # plot the solution
275
276 library(ggplot2)
277
278 ggplot(ex_sol, aes(x = addiction_level, y = mean_hours, fill = activity_type)) +
279 geom_col(position = "dodge") +
280 scale_fill_viridis_d(option = "mako", begin = 0.2, end = 0.8) +
281 theme_minimal() +
282 labs(title = "Social Media vs. Study Time by Addiction Level",
283 y = "Average Hours",
284 x = "Addiction Category")
285
```

# Step 1

```
> # summary
> ex <- df %>%
+ group_by(addiction_level) %>%
+ summarise(
+ avg_social = mean(social_media_hours, na.rm = TRUE),
+ avg_study = mean(study_hours, na.rm = TRUE)
+)
>
> print(ex)
A tibble: 4 × 3
 addiction_level avg_social avg_study
 <chr> <dbl> <dbl>
1 High 6.78 4.07
2 Low 1.48 4.14
3 Medium 3.38 4.06
4 NA 4.16 3.72
> |
```

## Hint / Step 2

We need to change this table so we have one column called '**Activity**' and one column called '**Hours**'

We need this make the graph do like 'Color the bars by Activity'."

# Step 2

```
4 NA 4.16 3.72
> # The solution
> ex_sol <- ex %>%
+ pivot_longer(
+ cols = c(avg_social, avg_study),
+ names_to = "activity_type",
+ values_to = "mean_hours"
+)
>
> print(ex_sol)
A tibble: 8 × 3
 addiction_level activity_type mean_hours
 <chr> <chr> <dbl>
1 High avg_social 6.78
2 High avg_study 4.07
3 Low avg_social 1.48
4 Low avg_study 4.14
5 Medium avg_social 3.38
6 Medium avg_study 4.06
7 NA avg_social 4.16
8 NA avg_study 3.72
> |
```

# Plot the Solution

```
234 # plot the solution
235
236 library(ggplot2)
237
238 ggplot(ex_sol, aes(x = addiction_level, y = mean_hours, fill = activity_type)) +
239 geom_col(position = "dodge") +
240 scale_fill_viridis_d(option = "mako", begin = 0.2, end = 0.8) +
241 theme_minimal() +
242 labs(title = "Social Media vs. Study Time by Addiction Level",
243 y = "Average Hours",|
244 x = "Addiction Category")
245
```

# Digital Habits vs. Human Productivity

*Using Tidyverse to investigate how daily screen time impacts sleep, focus, and productivity.*

Tools:

**dplyr**

Cleaning & Calc

**tidyr**

Reshaping

**ggplot2**

Visualisation

01

Data Import & Clean

02

dplyr Transformations

03

Tidyr Reshaping

04

Visualisations

05

Exercise & Q&A

# The Dataset — Raw View

Before cleaning: 6,000 entries with missing values and empty strings

|    | age | daily_screen_time | social_media_hours | study_hours | sleep_hours | notifications_per_day | focus_score | addiction_level | productivity_score |
|----|-----|-------------------|--------------------|-------------|-------------|-----------------------|-------------|-----------------|--------------------|
| 1  | 21  | 5.95              | 2.81               | 2.61        | 6.99        | 283                   | 100.00      | Medium          | 28.49              |
| 2  | 34  | 3.82              | 2.33               | 0.28        | 7.47        | 266                   | 93.65       | Medium          | 18.54              |
| 3  | 29  | 3.57              | 1.64               | 5.21        | 6.34        | 137                   | 100.00      | Low             | 68.52              |
| 4  | 25  | 10.27             | 4.37               | 4.28        | 4.49        | 247                   | 94.71       | Medium          | 27.82              |
| 5  | 22  | 2.42              | 1.60               | 3.67        | 6.34        | 28                    | 100.00      | Low             | 51.09              |
| 6  | 35  | 6.19              | 2.19               | 7.26        | 4.41        | 218                   | 100.00      | Medium          | 67.38              |
| 7  | 21  | 3.68              | 1.23               | 1.34        | 7.41        | 296                   | 100.00      | Low             | 37.22              |
| 8  | 33  | 10.79             | 5.65               | 0.08        | 7.95        | 148                   | 83.64       | High            | 0.00               |
| 9  | 37  | 7.57              | 5.54               | 0.59        | 8.81        | 81                    | 94.43       | High            | 0.00               |
| 10 | 25  | 4.31              | 2.73               | 1.79        | 4.74        | 21                    | 100.00      | Medium          | 41.86              |
| 11 | 25  | 7.02              | 3.89               | 0.27        | 6.66        | 191                   | 93.10       |                 | 3.38               |
| 12 | 38  | 9.31              | 5.95               | 4.02        | 6.84        | NA                    | 88.88       | High            | 16.42              |
| 13 | 35  | 11.58             | 6.01               | 2.51        | 8.61        | NA                    | 100.00      | High            | 9.91               |
| 14 | 18  | 4.20              | 3.62               | 1.44        | 7.14        | 102                   | 100.00      | Medium          | 8.89               |
| 15 | 22  | 10.86             | 3.62               | 6.00        | 4.73        | 24                    | 100.00      | Medium          | 53.49              |
| 16 | 38  | 11.34             | 7.98               | 1.64        | 6.18        | 58                    | 77.17       |                 | 0.00               |

Showing 1 to 17 of 6,000 entries, 9 total columns

# Dataset Variables

9 fields tracking digital habits and student academic performance

**age**

numeric

15 – 39 years

**daily\_screen\_time**

numeric

2 – 12 hrs

**social\_media\_hours**

numeric

0.6 – 10.6 hrs

**study\_hours**

numeric

0 – 8 hrs

**sleep\_hours**

numeric

4 – 9 hrs

**notifications\_per\_day**

numeric

20 – 299

**focus\_score**

numeric

47 – 100

**addiction\_level**

character

Low / Medium / High

**productivity\_score**

numeric

0 – 100

# The 'Messy Data' Challenge

Real-world datasets are never clean — here is what we faced

## ✘ Missing Values (NA)

R reads blank cells as NA. Functions like `mean()` return NA unless handled first.

## 👻 Invisible Junk — Empty Strings ""

Cells that look empty but contain "" — R does NOT treat these as NA automatically.

## ⚠️ Why This Matters

Running analysis on dirty data gives wrong results. Clean first, always.

```
● ● ●

Step 1 – tag empty strings
data[data == ""] <- NA
[object Object]

Step 2 – remove all NA rows
data_clean <- data %>%
 drop_na()
[object Object]

Step 3 – add unique ID
data_clean <- data_clean %>%
 mutate(user_id = row_number())
```

# The 'Messy Data' Challenge

Two steps = 100% accuracy. Skipping Step 1 leaves 107 contaminated rows.



## 💡 Why Two Steps?

`drop_na()` only removes rows with actual NA values — it completely ignores empty strings "".

By converting "" to NA first, we catch ALL missing data. Skipping Step 1 leaves 107 rows of dirty data in your analysis!

```
• • •

Correct two-step process:
data[data == ""] <- NA # Step 1
data_clean <- data %>% # Step 2
 drop_na()
[object Object]

WRONG — skips empty strings:
data_clean <- data %>%
 drop_na() # misses 107 rows!
```

# Dataset After drop\_na()

5,104 rows — still contains dirty data because empty strings were not converted first

|    | age | daily_screen_time | social_media_hours | study_hours | sleep_hours | notifications_per_day | focus_score | addiction_level | productivity_score |
|----|-----|-------------------|--------------------|-------------|-------------|-----------------------|-------------|-----------------|--------------------|
| 1  | 21  | 5.95              | 2.81               | 2.61        | 6.99        | 283                   | 100.00      | Medium          | 28.49              |
| 2  | 34  | 3.82              | 2.33               | 0.28        | 7.47        | 266                   | 93.65       | Medium          | 18.54              |
| 3  | 29  | 3.57              | 1.64               | 5.21        | 6.34        | 137                   | 100.00      | Low             | 68.52              |
| 4  | 25  | 10.27             | 4.37               | 4.28        | 4.49        | 247                   | 94.71       | Medium          | 27.82              |
| 5  | 22  | 2.42              | 1.60               | 3.67        | 6.34        | 28                    | 100.00      | Low             | 51.09              |
| 6  | 35  | 6.19              | 2.19               | 7.26        | 4.41        | 218                   | 100.00      | Medium          | 67.38              |
| 7  | 21  | 3.68              | 1.23               | 1.34        | 7.41        | 296                   | 100.00      | Low             | 37.22              |
| 8  | 33  | 10.79             | 5.65               | 0.08        | 7.95        | 148                   | 83.64       | High            | 0.00               |
| 9  | 37  | 7.57              | 5.54               | 0.59        | 8.81        | 81                    | 94.43       | High            | 0.00               |
| 10 | 25  | 4.31              | 2.73               | 1.79        | 4.74        | 21                    | 100.00      | Medium          | 41.86              |
| 11 | 25  | 7.02              | 3.89               | 0.27        | 6.66        | 191                   | 93.10       |                 | 3.38               |
| 12 | 18  | 4.20              | 3.62               | 1.44        | 7.14        | 102                   | 100.00      | Medium          | 8.89               |
| 13 | 22  | 10.86             | 3.62               | 6.00        | 4.73        | 24                    | 100.00      | Medium          | 53.49              |
| 14 | 38  | 11.34             | 7.98               | 1.64        | 6.18        | 58                    | 77.17       |                 | 0.00               |
| 15 | 17  | 11.16             | 9.32               | 5.50        | 7.27        | 94                    | 99.17       | High            | 18.36              |

Showing 1 to 15 of 5,104 entries, 9 total columns

# Dataset After Full Clean

4,997 rows — both steps applied, data is now reliable for analysis

|    | age | daily_screen_time | social_media_hours | study_hours | sleep_hours | notifications_per_day | focus_score | addiction_level | productivity_score |
|----|-----|-------------------|--------------------|-------------|-------------|-----------------------|-------------|-----------------|--------------------|
| 1  | 21  | 5.95              | 2.81               | 2.61        | 6.99        | 283                   | 100.00      | Medium          | 28.49              |
| 2  | 34  | 3.82              | 2.33               | 0.28        | 7.47        | 266                   | 93.65       | Medium          | 18.54              |
| 3  | 29  | 3.57              | 1.64               | 5.21        | 6.34        | 137                   | 100.00      | Low             | 68.52              |
| 4  | 25  | 10.27             | 4.37               | 4.28        | 4.49        | 247                   | 94.71       | Medium          | 27.82              |
| 5  | 22  | 2.42              | 1.60               | 3.67        | 6.34        | 28                    | 100.00      | Low             | 51.09              |
| 6  | 35  | 6.19              | 2.19               | 7.26        | 4.41        | 218                   | 100.00      | Medium          | 67.38              |
| 7  | 21  | 3.68              | 1.23               | 1.34        | 7.41        | 296                   | 100.00      | Low             | 37.22              |
| 8  | 33  | 10.79             | 5.65               | 0.08        | 7.95        | 148                   | 83.64       | High            | 0.00               |
| 9  | 37  | 7.57              | 5.54               | 0.59        | 8.81        | 81                    | 94.43       | High            | 0.00               |
| 10 | 25  | 4.31              | 2.73               | 1.79        | 4.74        | 21                    | 100.00      | Medium          | 41.86              |
| 11 | 18  | 4.20              | 3.62               | 1.44        | 7.14        | 102                   | 100.00      | Medium          | 8.89               |
| 12 | 22  | 10.86             | 3.62               | 6.00        | 4.73        | 24                    | 100.00      | Medium          | 53.49              |
| 13 | 17  | 11.16             | 9.32               | 5.50        | 7.27        | 94                    | 99.17       | High            | 18.36              |
| 14 | 36  | 8.35              | 6.53               | 2.79        | 7.50        | 120                   | 98.15       | High            | 0.00               |
| 15 | 35  | 8.30              | 3.41               | 4.09        | 8.32        | 160                   | 100.00      | Medium          | 37.84              |

Showing 1 to 15 of 4,997 entries, 9 total columns

# Transform: SELECT · FILTER · MUTATE

*The three most-used dplyr verbs for reshaping your data*

## select()

```

2. SELECT
Choose important columns

selected_data <- data_clean %>%
 select(user_id,
 age,
 daily_screen_time,
 social_media_hours,
 study_hours,
 sleep_hours)
```

Pick only the columns you need — reduces clutter

## filter()

```

3. FILTER
Keep users with high screen time

high_screen <-
 data_clean %>%
 filter(
 daily_screen_time > 6
)
```

2,962 users had high screen time

## mutate()

```

4. MUTATE
Create new columns

data_clean <- data_clean %>%
 mutate(
 screen_time_category =
 ifelse(daily_screen_time > 6,
 "High", "Low"
),
 sleep_category =
 ifelse(sleep_hours >= 7,
 "Good Sleep", "Less Sleep"
)
)
```

Creates 2 new categorical columns

# Dataset Overview

Transformed from 10 to 6 columns – and 2 new columns

|    | user_id | age | daily_screen_time | social_media_hours | study_hours | sleep_hours | screen_time_category | sleep_category |
|----|---------|-----|-------------------|--------------------|-------------|-------------|----------------------|----------------|
| 1  | 1       | 21  | 5.95              | 2.81               | 2.61        | 6.99        | Low                  | Less Sleep     |
| 2  | 2       | 34  | 3.82              | 2.33               | 0.28        | 7.47        | Low                  | Good Sleep     |
| 3  | 3       | 29  | 3.57              | 1.64               | 5.21        | 6.34        | Low                  | Less Sleep     |
| 4  | 4       | 25  | 10.27             | 4.37               | 4.28        | 4.49        | High                 | Less Sleep     |
| 5  | 5       | 22  | 2.42              | 1.60               | 3.67        | 6.34        | Low                  | Less Sleep     |
| 6  | 6       | 35  | 6.19              | 2.19               | 7.26        | 4.41        | High                 | Less Sleep     |
| 7  | 7       | 21  | 3.68              | 1.23               | 1.34        | 7.41        | Low                  | Good Sleep     |
| 8  | 8       | 33  | 10.79             | 5.65               | 0.08        | 7.95        | High                 | Good Sleep     |
| 9  | 9       | 37  | 7.57              | 5.54               | 0.59        | 8.81        | High                 | Good Sleep     |
| 10 | 10      | 25  | 4.31              | 2.73               | 1.79        | 4.74        | Low                  | Less Sleep     |
| 11 | 11      | 18  | 4.20              | 3.62               | 1.44        | 7.14        | Low                  | Good Sleep     |
| 12 | 12      | 22  | 10.86             | 3.62               | 6.00        | 4.73        | High                 | Less Sleep     |
| 13 | 13      | 17  | 11.16             | 9.32               | 5.50        | 7.27        | High                 | Good Sleep     |
| 14 | 14      | 36  | 8.35              | 6.53               | 2.79        | 7.50        | High                 | Good Sleep     |
| 15 | 15      | 35  | 8.30              | 3.41               | 4.09        | 8.32        | High                 | Good Sleep     |
| 16 | 16      | 16  | 6.03              | 2.88               | 7.47        | 4.90        | High                 | Less Sleep     |

# group\_by() + summarise() — The Magic Combo

Stop looking at individuals. Start seeing population-level trends.

```

6. GROUP_BY + SUMMARISE
Summary by addiction level

addiction_summary <- data_clean %>%
 group_by(addiction_level) %>%
 summarise(
 avg_screen_time = mean(daily_screen_time, na.rm = TRUE),
 avg_social_media = mean(social_media_hours, na.rm = TRUE),
 avg_focus = mean(focus_score, na.rm = TRUE),
 avg_productivity = mean(productivity_score, na.rm = TRUE),
 count = n()
)
```

## Results:

| Addiction Level | Count | Avg Screen | Avg Prod. | Avg Focus |
|-----------------|-------|------------|-----------|-----------|
| High            | 1,583 | 9.53 hrs   | 20.4      | 90.8      |
| Medium          | 2,605 | 6.48 hrs   | 42.2      | 99.8      |
| Low             | 809   | 3.26 hrs   | 57.3      | 98.7      |

Syntax: `rename( NEW_NAME = old_name )`

### Remember:

`rename()` only changes the label — never the data values. And always rename BEFORE your `ggplot2` charts to auto-label axes!

### ★ Pro-Tip for Your Report

Always rename your columns BEFORE building your final `ggplot2` charts. The column name becomes the default axis label — rename first and skip the manual `labs()` step entirely!

# rename() — Making Data "Human"

*Clean code is readable code. Good names save hours of confusion.*

```

7. RENAME
Rename columns for readability

renamed_data <- data_clean %>%
 rename(
 screen_time = daily_screen_time,
 social_time = social_media_hours,
 study_time = study_hours,
 sleep_time = sleep_hours
)
```

**BEFORE rename()**

daily\_screen\_time

social\_media\_hours

study\_hours

sleep\_hours

**AFTER rename()**

→ screen\_time

→ social\_time

→ study\_time

→ sleep\_time

## ★ Pro-Tip for Your Final Report

Always rename your columns BEFORE building your final ggplot2 charts. The column name becomes the default axis label — so if you rename first, you skip the manual labs() step entirely!

# Reshaping Data with tidyr

Humans love Wide data (columns for every day).

Computers love Long data (one column for "Day", one for "Value").

Tidyr makes this conversion a one-line process.

# tidyr · pivot\_longer() & pivot\_wider()

Reshape between wide and long formats — essential for ggplot2

## Wide Format (original)

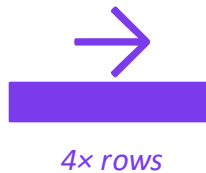
| user_id | age | daily_screen_time | social_media_hrs | study_hours | sleep_hours |
|---------|-----|-------------------|------------------|-------------|-------------|
| 1       | 21  | 5.95              | 2.81             | 2.61        | 6.99        |
| 2       | 34  | 3.82              | 2.33             | 0.28        | 7.47        |
| 3       | 29  | 3.57              | 1.64             | 5.21        | 6.34        |

### pivot\_longer() code:

```

13. TIDYR - pivot_longer
Convert wide to long

long_data_clean <- data_clean %>%
 select(user_id, age, daily_screen_time, social_media_hours, study_hours, sleep_hours) %>%
 pivot_longer(
 cols = c(daily_screen_time, social_media_hours, study_hours, sleep_hours),
 names_to = "activity_type",
 values_to = "hours"
)
```



## Long Format (after pivot\_longer)

| user_id | age | activity_type      | hours |
|---------|-----|--------------------|-------|
| 1       | 21  | daily_screen_time  | 5.95  |
| 1       | 21  | social_media_hours | 2.81  |
| 1       | 21  | study_hours        | 2.61  |
| 1       | 21  | sleep_hours        | 6.99  |
| 2       | 34  | daily_screen_time  | 3.82  |

19,988 rows after pivot (4,997 users × 4 activity types)

PART 3

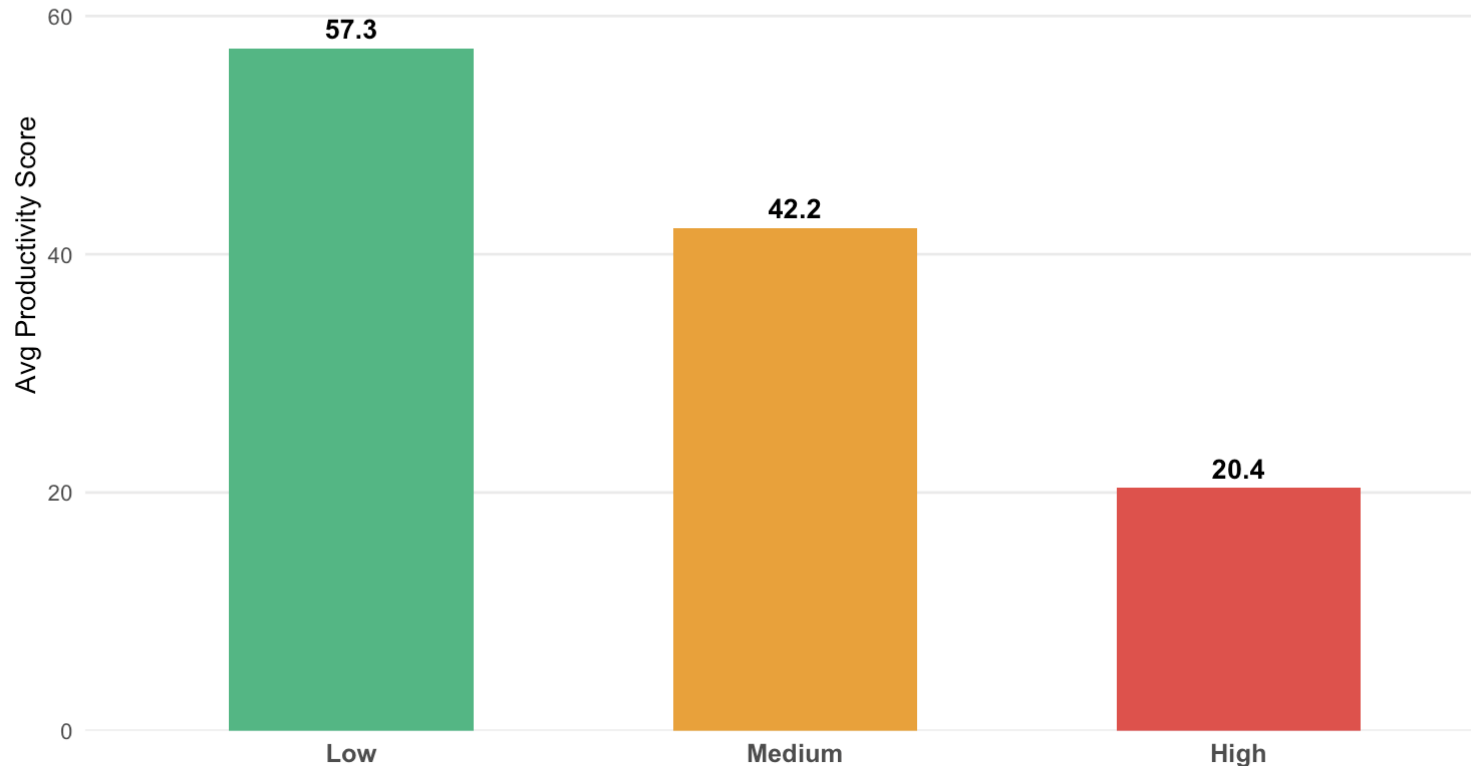
# Let's Look at the Visualisations



*3 charts · 3 insights · 1 clear story*

# More Addicted = Less Productive

*Average productivity score drops sharply as addiction level rises*



**57.3**

*avg productivity*

**Low Addiction**

**42.2**

*avg productivity*

**Medium Addiction**

**20.4**

*avg productivity*

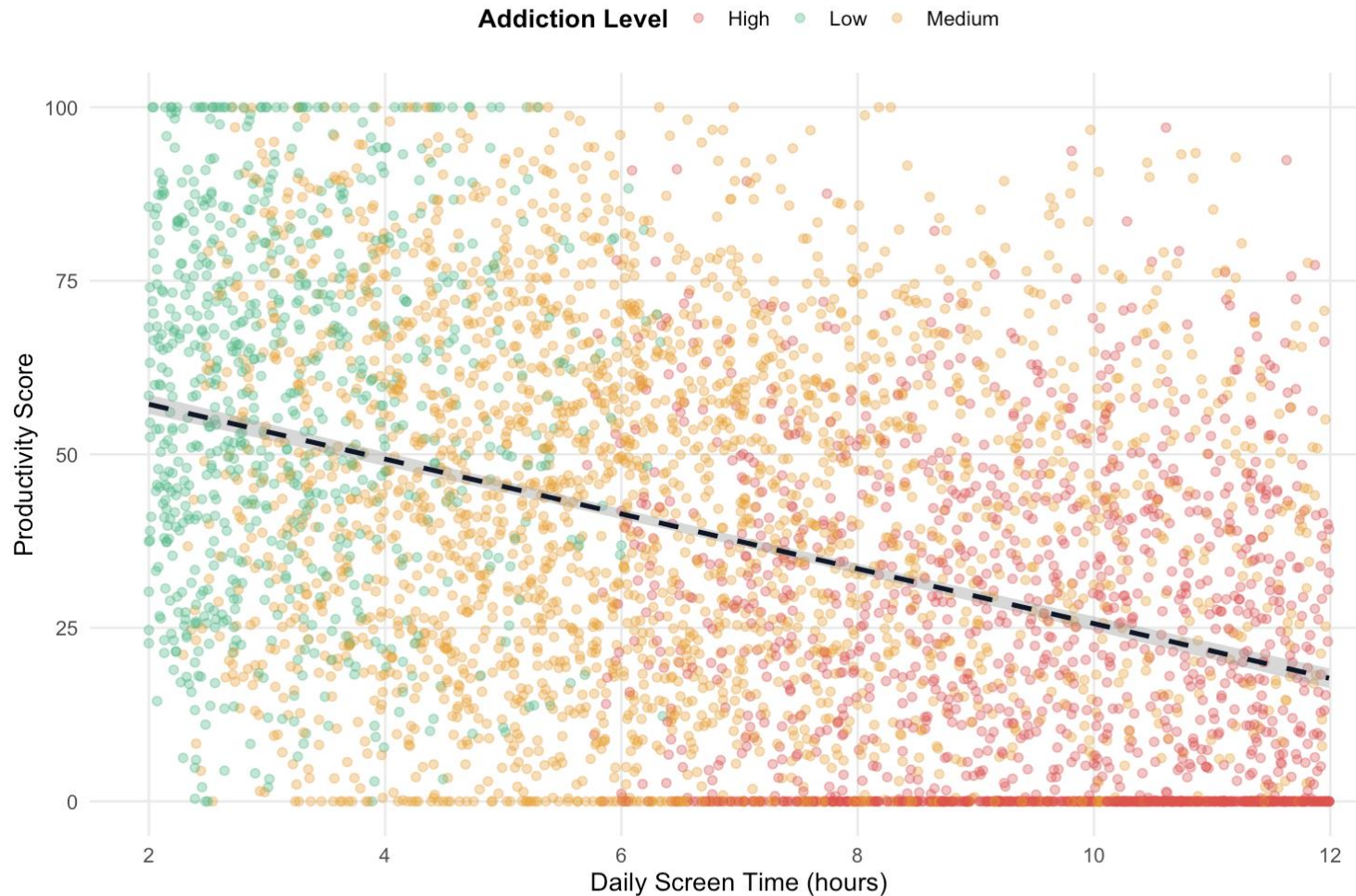
**High Addiction**



**Low is 4.9× more productive than High**

# More Screen Time, Less Done

Each dot is one student · Trend line shows the overall decline · Colored by addiction level



## What to look for:

### Downward trend

Dashed regression line slopes down — screen time up, productivity down.

### Green dots (Low)

Cluster bottom-left: low screen time & high productivity.

### Red dots (High)

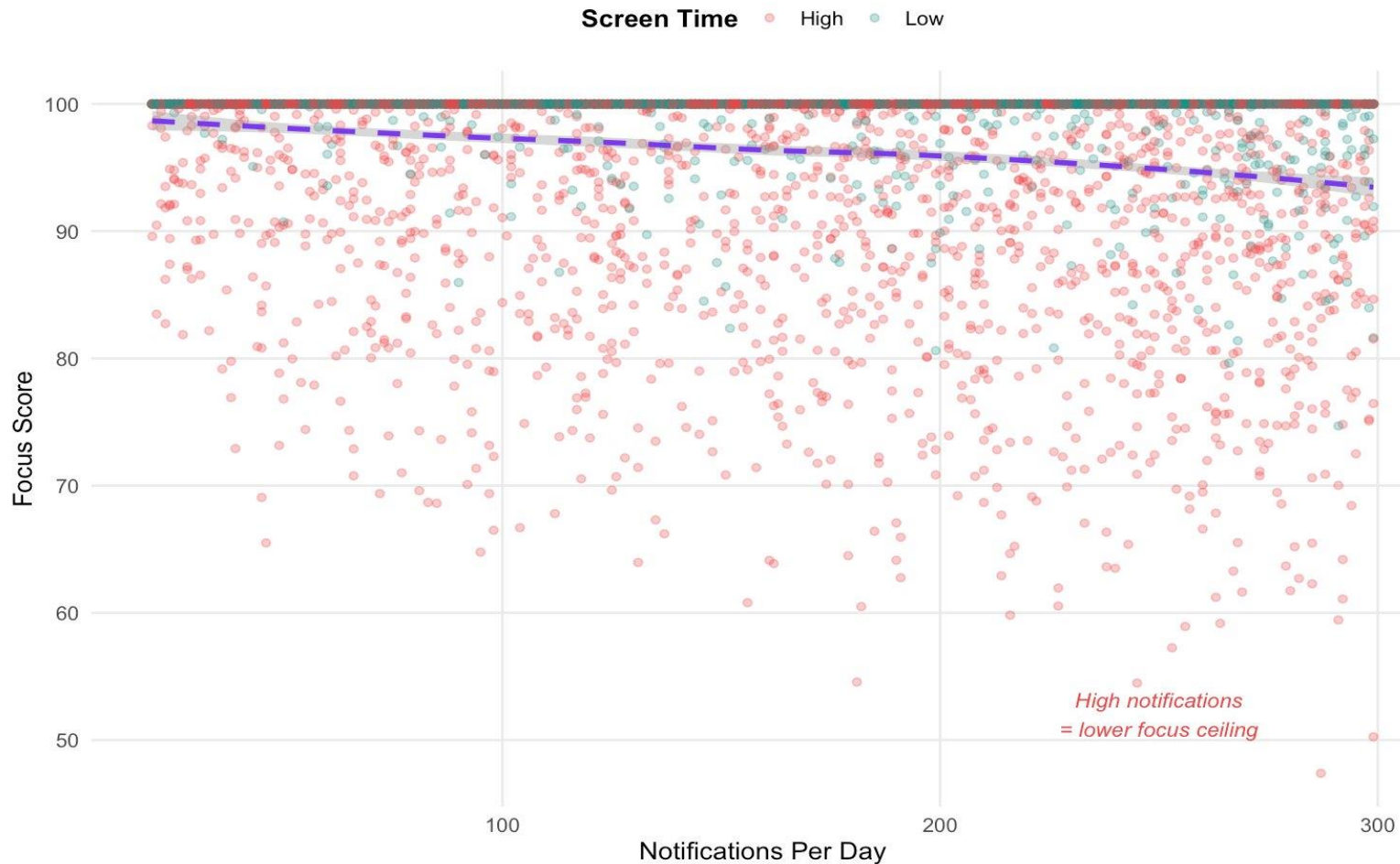
Cluster top-right: high screen time & low productivity.

### Outliers

A few red dots with high productivity — the interesting exceptions!

# Constant Pings Kill Concentration

*Students with more daily notifications trend toward lower focus scores*



## What to notice:

**● Pink dots = High screen time**

Spread much lower — some students scoring as low as 47–50.

**● Teal dots = Low screen time**

Stay mostly near 100 — low screen time protects focus.

**~ Purple dashed line**

Loess trend — gently slopes down as notifications increase.

**💡 Key takeaway**

It's not just notifications — HIGH screen time amplifies the damage.

# Exercise — Your Turn!


Apply what you have learned to answer a real research question

## Task

Use `filter()` and `summarise()` to find the average `study_hours` for only the users who have "Good Sleep" (`sleep_category == "Good Sleep"`).

## Starter Code (fill in the blanks):

```
data_clean %>%
 filter(sleep_category == "____") %>%
 summarise(
 avg_study = ____ (study_hours)
)
```

 Take 3 minutes — write your code, then we reveal the answer together!

## Hints

1. `filter()` keeps matching rows
2. The value is a text string
3. `summarise()` collapses data
4. Which function calculates the average?

# Solution — Revealed!

*Here is what the correct code looks like and what it tells us*



```
Complete solution:
```

```
data_clean %>%
 filter(sleep_category == "Good Sleep") %>%
 summarise(
 avg_study = mean(study_hours)
)
```

**~4.07 Hours**

*average study hours for Good Sleep students*

# Questions?

---

*Mastering dplyr and tidyr is 80%  
of the battle in Data Science.*

Once your data is tidy, the insights follow automatically.

**tidyr**

```
library(tidyr)
```

**dplyr**

```
library(dplyr)
```

**ggplot2**

```
library(ggplot2)
```