# Part 2: Data Cleaning

Matt Steele

2023-09-20

## Resources

- Tidyverse Documentation
- O'Reilly Learning Platform
    - R for Data Science, 2nd Edition
    - R Programming for Statistics and Data Science
    - Text Mining with R

# Part 1: About Tidyverse

```
install.packages("tidyverse")
```

```
library(tidyverse)
```

Tidyverse is a collection of packages focused on data analysis and data visualizations that share an underlying design philosophy, grammar, and data structures.

Packages Includes in Tidyverse

| | |
|---|---|
| tibble | lighter and more user-friendly version of data frames |
| tidyr | create tidy and meaningfully arranged data |
| readr | better importation of data into R |
| ggplot | data visualization functions |
| dplyr | data manipulation tools |
| lubridate | clean dates and times |
| purr | better functional programming |
| forcats | handle, clean, and manipulate categorical variables |
| haven | read and write data formats from proprietary statistical pacakges |

# Part 2: Loading Data with Tidyverse

## Loading Data

### Tibble function: read_csv

this function allows you to read a csv file into a tibble data frame

```
reviews_df <- read_csv("customer_reviews.csv")
```

### Tibble function: view

tibble also provides a lighter way for you to view you entire data frame.

```
view(reviews_df)
```

### The Pipe Operator

The pipe operator allows you to run commands or operation on a single object based on an order of operations

- let's say you want to see the **age**, **rating**, and **feedback** of customer who shopped in the *dresses* department

```
view(reviews_df)

reviews_df |> # object we are working on
  filter(Department_Name == "Dresses") |> # first operation
  select(Clothing_ID, Age, Rating, Positive_Feedback_Count) # second operation

# order of operations matter

reviews_df |> # object we are working on
  select(Clothing_ID, Age, Rating, Positive_Feedback_Count) |> # first operation
  filter(Department_Name == "Dresses") # second operation

# why did this not work?
```

## Readr function: read_excel | read_xlsx

this function allows you to read Excel files in a tibble data frame

```
breed_rank_all <- readxl::read_xlsx("dog_breeding.xlsx", sheet = 1)
breed_traits <- readxl::read_xlsx("dog_breeding.xlsx", sheet = 2)
```

## Dplyr function: mutating join functions

this function allows you to join multiple data frames together by a common variable(s)

mutating join functions

| | |
|---|---|
| left_join | if observations in (y) are in (x) then add (y) to (x) |
| right_join | if observations in (x) are in (y) then add (x) to (y) |
| full_join | add all observations from (x) to (y) |
| anti_join | remove entries from data frame (y) |

```
breed_rank_all # ranks of dog breeds
breed_traits # traits of dog breeds

breed_df <- breed_rank_all |>
  left_join(breed_traits, by = "Breed")

breed_df
```

## Haven package: Read non-proprietary data files

The package Haven allows you to read and export non-proprietary files for SPSS, SAS, and STATA

```
demographics_df <- haven::read_sav("demographics.sav")
```

### Glimpse function

like the str() function in base r, this allow you see the structure of your data but in a more compact manner

```
glimpse(demographics_df)
```

## Part 3: Cleaning Data

Main Tidyverse Functions

| | |
|---|---|
| filter | retains or filters out observations based on variable criteria |
| select | retains or filters out variables |
| arrange | sorts variables |
| mutate | change variable's observations OR create a new variable and observations using observations from another variable |
| group_by | group observations |
| summarise | get descriptive statistics about a variable |

## Dplyr function: filter

the filter function allows you to *filter* your data frame using a observations in a variable

```
reviews_df

# let's filter the data frame so we are seeing people who bought pants

reviews_pants <- reviews_df |>
  filter(Class_Name == "Pants")

reviews_pants
```

### Boolean operators

boolean operators allows you to build criteria in your code

Boolean operators

| | |
|---|---|
| & | AND |
| \| | OR |
| == | EQUAL |
| != | NOT EQUAL |
| < | LESS THAN |
| > | GREATER THAN |
| <= | LESS THAN OR EQUAL |
| >= | GREATER THAN OR EQUAL |

```
# let's filter the data frame for people who did not buy pants and are over 50 years ol
d

reviews_pants50 <- reviews_df |>
  filter(Class_Name != "Pants" & Age > 50)

reviews_pants50
```

## Dplyr function: select

the select function allows you to *keep* or *discard* variables

```
# let's just keep the numeric information

reviews_select <- reviews_df |>
  select(Clothing_ID, Age, Rating, Recommended_IND)

reviews_select

# let's remove the division and department name variables from our data frame

reviews_notSelect <- reviews_pants50 |>
  select(-Division_Name, -Department_Name)

reviews_notSelect
```

## Dplyr function: mutate

the mutate function allows you to change variables OR create new variables based on the observations in the variables

*NOTE: if you name your variable as an existing variable, it WILL overwrite the existing varaible. If you give it a new name, it will create a new variable

```
demographics_df

demographics_mutate <- demographics_df |>
  mutate(income_new = income/1000)  |>
  relocate(income_new, .after = income)

demographics_mutate

# let's overwrite the old variable

demographics_overwrite <- demographics_df |>
  mutate(income = income/1000)

demographics_overwrite
```

## Dplyr function: arrange

the arrange function allows you to sort variables

```
# let's look in the customers review data frame and see what customers gave the highest
 rating

reviews_df |>
  arrange(desc(Rating))

# you can add multiple criteria to your sort

reviews_df |>
  arrange(desc(Rating), Class_Name) |>
  select(Clothing_ID, Rating, Class_Name)
```

## Dplyr function: group_by & summarise

the **group_by** function allows you to group common observations in a variable and **summarise** function allows you to get descriptive statistics about the groupings

```
reviews_group <- reviews_df |>
  group_by(Class_Name) |>
  summarise(mean(Rating)) # get the average for ratings

reviews_group

# you can add as many summaries as you want

reviews_group <- reviews_df |>
  group_by(Class_Name) |>
  summarise(n = n(), # get the instances count
            positive_tot = sum(Positive_Feedback_Count), # get the sum of positive feed
back
            positive_avg = mean(Positive_Feedback_Count), # get the average of positive
 feedback
            rating = mean(Rating)) # get the average rating


# you can also add multiple groups

reviews_group <- reviews_df |>
  group_by(Class_Name, Recommended_IND) |>
  summarise(n = n(),
            positive_tot = sum(Positive_Feedback_Count),
            positive_avg = (sum(Positive_Feedback_Count)/n),
            rating = mean(Rating))
```

## Base Function: as.character

This function along with **mutate** will allow you to change the data type of a variable. For this example we are going to recode the *Clothing_ID* variable to interpret the data type as a *character* instead of a *double*

```
reviews_df <- reviews_df |>
  mutate(Clothing_ID = as.character(Clothing_ID))

reviews_df
```

## Forcats function: as_factor

This function allows you to redefine a variable value as a factor using the **mutate** function. For this example we are going redefine the recommend variable in the data frame we created when we grouped the customers.

```
is.factor(reviews_group$Recommended_IND)

reviews_df <- reviews_df |>
  mutate(Recommended_IND = as_factor(Recommended_IND))

reviews_df
```

## Dplyr function: mutate_at

You can also change the data type of multiple variables using the **mutate_at** function. For this example we will change the division, department, and class variables from *characters* into *factors*.

```
reviews_df <- reviews_df |>
  mutate_at(vars(Division_Name, Department_Name, Class_Name), as_factor)

reviews_df
```

---

## Dyplr Function: recode

we can rename the values of observations within a variable using the **mutate** function in combination with the **recode** or **recode_factor** functions

```
# we can look at what levels a factor variable has, so long as r is interpreting it as
a factor variable



levels(reviews_df$Recommended_IND)

# let's change the values from 1/0 to yes/no

reviews_df <- reviews_df |>
  mutate(Recommended_IND = recode_factor(Recommended_IND,
                                         "1" = "Yes",
                                         "0" = "No"))



reviews_recode
```

---

## Tidyr Function: Pivot wider

we can also do the inverse of pivot longer and widen our data

```
# let's create new variables from the recommend variable

reviews_wide <- reviews_df |>
  group_by(Class_Name, Recommended_IND) |>
  summarise(number = n(),
            feedback = mean(Positive_Feedback_Count)) |>
  pivot_wider(names_from = Recommended_IND, values_from = c(number, feedback))

reviews_wide
```

---

## Dplyr function: rename

the rename function allows you rename variables in your data frame

```
glimpse(reviews_wide)

reviews_wide <- reviews_wide |>
  rename("Not_Recommend" = number_No, "Recommend" = number_Yes)

reviews_wide
```

---

## Tidyr function: drop_na

we can remove **all** missing data from data frames or variables using the drop_na function

```
reviews_wide

# we can see if are data frame has missing NA values using the is.na function.

is.na(reviews_wide)
which(is.na(reviews_wide))

# because there are missing values we cannot calculate some descriptive statistics

mean(reviews_wide$Not_Recommend)
```

```r
# we can drop all NA values from the data frame

reviews_dropNA <- reviews_wide |>
  drop_na()

reviews_dropNA

mean(reviews_dropNA$Not_Recommend)

# we can also just drop NAs from a variable

reviews_dropNA_var <- reviews_wide |>
  drop_na(Not_Recommend)

reviews_dropNA_var
```

## Tidyr function: replace_na

you can also recode the NA values for observations with the replace_na function

```r
reviews_wide

# let's replace the NAs in the Not recommended to 0 since there were no reviews given

reviews_replaceNA <- reviews_wide |>
  mutate(Not_Recommend = replace_na(Not_Recommend, 0))

reviews_replaceNA

mean(reviews_replaceNA$Not_Recommend)
```

```r
covid19 <- read_csv("time_series_covid19_confirmed_US.csv")

view(covid19)
```

## Tidyr function: Pivot Data

pivoting data allows you to either move variables into observations or observations into variables.

```r
# let's turn the date variables into observations

covid19_longer <- covid19 |>
  pivot_longer(!Combined_Key, names_to = "Date", values_to = "Total")


covid19_longer
```

## Tidyr function: Separate, Unite, and Gather

these functions allow you to split up or combine variables in your data frame

```r
covid19_separate <- covid19_longer |>
  separate_wider_delim(Combined_Key,
                       delim = ", ",
                       names = c("City", "State", "Country"),
                       too_few = "align_start") |>
  select(-Country)

covid19_separate
```

## Lubridate function: as_date

the lubridate package allows you to clean and manipulate data variables

```r
glimpse(covid19_separate)

covid19_date <- covid19_separate |>
  mutate(Date = lubridate::dmy(Date))

covid19_date

## we can even use this to create new variables

covid19_df <- covid19_date |>
  mutate(Month = lubridate::month(Date, label = TRUE, abbr = FALSE)) |>
  relocate(Month, .after = Date) |>
  arrange(desc(Total))

covid19_df
```

## Readr function: write_csv

the write_csv function allows us to export data frames to a csv file once we are done cleaning it up or when we have done some analysis that we want to export

```r
# now that we have this date frame cleaned let's save it

covid19_df <- covid19_month

covid19_df

# let's export the file

write_csv(covid19_df, "covid19_time_series.csv")
```

## Haven function: Export as proprietary file

we can even export files that we have been working on as proprietary files to work on in SPSS, SAS, or STATA

```r
# export to SPSS

haven::write_sav(covid19_df, "covid19_time_series.sav")
```

## Part 4: Explore Your Data

- Psych Package

```r
install.packages("psych")
library(psych)
```

```r
describe(reviews_df)
```

- SummaryTools Package

```r
install.packages("summarytools")
library(summarytools)
```

```r
descr(reviews_df)

freq(reviews_df$Division_Name)

ctable(reviews_df$Recommended_IND, reviews_df$Division_Name)
```

- DataExplorer package

```r
install.packages("DataExplorer")
library(DataExplorer)
```

```r
create_report(reviews_df)
```