Data Cleaning with R and Tidyverse

Matt Steele

Resources

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

Recap

- Functions and Arguments
- Objects
- Run Code

Working Directory

The working directory in R is the folder where you are working. Hence, it's the place (the environment) where you have to store your files of your project in order to load them or where your R objects will be saved.

> *Session > Set Working Directory > Choose Directory*

```
getwd() # show current directory that you are in


setwd("path/to/your/directory") # sets the working directory
```

Packages

R packages are like toolkits or collections of pre-built functions, data sets, and tools that extend the capabilities of the R programming language.

Tidyverse

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

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

```
library(tidyverse)
```

Tidyverse

tibble	lighter and more user-friendly version of data f
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 vari
haven	read and write data formats from proprietary s

Loading Data with Readr

The **read_csv** function allows you to load data into R in a tibble data frame

> read_csv(“data_set.csv”)

```
1  sw_df <- read_csv("starwars.csv")
```

Loading Proprietary Data

- readXl - this package allows you to read Excel files in a tibble data frame
- haven - this package allows you to read and export non-proprietary files for SPSS, SAS, and STATA

The Pipe Operator (|>)

The pipe operator (|> or %>%) allows you to run commands or operation on a single object based on an order of operations

	PC	MAC
Pipe Operator	CTRL + SHIFT + M	CMD + SHIFT + M

The Pipe Operator (|>)

let's say you want to see the **name**, **height**, **mass**, and **species** of characters who were born on **Tatooine** and then relocate the **mass** variable

```
view(sw_df)

sw_df |> # object we are working on
  filter(homeworld == "Tatooine") |> # first operation
  select(name, height, mass, species) |>  # second operation
  relocate(mass, .before = name) # third operation
```

Order of Operations

With the pipe operator, your code becomes more organized and reads like a step-by-step process:

1. Take the data …
2. Do this
3. Then do this…
4. And finally, do this.

```
# order of operations matter

sw_df |> # object we are working on
  select(name, height, mass, species) |> # first operation
  filter(homeworld == "Tatooine") |>  # second operation
  relocate(mass, .before = name) #third operation
```

Explore Data - Tidyverse

view function: interactively explore the contents of a data frame in a separate viewer window or in the RStudio viewer pane.

```
1 view(sw_df)
```

glimpse function: a concise overview of the data, including variable types.

```
1 glimpse(sw_df)
```

Cleaning Data

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 obser
group_by	group observations
summarise	get descriptive statistics about a variable
relocate	change the position of variables in the data frame
rename	change the name of an individual variable
drop_na	remove ALL missing values from a data frame or variable
replace_na	replace missing values with a specified

Dplyr function: filter

allows you to *select rows* in your data frame that meet specific conditions or criteria in a variable

```
sw_df

# let's filter the data frame so we are seeing characters who have blue eye

sw_eye <- sw_df |>
  filter(eye_color == "blue")

sw_eye
```

Boolean operators

boolean operators allow you to build criteria in your code

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

Filter with Boolean

let's filter the data frame for characters who have blue eyes and were born after 50 BBY

```
sw_eye50 <- sw_df |>
  filter(eye_color == "blue" & birth_year < 50)

sw_eye50
```

Dplyr function: select

allows you to *keep* or *discard* variables

```r
# keep variables

sw_select <- sw_df |>
  select(name, height, mass)

sw_select

# remove variables

sw_not_select <- sw_df |>
  select(-height, -mass)

sw_not_select
```

Dplyr function: mutate

creates new variables in your data or *change* existing variables by performing calculations or transformations.

```
sw_df

sw_df <- sw_df |>
  mutate(bmi = height/mass) |>  # run mutate operation
  relocate(bmi, .after = mass) # relocate variable in data frame

sw_df
```

Dplyr function: mutate

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

```
# let's overwrite the old variable

sw_overwrite <- sw_df |>
  mutate(height = height/12) # overwrite variable

sw_overwrite
```

Dplyr function: arrange

allows you to *sort* variables

```
# oldest characters

sw_df |>
  arrange(desc(birth_year))

# characters with the same skin color than the same hair color

sw_df |>
  arrange(desc(skin_color), hair_color)
```

Dplyr function: group_by & summarise

the **group_by** function allows you to group common observations in a variable

summarise function allows you to get descriptive statistics about the groupings

```
sw_group <- sw_df |>
  group_by(sex) |>
  summarise(avg_height = mean(height)) # get mean

sw_group

write_csv(sw_group, "sw_sex_dv.csv") # export
```

Recode Data Values

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

```
sw_df <- sw_df |>
  mutate(character_id = as.character(character_id))

sw_df
```

Dyplr Function: recode observations

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

```
# let's create a new variable with to give numeric levels instead of value

sw_df <- sw_df |>
  mutate(gender_num = recode(gender,
                             "masculine" = 0,
                             "feminine" = 1)) |>
  relocate(gender_num, .after = gender)


view(sw_df)
```

Dplyr function: rename

allows you rename variables in your data frame

```
glimpse(sw_df)

sw_df <- sw_df |>
  rename("gender_label" = gender)

glimpse(sw_df)
```

Missing Data

missing data in numeric fields can cause an issue when trying to calculate descriptive statistics

```
# are there missing NA values

which(is.na(sw_df$bmi))

# with missing values we cannot calculate descriptive statistics

mean(sw_df$bmi)
```

Tidyr function: drop_na

removes **all** missing data from data frames or variables

```
sw_dropNA <- sw_df |>
  drop_na()

mean(sw_dropNA$bmi)
```

we can also just drop NAs from a **variable**

```
sw_dropNA_var <- sw_df |>
  drop_na(bmi)

mean(sw_dropNA_var$bmi)
```

Tidyr function: replace_na

you can also recode the NA values for observations with *mutate* and *replace_na*

```
# let's replace the NAs the gender_label variable with "unknown"


sw_df <- sw_df |>
  mutate(gender_label = replace_na(gender_label, "unknown")) |>
  mutate(gender_label = as_factor(gender_label))

levels(sw_df$gender_label)
```

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

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

# let's export the file

write_csv(sw_df, "starwars_clean.csv")
```

Psych Package

Psych Package - built-in functions for factor analysis, reliability analysis, descriptive statistics and data visualization.

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

```
sw_ds <- describe(sw_df)

write_csv(sw_ds, "starwars_ds.csv")
```

Summarytools Package

SummaryTools Package - simplifies data exploration and descriptive statistics generation for data frames and vectors.

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

```
descr(sw_df)

freq(sw_df$sex)

ctable(sw_df$sex, sw_df$gender)
```

DataExplorer

DataExplorer package - automates and streamlines the process of exploring and visualizing datasets.

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

```
create_report(sw_df)
```