Data Cleaning and Analysis with R and RStudio

Yiheng Wu, Jeff Werst, Matt Steele

Resources

- Tidyverse Documentation

```
setwd("path_to_folder")
```

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.

Table 1: Packages Included 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 packages

---

Part 2: Loading Data with Tidyverse

Readr function: read_csv

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

```
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 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 **name**, **height**, **mass**, and **species** of characters who were born on **Tatooine**

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

# order of operations matter

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

# why did this not work?
```

Tibble function: view

view the contents of a data frame in a separate viewer window or in the RStudio viewer pane.

```
view(sw_df)
```

Tibble function: glimpse

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

```r
glimpse(sw_df)
```

Part 3: Cleaning Data

Table 2: 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
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

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

```r
sw_df

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

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

sw_eye
```

Boolean operators

boolean operators allows you to build criteria in your code

Table 3: 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 characters who
  # do have blue eyes
    # AND were born after 50 BBY

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

sw_eye50

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

sw_eyeor50 <- sw_df |>
  filter(eye_color == "blue" | birth_year < 50)

sw_eyeor50
```

Dplyr function: select

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

```
# 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

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

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*

```r
# create a new variable

sw_df

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

# let's overwrite the old variable

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

sw_overwrite
```

Dplyr function: arrange

the **arrange** function allows you to *sort* variables

```r
# 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 and **summarise** function allows you to get descriptive statistics about the groupings

```r
sw_group <- sw_df |>
  group_by(sex) |>
  summarise(avg_height = mean(height)) # get the mean of height for each group

sw_group

# you can add as many summaries as you want

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

```r
sw_group

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

Base Function: as.character

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*

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

sw_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

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

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

sw_df
```

Dplyr function: rename

the **rename** function allows you rename variables in your data frame

```r
glimpse(sw_df)

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

glimpse(sw_df)
```

Tidyr function: drop_na

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

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

which(is.na(sw_df$bmi))

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

mean(sw_df$bmi)

# we can drop all NA values from the data frame

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 the **replace_na** function

```
# 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
```

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