Social Sciences Intro to Statistics

Week 1.2 Basics of R

Week 1: Learning goal - Understand what and how to access R and R studio.

Introduction

Libraries we will use today

"Load" the package we will use today (output omitted)

- **you must run this code chunk**

```r
library(tidyverse)
```

If package not yet installed, then must install before you load. Install in "console" rather than .Rmd file

- Generic syntax: `install.packages("package_name")`
- Install "tidyverse": `install.packages("tidyverse")`

Note: when we load package, name of package is not in quotes; but when we install package, name of package is in quotes:

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

tidyverse

The package we just downloaded, tidyverse, is a programming package in R that helps us transform data. This package is important for data mutation and visualization.

Investigating data patterns

Introduction to the `dplyr` library

`dplyr`, a package within the `tidyverse` suite of packages, provide tools for manipulating data frames

- Wickham describes functions within `dplyr` as a set of "verbs" that fall in the broader categories of **subsetting**, **sorting**, and **transforming**

-select() extracts columns and returns a tibble.

-arrange() changes the ordering of the rows.

-filter() picks cases based on their values.

-mutate() adds new variables that are functions of existing variables.

-rename() easily changes the name of a column(s).

-pull() extracts a single column as a vector.

Today	Upcoming weeks
Subsetting data - `select()` variables - `filter()` observations - `mutate()` creates new variables	**Transforming data** - `summarize()` calculates across rows - `group_by()` to calculate across rows within groups

- `pull()` variables **Sorting data** \|
- `arrange()` \| **Transforming data**
- `rename`() variables \|

All `dplyr` verbs (i.e., functions) work as follows

1. first argument is a data frame
2. subsequent arguments describe what to do with variables and observations in data frame
    - refer to variable names without quotes
3. result of the function is a new data frame

Data for lecture sections on select(), arrange(), filter(), mutate(), rename(), and pull() functions

Lecture overview

- Introduction to Netflix data on IMDb score and votes
- Brief review of statistics (selected concepts)

Libraries we will use

```r
#install.packages('tidyverse') # if you haven't installed already
#install.packages('labelled') # if you haven't installed already

library(tidyverse) # load tidyverse package
library(labelled) # load labelled package package
```

Netflix Data

The Netflix Data is a dataset created on *Data World* that compiled the best shows and movies on Netflix (as of May 2022).

- We will use data from the *Netflix Data* in lecture and potentially for some assignments
- Relevant links
  - Data documentation
  - Data download

In the following sub-sections, we "load" the data, create modified datasets, investigate the data, and run some basic descriptive statistics

- **Your are not responsible for knowing the below code**
  - You will only be responsible for knowing code that we explicitly teach you during the quarter
- But try to follow the general logic of what the code is doing
- And try running the below "code chunks" on your own computer

Data for lecture sections on pipes and mutate() function

Load .csv data frame `netflix_data`

```r
#load netflix data
netflix_data <- read_csv("https://raw.githubusercontent.com/bcl96/Social-Sciences-Stats/main/

#print(netflix_data)
```

Object `netflix_data`

- Collection of all movies that have at least an IMDb score of 6.9 and at least 10,000 votes

Observations on `netflix_data`

- each observation represents a movie or tv series

```r
typeof(netflix_data)
#> [1] "list"
class(netflix_data)
#> [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
dim(netflix_data)
#> [1] 246  22
```

Variables on `netflix_data`

- some vars provide details about the movie or tv show
  - e.g., `TITLE`, `MAIN_GENRE`, `MAIN_PRODUCTION`
- some vars provide data about the IMDb votes and scores
  - e.g., `SCORE`, `NUMBER_OF_VOTES`
- some vars provide data about year the media was released
  - e.g., `RELEASE_YEAR` is year the movie or tv show was first released

```r
names(netflix_data)
str(netflix_data)
glimpse(netflix_data) # tidyverse function, similar to str()
```

Variable `MAIN_PRODUCTION` identifies where the movie or film was produced

Imagine we want to isolate all the U.S. productions

1. Investigate variable type/structure.

- A dichotomous var, but stored as character in `netflix_data`. So must use quotes (`''` or `""`) to filter/subset based on values of `MAIN_PRODUCTION`

```
str(netflix_data$MAIN_PRODUCTION)
#>  chr [1:246] "US" "US" "US" "GB" "CA" "JP" "KR" "US" "US" "JP" "CA" "JP" ...
```

2. Create frequency table to identify possible values of `MAIN_PRODUCTION`

```
table(netflix_data$MAIN_PRODUCTION, useNA = "always")
#> 
#>   AU   BE   BR   CA   DE   DK   ES   FI   FR   GB   IL   IN   IT   JP   KR   NO 
#>    1    2    1   13    5    2    4    1    5   27    1    3    1   26    9    4 
#>   SE   TR   US <NA> 
#>    3    4  134    0
```

3. Isolate all the US production (output omitted)

```
filter(netflix_data, MAIN_PRODUCTION == "US")
```

select() variables

Select variables using `select()` function

Printing observations is key to investigating data, but datasets often have hundreds, thousands of variables

`select()` function selects **columns** of data (i.e., variables) you specify

- first argument is the name of data frame object
- remaining arguments are variable names, which are separated by commas and without quotes

Without **assignment** (`<-`), `select()` by itself simply prints selected vars

```
#?select
#duration keep it, to see who wants to watch longer stuff
#select but use minus to drop season and release year, to show it yields the same as just pi
names(netflix_data)
#> [1] "TITLE"             "RELEASE_YEAR"      "SCORE"            
#> [4] "NUMBER_OF_VOTES"   "DURATION"          "NUMBER_OF_SEASONS"
#> [7] "MAIN_GENRE"        "MAIN_PRODUCTION"   "contentType"
#> [10] "description"       "contentRating"     "genre"
#> [13] "formattedDuration" "releasedDate"      "Hours Viewed"
#> [16] "actors"            "director"          "creator"
#> [19] "audio"             "subtitle"          "numberOfSeasons"
#> [22] "seasonStartDate"
select(netflix_data, TITLE, SCORE, NUMBER_OF_VOTES, DURATION, MAIN_GENRE, MAIN_PRODUCTION)
#> # A tibble: 246 x 6
#>    TITLE               SCORE NUMBER_OF_VOTES DURATION MAIN_GENRE MAIN_PRODUCTION
#>    <chr>               <dbl>           <dbl>    <dbl> <chr>      <chr>
#>  1 13 Reasons Why        7.5          282373       58 drama      US
#>  2 30 Rock               8.3          121514       25 comedy     US
#>  3 A Series of Unfort~   7.8           59239       47 action     US
#>  4 After Life            8.5          124972       28 comedy     GB
#>  5 Alias Grace           7.7           31577       44 drama      CA
#>  6 Alice in Borderland   7.6           47651       47 action     JP
#>  7 All of Us Are Dead    7.5           41393       61 action     KR
#>  8 Altered Carbon        7.9          162018       52 scifi      US
#>  9 American Vandal       8.1           29972       33 comedy     US
#> 10 Angel Beats!          7.7           13848       26 scifi      JP
#> # i 236 more rows
```

Select variables using `select()` function

Recall that all `dplyr` functions (e.g., `select()`) return a new data frame object

- **type** equals "list"
- **length** equals number of vars you select

```
typeof(select(netflix_data, TITLE, SCORE, NUMBER_OF_VOTES, MAIN_GENRE, MAIN_PRODUCTION))
#> [1] "list"
length(select(netflix_data, TITLE, SCORE, NUMBER_OF_VOTES, MAIN_GENRE, MAIN_PRODUCTION))
#> [1] 5
```

`glimpse()`: tidyverse function for viewing data frames

- a cross between `str()` and simply printing data

```
?glimpse
glimpse(netflix_data)
```

`glimpse()` a `select()` set of variables

```
glimpse(select(netflix_data, TITLE, SCORE, NUMBER_OF_VOTES, MAIN_GENRE, MAIN_PRODUCTION))
#> Rows: 246
#> Columns: 5
#> $ TITLE           <chr> "13 Reasons Why", "30 Rock", "A Series of Unfortunate ~
#> $ SCORE           <dbl> 7.5, 8.3, 7.8, 8.5, 7.7, 7.6, 7.5, 7.9, 8.1, 7.7, 8.7,~
#> $ NUMBER_OF_VOTES <dbl> 282373, 121514, 59239, 124972, 31577, 47651, 41393, 16~
#> $ MAIN_GENRE      <chr> "drama", "comedy", "action", "comedy", "drama", "actio~
#> $ MAIN_PRODUCTION <chr> "US", "US", "US", "GB", "CA", "JP", "KR", "US", "US", ~
```

Select variables using `select()` function

With **assignment** (`<-`), `select()` creates a new object containing only the variables you specify

```
netflix_small <- select(netflix_data, TITLE, SCORE, NUMBER_OF_VOTES, MAIN_GENRE, MAIN_PRODUC

glimpse(netflix_small)
#> Rows: 246
#> Columns: 5
#> $ TITLE           <chr> "13 Reasons Why", "30 Rock", "A Series of Unfortunate ~
#> $ SCORE           <dbl> 7.5, 8.3, 7.8, 8.5, 7.7, 7.6, 7.5, 7.9, 8.1, 7.7, 8.7,~
#> $ NUMBER_OF_VOTES <dbl> 282373, 121514, 59239, 124972, 31577, 47651, 41393, 16~
#> $ MAIN_GENRE      <chr> "drama", "comedy", "action", "comedy", "drama", "actio~
#> $ MAIN_PRODUCTION <chr> "US", "US", "US", "GB", "CA", "JP", "KR", "US", "US", ~
```

Select

`select()` can use "helper functions" `starts_with()`, `contains()`, and `ends_with()` to choose columns

Example:

```
names(netflix_data)
#>  [1] "TITLE"             "RELEASE_YEAR"      "SCORE"
#>  [4] "NUMBER_OF_VOTES"   "DURATION"          "NUMBER_OF_SEASONS"
#>  [7] "MAIN_GENRE"        "MAIN_PRODUCTION"   "contentType"
#> [10] "description"       "contentRating"     "genre"
#> [13] "formattedDuration" "releasedDate"      "Hours Viewed"
#> [16] "actors"            "director"          "creator"
#> [19] "audio"             "subtitle"          "numberOfSeasons"
#> [22] "seasonStartDate"
```

```
select(netflix_data, starts_with("MAIN"))
#> # A tibble: 246 x 2
#>    MAIN_GENRE MAIN_PRODUCTION
#>    <chr>      <chr>
#>  1 drama      US
#>  2 comedy     US
#>  3 action     US
#>  4 comedy     GB
#>  5 drama      CA
#>  6 action     JP
#>  7 action     KR
#>  8 scifi      US
#>  9 comedy     US
#> 10 scifi      JP
#> # i 236 more rows
select(netflix_data, contains("OF"))
#> # A tibble: 246 x 3
#>    NUMBER_OF_VOTES NUMBER_OF_SEASONS numberOfSeasons
#>              <dbl>             <dbl>           <dbl>
#>  1          282373                 4              NA
#>  2          121514                 7              NA
#>  3           59239                 3              NA
#>  4          124972                 3               3
#>  5           31577                 1              NA
#>  6           47651                 2              NA
#>  7           41393                 1               1
#>  8          162018                 2               2
#>  9           29972                 2              NA
#> 10           13848                 1              NA
#> # i 236 more rows
select(netflix_data, ends_with("RE"))
#> # A tibble: 246 x 3
#>    SCORE MAIN_GENRE genre
#>    <dbl> <chr>      <chr>
#>  1   7.5 drama      <NA>
#>  2   8.3 comedy     <NA>
#>  3   7.8 action     <NA>
#>  4   8.5 comedy     TV Dramas
#>  5   7.7 drama      <NA>
#>  6   7.6 action     <NA>
#>  7   7.5 action     Horror TV Serials
#>  8   7.9 scifi      TV Shows Based on Books
#>  9   8.1 comedy     <NA>
#> 10   7.7 scifi      <NA>
#> # i 236 more rows
```

1. Use `select()` to familiarize yourself with variables in the data frame
2. Practice using the `contains()` and `ends_with()` helper functions to to choose variables

Arrange()

`arrange()` function can change the ordering of the rows (i.e., sort rows), it "arranges" rows in a data frame by sortsing the observations

Syntax: `arrange(x,...)`

- First argument, `x`, is a data frame
- Subsequent arguments are a "comma separated list of unquoted variable names"

```
netflix_data
arrange(netflix_data, SCORE)
```

Data frame goes back to previous order unless you **assign** the new order

```
netflix_data
netflix_data <- arrange(netflix_data, SCORE)
netflix_data
```

Ascending and descending order

- `arrange()` sorts in **ascending** order by default
- use `desc()` to sort a column by descending order

```
arrange(netflix_data, desc(SCORE))
```

Can sort by multiple variables

```
arrange(netflix_data, desc(SCORE), desc(RELEASE_YEAR), TITLE)

#sort by descending by IMDb score and descending by release year and by title; combine with s

select(arrange(netflix_data, desc(SCORE), desc(RELEASE_YEAR), TITLE),
       TITLE, RELEASE_YEAR, SCORE, NUMBER_OF_VOTES, NUMBER_OF_SEASONS, contentType)
```

`arrange()`, missing values sorted at the end

Missing values automatically sorted at the end, regardless of whether you sort ascending or descending

Below, we sort by score, then by release year, then by title

```
#by descending score, descending release year, title
select(arrange(netflix_data, desc(SCORE), desc(RELEASE_YEAR), TITLE),
       TITLE, RELEASE_YEAR, SCORE, NUMBER_OF_VOTES, NUMBER_OF_SEASONS, contentType)

#by score, release year, descending title
select(arrange(netflix_data, SCORE, RELEASE_YEAR, desc(TITLE)),
       TITLE, RELEASE_YEAR, SCORE, NUMBER_OF_VOTES, NUMBER_OF_SEASONS, contentType)
```

Can sort by `is.na` to put missing values first

```
select(arrange(netflix_data, desc(is.na(contentType)), desc(SCORE), desc(RELEASE_YEAR), TITL
       TITLE, RELEASE_YEAR, SCORE, NUMBER_OF_VOTES, NUMBER_OF_SEASONS, contentType)
#> # A tibble: 246 x 6
#>    TITLE        RELEASE_YEAR SCORE NUMBER_OF_VOTES NUMBER_OF_SEASONS contentType
#>    <chr>               <dbl> <dbl>           <dbl>             <dbl> <chr>
#>  1 Breaking Bad         2008   9.5         1727694                 5 <NA>
#>  2 Kota Factory         2019   9.3           66985                 2 <NA>
#>  3 Our Planet           2019   9.3           41386                 1 <NA>
#>  4 Avatar: The~         2005   9.3          297336                 3 <NA>
#>  5 The Last Da~         2020   9.1          108321                 1 <NA>
#>  6 Attack on T~         2013   9            325381                 4 <NA>
#>  7 Hunter x Hu~         2011   9             87857                 3 <NA>
#>  8 DEATH NOTE           2006   9            302147                 1 <NA>
#>  9 Heartstopper         2022   8.9           28978                 1 <NA>
#> 10 When They S~         2019   8.9          114127                 1 <NA>
#> # i 236 more rows
```

Rename variables

`rename()` function renames variables within a data frame object

Syntax:

- `rename(obj_name, new_name = old_name,...)`

```r
rename(netflix_data, CONTENT_TYPE = contentType, DESCRIPTION = description, CONTENT_RATING =

names(netflix_data)
```

Variable names do not change permanently unless we combine rename with assignment

```r
rename_netflix_data <- rename(netflix_data, CONTENT_TYPE = contentType, DESCRIPTION = descri

names(rename_netflix_data)
rm(rename_netflix_data)
```

filter() rows

The `filter()` function

`filter()` allows you to **select observations** based on values of variables

- Arguments
  - first argument is name of data frame
  - subsequent arguments are *logical expressions* to filter the data frame
  - Multiple expressions separated by commas work as **AND** operators (e.g., condtion 1 `TRUE` AND condition 2 `TRUE`)
- What is the result of a `filter()` command?
  - `filter()` returns a data frame consisting of rows where the condition is `TRUE`

```r
?filter
```

Example from data frame object `netflix_data`, each obs is a show or film

- Show all obs where the show had a IMDb score of 8.0 [output omitted]

```r
filter(netflix_data, SCORE == 8.0)
filter(netflix_data, SCORE == 8) #same
```

Note that resulting object is list, consisting of obs where condition `TRUE`

```r
nrow(netflix_data)
#> [1] 246
nrow(filter(netflix_data, SCORE == 8))
#> [1] 14
```

The `filter()` function, base R equivalents

Task: Count the number of shows that had an IMDb score of 8.

tidyverse Using `filter()`:

```r
nrow(filter(netflix_data, SCORE == 8))
#> [1] 14
```

[**base R**] Using `[]` and `$`:

```r
nrow(netflix_data[netflix_data$SCORE == 8, ])
#> [1] 14
```

[**base R**] Using `subset()`:

```r
nrow(subset(netflix_data, SCORE == 8))
#> [1] 14
```

Filter, character variables

Use single quotes `''` or double quotes `""` to refer to values of character variables

```r
glimpse(select(netflix_data, SCORE, MAIN_GENRE))
#> Rows: 246
#> Columns: 2
#> $ SCORE      <dbl> 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5,~
#> $ MAIN_GENRE <chr> "drama", "action", "action", "scifi", "scifi", "comedy", "w~
```

Identify all shows that had an IMDb score of 8 and 2 seasons

- Shows that had an IMBd score of 8

```r
filter(netflix_data, SCORE == 8)
```

- Shows that had 2 seasons

```r
filter(netflix_data, NUMBER_OF_SEASONS == 2)
```

- Shows that had an IMDb score of 8 and two seasons

```r
filter(netflix_data, SCORE == 8, NUMBER_OF_SEASONS == 2)
```

Filter by multiple conditions, base R equivalents

Task: Count the number of shows that had an IMDb score of 8, had 2 seasons, and was a drama.

tidyverse Using `filter()`:

```r
nrow(filter(netflix_data, SCORE == 8, NUMBER_OF_SEASONS == 2,
       MAIN_GENRE == "drama"))
#> [1] 2
```

[**base R**] Using `[]` and `$`:

```r
nrow(netflix_data[netflix_data$SCORE == 8 &
                    netflix_data$NUMBER_OF_SEASONS == 2 &
                    netflix_data$MAIN_GENRE == "drama", ])
#> [1] 2
```

[**base R**] Using `subset()`:

```r
nrow(subset(netflix_data, SCORE == 8 & NUMBER_OF_SEASONS == 2 &
              MAIN_GENRE == "drama"))
#> [1] 2
```

Logical operators for comparisons

logical operators useful for: filter obs w/ `filter()`; create variables w/ `mutate()`

- logical operators also work when using Base R functions

Operator symbol	Operator meaning
`==`	Equal to
`!=`	Not equal to
`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to
`&`	AND
`|`	OR
`%in%`	includes

- Visualization of "Boolean" operators (e.g., AND, OR, AND NOT)

![“Boolean” operations, x=left circle, y=right circle, from Wichkam (2018)]

Aside: `count()` function

`count()` function from `dplyr` package counts the number of obs by group

Syntax [see help file for full syntax]

- `count(x,...)`

Arguments [see help file for full arguments]

- `x`: an object, often a data frame
- `...`: variables to group by

Examples of using `count()`

- Without vars in `...` argument, counts number of obs in object

```
count(netflix_data)
  # netflix_data %>% count() # same as above but using pipes
str(count(netflix_data))
  # netflix_data %>% count() %>% str() # same as above but using pipes
```

- With vars in `...` argument, counts number of obs per variable value
  - This is the best way to create frequency table, better than `table()`
  - note: by default, `count()` always shows `NAs` [this is good!]

```
count(netflix_data, MAIN_GENRE)
  # netflix_data %>% count(MAIN_GENRE) # same as above but using pipes
str(count(netflix_data, MAIN_GENRE))
  # netflix_data %>% count(MAIN_GENRE) %>% str() # same as above but using pipes
```

Filters and comparisons, Demonstration

Shows that had an IMDb score of 8 and/or 2 seasons

```
# a score of 8 AND  2 seasons

filter(netflix_data, SCORE == 8, NUMBER_OF_SEASONS == 2)
filter(netflix_data, SCORE == 8 & NUMBER_OF_SEASONS == 2) #same

netflix_data[netflix_data$SCORE == 8 &
            netflix_data$NUMBER_OF_SEASONS == 2, ] # using [] and $

subset(netflix_data, SCORE == 8 & NUMBER_OF_SEASONS == 2) # using subset()

nrow(subset(netflix_data, SCORE == 8 & NUMBER_OF_SEASONS == 2))
```

```
#  a score of 8 OR  2 seasons
filter(netflix_data, SCORE == 8 | NUMBER_OF_SEASONS == 2)

netflix_data[netflix_data$SCORE == 8 |
            netflix_data$NUMBER_OF_SEASONS == 2, ] # using [] and $

subset(netflix_data, SCORE == 8 |
         NUMBER_OF_SEASONS == 2) # using subset()
```

Filters and comparisons, Demonstration (cont.)

Apply `count()` function on top of `filter()` function to count the number of observations that satisfy criteria

- Avoids printing individual observations

```
# Number of shows that get an IMDb score of 8 AND 2 seasons
count(filter(netflix_data, SCORE == 8, NUMBER_OF_SEASONS == 2))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1     5

# Number of shows that get an IMDb score of 8 OR 2 seasons
count(filter(netflix_data, SCORE == 8 | NUMBER_OF_SEASONS == 2))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    49
```

- Note: You could also use any of the base R equivalents from the previous slide

Filters and comparisons, >=

Number of action shows that have at least two seasons compared to number of shows that have an IMDb score of 8.

```
# at least have two seasons
count(filter(netflix_data, MAIN_GENRE == "action", NUMBER_OF_SEASONS >= 2))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    22

# at least have two seasons and an IMDb score of 8
count(filter(netflix_data, MAIN_GENRE == "action", NUMBER_OF_SEASONS >= 2,
             SCORE == 8))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1     2
```

Filters and comparisons, >= (cont.)

Number of action shows that have at least an IMDb score of 8 compared to number of shows that have 2 seasons:

```
# at least have an IMDb score of 8
count(filter(netflix_data, MAIN_GENRE == "action", SCORE >= 8))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    14

# at least have an IMDb score of 8 and two seasons
count(filter(netflix_data, MAIN_GENRE == "action", SCORE >= 8,
             NUMBER_OF_SEASONS == 2))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1     5
```

Filters and comparisons, not equals (`!=`)

Count the number of shows that have two seasons that are not action.

```
#number of shows with two seasons
count(filter(netflix_data, NUMBER_OF_SEASONS == 2))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    40

#number of shows with two seasons that are not action
count(filter(netflix_data, NUMBER_OF_SEASONS == 2, MAIN_GENRE != "action"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    32

#number of shows with two seasons that are action
#count(filter(netflix_data, NUMBER_OF_SEASONS == 2, MAIN_GENRE == "action"))
```

Filters and comparisons, `%in%` operator

What if you wanted to count the number of shows with an IMDb score of 8 in a group of genres?

```
count(filter(netflix_data, SCORE == 8, MAIN_GENRE == "action" |
               MAIN_GENRE == "drama" | MAIN_GENRE == "comedy"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    11
```

Easier way to do this is with `%in%` operator

```
count(filter(netflix_data, SCORE == 8, MAIN_GENRE %in% c("action","drama","comedy")))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    11
```

Select the shows that are drama with either an IMDb score of 8 or 8.1

```
count(filter(netflix_data, SCORE %in% 8:8.1, MAIN_GENRE == "drama"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1     6
```

Identifying data type and possible values helpful for filtering

- `typeof()` and `str()` shows internal data type of a variable
- `table()` to show potential values of categorical variables

```
typeof(netflix_data$MAIN_GENRE)
#> [1] "character"
str(netflix_data$MAIN_GENRE) # double quotes indicate character
#>  chr [1:246] "drama" "action" "action" "scifi" "scifi" "comedy" "war" ...
table(netflix_data$MAIN_GENRE, useNA="always")
#> 
#>      action   animation      comedy       crime documentary       drama 
#>          28           4          43          20           7          82 
#>     reality     romance       scifi    thriller         war     western 
#>           2           1          45           4           8           2 
#>        <NA> 
#>           0
```

```
typeof(netflix_data$MAIN_PRODUCTION)
#> [1] "character"
str(netflix_data$MAIN_PRODUCTION)
#>  chr [1:246] "US" "KR" "US" "JP" "CA" "US" "US" "DE" "US" "US" "GB" "US" ...
```

Now that we know `MAIN_GENRE` is a character, we can filter values

```
count(filter(netflix_data, MAIN_GENRE == "drama", MAIN_PRODUCTION == "US"))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    43
#below code would return an error because variables are character
#count(filter(netflix_data, MAIN_GENRE == drama, MAIN_PRODUCTION == US))
```

Filtering and missing values

Wickham (2018) states:

- "`filter()` only includes rows where condition is TRUE; it excludes both `FALSE` and `NA` values. To preserve missing values, ask for them explicitly:"

Investigate var `netflix_data$NUMBER_OF_VOTES`, number of votes the show received

- only shows produced in the U.S.

```
#shows produced in the U.S. with less than 50,000 votes
count(filter(netflix_data, MAIN_PRODUCTION == "US", NUMBER_OF_VOTES<50000))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    57
#shows produced in the U.S. with the number of votes is missing
count(filter(netflix_data, MAIN_PRODUCTION == "US", is.na(NUMBER_OF_VOTES)))
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1     0
#shows produced in the U.S. with less than 50,000 votes OR votes is missing
count(filter(netflix_data, MAIN_PRODUCTION == "US", NUMBER_OF_VOTES<50000 | is.na(NUMBER_OF_V
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1    57
```

Introduce `mutate()` function

`mutate()` is **tidyverse** approach to creating variables (not **Base R** approach)

Description of `mutate()`

- creates new columns (variables) that are functions of existing columns
- After creating a new variable using `mutate()`, every row of data is retained
- `mutate()` works best with pipes `%>%`

Task:

- Using data frame `school_v2` create new variable that measures the pct of students on free/reduced lunch (output omitted)

```
# create new dataset with fewer vars; not necessary to do this
netflix_new <- netflix_data %>%  select(TITLE, SCORE, MAIN_PRODUCTION, NUMBER_OF_VOTES)

# create new var
netflix_new %>% mutate(total_score = SCORE*NUMBER_OF_VOTES)

# remove data frame object
rm(netflix_new)
```

Investigate `mutate()` syntax

Usage (i.e., syntax)

- `mutate(.data,...)`

Arguments

- `.data`: a data frame
  - if using `mutate()` after pipe operator `%>%`, then this argument can be omitted
    - Why? Because data frame object to left of `%>%` "piped in" to first argument of `mutate()`
- `...`: expressions used to create new variables

    – “Name-value pairs of expressions”
    – “The name of each argument will be the name of a new variable, and the value will be its corresponding value.”
    – “Use a `NULL` value in mutate to drop a variable.”
    – “New variables overwrite existing variables of the same name”

Value

- returns a (data frame) object that contains the original input data frame and new variables that were created by `mutate()`

Investigate `mutate()` syntax

Can create variables using standard mathematical or logical operators [output omitted]

```
glimpse(netflix_data)
#> Rows: 246
#> Columns: 22
#> $ TITLE             <chr> "13 Reasons Why", "All of Us Are Dead", "Arrow", "Bl~
#> $ RELEASE_YEAR      <dbl> 2017, 2022, 2012, 2011, 2015, 2020, 2016, 2018, 2018~
#> $ SCORE             <dbl> 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.5, 7.~
#> $ NUMBER_OF_VOTES   <dbl> 282373, 41393, 425716, 12741, 41867, 16978, 88019, 1~
#> $ DURATION          <dbl> 58, 61, 42, 24, 43, 25, 44, 60, 24, 48, 25, 18, 47, ~
#> $ NUMBER_OF_SEASONS <dbl> 4, 1, 8, 2, 3, 1, 3, 1, 1, 1, 2, 3, 1, 1, 1, 2, 3, 1~
#> $ MAIN_GENRE        <chr> "drama", "action", "action", "scifi", "scifi", "come~
#> $ MAIN_PRODUCTION   <chr> "US", "KR", "US", "JP", "CA", "US", "US", "DE", "US"~
#> $ contentType       <chr> NA, "TVSeries", NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
#> $ description       <chr> NA, "A high school becomes ground zero for a zombie ~
#> $ contentRating     <dbl> NA, 18, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 18, ~
#> $ genre             <chr> NA, "Horror TV Serials", NA, NA, NA, NA, NA, NA, NA,~
#> $ formattedDuration <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
#> $ releasedDate      <chr> NA, "1/28/22", NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
#> $ `Hours Viewed`    <dbl> NA, 94600000, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
#> $ actors            <chr> NA, "Park Ji-hu, Yoon Chan-young, Cho Yi-hyun, Lomon~
#> $ director          <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, "Dan~
#> $ creator           <chr> NA, "Lee JQ, Chun Sung-il, Kim Nam-su", NA, NA, NA, ~
#> $ audio             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
#> $ subtitle          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
#> $ numberOfSeasons   <dbl> NA, 1, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 1, NA~
#> $ seasonStartDate   <chr> NA, "1/28/22", NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
netflix_data %>%
  select(TITLE, SCORE, MAIN_PRODUCTION, NUMBER_OF_VOTES) %>%
  mutate( # each argument creates a new variable, name of argument is name of variable
    TOTAL_SCORE = SCORE*NUMBER_OF_VOTES,
    RETURN_SCORE = TOTAL_SCORE/NUMBER_OF_VOTES,
    ) %>%
  select(TITLE, SCORE, MAIN_PRODUCTION, NUMBER_OF_VOTES, RETURN_SCORE)
#> # A tibble: 246 x 5
#>    TITLE                SCORE MAIN_PRODUCTION NUMBER_OF_VOTES RETURN_SCORE
#>    <chr>                <dbl> <chr>                     <dbl>        <dbl>
#>  1 13 Reasons Why         7.5 US                       282373          7.5
#>  2 All of Us Are Dead     7.5 KR                        41393          7.5
#>  3 Arrow                  7.5 US                       425716          7.5
#>  4 Blue Exorcist          7.5 JP                        12741          7.5
#>  5 Dark Matter            7.5 CA                        41867          7.5
#>  6 Dash & Lily            7.5 US                        16978          7.5
#>  7 Designated Survivor    7.5 US                        88019          7.5
#>  8 Dogs of Berlin         7.5 DE                        12453          7.5
#>  9 Everything Sucks!      7.5 US                        18023          7.5
#> 10 Evil Genius            7.5 US                        27516          7.5
#> # i 236 more rows
```

Can create variables using "helper functions" called within `mutate()` [output omitted]

- These are standalone functions can be called *within* `mutate()`
    - e.g., `if_else()`, `recode()`, `case_when()`

```
table(netflix_data$MAIN_PRODUCTION, useNA = "always")

netflix_data %>%
  select(TITLE, SCORE, MAIN_PRODUCTION, NUMBER_OF_VOTES) %>%
  mutate(AMERICAN = if_else(MAIN_PRODUCTION == "US", 1, 0))
```

Introduce `mutate()` function

New variable not retained unless we **assign** `<-` it to an object (existing or new)

- **`mutate()` without assignment**

```
netflix_data %>%
  mutate(TOTAL_SCORE = SCORE*NUMBER_OF_VOTES, RETURN_SCORE = TOTAL_SCORE/NUMBER_OF_VOTES)

names(netflix_data)
```

- **`mutate()` with assignment**

```
new_netflix_data <- netflix_data %>%
  select(TITLE, SCORE, MAIN_PRODUCTION, NUMBER_OF_VOTES) %>%
  mutate(TOTAL_SCORE = SCORE*NUMBER_OF_VOTES,
    RETURN_SCORE = TOTAL_SCORE/NUMBER_OF_VOTES)

names(new_netflix_data)
rm(new_netflix_data)
```

Introduce `rename()`‘ function

`rename()` is **tidyverse** approach to rename variables but will keep variables that are not explicitly mentioned

If we want to rename all of our variables to lower case, we can use the help of the `tolower()` function.

```
# Example of what `tolower()` function does
text <- "Hello, World!"
lower_text <- tolower(text)
print(lower_text)
#> [1] "hello, world!"

# Renaming our variables in `netflix_data` to lower case
names(netflix_data) <- tolower(names(netflix_data))

netflix_data
#> # A tibble: 246 x 22
#>    title            release_year score number_of_votes duration number_of_seasons
#>    <chr>                   <dbl> <dbl>           <dbl>    <dbl>             <dbl>
#>  1 13 Reasons Why           2017   7.5          282373       58                 4
#>  2 All of Us Are ~          2022   7.5           41393       61                 1
#>  3 Arrow                    2012   7.5          425716       42                 8
#>  4 Blue Exorcist            2011   7.5           12741       24                 2
#>  5 Dark Matter              2015   7.5           41867       43                 3
#>   6 Dash & Lily           2020   7.5           16978       25                 1
#>   7 Designated Sur~       2016   7.5           88019       44                 3
#>   8 Dogs of Berlin        2018   7.5           12453       60                 1
#>   9 Everything Suc~       2018   7.5           18023       24                 1
#>  10 Evil Genius           2018   7.5           27516       48                 1
#> # i 236 more rows
#> # i 16 more variables: main_genre <chr>, main_production <chr>,
#> #   contenttype <chr>, description <chr>, contentrating <dbl>, genre <chr>,
#> #   formattedduration <lgl>, releaseddate <chr>, `hours viewed` <dbl>,
#> #   actors <chr>, director <chr>, creator <chr>, audio <lgl>, subtitle <lgl>,
#> #   numberofseasons <dbl>, seasonstartdate <chr>
```

Introduce `pull()` function

`pull()` is **tidyverse** approach to extract the specified column from a data frame or tibble and returns it as a vector

```
netflix_data %>% pull(release_year)
#>   [1] 2017 2022 2012 2011 2015 2020 2016 2018 2018 2018 2020 2016 2021 2020 2020
#>  [16] 2020 2021 2021 2020 2019 2020 2013 2016 2013 2014 2019 2021 2020 2016 2017
#>  [31] 2015 2019 2008 2018 2016 1995 2005 2021 2014 2020 2010 2021 2020 2012 2020
#>  [46] 2014 2019 2014 2015 2021 2015 2017 2010 2016 2018 2019 2019 2016 2018 2021
#>  [61] 2015 2011 2016 2018 2021 2009 2018 2017 2019 2015 2021 2017 2019 2014 2011
#>  [76] 2017 2015 2014 2021 2003 2020 2019 2018 2021 2021 2004 2019 2017 2018 2017
#>  [91] 2015 2009 2019 2019 2019 2015 2000 2013 2012 2016 2008 2018 2016 2014 2020
#> [106] 2017 2017 2011 2016 2014 2017 2019 2015 2012 2019 2019 2015 2017 2014 2021
#> [121] 2017 2013 2018 2019 2008 2016 2020 2017 2018 2005 2013 2017 2014 2014 2016
#> [136] 2017 2013 2018 1993 2017 2020 2018 2011 2020 2011 2000 2015 2020 2015 2017
#> [151] 2015 2016 2016 2018 2010 2019 2006 2017 2015 2017 2015 2007 2016 2019 2012
#> [166] 2015 2017 2019 2013 2021 2015 2017 2018 2018 2007 2017 2021 2018 2002 2014
#> [181] 1997 2019 2018 2012 2019 2011 2016 2019 2021 2018 2019 2010 2012 2009 2019
#> [196] 2016 2018 1995 2017 2018 2015 2005 2006 2015 2018 2019 2020 2015 2017 2018
#> [211] 2011 2018 2020 2001 2013 2017 2003 2006 2017 2019 2010 2006 2013 2015 2016
#> [226] 2016 2015 2014 2003 1969 2015 1999 2013 1998 2022 1989 2019 2013 2006 2011
#> [241] 2021 2020 2005 2019 2019 2008
```

- you can also use column indices or names with pull():

```
# Extract by column name
netflix_data %>% pull("score")
#>   [1] 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5
#>  [19] 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6
#>  [37] 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.7 7.7 7.7
#>  [55] 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.8 7.8 7.8 7.8 7.8
#>  [73] 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.9 7.9
#>  [91] 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9
#> [109] 7.9 7.9 7.9 7.9 7.9 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0
#> [127] 8.0 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.2 8.2
#> [145] 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.3 8.3 8.3 8.3 8.3 8.3
#> [163] 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4
#> [181] 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5
#> [199] 8.5 8.5 8.5 8.5 8.5 8.5 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.7
#> [217] 8.7 8.7 8.7 8.7 8.7 8.7 8.7 8.7 8.7 8.7 8.8 8.8 8.8 8.8 8.8 8.8 8.8 8.9
#> [235] 8.9 8.9 8.9 9.0 9.0 9.0 9.1 9.1 9.3 9.3 9.3 9.5

# Extract by column index
netflix_data %>% pull(3)  # This would extract the 3rd column of our data frame, which is "s
#>   [1] 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5
#>  [19] 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.5 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6
#>  [37] 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.6 7.7 7.7 7.7
#>  [55] 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.7 7.8 7.8 7.8 7.8 7.8
#>  [73] 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.8 7.9 7.9
#>  [91] 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9 7.9
#> [109] 7.9 7.9 7.9 7.9 7.9 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0 8.0
#> [127] 8.0 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.1 8.2 8.2
#> [145] 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.2 8.3 8.3 8.3 8.3 8.3 8.3
#> [163] 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.3 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4
#> [181] 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.4 8.5 8.5 8.5 8.5 8.5 8.5 8.5 8.5
#> [199] 8.5 8.5 8.5 8.5 8.5 8.5 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.6 8.7
#> [217] 8.7 8.7 8.7 8.7 8.7 8.7 8.7 8.7 8.7 8.7 8.8 8.8 8.8 8.8 8.8 8.8 8.8 8.9
#> [235] 8.9 8.9 8.9 9.0 9.0 9.0 9.1 9.1 9.3 9.3 9.3 9.5
```