Programming day 1: Data exploration and manipulation

Princeton Sociology Methods Camp

Angela Li[1]

Princeton University

August 24, 2023

[1] These slides are the collective effort of everyone who has contributed to the Princeton Sociology Methods Camp.

Logistics

- Feedback from last session
- Code-along materials
- Quarto
- Homework
- ChatGPT as a coding tool

Logistics: Feedback from last session

Feedback from daily surveys will be discussed here during future days

Logistics: Code-along materials

There is code from these slides available in the “coding-examples” folder. Please code along and try out functions in R as we go through the slides!

Logistics: Quarto

We want to mention that we encourage you to try out Quarto to write documents. In RStudio, go to File > New File > Quarto document. Quarto[2] is an authoring system that uses the exact same syntax as R Markdown, but includes features that make it easier to write in RStudio.

You can also keep using R Markdown for assignments, and that's fine!

You should be able to write in Quarto in the same way you have been in R Markdown, with a few small changes.

Tutorial available online.[3]

[2] https://quarto.org/

[3] https://quarto.org/docs/get-started/hello/rstudio.html

Logistics: Homework-in-camp

We'll practice concepts with the homework assignment, looking at opposition to free trade and support for presidential candidates.

- You'll have time to work on this during camp, and anything you don't finish will be homework.
- Work with your buddy. Only one of you needs to submit the assignment (PDF and raw file) via email by 9 AM tomorrow.

P.S. Many of you will encounter latex/knitting errors (the worst kind!), please try this guide[4].

[4] https://www.eng.famu.fsu.edu/~dommelen/l2h/errors.html#misdol

Logistics: ChatGPT policy

One thing that we'd like you to try in Methods Camp is asking ChatGPT[5] to **help you debug** if you get stuck. The tool can often be wrong, but it can be good for brainstorming what to do next!

Try prompting it in the following way:

> *"I am a social scientist using the tidyverse to analyze a dataset called addh. It has the following columns: age, gender.... Could you explain why the following code doesn't run?"*

- Create the question persona ("I am a social scientist")
- Specify packages you're using ("using tidyverse")
- Specify basics about the data (dataset name, variables)
- Ask it to explain.
- Keep the code input small.

[5] https://chat.openai.com/

Outline of today

- Programming Overview: Philosophies, Practices and Practicalities
- Data Exploration Basics
- Data Manipulation with Dplyr (review from summer)

Section 1

Programming Overview: Philosophies, Practices and Practicalities

Programming Overview: Philosophies, Practices and Practicalities

- R originated from statisticians: maximize statistical performance.
- Most recently R is used by a much wider group, including computer scientists and social scientists.
- There is a distinct movement to push towards reproducible and readable code with a certain bent:
    - Consistency between readability across languages
    - Reproducibility and Version Control (write code for humans, write data for computers), working with Git right away, commenting extensively, making use of logical names (no spaces), avoid duplicate/incremental files, produce markdown documents with all reproducible steps included etc.

Tidyverse

All of this culminates in the Tidyverse (a philosophy and a collection of packages).

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs.

Install the complete tidyverse with:

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

What we teach (and other options)

- We will focus on the tidyverse to start, mainly because it is easier to understand and you can do most data manipulation you need in social science research with these tools. Read more here[6].
- However, we will also expose you to "base R" family functions like $ and [ ], loops and conditionals, data types, and *apply*:
    - You will encounter base R code (most relevantly, in the SOC 500 solution keys) and should know how to read this code, which has been written by multiple TAs over the years.
    - It is occasionally easier to use base R functions for certain tasks, though almost all of the data manipulation you'll need to do has a good tidyverse equivalent.
- As you get more advanced, you may want to look into **data.table** for certain speed improvements. And if you want to write an R package or need to debug a weird issue, understanding **base R** will be useful.

[6] http://varianceexplained.org/r/teach-tidyverse/

What we expect you to know

- We assume you have completed the Posit Primers assignment we gave you over the summer to gain familiarity with the tidyverse
- If you have not, please complete the following Posit Primers[7] before proceeding with the rest of the material:
  - The Basics (all sub-modules)
  - Work With Data (all sub-modules)
  - Visualize Data - (Exploratory Data Analysis and Scatterplots sub-modules)
  - Write Functions (complete Function Basics and How to Write a Function)
- If you would like more practice, you can also complete the rest of the Primers (except for "Build Interactive Web Apps")

[7] https://posit.cloud/learn/primers

Good practices for research code

Trevor Branch
@TrevorABranch

My rule of thumb: every analysis you do on a dataset will have to be redone 10–15 times before publication. Plan accordingly. #Rstats

Good practices for research code

Project Organization WILL Save You

Expect that you'll have to run every piece of research code multiple times in the process of writing a paper, and that you'll want to be able to share your results and figures in a reproducible way.

The earlier you learn good practices, the better!

Good practices for research code

We encourage the following practices to keep your code organized, inspired by Jenny Bryan's "What They Forgot to Teach You About R"[8] and the "Workflows: scripts and projects" chapter in *R for Data Science*[9]:

- Project-oriented workflow (Use R projects, and use the here() package)
- Keep everything you need in source (ie, your Rmd or .R file), not in your RStudio environment
- Use meaningful filenames readable by you and your computer ("01_clean-data_20230823.R")
- Split up your research code into smaller files for cleaning, analysis, and figures, so it's easier to update in the future.
- Keep a copy of your raw data untouched, and create intermediate data output as you go.

Exercise/Live Demo: Set up an R project and organize the files we sent you in a reasonable way.

[8] https://rstats.wtf/
[9] https://r4ds.hadley.nz/workflow-scripts

Section 2

Data Exploration Basics

Data we'll be working with

In-class lecture example: data from 3rd wave of AddHealth containing people's ratings of how important the respondent believes the following are for a "successful marriage or serious committed relationship":

- love
- no cheating
- money

Data we'll be working with

In-class lecture example: data from 3rd wave of AddHealth on how demographic characteristics relate to how important the respondent believes the following are for a "successful marriage or serious committed relationship":

- love
- no cheating
- money

Today's Homework: data from the American National Election Studies (ANES) on how a respondent's degree of opposition to free trade is related to their views about three presidential candidates (at the time): Trump, Sanders, and Clinton

Documentation for AddHealth is online[10], and documentation for ANES is included in homework files.[11]

[10] https://addhealth.cpc.unc.edu/documentation/codebook-explorer/#/variable_collection/1573

[11] The Add Health data included in the open-source version of these slides is fake for privacy purposes. However, you can download publicly released Add Health data from the following link: https://www.icpsr.umich.edu/web/ICPSR/studies/21600/datadocumentation

Today's homework

Preliminary: loading data

- When you use an R project, the working directory is automatically set to where the .Rproj file is located
  - R's commands for reading in data are specific to the file type – the most common is read_csv() for csv files
  - Another common one is importing foreign data types like STATA .dta files using haven::read_dta()

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

## check working directory
getwd()
```

```
## [1] "/Users/al49/Dropbox (Princeton)/Teaching and Mentoring/Metho
```

```
## read in from where .Rproj file is
addh <- read_csv("data/addhealthfake.csv")
```

Preliminary: loading data

- Use the "here" package to ensure that your file paths will work across different computer systems (Mac / vs. PC \)

```
# install.packages("here")
library("tidyverse")
library("here")

## usual way
addh <- read_csv("data/addhealthfake.csv")

## using here()
addh <- read_csv(here("data", "addhealthfake.csv"))

## using here() + creating a separate var called "path"
## in case you need to read it in multiple places in your code
path <- here("data", "addhealthfake.csv")
addh <- read_csv(path)
```

Preliminary: explore data

- While we will primarily teach you the "tidyverse" methods for doing things, there are a few operations that are easier to do with base R, including preliminary data exploration.
- The following sections describes a **few functions from "base R"** that are useful to know in early dataset exploration.
  - Note that the `dataframe$variable` format is also from base R.
  - In the tidyverse, you generally pass the `dataframe` and the `variable` as two separate arguments to a function, for example - `arrange(dataframe, variable)`.

Preliminary: explore data

When you load a tabular dataset, you should:

- check the data types of each variable
- check the dimensions of the data
- look at a few rows and variables

To look at the type for a single variable, use **base R notation with a $**:

```
class(addh$age)
```

```
## [1] "numeric"
```

```
class(addh$gender)
```

```
## [1] "character"
```

Preliminary: explore data

To look at the type of an R object, put the whole object into the `class` function:

```
class(addh)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

You can see it is a tibble (a tidyverse dataframe, indicated by `tbl_df`), which is a special type of `data.frame`.

You can use certain functions like `class()` and `summary()` on vectors as well as dataframes (more on these terms later).

Preliminary: explore data

To get information about the whole tibble, use the following functions:

- `summary()`: numeric summaries
- `str()`: data types and sample data
- `colnames()` or `names()`: names of columns/variables
- `dim()`: dimensions
- `View()`: view all data in RStudio viewer (can be slow if data is large)
- `head()`: top 10 rows, can adjust `n`
- `tail()`: bottom 10 rows, can adjust `n`
- (in dplyr) `slice_sample()`: randomly select `n` rows

Run `?head()` to look at the help documentation for the function in R.

Preliminary: explore data

The `str()` function gives you a lot of information!

```
str(addh)
```

```
## spc_tbl_ [3,000 x 11] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ id           : num [1:3000] 1 2 3 4 5 6 7 8 9 10 ...
##  $ age          : num [1:3000] 18 22 18 26 27 21 19 27 18 25 ...
##  $ gender       : chr [1:3000] "female" "male" "female" "female" ...
##  $ income       : num [1:3000] 19252 11617 16189 18194 24484 ...
##  $ logincome    : num [1:3000] 9.87 9.36 9.69 9.81 10.11 ...
##  $ debt         : chr [1:3000] "yesdebt" "nodebt" "yesdebt" "yesdebt" ...
##  $ love         : num [1:3000] 1 10 10 2 5 10 3 4 1 6 ...
##  $ nocheating   : num [1:3000] 7 10 3 1 10 4 10 10 10 3 ...
##  $ money        : num [1:3000] 9 3 5 3 9 9 9 7 3 8 ...
##  $ paypercent   : num [1:3000] 46 56 42 82 93 42 89 55 43 53 ...
##  $ logpaypercent: num [1:3000] 3.83 4.03 3.74 4.41 4.53 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   id = col_double(),
##   ..   age = col_double(),
##   ..   gender = col_character(),
##   ..   income = col_double(),
##   ..   logincome = col_double(),
##   ..   debt = col_character(),
##   ..   love = col_double(),
##   ..   nocheating = col_double(),
##   ..   money = col_double(),
##   ..   paypercent = col_double(),
```

Preliminary: explore data

The head function is useful for looking at the first few rows. You can also sample random rows.

```
head(addh, n = 5)
```

```
## # A tibble: 5 x 11
##      id   age gender income logincome debt     love nocheating money paypercent
##   <dbl> <dbl> <chr>   <dbl>     <dbl> <chr>   <dbl>      <dbl> <dbl>      <dbl>
## 1     1    18 female 19252.      9.87 yesdebt     1          7     9         46
## 2     2    22 male   11617.      9.36 nodebt     10         10     3         56
## 3     3    18 female 16189.      9.69 yesdebt    10          3     5         42
## 4     4    26 female 18194.      9.81 yesdebt     2          1     3         82
## 5     5    27 female 24484.     10.1  yesdebt     5         10     9         93
## # i 1 more variable: logpaypercent <dbl>
```

```
slice_sample(addh, n = 5) # from dplyr
```

```
## # A tibble: 5 x 11
##      id   age gender income logincome debt    love nocheating money paypercent
##   <dbl> <dbl> <chr>   <dbl>     <dbl> <chr>  <dbl>      <dbl> <dbl>      <dbl>
## 1   952    18 male   27036.     10.2  nodebt    10          1     3          5
## 2  1119    21 female 27014.     10.2  nodebt     1         10     6         84
## 3  1488    19 male   21159.      9.96 nodebt     4          7     6         68
## 4   224    18 male    7367.      8.90 nodebt    10          4     6         87
## 5   589    23 male   24513.     10.1  nodebt     8          7     7         83
## # i 1 more variable: logpaypercent <dbl>
```

Preliminary: explore data

To get information about one variable, you can pull out a single variable using “$” and use the following functions:

- `table()`: get a table summarizing counts
- `unique()`: get the unique responses for a variable
- `sort()`: sort the numerically (or alphabetically)
- `hist()`: produce a histogram summary (for a numeric variable)

```
table(addh$gender) # number of respondents in each category
```

```
##
## female   male
##   1503   1497
```

Preliminary: explore data

To get information about one variable, you can pull out a single variable using "$" and use the following functions:

- `table()`: get a table summarizing counts
- `unique()`: get the unique responses for a variable
- `sort()`: sort the numerically (or alphabetically)
- `hist()`: produce a histogram summary (for a numeric variable)

```
unique(addh$age)
```

```
##  [1] 18 22 26 27 21 19 25 24 20 23
```

```
sort(unique(addh$age))
```

```
##  [1] 18 19 20 21 22 23 24 25 26 27
```

```
sort(unique(addh$gender))
```

```
## [1] "female" "male"
```

Preliminary: subset data

To subset sections of your data, use the **base R subsetting syntax**[12] with `[row index, column index]`.

Exercise: How would you subset the observation in the fourth row and the second column?

```
# get first row
addh[1, ]
```

```
## # A tibble: 1 x 11
##      id   age gender income logincome debt     love nocheating money paypercent
##   <dbl> <dbl> <chr>   <dbl>     <dbl> <chr>   <dbl>      <dbl> <dbl>      <dbl>
## 1     1    18 female 19252.      9.87 yesdebt     1          7     9         46
## # i 1 more variable: logpaypercent <dbl>
```

```
# get first column, rows 1 through 3 (colon means "through")
addh[1:3, 1]
```

```
## # A tibble: 3 x 1
##      id
##   <dbl>
## 1     1
## 2     2
## 3     3
```

[12] More about subsetting, base R style: https://r4ds.hadley.nz/base-r#subsetting-data-frames

Preliminary: subset data

You can use the - (minus) sign to subset everything except for a row or column.

Exercise: How would you subset everything besides the fifth column?

```
# get everything besides first row
addh[-1, ]
```

```
## # A tibble: 2,999 x 11
##       id   age gender income logincome debt     love nocheating money paypercent
##    <dbl> <dbl> <chr>   <dbl>     <dbl> <chr>   <dbl>      <dbl> <dbl>      <dbl>
##  1     2    22 male   11617.      9.36 nodebt     10         10     3         56
##  2     3    18 female 16189.      9.69 yesdebt    10          3     5         42
##  3     4    26 female 18194.      9.81 yesdebt     2          1     3         82
##  4     5    27 female 24484.     10.1  yesdebt     5         10     9         93
##  5     6    21 female 22353.     10.0  nodebt     10          4     9         42
##  6     7    19 male   11842.      9.38 yesdebt     3         10     9         89
##  7     8    27 female 19874.      9.90 nodebt      4         10     7         55
##  8     9    18 male   27422.     10.2  nodebt      1         10     3         43
##  9    10    25 female  9968.      9.21 yesdebt     6          3     8         53
## 10    11    24 female 26354.     10.2  nodebt     10         10    10         52
## # i 2,989 more rows
## # i 1 more variable: logpaypercent <dbl>
```

Preliminary: ask a few questions

You may want to answer a few questions about your data before analyzing it. Here are a few things you could answer:

- What's the median income of this sample? What's the mean age?

Preliminary: ask a few questions

You may want to answer a few questions about your data before analyzing it. Here are a few things you could answer:

- What's the median income of this sample? What's the mean age?
- On average, do the young adults surveyed think money, no cheating, or love is more important in a relationship?

Preliminary: ask a few questions

You may want to answer a few questions about your data before analyzing it. Here are a few things you could answer:

- What's the median income of this sample? What's the mean age?
- On average, do the young adults surveyed think money, no cheating, or love is more important in a relationship?
- What are the answer choices for `debt`?

Preliminary: ask a few questions

You may want to answer a few questions about your data before analyzing it. Here are a few things you could answer:

- What's the median income of this sample? What's the mean age?
- On average, do the young adults surveyed think money, no cheating, or love is more important in a relationship?
- What are the answer choices for `debt`?

- **Exercise**: In your buddy groups, manually write down R code to answer one (or more) of these questions. Can you think of other questions?

Preliminary: ask a few questions

What's the median income of this sample? What's the mean age?

```
median(addh$income)
```

```
## [1] 15127.34
```

```
mean(addh$age)
```

```
## [1] 22.51133
```

Preliminary: ask a few questions

On average, do the young adults surveyed think money, no cheating, or love is more important in a relationship?

```
# using dplyr
summarize(addh,
          mean_money = mean(money),
          mean_nocheating = mean(nocheating),
          mean_love = mean(love))
```

```
## # A tibble: 1 x 3
##   mean_money mean_nocheating mean_love
##        <dbl>           <dbl>     <dbl>
## 1       5.57            7.69      7.71
```

Preliminary: ask a few questions

What are the answer choices for `debt`?

```
unique(addh$debt)
```

```
## [1] "yesdebt" "nodebt"
```

Section 3

Data Manipulation with Dplyr

Outline of dplyr review

- dplyr "verbs":
  - select
  - filter
  - arrange
  - mutate
  - group_by
  - summarise
- rename
- chaining together verbs with pipe operator %>%
  - "base R" now has a native pipe |> that also works!

dplyr: basic structure of verbs

verb(name of data.frame or object, operation 1 to perform, operation 2 to perform...)

select: a way to extract columns

Can be used in combination with other dplyr verbs such as: contains, starts_with, and ends_with

Example: extract any column with the word "pay": paypercent and logpaypercent

```
paycold <- select(addh, contains("pay"))
head(paycold, 3)
```

```
## # A tibble: 3 x 2
##   paypercent logpaypercent
##        <dbl>         <dbl>
## 1         46          3.83
## 2         56          4.03
## 3         42          3.74
```

filter: a way to extract rows

Can be used in combination with logical statements

Example: extract observations with an income <20,000 year but no debt

```
nodebtd <- filter(addh, debt == "nodebt" &
                    income < 20000)
nrow(nodebtd)
```

```
## [1] 1087
```

filter: a way to extract missing rows

Can be used in combination with logical statements for missing data

Example: remove observations where income is missing

```
nomissinginc <- filter(addh, !is.na(income)) # only keep obs that are NOT (!) na

nomissinginc <- drop_na(addh, income) # alternate function from tidyr

nrow(nomissinginc)
```

```
## [1] 3000
```

arrange: a way to arrange rows by the order of their column values

Example: find the two observations who think money is extremely important for a relationship (10 on money variable) but who pay for the fewest percentage of dates (paypercent)

```
addh %>%
  filter(money == 10) %>%
  arrange(paypercent) %>%
  head(2)
```

```
## # A tibble: 2 x 11
##      id   age gender income logincome debt     love nocheating money paypercent
##   <dbl> <dbl> <chr>   <dbl>     <dbl> <chr>   <dbl>      <dbl> <dbl>      <dbl>
## 1   811    22 male   34161.     10.4  yesdebt    10          9    10          2
## 2  2086    20 male    4816.      8.48 yesdebt    10         10    10          2
## # i 1 more variable: logpaypercent <dbl>
```

mutate: a way to add new variables to the data.frame

Example: add a variable with the average rating for nocheating, money, and love's importance for a relationship (sum divided by 3) and another variable that logs that rating

```
addhd <- mutate(addh,
                rateavg = (love + money + nocheating)/3,
                rateavglog = log(rateavg))

# look at the first 3 rows and some columns
addhd %>%
  select(love, money, nocheating, rateavg, rateavglog) %>%
  head(3)
```

```
## # A tibble: 3 x 5
##    love money nocheating rateavg rateavglog
##   <dbl> <dbl>      <dbl>   <dbl>      <dbl>
## 1     1     9          7    5.67       1.73
## 2    10     3         10    7.67       2.04
## 3    10     5          3    6          1.79
```

mutate: a way to add new variables to the data.frame

Be aware that your choice of names affects whether a new object or column is created with mutate!

- By using the same column name or same object name, you overwrite the original object or column.
- Unlike Stata, the default in R is not to change the underlying data, so you must intentionally save it with <-.

Example: multiple ways to store a new variable that logs the rating of love

```
# New column, new dataframe
addhnew <- mutate(addh,
                  loglove = log(love))

# New column, same dataframe
addh <- mutate(addh,
                loglove = log(love))

# Overwrite old column, same dataframe
addh <- mutate(addh,
               love = log(love))

# Overwrite old column, new dataframe
addhnew <- mutate(addh,
                  love = log(love))
```

group_by and summarise: a way to collapse data by category and generate summary statistics

Example:

1. Group by gender
2. Generate a summary statistic of not cheating's importance on that grouped data

```
gender_group <- group_by(addh, gender)
summarise(gender_group,
          meannocheat = mean(nocheating))
```

```
## # A tibble: 2 x 2
##   gender meannocheat
##   <chr>        <dbl>
## 1 female        7.79
## 2 male          7.60
```

group_by and summarise: a way to collapse data by category and generate summary statistics

Summarise also has a number of verbs for creating summary statistics:

1. n(): count the elements in a group

group_by and summarise: a way to collapse data by category and generate summary statistics

Summarise also has a number of verbs for creating summary statistics:

1. n(): count the elements in a group
2. n_distinct(): count the distinct elements in a group

group_by and summarise: a way to collapse data by category and generate summary statistics

Summarise also has a number of verbs for creating summary statistics:

1. n(): count the elements in a group
2. n_distinct(): count the distinct elements in a group
3. first: list the first element (would usually use in combo with arrange)

group_by and summarise: a way to collapse data by category and generate summary statistics

Summarise also has a number of verbs for creating summary statistics:

1. n(): count the elements in a group
2. n_distinct(): count the distinct elements in a group
3. first: list the first element (would usually use in combo with arrange)
4. last: list the lest element (same as above)

group_by and summarise

Example: find: 1) the number of females and males by debt status, 2) the percentage in each debt x gender category as a fraction of all observations; 3) the number of distinct ratings of love's importance in each of these debt x gender categories

```
genderdebt <- group_by(addh, gender, debt)
summarise(genderdebt,
          count = n(),
          percent = n()/nrow(addh),
          distinctlove = n_distinct(love))
```

```
## # A tibble: 4 x 5
## # Groups:   gender [2]
##   gender debt    count percent distinctlove
##   <chr>  <chr>   <int>   <dbl>        <int>
## 1 female nodebt    768   0.256           10
## 2 female yesdebt   735   0.245           10
## 3 male   nodebt    745   0.248           10
## 4 male   yesdebt   752   0.251           10
```

Rename

- Rename: you can use rename() as a function to modify names of columns.
- You can rename numerous columns by using c() to produce a 1-D array to pass to the replace position (more details later)

```
# let's use one of the built in R datasets, mtcars
head(mtcars, 3)
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

```
# what if we want the names to be more informative
mtcars2 <- rename(mtcars,
                    c("displacement" = "disp",
                      "milespergal" = "mpg"))
head(mtcars2, 3)
```

```
##               milespergal cyl displacement  hp drat    wt  qsec vs am gear carb
## Mazda RX4            21.0   6          160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag        21.0   6          160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710           22.8   4          108  93 3.85 2.320 18.61  1  1    4    1
```

Combining multiple verbs with piping

- You'll notice that the example on the previous slides combines multiple actions:
- Pipes provide a way to chain together multiple verbs in a specified order.
- Pipes (%>%) comes from the *magrittr* package with two aims: to decrease development time and to improve readability and maintainability of code.
- This operator %>% allow you to pipe a value forward into an expression or function call; something along the lines of *x %>% f*, rather than *f(x)*. It might be helpful to think of this as ...then...

Piping Functional Sequence

The basic (pseudo) usage of the pipe operator goes something like this:

```
awesome_data <-
  raw_interesting_data %>%
  transform(somehow) %>%
  filter(the_good_parts) %>%
  finalize
```

This takes an input, an output, and a sequence transformations. That's suprisingly close to the definition of a function, so magrittr is really just a convenient way of of defining and applying a function. (Also try command + shift + m for a nice short cut!)

Base R now has a native pipe, |>, that you can also use with tidyverse functions.

Example of combining multiple verbs with piping

Example: - Group the data by gender and debt status - Find the average rating of love, no cheating, and money's importance for a relationship in each group - Arrange the groups by their rating of money's importance to a relationship from the highest to rating to the lowest rating

What would this look like, still using dplyr, but without piping? A nested mess...

```
arrange(summarise(group_by(addh, gender, debt),
                       nocheatavg = mean(nocheating),
                       loveavg = mean(love),
                       moneyavg = mean(money)), desc(moneyavg))
```

```
## # A tibble: 4 x 5
## # Groups:   gender [2]
##   gender debt    nocheatavg loveavg moneyavg
##   <chr>  <chr>        <dbl>   <dbl>    <dbl>
## 1 male   yesdebt       7.72    1.91     5.66
## 2 female yesdebt       7.75    1.88     5.59
## 3 female nodebt        7.83    1.93     5.54
## 4 male   nodebt        7.47    1.89     5.49
```

Piping: begin from the "most nested"/first operation and move to the last

1) Group the data by gender and debt status; 2) find the avg. rating of love, no cheating, and money's importance; 3) arrange the groups from rating money's importance the highest to rating it the lowest

- Without piping:

```
arrange(summarise(group_by(addh, gender, debt),
nocheatavg = mean(nocheating), loveavg = mean(love), moneyavg =
mean(money)), desc(moneyavg))
```

- With piping:

```
addh %>%
group_by(gender, debt) %>%
summarise(nocheatavg = mean(nocheating), loveavg = mean(love), moneyavg =
mean(money)) %>%
arrange(desc(moneyavg))
```

Implementing in R with pipes

And as a bonus, rename the columns into something readble.

```
addh %>%
  group_by(gender, debt) %>%
  summarise(nocheatavg = mean(nocheating),
            loveavg = mean(love),
            moneyavg = mean(money)) %>%
  arrange(desc(moneyavg)) %>%
  rename(c("No cheating average" = "nocheatavg",
           "Love average" = "loveavg",
           "Money average" = "moneyavg"))
```

```
## `summarise()` has grouped output by 'gender'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 4 x 5
## # Groups:   gender [2]
##   gender debt    `No cheating average` `Love average` `Money average`
##   <chr>  <chr>                   <dbl>          <dbl>           <dbl>
## 1 male   yesdebt                  7.72           1.91            5.66
## 2 female yesdebt                  7.75           1.88            5.59
## 3 female nodebt                   7.83           1.93            5.54
## 4 male   nodebt                   7.47           1.89            5.49
```

Summing up

In this lecture, we've reviewed:

- Practices for organizing research code
- Basics of data exploration and working with variables
- dplyr and pipes as a tool for data manipulation

For tomorrow

- Topics:
  - Recoding variables as a case study for programming concepts in R
    - Data types
    - Logical statements
    - Control structures
  - For loops
- Fill out feedback form
- Complete homework with your buddy