Survey Research and Design

Data Manipulation and Summary Using tidyverse R

William Marble

September 7, 2023

Last Time

Process of survey research:

1. Develop construct

Last Time

Process of survey research:

1. Develop construct
2. Write survey questions to measure the construct

Last Time

Process of survey research:

1. Develop construct
2. Write survey questions to measure the construct
3. Identify population and sampling frame

Last Time

Process of survey research:

1. Develop construct
2. Write survey questions to measure the construct
3. Identify population and sampling frame
4. Administer survey to a sample

Last Time

Process of survey research:

1. Develop construct
2. Write survey questions to measure the construct
3. Identify population and sampling frame
4. Administer survey to a sample
5. **Analyze the results**

R for Survey Research

- R is an open-source statistical programming language
- Two flavors of R: “base” and “tidyverse”
- Base R: minimal packages, sometimes awkward syntax
- **tidyverse**: suite of packages (dplyr, tidyr) that are meant to be more expressive, creating code that’s faster to write and easier to read

Roadmap for Today

1. Survey analysis concepts
2. Loading data
3. (Re?) Introducing the "pipe" operator
4. Data manipulation
5. Some simple data summaries

Survey Analysis Concepts

Quantitative Descriptors

Surveys are aimed at **quantitative descriptions** of the target population.
Typically involves:

- Means (averages)
- Distributions (quantiles, standard deviation)
- Cross-tabs

Quantitative Descriptors

Surveys are aimed at **quantitative descriptions** of the target population. Typically involves:

- Means (averages)
- Distributions (quantiles, standard deviation)
- Cross-tabs

Surveys often are **weighted** to adjust for discrepancies between sample and population $\rightsquigarrow$ need to account for these weights in analysis.

Means

The mean is the average of a variable — provides a *measure of central tendency*

Sum up the value in each observation and divide by the number of observations:

Means

The mean is the average of a variable — provides a *measure of central tendency*

Sum up the value in each observation and divide by the number of observations:

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_N}{N} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Often "bars" over variables indicate the mean of that variable, e.g. $\bar{x}$

In `R`: `mean(x)`

Weighted Means

Weighted means allow some observations to count more than others.

Simple example:

- Suppose the population of interest is all adults in the U.S.
- But for some reason, we find that the share of people under 35 in our sample is twice as big as in the population
- Why might that be a problem?

Weighted Means

Weighted means allow some observations to count more than others.

Simple example:

- Suppose the population of interest is all adults in the U.S.
- But for some reason, we find that the share of people under 35 in our sample is twice as big as in the population
- Why might that be a problem?
- We should *downweight* people under 35 and *upweight* those over 35

Weighted Means

Weighted means allow some observations to count more than others.

Simple example:

- Suppose the population of interest is all adults in the U.S.
- But for some reason, we find that the share of people under 35 in our sample is twice as big as in the population
- Why might that be a problem?
- We should *downweight* people under 35 and *upweight* those over 35

To compute weighted means, we multiply each value by its weight before summing, then divide by the sum of the weights:

$$\bar{x}_w = \frac{x_1 w_1 + x_2 w_2 + \cdots + x_N w_N}{w_1 + w_2 + \cdots + w_N} = \frac{\sum_{i=1}^{N} x_i w_i}{\sum_{i=1}^{N} w_i}$$

In R: `weighted.mean(x, weights = w)`

Mean of Binary Variables

Binary (0/1) variables are important in survey research. We can code any question with two response options as a binary variable:

- Agree = 1, Disagree = 0
- Plan to vote = 1, Don't plan to vote = 0
- Democrat = 1, Republican = 0

The mean of a binary variable is simply the proportion of 1's!

$$\bar{x} = \frac{1}{N}\sum_i x_i = \frac{1}{N}(\#\text{ of } i \text{ with } x_i = 1)$$

Mean of Binary Variables

Binary (0/1) variables are important in survey research. We can code any question with two response options as a binary variable:

- Agree = 1, Disagree = 0
- Plan to vote = 1, Don't plan to vote = 0
- Democrat = 1, Republican = 0

The mean of a binary variable is simply the proportion of 1's!

$$\bar{x} = \frac{1}{N}\sum_i x_i = \frac{1}{N}(\#\text{ of } i \text{ with } x_i = 1)$$

So mean of a turnout variable (0/1) is the share of the sample that voted. Mean of an agree/disagree variable is the share of the sample that agree.

Proportion Tables

Rather than restricting ourselves to binary variables, if there are multiple response options we can compute the share of the sample giving each response.

In survey world, sometimes called the survey "topline":

How important will each of the following issues be in your vote for Congress this November? - The economy

Extremely impt	Very impt	Somewhat impt	Not so impt	Not impt at all
50.7	33.0	12.3	2.0	2.0

In `R`: `table()` computes the number in each category. `prop.table()` normalize the output of `table()` to sum to 1.

Could also compute this as a series of means (how?)

Cross-Tabs

"Cross-tab" in survey research refers to the distribution of answers to *two* questions – e.g. importance of economy by education:

Education	Extremely impt	Very impt	Somewhat impt	Not so impt	Not impt at all
$<$ HS	66.7	33.3	0.0	0.0	0.0
HS	51.2	29.3	16.3	0.8	2.4
Some college	56.3	35.2	7.0	0.0	1.4
College grad	45.0	33.5	14.9	4.1	2.5

Cross-Tabs

"Cross-tab" in survey research refers to the distribution of answers to *two* questions – e.g. importance of economy by education:

Education	Extremely impt	Very impt	Somewhat impt	Not so impt	Not impt at all
$<$ HS	66.7	33.3	0.0	0.0	0.0
HS	51.2	29.3	16.3	0.8	2.4
Some college	56.3	35.2	7.0	0.0	1.4
College grad	45.0	33.5	14.9	4.1	2.5

In R: `table(row.var, col.var)`

`prop.table(table.output, margin = 1)` to normalize rows to sum to 1

`prop.table(table.output, margin = 2)` to normalize columns to sum to 1

Loading Data

Data We'll Work With

`https://doi.org/10.25940/ROPER-31119618`

Country:	United States
Title:	ABC News/Ipsos Poll: 2022 Wave 44
Survey Organization(s):	Ipsos
Sponsor(s):	ABC News
Field Dates:	June 3 - 4, 2022
Sample:	National adult
Sample Size:	542
Sample Notes:	None
Interview method:	Web-based survey
Weight Location:	Columns 196-201 (x.xxxx) -- Varname: WEIGHTS_
No. of records per respondent:	1

Four available formats:

- SPSS portable (.por)
- Stata (.dta)
- CSV (.csv), plain text file (often less info – e.g. no variable labels)
- ASCII (.dat), plain text file – avoid if possible

Most data can be loaded using `import` from the `rio` package

Other Data Formats

Other R-specific formats you might see:

- RData format (.rdata, .Rdata, .rda, etc.) – might contain multiple objects. Don't save these files, use .rds instead
- RDS format (.rds) – contains a single object

Reading in Data: .por

```
> library(rio)
> dat <- import("data/31119618.por")
> names(dat)

 [1] "ID"       "XSPANISH" "COMPLETE" "PPAGE"    "PPEDUC5"  "PPEDUCAT"
 [7] "PPGENDER" "PPETHM"   "PPHHSIZE" "PPINC7"   "PPMARIT5" "PPMSACAT"
[13] "PPREG4"   "PPRENT"   "PPSTATEN" "PPWORKA"  "PPEMPLOY" "Q1_A"
[19] "Q1_B"     "Q1_C"     "Q1_D"     "Q1_F"     "Q1_G"     "Q1_I"
[25] "Q1_J"     "Q1_K"     "Q1_L"     "Q1_M"     "Q2"       "Q3_A"
[31] "Q3_B"     "Q3_C"     "Q3_D"     "Q3_E"     "Q3_F"     "Q3_G"
[37] "Q3_H"     "Q3_I"     "Q3_J"     "Q3_K"     "Q4"       "Q5"
[43] "QPID"     "ABCAGE"   "CONTACT"  "WEIGHTS_"
```

Examining Data

```
> head(dat[,c(1:10, 20)])

        ID XSPANISH  COMPLETE PPAGE PPEDUC5 PPEDUCAT PPGENDER PPETHM PPHHSIZE
1 60300001        1 qualified    52       4        4        1      1        2
2 60300002        1 qualified    58       4        4        2      1        2
3 60300003        1 qualified    62       1        1        2      5        2
4 60300004        1 qualified    75       5        4        1      2        2
5 60300005        1 qualified    46       5        4        1      4        2
6 60300006        1 qualified    57       3        3        2      1        3
  PPINC7 Q1_C
1      4    2
2      5    2
3      7    2
4      7    1
5      7    1
6      3    2
```

Data Cleaning/Management Tasks

From looking at the data, we can see some data pre-processing we'll have to do:

- Give columns informative names
- Recode responses with informative labels
- (Perhaps) recode some responses as `NA`'s

But first some basics...

The “Pipe” Operator %>%

Working with the "Pipe" Operator

- The "pipe" operator %>% is widely used in tidyverse dialect
- In RStudio: type `cmd + shift + M` in Mac or `ctrl + shift + M`
- Use `library(tidyverse)` to access it
- It takes the object on the left and "pipes" it to the function on the right
- It looks like this:

  ```
  > x %>% some_function()
  ```

- Equivalent to:

  ```
  > some_function(x)
  ```

- Useful for stringing together complex data transformations

Pipe Operator Example

```
> library(tidyverse)
> x = 1:99
> head(x, 10)

 [1]  1  2  3  4  5  6  7  8  9 10
```

Pipe Operator Example

```
> library(tidyverse)
> x = 1:99
> head(x, 10)

 [1]  1  2  3  4  5  6  7  8  9 10

> x %>% mean()

[1] 50

> mean(x)

[1] 50

> x %>% sd()

[1] 28.72281

> sd(x)

[1] 28.72281
```

Pipe Operator Example

By default, the pipe passes the object as the first argument. We can include additional arguments:

```
> x = 1:99
> weights = runif(n = 99, min = 0, max = 1)
> x %>% weighted.mean(w = weights)
```

```
[1] 49.60341
```

```
> weighted.mean(x, w = weights)
```

```
[1] 49.60341
```

Pipe Operator Example

By default, the pipe passes the object as the first argument. We can include additional arguments:

```
> x = 1:99
> weights = runif(n = 99, min = 0, max = 1)
> x %>% weighted.mean(w = weights)
```

```
[1] 49.60341
```

```
> weighted.mean(x, w = weights)
```

```
[1] 49.60341
```

We can also pass the object on the left to another argument by referring to it as `.` (a period/dot) in the function:

```
> weights %>% weighted.mean(x = x, w = .)
```

```
[1] 49.60341
```

Managing Data Frames

Accessing Elements of a Data Frame

- When we load the data, it is stored in an object called a `data.frame`

Accessing Elements of a Data Frame

- When we load the data, it is stored in an object called a `data.frame`
- This is a collection of columns, which you can access using the $ operator:

```
> dat <- import("data/31119618.csv")
> class(dat$ppage)
```

```
[1] "integer"
```

```
> head(dat$ppage, 3)
```

```
[1] 52 58 62
```

Accessing Elements of a Data Frame

- When we load the data, it is stored in an object called a `data.frame`
- This is a collection of columns, which you can access using the $ operator:

```
> dat <- import("data/31119618.csv")
> class(dat$ppage)
```

```
[1] "integer"
```

```
> head(dat$ppage, 3)
```

```
[1] 52 58 62
```

Accessing Elements of a Data Frame

- When we load the data, it is stored in an object called a `data.frame`
- This is a collection of columns, which you can access using the $ operator:

```
> dat <- import("data/31119618.csv")
> class(dat$ppage)
```

```
[1] "integer"
```

```
> head(dat$ppage, 3)
```

```
[1] 52 58 62
```

- **For tidyverse functions, we can usually pass the function a data frame using the pipe operator then refer to variable names directly**

Key `tidyverse` functions

tidyverse has a few key functions:

- `rename()`: rename variables

Key `tidyverse` functions

tidyverse has a few key functions:

- `rename()`: rename variables
- `select()`: select specified variables (dropping all others) and (optionally) rename them

Key `tidyverse` functions

tidyverse has a few key functions:

- `rename()`: rename variables
- `select()`: select specified variables (dropping all others) and (optionally) rename them
- `mutate()`: create or modify a variable

Key `tidyverse` functions

tidyverse has a few key functions:

- `rename()`: rename variables
- `select()`: select specified variables (dropping all others) and (optionally) rename them
- `mutate()`: create or modify a variable
- `filter()`: select rows that meet some criterion

Key `tidyverse` functions

tidyverse has a few key functions:

- `rename()`: rename variables
- `select()`: select specified variables (dropping all others) and (optionally) rename them
- `mutate()`: create or modify a variable
- `filter()`: select rows that meet some criterion
- `group_by()`: create groups; subsequent operations happen separately for each group

Key `tidyverse` functions

tidyverse has a few key functions:

- `rename()`: rename variables
- `select()`: select specified variables (dropping all others) and (optionally) rename them
- `mutate()`: create or modify a variable
- `filter()`: select rows that meet some criterion
- `group_by()`: create groups; subsequent operations happen separately for each group
- `summarise()`: summarize the data frame

Renaming Variables

Survey datasets often have uninformative variable names:

```
> names(dat)

 [1] "id"       "xspanish" "complete" "ppage"    "ppeduc5"  "ppeducat"
 [7] "ppgender" "ppethm"   "pphhsize" "ppinc7"   "ppmarit5" "ppmsacat"
[13] "ppreg4"   "pprent"   "ppstaten" "ppworka"  "ppemploy" "q1_a"
[19] "q1_b"     "q1_c"     "q1_d"     "q1_f"     "q1_g"     "q1_i"
[25] "q1_j"     "q1_k"     "q1_l"     "q1_m"     "q2"       "q3_a"
[31] "q3_b"     "q3_c"     "q3_d"     "q3_e"     "q3_f"     "q3_g"
[37] "q3_h"     "q3_i"     "q3_j"     "q3_k"     "q4"       "q5"
[43] "qpid"     "abcage"   "contact"  "weights_"
```

Renaming Variables

Check the codebook to figure out what each variable means. E.g., here's question 4

4. Which of these will be the single most important issue in your vote for Congress this November?

	June 3-4
Inflation	21
The economy	19
Gun violence	17
Abortion	12
Gas prices	8
Immigration	6
Climate change	5
Crime	3
The coronavirus pandemic	2
The situation with Russia and Ukraine	1
Taxes	1
Something else	6
Skipped	1

Renaming Variables

We can use the tidyverse function `rename` to give more informative variable names using the syntax:

```
> df %>%
+   rename(newname = oldname)
```

Renaming Variables

We can use the tidyverse function `rename` to give more informative variable names using the syntax:

```
> df %>%
+   rename(newname = oldname)
```

Compare this to the base R version:

```
> names(df)[names(df) == "oldname"] = "newname"
```

Renaming Variables

Applying this to our dataset:

```
> dat <- dat %>%
+   rename(most_imp_iss = q4,
+          gun_laws_rights = q5,
+          vote_enthusiasm = q2)
> names(dat)
```

```
 [1] "id"              "xspanish"        "complete"        "ppage"
 [5] "ppeduc5"         "ppeducat"        "ppgender"        "ppethm"
 [9] "pphhsize"        "ppinc7"          "ppmarit5"        "ppmsacat"
[13] "ppreg4"          "pprent"          "ppstaten"        "ppworka"
[17] "ppemploy"        "q1_a"            "q1_b"            "q1_c"
[21] "q1_d"            "q1_f"            "q1_g"            "q1_i"
[25] "q1_j"            "q1_k"            "q1_l"            "q1_m"
[29] "vote_enthusiasm" "q3_a"            "q3_b"            "q3_c"
[33] "q3_d"            "q3_e"            "q3_f"            "q3_g"
[37] "q3_h"            "q3_i"            "q3_j"            "q3_k"
[41] "most_imp_iss"    "gun_laws_rights" "qpid"            "abcage"
[45] "contact"         "weights_"
```

`select`ing columns

You can get rid of extraneous columns using the tidyverse function `select`:

```
> df %>%
+   select(var1, var2, keep_this_variable)
```

selecting columns

You can get rid of extraneous columns using the tidyverse function `select`:

```
> df %>%
+   select(var1, var2, keep_this_variable)
```

Select can also rename at the same time using same syntax as `rename`

```
> df %>%
+   select(new_name = var1, var2, keep_this_variable)
```

selecting columns

You can get rid of extraneous columns using the tidyverse function `select`:

```
> df %>%
+   select(var1, var2, keep_this_variable)
```

Select can also rename at the same time using same syntax as `rename`

```
> df %>%
+   select(new_name = var1, var2, keep_this_variable)
```

There are several helper functions you can use here. See tidyverse website for more.

```
> df %>%
+   select(starts_with("party"), matches("match"))
```

`select` on example data

Suppose we only want to keep party ID, age, race, gender, education, survey weight, most important issue, voter enthusiasm, and gun control vs. gun rights.

```
> dat <- dat %>%
+   select(pid = qpid,   # party ID
+          age = ppage,  # age
+          race = ppethm,  # race/etbnicity
+          gender = ppgender, # gender
+          educ = ppeducat, # education
+          weight = weights_, # survey weight
+          most_imp_iss,
+          gun_laws_rights,
+          vote_enthusiasm)
> names(dat)

[1] "pid"              "age"              "race"             "gender"
[5] "educ"             "weight"           "most_imp_iss"     "gun_laws_rights"
[9] "vote_enthusiasm"
```

filter

To only keep rows in the data that meet some condition use the tidyverse function `filter`

```
> df %>%
+   filter(var1 == value_to_keep,
+          var2 >  some_other_value,
+          !is.na(keep_this_variable))
```

filter

To only keep rows in the data that meet some condition use the tidyverse function `filter`

```
> df %>%
+   filter(var1 == value_to_keep,
+          var2 >  some_other_value,
+          !is.na(keep_this_variable))
```

The expression inside `filter` is implictly an "AND" statement. The whole statement evaluates to TRUE if all elements are TRUE and evaluates to FALSE otherwise.

filter

To only keep rows in the data that meet some condition use the tidyverse function `filter`

```
> df %>%
+   filter(var1 == value_to_keep,
+          var2 >  some_other_value,
+          !is.na(keep_this_variable))
```

The expression inside `filter` is implictly an "AND" statement. The whole statement evaluates to TRUE if all elements are TRUE and evaluates to FALSE otherwise.

Compare to base R:

```
> df[df$var1 == value_to_keep & df$var2 > some_other_value & !is.na(df$keep_this_variable)]
```

`filter`ing rows in our data

What if we wanted to create a separate data frame with just Democrats?

`filter`ing rows in our data

What if we wanted to create a separate data frame with just Democrats?

```
> dems = dat %>%
+   filter(pid == "A Democrat")
> nrow(dat)

[1] 542

> nrow(dems)

[1] 175
```

`mutate`: Creating and Modifying Variables

The tidyverse function for creating/modifying variables in a data.frame is called `mutate`. It works like this:

```
> df %>%
+   mutate(variable = some_expression)
```

`mutate`: Creating and Modifying Variables

The tidyverse function for creating/modifying variables in a data.frame is called `mutate`. It works like this:

```
> df %>%
+   mutate(variable = some_expression)
```

For example:

```
> dat %>%
+   mutate(new_variable = "this is my new variable") %>%
+   select(pid, new_variable) %>%
+   head()
```

```
             pid            new_variable
1 An Independent this is my new variable
2 An Independent this is my new variable
3 An Independent this is my new variable
4     A Democrat this is my new variable
5   A Republican this is my new variable
6   A Republican this is my new variable
```

Creating and Modifying Variables

Let's create an indicator variable for being a Republican, Democrat, or Independent, using the `ifelse` function.

```
> dat <- dat %>%
+   mutate(dem = ifelse(pid == "A Democrat", 1, 0),
+          rep = ifelse(pid == "A Republican", 1, 0),
+          ind = ifelse(pid == "An Independent", 1, 0))
> dat %>%
+   select(pid, dem, ind, rep) %>%
+   head()

             pid dem ind rep
1 An Independent   0   1   0
2 An Independent   0   1   0
3 An Independent   0   1   0
4     A Democrat   1   0   0
5   A Republican   0   0   1
6   A Republican   0   0   1
```

Creating a Table of Responses

For surveys, it's often useful to look at the proportion of people who give different answers to a question. We can use the `table` and `prop.table` functions to do this quickly:

```
> table(dat$pid)

    A Democrat   A Republican An Independent        Skipped Something else
           175            143            185              2             37

> prop.table(table(dat$pid))

    A Democrat   A Republican An Independent        Skipped Something else
   0.322878229    0.263837638    0.341328413    0.003690037    0.068265683

> round(prop.table(table(dat$pid))*100, 1)

    A Democrat   A Republican An Independent        Skipped Something else
          32.3           26.4           34.1            0.4            6.8
```

To use weights, use the `wtd.table` function from the `Hmisc` package:

```
> library(Hmisc)
> wtd.table(dat$pid, dat$weight, type = "table")

    A Democrat   A Republican An Independent        Skipped Something else
      151.7603       140.9195       216.8010        1.0584        31.4615
```

Creating a Summary Data Frame

Ultimately we need to create a summary of the data we have.

- We've already seen some functions we can use to summarize survey data:
  - `mean()` – the average value
  - `weighted.mean()` – the average value after applying weights
  - `table()` – the number of rows that have each value of a variable
  - `prop.table()` – the *proportion* of rows that have each value of a variable
- We can use these functions directly, or use them within the tidyverse function `summarise()`

summarise

We can compute many summaries at once using the tidyverse function `summarise()`. This creates a new data frame with the summary values as the variables. The syntax is:

```
> df %>%
+   summarise(sum1 = function1(var1),
+             sum2 = function2(var2, var3),
+             ...)
```

For example:

```
> sum_df <- dat %>%
+   summarise(prop_dem1 = mean(dem),
+             prop_dem2 = mean(pid == "A Democrat"),
+             prop_dem_wtd = weighted.mean(dem, w = weight),
+             avg_age = mean(age),
+             sd_age = sd(age))
> sum_df

  prop_dem1 prop_dem2 prop_dem_wtd  avg_age   sd_age
1 0.3228782 0.3228782    0.2800002 53.63469 17.58695
```

Grouped Summaries

- So far, `summarise()` isn't any easier than writing the functions one-by-one
- Where tidyverse shinse is flexibility in generating grouped summaries
- `group_by()` creates a grouped data frame according to the variables you specify
- E.g. to create a grouped data frame by combinations of age and race, we can write:

```
> dat %>%
+   group_by(age, race) %>%
+   some_other_function()
```

Grouped Summaries

- So far, `summarise()` isn't any easier than writing the functions one-by-one
- Where tidyverse shinse is flexibility in generating grouped summaries
- `group_by()` creates a grouped data frame according to the variables you specify
- E.g. to create a grouped data frame by combinations of age and race, we can write:

```
> dat %>%
+   group_by(age, race) %>%
+   some_other_function()
```

- We can then use `summarise()` or `mutate()`, and it will automatically perform the operations within groups

Grouped Summaries

Let's get the proportion of people saying they were very enthusiastic about voting, according to what they think the most important issue is.

Start by figuring out the grouping variable:

```
> dat <- dat %>%
+   group_by(most_imp_iss)
```

Grouped Summaries

Let's get the proportion of people saying they were very enthusiastic about voting, according to what they think the most important issue is.

Start by figuring out the grouping variable:

```
> dat <- dat %>%
+   group_by(most_imp_iss)
```

Then figure out what summary you'd like:

```
> unique(dat$vote_enthusiasm)
```

```
[1] "Somewhat enthusiastic"   "Very enthusiastic"
[3] "Not so enthusiastic"     "Not enthusiastic at all"
[5] "Skipped"
```

Grouped Summarise

```
> enthusiasm <- dat %>%
+   group_by(most_imp_iss) %>%
+   summarise(very_enthusiastic = mean(vote_enthusiasm == "Very enthusiastic"),
+             n = n())
> enthusiasm %>%
+   arrange(desc(very_enthusiastic))

# A tibble: 13 × 3
   most_imp_iss                          very_enthusiastic     n
   <chr>                                             <dbl> <int>
 1 Abortion                                          0.609    64
 2 Immigration                                       0.581    31
 3 Taxes                                             0.571     7
 4 Inflation                                         0.533   107
 5 The economy                                       0.476   103
 6 Gas prices                                        0.474    38
 7 Something else                                    0.429    35
 8 Climate change                                    0.419    31
 9 The situation with Russia and Ukraine             0.4      10
10 Gun violence                                      0.394    94
11 The coronavirus pandemic                          0.375     8
12 Skipped                                           0.2       5
13 Crime                                             0         9
```

R code