PS 138Z - The Politics of Immigration: Section 4

2024-02-15

Introduction

We are now entering the more hands-on series of discussion sections. Throughout the following weeks, you will learn and practice basic skills for quantitative data analysis in political science.

First, we are going to explore an important source of data to understand Mexico-United States immigration and explore its structure and meaning through the codebook. A codebook goes through the process researchers undertake when assigning values to their data. It explains how researchers think about their data, code their variables, define them, and perform calculations. The codebook is the recipe, and the dataframe is the dish.

The **Mexican Migration Project (MMP) Database** describes patterns and processes of contemporary Mexican immigration to the United States. It contains data gathered since 1982 in surveys administered yearly in both countries. The surveys have the following study design:

- Each year, during the winter months (when seasonal migrants tend to return home), the MMP randomly samples households in communities throughout Mexico.
- Interviewers collect social, demographic, and economic information on the household and its members, basic immigration information on each person's first and last trip to the United States, and detailed year-by-year labor history and migration information from household heads and spouses.
- After completing the Mexican surveys, interviewers travel to destination areas in the U.S. to administer identical questionnaires to migrants from the same communities sampled in Mexico who have settled north of the border and no longer return home.

You can find more information about the dataset and the study design at `https://mmp.opr.princeton.edu/`.

The **PERS** file in this dataset is a person-level file that provides general demographic information and brief migration measures for each member of a surveyed household, including children of the head of household not presently residing with their parents.

The data you will be using for the section is available in the file `mmp_subset.csv`, which you can find on bCourses under Section Materials -> Data Sources. More experienced R users can download the data and perform tasks directly on their desktops using RStudio.

This file is a subset of the PERS file that includes variables related to information about Mexican and U.S. households surveyed since 2017. Most of the variables included in this file ask about the history of migration to the U.S. or return migration to Mexico.

Jupyter Notebook

A Jupyter Notebook is an online, interactive computing environment composed of different types of cells. Cells are chunks of code or text that are used to break up a larger notebook into smaller, more manageable parts and to let the viewer modify and interact with the elements of the notebook.

This notebook is an R Markdown (RMD) file consisting of 2 different kinds of cells: text and code. A text cell (like this one) contains text, while a code cell contains expressions in R, the programming language you will use.

For sections, I recommend that you take notes on each notebook as we go: use both code and text chunks. We will work with R Markdown files because they allow us to visualize code and the related output and take notes in a single document.

Today, we will address the following key questions:

- **What is a dataframe?**
- **What do columns and rows represent?**
- **What is a variable?**
- **What is an observation?**
- **What is an object?**
- **What is a command?**

Loading data and taking a first look

To begin, let's load the data using the `read.csv()` function:

```
mmp <- read.csv("mmp_subset.csv")
```

Note that there are different file types for datasets, each one requiring its own command (e.g., `read.xlsx`, `read.dta`). Now, store the dataframe into a new object called `df` using the code below:

```
df <- mmp
```

- **Why might we want to create objects?**

Let's take a look at the header of our dataframe using the command `head()`:

```
# head()
```

- **What argument should we add to the function if we only want to see the first three rows?**

If we want to view the entire dataframe, we can use the `print()` function:

```
#`print()
```

We can also use the `view()` function:

```
# view()
```

Finally, let's find the dimensions of the dataframe using the function `dim()`

```
# dim()
```

- **What do dimensions mean in this context?**
- **How many rows does the data set have? How many columns? What do these represent?**

In R, functions are blocks of code that perform a specific task or computation. The functions we used above are basic functions included in R, but we can also create our custom functions and assign them a name.

What is the difference between a variable and an observation?

An observation is a single data point containing information for a specific entity (e.g., an individual, household, neighborhood, county, state, or country) at a particular time (e.g., a day, month, or year). Each observation typically consists of several components or attributes, known as variables, representing different aspects or characteristics of the observed entity. In a dataframe, **each row corresponds to an observation and each column to a variable**.

Another way to find out the number of observations and variables in the dataframe is by using the `nrow()` and `ncol()` functions:

```
# nrow()
# ncol()
```

- **How many observations and variables you can identify?**
- **Do these numbers coincide with the values you got when using the `dim()` function?**

Let's dive into the MMP-PERS codebook Reading the codebook is vital to understand the meaning of the variable names and values. As mentioned above, a codebook describes each variable's name, label, data type, and values. It also includes information about how the variable was measured, how missing values are coded, and any special codes or categories used in the variable.

- **Why might researchers want to create a thorough and clear codebook?**

This dataframe contains many variables! Of these, we might only be interested in some. Today, we will focus on a few variables related to demographic information and characteristics of immigrants' first and last trips to the U.S.; after all, it is a politics of immigration class!

First, open the codebook. You can do this either on bCourses or the working directory set for this class on Jupyter Hub. Scroll down through the codebook and extract the following details:

- **What is the unit of observation?**
- **What type of data is this? Time series, panel, cross-sectional, pooled cross-sectional?**
- **Why does it matter for this study?**
- **How many people were surveyed?**

Remember that pooled cross-sectional data combines multiple cross-sectional datasets collected at different times, creating a "pooled" set of cross-sectional observations over several periods. Panel data consists of observations on a constant set of subjects over multiple time periods. The MMP-PERS data does not have a panel structure because it does not follow the same people over time. Every year, the data is collected by taking a different sample of communities and conducting household surveys.

Descriptive statistics

This exercise focuses on people migrating to the U.S. from Mexico. For this, we have created an indicator (i.e., dummy) variable that takes on a value of `1` for people who have migrated at least once into the U.S. and a value of `0` otherwise. We can use this variable to filter the data we are interested in:

```
library(tidyverse)
```

```
## Warning: package 'tibble' was built under R version 4.2.3
```

```
df.filter <- df %>% filter(us_immigrant == 1)
```

Note that we assigned the new filtered dataframe into a new object called `df.filter`. We will use this dataframe for the next steps in the data analysis.

Now that we know the connection between the codebook and the data, we are going to produce some descriptive statistics on key demographic variables. We already used the `head`, `print()`, `view()`, `nrow()`, and `ncol()` functions to inspect the original dataframe. Let's use the `str()` function to display the internal structure of the filtered dataframe in a compact way:

```
# str()
```

- **What type of variables does the database have?**
- **Are there potential problems with any variables?**

Missing values can pose challenges for data analysis. Depending on how they are coded in the dataframe, we can use different strategies to deal with them.

In R, missing values are commonly represented by `NA` (Not Available). Many R functions do not handle `NA` values by default and will return `NA` if the input data contains any missing values, leading to incomplete analyses or errors in results. Some functions like `mean()` have the argument `na.rm` (NA remove), which must be set to `TRUE` to ignore `NA` values in calculations.

Other times, numerical codes (for example, `8888` and `9999` in our dataframe) are used to represent missing values. In these cases, R can process data, but numerical calculations will be erroneous. Therefore, we must eliminate observations with missing values or recode them as `NA` and use the argument `na.rm = TRUE` in the corresponding function.

Now, go back to the codebook and identify the variables describing the **survey year** and participants' **age**, **sex**, *school years completed*, and **marital status**. We can use the functions `any()` and `is.na` to evaluate if these variables contain any missing values coded as `NA`. We must use the symbol `$` to select a specific variable from the dataframe. For example, to find out if the variable describing the year of the survey has missing values coded as `NA`, we can use the following code:

```
any(is.na(df.filter$surveyyr))
```

```
## [1] FALSE
```

The output is `FALSE`, so we can be sure that this variable has no missing values coded as `NA`. Try to understand why this is the case. Use a similar code to evaluate if the variables describing participants' age, sex, and marital status have missing values coded as `NA`.

```
# any(is.na())
# any(is.na())
# any(is.na())
# any(is.na())
```

Use the `table()` function to check how many individuals were surveyed each year, their age and schooling distribution, how many are female, how many are male, and how many belong to each marital status category.

```
# table()
# table()
# table()
# table()
# table()
```

- **What problem(s) can you see in the age and marital status tables? Is there any indication of missing values?**

Check the mean and range of participants' age and school years completed using the `mean()` and `range()` functions. Remember that you must use the `na.rm = TRUE` argument to avoid issues causes by missing values coded as `NA`.

```
mean(df.filter$age, na.rm = TRUE)
```

```
## [1] 50.18897
```

```
# mean()
# range()
# range()
```

- **Why are these calculations wrong if we used the argument `na.rm = TRUE`? How could we correct them?**

We can solve these issues by recoding replacing the `8888` value by `NA`:

```
df.filter$age[df.filter$age == 8888] <- NA
df.filter$marstat[df.filter$marstat == 8888] <- NA
df.filter$edyrs[df.filter$edyrs == 8888] <- NA
```

Let's produce now similar descriptive statistics on the first U.S. migration year but now use the functions `min()` and `max ()` instead of `range()`:

```
# min()
# max()
```

Finally, create a basic histogram to graphically represent this variable's distribution using the function `hist()`. You can customize the graph in multiple ways, but the function works well if you use `df.filter$usyr1` as the only argument.

```
# hist()
```

- **What conclusion can you draw from the histogram?**

Education and earnings among immigrants

What is the relationship between education and earnings among recent immigrants? We can approximate an answer to this question by estimating a correlation between two variables: *school years completed* (`edyrs`) and the **first wage after migration** (`uswage1`). We should start by replacing the `8888` and `9999` values by `NA` in the second variable:

```
df.filter$uswage1[df.filter$uswage1 == 8888] <- NA
df.filter$uswage1[df.filter$uswage1 == 9999] <- NA
```

Then, we can extract both variables and assign them to a new dataframe object using the functions `cbind()` and `as.data.frame()`:

```
corre.df <- as.data.frame(cbind(df.filter$edyrs, df.filter$uswage1))
```

We then use the function `na.omit()` to keep only the observations for which the **first wage after migration** has no missing values. Note that, for any giving observation, a missing value in one of the two variables is enough for the function to drop the entire row:

```
corre.df <- na.omit(corre.df)
```

Let's pause for a moment and check what are the variable names in this new dataframe using the de function `names()` with `corre.df` as the only argument:

```
# names()
```

Variables' names have changed, so we have to reassign them with the same function:

```
names(corre.df)[1] <- "edyrs"
names(corre.df)[2] <- "uswage1"
```

Finally, use the `cor()` function to calculate the correlation coefficient of interest. Remember that the symbol `$` allows you to select specific variables from the dataframe.

```
cor(corre.df$edyrs, corre.df$uswage1)
```

```
## [1] -0.01477224
```

- **What can we conclude from this coefficient?**