NC Voter File Intro

DPI 610

Ben Schneer

2026-01-29

```r
# Load required packages
library(tidyverse)
```

This exercise is designed to give you more experience with some of the basic functionality of **R** while also familiarizing yourself with a data set that we will be using in our upcoming problem set.

Pre-Lab Exercise (Complete Before Class)

Before coming to the lab session, complete the following exercise.

Step 1: Review the Data Dictionary

Open the file `nc_dictionary.csv` and familiarize yourself with the variables available in the NC voter file data.

```r
nc_dict <- read_csv("nc_dictionary.csv")
```

Step 2: Choose a Variable

Pick one variable from the data dictionary that you think might be useful for predicting voter turnout.

Step 3: Write a Brief Reflection (2 sentences max for each)

For the variable you chose, consider:

- **Predictive Value**: Why do you think this variable might (or might not) be correlated with voter turnout?
- **Measurement Quality**: How confident are you that this variable is accurately measured? What are potential limitations?

Answer to Pre-Lab Exercise

Variable chosen: college_grad_prob

Predictive Value: According to Pew Research Center (2025), Almost half of nonvoters (48%) had only a high school education or less, compared with 28% among voters. College graduates made up about twice the share of voters (41%) as nonvoters (22%).

Measurement Quality: While I know it's possible to certify whether someone graduated college, I'm not sure how this would work at scale. While it would be easy to get info from self-reports (i.e., Facebook status or LinkedIn) there's the chance someone lies and that's only if we can find information about someone.

Load data

First, we use `read_csv()` from the readr package (part of tidyverse) to load external data. For this project, let's load in some (synthetic) NC voter file data from a vendor (file is named `nc.csv`), along with a dictionary file that describes the variables (`nc_dictionary.csv`).

In **R**, we create a new object in the environment by using the `<-` symbol followed by a value, vector or data.

```r
# load data using read_csv() function from readr
nc_data <- read_csv("nc.csv")
```

Question 1

Following the above code, load the dataset `nc_dictionary.csv` onto the environment, and assign the data to a new object named `dictionary`.

Answer 1

```
# Insert answer here
dictionary <- read_csv("nc_dictionary.csv")
```

Access Variables in Data Frames

Typically, a dataset consists of multiple variables, and each column corresponds to a variable. With tidyverse, we can examine our data using `glimpse()`:

```
#look at the data! glimpse() is the function; nc_data is the object
glimpse(nc_data)
```

```
Rows: 10,402
Columns: 25
$ id                <dbl> 36974, 37362, 36635, 38352, 36619, 37731, 38205, 380~
$ state             <chr> "NC", "NC", "NC", "NC", "NC", "NC", "NC", "NC", "NC"~
$ age               <dbl> 30, 58, 65, 65, 52, 40, 55, 64, 72, 65, 47, 87, 74, ~
$ gender            <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F~
$ party             <chr> "DEM", "DEM", "DEM", "DEM", "DEM", "DEM", "DEM", "DE~
$ race              <chr> "C", "C", "C", "C", "C", "C", "C", "C", "C", "C", "C~
$ socioecon_bg      <chr> "VERY WEALTHY", "VERY WEALTHY", "VERY WEALTHY", "VER~
$ familyincome      <dbl> 135000, 83000, 73000, 176000, 138000, 137000, 100000~
$ density           <chr> "URBAN", "URBAN", "URBAN", "URBAN", "RURAL", "URBAN"~
$ ever_donor        <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1~
$ voted2008         <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1~
$ voted2010         <dbl> 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1~
$ voted2012         <dbl> 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ voted2014         <dbl> 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1~
$ vote2012_prob     <dbl> 49.5, 98.8, 93.7, 97.3, 91.0, 98.1, 80.5, 64.8, 46.2~
$ vote2014_prob     <dbl> 16.63, 84.80, 87.73, 34.64, 78.03, 84.39, 57.60, 64.~
$ ideology_score    <dbl> 80.1, 31.1, 38.8, 63.5, 72.5, 69.2, 67.1, 56.5, 62.2~
$ dem_score         <dbl> 70.65, 75.18, 69.74, 73.75, 94.15, 69.74, 83.76, 44.~
$ dem2012_score     <dbl> 92.67, 70.16, 85.89, 97.60, 96.62, 88.55, 92.80, 88.~
$ homeowner_score   <dbl> 9, 9, 7, 9, 9, 9, 9, 9, 9, 5, 9, 9, 9, 9, 9, 8, 9, 9~
$ college_grad_prob <dbl> 0.857, 0.697, 0.451, 0.650, 0.840, 0.730, 0.496, 0.6~
$ gun_owner_prob    <dbl> 0.140, 0.168, 0.206, 0.267, 0.176, 0.249, 0.160, 0.2~
$ hunter_prob       <dbl> NA, 0.046, 0.046, 0.231, 0.046, 0.046, 0.205, 0.046,~
$ married_prob      <dbl> 0.596, 0.173, 0.760, 0.726, 0.272, 0.957, 0.376, 0.6~
$ religion          <chr> "UNCODED CHRISTIAN", "CATHOLIC", "UNCODED", "PROTEST~
```

We can access individual variables using the select() function. For example, to access the party variable:

```
# use slice_head() to show just first five elements
# use the %>% "pipe" to perform multiple commands on an object
nc_data %>%
  select(party) %>%
  slice_head(n = 5)
```

```
# A tibble: 5 x 1
  party
  <chr>
1 DEM
2 DEM
3 DEM
4 DEM
5 DEM
```

Question 2

Examine the structure of dataset `dictionary` using `View()` to see descriptions of the variables included in the dataset.

Answer 2

```
# Insert answer here
view(dictionary)
```

Question 3

Access the variable named `age` in dataset `nc_data` and show the first five elements.

Answer 3

```
# Insert answer here
nc_data %>%
  select(age) %>%
  slice_head(n = 5)
```

```
# A tibble: 5 x 1
    age
  <dbl>
1    30
2    58
3    65
4    65
5    52
```

More on Functions

Many commands in R take the form of functions. Functions save time since we can write one function and then call it multiple times, rather than writing the same code over and over.

Functions allow us to:

- Create objects
- Perform complex calculations
- Estimate statistical models
- Create graphs & maps
- Generate output (e.g. tables)

Some useful starter functions from base R:

- `c()`
- `length()` & `dim()`
- `seq()` & `rep()`
- `sum()`, `mean()`
- `min()`, `max()`
- `nchar()`, `length()`
- `nrow()`, `ncol()`

```r
# Add numbers
sum(1, 2, 4, 5)
```

```
[1] 12
```

```r
# Create a sequence of numbers from 1 to 10
seq(1, 10)
```

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

```r
# Create a sequence of numbers from 0 to 10 by 2
seq(0, 10, 2)
```

```
[1]  0  2  4  6  8 10
```

```r
# Create a vector of characters
c("A", "B", "C") # Create a list or vector
```

```
[1] "A" "B" "C"
```

Some useful tidyverse functions:

- `tibble()`: Create tibble to store data
- `select()`: Choose columns
- `filter()`: Choose rows
- `mutate()`: Create or modify columns
- `group_by()`: Group data
- `summarize()`: Summarize data
- `arrange()`: Sort data
- `pull()`: Pull the values of a variable in a tibble out as a vector

```r
# Create a tibble (tidyverse's enhanced data frame)
example_tibble <- tibble(
  letters = c("A", "B", "C"),
  numbers = c(2,1,3)
)

#Calculate the mean of the numbers variable
example_tibble %>%
  summarize(mean_numbers = mean(numbers))
```

```
# A tibble: 1 x 1
  mean_numbers
         <dbl>
1            2
```

```r
#See the number of rows
example_tibble %>%
  nrow()
```

```
[1] 3
```

Question 4

The data `nc_data` includes a variable `age`. What is the oldest person in the data set?

Answer 4

```r
# Insert answer here
max(nc_data$age)
```

```
[1] 104
```

Working with Data Frames

R and tidyverse provide powerful tools for working with data frames. Let's look at some common operations:

To filter rows based on a condition:

```r
old_people <- nc_data %>%
  filter(age > 80)
```

To select specific columns:

```r
id_ages <- nc_data %>%
  select(id, age)

id_ages %>%
  slice_head(n = 5)
```

```
# A tibble: 5 x 2
     id   age
  <dbl> <dbl>
1 36974    30
2 37362    58
3 36635    65
4 38352    65
5 36619    52
```

Question 5

How many people are over the age of 90 in the data set?

Answer 5

```
# Insert answer here
nc_data %>%
  filter(age > 90) %>%
  count()
```

```
# A tibble: 1 x 1
      n
  <int>
1    68
```

We can also sort data using the function `arrange()`:

Question 6

Can you sort the ages of the first 10 people in the data set in order of increasing magnitude? How would you reorder the entire dataset by age?

Answer 6

```
# Insert answer here

nc_data %>%
  head(n = 10) %>%
  arrange(age)
```

```
# A tibble: 10 x 25
      id state   age gender party race  socioecon_bg        familyincome density
   <dbl> <chr> <dbl> <chr>  <chr> <chr> <chr>                      <dbl> <chr>
 1 36974 NC       30 F      DEM   C     VERY WEALTHY              135000 URBAN
 2 37731 NC       40 F      DEM   C     VERY WEALTHY              137000 URBAN
 3 36619 NC       52 F      DEM   C     MODERATELY WEALTHY        138000 RURAL
 4 38205 NC        55 F      DEM   C     MODERATELY WEALTHY        100000 URBAN
 5 37362 NC        58 F      DEM   C     VERY WEALTHY               83000 URBAN
 6 38014 NC        64 F      DEM   C     VERY WEALTHY              125000 URBAN
 7 36635 NC        65 F      DEM   C     VERY WEALTHY               73000 URBAN
 8 38352 NC        65 F      DEM   C     VERY WEALTHY              176000 URBAN
 9 38221 NC        65 F      DEM   C     NON-FAMILY, WORKING        40000 URBAN
10 36852 NC        72 F      DEM   C     MODERATELY WEALTHY         56000 URBAN
# i 16 more variables: ever_donor <dbl>, voted2008 <dbl>, voted2010 <dbl>,
#   voted2012 <dbl>, voted2014 <dbl>, vote2012_prob <dbl>, vote2014_prob <dbl>,
#   ideology_score <dbl>, dem_score <dbl>, dem2012_score <dbl>,
#   homeowner_score <dbl>, college_grad_prob <dbl>, gun_owner_prob <dbl>,
#   hunter_prob <dbl>, married_prob <dbl>, religion <chr>
```

Modifying Data Frames and Variables

With tidyverse, we use `mutate()` to create or modify variables. For example, we can create a new variable that standardizes age in standard deviations from its mean:

There are three key steps to creating a new function:

1. You need to pick a name for the function.
2. You list the inputs, or arguments, to the function inside function; function(x, y, z).
3. You place the code you have developed in body of the function, a { block that immediately follows function(…).

```
# we are overwriting the old data with the new version, with the additional variable!
# use the mutate() function to create new variables!
nc_data <- nc_data %>%
  mutate(age_std = (age - mean(age))/sd(age))
```

We can also write our own functions that can be applied to objects or to variables themselves.

Question 7

Can you write a function that standardizes any numeric variable as we did with age? After writing it, use it to define a new variable `age_std2` in `nc_data`.

Answer 7

```r
# Insert answer here

std_any <- function(x) {
  (x - mean(x))/sd(x)
}

nc_data %>%
  mutate(age_std2 = std_any(age)) %>%
  select(age_std, age_std2)
```

```
# A tibble: 10,402 x 2
   age_std age_std2
     <dbl>    <dbl>
 1 -1.48    -1.48
 2  0.279    0.279
 3  0.719    0.719
 4  0.719    0.719
 5 -0.0979  -0.0979
 6 -0.852   -0.852
 7  0.0907   0.0907
 8  0.657    0.657
 9  1.16     1.16
10  0.719    0.719
# i 10,392 more rows
```

Question 8

Create a new variable that counts the number of times people voted between 2008 and 2014 using the variables `voted2008`, `voted2010`, `voted2012`, and `voted2014`.

Answer 8

```r
# Insert answer here

elections <- c("voted2008", "voted2010", "voted2012", "voted2014")

nc_data %>%
  mutate(voted = rowSums(across(elections))) %>%
  select(all_of(elections), voted)
```

```
# A tibble: 10,402 x 5
   voted2008 voted2010 voted2012 voted2014 voted
       <dbl>     <dbl>     <dbl>     <dbl> <dbl>
 1         1         0         0         0     1
 2         1         1         1         1     4
 3         1         1         1         1     4
 4         1         1         0         1     3
 5         1         1         1         1     4
 6         1         1         1         1     4
 7         1         0         1         0     2
 8         1         0         1         1     3
 9         0         0         1         0     1
10         1         1         1         1     4
# i 10,392 more rows
```

Summarizing Data by Groups

R makes it easy to group and summarize data using `group_by()` and `summarize()`. For example, to see what average turnout was for 2008 by gender:

```r
nc_data %>%
  group_by(gender) %>%
  summarize(avg_turnout_2008 = mean(voted2008))
```

```
# A tibble: 2 x 2
  gender avg_turnout_2008
  <chr>             <dbl>
1 F                 0.786
2 M                 0.749
```

Question 9

Now determine average turnout in 2008 by gender *and* party. Which gender/party combination has the highest turnout rate?

Answer 9

```r
# Insert answer here

nc_data %>%
  group_by(gender, party) %>%
  summarize(avg_turnout_2008 = mean(voted2008)) %>%
  arrange(desc(avg_turnout_2008))
```

```
# A tibble: 8 x 3
# Groups:   gender [2]
  gender party avg_turnout_2008
  <chr>  <chr>            <dbl>
1 F      REP              0.816
2 F      DEM              0.814
3 M      REP              0.791
4 M      DEM              0.756
5 F      NPA              0.688
6 M      NPA              0.683
7 M      LIB              0.667
8 F      LIB              0.615
```

- Female Republicans had the highest turnout in the 2008 election at 81.6%

Question 10

Super Challenge Question: Write a function that takes as an input a factor that is a voter characteristic (e.g., party, gender, race, etc.) and a turnout rate and then, for that characteristic, identifies the sub-group with the highest level of turnout.

Use this function to return the socioeconomic group `socioecon_bg` with the highest level of turnout in 2014

Two hints: (1) Make sure the function takes your data object / tibble as an input (2) To refer to a variable within a function, use double curly brackets `{ }`.

Answer 10

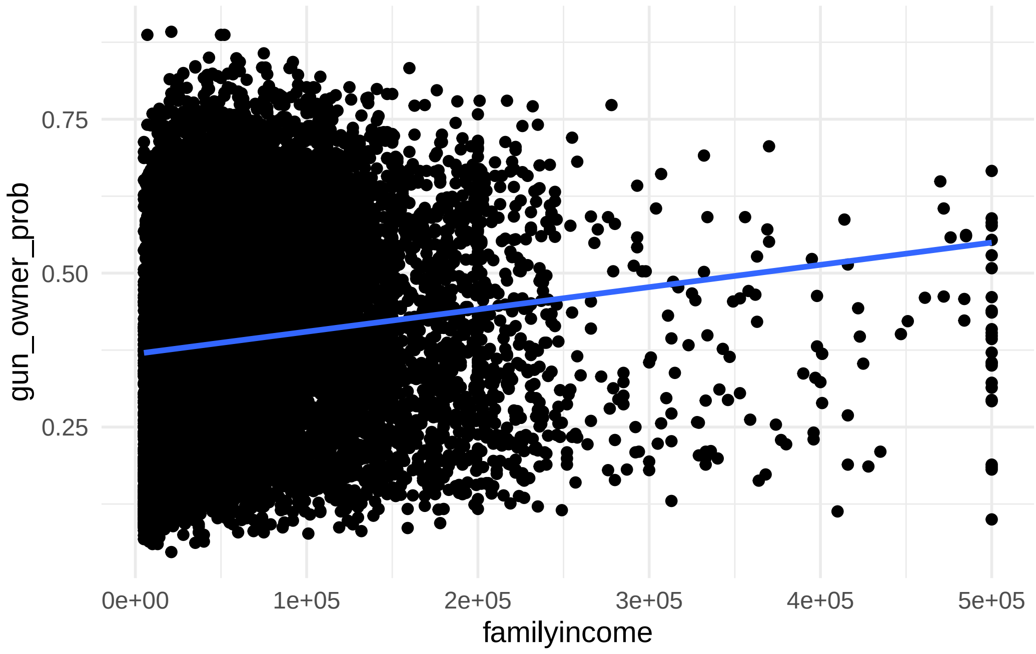
```r
# Insert answer here

highest_turnout <- function(df, voter_char, election) {
  df %>%
    group_by({{ voter_char }}) %>%
    summarize(avg_turnout_election = mean({{ election }})) %>%
    slice_max(avg_turnout_election, n = 1)
}

highest_turnout(nc_data, socioecon_bg, voted2014)
```

```
# A tibble: 1 x 2
  socioecon_bg                           avg_turnout_election
  <chr>                                                 <dbl>
1 TRADITIONALLY LIBERAL, WHIE COLLAR                    0.696
```

Question 11

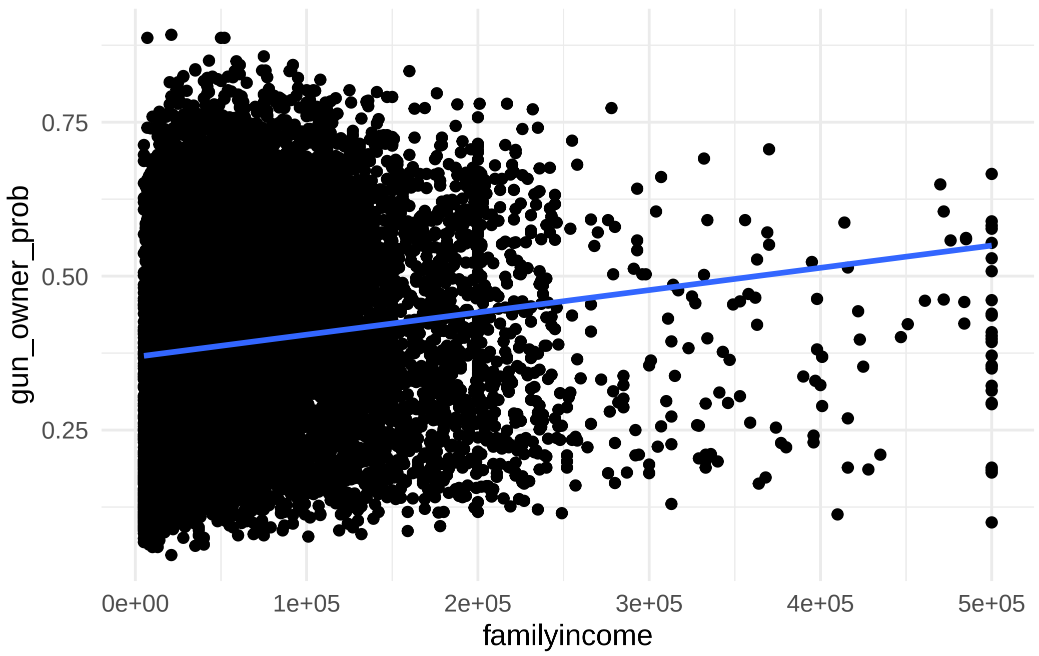
Super Challenge Question: Edit the code below, which makes a graph showing the relationship between `familyincome` and `gun_owner_prob`, so that it takes as an argument any of the several numeric variables in the data and creates a scatter plot with `familyincome` on the x-axis and the variable you input on the y-axis.

```r
# call ggplot for graphics; aes(x=...,y=...) specifies variables
# geom_point() plots the data
# geom_smooth() plots line of best fit (e.g., regression)
# theme_minimal() chooses a minimal theme
nc_data %>%
  ggplot(aes(x=familyincome,y=gun_owner_prob)) +
  geom_point() +
  geom_smooth(method = "lm",se=F) +
  theme_minimal()
```



Answer 11

```r
# Insert answer here

income_any <- function(y) {
  nc_data %>%
    ggplot(aes(x=familyincome, y={{ y }})) +
    geom_point() +
    geom_smooth(method = "lm",se=F) +
    theme_minimal()
}

income_any(gun_owner_prob)
```


gun_owner_prob
0.25
0.50
0.75
0e+00
1e+05
2e+05
3e+05
4e+05
5e+05
familyincome