Descriptive Analysis

R for Advanced Stata Users

Luiza Andrade, Leonardo Viotti, Rob Marty, Rony Rodriguez, Luis Eduardo San Martin
DIME |The World Bank

30 November 2020

Table of contents

1. Quick summary statistics
2. Descriptive statistics tables
3. Exporting descriptive statistics tables
4. Formatting tables
5. Aggregating observations
6. Running regressions
7. Exporting regression tables

Workflows for outputs

Not reproducible

Anything that requires

Copy-pasting

Manual formatting after exported

Reproducible

R Markdown: dynamic document containing code and text that is exported directly from R into PDF, HTML, Word, Power Point and other formats

LaTeX: typesetting system used for scientific publications that automatically reloads tables and figures every time the document is rendered

Setting the stage

Set folder paths to your computer

```
# Set folder paths
projectFolder ← file.path("EDIT/THIS/FILE/PATH/dime-r-training")


dataWorkFolder    ← file.path(projectFolder,"DataWork")
Data              ← file.path(dataWorkFolder,"DataSets")
finalData         ← file.path(Data,"Final")
rawOutput         ← file.path(dataWorkFolder,"Output","Raw")
```

Load the packages that we will use today

```
# Load packages
packages ← c("tidyverse", "skimr", "huxtable", "lfe")
pacman::p_load(packages, character.only = TRUE)
```

Load the data that we will use today: Stata's `census` dataset

```
# Load data
 census ←
  readRDS(file.path(finalData, "census.RDS"))
```

Taking a peek at the data

```r
str(census)
```

```
## 'data.frame':    50 obs. of  13 variables:
##  $ state   : chr  "Alabama" "Alaska" "Arizona" "Arkansas"  ...
##  $ state2  : chr  "AL" "AK" "AZ" "AR"  ...
##  $ region  : Factor w/ 4 levels "NE","N Cntrl",..: 3 4 4 3 4 4 1 3 3 3  ...
##  $ pop     : int  3893888 401851 2718215 2286435 23667902 2889964 3107576 594338 9746324 5463105  ...
##  $ poplt5  : int  296412 38949 213883 175592 1708400 216495 185188 41151 570224 414935  ...
##  $ pop5_17 : int  865836 91796 577604 495782 4680558 592318 637731 125444 1789412 1231195  ...
##  $ pop18p  : int  2731640 271106 1926728 1615061 17278944 2081151 2284657 427743 7386688 3816975  ...
##  $ pop65p  : int  440015 11547 307362 312477 2414250 247325 364864 59179 1687573 516731  ...
##  $ popurban: int  2337713 258567 2278728 1179556 21607606 2329869 2449774 419819 8212385 3409081  ...
##  $ medage  : num  29.3 26.1 29.2 30.6 29.9  ...
##  $ death   : int  35305 1604 21226 22676 186428 18925 26005 5123 104190 44230  ...
##  $ marriage: int  49018 5361 30223 26513 210864 34917 26048 4437 108344 70638  ...
##  $ divorce : int  26745 3517 19908 15882 133541 18571 13488 2313 71579 34743  ...
##  - attr(*, "datalabel")= chr "1980 Census data by state"
##  - attr(*, "time.stamp")= chr "11 Nov 2020 18:02"
##  - attr(*, "formats")= chr [1:13] "%-14s" "%-2s" "%-8.0g" "%12.0gc"  ...
##  - attr(*, "types")= int [1:13] 14 2 65529 65528 65528 65528 65528 65528 65528 65527  ...
##  - attr(*, "val.labels")= Named chr [1:13] "" "" "cenreg" ""  ...
##   ..- attr(*, "names")= chr [1:13] "" "" "cenreg" ""  ...
##  - attr(*, "var.labels")= chr [1:13] "State" "Two-letter state abbreviation" "Census region" "Population"  ...
##  - attr(*, "version")= int 118
##  - attr(*, "label.table")=List of 1
##   ..$ cenreg: Named int [1:4] 1 2 3 4
##   .. ..- attr(*, "names")= chr [1:4] "NE" "N Cntrl" "South" "West"
##  - attr(*, "expansion.fields")= list()
##  - attr(*, "byteorder")= chr "LSF"
##  - attr(*, "orig.dim")= int [1:2] 50 13
```

Quick summary statistics

Exploring a dataset

`summary(x, digits)`

Equivalent to Stata's `codebook`. Its arguments are:

- **x:** the object you want to summarize, usually a vector or data frame
- *digits:* the number of decimal digits to be displayed

Exercise ⌨

Use the `summary()` function to describe the `census` data frame.

Exploring a dataset

```
summary(census)
```

```
##     state              state2             region        pop          
##  Length:50          Length:50          NE     : 9   Min.   :  401851  
##  Class :character   Class :character   N Cntrl:12   1st Qu.: 1169218  
##  Mode  :character   Mode  :character   South  :16   Median : 3066433  
##                                        West   :13   Mean   : 4518149  
##                                                     3rd Qu.: 5434033  
##                                                     Max.   :23667902  
##      poplt5           pop5_17            pop18p             pop65p       
##  Min.   :  35998   Min.   :  91796   Min.   :  271106   Min.   :  11547  
##  1st Qu.:  98831   1st Qu.: 257949   1st Qu.:  823702   1st Qu.: 118660  
##  Median : 227468   Median : 629654   Median : 2175130   Median : 370495  
##  Mean   : 326278   Mean   : 945952   Mean   : 3245920   Mean   : 509503  
##  3rd Qu.: 361321   3rd Qu.:1143292   3rd Qu.: 3858173   3rd Qu.: 580087  
##  Max.   :1708400   Max.   :4680558   Max.   :17278944   Max.   :2414250  
##     popurban            medage          death            marriage     
##  Min.   :  172735   Min.   :24.20   Min.   :  1604   Min.   :  4437  
##  1st Qu.:  826651   1st Qu.:28.73   1st Qu.:  9087   1st Qu.: 14840  
##  Median : 2156905   Median :29.75   Median : 26176   Median : 36279  
##  Mean   : 3328253   Mean   :29.54   Mean   : 39474   Mean   : 47701  
##  3rd Qu.: 3403450   3rd Qu.:30.20   3rd Qu.: 46532   3rd Qu.: 57338  
##  Max.   :21607606   Max.   :34.70   Max.   :186428   Max.   :210864  
##     divorce      
##  Min.   :  2142  
##  1st Qu.:  6898  
##  Median : 17112  
##  Mean   : 23679  
##  3rd Qu.: 27986  
##  Max.   :133541
```

Summarizing continuous variables

- `summary()` can also be used with a single variable.
- When used with continuous variables, it works similarly to `summarize` in Stata.
- When used with categorical variables, it works similarly to `tabulate`.

Summarizing continuous variables

Exercise ⌨

Use the `summary()` function to display summary statistics for a continuous variable in the `census` data frame.

```
summary()
```

Summarizing continuous variables

Exercise ⌨

Use the `summary()` function to display summary statistics for a continuous variable in the `census` data frame.

```
summary(census$pop)
```

```
##      Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##   401851  1169218  3066433  4518149  5434033 23667902
```

Summarizing categorical variables

`table()`

Equivalent to `tabulate` in Stata, creates a frequency table. Its main arguments are vectors to be tabulated.

Exercise ⌨

Use the `table()` function to display frequency tables for:

1. The variable `region` in the `census` data frame
2. The variables `region` and `state` in the `census` data frame, simultaneously

Summarizing categorical variables

One way tabulation

```
table()
```

Summarizing categorical variables

One way tabulation

```
table(census$region)
```

```
##
##       NE N Cntrl   South    West
##        9      12      16      13
```

Two way tabulation

```
table()
```

Summarizing categorical variables

```r
table(census$region, census$state)
```

```
##
##            Alabama Alaska Arizona Arkansas California Colorado Connecticut
##   NE             0      0       0        0          0        0           1
##   N Cntrl        0      0       0        0          0        0           0
##   South          1      0       0        1          0        0           0
##   West           0      1       1        0          1        1           0
##
##            Delaware Florida Georgia Hawaii Idaho Illinois Indiana Iowa Kansas
##   NE              0       0       0      0     0        0       0    0      0
##   N Cntrl         0       0       0      0     0        1       1    1      1
##   South           1       1       1      0     0        0       0    0      0
##   West            0       0       0      1     1        0       0    0      0
##
##            Kentucky Louisiana Maine Maryland Massachusetts Michigan Minnesota
##   NE              0         0     1        0             1        0         0
##   N Cntrl         0         0     0        0             0        1         1
##   South           1         1     0        1             0        0         0
##   West            0         0     0        0             0        0         0
```

Descriptives tables

Descriptives tables

What if you want to...

- ...export the summary statistics to another software?
- ...customize which statistics to display?
- ...format the table?

Well, then you will need to go beyond base R

- There are many packages that can be used both for displaying and exporting summary statistics
- Today we will show you a combination of two packages: `skimr` and `huxtable`
- We chose this combination because together, they can perform all the tasks we are interested in

Exploring datasets with *skimr*

- The `skimr` package features are very similar to those of the functions `summary`.
- It is used to present summary statistics for a dataset.
- Like `summary`, the statistics presented vary with the class of each variable.
- `skimr`'s main function is called `skim()`, and its syntax is also very similar to `summary`.

Exploring datasets with *skimr*

```
skim(census)
```

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
state	0	1	4	13	0	50	0
state2	0	1	2	2	0	50	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
region	0	1	FALSE	4	Sou: 16, Wes: 13, N C: 12, NE: 9

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
pop	0	1	4518149.44	4715037.75	401851.0	1169218.50	3066433.00	5434033.25	23667902.0	▇▂▁▁▁
poplt5	0	1	326277.78	331585.14	35998.0	98831.00	227467.50	361321.25	1708400.0	▇▁▁▁▁
pop5_17	0	1	945951.60	959372.83	91796.0	257948.75	629654.00	1143292.50	4680558.0	▇▂▁▁▁
pop18p	0	1	3245920.06	3430531.31	271106.0	823701.50	2175130.00	3858173.25	17278944.0	▇▂▁▁▁
pop65p	0	1	509502.80	538932.38	11547.0	118660.00	370495.00	580087.25	2414250.0	▇▂▁▁▁

Exploring datasets with *skimr*

The main advantage of `skimr` is that it is designed to fit well with the `tidyverse` syntax and within a data pipeline.

So, for example, if you only want to summarize a few variables, you can write the following:

```
census %>%
    skim(pop,
         popurban,
         medage,
         death,
         marriage,
         divorce)
```

Exploring datasets with *skimr*

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
pop	0	1	4518149.44	4715037.75	401851.0	1169218.50	3066433.00	5434033.25	23667902.0	▇▂▁▁▁
popurban	0	1	3328253.18	4090177.93	172735.0	826651.00	2156905.00	3403449.50	21607606.0	▇▁▁▁▁
medage	0	1	29.54	1.69	24.2	28.73	29.75	30.20	34.7	▁▂▇▂▁
death	0	1	39474.26	41742.35	1604.0	9087.00	26176.50	46532.50	186428.0	▇▂▁▁▁
marriage	0	1	47701.40	45130.42	4437.0	14839.50	36279.00	57338.25	210864.0	▇▂▂▁▁
divorce	0	1	23679.44	25094.01	2142.0	6897.50	17112.50	27986.50	133541.0	▇▂▁▁▁

Customizing *skimr*

You can also create your own *skimr function list (sfl)* for each class of variables.

```
summary_stats ←
  skim_with(numeric = sfl(Mean = mean, # Variable name = statistic
                          Median = median,
                          SD = sd,
                          Min = min,
                          Max = max),
            append = FALSE) # Remove all default statistics

census %>%
    summary_stats()
```

Here are a few functions that can be used within `sfl()`:

- Center: `mean()`, `median()`
- Spread: `sd()`, `IQR()`, `mad()`
- Range: `min()`, `max()`, `quantile()`
- Position: `first()`, `last()`, `nth()`,
- Count: `n()`, `n_distinct()`
- Logical: `any()`, `all()`

Customizing *skimr*

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
state	0	1	4	13	0	50	0
state2	0	1	2	2	0	50	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
region	0	1	FALSE	4	Sou: 16, Wes: 13, N C: 12, NE: 9

Variable type: numeric

skim_variable	n_missing	complete_rate	Mean	Median	SD	Min	Max
pop	0	1	4518149.44	3066433.00	4715037.75	401851.0	23667902.0
poplt5	0	1	326277.78	227467.50	331585.14	35998.0	1708400.0
pop5_17	0	1	945951.60	629654.00	959372.83	91796.0	4680558.0
pop18p	0	1	3245920.06	2175130.00	3430531.31	271106.0	17278944.0
pop65p	0	1	509502.80	370495.00	538932.38	11547.0	2414250.0
popurban	0	1	3328253.18	2156905.00	4090177.93	172735.0	21607606.0
medage	0	1	29.54	29.75	1.69	24.2	34.7

Customizing *skimr*

```
census %>%
  summary_stats() %>%
  yank("numeric") %>% # keep only numeric variables on the table
  select(-n_missing, -complete_rate) # remove default statistics
```

Variable type: numeric

skim_variable	Mean	Median	SD	Min	Max
pop	4518149.44	3066433.00	4715037.75	401851.0	23667902.0
poplt5	326277.78	227467.50	331585.14	35998.0	1708400.0
pop5_17	945951.60	629654.00	959372.83	91796.0	4680558.0
pop18p	3245920.06	2175130.00	3430531.31	271106.0	17278944.0
pop65p	509502.80	370495.00	538932.38	11547.0	2414250.0
popurban	3328253.18	2156905.00	4090177.93	172735.0	21607606.0
medage	29.54	29.75	1.69	24.2	34.7
death	39474.26	26176.50	41742.35	1604.0	186428.0
marriage	47701.40	36279.00	45130.42	4437.0	210864.0
divorce	23679.44	17112.50	25094.01	2142.0	133541.0

Exporting tables

Exporting tables

To export the tables to a different software, we will need a different package, `huxtable`. The easiest way to save tables is through this family of commands:

quick_latex(..., file)
quick_pdf(..., file)
quick_html(..., file)
quick_docx(..., file)
quick_pptx(..., file)
quick_xlsx(..., file)
quick_rtf(..., file)

- **...**: the huxtable objects or data frames to be exported
- *file*: the file path to where the table should be saved, including the file extension

Exporting tables

The code below exports the table we just created to Excel and LaTeX

```
# Store table so it can be exported twice
summary_stats_table <-
  census %>%
  summary_stats() %>%
  yank("numeric") %>% # keep only numeric variables on the table
  select(-n_missing, -complete_rate) # remove daulft statistics

# Export to Excel
quick_xlsx(summary_stats_table,
           file = file.path(rawOutput, "summary-stats.xlsx"))

# Export to LaTeX
quick_latex(summary_stats_table,
            file = file.path(rawOutput, "summary-stats.tex"))
```

Formatting tables

Beautifying tables

`huxtable` also allows you to edit your table as a data frame in R, and set the formatting so it can be exported with the same layout to multiple software. The code below shows how to do edit `summary_stats_table`

```
# Extract variable labels from data frame
census_dictionary <-
  data.frame("Variable" = attributes(census)$var.labels,
             "name" = names(census))
```

Beautifying tables

`huxtable` also allows you to edit your table as a data frame in R, and set the formatting so it can be exported with the same layout to multiple software. The code below shows how to do edit `summary_stats_table`

```r
# Extract variable labels from data frame
census_dictionary <-
  data.frame("Variable" = attributes(census)$var.labels,
             "name" = names(census))

summary_stats_table <-
   summary_stats_table %>%
   rename(name = skim_variable) %>% # Rename var with var names so we can merge the datasets
   left_join(census_dictionary) %>% # Merge to variable labels
   select(-name) %>% # Keep only variable labels instead of names
   as_hux # Convert it into a huxtable object
```

Beautifying tables

`huxtable` also allows you to edit your table as a data frame in R, and set the formatting so it can be exported with the same layout to multiple software. The code below shows how to do edit `summary_stats_table`

```
# Extract variable labels from data frame
census_dictionary ←
  data.frame("Variable" = attributes(census)$var.labels,
             "name" = names(census))

summary_stats_table ←
   summary_stats_table %>%
   rename(name = skim_variable) %>% # Rename var with var names so we can merge the datasets
   left_join(census_dictionary) %>% # Merge to variable labels
   select(-name) %>% # Keep only variable labels instead of names
   as_hux # Convert it into a huxtable object


summary_stats_table ←
   summary_stats_table %>%
   relocate(Variable) %>%  # Make variable labels the first column
   set_header_rows(1, TRUE) %>% # Use stats name as table header
   set_header_cols("Variable", TRUE)  %>%  # Use variable name as row header
   theme_basic() # Set a theme for quick formatting
```

Beautifying tables

`huxtable` also allows you to edit your table as a data frame in R, and set the formatting so it can be exported with the same layout to multiple software. The code below shows how to do edit `summary_stats_table`

```r
# Extract variable labels from data frame
census_dictionary <-
  data.frame("Variable" = attributes(census)$var.labels,
             "name" = names(census))

summary_stats_table <-
   summary_stats_table %>%
   rename(name = skim_variable) %>% # Rename var with var names so we can merge the datasets
   left_join(census_dictionary) %>% # Merge to variable labels
   select(-name) %>% # Keep only variable labels instead of names
   as_hux # Convert it into a huxtable object


summary_stats_table <-
   summary_stats_table %>%
   relocate(Variable) %>%  # Make variable labels the first column
   set_header_rows(1, TRUE) %>% # Use stats name as table header
   set_header_cols("Variable", TRUE)  %>%  # Use variable name as row header
   theme_basic() # Set a theme for quick formatting

# Now export it
quick_xlsx(summary_stats_table,
           file = file.path(rawOutput, "summary-stats-basic.xlsx"))

quick_latex(summary_stats_table,
           file = file.path(rawOutput, "summary-stats-basic.tex"))
```

Other themes to play with

Aggregating observations

Aggregating observations

- If you want to show aggregated statistics, the function `summarise` is a powerful tool.
- It is similar to `skim` in that it calculates a series of statistics for a data frame.
- However, it does not have pre-defined statistics, so it requires more manual input.
- On the other hand, its output is a regular data frame, so it is also useful to create constructed data sets.
- Its Stata equivalent would be `collapse`

```
summarise(.data, ... ,)
```

- **data**: the data frame to be summarized
- **...**: Name-value pairs of summary functions. The name will be the name of the variable in the result.

The "name-value" pairs mentioned under `...` look like this: `new_variable = stat(existing_variable)`, where `stat` takes the same functions as `sfl`

Aggregating observations

```
region_stats ←
  census %>%
  group_by(region) %>%
  summarise(`Number of States` = n_distinct(state),
            `Total Population` = sum(pop))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

region	Number of States	Total Population
NE	9	49135283
N Cntrl	12	58865670
South	16	74734029
West	13	43172490

Aggregating observations

Exercise ⌨

Recreate the `region_stats` data set, now including the average and the standard deviation of the population.

```
region_stats <-
  census %>%
  group_by(region) %>%
  summarise(`Number of States` = n_distinct(state),
            `Total Population` = sum(pop))
```

Aggregating observations

```
region_stats ←
  census %>%
  group_by(region) %>%
  summarise(`Number of States` = n_distinct(state),
            `Total Population` = sum(pop),
            `Average Population` = mean(pop),
            `SD of Population` = sd(pop))
```

region	Number of States	Total Population	Average Population	SD of Population
NE	9	49135283	5459476	5925235
N Cntrl	12	58865670	4905472	3750094
South	16	74734029	4670877	3277853
West	13	43172490	3320961	6217177

Aggregating observations

Exercise ⌨

Use `huxtable` to format and export the object `region_stats`.

Region	Number of States	Total Population	Average Population	SD of Population
NE	9	49135283	5459476	5925235
N Cntrl	12	58865670	4905472	3750094
South	16	74734029	4670877	3277853
West	13	43172490	3320961	6217177

Aggregating observations

```r
region_stats_table ←
  region_stats %>%
  rename(Region = region) %>%
  as_hux %>%
  set_header_cols("Region", TRUE)  %>%
  theme_bright()

quick_xlsx(region_stats_table,
         file = file.path(rawOutput, "region-stats.xlsx"))

quick_latex(region_stats_table,
           file = file.path(rawOutput, "region-stats.tex"))
```

Ok, can we run some regressions now?!

Running regressions

The base R command for linear regressions is called `lm`

lm(formula, data, subset, weights, ...)

- **formula:** an object of class "formula" containing a symbolic description of the model
- **data:** a data frame containing the variables indicated in the formula
- *subset:* an optional vector specifying a subset of observations to be used in the regression
- *weights:* an optional vector of weights to be used in the regression

Formulas can take three specifications:

- `y ~ x1 + x2` regresses variable `y` on covariates `x1` and `x2`
- `y ~ x1:x2` regresses variable `y` on the interaction of covariates `x1` and `x2`
- `y ~ x1*x2` is equivalent to `y ~ x1 + x2 + x1:x2`

Running regressions

Exercise ⌨

Using the `census` data, run a regression of the number of divorces on population, urban population and number of marriages.

```
lm(y ~ x1 + x2,
   data)
```

Running regressions

Exercise ⌨

Using the `census` data, run a regression of the number of divorces on population, urban population and number of marriages.

```
lm(divorce ~ pop + popurban + marriage,
    census)
```

```
##
## Call:
## lm(formula = divorce ~ pop + popurban + marriage, data = census)
##
## Coefficients:
## (Intercept)          pop     popurban     marriage
##   1.207e+02    1.044e-03    1.954e-03    2.587e-01
```

- The output of regression commands is a list of relevant information.
- By default, it prints only a small portion of this information.
- The best way to visualize results is to store this list in an object and then access its contents using the function `summary`

Running regressions

```
reg1 ←
  lm(divorce ~ pop + popurban + marriage,
     census)

summary(reg1)
```

```
##
## Call:
## lm(formula = divorce ~ pop + popurban + marriage, data = census)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -22892.3  -1665.1    796.5   4138.0  17212.2
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.207e+02  1.838e+03   0.066    0.948
## pop         1.044e-03  1.633e-03   0.639    0.526
## popurban    1.954e-03  1.796e-03   1.088    0.282
## marriage    2.587e-01  5.958e-02   4.342  7.7e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7466 on 46 degrees of freedom
## Multiple R-squared:  0.9169,    Adjusted R-squared:  0.9115
## F-statistic: 169.2 on 3 and 46 DF,  p-value: < 2.2e-16
```

Running regressions

The `lfe` command `felm` allows for more flexibility in model specification

felm(formula, data, subset, weights, ...)

- **formula:** an object of class "formula" containing a symbolic description of the model
- **data:** a data frame containing the variables indicated in the formula
- *subset:* an optional vector specifying a subset of observations to be used in the regression
- *weights:* an optional vector of weights to be used in the regression

Formulas for `felm` are more complex, and take the following format: `y ~ x1 + x2 | fe1 + fe2 | (Q|W ~ iv3+iv4) | clu1 + clu2`

- `y ~ x1 + x2` takes all the same formulas as `lm`
- `fe1 + fe2` list the variables to be included as fixed effects
- `(Q|W ~ iv3 + iv4)` uses instruments `iv3` and `iv4` for variables `Q` and `W`
- `clu1 + clu2` indicates that standard errors should be clustered using variables `clu1` and `clu2`

Running regressions

Exercise ⌨

Using the `census` data, run a regression of the number of divorces on population, urban population and number of marriages controlling for region fixed effects.

```
felm(y ~ x1 + x2 | fe1 + fe2 | 0 | 0,
  data)
```

Running regressions

Exercise ⌨

Using the `census` data, run a regression of population, urban population and number of marriages controlling for region fixed effects.

```r
felm(divorce ~ pop + popurban + marriage | region | 0 | 0,
    census)
```

```
##       pop  popurban  marriage
## 0.0003951 0.0035532 0.1836593
```

Running regressions

```
reg2 ←
  felm(divorce ~ pop + popurban + marriage | region | 0 | 0,
     census)

summary(reg2)
```

```
##
## Call:
##    felm(formula = divorce ~ pop + popurban + marriage | region |      0 | 0, data = census)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -17919  -3112   -448   3047  13830
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## pop      0.0003951  0.0017881   0.221  0.82615
## popurban 0.0035532  0.0019981   1.778  0.08243 .
## marriage 0.1836593  0.0580271   3.165  0.00285 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6748 on 43 degrees of freedom
## Multiple R-squared(full model): 0.9365   Adjusted R-squared: 0.9277
## Multiple R-squared(proj model): 0.9354   Adjusted R-squared: 0.9264
## F-statistic(full model):105.8 on 6 and 43 DF, p-value: < 2.2e-16
## F-statistic(proj model): 207.7 on 3 and 43 DF, p-value: < 2.2e-16
```

Some notes on regressions

- Whenever a factor is included in the list of covariates, it is treated as a categorical variable, i.e., as if you had written `i.x` in Stata.
- Whenever a boolean is included in the list of covariates, it is treated as a dummy variable, where `TRUE` is `1` and `FALSE` is `0`.
- `felm` also allows for bootstrapping, but this is beyong the scope of this session.

Exporting regression tables

Exporting regression tables

`huxtable` also has a quick wrapper for regression tables

huxreg(...)

- **...**: Models, or a single list of models. Names will be used as column headings.
- *number_format:* Format for numbering. See number_format() for details.
- *stars:* Levels for p value stars.
- *bold_signif:* Where p values are below this number, cells will be displayed in bold.
- *note:* Footnote for bottom cell, which spans all columns.
- *statistics:* A vector of summary statistics to display.
- *coefs:* A vector of coefficients to display. To change display names, name the coef vector: c("Displayed title" = "coefficient_name", ...)

Exporting regression tables

```
huxreg(reg1, reg2)
```

	(1)	(2)
(Intercept)	120.730	
	(1838.216)	
pop	0.001	0.000
	(0.002)	(0.002)
popurban	0.002	0.004
	(0.002)	(0.002)
marriage	0.259 ***	0.184 **
	(0.060)	(0.058)
N	50	50
R2	0.917	0.937
logLik	-514.766	
AIC	1039.531	
*** p < 0.001; ** p < 0.01; * p < 0.05.		

Formatting regression tables

```r
huxreg(reg1, reg2,
       coefs = c("Population" = "pop", # Show variable labels instead of names
                 "Urban population"  = "popurban",
                 "Number of marriages" = "marriage"),
       statistics = c("N. obs." = "nobs")) %>%
       add_rows(c("Region FE", "No", "Yes"),
                after = 7)
```

	(1)	(2)
Population	0.001	0.000
	(0.002)	(0.002)
Urban population	0.002	0.004
	(0.002)	(0.002)
Number of marriages	0.259 ***	0.184 **
	(0.060)	(0.058)
Region FE	No	Yes
N. obs.	50	50
*** p < 0.001; ** p < 0.01; * p < 0.05.		

References and recommendations

- Skimr documentation: https://qiushi.rbind.io/post/introduction-to-skimr/
- Introduction to `huxtable`: https://cran.r-project.org/web/packages/huxtable/vignettes/huxtable.html
- Using `huxtable` for regression tables: https://cran.r-project.org/web/packages/huxtable/vignettes/huxreg.html
- Johns Hopkins Exploratory Data Analysis at Coursera: https://www.coursera.org/learn/exploratory-data-analysis
- Udacity's Data Analysis with R: https://www.udacity.com/course/data-analysis-with-r--ud651
- Applied econometrics with R https://www.springer.com/us/book/9780387773162

Since we talked about LaTeX so much...

- DIME LaTeX templates and trainings: https://github.com/worldbank/DIME-LaTeX-Templates
- All you need to know about LaTeX: https://en.wikibooks.org/wiki/LaTeX

Thank you!