POLI 30 D: Political Inquiry

TA Sessions

Lab 07 | R Plots and R Data Analysis III

Before we start

Announcements:

- GitHub page: **https://github.com/umbertomig/POLI30Dpublic**
- Piazza forum: The link in the slides needs to be fixed. Check with instructors for an alternative link.

Before we start

Recap: In the Lab sessions, you learned:

- How to install R and R Studio on your computer.
- How to do basic and advanced operations with vectors and data frames.
- How to install packages and work with R Markdown.
- How to create plots and data viz.
- How to do data analysis.

Great job!

- Do you have any questions about these contents?

Plan for Lab 07

- Selecting variables
- Filtering cases in the dataset
- Grouping and summarizing data
- Visualizing correlations

Getting started

Getting started

- To get started, we need to load the datasets we will need in the lab.
- We also need to load the `tidyverse` package, which has all the R functions we use.

```
library(tidyverse)
```

Getting started - Education expenditure data

```r
educexp <- read.csv("https://raw.githubusercontent.com/umbertom
head(educexp)
```

```
##   education income young urban states
## 1       189   2824 350.7   508     ME
## 2       169   3259 345.9   564     NH
## 3       230   3072 348.5   322     VT
## 4       168   3835 335.3   846     MA
## 5       180   3549 327.1   871     RI
## 6       193   4256 341.0   774     CT
```

Getting started - Chile survey data

```r
chilesurv <- read.csv("https://raw.githubusercontent.com/umbert
head(chilesurv)
```

```
##   statusquo vote voteYES
## 1   3.02460    Y       1
## 2  -3.88851    N       0
## 3   3.69216    Y       1
## 4  -3.09489    N       0
## 5  -3.31488    N       0
## 6  -3.14055    N       0
```

Getting started - Voting

```r
voting <- read.csv("https://raw.githubusercontent.com/umbertomi
head(voting)
```

```
##   birth message voted
## 1  1981      no     0
## 2  1959      no     1
## 3  1956      no     1
## 4  1939     yes     1
## 5  1968      no     0
## 6  1967      no     0
```

The `tidyverse` package

The **`tidyverse`** package

- The `tidyverse` package is an eco-system for data analytics.
- It has packages for processing, plot, and wrangle data.
- We have seen the package for plotting: `ggplot2`.
- But there are other important packages in the `tidyverse` constellation.
- And in the links we provide URLs for cheat sheets.

The `tidyverse` package

Other imporatant packages:

- `forcats`: Package to deal with categorical variables.
- `stringr`: Package to deal with strings and texts in R.
- `purrr`: Package to work with functions and lists.
- `tidyr`: Package to clean up and reshape datasets.
- `dplyr`: Package to manipulate (wrangling) datasets.

We are going to do some `dplyr` today!

`dplyr`

- `dplyr` is the package on `tidyverse` for data wrangling.
- Data wrangling is a fancy way to say that you manipulate your dataset until it is in shape for analysis.
- Most of the time, we collect data from the internet, documents, or even old archives.
- We then need to do some *wrangling* to ensure it is ready to extract the statistics of interest.

`dplyr`

`dplyr` is based on a bunch of verbs that do what we intuitively think each of them should do:

- Variable/column:
  - `select`: Select variables
  - `rename`: Rename variables
  - `mutate`: Mutate variables
- Observation/row:
  - `filter`: Filter dataset
  - `arrange`: Arrange observations
- Get summaries:
  - `summarize`: Summarize the dataset
  - `group_by`: Group summary by a variable

Selecting Variables (column operations)

Selecting Variables (column)

Suppose we want to select only the numeric variables in the `educexp` dataset. We do the following:

```
educ2 <- select(educexp, education, income, young, urban)
head(educ2, 2)
##   education income young urban
## 1       189   2824 350.7   508
## 2       169   3259 345.9   564
```

- The syntax is:

```
newdat <- select(olddat, v1, ..., vn)
```

Selecting Variables (column)

Or a more compact notation for this uses *pipes*:

```
educ3 <- educexp %>%
  select(education, income, young, urban)
head(educ3, 2)
##   education income young urban
## 1       189   2824 350.7   508
## 2       169   3259 345.9   564
```

- Note that the *pipe* operator (%>%) works like a composite function: takes what is in the left and passes through to the right.
- The first thing that the select is looking for is the dataset. The *pipe* passes the dataset, so you do not need to mention it.

Selecting Variables (column)

And the methods we can apply in the `select` are the following:

Method	Effect
v1, v2, v3 (etc)	Select given variables
starts_with('xyz')	Select starting with `xyz`
ends_with('xyz')	Select ending with `xyz`
contains('xyz')	Select variables that have `xyz` in their names
vk:vn	All variables between `vk` and `vn`
-(vk:vn)	All but the variables between `vk` and `vn`

We can even rename variables using `select`.

Selecting Variables (column)

Example 1 (same selection as before):

```
educ4 <- educexp %>% select(-states)
head(educ4, 3)
##   education income young urban
## 1       189   2824 350.7   508
## 2       169   3259 345.9   564
## 3       230   3072 348.5   322
```

Example 2 (in the Chile dataset now):

```
chile2 <- chilesurv %>% select(starts_with('vote'))
head(chile2, 3)
##   vote voteYES
## 1    Y       1
## 2    N       0
## 3    Y       1
```

Rename

We can change the names of the variables using `rename`. The basic syntax is:

```
newdat <- olddat %>%
  rename(newname1 = oldname1,
         newname2 = oldname2,
         etc)
```

Example:

```
chile3 <- chilesurv %>% rename(votebinary = voteYES)
head(chile3, 2)
##   statusquo vote votebinary
## 1   3.02460    Y          1
## 2  -3.88851    N          0
```

Pipeing like a boss

You can *pipe* as many commands as you want. Example:

```
chile4 <- chilesurv %>%
  select(starts_with('vote')) %>%
  rename(votebinary = voteYES)
head(chile4, 2)
##   vote votebinary
## 1    Y          1
## 2    N          0
```

- Note the *chain* pipe!

Filtering Rows

Filtering Rows

- Filtering the data means subset the dataset based on a logical condition.
  - Example: In the voting pressure experiment, we may want to study the effect of messaging young people.
  - Reason: Younger people may not be as sensitive to peer pressure.

```
voting2 <- voting %>% filter(birth >= 1975)
head(voting2, 4)
##   birth message voted
## 1  1981      no     0
## 2  1983      no     0
## 3  1979      no     0
## 4  1985     yes     1
```

Filtering Rows

- Syntax:

```
newdat <- filter(olddat, condition1, condition2, etc)
or
newdat <- olddat %>% filter(condition1, condition2,
etc)
```

- Note that you may or may not use the pipe operator. But we can all agree that it makes the code much cleaner.

Filtering rows

- The filter operators are the following:

Operator	Meaning
< or <=	Smaller than or smaller than or equal
> or >=	Greater than or greater than or equal
==	Equal
!=	Different
!	Negation (turns a TRUE into a FALSE)
|	Or
&	And

Arrange

- We can sort the dataset by variable content according to the needs of our analysis.

```
newdat <- olddat %>% arrange(var1, desc(var2), etc)
```

- The `desc()` command sorts descending (highest to the lowest). The default is to sort ascending.

Arrange

- Example: Sorting by education expenditure (high to low):

```
educexp %>% arrange(desc(education)) %>% head()
##   education income young urban states
## 1       372   4146 439.7   484     AK
## 2       273   3968 348.4   909     CA
## 3       262   3341 365.4   664     MN
## 4       261   4151 326.2   856     NY
## 5       248   3795 375.9   722     DE
## 6       247   3742 364.1   766     MD
```

Mutating (altering the content)

Mutating

- We frequently need to operate with our variables. For example, suppose that you want to apply a log function to `income`.
- `log` has the advantage of transforming large changes into smaller ones.
- It also has the advantage of interpreting regression models. Check out this UofV Library entry.
- Syntax:

```
newdat <- olddat %>% mutate(vnew = calcs(vold), etc)
```

Mutating

Example: Log of `education` and the sum of variables:

```
educ5 <- educexp %>%
  mutate(logeduc = log(education),
         sum_iyu = income + young + urban)
head(educ5, 4)
```

```
##   education income young urban states  logeduc sum_iyu
## 1       189   2824 350.7   508     ME 5.241747  3682.7
## 2       169   3259 345.9   564     NH 5.129899  4168.9
## 3       230   3072 348.5   322     VT 5.438079  3742.5
## 4       168   3835 335.3   846     MA 5.123964  5016.3
```

Your turn: Adapt this code to arrange the dataset in terms of `logeduc`.

Grouping(-by) and Summarizing

Summarize

- The `summarize` function is helpful when computing summary statistics in the whole sample.
- Syntax:

```
olddat %>% summarize(stat1 = calcs(vars1), stat2 = calcs(vars
```

- Note that we only save the results if we need them! The idea is that those summary statistics you compute to check, not save.

Summarize

- Example: Suppose we want to compute the mean, standard deviation, and the number of observations for `education` and `income`.

```
educexp %>%
  summarize(avgeduc = mean(education),
            stdeveduc = sd(education), neduc = n(),
            avginc = mean(income),
            stdevinc = sd(income), ninc = n())
##    avgeduc stdeveduc neduc   avginc stdevinc ninc
## 1 196.3137  46.45449    51 3225.294  560.026   51
```

- Besides the number of observations, these are pretty informative statistics.

Group-by and Summarize

- But we may want the summary by a group. For example, to compute the *differences-in-means* estimator, we need:
  - The average in the treatment group
  - The average in the control group.
- Group-by can help us here:
  - First, we group our results by the treatment status.
  - Then, we summarize.
- It will create one summary for each group.
- Syntax:

```
dat %>% group_by(groupvar) %>%
  summarize(stat1 = calcs(vars1), etc)
```

Group-by and Summarize

- Example:

```
voting %>% group_by(message) %>%
  summarize(avgvote = mean(voted) * 100, ncases = n())
## # A tibble: 2 x 3
##   message avgvote ncases
##   <chr>     <dbl>  <int>
## 1 no         29.7 191243
## 2 yes        37.8  38201
```

- And the differences-in-means estimator is then straightforward to compute.
- Note that we also saved the number of cases in each treatment status.
    - It makes more sense: Number of control and treatment groups cases.

Group-by and Summarize

Your turn: Adapt this example here to compute the percentage of voters in the treatment and the control group, filtering by younger voters (born on or after 1975).

- Example:

```
voting %>%
  group_by(message) %>%
  summarize(avgvote = mean(voted) * 100,
            ncases = n())
```

Question: Does the treatment effect increase or decrease? Why do you think that is the case?

Detour: `tibble` versus `data.frame`

- When using `dplyr` functions such as the ones we have used in here, `tidyverse` sometimes saves the results in `tibble`s.

- `tibble`s are pretty much the same as `data.frame`s.

- To go back to `data.frame`, you should do the following: `dfdat <- data.frame(tibbledat)`.

- Or you can add a `%>% data.frame()` to the end of your pipeing. For this class, it does not matter which one you use.

- If you want to check about `tibble`s, here is a good source.

Plotting Correlations

`ggcorrplot` and `GGally`

- These packages are helpful to add some more spice to our plots.
- For example, `ggcorrplot` is good for plotting correlations of multiple variables.
  - The command `cor` computes two-by-two correlations for all the combinations of variables when more the two vars.
- Installing packages: Run this code in your console:

```
install.packages('ggcorrplot')
install.packages('GGally')
```

- Warning: You should ***never*** install packages on R Markdown!

ggcorrplot and GGally

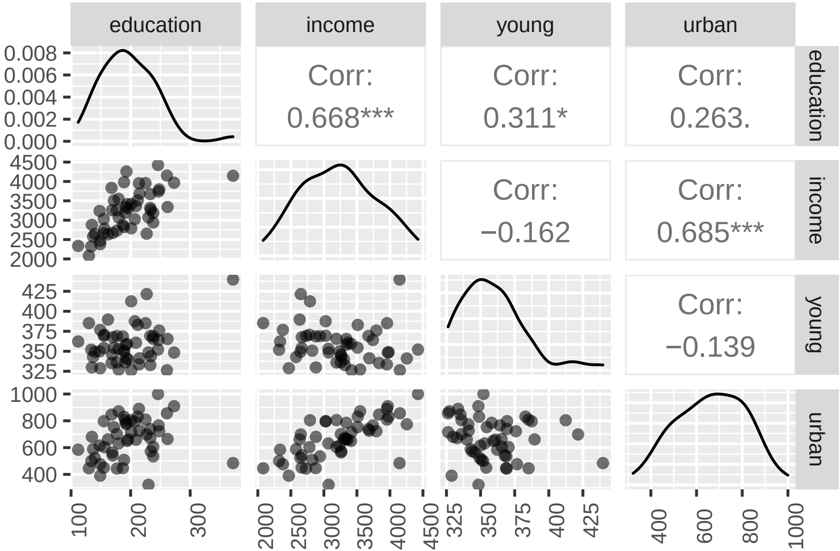
- Now, to load both packages:

```r
library('ggcorrplot')
library('GGally')
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

- And we now have all the functionalities of these packages at our disposal!

```r
ggpairs(educexp %>% select(-states), aes(alpha = 0.4)) +
  theme(text = element_text(size = 10),
        axis.text.x = element_text(angle = 90) )
```



Correlations

- And the correlation between these variables is:

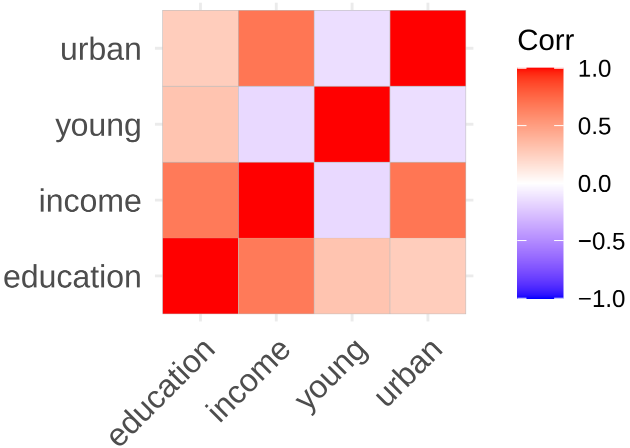
```
educexp %>% select(-states) %>% cor()
##           education     income      young      urban
## education 1.0000000  0.6675773  0.3114855  0.2633238
## income    0.6675773  1.0000000 -0.1623600  0.6854580
## young     0.3114855 -0.1623600  1.0000000 -0.1386334
## urban     0.2633238  0.6854580 -0.1386334  1.0000000
```

- The correlation between the variable and itself is always one.
- The correlation is a **`symmetric matrix`**:
  - The correlation between `income` and `education` is the same as the correlation between `education` and `income`.

Correlations

Using squares:

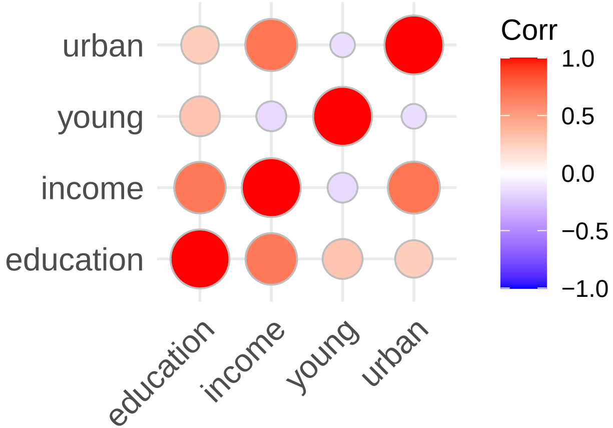
```
ggcorrplot(educexp %>% select(-states) %>% cor())
```



Correlations

And using circles:

```
ggcorrplot(educexp %>% select(-states) %>% cor(),
           method = 'circle')
```



`ggcorrplot` and `GGally`

- Note that both commands use **only numeric variables.**
- Another thing to note is that you can do correlation plots using `GGally`.
  - Function name: `ggcorr`. The syntax is different, but it also produces a nice plot.
- Syntax for `ggpairs`:

```
dat %>% select(numericvars) %>% ggpairs()
```

- Syntax for `ggcorrplot`:

```
dat %>% select(numericvars) %>%
  cor() %>% ggcorrplot()
```

Today's Lab

- Selecting variables
- Filtering cases in the dataset
- Grouping and summarizing data
- Visualizing correlations

Next Lab

- More data wrangling
- One more plot
- Dealing with missing data
- Lots of analysis

Questions?

See you in the next lab!