Quantitative Content Analysis: Lecture 4

Olga Kononykhina

01 March 2017

Today's Outline

Intro to R (Part III)

- Creating and applying functions
- Tidy data
- Visualizations
- First assignment

For() loops

For() loops are used to loop around a vector/matrix to do something.

```
m <- matrix(1:5, nrow=1, ncol=5)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
```

```
for (j in 1:3){
    m[,j]=0
}
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    4    5
```

For() loops (II)

You can also 'nest' a `for()` loop in another `for()` loop

```
m <- matrix(1:15, nrow=3, ncol=5)
for (i in 1:2){
  for (j in 1:4){
    m[i,j]=0
  }
}
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0   13
## [2,]    0    0    0    0   14
## [3,]    3    6    9   12   15
```

Task 4

Create a matrix with `matrix(1:20,nrow=4,ncol=5)` and another with `matrix(NA,nrow=4,ncol=5)`. Adapt the second to the first matrix using `for()`

- hint: define a 'counter' variable that increments by 1 each time you move to the next cell

Task 4 (solution)

Create a matrix with `matrix(1:20,nrow=4,ncol=5)` and another with `matrix(NA,nrow=4,ncol=5)`. Adapt the second to the first matrix using `for()`

```
counter=1
m1 <- matrix(1:20,nrow=4,ncol=5)
m2 <- matrix(NA,nrow=4,ncol=5)
for (j in 1:5){
  for (i in 1:4){
    m2[i,j]=counter
    counter=counter+1
  }
}
```

Task 4 (alternative solution)

Create a matrix with `matrix(1:20,nrow=4,ncol=5)` and another with `matrix(NA,nrow=4,ncol=5)`. Adapt the second to the first matrix using `for()`

```
m1 <- matrix(1:20,nrow=4,ncol=5)
m2 <- matrix(NA,nrow=4,ncol=5)
for (j in 1:5){
  for (i in 1:4){
    m2[i,j]=m1[i,j]
  }
}
```

If() statements

If() statements are used to make conditions on executing some code. If condition is true, action is done.

```
a <- 3
b <- 4
number <- 0
if(a<b){
  number=number+1
}
number
```

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

Tests for conditions: `==; >; <; >=; <=; !=`

Task 5

Create the two objects `a <- sample(c(4,0),20,replace=TRUE)` and `m <- matrix(a,nrow=4,ncol=5)`. Now recode all the 4s into 1s using if() and for() statements.

Task 5 (solution)

Create the objects `a <- sample(c(4,0),20,replace=TRUE)` and `b <- matrix(a,nrow=4,ncol=5)`. Now recode all the 4s into 1s in `b` using if() and for() statements.

```
a <- sample(c(4,0),20,replace=TRUE)
b <- matrix(a,nrow=4,ncol=5)

for (j in 1:5){
  for (i in 1:4){
    if (b[i,j]==4){
      b[i,j]=1
    }
  }
}
```

Creating Functions

If you want to repeat an operation it is often useful to create your own Function. Functions have names, inputs and outputs and simplify your code.

```
# Find the sample mean of a vector
fun_mean <- function(x){
    sum(x) / length(x)
}

## Find the mean
data(swiss)
fun_mean(x = swiss$Infant.Mortality)
```

```
## [1] 19.94255
```

Task 6

Write a function that takes a number and doubles it.

Task 6 (solution)

Write a function that takes a number and doubles it.

```
double <- function(x){
output <- x * 2
output
}
double(8)
```

```
## [1] 16
```

Apply function

`Apply` allows you to apply a function to every row or every column. This can be done with a `for()` loop, but `apply()` is usually much faster and simpler. `Apply()` takes the following form: `apply(X, MARGIN, FUN, ...)`.

```
m <- matrix(c(1:10, 11:20), nrow = 10, ncol = 2)
# mean of the rows
apply(m, 1, mean)
```

```
##  [1]  6  7  8  9 10 11 12 13 14 15
```

```
# mean of the columns
apply(m, 2, mean)
```

```
## [1]  5.5 15.5
```

Task 7

Load up the build-in R dataset 'iris' and use `apply()` to get the mean of the first 4 variables.

Task 7 (solution)

Load up the build-in R dataset 'iris' and use `apply()` to get the mean of the first 4 variables.

```
attach(iris)
apply(iris[,1:4], 2, mean)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

Packages

You can greatly expand the number of functions by installing and loading user-created packages.

```
# Install dplyr package
install.packages('dplyr')

# Load dplyr package
library(dplyr)
```

You can also call a function directly from a specific package with the double colon operator (`::`).

```
Grouped <- dplyr::group_by(combined_df, character_vector)
```

Tidy data

Most of the time data sets have to be cleaned before you can run statistical analyses on them. To help streamline this process Hadley Wickham laid out principles of data tidying which links the physical structure of a data set to its meaning (semantics).

- In tidy data:
  - Each variable is placed in its own column
  - Each observation is placed in its own row
  - Each type of observational unit forms a table

Tidy data

Not tidy:

Person	treatmentA	treatmentB
John Smith		2
Jane Doe	16	11

Tidy:

Person	treatment	result
John Smith	a	
Jane Doe	a	16
John Smith	b	2
Jane Doe	b	11

Messy to tidy data

```r
# Create messy data
messy <- data.frame(
  person = c("John Smith", "Jane Doe"),
  a = c(NA, 16), b = c(2, 11))
# Gather the data into tidy format
library(tidyr)
tidy <- gather(messy, treatment, result, a:b)
tidy
```

```
##       person treatment result
## 1 John Smith         a     NA
## 2   Jane Doe         a     16
## 3 John Smith         b      2
## 4   Jane Doe         b     11
```

Merging data

Once you have tidy data frames, you can merge them for analysis. Each observation must have a unique identifier to merge them on.

```r
data(swiss)
swiss$ID <- rownames(swiss) # Create ID
df <- merge(swiss[,c(1:3,7)], swiss[,4:7], by = "ID")
```

Appending data

You can also add observations to a data frame.

```
data(swiss)
df <- swiss[1:3,1:3]
df2 <- rbind(df, swiss[4,1:3])
df2
```

```
##              Fertility Agriculture Examination
## Courtelary        80.2        17.0          15
## Delemont          83.1        45.1           6
## Franches-Mnt      92.5        39.7           5
## Moutier           85.8        36.5          12
```

Using dplyr

The dplyr package has powerful capabilities to manipulate data frames quickly.

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
data(swiss)
```

Piping

Piping allows to pass a value forward to a function call and produces faster compilation and enhanced code readability. In R use %>% from the dplyr package.

```
# Not piped:
values <- rnorm(1000, mean = 10)
value_mean <- mean(values)
round(value_mean, digits = 2)
```

```
## [1] 9.94
```

```
# Piped:
library(dplyr)
rnorm(1000, mean = 10) %>% mean() %>% round(digits = 2)
```

```
## [1] 9.92
```

Tabulation

There are a number of ways to generate tables in R. Two useful tools that are good to know are:

- `kable`: for creating tables from data frames
- `stargazer`: for creating tables of regression model output

In R, models are fit using the adequate functions e.g. `lm()` for OLS. Many models are packaged in one function, e.g. logistic regression is used with the `glm()` (generalized linear model) function by specifying the model type.

kable

```
library(knitr)
kable(head(mtcars[,1:6]), digits = 2)
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.62
Mazda RX4 Wag	21.0	6	160	110	3.90	2.88
Datsun 710	22.8	4	108	93	3.85	2.32
Hornet 4 Drive	21.4	6	258	110	3.08	3.21
Hornet Sportabout	18.7	8	360	175	3.15	3.44
Valiant	18.1	6	225	105	2.76	3.46

stargazer

`kable` is limited if we want to create regression output tables, especially for multiple models. `stargazer` is good for this. `stargazer` can output tables in various formats.

To export a table to word document use: `type = 'html'`.

stargazer example

```r
library(stargazer)
# Run regressions
output <- lm(rating ~ complaints + privileges + learning
                        + raises + critical, data=attitude)
output2 <- lm(rating ~ complaints + privileges + learning,
              data=attitude)

# Create table
stargazer(output, output2, type="html",
          out="attitude.htm")
```

stargazer example

	Dependent variable:	
	rating	
	(1)	(2)
complaints	0.692***	0.682***
	(0.149)	(0.129)
privileges	-0.104	-0.103
	(0.135)	(0.129)
learning	0.249	0.238*
	(0.160)	(0.139)
raises	-0.033	
	(0.202)	
critical	0.015	
	(0.147)	
Constant	11.011	11.258
	(11.704)	(7.318)
Observations	30	30
R^2	0.715	0.715
Adjusted R^2	0.656	0.682
Residual Std. Error	7.139 (df = 24)	6.863 (df = 26)
F Statistic	12.063*** (df = 5; 24)	21.743*** (df = 3; 26)
Note:		*p<0.1; **p<0.05; ***p<0.01

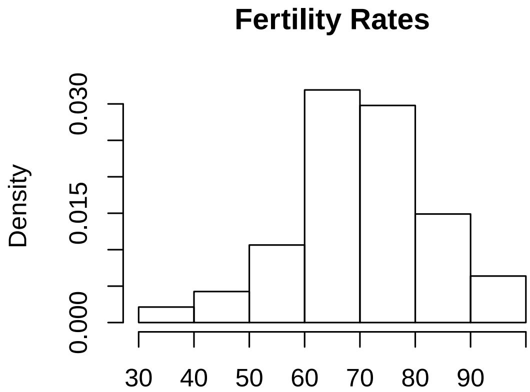
Plotting with R

R has a powerful graphics engine to produce high quality graphs e.g.:

- `plot`: Basic plotting function (e.g. for scatterplots)
- `hist()`: Histograms
- `dotchart()`: Dot plots
- `boxplot()`: Box-and-whisker plots

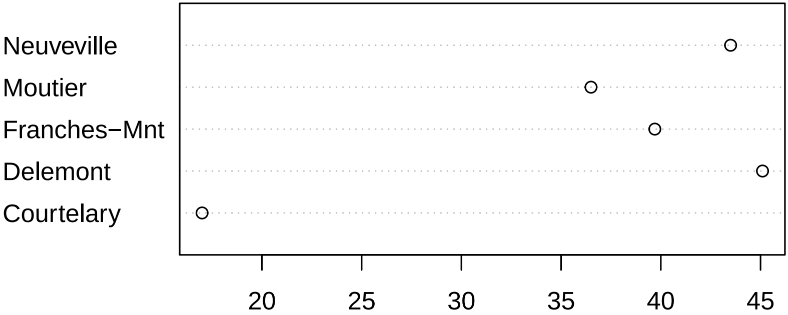
Histograms

```
# Create a histogram
data(swiss)
hist(swiss$Fertility, freq=FALSE, main="Fertility Rates")
```



Dotchart

```r
# Create a dot plot
data(swiss)
dotchart(swiss[1:5,2], labels=rownames(swiss))
```



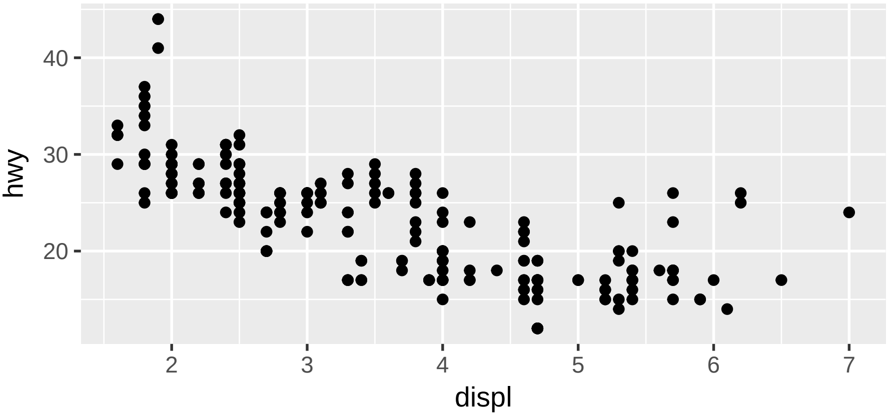
ggplot2

R has several systems for making graphs, but `ggplot2` is one of the most elegant and most versatile. `ggplot2` implements the grammar of graphics, a coherent system for describing and building graphs.

- Each plot is made of layers. Layers include the coordinate system (x-y), points, labels, etc.
- Each layer has aesthetics (`aes`) including x & y, size, shape, and color.
- The main layer types are called geometrics(`geom`) and include lines, points, etc.

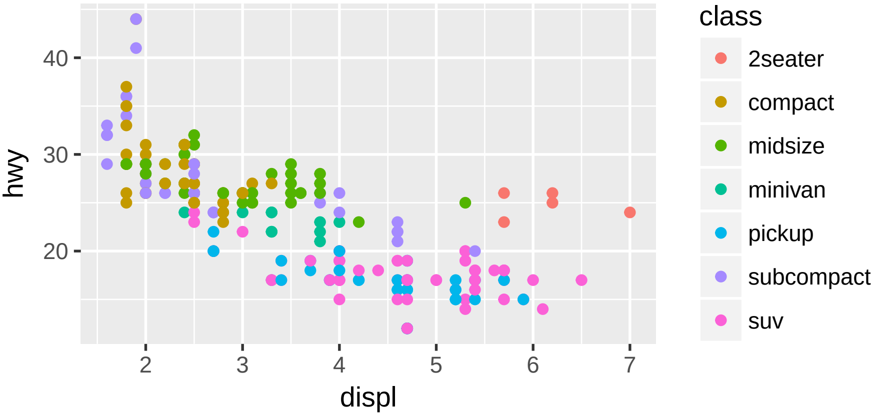
ggplot2 example

```
library(ggplot2)
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



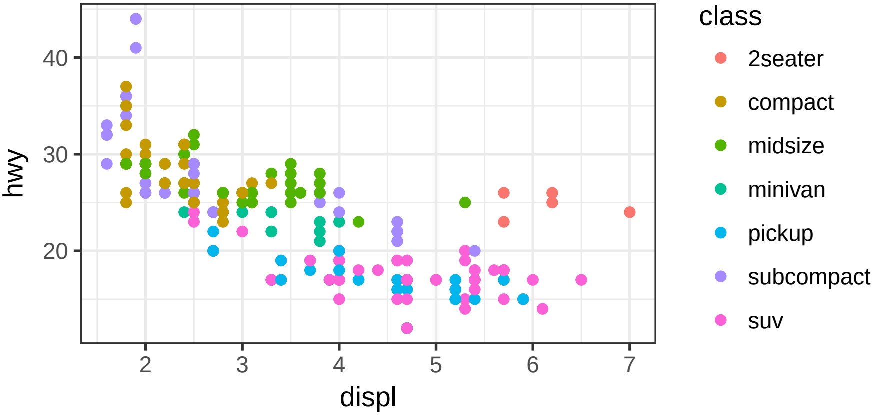
ggplot2 customization: color

```
library(ggplot2)
ggplot(data = mpg) +
 geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



ggplot2 customization: theme

```
library(ggplot2)
ggplot(data = mpg) +  theme_bw() +
 geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



Assignment 1

The Goal is to replicate some of the results from Baumann, M., M. Debus and J. Müller. 2015. “Personal Characteristics of MPs and Legislative Behavior in Moral Policymaking.” *Legislative Studies Quarterly 40(2)*: 179-211.

The article deals with MPs‘ legislative behavior in moral policy making.

- Legislative behavior: taking positions in speeches and voting
- Behavior is influenced by personal characteristics and features of the MPs constituency

Assignment 1: Tasks

You will work with two data sets used in the article:

- MPs‘ demographics (mpData.dta)
- German Longitudinal Election Study (ZA5302_en_v6-0-0.dta)

You have two main tasks:

Create and describe variables

- `Church Attendance in Constituency` is not in mpData.dta
- Use the GLES data to construct your own mean constituency church attendance (vndkirchg) and merge it with the MP level data
- Provide descriptive statistics for your variable

Replicate results

- OLS regression explaining MPs‘ positions (Table 3)
- Logit model explaining roll-call voting behavior (Table 4)

Assignment 1: Deliverables

Short essay:

- Briefly explain what you are doing
- Provide your ouput tables
- Max. 2 pages
- Code: .r file that must reproduce your results
- Due on 15 March (in class)

Next week

- Concepts in quantitative text analysis