Data management in R: Session 2

USC Security and Political Economy Lab

Therese Anders

1 Working with data frames in R

1.1 Importing data

Most data formats we commonly use are not native to R and need to be imported. Luckily, there is a variety of packages available to import just about any non-native format. One of the essential libraries is called `foreign` and includes functions to import .csv, .dta (Stata), .dat, ,.sav (SPSS), etc. The `foreign` package is pre-installed. However, we need to tell R that we want to use functions from this package by calling it into the working environment using the `library()` command.

```
library(foreign)
```

In this example, we will use the data from Imai (2017), that is saved in a .csv file and can be accessed online http://qss.princeton.press/student-files/INTRO/UNpop.csv. Download and save this file on your machine. When reading the data, remember that you need to either specify the complete file name or change your working directory using `setwd()` to the folder in which you saved the data.

```
mydata <- read.csv("UNpop.csv")
class(mydata)
```

```
## [1] "data.frame"
```

1.2 Dimensions of a data frame

Let's find out what this data looks like. First, use the `str()` function to explore the variable names and which data class they are stored in. Note: `int` stands for `integer` and is a special case of the class `numeric`.

```
str(mydata)
```

```
## 'data.frame':    7 obs. of  2 variables:
##  $ year     : int  1950 1960 1970 1980 1990 2000 2010
##  $ world.pop: int  2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

If we are only interested in what the variables are called, we can use the `names()` function.

```
names(mydata)
```

```
## [1] "year"      "world.pop"
```

As we saw in the last session, we can alter the names of vectors by using the `names()` function and indexing. Because data frames are essentially just combinations of vectors, we can do the same for variable names inside data frames. Suppose we didn't like the `.` in a variable name and wanted to change this to an underscore.

```
names(mydata)[2] <- "world_pop"
names(mydata)
```

```
## [1] "year"      "world_pop"
```

We can use the `summary()` function to get a first look at the data using measures of location.

```r
summary(mydata)
```

```
##       year        world_pop      
##  Min.   :1950   Min.   :2525779  
##  1st Qu.:1965   1st Qu.:3358588  
##  Median :1980   Median :4449049  
##  Mean   :1980   Mean   :4579529  
##  3rd Qu.:1995   3rd Qu.:5724258  
##  Max.   :2010   Max.   :6916183
```

A data frame has two dimensions: rows and columns.

```r
nrow(mydata) # Number of rows
```

```
## [1] 7
```

```r
ncol(mydata) # Number of columns
```

```
## [1] 2
```

```r
dim(mydata) # Rows first then columns.
```

```
## [1] 7 2
```

1.3 Accessing elements of a data frame

As a rule, whenever we use two-dimensional indexing in `R`, the order is: `[row,column]`. To access the first row of the data frame, we specify the row we want to see and leave the column slot following the comma empty.

```r
mydata[1,]
```

```
##   year world_pop
## 1 1950   2525779
```

We can use the concatenate function `c()` to access multiple rows (or colums) at once. Below we print out the first and second row of the dataframe.

```r
mydata[c(1,2),]
```

```
##   year world_pop
## 1 1950   2525779
## 2 1960   3026003
```

If we try to access a data point that is out of bounds, `R` returns the value `NULL`. Here, there is no third column!

```r
mydata[3,3]
```

```
## NULL
```

Exercise 1 Access the element of the data frame `mydata` that is stored in row 1, column 1.

```
## [1] 1950
```

Exercise 2 Access the element of the data frame `mydata` that is stored in column 2, row 3.

```
## [1] 3691173
```

1.3.1 The $ operator

The $ operator in R is used to specify a variable within a data frame. This is an alternative to indexing.

```
mydata$year
```

```
## [1] 1950 1960 1970 1980 1990 2000 2010
```

```
mydata$world_pop
```

```
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

Exercise 3: How would you access all elements of the variable `year` (first variable) using indexing rather than the $ operator?

```
## [1] 1950 1960 1970 1980 1990 2000 2010
```

```
## [1] 1950 1960 1970 1980 1990 2000 2010
```

Exercise 4: Print out every second element from the variable `world_pop` using indexing methods and the sequence function `seq()`.

```
## [1] 2525779 3691173 5320817 6916183
```

Exercise 5: What are two ways to find the mean value of the variable `world_pop` using indexing (i.e. not using the $ operator)?

```
## [1] 4579529
```

```
## [1] 4579529
```

Exercise 6: How would you find the maximum world population value using the $ operator?

```
## [1] 6916183
```

Exercise 7: Print the year that corresponds to the maximum world population value using the $ operator and indexing!

```
## [1] 2010
```

1.4 NAs in R

NA is how R denotes missing values. For certain functions, NAs cause problems.

```
vec <- c(4, 1, 2, NA, 3)
mean(vec) #Result is NA!
```

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

```
sum(vec) #Result is NA!
```

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

We can tell R to remove the NA and execute the function on the remainder of the data.

```
mean(vec, na.rm = T)
```

```
## [1] 2.5
```

```
sum(vec, na.rm = T)
```

```
## [1] 10
```

1.5 Adding observations

First, lets add another observation to the data. Suppose we wanted to add an observation for the year 2020, that for right now will be a missing value. We can use the same operations we used for vectors to add data. Here, we will will use the `rbind()` function to do so. `rbind()` stands for "row bind." Save the output in a new data frame!

```
obs <- c(2020, NA)
obs
```

```
## [1] 2020   NA
```

```
mydata_new <- rbind(mydata, obs)
dim(mydata_new)
```

```
## [1] 8 2
```

We can also create new variables that use information from the existing data. The population value is expressed in thousands. To express the world population in its original unit of measurement, we multiply each value by the scalar 1000 and store it in a new value called `world_pop_og`. By using the `$` operator, we can directly assign the new variable to the data frame `mydata_new`.

```
mydata_new$world_pop_og <- mydata_new$world_pop * 1000
head(mydata_new, 3) #prints out the first 3 rows of they data frame
```

```
##   year world_pop world_pop_og
## 1 1950   2525779   2525779000
## 2 1960   3026003   3026003000
## 3 1970   3691173   3691173000
```

We can use indexing and logical expressions to compute the population growth rate relative to the value in 1950. The general formula for computing a growth rate is

$$growth = (new - old)/old * 100.$$

To make the code more legible, I will first store the 1950 value in a separate object called `pop1950`.

```
pop1950 <- mydata_new$world_pop[mydata_new$year == 1950]
mydata_new$world_pop_growth1950 <- (mydata_new$world_pop - pop1950)/pop1950 * 100
```

1.6 Saving data

Suppose we wanted to save this newly created data frame. We have multiple options to do so. If we wanted to save it as a native `.RData` format, we would run the following command.

```
# Make sure you specified the right working directory!
save(mydata_new, file = "mydata_new.RData")
```

Most of the time, however, we would want to save our data in formats that can be read by other programs as well. .csv is an obvious choice. After saving the new file, check the folder that is your working directory to see whether the newly saved file shows up there!

```
write.csv(mydata_new, file = "mydata_new.csv")
```

2 (Very basic) data visualization

Today, we will be covering some basics of data visualization in `R` using the native plotting functions. For more advanced data visualization functions, see the `ggplot2` package and the related material for a two-session introductory workshop on data visualization https://github.com/thereseanders/Workshop-Intro-to-ggplot2.

We will work with data on US states' population, area, and population density values in 2015, derived from https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_States_by_population_density. First, read in the data, and take a look at it using the `head()` function.

```
dat <- read.csv("wiki_us_pop.csv")
head(dat)
```

```
##            state pop_dens     pop    area
## 1     New Jersey    467.2 8958013 19046.8
## 2   Rhode Island    394.4 1056298  2678.0
## 3  Massachusetts    336.3 6794422 20201.9
## 4    Connecticut    286.3 3590886 12540.7
## 5       Maryland    238.9 6006401 25141.0
## 6       Delaware    187.4  945934  5047.9
```

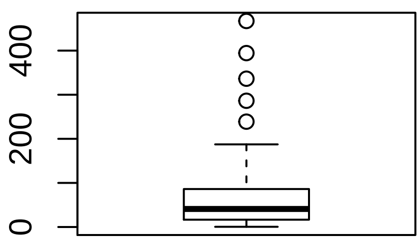
2.1 Basic graphical summaries of data

Type	Operator
Histogram	`hist()`
Stem and leaf plot	`stem()`
Boxplot	`boxplot()`
Kernel density plot	`plot(density())`
Basic scatterplot	`plot()`

2.1.1 Boxplot of population density

First, we get a overview of the population density of US states by using the `boxplot()` function. It seems that there are quite a few outliers when it comes to population density!

```
boxplot(dat$pop_dens)
```



Suppose we wanted to know whether larger states on average are more or less densely populated than smaller states. Let's first look at the distribution of the land area (in square km).

```
summary(dat$area) #mean 182949, median 139578
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2678   95159  139578  182949  210374 1477953
```

Create a new dummy variable that codes whether a state is smaller or larger equal than the **median** state size.

```
median(dat$area)
```

```
## [1] 139578.4
```

```
dat$largestate <- ifelse(dat$area < median(dat$area), 1, 0)
head(dat$largestate)
```

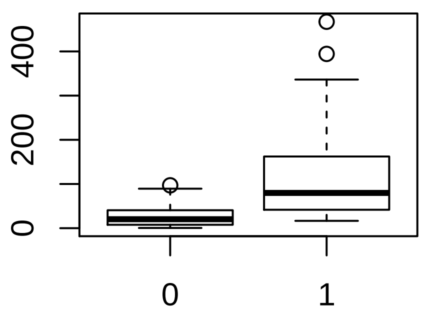
```
## [1] 1 1 1 1 1 1
```

```
table(dat$largestate) # We split the observations exactly in half.
```

```
##
##  0  1
## 25 25
```

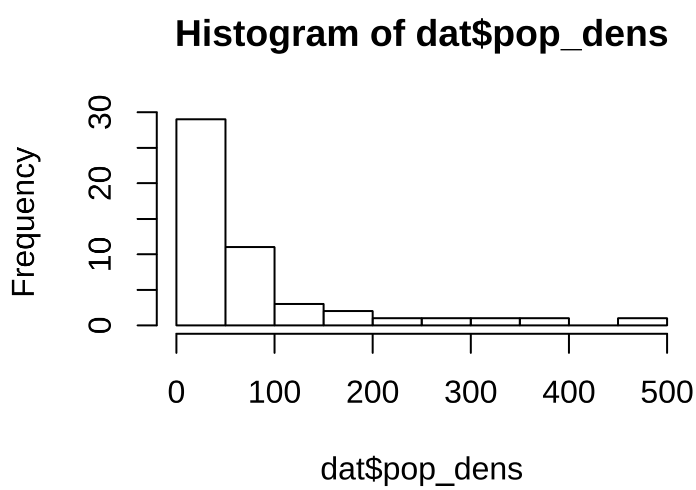
We can display two indicators in the same boxplot. We can use this feature to answer the question whether larger states are on average more or less dense than smaller states.

```
boxplot(dat$pop_dens ~ dat$largestate)
```



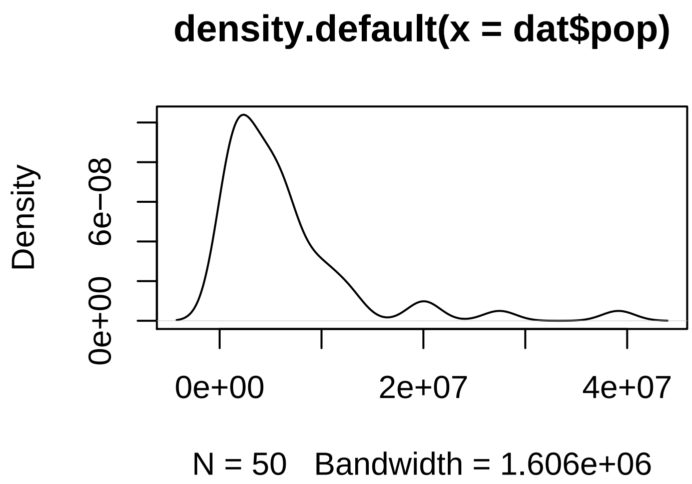
2.1.2 Histogram of population density

```
hist(dat$pop_dens)
```



2.1.3 Density plot of population size

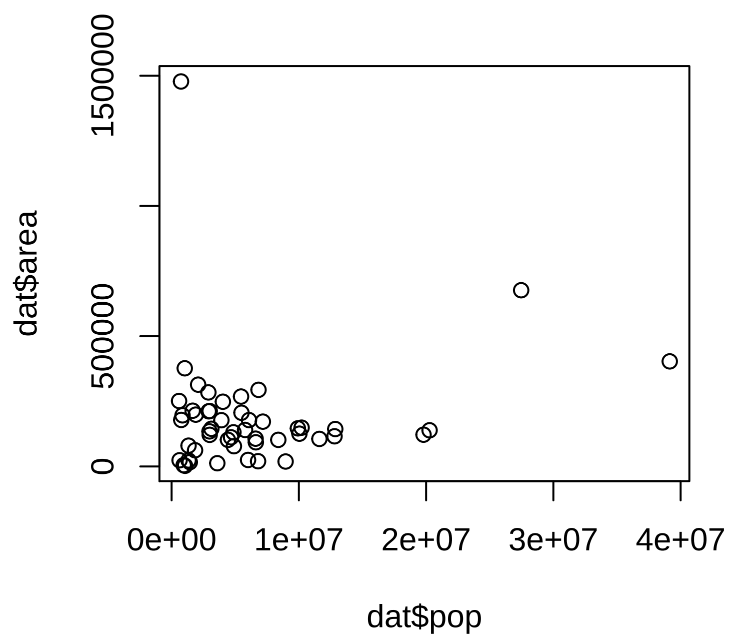
```r
plot(density(dat$pop))
```



2.2 Basic scatter plots

How does population size vary with area? Are the correlated at all?

```r
plot(dat$pop,dat$area)
```

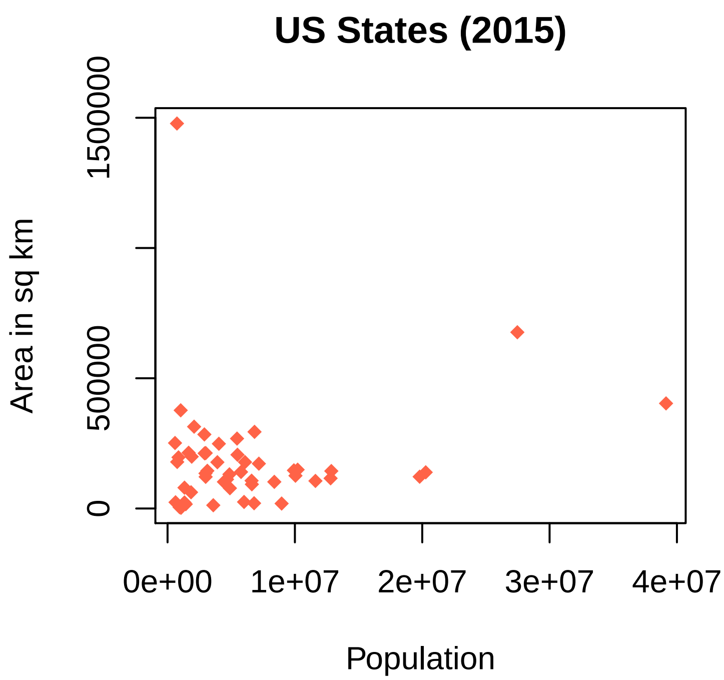


2.2.1 Basic graphic options

Aside from arguably not being very informative, the plot above is not very pretty. Lets give it titles, use color and shapes!

```r
plot(dat$pop, dat$area,
     main = "US States (2015)", #Adding a main title.
```

```
    xlab = "Population", #Adding a x-axis title.
    ylab = "Area in sq km", #Adding a y-axis title.
    col = "tomato", #Changing the color of the data points.
    pch = 18) #Changing the shape
```



Yeah, ok. Its not much prettier (especially the labeling on the axes), but you get the point...

A few additional notes on graphical options:

- `R` can display any color in the RBG or HEX system. However it also has a ton of colors that you can just refer to by name, see http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf.
- Same with the shapes and line types, see http://www.cookbook-r.com/Graphs/Shapes_and_line_types/.
- `R` colors in all their glory: http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf.

References

Imai, Kosuke (2017): *Quantitative Social Science. An Introduction.* Princeton and Oxford: Princeton University Press.