REM Tutorial 2: R Markdown, Descriptive Statistics, Maps – Workbook

Prof. Dr. Kathleen Kürschner Rauck

AT 2023

Contents

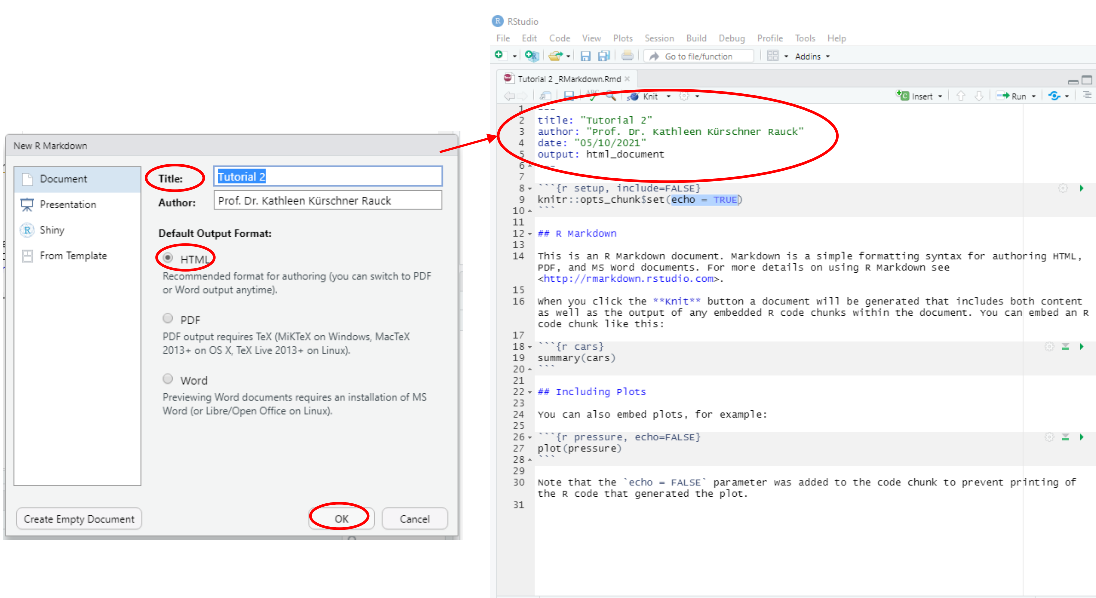
Remember to **organise your workspace** (create a working folder and store the **Lucas_County_data** I uploaded on Canvas), **set up a new RScript** (go to **File > New File > R Script**) and **set your working directory** (got to **Session > Set Working Directory > Choose Directory**) and save the R Script in your working directory by clicking the **save current document** icon (top left pane) or typing **Ctrl+S**.

R Markdown

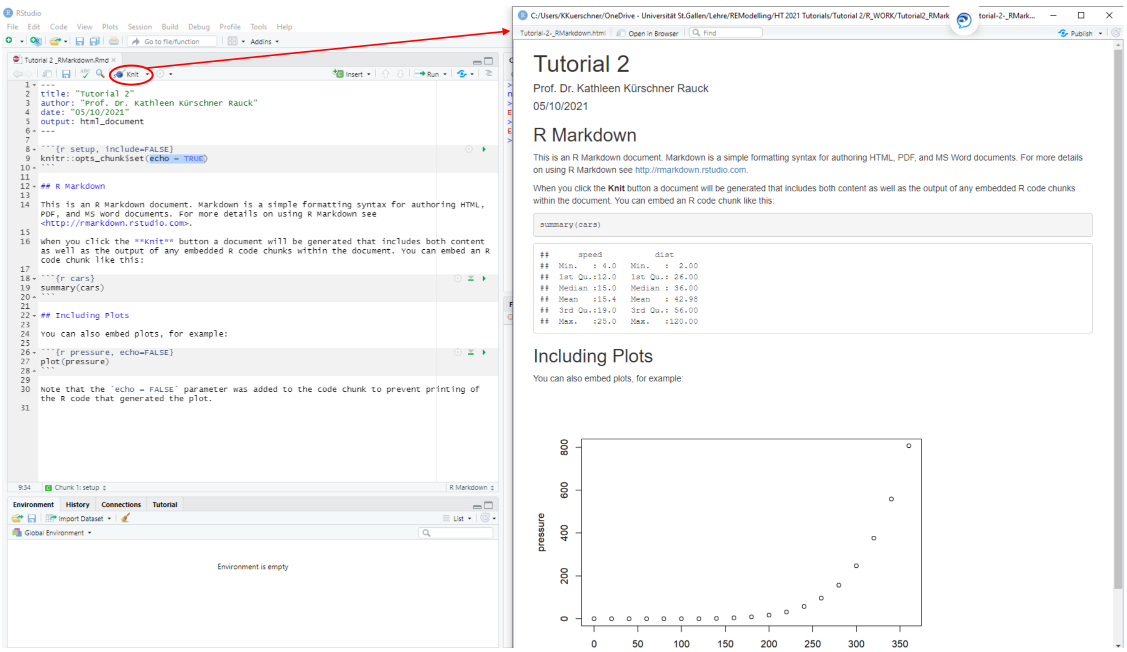
I have mentioned in *Tutorial 1* that you are **required to submit your solutions to the four R assignments in html (hypertext markup language) format**. Such html-files can be generated using **R Markdown** in RStudio. Therefore, we will start with a brief excursion on what R Markdown is and how you can utilise it for your work. R Markdown is a file format that can be used to produce interactive documents to report your R code and the corresponding output. The idea is to document your work in an accessible way, enabling the reader of the R Markdown document to understand and possibly replicate it. You create such a file by writing a report of your work in markdown and render/compile it into a html document, which contains the embedded R code, appendant output and your annotations in a clearly distinguishable way.

To generate an R Markdown file, open up RStudio (click **Start** and type in **RStudio** or double-click the icon on the desktop). You may need to install the `rmarkdown` package (and its dependencies - this is usually done automatically - `knitr`, `yaml`, `htmltools`, `htmlwidgets`, `evaluate`, `markdown` and `highr`). (You may also need to install further packages if it doesn't work outright - check if you can find any helpful prompts in the error messages, e.g.: `glue`, `digest`, `base64enc`, `xfun`, `mime`, `magrittr`, `stringi`, `stringr`.) Reminder: to install packages type `install.packages("")`, entering the name of the package you wish to install in quotation marks.

Then go to **File > New File > R Markdown...**, assign a title to the document, select the HTML output format and click ok. The new RMarkdown file is opened in the *Editor* pane. The file's metadata and output format appear in the document's YAML (yet another markup language) header:



Double-check your working directory is set correctly. Then proceed to **save the R markdown file in your folder, using extension .Rmd**. The R Markdown report you opened contains some sample code chunks and annotations for illustrative purposes. To see what the corresponding output in html-format looks like click the **Knit** icon and wait until a new window opens.



I have uploaded an **rmarkdown_cheat_sheet** on Canvas, which summarises the main settings and formatting options for generating documents in markdown.

Next, we can move on to solve some practical problems provided in *Tutorial 2 - Problem Set*.

Practical 1: Descriptive Statistics

a. Load the Lucas_County_data into RStudio and store it in a data frame, called dat1. What are its dimensions?

We start by loading the **Lucas_County_data**, provided on Canvas, and save it into an R object, called **dat1**. The data comes in txt-file format, so we use again the `read.table()` function.

```r
setwd("C:/Users/USERNAME/REModelling/Tutorial 2/R_WORK")
dat1 <- read.table("Lucas_County_data.txt", header = TRUE)
```

dat1 is a relatively large data set, containing individual property information. Use the function `class()`, introduced in *Tutorial 1*, to determine the class of the R object. By means of the `dim()` function, you can query information on its dimension. Type:

```r
class(dat1) # object class of dat1 is data frame
```

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

```r
dim(dat1) # first number refers to rows, the second to columns
```

```
## [1] 25357    13
```

b. Transform variable *wall* into a factor variable, determine its levels and the distribution of observations across the different levels, storing the information in data frame Tab.wall. Name the column, containing the names of the levels, *Wall Category*.

The variables in this data set have different object classes. For instance, *wall* is of class character:

```
class(dat1$wall)
```

```
## [1] "character"
```

Let us overwrite/convert variable *wall* into a factor variable (i.e., a categorical variable with different levels):

```
dat1$wall <- as.factor(dat1$wall)
class(dat1$wall) # to check
```

```
## [1] "factor"
```

To see which factors *wall* contains type:

```
levels(dat1$wall) # wall has seven levels
```

```
## [1] "Aluminum, vinyl, or steel siding"
## [2] "Concrete block or tile"
## [3] "Full brick exterior"
## [4] "half or less brick exterior"
## [5] "Horizontal or vertical wood siding or shakes"
## [6] "Stone exterior"
## [7] "Stucco or Dryvit plaster"
```

Use the `table()` function to see how the observations are distributed across the different categories and save these information in an R object, called **Tab.wall**:

```
table(dat1$wall) # most houses are made of wood
```

```
##
##             Aluminum, vinyl, or steel siding
##                                         4235
##                       Concrete block or tile
##                                          129
##                          Full brick exterior
##                                         3633
##                  half or less brick exterior
##                                         5896
## Horizontal or vertical wood siding or shakes
##                                        11174
##                               Stone exterior
##                                           86
##                     Stucco or Dryvit plaster
##                                          204
```

```
Tab.wall <- table(dat1$wall) # store the information in R object
class(Tab.wall) # stored in a table
```

```
## [1] "table"
```

Instead of a vector/table, we would like to save the information as a data frame for better legibility. To convert and store the information in a data frame, called **Tab.wall2** type:

```
Tab.wall2 <- as.data.frame(Tab.wall) # convert to a data frame
View(Tab.wall2) # look at the tabulated data
```

and assign a name to the first column:

```
colnames(Tab.wall2)[1] <- "Wall Category"
```

c. Which property observation has the lowest price?

Useful functions to learn more about extreme values/outliers in the data set are `which.min()` and `which.max()`. To query, for instance, which house has the lowest price, type:

```
dat1[which.min(dat1$price),]
```

```
##       price yrbuilt stories tla                                     wall
## 13321  2000    1902     one  47 Horizontal or vertical wood siding or shakes
##       beds baths frontage    garage rooms lotsize longitude latitude
## 13321    1     1        9 no garage     3     111 -83.53762 41.66791
```

Observation 13,321 is the house with the lowest price (i.e., 2,000 money units). It is a single bed property with one bathroom, built in 1902 without garage.

d. Obtain selected descriptive statistics (i.e., the number of observations, mean, standard deviation, minimum and maximum values) for the numerical variables of dat1, using the pastecs package.

We can generate a set of useful descriptive statistics, which are typically included in academic/research papers, using the `pastecs` package. Install the package and use function `stat.desc()` to obtain a set of summary statistics for the **Lucas_County_data** and store these in a table:

```
library(pastecs) # loads the package
```

```
desc <- stat.desc(dat1) # generates table with:
# nbr.val, nbr.null, nbr.na, min max, range, sum
# median, mean, SE.mean, CI.mean, var, std.dev, coef.var
View(desc) # to take a look

# reduced table with selected summary stats (nbr.val mean std.dev min max)
desc2 <- desc[c(1, 9, 13, 4:5),]
desc2 # to show (note: appear in specified order)
```

```
##             price     yrbuilt stories         tla wall         beds             baths
## nbr.val  25357.00 25357.00000      NA 25357.00000   NA 2.535700e+04 nbr.val 2.535700e+04
## mean     79017.94  1945.34133      NA   135.84206   NA 2.987459e+00 mean    1.242024e+00
## std.dev  59655.02    27.88391      NA    56.96893   NA 7.226098e-01 std.dev 4.873255e-01
## min       2000.00  1835.00000      NA    11.00000   NA 0.000000e+00 min     0.000000e+00
## max     875000.00  1998.00000      NA   708.00000   NA 9.000000e+00 max     7.000000e+00
##     frontage garage        rooms  lotsize     longitude             latitude
##  25357.00000     NA 25357.000000 25357.000  2.535700e+04 nbr.val 25357.000000
##     19.50290     NA     6.114722  1238.598 -8.360282e+01 mean       41.657909
```

```
##     13.46975     NA     1.303297  2688.689  8.312366e-02 std.dev     0.046163
##      0.00000     NA     1.000000    65.000 -8.388239e+01 min        41.416896
##    247.00000     NA    20.000000 39865.000 -8.323993e+01 max        41.732258
```

```r
desc2.t <- t(desc2) # transposed version
desc2.t # to show
```

```
##             nbr.val         mean      std.dev        min          max
## price         25357 79017.943881 5.965502e+04 2000.00000 875000.00000
## yrbuilt       25357  1945.341326 2.788391e+01 1835.00000   1998.00000
## stories          NA           NA           NA         NA           NA
## tla           25357   135.842055 5.696893e+01   11.00000    708.00000
## wall             NA           NA           NA         NA           NA
## beds          25357     2.987459 7.226098e-01    0.00000      9.00000
## baths         25357     1.242024 4.873255e-01    0.00000      7.00000
## frontage      25357    19.502899 1.346975e+01    0.00000    247.00000
## garage           NA           NA           NA         NA           NA
## rooms         25357     6.114722 1.303297e+00    1.00000     20.00000
## lotsize       25357  1238.597941 2.688689e+03   65.00000  39865.00000
## longitude     25357   -83.602818 8.312366e-02  -83.88239    -83.23993
## latitude      25357    41.657909 4.616300e-02   41.41690     41.73226
```

Practical 2: Maps

In this practical we will explore how to plot geographical locations, e.g., of properties, using the `leaflet` package and related applications. To do this, we need to install the following packages: `leaflet`, `leaflet.extras`, `leaflet.providers`, `quadprog`, `tseries`, `tmap`, `tmaptools`, `shiny`, `shinyjs` and their dependencies.

a. Using the `leaflet` package, prepare a map widget for the Lucas_County_data in RStudio.

Once the packages have been installed, we can load the required packages into R Studio:

```r
# load required packages
library(leaflet)
library(leaflet.extras)
```

and start with the preparation of our map by generating a widget, called *m*:

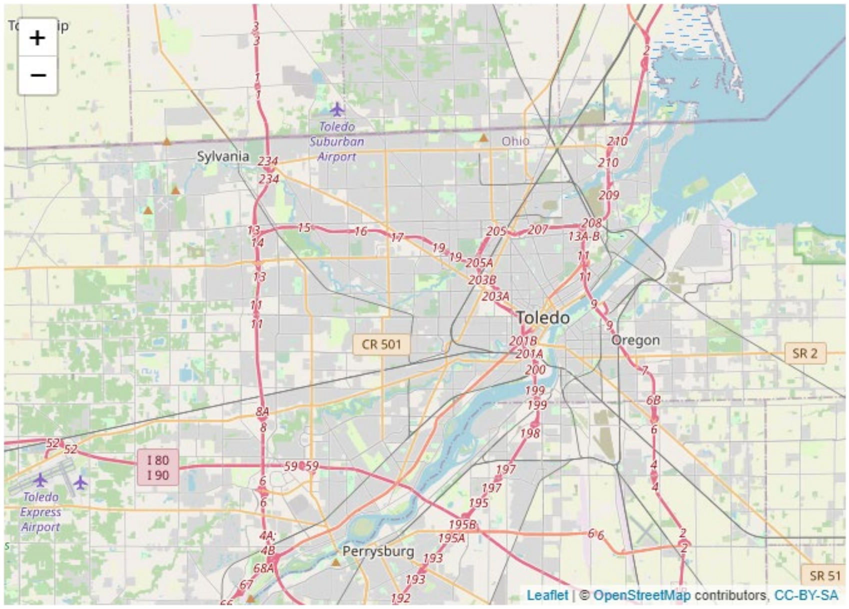
```r
m <- leaflet() # creates a map widget
```

Before we plot any data, we can adjust the map widget further to our needs. For example, we can center the map at a certain location (here: mean of the coordinates in the data frame **dat1**) and adjust the zoom level:

```r
# adjust view settings
m <- setView(m, lng = mean(dat1$longitude), lat = mean(dat1$latitude), zoom = 11)
```

We can also add further layers to the map (layer functions are: `addTiles()`, `addMarkers()`, `addPolygons()`).

```r
m <- addTiles(m) # adds tiles to map widget
m # display the map
```



Alternatively, tiles from certain providers can be added, using the `addProviderTiles()` function:

```r
# provider: ESRI & National Geographic World Map
m <- addProviderTiles(m, providers$Esri.NatGeoWorldMap)
m # display the map
```

ESRI is an acronym for *Environmental Systems Research Institute*, which is a *geographic information system (GIS)* software provider. Examples of other providers are: `Stamen.Toner`, `Stamen.TerrainBackground`, `NASAGIBS.ViirsEarthAtNight2012`.

b. Use the map widget to depict the variable *price* according to deciles, using the exact (geo-referenced) property locations.

We can also select colours to be used when plotting data in a map. The package `tmaptools`, which is part of `tmap` provides a handy tool to select color schemes and appendant R code. Install these packages and install and load `shiny` and `shinyjs` (possibly also requires the `Rcpp` package).

```r
# load required packages
library(shiny)
library(shinyjs)
library(tmaptools)
```

Then call the colour palette by typing (optional):

```r
tmaptools::palette_explorer() # shows color schemes and corresponding code
```

Note, that we have used the double colon operator (`::`), which is used to address/call a function (RHS of `::`) from a specific package/library (LHS of `::`).

Close the palette explorer before you continue with your work. We select the color scheme *plasma* and tell our computer to split it in ten groups and apply it in reverse order when generating deciles (ten quantiles) within the value range of variable *price* of data frame **dat1**, i.e., the house price data we loaded in Practical 1. Store the information in R object **pal.quantile**.

```r
pal.quantile <- colorQuantile("plasma",
                              domain =  dat1$price, reverse = TRUE, n = 10)
class(pal.quantile) # stored as a function
```

```
## [1] "function"
```

Note that line breaks are not a problem in R Script, i.e., they are treated like an unfinished command line in the *Console* (cf. *Tutorial 1*) and code is only run upon completion.

We can store the colour classification of the ten quantiles (i.e. deciles) in a new variable, called *price.colors.quant*, of **dat1**.

```r
# colour code for each observation, based on value expressions of variable price
dat1$price.colors.quant <- pal.quantile(dat1$price)
head(dat1$price.colors.quant) # see how it came through
```

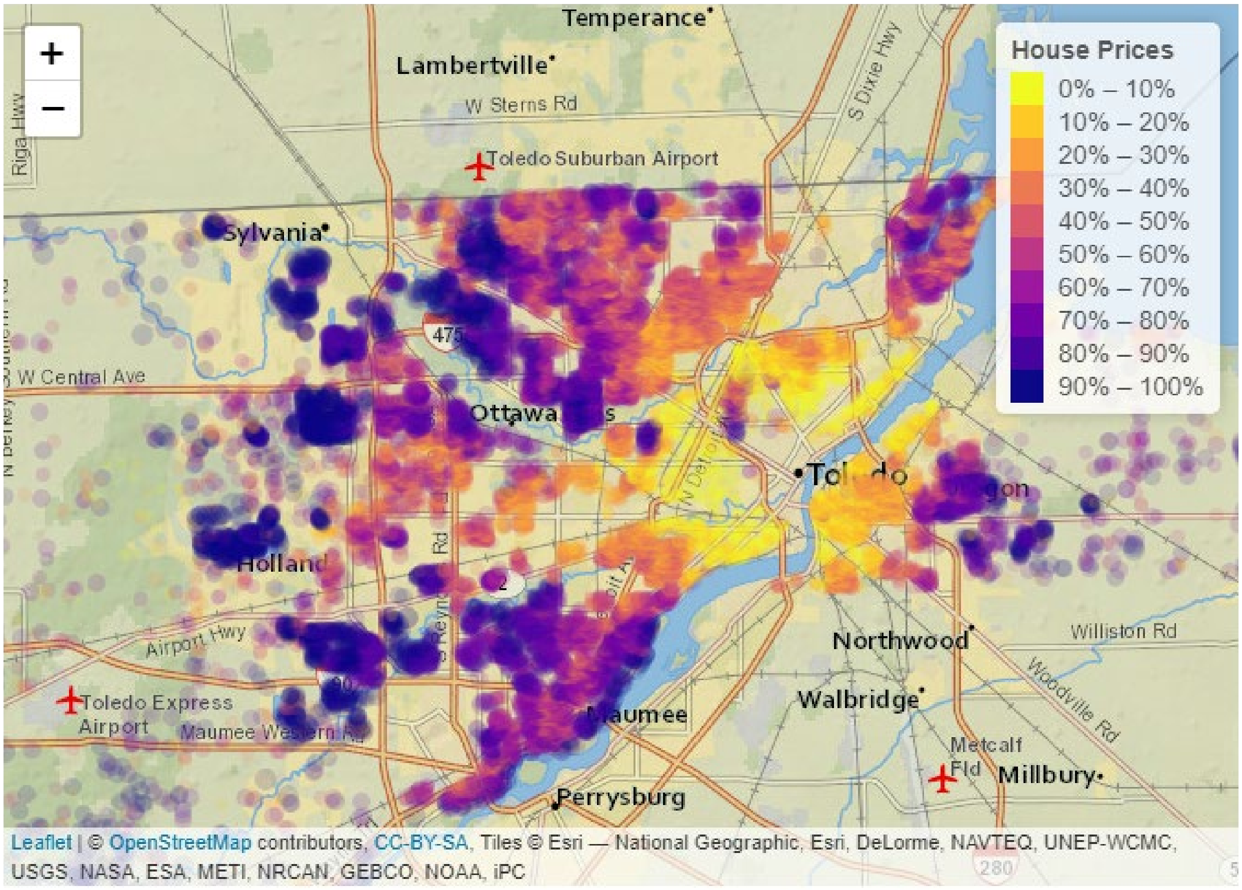
```
## [1] "#0D0887" "#7301A8" "#7301A8" "#0D0887" "#0D0887" "#7301A8"
```

We want to plot the property locations (coordinates are given in **dat1** by variables *longitude* and *latitude*) using circle markers. The size of the circle markers could also represent information on a property's price (in thousands), i.e., larger circles represent higher prices, and the colours should indicate the price quantiles, respectively. To produce such a map type:

```r
m <- addCircleMarkers(m, lng = dat1$longitude,lat = dat1$latitude,
                      radius = log(dat1$price/1000), stroke = FALSE,
                      fillOpacity = 0.15, fill = T,
                      fillColor =  dat1$price.colors.quant)
m # to view (output omitted in this workbook)
```

To add a legend to the map use the `addLegend()` function after running the previous code.

```r
m <- addLegend(m, pal = pal.quantile, values = dat1$price, opacity = 1,
               title = "House Prices")
m # to view
```



To show the quantiles of *price* type:

```r
quantile(dat1$price, probs = seq(0,1,0.1))
```

```
##     0%    10%    20%    30%    40%    50%    60%    70%    80%    90%   100%
##   2000  22819  36000  47000  56900  65500  76000  88100 111000 148404 875000
```

Note that the `seq()` function is another way to produce a sequence (i.e., other than the colon (`:`) operator, introduced in *Tutorial 1*). The values in the brackets tell your computer to produce a sequence running from "0" to "1" in steps of "0.1".

c. Prepare a new map widget for the Lucas_County_data. Use cluster markers to depict the spatial density of property observations.

We set up a fresh widget for the new map and add a layer from a different provider this time.

```r
# pieces of code from before
m2 <- leaflet()
m2 <- setView(m2, lng = mean(dat1$longitude), lat = mean(dat1$latitude), zoom = 11)
```

```r
# new: add tile from different provider
m2 <- addProviderTiles(m2, providers$Stamen.Toner)
m2 # to view (output omitted in this workbook)
```

Then we add the circle markers:

```r
m2 <- addCircleMarkers(m2, lng = dat1$longitude, lat = dat1$latitude,
                       clusterOptions = markerClusterOptions())
m2 # to view
```