Data Analytics in Energy Informatics

Section 3, Introduction to R 3

Prof. Dr. Thorsten Staake, Carlo Stingl and Joanna Graichen

Version 1.1

Agenda

1. Advanced R for Data Analysis – dplyr and Piping
    1. Cheatsheets
    2. Basic Data Operations
    3. Grouped Calculations
    4. Pipelining
    5. Joining data frames

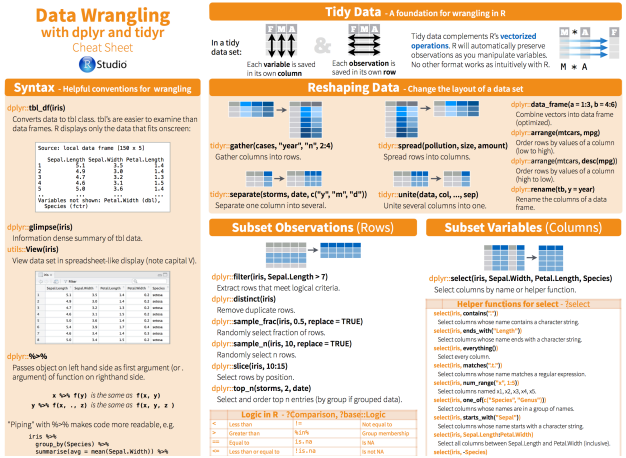
2. Programming with R
    1. Control structures
    2. Missing data
    3. Debugging
    4. Plotting

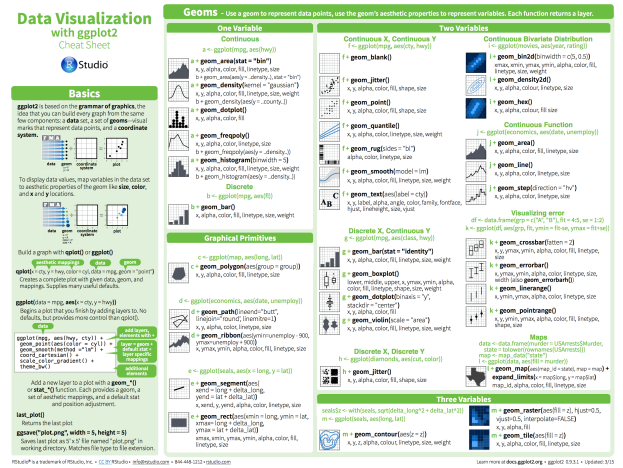
Adapted version of Hadley Wickham's "Data Manipulation with Dplyr"

Cheatsheets (i.e., dplyr and ggplot2)

- A cheatsheet provides an in-depth overview of functionalities of a certain package.
- When learning how to apply certain packages, first check whether there exists a cheatsheet for it.
- Some examples for cheatsheets:

Package	Link
dplyr/tidyr	https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
ggplot2	https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf





Mutate

- mutate allows to add new columns based on other row-specific values

```
mutate(shower_data, squaredVolume = Volume^2)
```

Hh_ID	Volume
1	30
1	33
1	23
2	40
2	35

Hh_ID	Volume	squaredVolume
1	30	900
1	33	1089
1	23	529
2	40	1600
2	35	1225

Summarize

- Summarize transforms variables to numbers

```
summarize(shower_data, sumVolume = sum(Volume),
                       maxVolume = max(Volume))
```

Hh_ID	Volume
1	30
1	33
1	23
2	40
2	35

sumVolume	maxVolume
161	40

- Typically, you want to perform such operations on grouped data (i. e., on each household)

Grouping

- Grouping based on certain variables (here: Hh_ID)

```
grouped_showers <- group_by(data, Hh_ID)
summarize(grouped_showers, sumVolume = sum(Volume),
                           maxVolume = max(Volume))
```

Hh_ID	Volume
1	30
1	33
1	23
2	40
2	35

Hh_ID	sumVolume	maxVolume
1	86	33
2	75	40

Exercise II (dplyr)

- Up to you:

1. Determine the maximum and the minimum shower duration of the data set.
2. Introduce a new column “Avgtemperaturefahrenheit”. Therefore, convert the column “Avgtemperature” to the Fahrenheit unit using the package weathermetrics.
3. Calculate the average shower volume for each household , the average shower temperature and the average shower duration.

Data operations create a lot of intermediate variables

- In order to calculate specific measures, one often has to filter, group, summarize and arrange a data set.

- However, this procedure creates a lot of intermediate variables, which are not needed – resulting in very verbose code , such as:

```
filtered_data <- filter(exemplary_data, Hh_ID != 1)

grouped_showers <- group_by(filtered_data, Hh_ID)

summarized_showers <- summarize(grouped_showers, sumValue =
sum(Volume), maxValue = max(Volume))

ordered_showers <- arrange(summarized_showers, Hh_ID)
```

Data operations create a lot of intermediate variables (cont'd)

- The presented data operations always take a data frame as first argument and return a data frame
- dplyr provides a pipe operator, which can be used to pass the resulting data frame to the next operation

```
exemplary_data %>% filter(Hh_ID != 1) %>%
                   group_by(Hh_ID) %>%
                   summarise(sumValue = sum(Volume),
                             maxValue = max(Volume)) %>%
                   arrange(Hh_ID)
```

Exercise III (dplyr)

- Up to you! Use data pipes to

1. Calculate for each household the average shower volume, the average shower temperature and the average shower duration.
2. Determine the number of devices (=households), which have recorded more than 50 showers.
3. Calculate the average number of recorded showers per group.

Joining data frames

- When using data from different files we need to efficiently combine data sets
- Exemplary data set:

```
shower_data <- data.frame(Hh_ID = c(1,1,1,2,2),
                          Volume = c(30, 33, 23, 40, 35))

survey_data <- data.frame(Hh_ID = c(1,2),
                          Longhair = c("male", "female"))
```

Hh_ID	Volume
1	30
1	33
1	23
2	40
2	35

?

Hh_ID	gender
1	male
2	female

Joining data frames – Methods

Type	Syntax	Action
Inner	inner_join(x, y)	Includes only rows in both x and y.
Left	left_join(x, y)	Includes all of x, and matching rows of y.
Semi	semi_join(x, y)	Keeps all observations in x that have a match in y.
Anti	anti_join(x, y)	Drops all observations in x that have a match in y.

Source: Wickham & Grolemund (2017): R for data science

Joining data frames – Example

x

Hh_ID	Volume
1	30
1	33
1	23
2	40
2	35
3	55

+

y

Hh_ID	Gender
1	male
2	female
3	male

=

Hh_ID	Volume	Gender
1	30	male
1	33	male
1	23	male
2	40	female
2	35	female
3	55	male

```
inner_join(x, y)
```

Exercises (Joining data frames)

- In addition to the shower data, we have also collected survey data as part of a study. We want to evaluate these in order to investigate the influence of individual parameters (e. g., age, gender, hair length, environmental attitude) on the recorded water and energy use.

- Now it's up to you:
  1. Download the data set "Shower_survey_data.csv" from the edX site.
  2. Perform an inner join on the shower and the survey data (based on "Hh_id").
  3. Group the data frame by the column "X03d_longhair" and "gruppe".
  4. Summarize the data frame by calculating the average shower volume and the average shower duration.

Control structures

- conditionals: `if`
  - `if(<condition>){<if_true>}else{<if_false>}`
  - `ifelse(<condition>,<if_true>,<if_false>)` --> vectorized if
- loops with `while, repeat, for`:
  - `while(<condition>){<instructions>}`
  - `repeat{<instructions>}`
  - `for(<counter> in <vector>){<instructions>}`
- loop control with `break, next`

Apply

- Much faster than loops due to the vectorized execution of the calculation
- To apply a function to all columns or rows of a matrix:

```
apply(<matrix>,<1/2>,<function>)
```

- The second argument specifies the dimension for which the function is applied (E.g., 1-rows, 2-columns)
- Further versions exist
  - `sapply(<vector>,<function>)`
  - `lappy(<list/vector>)`

Working with missing data

- In R, missing values are represented as `NA` (not available)
- Test for missing values with `is.na(x)`
- Excluding missing values with `na.omit(x)` or `complete.cases()`, mind that all columns are considered!
- Marking missing values, e.g.
  `mydata$v1[mydata$v1==99] <- NA`

Debugging in R

- An error aborts the program flow and prints it‘s error message. `traceback()` can be used to show the call stack.
- The command `browser()` allows to execute functions step by step. During this process it is possible to see and change all values.
- `debug(<function_name>)` enters the browser each time the function with this name is executed.
- Using `options(error=recover)` it is possible to enter the browser if an error occurs.
- Read more on debugging and exception handling: http://adv-r.had.co.nz/Exceptions-Debugging.html

Brief introduction to ggplot2

- ggplot2 offers a grammar of graphics, providing methods for dealing with the following fundamental parts of a data graphs:

Source: http://www.science-craft.com/wp-content/uploads/2014/06/ggplot-3.png

Creating our first scatter plot

- Our first ggplot2 visualization on the data

Basic plot object

Data frame

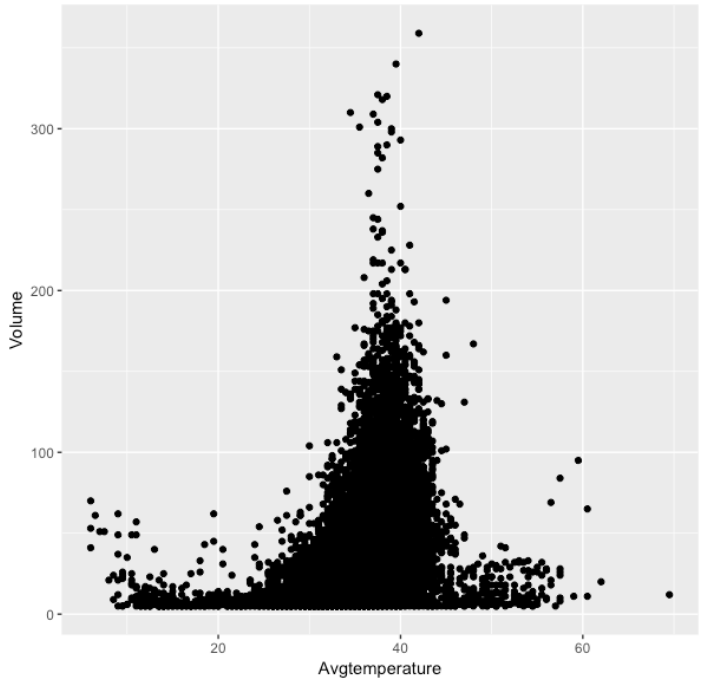
x-column

y-column

```
g <- ggplot(Shower_data, aes(x=Avgtemperature, y=Volume))
g <- g + geom_point()
g
```

Instruction layer

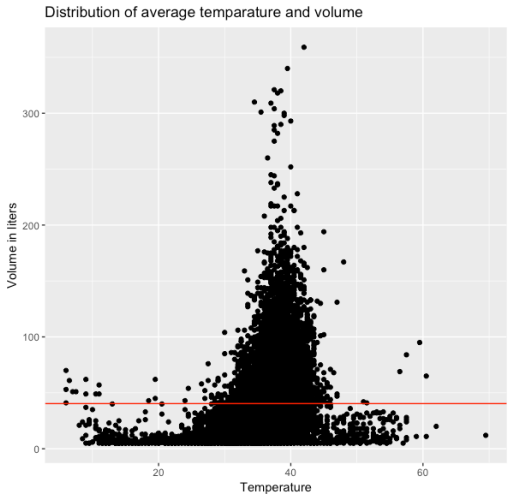
Print the plot



Refining the plot

- We can refine the plot by adding additional labels and layers to the plot

```
g <- g + ggtitle("Distribution of average temparature and
                  volume")
g <- g + xlab("Temperature")
g <- g + ylab("Volume in liters")
g <- g + geom_hline(yintercept = mean(Shower$Volume),
                    color="red")
```



Geom types

- Besides the presented scatter plot, ggplot2 allows to create the following types of plots:
  - Histograms
  - Boxplot
  - Bars and Columns
  - Stacked Bar Chart and Pie Chart
  - Line chart
  - Bubble Chart
  - Area Chart

- Most of the require only few modifications on the presented basic principle

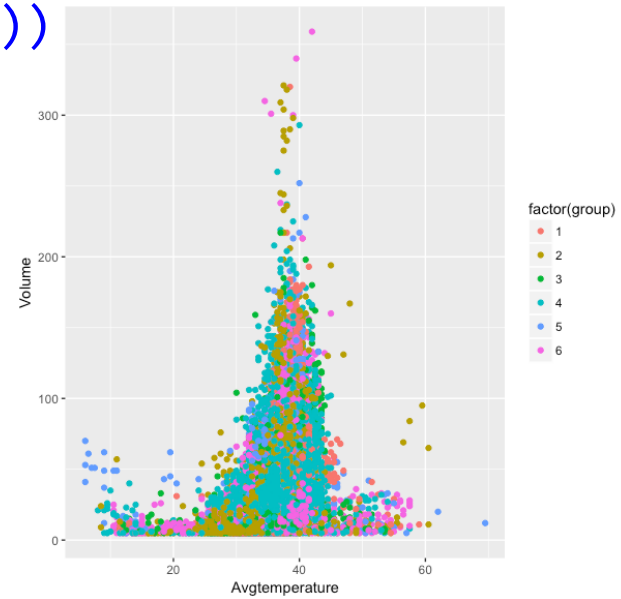
Going beyond standard graphics

- ggplot2 allows for much more sophisticated data visualizations by leveraging meta data of the supplied data.
- As such, we could color the data points of the presented plot based on the study group membership of the individual households.

```
g <- ggplot(Shower, aes(x=Avgtemperature, y=Volume,
                        color=factor(group)))

g <- g + geom_point()
g
```

Conditional formatting

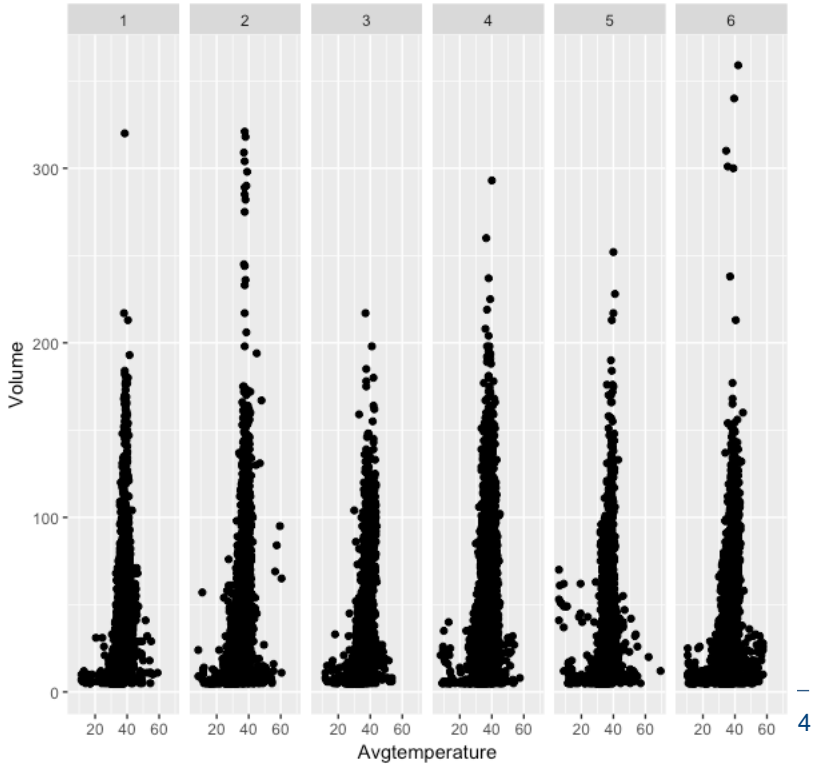


Facetting (Trellis Plots)

- Such overlapping graphics often can get confusing and hart to understand. Thus, one often wants to create a plot per condition (or here: group membership).
- With ggplot2, you can do this with one line of code.

```
g <- ggplot(Shower, aes(x=Avgtemperature,
y=Volume))
g <- g + geom_point()
g <- g + facet_wrap(~group, nrow = 1)
g
```

Facetting method



Exercises (ggplot2)

Up to you:

1. Visualize shower time and volume by creating a scatter plot.
2. Use logarithmic scale for the plot.
3. Create a boxplot of shower time.
4. Plot a histogram of the income column (displaying the number of households per income category).
5. Create a density plot of the shower volume using facetting on the group variable.

End of this subsection.