Advanced R workshop
Day 2

Dr Christina Ramsenthaler MSc PhD
ramn@zhaw.ch, ramsenthalerchristina@gmail.com (Dr R's Stats Corner)
Research methods and Statistics
Winterthur (ZHAW), King's College London & Hull York Medical School

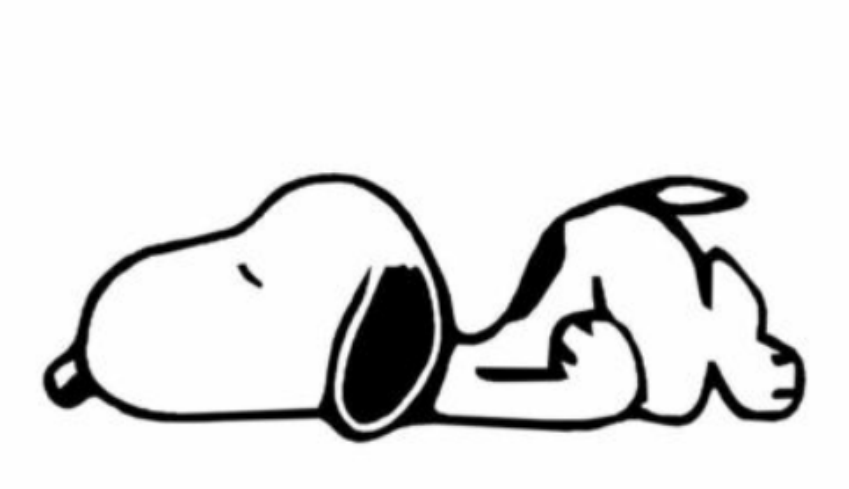
Advanced R workshop
Freiburg, 2022-10-19

Roadmap

Day 2	R projects, Workflow, Data management
Session 1: 09.00-10.50	
09.00-09.30	Data cleaning functions 3
09.35-10.05	R function quiz
10.10-10.40	Analysis, tables and graphs 1
10.40-10.50	Longer break, buffer 1
Session 2: 10.50-12.30	
10.50-11.20	Analysis, tables and graphs 2
11.25-11.55	Workshop: Analysis scripts
12.00-12.30	Working on own dataset
12.30-13.30	1h lunch break
Session 3: 13.30-15.00	
13.30-14.00	.Rmd, dashboards etc
14.05-14.35	Git and version control
14.40-15.10	Working on own dataset
15.10-15.20	Longer break, buffer 2
Session 4: 15.20-17.00	
15.20-15.50	Wildcard: whatever you like talking about
15.55-16.25	Wildcard: Looking at data analysis questions
16.30-17.00	Working on own dataset / Q&A

Over to the R quiz

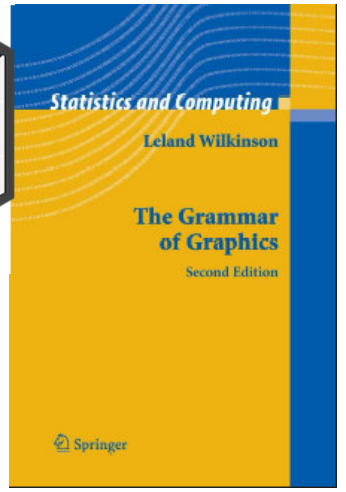
Short break



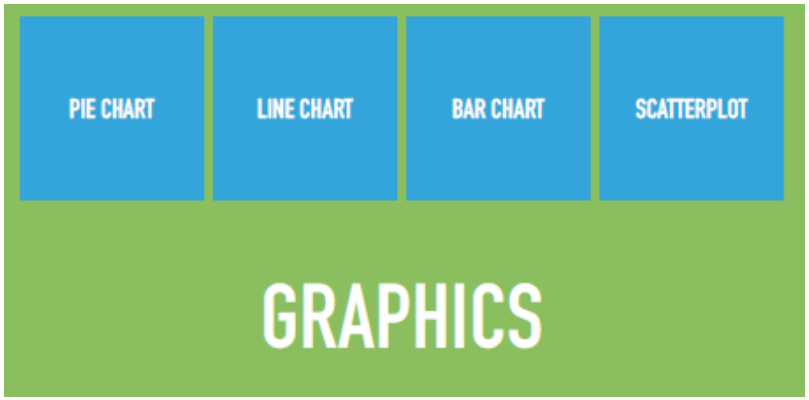
Intro to ggplot2

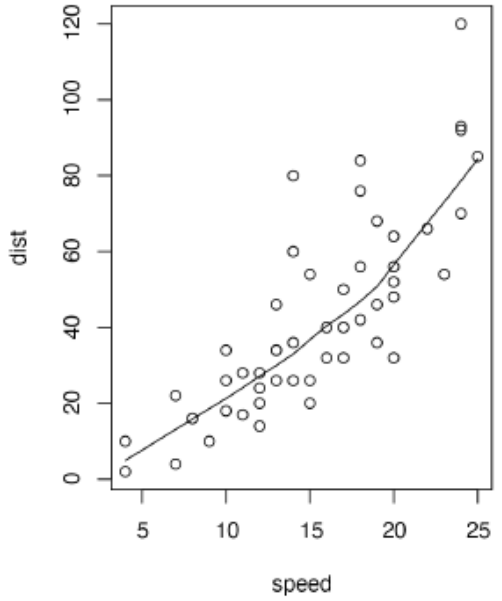
1. **ggplot: The grammar of graphics**
2. The basics – layers
3. The most important geoms
4. Faceting
5. Adding elements
6. Extensions to ggplot and literature

ggplot – The grammar of graphics



- Book Leland Wilkinson (1999): theoretical deconstruction of graphs
- Inspired Tableau, ggplot2 and others
- The book is very technical. However, it breaks down each type of graph into its building-block elements. In combination, these can give very nice graphs.
- ggplot2 is essentially a programming representation of this building-block system, just like Lego®.





ggplot – The idea (Building blocks)

How should my plot look like?

Inserting data points on the plot using x,y,z coordinates

Are data of groups broken up into panels?

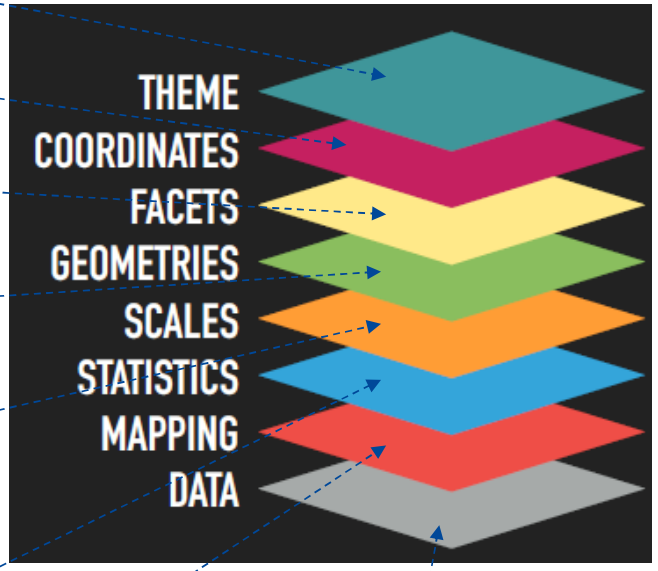
Geom(etrie)s define the type of graph for the data – bar plot, scatter plot, histogram, line plot, tile plot.

Scales are assigned. Range of scale can be determined. Categories -> Colours; Numbers -> positions on the graph.

Statistics define how data shown on the plot are transformed, i.e., categorical data for frequencies.

Aesthetic Mapping: Linking variables in the data with elements in the plot (x,y).
Facet Mapping: Linking variables in the data with single panels in the plot.

Data form the basis. They should be cleaned for ggplot, meaning 1 variable = x axis etc.



Intro to ggplot2

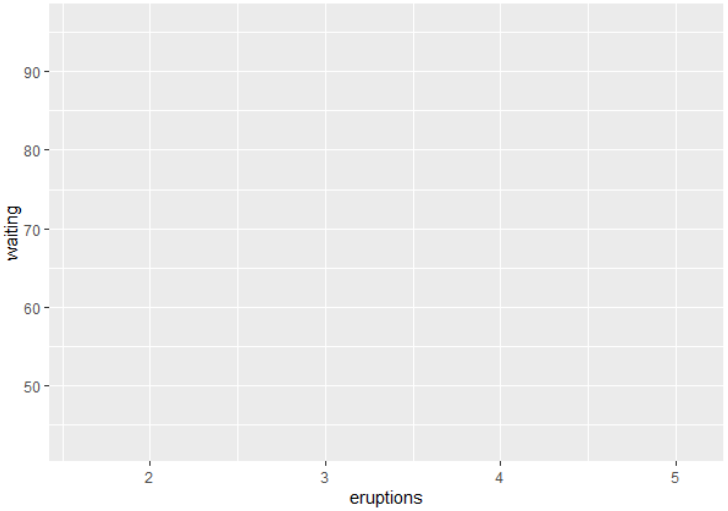
1. **ggplot: The grammar of graphics**
2. The basics – layers
3. The most important geoms
4. Faceting
5. Adding elements
6. Extensions to ggplot and literature

ggplot – The basics

- ggplot2 is a graph system. Syntax logic is slightly different to everything else in R. However, ggplot is fully integrated with base R and dplyr functions.
- ggplot2 syntax works in layers. You draw on a canvas like you would draw on piece of paper.
- A lot of the basics here are shown using a built-in dataset called faithful (number of eruptions of the Faithful volcano).
- At its basis, the mapping produces a coordinates system of x and y axis. The dataset needs to contain an x and a y variable for most geoms.

```
library(ggplot2)
ggplot(data = faithful,
       mapping = aes(x = eruptions,
                     y = waiting))
```

Load the package
ggplot: Build me a grammar of graphics plot.
data: Use the dataset faithful.
mapping: What are my x and my y coordinates? The x axis should contain the number of eriptions. The y axis is the waiting time. -> ggplot always needs to map variables in the dataset to the axes and other elements in the plot.



Via the ggplot (data, aes) call, we haven't yet told R what should be drawn on the canvas.
This happens via choosing a specific geom.

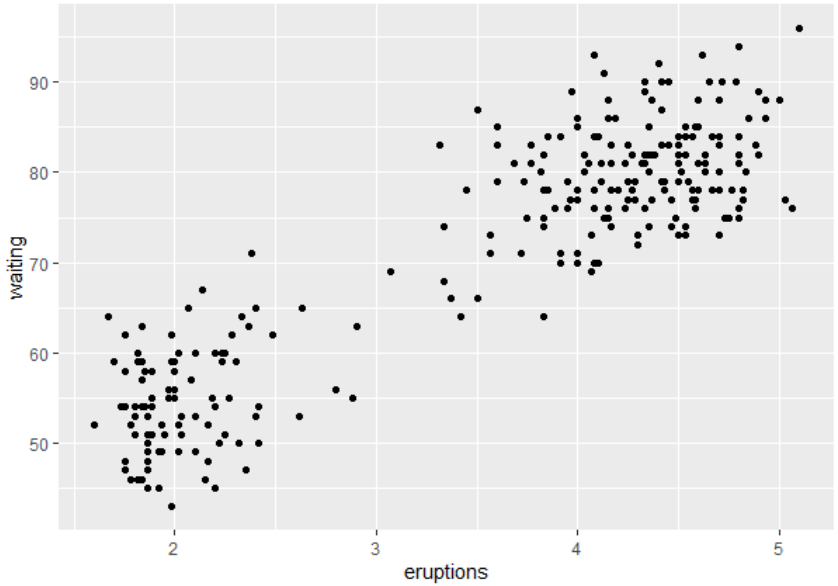
ggplot – The basics

- To show data on a canvas, we need to tell R what should be drawn. Here, we want dots in scatterplot representing pairs of observations of eruptions and waiting time.

```
library(ggplot2)
ggplot(data = faithful,
       mapping = aes(x = eruptions,
                     y = waiting))+
  geom_point()
```

geom_point(): Geoms specify the type of graph. Here, we want a scatterplot.

ggplot2 works with + as the sign for building blocks of the graph. This is unlike any other type of syntax in R.
Whenever your graph does not run, the most likely reason is forgetting the '+' at the end of a building block syntax line.



ggplot – The basics

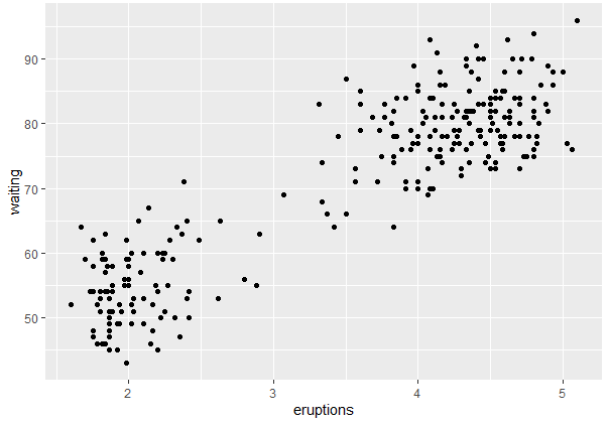
- There is always more than one way in R: ggplot syntax is flexibel, the + allows free combination of layers and elements. The result is the same. The choice depends largely on how you want to introduce colour.

```
ggplot(data = faithful,
       mapping = aes(x = eruptions,
                     y = waiting))+
  geom_point()
```

```
ggplot(data = faithful) +
  geom_point(mapping = aes(x = eruptions,
                           y = waiting))
```

```
ggplot() +
  geom_point(mapping = aes(x = eruptions,
                           y = waiting),
             data = faithful)
```

All these are variations that produce the same graph.

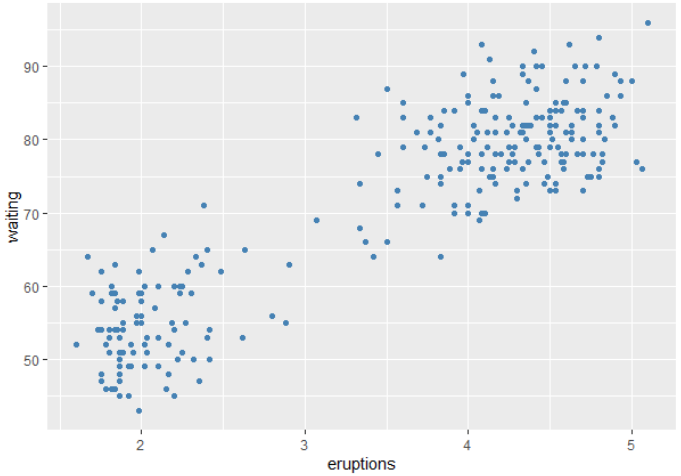


ggplot – The basics

- How can we get this colourful? This depends on what colour should represent. You might want to give each dot the same colour. Or you want to use colour only for certain groups or for certain ranges of a continuous variable.

Here we use the same colour for each dot.

```
ggplot(faithful) +
  geom_point(aes(x = eruptions,
                 y = waiting),
             colour = 'steelblue')
```



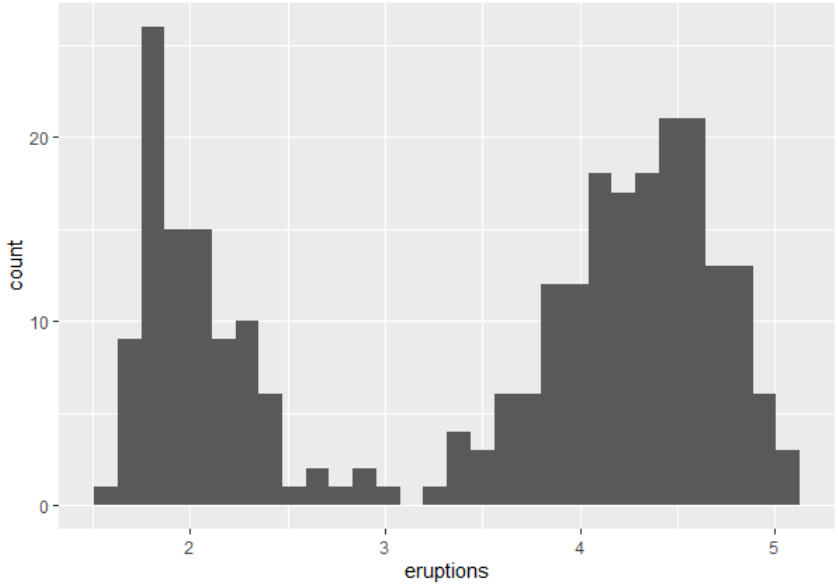
A different colour is mapped to different ranges of the variable eruptions. One colour represents less than 3 eruptions, the other 3+ eruptions.

```
ggplot(faithful) +
  geom_point(aes(x = eruptions,
                 y = waiting,
                 colour = eruptions < 3))
```

ggplot – The basics

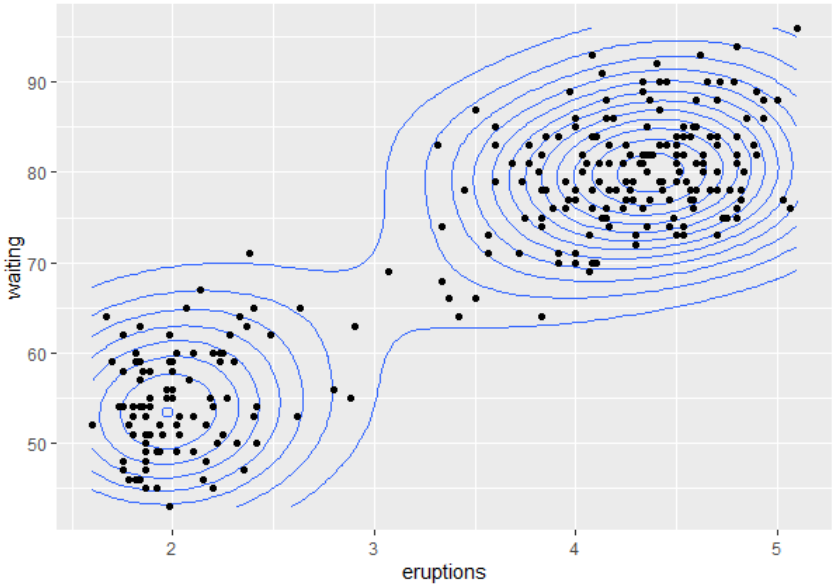
- There is a large variety of geoms. Each geom is a different type of graph.

```
ggplot(faithful) +
  geom_histogram(aes(x = eruptions))
```



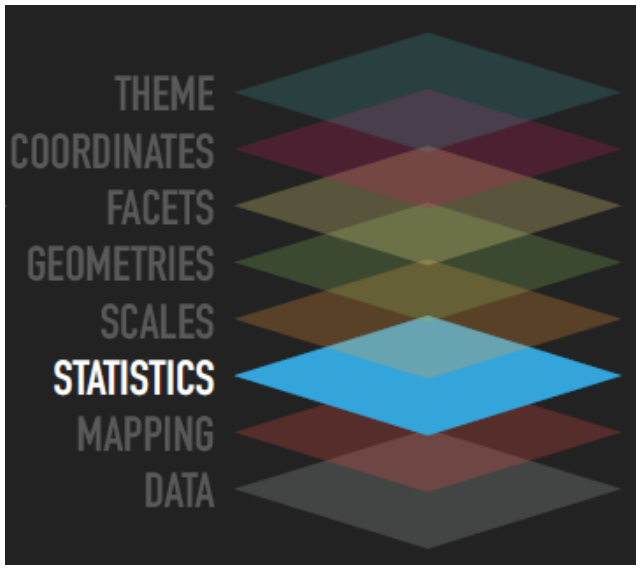
- R also combines low-level geoms in the order they are specific in the ggplot function call.

```
ggplot(faithful,
       aes(x = eruptions, y = waiting)) +
  geom_density_2d() +
  geom_point()
```



ggplot – The basics

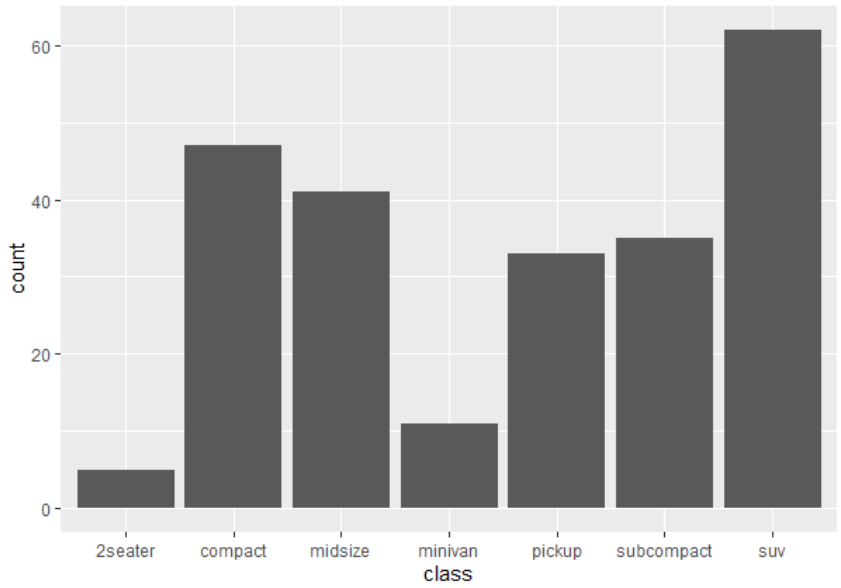
- After specifying the data and what should be drawn on the x and y axis, R wants to know what type of descriptive statistics it should use. This depends on the geom and different geoms need different specifications.
- However, again we have a lot of flexibility:
  - You can simply use the default option for a geom, e.g., geom_bar(). Geom_bar() without further information uses counts as the default statistic.
  - You can also skip this step and define your descriptive stats in your dataset (explicitly in a variable). You also work with stat = “identity” in this case. (This is the thing I most often do.)
  - You could also use a more specific geom like geom_col (specifying that you want a bar chart via this geom). Then you define the x and y variable via aes().
  - You can use the after_stat() function to specify the type of statistic you want to calculate (e.g., percentage frequencies)



ggplot – The basics

- We use a geom_bar() for a factor variable and that‘s it – counts appear by magic.

```
ggplot(mpg) +
  geom_bar(aes(x = class))
```



- Sometimes, you will have the counts as a variable in your dataset. You can now give the count as the y variable directly to the aes() call.
- You need to specify stat = “identity“ to take the values of the y variable as they are.

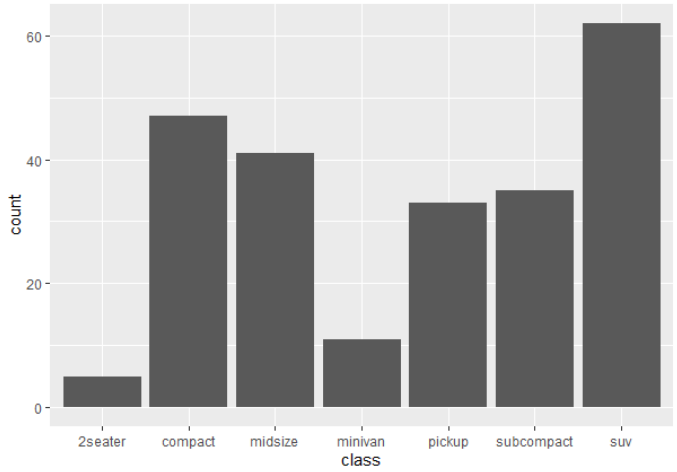
```
library(dplyr)
mpg_counted <- mpg %>%
  count(class, name = 'count')
ggplot(mpg_counted) +
  geom_bar(aes(x = class, y = count),
           stat = 'identity')
```

This is a dplyr reprsentation of table(mpg$class).

ggplot – The basics

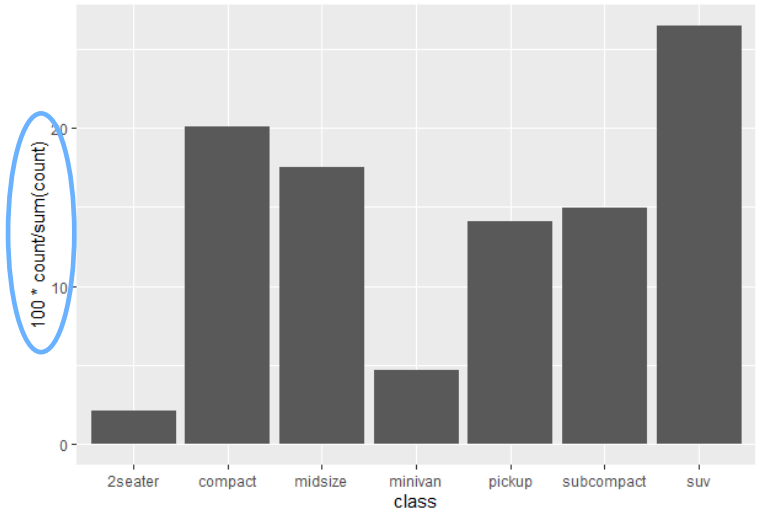
- We get the same result via geom_col as long as count holds the absolute frequencies in the dataset.

```
mpg_counted <- mpg %>%
  count(class, name = 'count')
ggplot(mpg_counted) +
  geom_col(aes(x = class, y = count))
```



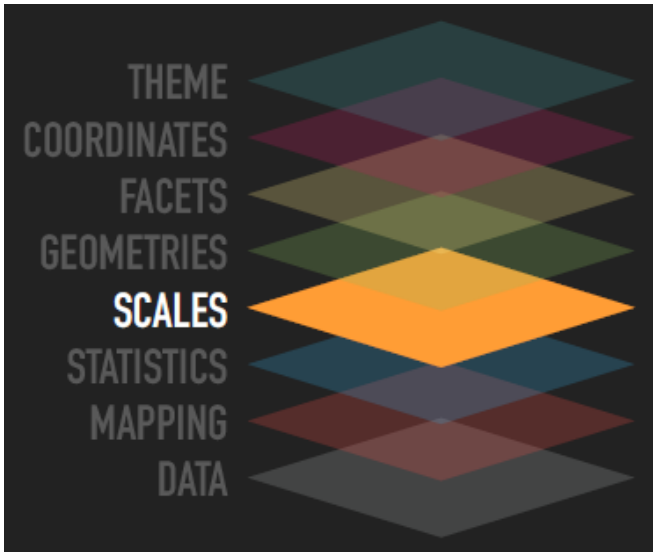
- This uses after_stat() to calculate a percentage using the y = count variable.
- The calculation is added as the y axis label.

```
ggplot(mpg) +
  geom_bar(
    aes(
      x = class,
      y = after_stat(100 * count / sum(count))
    )
  )
```

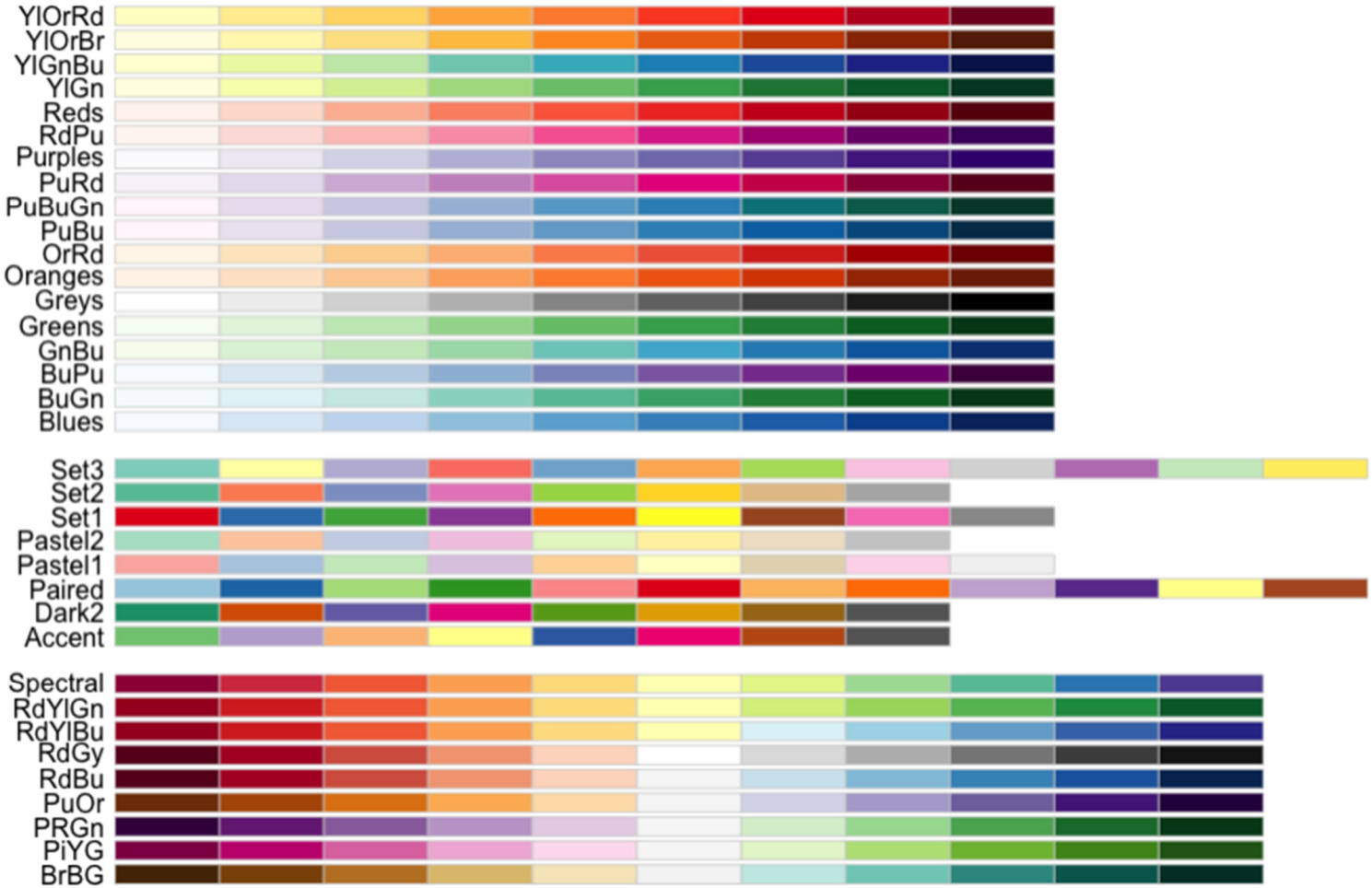


ggplot – The basics

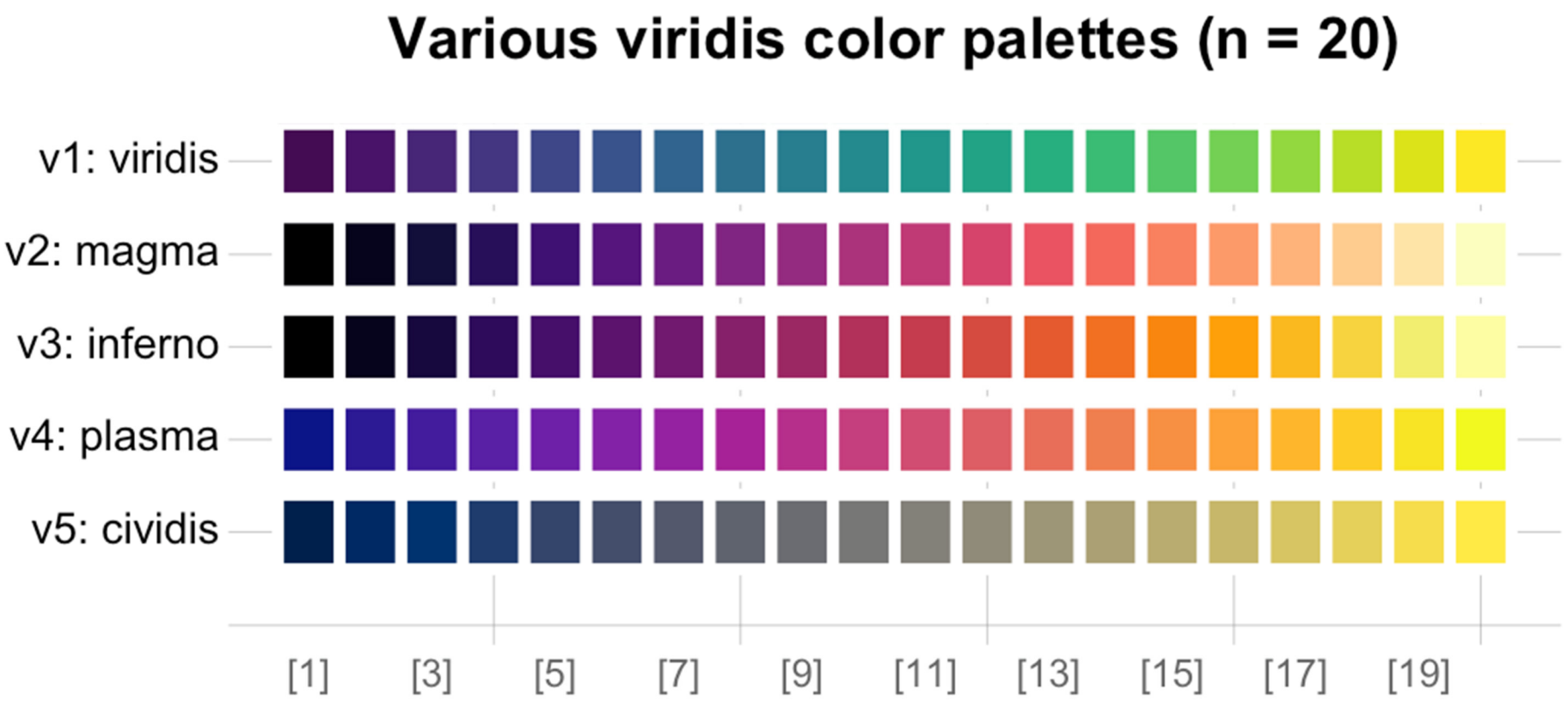
- Scales let you do stuff to the scales of your plot. Most often, the scales argument will be the place where you define colour.
  - The x and y axis are controlled via the scale_() argument. You can change the range of an axis, how tick marks are handled and displayed, the units of measurement etc.
  - A word about colour: Mapping colours to the plot (either as fill colours or line colours) requires a specification of colour or fill in the aes() argument (mapping layer). There will be a difference between defining the mapping layer in the ggplot() or in the geom() call.
  - If you want to use other than built-in colour schemes, you will need to map colour/fill in aes() and then proceed by specifying the colour names or colour/fill palettes in a scale_() argument.
  - There are lovely colour palettes available for R. I work with:
    - RColorBrewer
    - viridis palette (great for accomodating colour blindness)
    - yarrr (The Pirate‘s Guide to R) palette
    - hrbrthemes



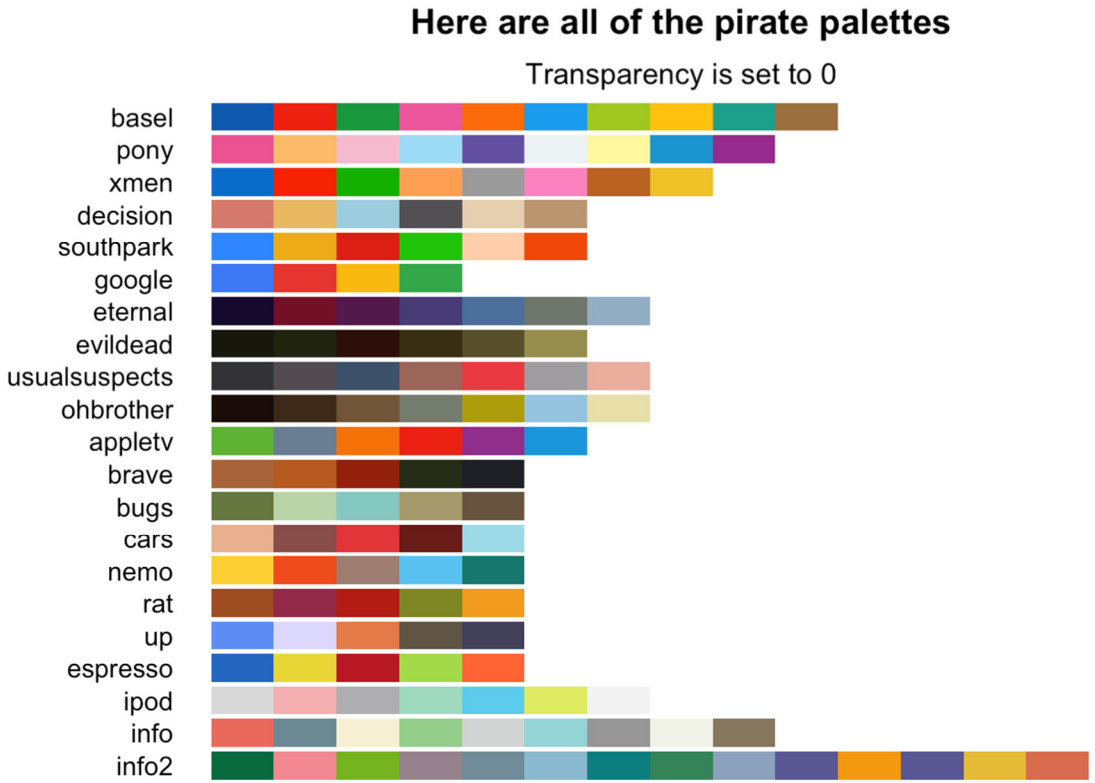
RColorBrewer Palettes



Viridis palette



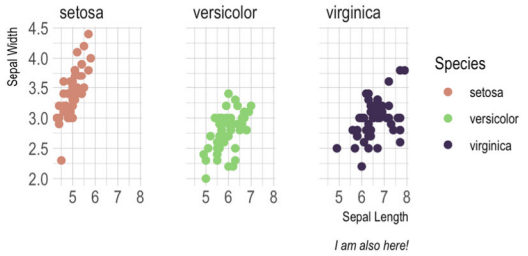
YaRrr pirate palette



hrbrthemes (great autumn colours palette)

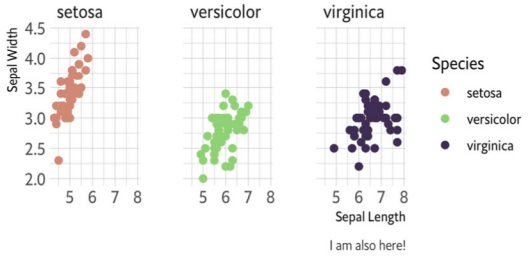
theme_ipsum

Arial Narrow



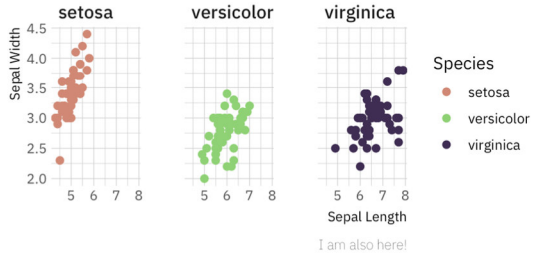
theme_ipsum_es

Econ Sans



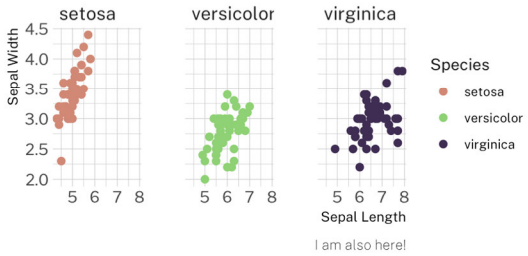
theme_ipsum_ps

IBM Plex Sans



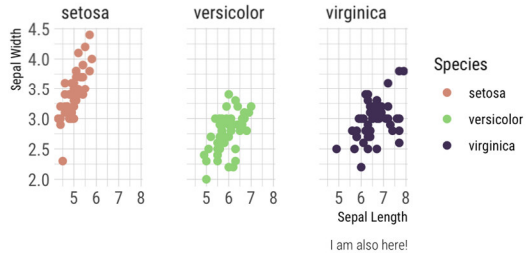
theme_ipsum_pub

Public Sans



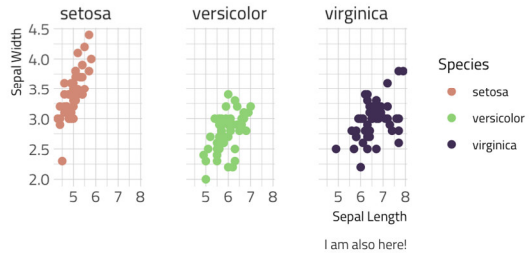
theme_ipsum_rc

Roboto Condensed



theme_ipsum_tw

Titillium Web

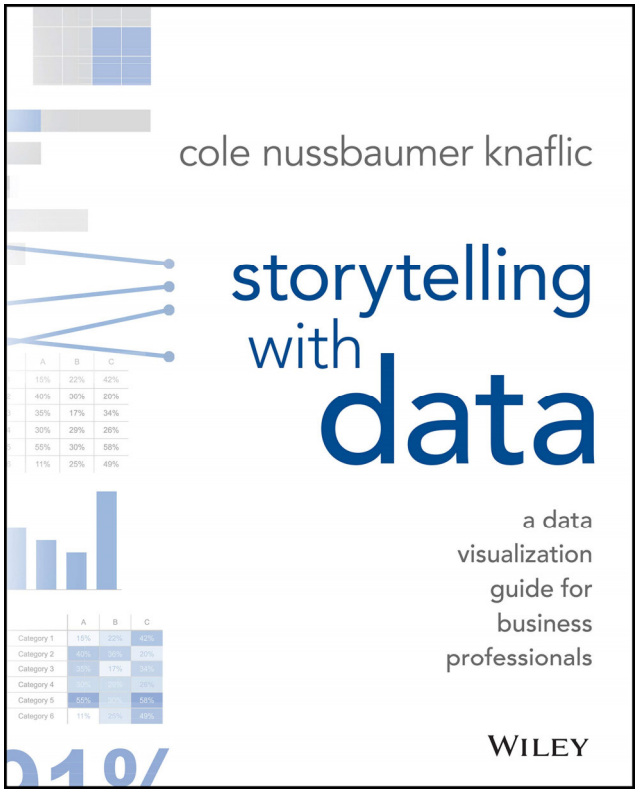


Colour palettes

- You can always build your own colour palettes.
- Colour schemes are tied to corporate branding. I often need to work using a corporate branding scheme. It‘s relatively easy in R to build user-defined functions using your colour palette.

- It can be highly satisfying to spend time on creating beautiful visualisations. I can highly recommend working with a designer. Consider using hex colour codes. See: https://coolors.co/gradient-palette/ for building your own hex colour palettes.

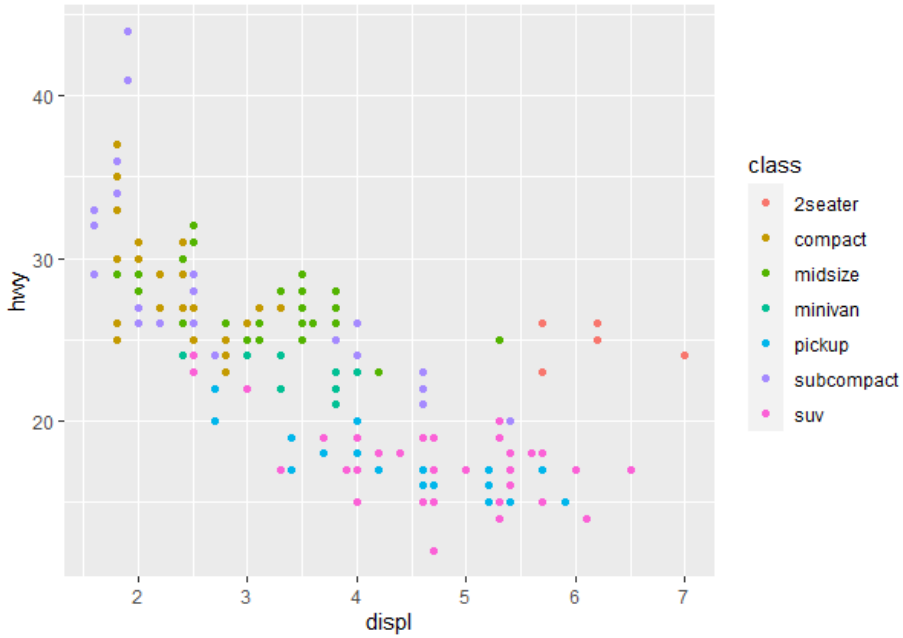
- The book Storytelling with data is highly recommended!



ggplot – The basics

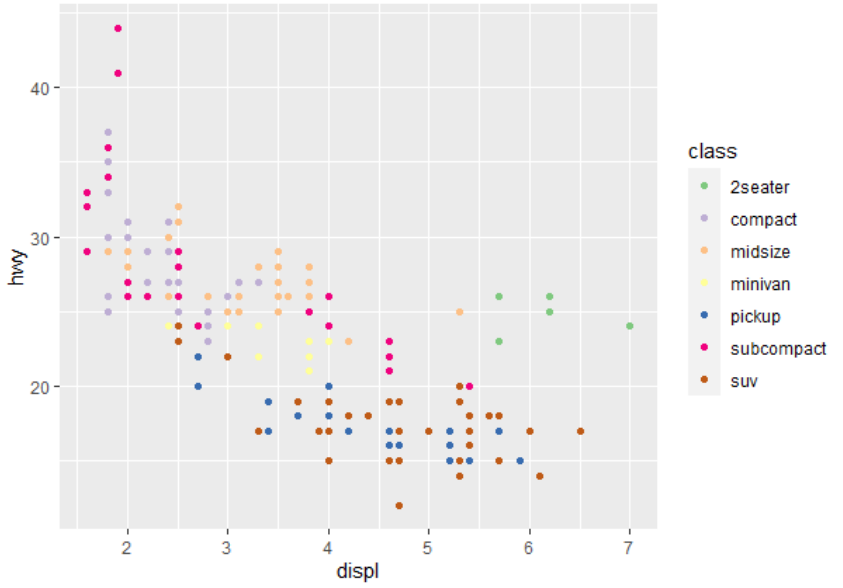
- R‘s built-in colour scheme

```
ggplot(mpg) +
  geom_point(
    aes(x = displ, y = hwy, colour = class)
  )
```



- RColorBrewer Palette:
  - assign variable to colour in aes() call
  - Use scale_colour_brewer() function.

```
ggplot(mpg) +
  geom_point(
    aes(x = displ, y = hwy, colour = class)) +
  scale_colour_brewer(type = 'qual')
```



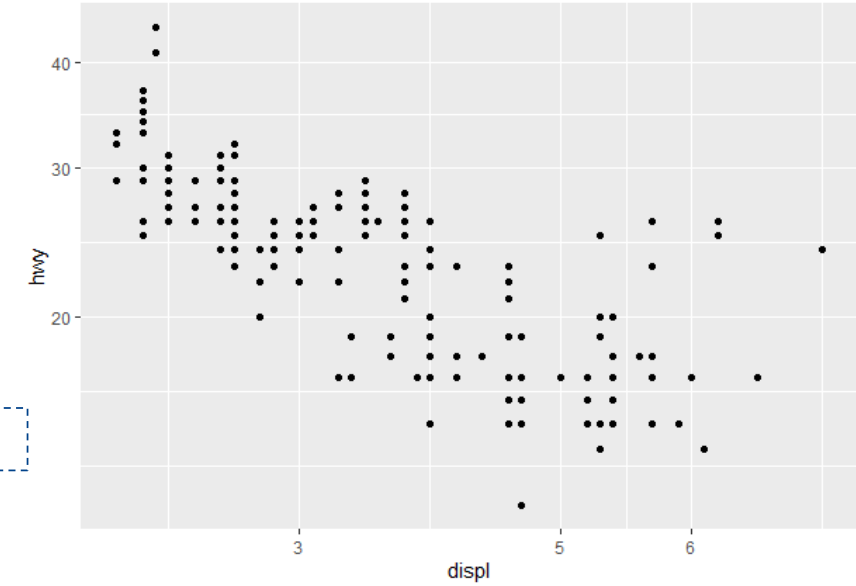
ggplot – The basics

- Changing axes appearance also works via the scale() argument.

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  scale_x_continuous(breaks = c(3, 5, 6)) +
  scale_y_continuous(trans = 'log10')
```

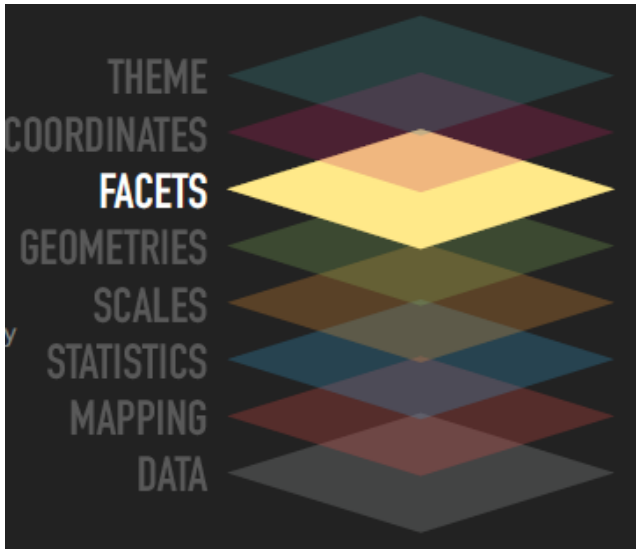
This specifies breaks for the x axis – I only want axis tick labels at 3, 5 and 6.

This specifies a log (base 10) transformation of the y axis.



ggplot – The basics

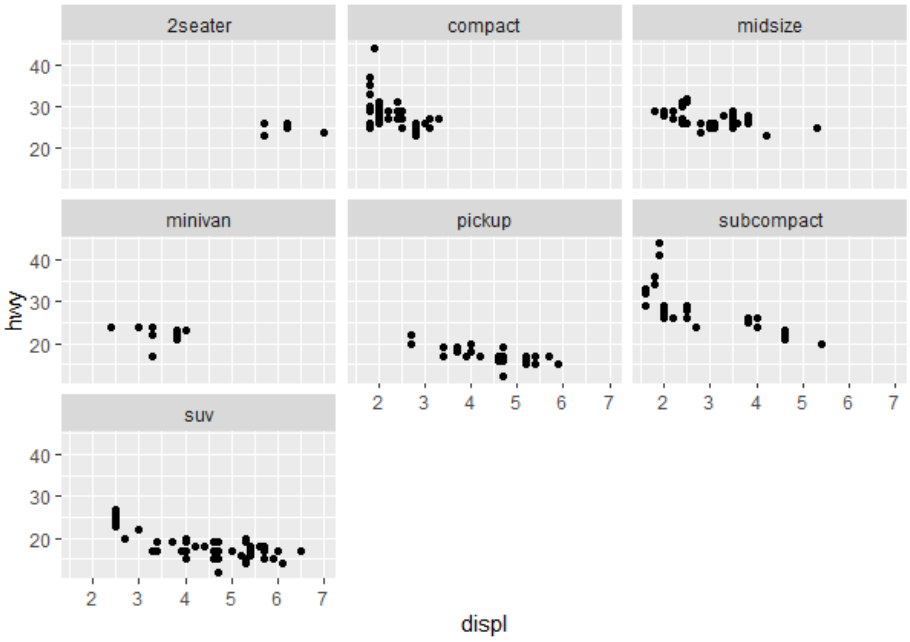
- We have already looked at geoms (different types of diagrams).
- Facets are layers that allow the representation of a categorical variable in several panels within one plot (instead of showing a different plot per group).
- This is particularly useful for experimental data which will have grouping variables (one or several for factorial experiments) as independent variables.
- Facets allow the combination of different diagrams for each group into one. Facets do not allow the combination of different geoms (types of diagrams) within the same plot.



ggplot – The basics

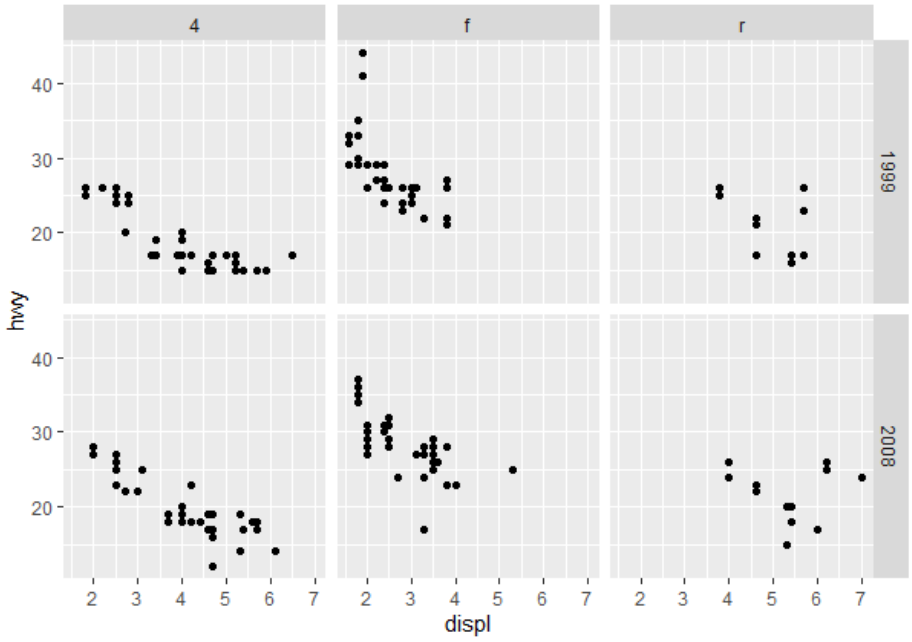
- We use the factor variable class to show a scatterplot for each group of displ and hwy.

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~ class)
```



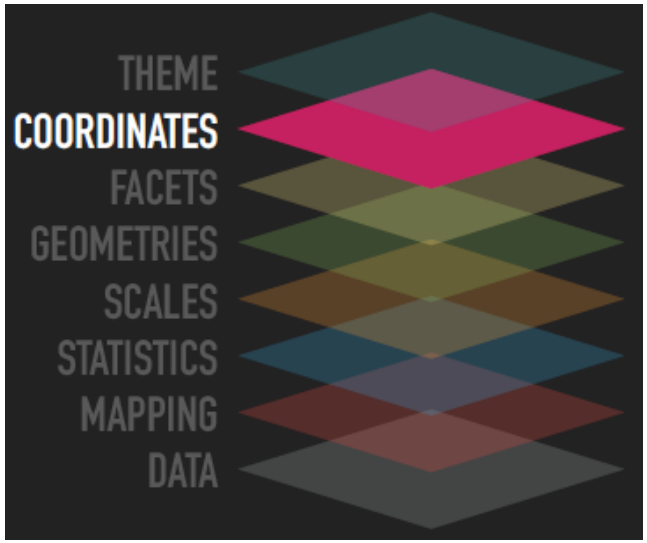
- You can use more than one facet variable.
- Here, we use facet_grid to facet by drv and year.

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_grid(year ~ drv)
```



ggplot – The basics

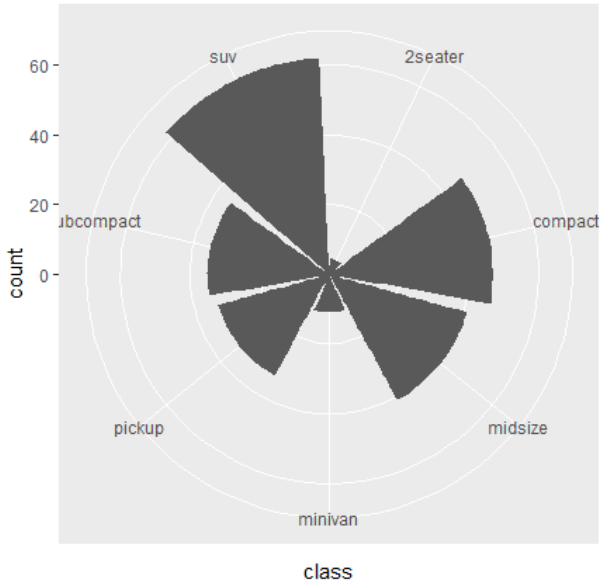
- Coordinates changes the appearance of the graph. For example, we can put a barplot into a circle, we can also flip axes.
- You can change value ranges of axes and do transformations of axes via the scales()argument or the coordinates() call.
  - Scales will result in a whole transformation of the plot early on.
  - Coordinates changes the appearance of the scale at the end of the call. It is often advisable to use coordinates() rather than the scale() argument.



ggplot – The basics

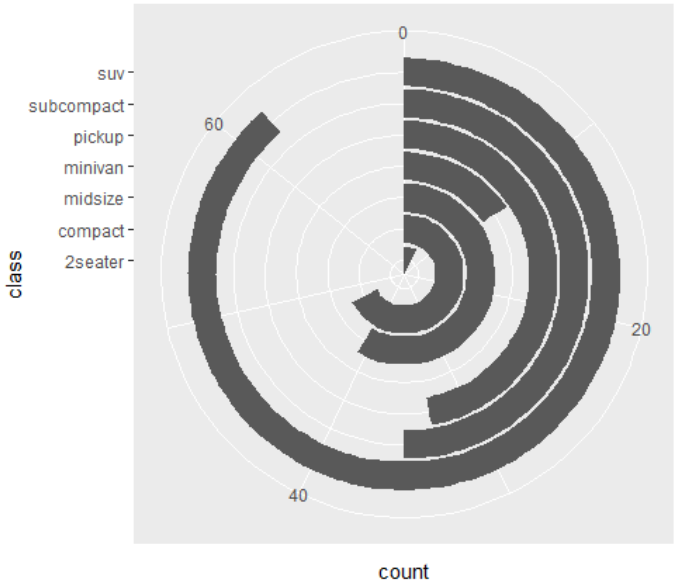
- This coord_polar() let‘s you put your barplot transformed into a 360° circle.

```
ggplot(mpg) +
  geom_bar(aes(x = class)) +
  coord_polar()
```



- Changing the angle will result in a lovely circle graph with bars becoming bendy.

```
ggplot(mpg) +
  geom_bar(aes(x = class)) +
  coord_polar(theta = 'y') +
  expand_limits(y = 70)
```

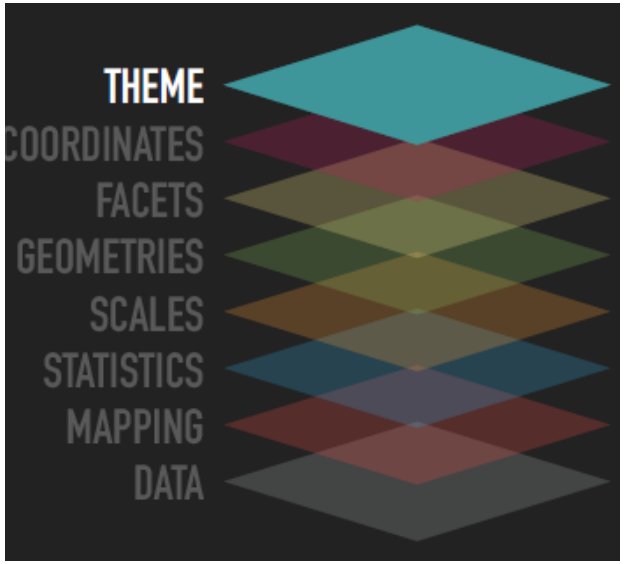


ggplot – The basics

- The theme() is the last part and directly regulates how the graph looks as a whole. It is a very easy way to manage how the whole thing will look – whether the coordinates system uses boxed lines, whether the grid is shown, the type of font and font size that is used for different elements.

- In general, it is advisable to use theme_minimal() or even theme_void(). This gets rid off all major and minor grid lines.
- You should abolish all graph clutter from your graph.
- Minimal is beautiful. What does spark joy in your graph? What is superfluous information (and thus distracts from the main message)?

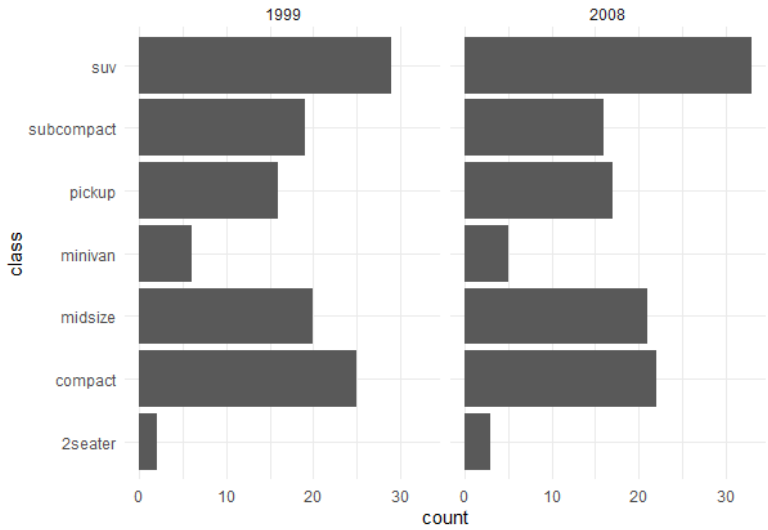
- Graphing your data is were creativity, design thinking and programming meet.



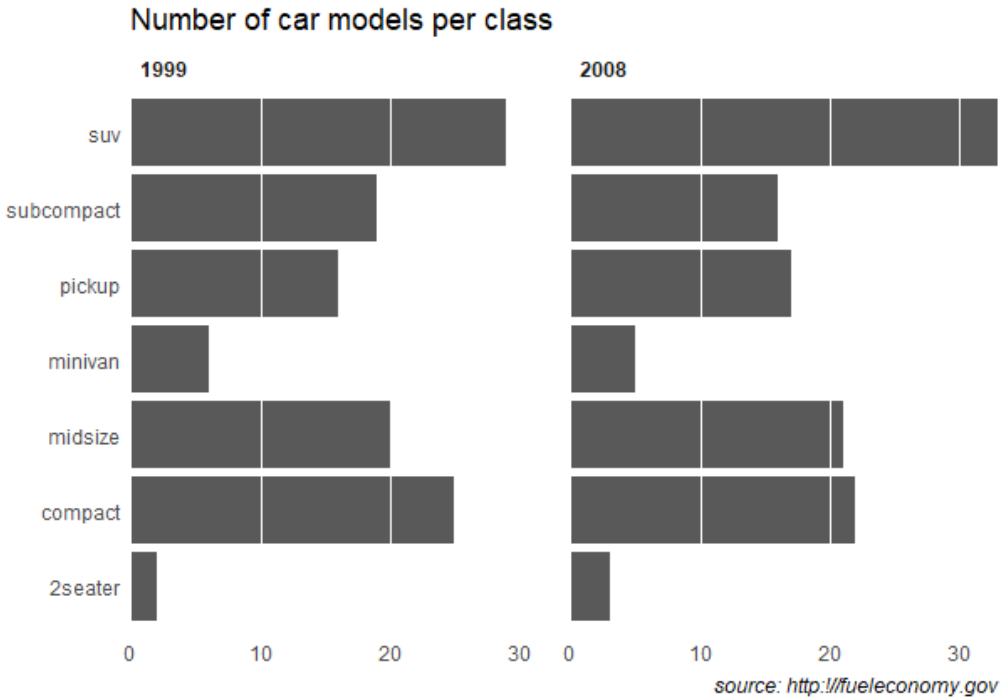
ggplot – The basics

- Minimal is beautiful…

```
ggplot(mpg) +
  geom_bar(aes(y = class)) +
  facet_wrap(~year) +
  theme_minimal()
```



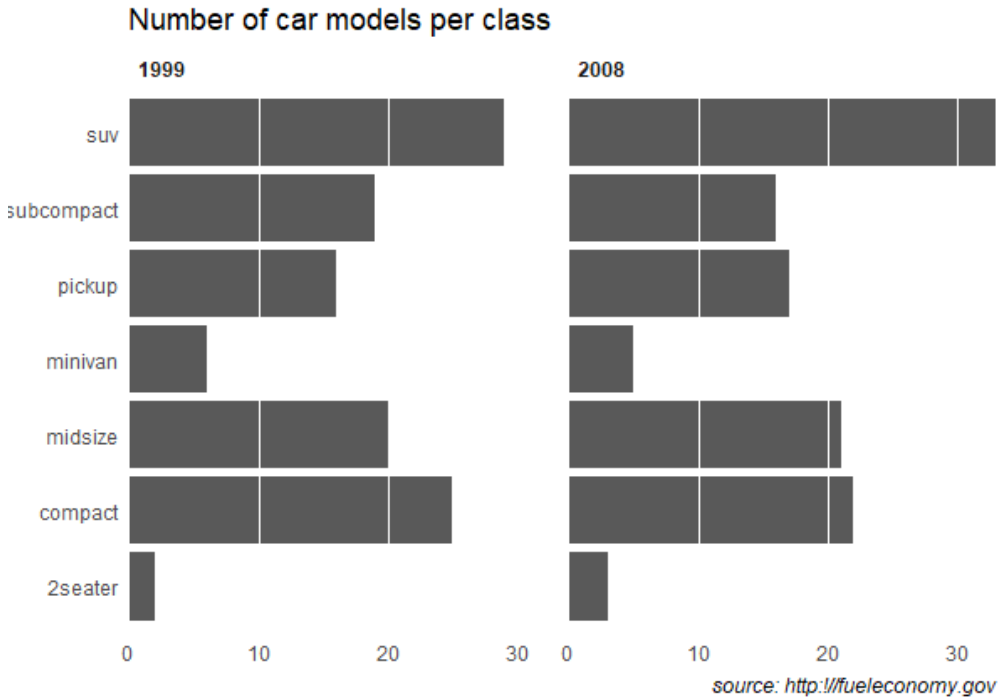
- Annotations are usually done via the labs() call.



ggplot – The basics

- Annotations are usually done via the labs() call.

```
ggplot(mpg) +
  geom_bar(aes(y = class)) +
  facet_wrap(~year) +
  labs(title = "Number of car models per class",
       caption = "source: http://fueleconomy.gov",
       x = NULL,
       y = NULL) +
  scale_x_continuous(expand = c(0, NA)) +
  theme_minimal() +
  theme(
    text = element_text('Avenir Next Condensed'),
    strip.text = element_text(face = 'bold',
                              hjust = 0),
    plot.caption = element_text(face = 'italic'),
    panel.grid.major = element_line('white',
                                    size = 0.5),
    panel.grid.minor = element_blank(),
    panel.grid.major.y = element_blank(),
    panel.ontop = TRUE  )
```



Intro to ggplot2

1. ggplot: The grammar of graphics
2. The basics – layers
3. **The most important geoms**
4. Faceting
5. Adding elements
6. Extensions to ggplot and literature

Some important geoms

- In ggplot2, graphs are represented as list elements.
- The base layer defines how features of data are translated into visual elements. The base syntax is:

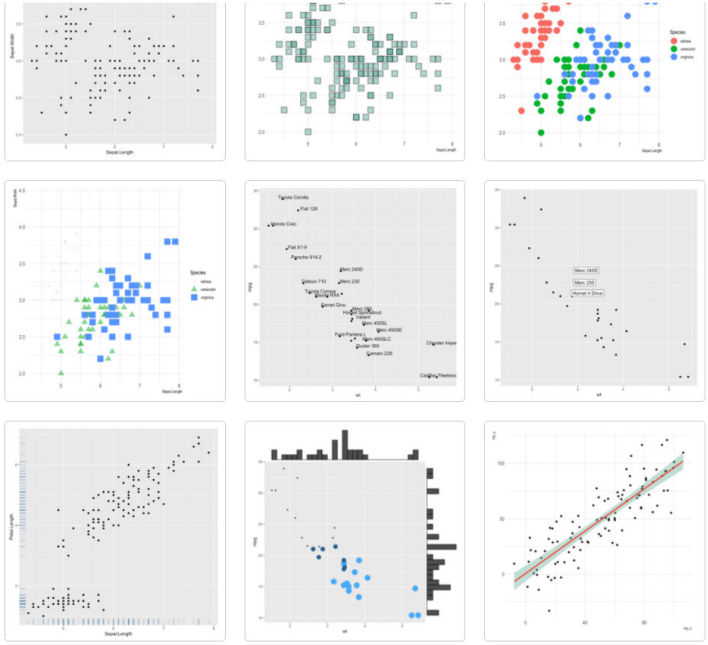
```
ggplot(data, aes(x = x axis, y = y axis,
                 colour = variable,
                 fill = variable,
                 shape = factor,
                 linetype = factor,
                 group = factor))
```

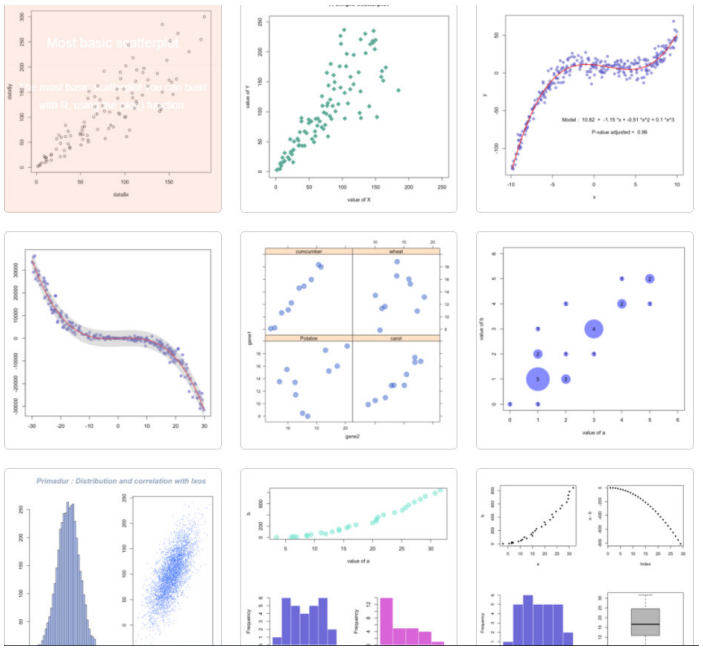
- Data: First argument, most often long format.
- Aesthetics: What is drawn on the x axis, what on the y axis?, additional elements control the assignment of further variables to visual attributes of the graph
- All variable names are used without " ".
- colour: Show data based on categorical variables (factor)
- fill: Used for area elements (e.g., column/bar charts) to control their colour.
- shape: Used to define the shape of the point in geom_point() scatterplots.
- linetype: The factor variable defines the linetype in geom_line().
- group: A given factor variable defines how layers (data from that group) is represented (most often used to later define scale_fill or scale_colour).

Base geoms

- Different types of graphs are represented by geoms in ggplot. Geoms specify how data defined by the aes()-call is translated into a representation on the graph.
- Geoms are added after the base layer is defined.
- All geom functions have distinct arguments controlling features specific to this type of diagram.
  - geom_point() for a scatterplot
  - geom_bar() for a bar chart
  - geom_histogram() for a histogram
  - geom_density() for a Kernel density diagram
  - geom_line() for a line plot
  - geom_boxplot() for a boxplot
- It is possible to combine base geoms. For instance, the base layer could be formed by a boxplot or a scatterplot and you can add further elements like a regression line to the scatterplot.
- You can use different data assignments within the geoms, even if you have defined the aes-mapping in the base ggplot(data, aes()) call.

Base geoms



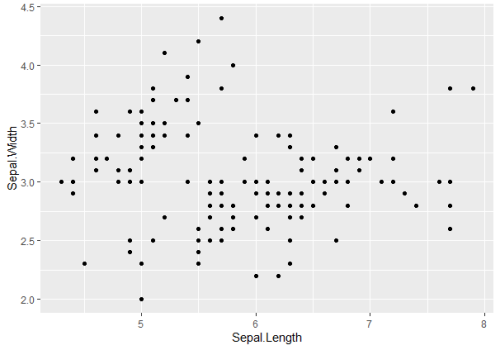


Correlation

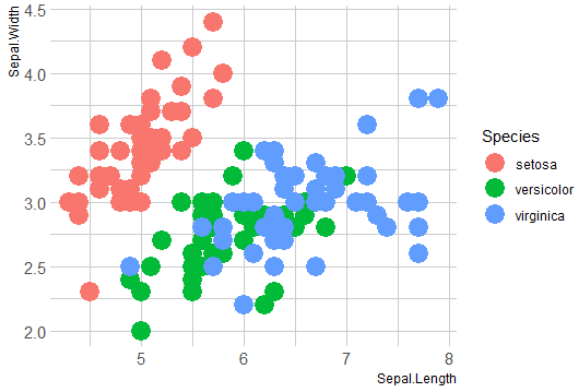
Base geoms

- In ggplot:

```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point()
```

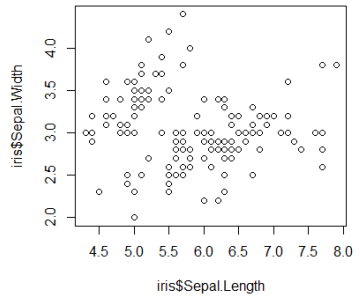



```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point(size=6) +
  theme_ipsum()
```



- In Base R:

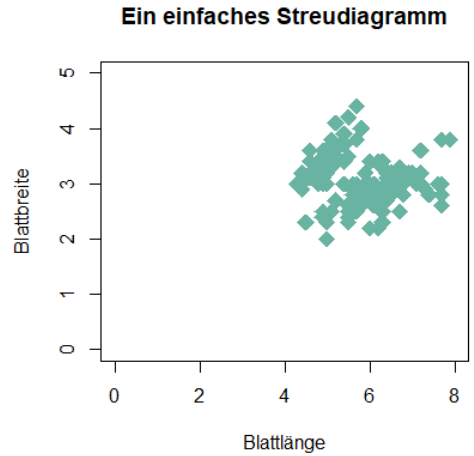
```
plot(iris$Sepal.Length, iris$Sepal.Width)
```



Elements:

- cex: Size of points
- xlim and ylim: limits of x and y axis
- pch: Type of marker
- xlab and ylab: axis labels
- col: colour of marker
- main: title of graph

```
plot(iris$Sepal.Length, iris$Sepal.Width,
     xlim=c(0,8.0) , ylim=c(0,5.0),
     pch=18,
     cex=2,
     col="#69b3a2",
     xlab="Blattlänge", ylab="Blattbreite",
     main="Ein einfaches Streudiagramm")
```



Base geoms

- This shows what is possible with the building block system of ggplot:
- We build a scatter plot of the height of a person and their mood.
- We assign a different colour by sex. The experimental group is represented by different shapes.
- geom_point(size = 3) specifies the size of dots.

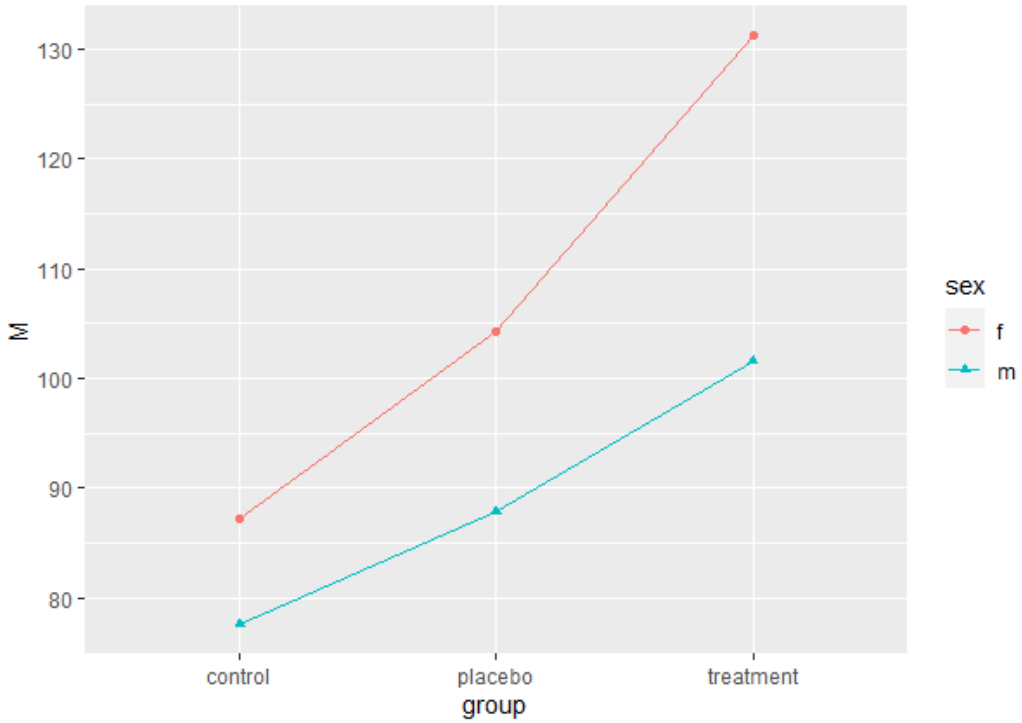
```
paa1<-ggplot(myDf, aes(x=height,
                       y=mood,
                       colour=sex,
                       shape=group))
paa1
paa2<-paa1+geom_point(size=3)
paa2
```

Base geoms

- geom_line() is used for drawing interaction plots for experimental data. To do so, we need to calculate the descriptive statistics to be drawn.
- The interaction plot in ggplot is a combination of geom_point() and geom_line() geoms.

```
groupM<-aggregate(mood ~ sex + group, data=myDf, FUN=mean)
groupM<-transform(groupM, M=mood, mood=NULL)
groupM

library(dplyr)
myDf %>%
  group_by(sex, group) %>%
  summarise(mean(mood))
```

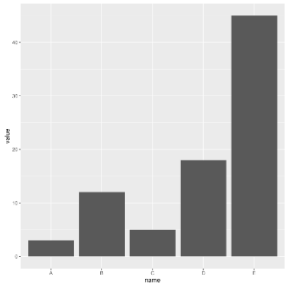
```
pab<-ggplot(groupM,
            aes(x=group, y=M,
                color=sex,
                shape=sex,
                group=sex))+
  geom_point()+
  geom_line()
pab
```



Basic geoms – Bar plot

Basic geoms – Bar plot

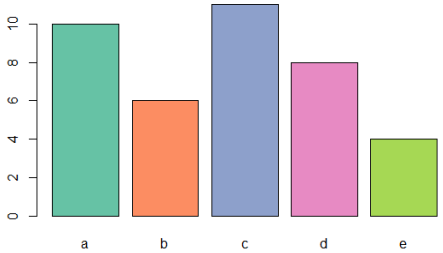
- In ggplot:



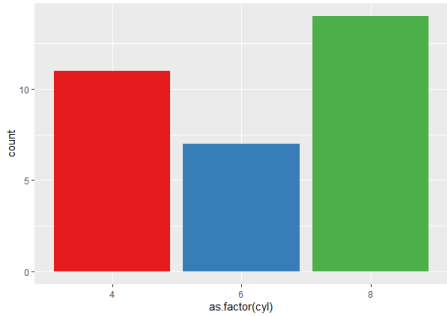
```
# Load ggplot2
library(ggplot2)

# Create data
data <- data.frame(
  name=c("A","B","C","D","E") ,
  value=c(3,12,5,18,45)
  )

# Barplot
ggplot(data, aes(x=name, y=value)) +
  geom_bar(stat = "identity")
```

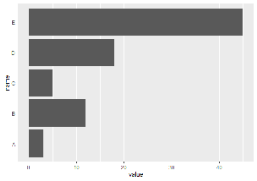
- In base R:
  - Count var in data OR
  - barplot(table(data))

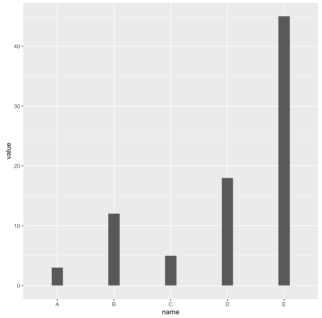
```
library(RColorBrewer)
coul <- brewer.pal(5, "Set2")
barplot(height=data$value, names=data$name, col=coul )
```




```
##Farben
#Einheitliche Farbe
ggplot(mtcars, aes(x=as.factor(cyl) )) +
  geom_bar(color="blue", fill=rgb(0.1,0.4,0.5,0.7) )
#Palette aus RColorBrewer
ggplot(mtcars, aes(x=as.factor(cyl), fill=as.factor(cyl) )) +
  geom_bar( ) +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.position="none")
#Manuell
ggplot(mtcars, aes(x=as.factor(cyl), fill=as.factor(cyl) )) +
  geom_bar( ) +
  scale_fill_manual(values = c("red", "green", "blue") ) +
  theme(legend.position="none")
##Horizontale Anordnung
data <- data.frame(
  name=c("A","B","C","D","E") ,
  value=c(3,12,5,18,45)
)
ggplot(data, aes(x=name, y=value)) +
  geom_bar(stat = "identity") +
  coord_flip()
##Breite der Balken mit width-Argument kontrollieren
ggplot(data, aes(x=name, y=value)) +
  geom_bar(stat = "identity", width=0.2)
```







Bar plots

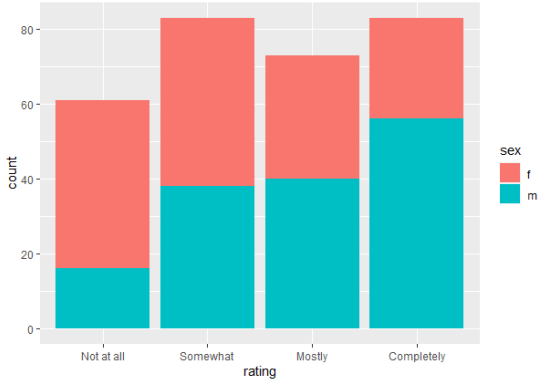
- For bar plots, in the aes(x = variable, y = variable) call you only need to define the x variable. The x variable regulates the length of the bars (usually representing a count or percentage frequency).
- The x variable should be a factor/categorical variable whose frequencies you can determine via geom_bar(stat = „count“).
- Relative frequencies can be given directly if they are in the data as the y (in this case put geom_bar(stat = “identity“) or put the calculation into the geom_bar() call.
- geom_bar(stat=„count“, aes(y =(..count..)/sum(..count..)))
- or use y = after_stat.

- geom_bar(position = postion_dodge()) results in a grouped bar chart
- position_stack() produces a stacked bar chart

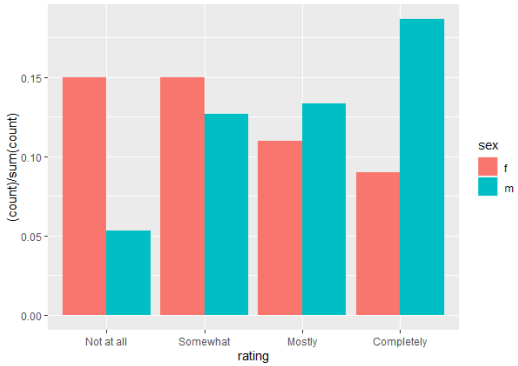
```
ggplot(mpg) +
  geom_bar(
    aes(
      x = class,
      y = after_stat(100 * count / sum(count))
    )
  )
```

Bar plots

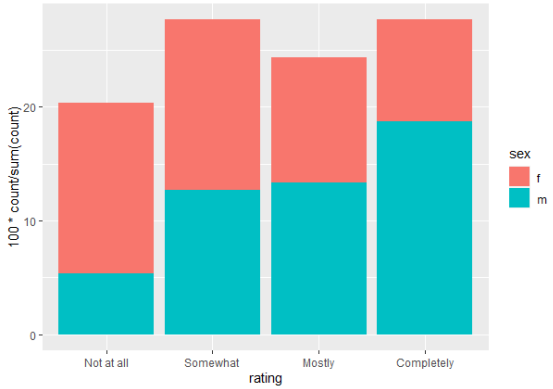
```
#Säulendiagramm
#Grunddiagramm
pba1<-ggplot(myDf, aes(x=rating, group=sex, fill=sex))
pba1
#Da Gruppe = Geschlecht, wird hier ein stacked bar chart ausgegeben.
(pba2<-pba1+geom_bar(stat="count", position=position_stack()))
```




```
#Relative Häufigkeiten + gruppiertes Balkendiagramm
(pbb<-pba1 + geom_bar(stat="count",
                      aes(y=(..count..) / sum(..count..)),
                      position=position_dodge()))
```



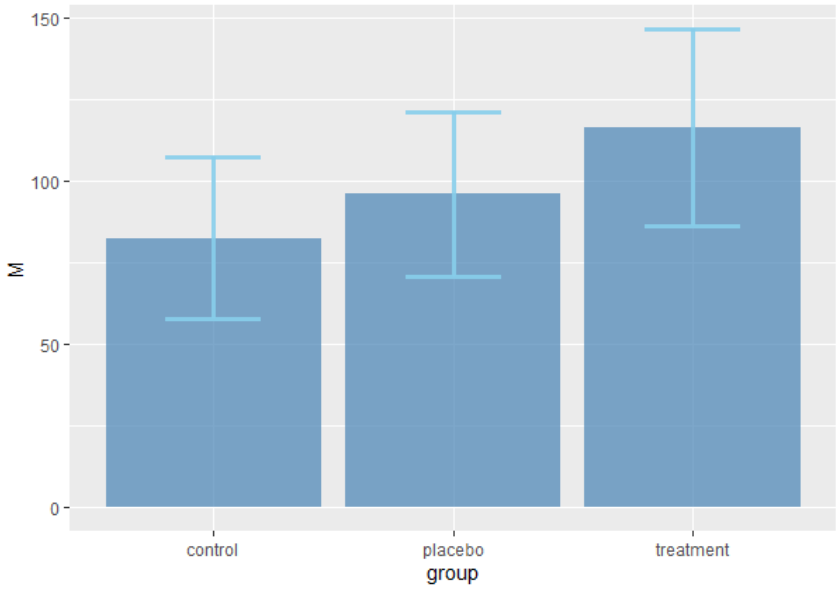

```
#Hier ist der Weg über after_stat()
(pbc<-ggplot(myDf, aes(x=rating,
                       y=after_stat(100 * count/sum(count)),
                       group=sex, fill=sex))+
    geom_bar(stat="count"))
```



Bar plot: Special case #1 – Means as bars with error bars

- You often see a type of diagram in which the means of groups are presented as bars with error bars.
- Using these is actively discouraged by statisticians.
- If you must use them, you can add error bars via geom_errorbar(). Please never forget to accurately label in a title or legend what the error bar is (is it the SD, SE, 95% CI, range?)

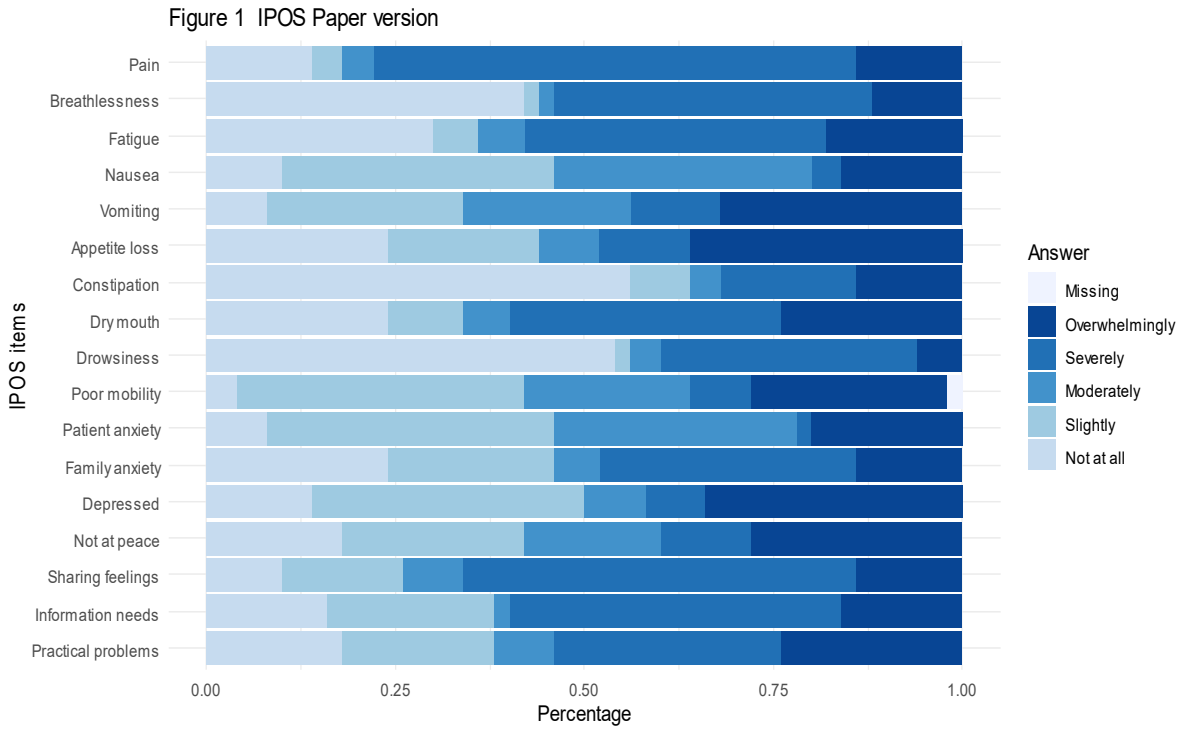
```
groupN<-aggregate(mood ~ group, data=myDf, FUN="mean")
sdN<-stats::aggregate(mood ~ group, data=myDf, FUN="sd")
groupN<-transform(groupN, M=mood, mood=NULL)
groupN$SD<-sdN$mood
groupN
ggplot(groupN)+
  geom_bar(aes(x=group, y=M),
           stat="identity",
           fill="steelblue", alpha=0.7)+
  geom_errorbar(aes(x=group, ymin=M-SD, ymax=M+SD),
                width=0.4, colour="skyblue", alpha=0.9, size=1.3)
```



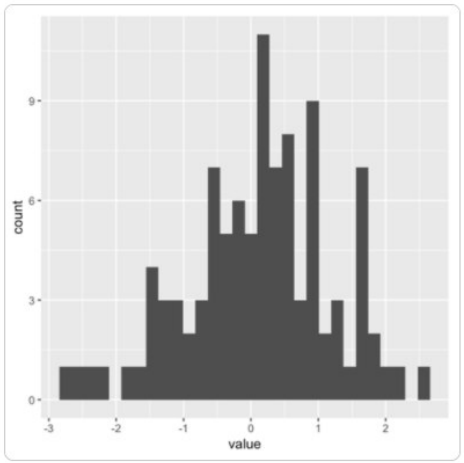
Bar plot: Special case #2 – Stacked bar chart for symptom burden

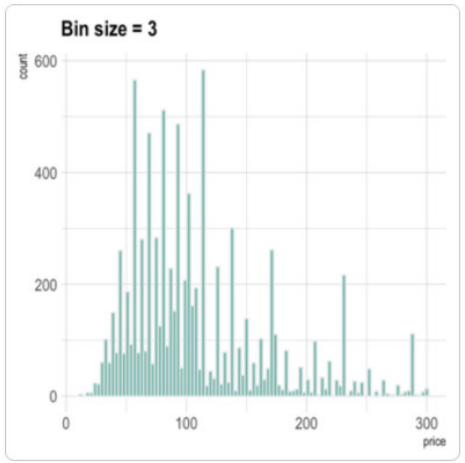
- When you work with questionnaire data (Likert type scaling), it can be a good idea to show the percentage of answer categories per symptom (all adding up to 100% per symptom).
- You could give the scaling in percent via:
- scale_y_continuous(labels = scales::percent_format(accuracy = 5L))+

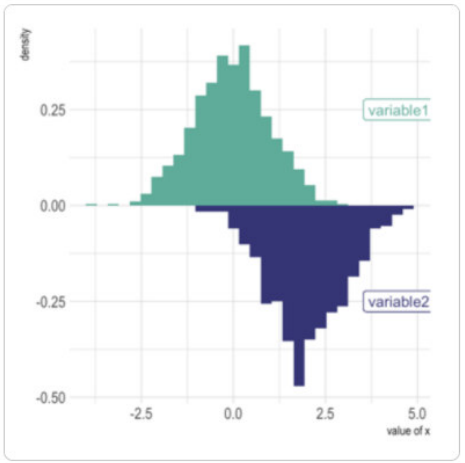
```
library(ggplot2)
ggplot(stacked, aes(fill=answer, y=paper_score, x=item)) +
  geom_bar(position="fill", stat="identity") + coord_flip() +
  scale_fill_manual(values=c("#EFF3FF", "#084594", "#2171B5",
                             "#4292CB", "#9ECAE1", "#C6DBEF")) +
  labs(title = "Figure 1  IPOS Paper version",
       y = "Percentage", x = "IPOS items", fill = "Answer")+
  theme_minimal()
```

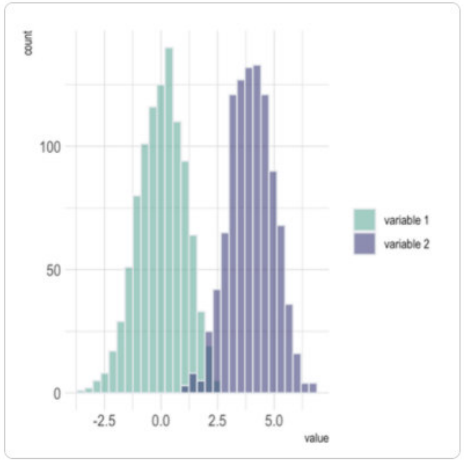


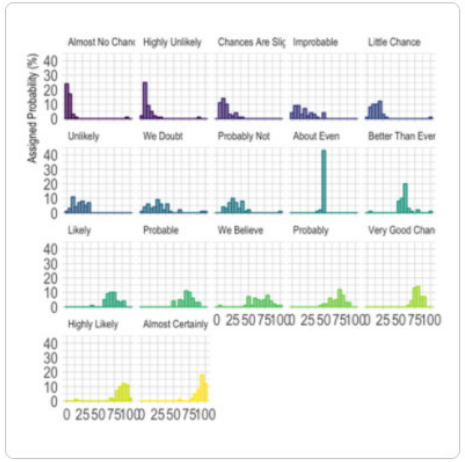
Basic geoms - Histogram

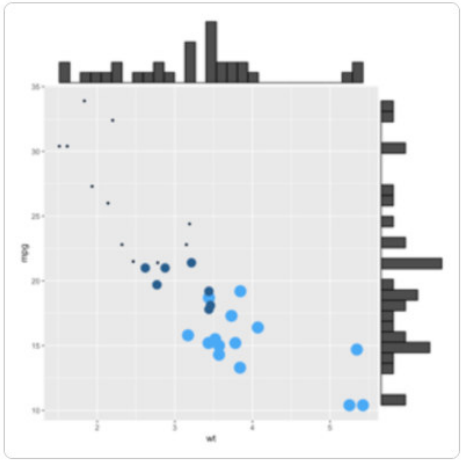








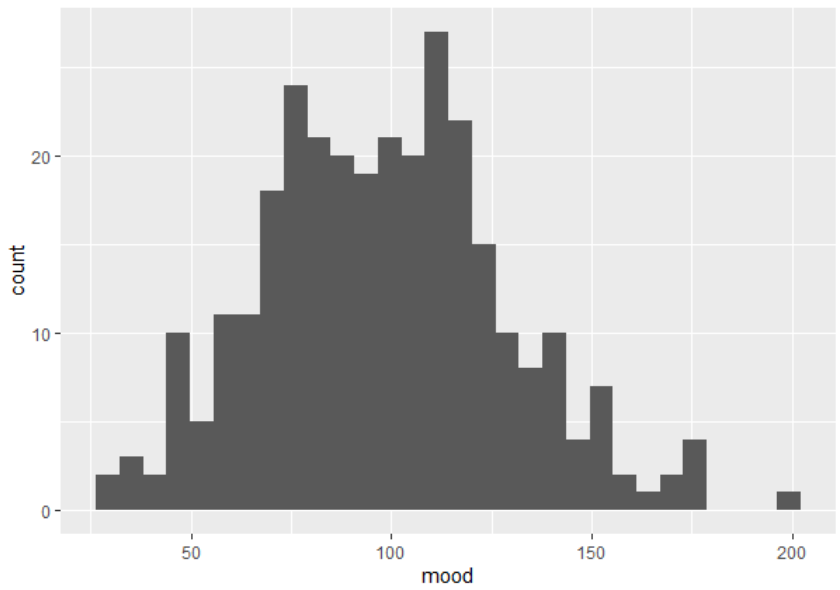




Basic geoms - Histogram

- Use geom_histogram() for histograms. You only need to specify the x variable in the aes() call.
- You can add the predicted density instead of the histogram by adding (y=..density..). You can control the number of bins via binwidth=area of the binning.

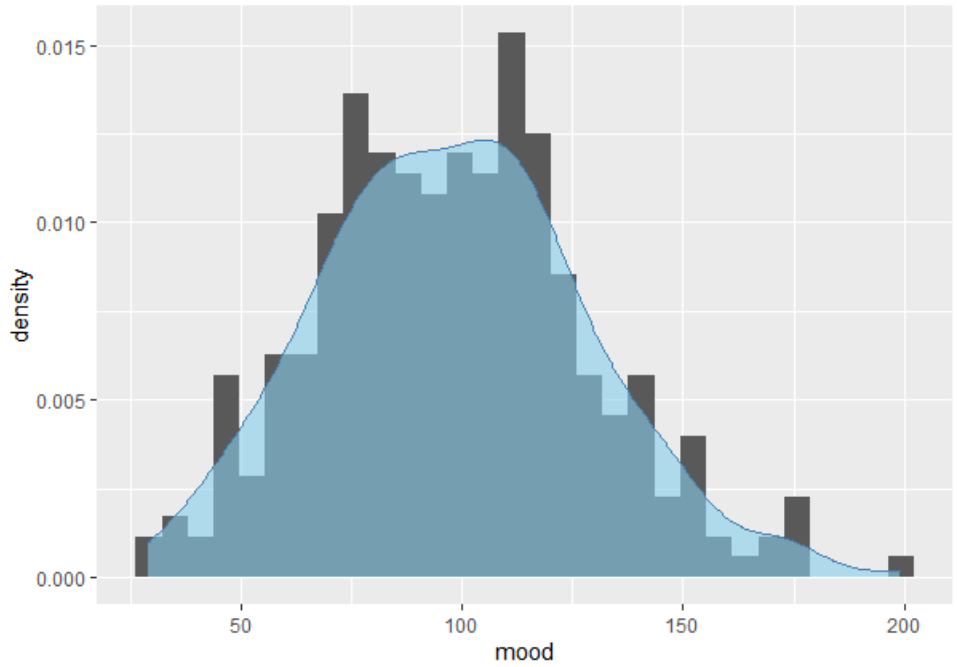
```
pCa1<-ggplot(myDf, aes(x=mood))+
  geom_histogram()
pCa1
```



Basic geoms - Histogram

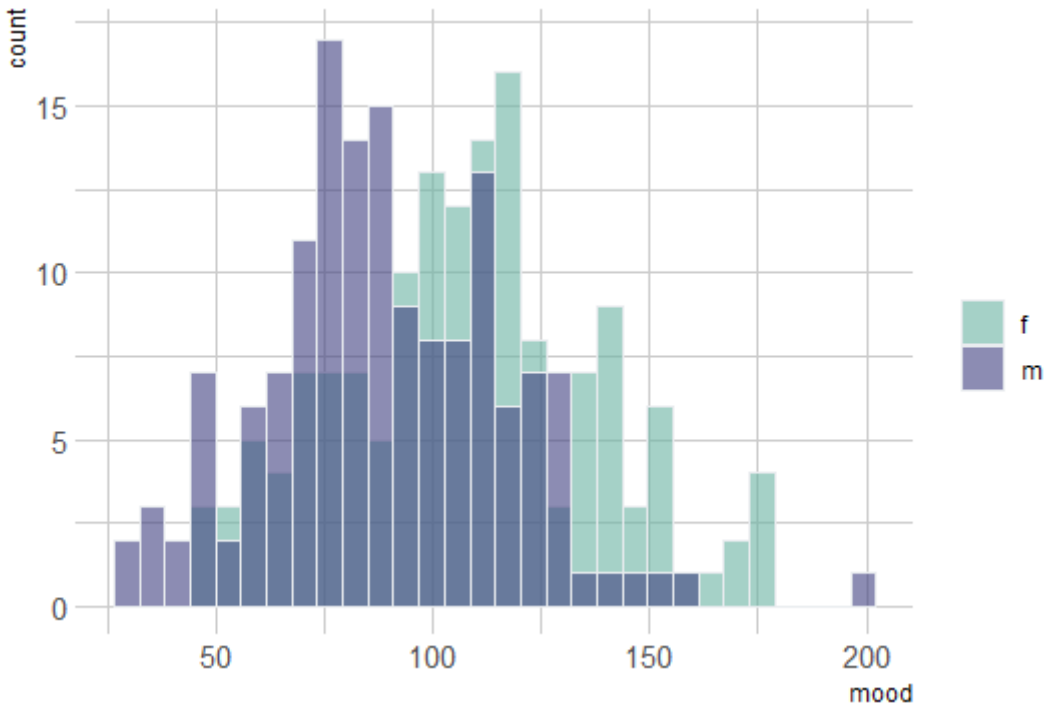
- It‘s often a very good idea to add a Kernel smoother to a histogram.

```
pD<-ggplot(myDf, aes(x=mood))+
  geom_histogram(aes(y=..density..))+
  geom_density(color="steelblue", fill="skyblue", alpha=0.6)
pD
```

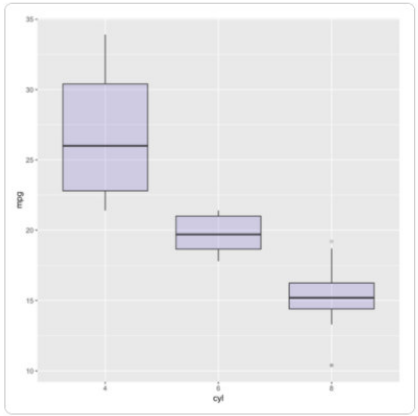


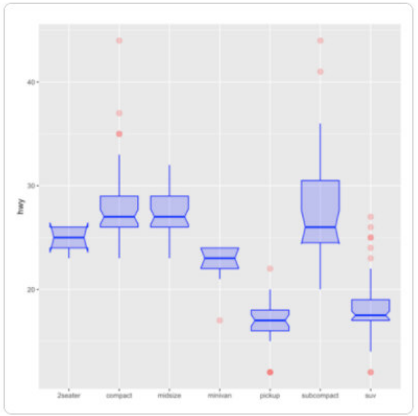
Basic geoms - Histogram

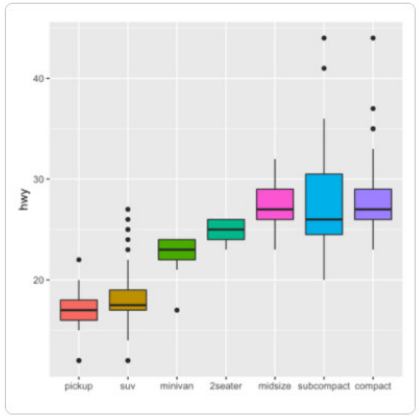
```
library(ggplot2)
library(dplyr)
library(hrbrthemes)
D2 <- myDf %>%
  ggplot( aes(x=mood, fill=sex)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity') +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
  theme_ipsum() +
  labs(fill="")
D2
```

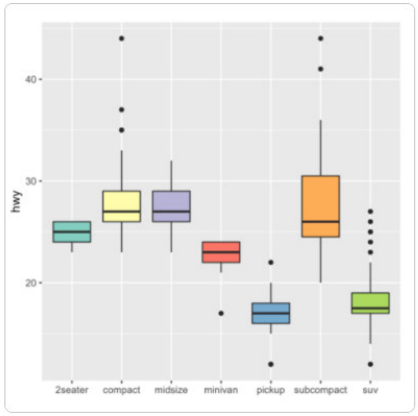


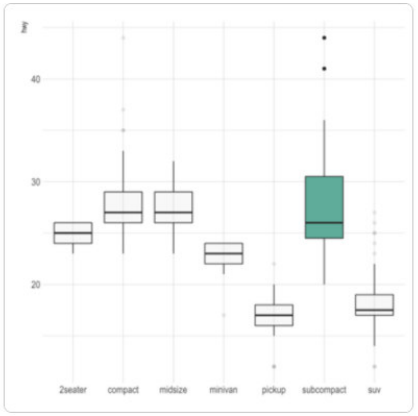
Basic geoms - Boxplots

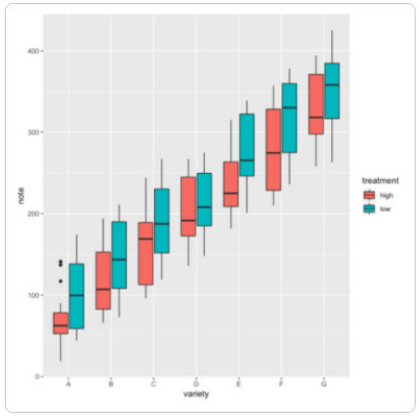








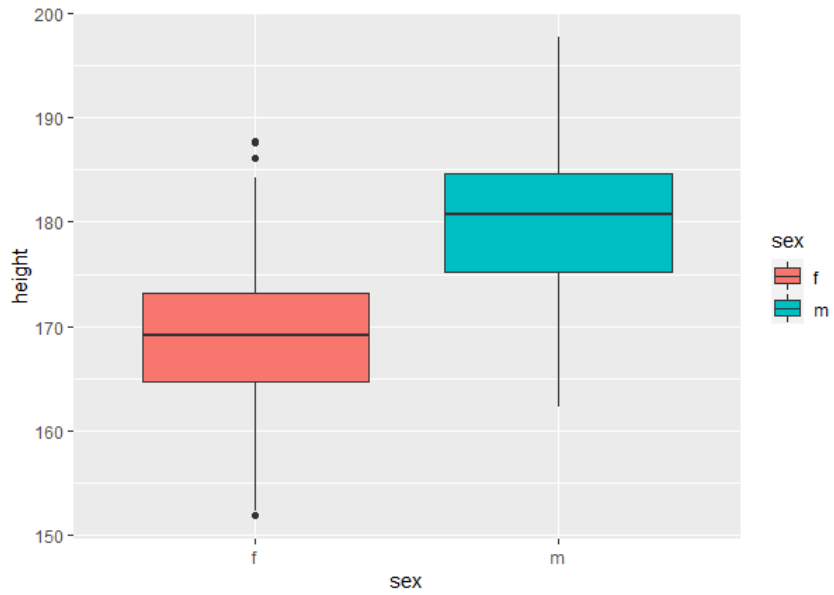




Basic geoms - Boxplots

- geom_boxplot() creates a boxplot. You need to define ggplot(data, aes(x = factor, y = continuous var, …)).
- A boxplot shows the continuous data of the dependent variable (y axis) by group. The grouping variable (independent variable) should be shown on the x axis.
- It is customary to put the IV on the x and the DV on the y.

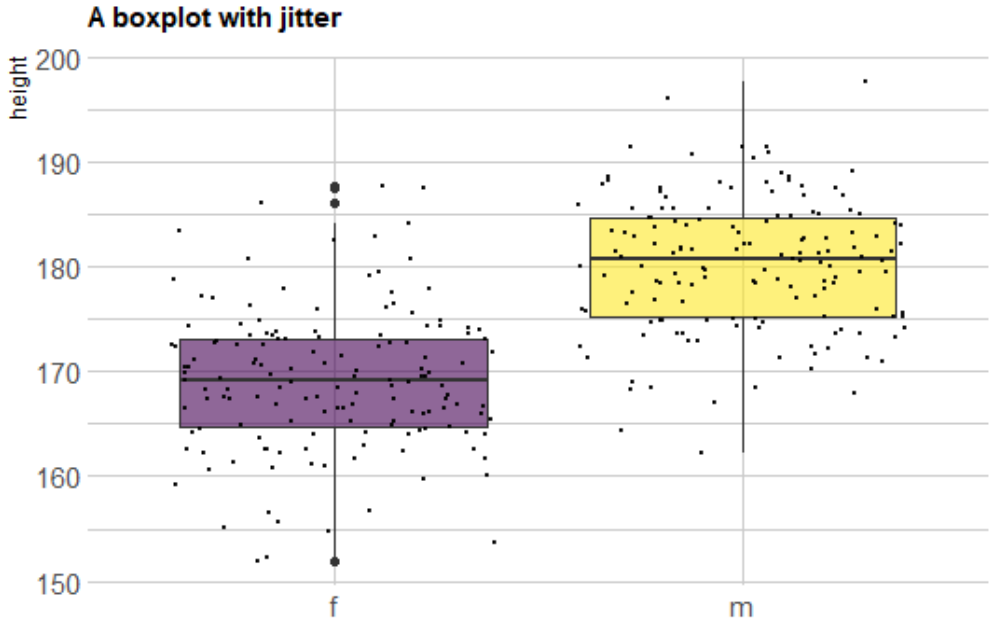
```
pBox<-ggplot(myDf, aes(x=sex, y=height, fill=sex))+
  geom_boxplot()
pBox
```



Boxplots

- There are special types of boxplots which can be useful:
  - Show raw/individual data points on boxplot with geom_jitter(), alpha controls how transparent the dots are:

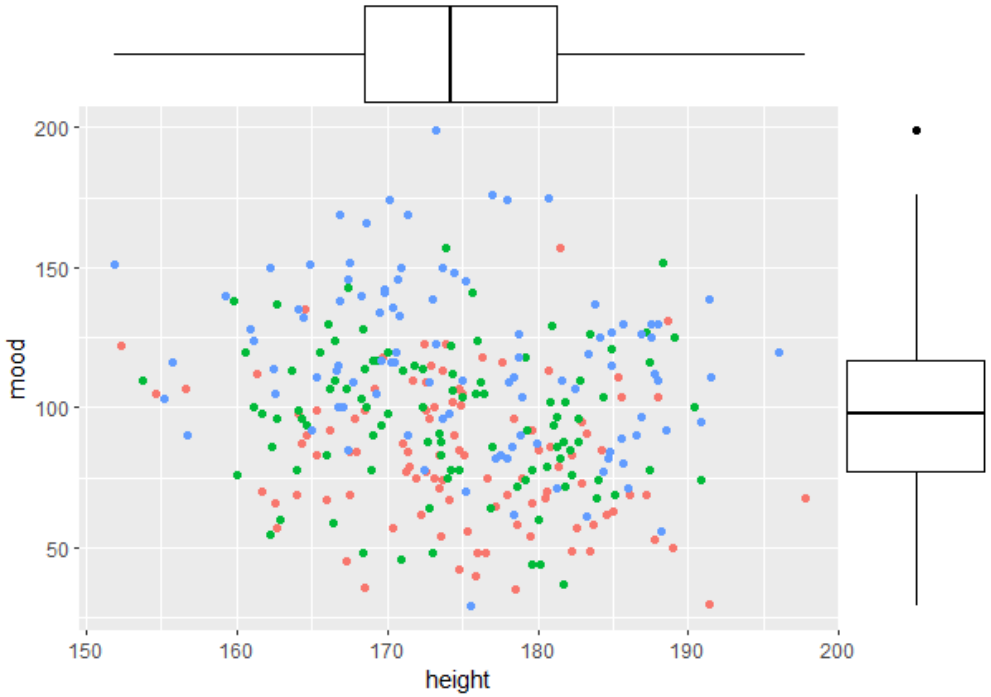
```
library(viridis)
myDf %>%
  ggplot( aes(x=sex, y=height, fill=sex)) +
  geom_boxplot() +
  scale_fill_viridis(discrete = TRUE, alpha=0.6) +
  geom_jitter(color="black", size=0.4, alpha=0.9) +
  theme_ipsum() +
  theme(
    legend.position="none",
    plot.title = element_text(size=11)
  ) +
  ggtitle("A boxplot with jitter") +
  xlab("")
```



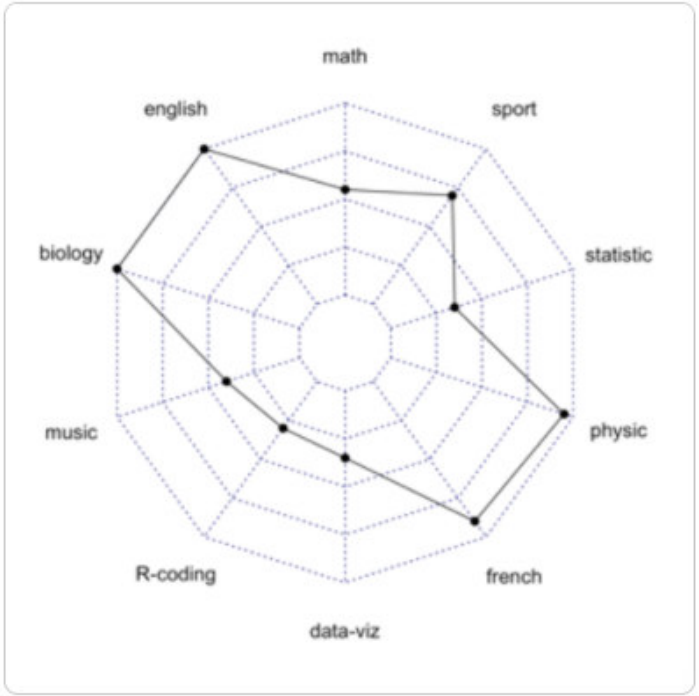
Boxplots

- A special type of boxplot is a marginal boxplot added to a chart.
  - Marginal boxplots are great for showing marginal distribution of variables. This lets you diagnose skewed distributions easily.

```
install.packages("ggExtra")
library(ggExtra)
margbp <- ggplot(myDf, aes(x=height, y=mood, color=group)) +
  geom_point() +
  theme(legend.position="none")
margbpd <- ggMarginal(margbp, type="boxplot")
margbpd
```



The most important geoms – Spider plots/Radar charts



Intro to ggplot2

1. ggplot: The grammar of graphics
2. The basics – layers
3. The most important geoms
4. **Faceting**
5. Adding elements
6. Extensions to ggplot and literature

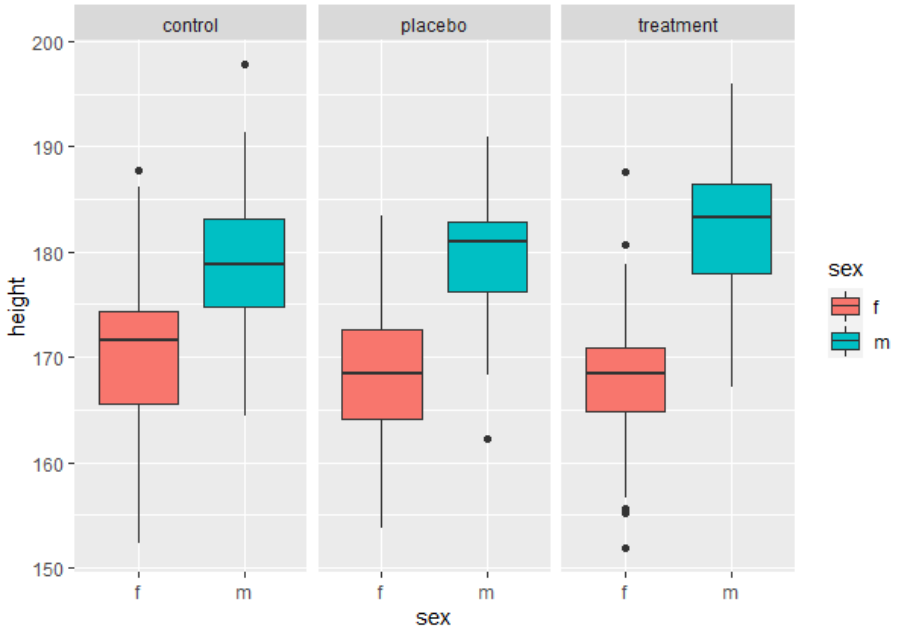
Types of diagrams – Conditional facets

- Multivariable data with categorical variables can be shown in facet graphs.
- The whole graph area will be facetted into panels. Each panel shows data for one category. You must use the same type of data diagram (geom) in each panel.
  - facet_grid(rows ~ cols) controls the different layers.
  - facet_wrap(~ factor, nrow=number, ncol=number) creates a facet for each group of the factor. You can combine several factors.
  - The arguments scales="free_y", scales="free_x" allow you to dynamically adapt the scales to the range in data for each axis. Can only be used in facet_grid() call.
  - The difference between facet_grid and facet_wrap is subtle. Most often, you will need facet_grid. I often use facet_wrap with geom_tile for heatmaps on which I want to show different groups.
  - Properly formatting the grid/wrap areas is a pain iunder facet_wrap.

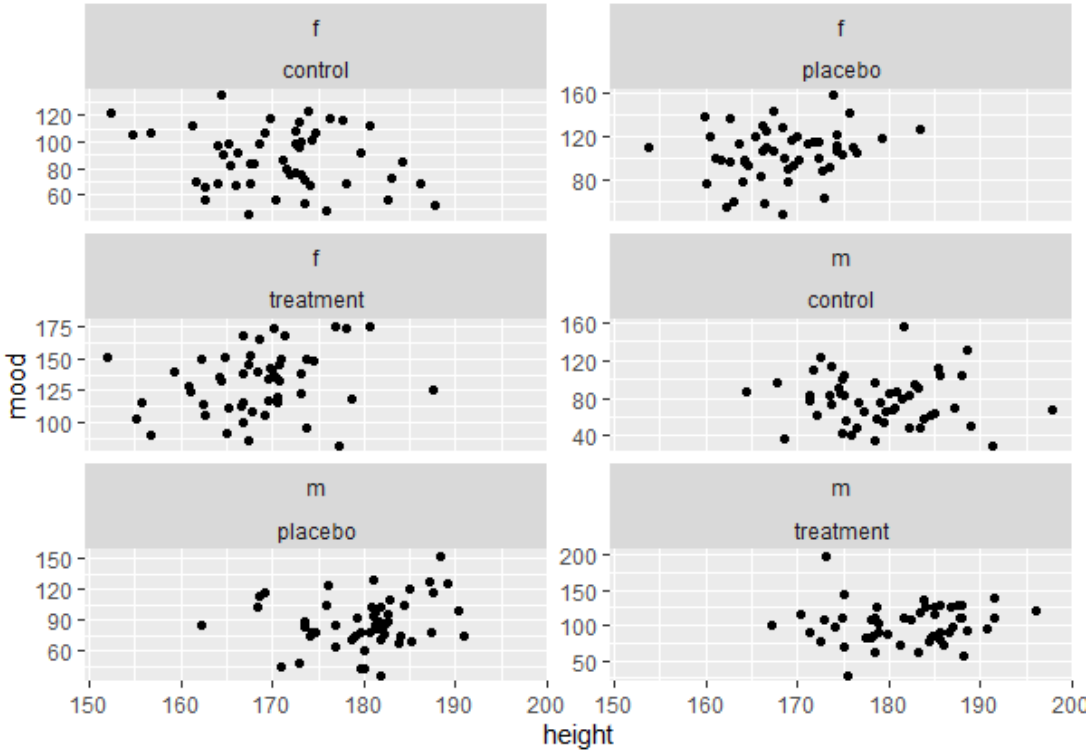
```
pEa<-ggplot(myDf, aes(x=sex, y=height, fill=sex))+
  geom_boxplot()+
  facet_grid(. ~ group)
pEa
```

Types of diagrams – Conditional facets

Example with facet_grid()



Example with facet_wrap()



Intro to ggplot2

1. ggplot: The grammar of graphics
2. The basics – layers
3. The most important geoms
4. Faceting
5. **Adding elements**
6. Extensions to ggplot and literature

Adding elements

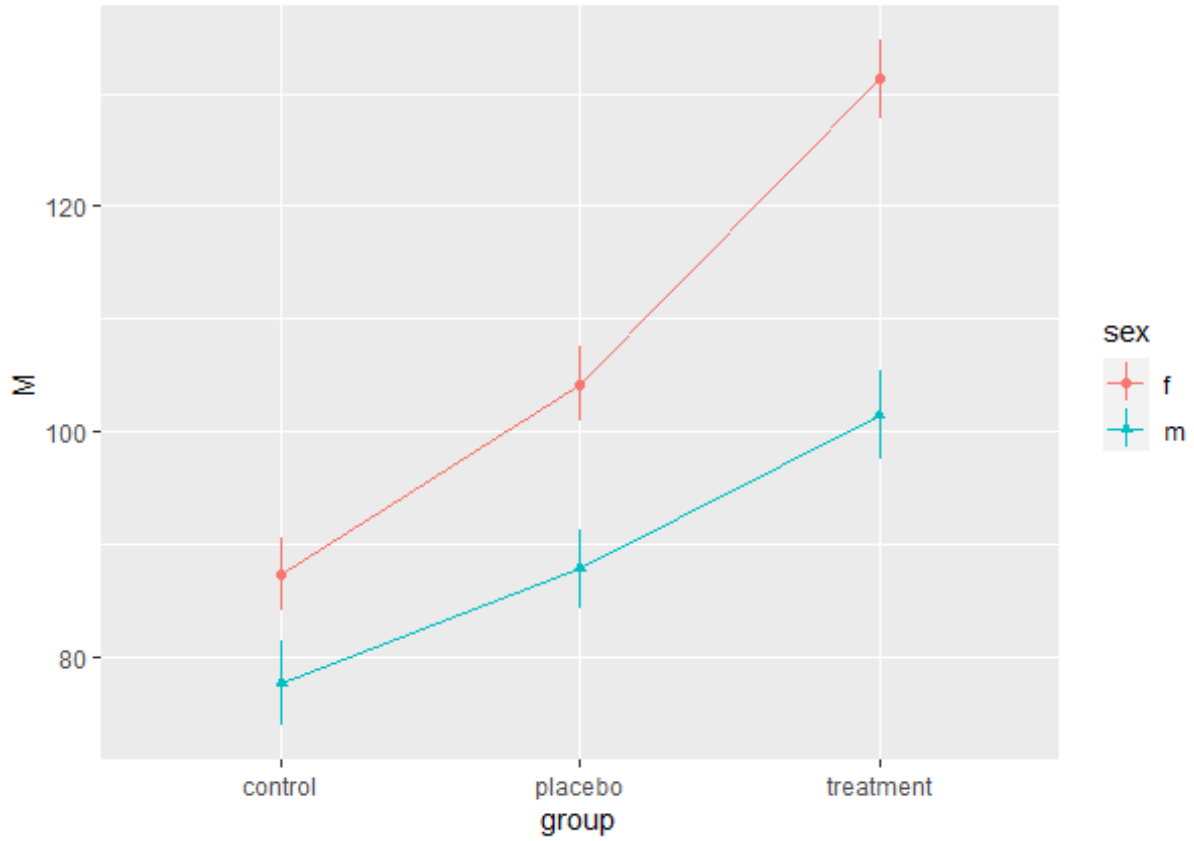
- Further geoms:
  - geom_segment(x=vector, y=vector, xend= vector, yend= vector) for line segments beginning at coordinates x,y and ending at xend, yend. Argument arrow draws arrows.
  - geom_polygon() for polygons, e.g. outlines on maps.
  - geom_rect(xmin=vector, xmax=vector, ymin=vector, ymax=vector) for rectangles, left lower corner: xmin, ymin coordinates, right upper corner: xmax, ymax coordinates.
- Further elements:
- labs(title="text", subtitle="text") for diagram title and subtitle.
- geom_vline(aes(xintercept=x axis)) for vertical lines which cross the x axis.
- geom_hline(aes(yintercept=y axis)) for horizontal lines that cross the y axis at yintercept.
- geom_ribbon() for an area defined by aes() y coordinates ymin = lower threshold and ymax = upper threshold. Possible usage: continuous confidence area around a prediction line.
- geom_linerange() for vertical line segment. Define in aes() via ymin = vector and ymax = vector. Possible usage: pointwise confidence intervals on forest plots.
- geom_errorbar() for classic error bars. Use aes() arguments ymin = vector and ymax = vector to define error bars ends. Difference to geom_linerange(): geom_errorbar() draws little perpendicular lines at end of min and max interval.
- geom_rug() for adding little vertical lines representing individual values above x axis or y axis. Often used in combination with a Kernel smoother. Often used in histograms or scatterplots.

Adding elements

- geom_smooth(method="lm","loess", se=TRUE,level=0.95,fullrange= FALSE) for adding a regression line to a graph. You can specify the method (linear model = "lm" or local Kernel smoother = "loess" via method). Using the argument se = TRUE produces a 95% CI-shaded area. fullrange = TRUE extends the regression line to the range of the x axis scale (might not be desirable because of extrapolating to values that are not contained within the data).
- geom_text(aes(x=x axis, y=y axis, label=variable)) for text shown in each facet of the diagram at coordinates x and y. The label is the variable containing the text to be shown.
- annotate("text", x=x axis, y=y axis, label= "annotation")) for text annotations which should be shown in each facet of the diagram at x,y coordinates. The text to be shown should be given via the label argument.

Adding error bars (here: SE)

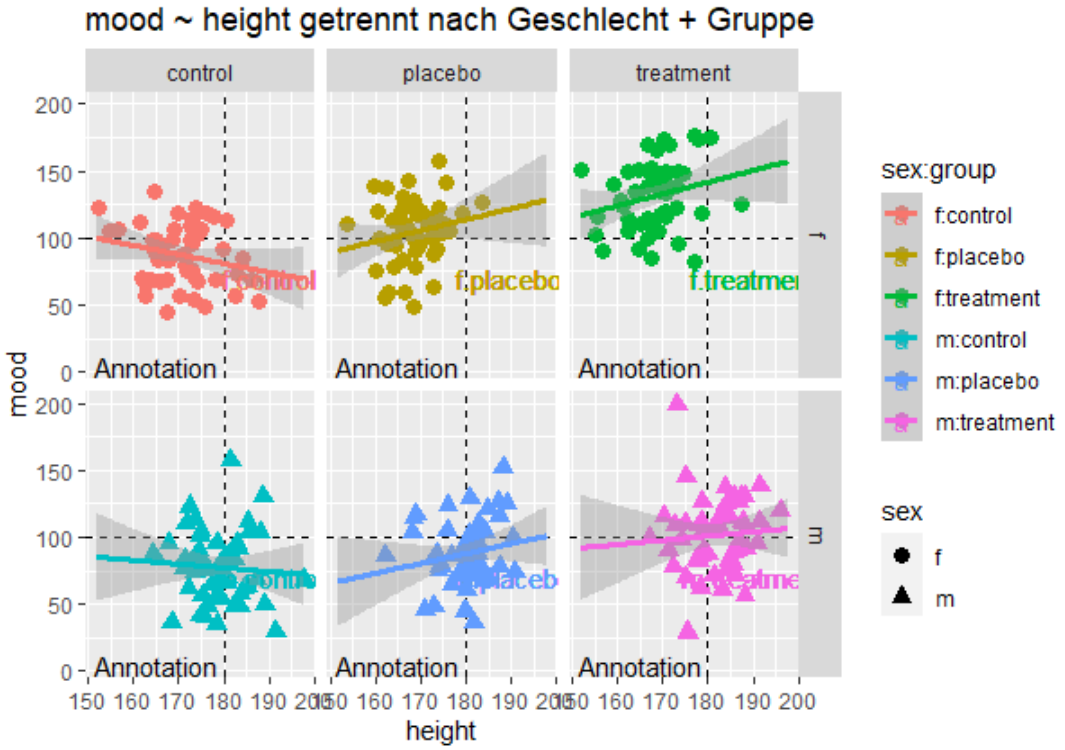
```
pFa <- ggplot(groupMSDN, aes(x=group, y=M, ymin=SEMlo, ymax=SEMup,
                             color=sex, shape=sex, group=sex)) +
      geom_point() +
      geom_line() +
      geom_linerange()
```



Changing graph elements

- Graph shows scatter plots for each experimental group with regression line + 95% CI for men and women

```
#Multipanelgrafik
pFb <- ggplot(myDf, aes(x=height, y=mood,
                        colour=sex:group, shape=sex)) +
    geom_hline(aes(yintercept=100), linetype=2) +
    geom_vline(aes(xintercept=180), linetype=2) +
    geom_point(size=3) +
    geom_smooth(method=lm, se=TRUE, size=1.2, fullrange=TRUE) +
    facet_grid(sex ~ group) +
    labs(title="mood ~ height getrennt nach Geschlecht + Gruppe") +
    geom_text(aes(x=190, y=70, label=sgComb)) +
    annotate("text", x=165, y=5, label="Annotation")
pFb
```



Formatting graphs

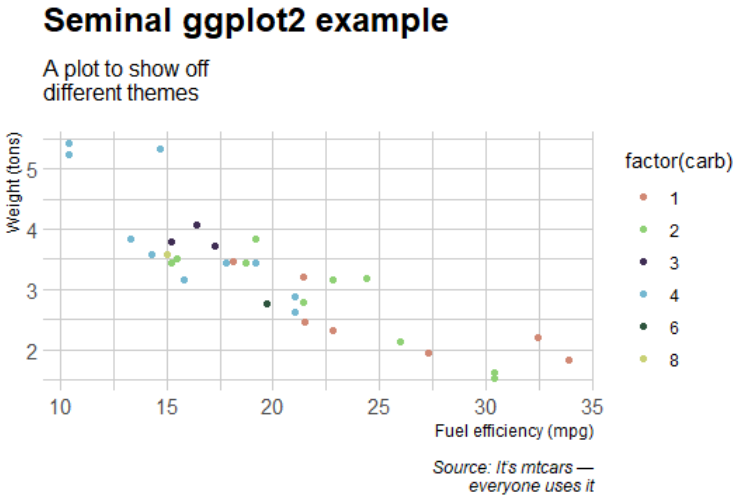
- Useful functions:
  - Control position of elements: position_stack(), position_fill(), position_dodge(), position_jitter()
  - Control axes: labs, theme, scale_x_continuous(), limits, coord_cartesian (xlim, ylim), coord_fixed, coord_flip
  - Change legends: theme(legend.position = „none“), guides
  - Change colours, point symbols, line types: colour, linetype = “solid“, “dashed“, “dotted“, stroke
    - scale_colour_hue, scale_colour_grey, scale_colour_brewer
- Change appearance globally via themes (you can define your own standard based on a theme):
  - theme_grey()
  - theme_bw()
  - theme_minimal()
  - theme_classic()

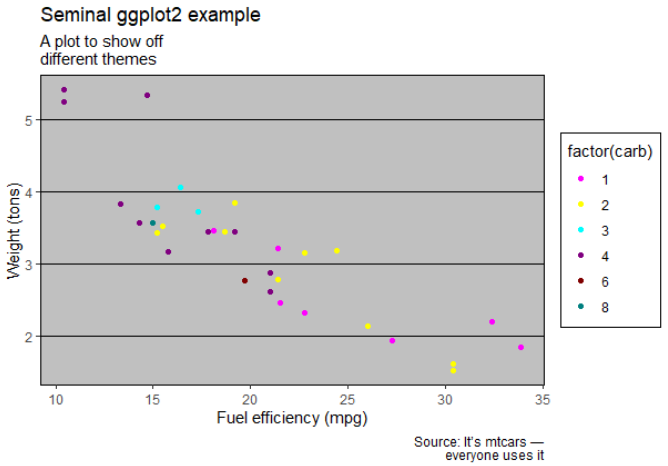
Intro to ggplot2

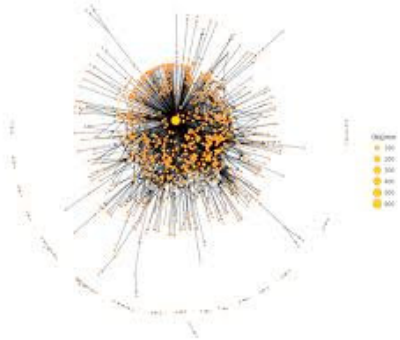
1. ggplot: The grammar of graphics
2. The basics – layers
3. The most important geoms
4. Faceting
5. Adding elements
6. **Extensions to ggplot and literature**

Extensions to ggplot

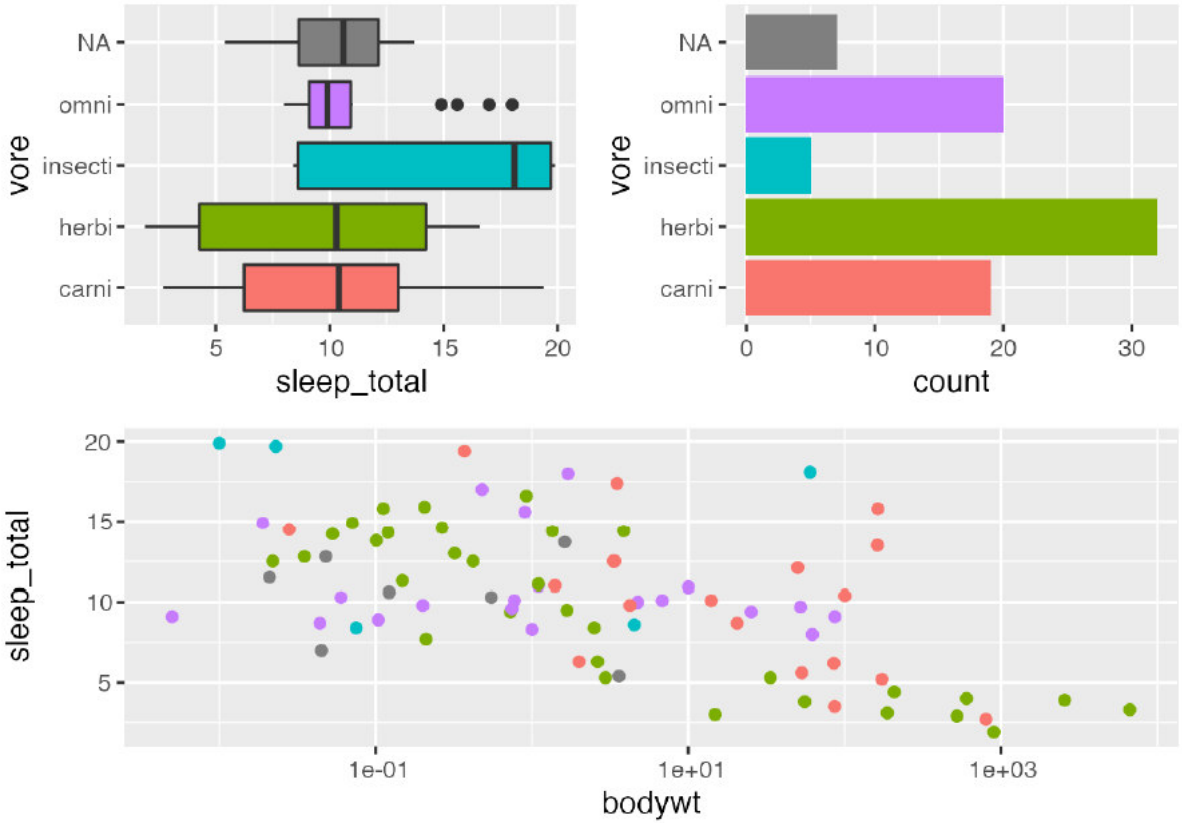
- **Patchwork**: Putting very different graphs into the same plotting window (alternative: Claus Wilke's cowplot package), can be arranged like in par(mfrow=c(2,2).
- **gganimate**: Animate your graphs. Unnecessary for some, but can look good when incorporated on webpages. Also useful for Shiny applications.
- **ggrepel** and **ggforce**: Two packages for annotating graphs.
- **ggnetwork** or **ggraph**, **ggtree** etc: Good for showing networks. Useful for ecology, biology and social science networks.
- **ggthemes/hrbrthemes**: Further themes for ggplot. For instance, can be used to recreate the Windows 95 aesthetic.





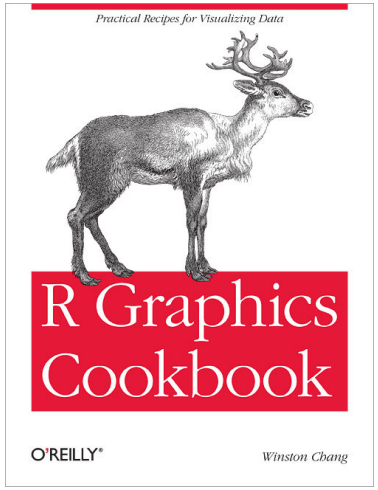


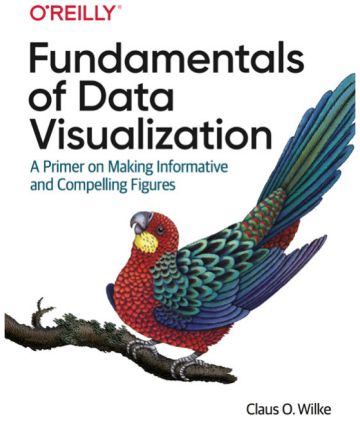
Patchwork in action

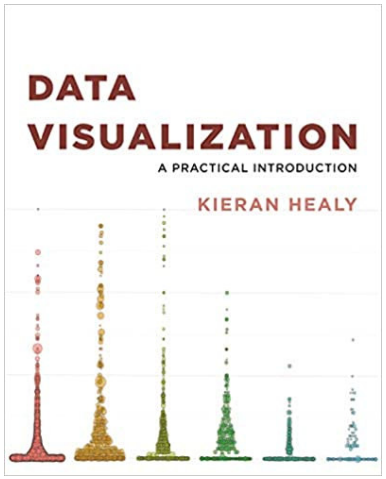


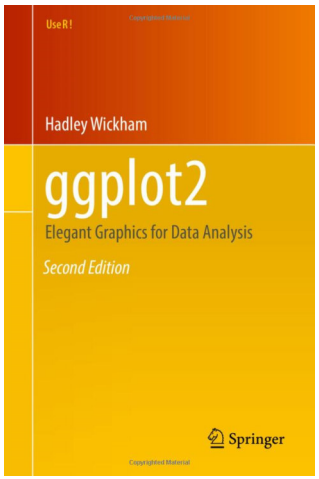
Further reading

- https://ggplot2-book.org/ : The ground-breaking book by Hadley Wickham. Freely available via URL.
- Kieran Healy. (2019). Data visualization. Princeton University Press. https://socviz.co/ - Will cost you a bit of money (only available via ILL in uni lib), worth it.
- Claus Wilke. Fundamentals of Data Visualization. Sebastopol: O'Reilly Media. Free bookdown version available at: https://clauswilke.com/dataviz/
- Chang, Winston (2021). R graphics cookbook. O'Reilly. Bookdown version: https://r-graphics.org/





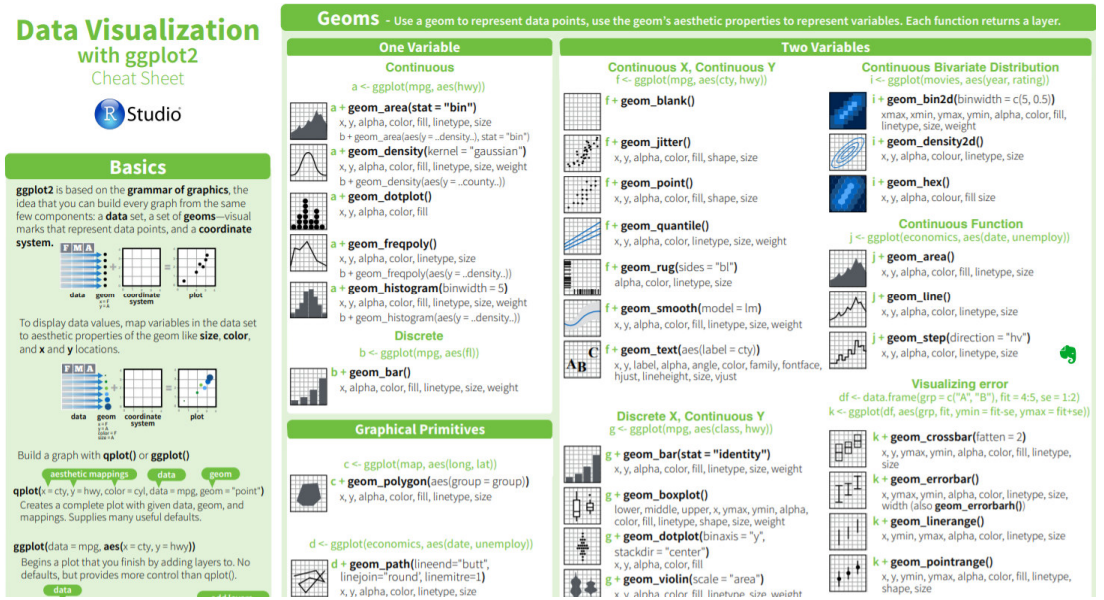




Further websites

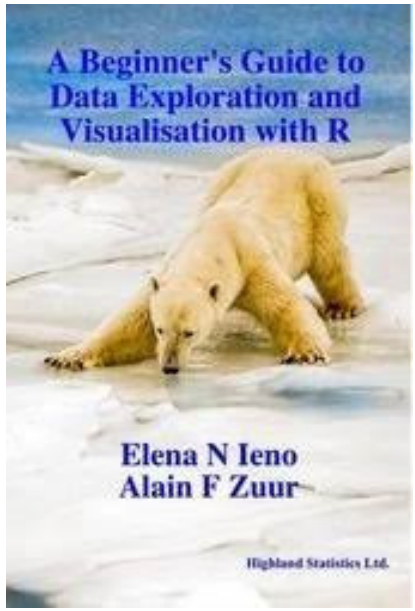
- https://nsgrantham.shinyapps.io/tidytuesdayrocks/ Lovely graphs.
- http://www.sthda.com/english/wiki/ggplot2-essentials Very helpful for getting started/overview.
- https://www.r-graph-gallery.com/ Really great website with most common chart types.
- https://ggplot2.tidyverse.org/ This is the package's main site. Also has the cheat sheets.
- https://patchwork.data-imaginist.com/ Great website by one of the developers of ggplot and patchwork.

- Cheat sheets will help you

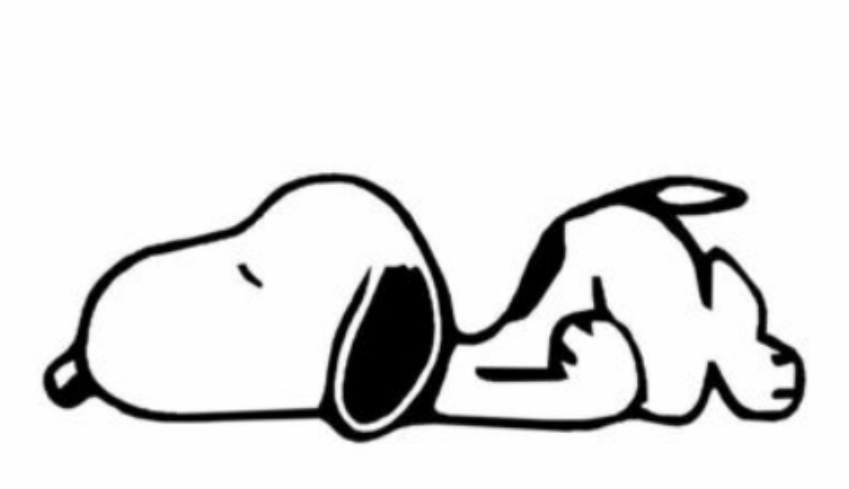


Further reading for modelling nerds

- If you do a lot of statistical modelling, testing assumptions via residual diagnostics should sound familiar. ggplot2 offers a great resource to do so graphically via facet plots and the lattice package.
- A very good overview of different types of graphs that could be used to test assumptions for modelling (also for experimental studies) is provided by Alain zuur & Elena Ieno.
- The book can be bought as a pdf from their website, an old copy is in the Freiburg uni library.

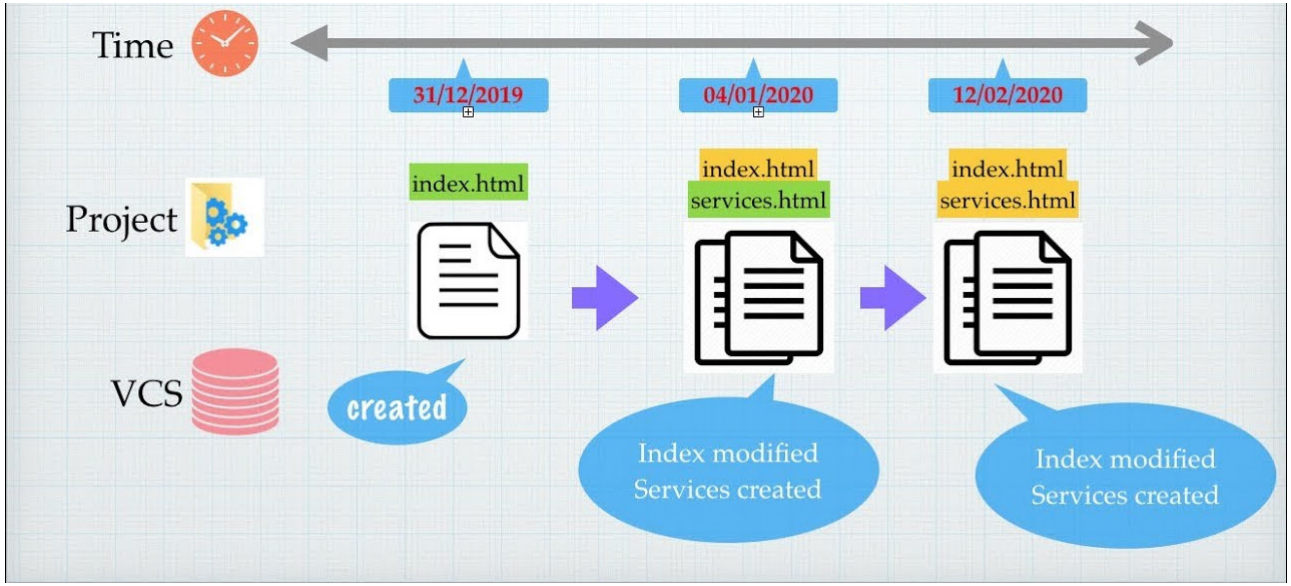


Short break



Introduction to working with Git for version control

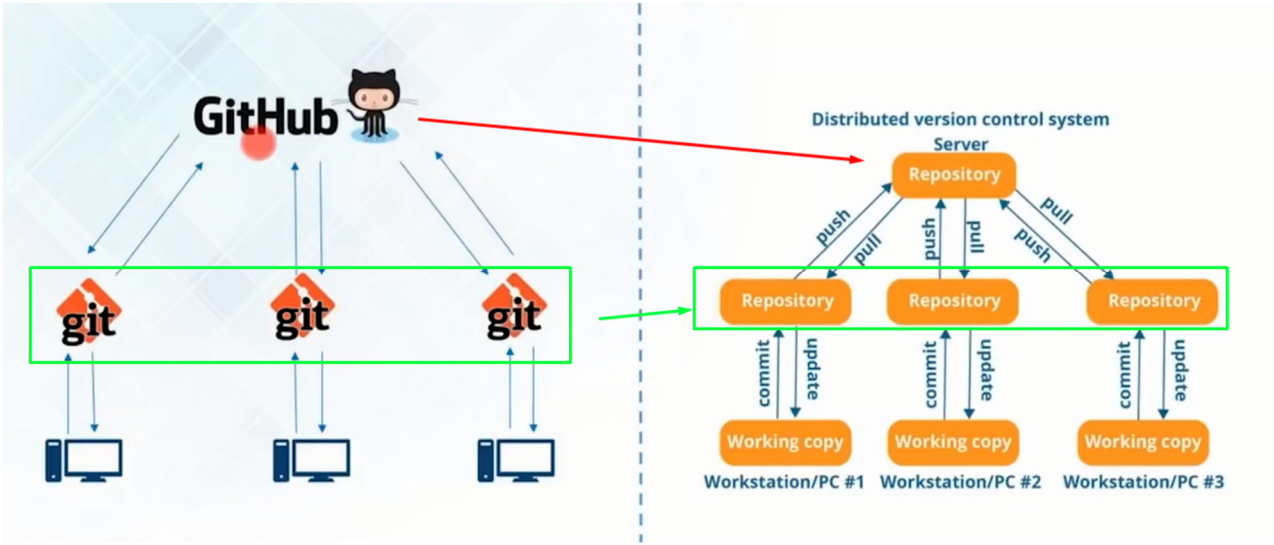
What is version control?



- Keeping track of changes is one of the most difficult parts of writing code to work with data
- Maintaining a clear and well-documented history of your work is crucial for transparency and collaboration
- Tracking changes allows reverting to earlier versions, more easily identify mistakes
- Having dozens of versions of the same file is an unstructured way of backing up work. Versioning via creating new files may also be error-prone.

What is Git?

- Git is a version control system.
- You can think of it like a programme to track changes (like in a Word file).
- Git works locally on your computer.
- Any changes to files in a folder (directory, Git speak: repository) that are tracked by Git can be stored in the cloud. This is the remote Github server.
- Github stores a copy of your files.
- Everytime you commit and push changes to Git/Github, you essentially take a snapshot of your project to the cloud.
- You can browse through the history of your project and go back to each commit.
- You can specify for which files in a project this should be done.



What is version control?

- A version control system is a tool for managing a collection of program code that provides you with three important capabilities: reversibility, concurrency, and annotation (Raymond, E. S. (2009). Understanding version-control systems. http://www.catb.org/esr/writings/version-control/version-control.html [accessed 17/10/2022]
  - Essentially, you can roll back to previous versions.
  - Each new version (or checkpoint) of your files needs to be explicitly committed. Git tracks changes in individual files and provides a snapshot of changes since the last time you committed.
  - For text files (R files), Git tracks changes line by line. You can track changes coming from different collaborators line by line as well.

Git core concepts

Repository (repo)	A database of your file hisotry, containing all the checkpoints of all your files along with some meta-data. This database is stored in a hidden sub-directory .git within your project directory. Sometimes your project folder is also called a „repo“.
Commit	A snapshot of your work at a given time that has been added to the repository. It comes with a short description (commit message) of what was changed. Committed snapshots are browsable as „History“ on Github (or any other remote Git system).
Remote	A link to a copy of your repository on a different machine. This link points to a location on the web where the copy is stored. Typically a central version of the project („master“).
merging	Git supports having different versions of your work all living side by side (in what we call branches), which may be created by one person or by collaborators. Changes made in different branches are merged back together without the need to manually copying or pasting files. Basically, it keeps your sanity once several people work on a project or you work on a project on different machines.

When is Git for me?

- Let‘s be honest here – setting up Git, Github and R projects so that all systems talk to each other does take some getting used to.
- You may go through the “pain“ in case the following applies to you:
  - You want to show your work and communicate via a platform that is used by large parts of the R community and increasingly by the scientific community.
  - You work collaboratively with others on a project so an extent that all of you contribute R syntax to a project. You want the equivalent to editing a file collaboratively like in GoogleDocs. Git and Github are the equivalent to GoogleDocs, just for R.
  - The pain is less harsh when you work on a Mac or in Linux. Unlike with Windows machine, you do not need to set up Git in a separate step.
  - Your organisation or your research partners leave no choice. An organisation can apply (and pay) for an advanced account that allows an unlimited number of private repositories.
  - You find remote working attractive or are left no choice because your workplace is not data-friendly. Having a Github account at least let‘s you commit changes manually from your work machine into Github from which you can pull it back onto your other devices. It‘s advisable to negotiate with them and .gitignore all data files (if your data falls under GDPR).
  - You work on several machines with one project. You can keep track of the project via branches.

When shouldn‘t I bother?

- You already have a system in place that lets you track changes to a sufficient amount.
- You do very specific data analysis projects and you document into other systems (i.e., Open Science Framework) than Github.
- You currently feel like it‘s too overwhelming.
- You are the only person in your group working on data analysis. You are not in a situation in which you need to exchange R files with anybody.

Some of the added benefits of Github

- All of R's development work and a lot of cool packages have a representation on Github. At least finding your way around Github is a good idea to get involved with the community a bit more.
- Exposure
  - Others can see your work. It's a good advertisement.
  - Giving back to the community. I often try to find public R projects on Github to look at their code and learn by example.
  - You find the most surprising stuff there. As a person teaching stats, the exercises and solutions that are maintained there are a life saver.
  - It is very easy to host webpages (blogs, data analysis portfolios, bookdowns) via Github pages without having to pay for server space.

- Even without using version control, using Github is the prime way to interact with the community (apart from your local R group, stackoverflow and R bloggers ;-).
- I follow all my R friends and their package development endeavours via Github (check out Daniel Lüdecke, aka „Strenge Jacke", developer of sjmisc and sjplot).

Alternatives to Github

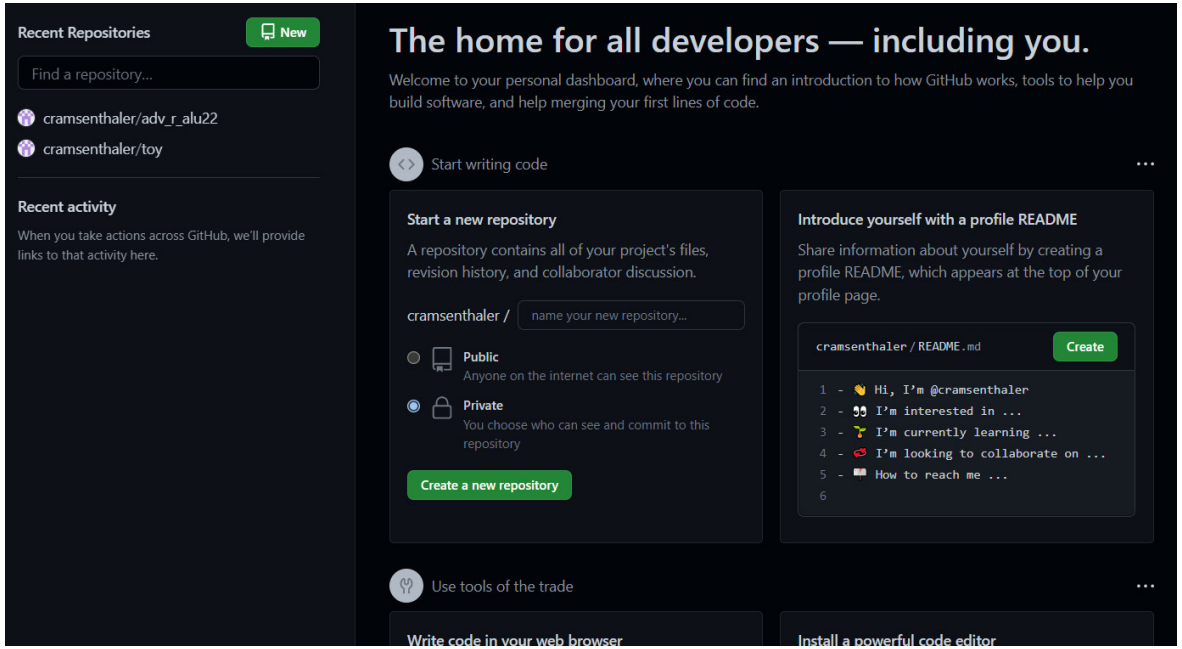
- You will need to install Git on your local machine.
- The remote server (where a copy of your repository is kept in the cloud) can be hosted by different systems.
- Alternatives to Github are:
  - BitBucket: https://bitbucket.org (different pricing model – unlimited private repos but limited number of collaborators)
  - GitLab: https://gitlab.com (more geared towards genuine software projects)

- When signing up to any service, you can usually say you are a teacher. Teachers get stuff for free. The service usually wants to see an email address from a university to verify your status. You can sign up to the service with a private address and email them your „official“ email address later.
- It is up to your department to negotiate buying a departmental/corporate account on these services. For projects to be private (cannot be accessed by anybody from the URL), they often need to be tied to a corporate account. You will also need to clear this with the Data Protection Unit in the uni/hospital.
- If you use third-party funding that comes with the necessity to adhere to the TOP guidelines, you can usually apply for part of your funding to be spent on a Github account (DGP, BMBF).

1. Signing up to Github

- I recommend signing up to Github as the first step.
- https://github.com/
- Sign up: Username (can be anything), email address (I use my private for my own stuff).
  - It asks you to complete some information about yourself.
  - The help pages are brilliant.
  - You can create a repository on there
  - Or push this from your local machine via Git.

- Github has a Desktop app.
  - Does the same stuff as the web interface.
  - I usually work in the web interface.



2. Installing Git

- You can work with Git straight away if you are on a Mac or a Linux machine. You access this through the command shell (cmd).
- For poor Windows folks, go to:
  - https://git-scm.com/download/win
  - Follow the setup file. Install everything as recommended.
  - In your Menu, find „Git Bash“. Open this.
  - You are faced with having to setup (configure) Git once. (Also applies to first-time Git users on Mac or Linux).

```
# Configure ‘git‘ on your machine (only needs to be done once)

# Set you name to appear alongside your commits
# This *does not* need to be your Github username
git config --global user.name “YOUR FULLNAME“

# Set your email address
# This *does* need to be the email associated with your GitHub account
git config --global user.email “YOUR_EMAIL_ADDRESS“
```


2. Installing Git

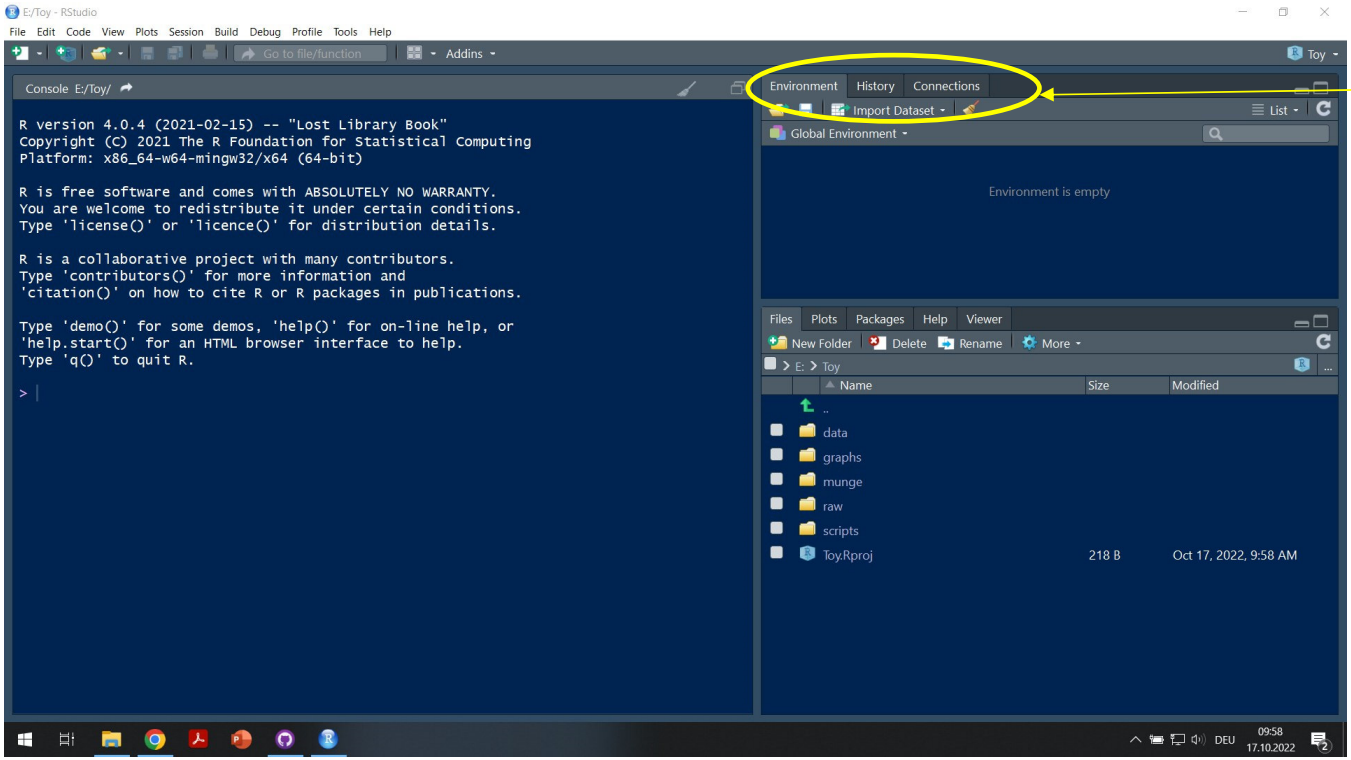
- You can actually control Git only via the command shell.
- Some folks do this. I don‘t (I am bear of very little brain…). It‘s one of my pet projects to learn to speak Git a bit better. There are a number of great resources out there. I have provided a few in our Workshop material.

- You may want to work with an SSH key passphrase, please see here: https://docs.github.com/en/authentication/connecting-to-github-with-ssh/working-with-ssh-key-passphrases

- I recommend doing this as a small added layer of security. The data security people in your organisation may require more safety protocols.

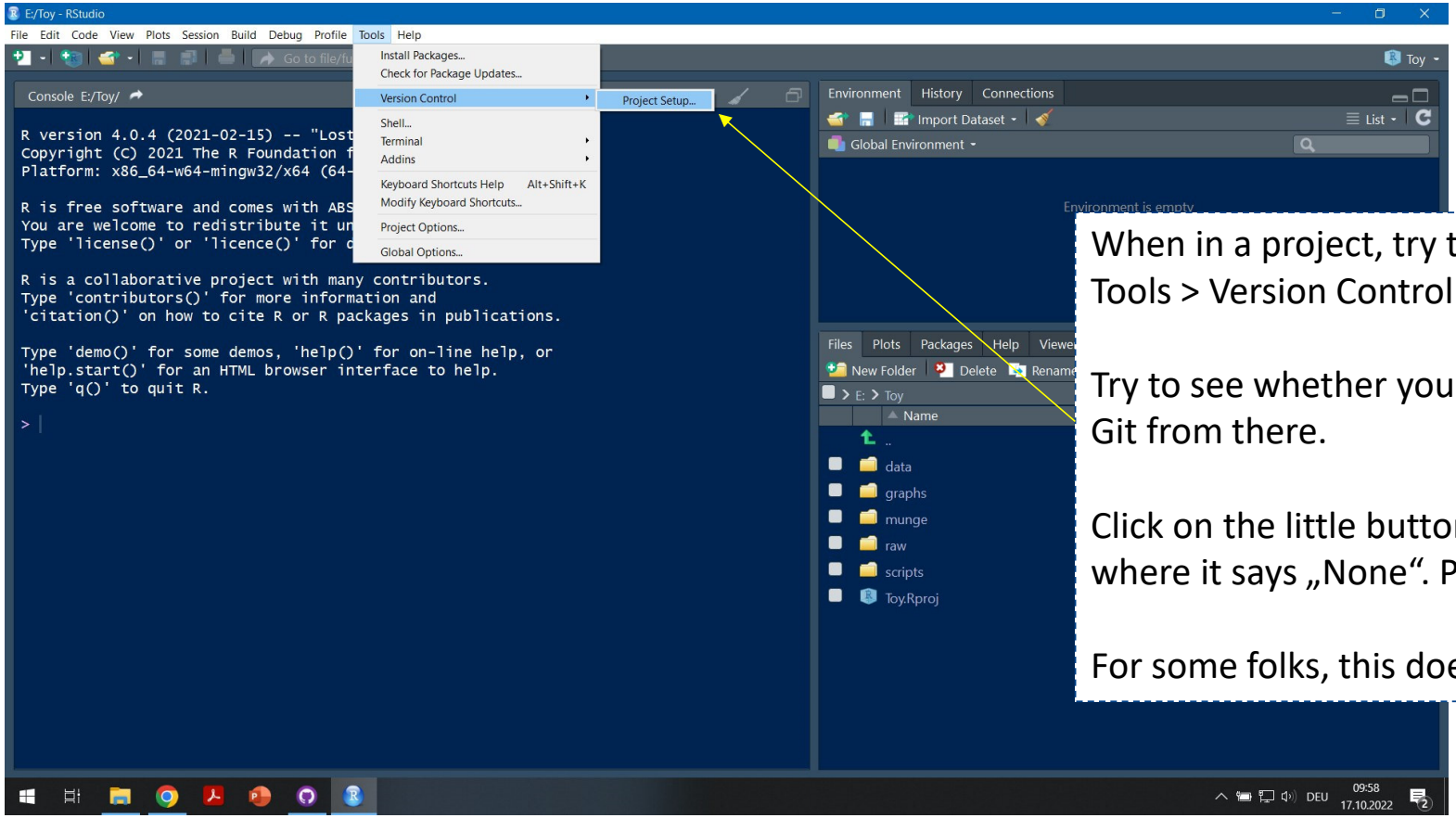
3. Getting R Studio to talk to Git and Github

- And now for the fun part. We need to get RStudio to talk to Git. We will now install version control **for an existing R** project.
- Open RStudio. Do you have a „Git“ tab in your Environment pane?



Git tab is missing.

3. Getting R Studio to talk to Git and Github



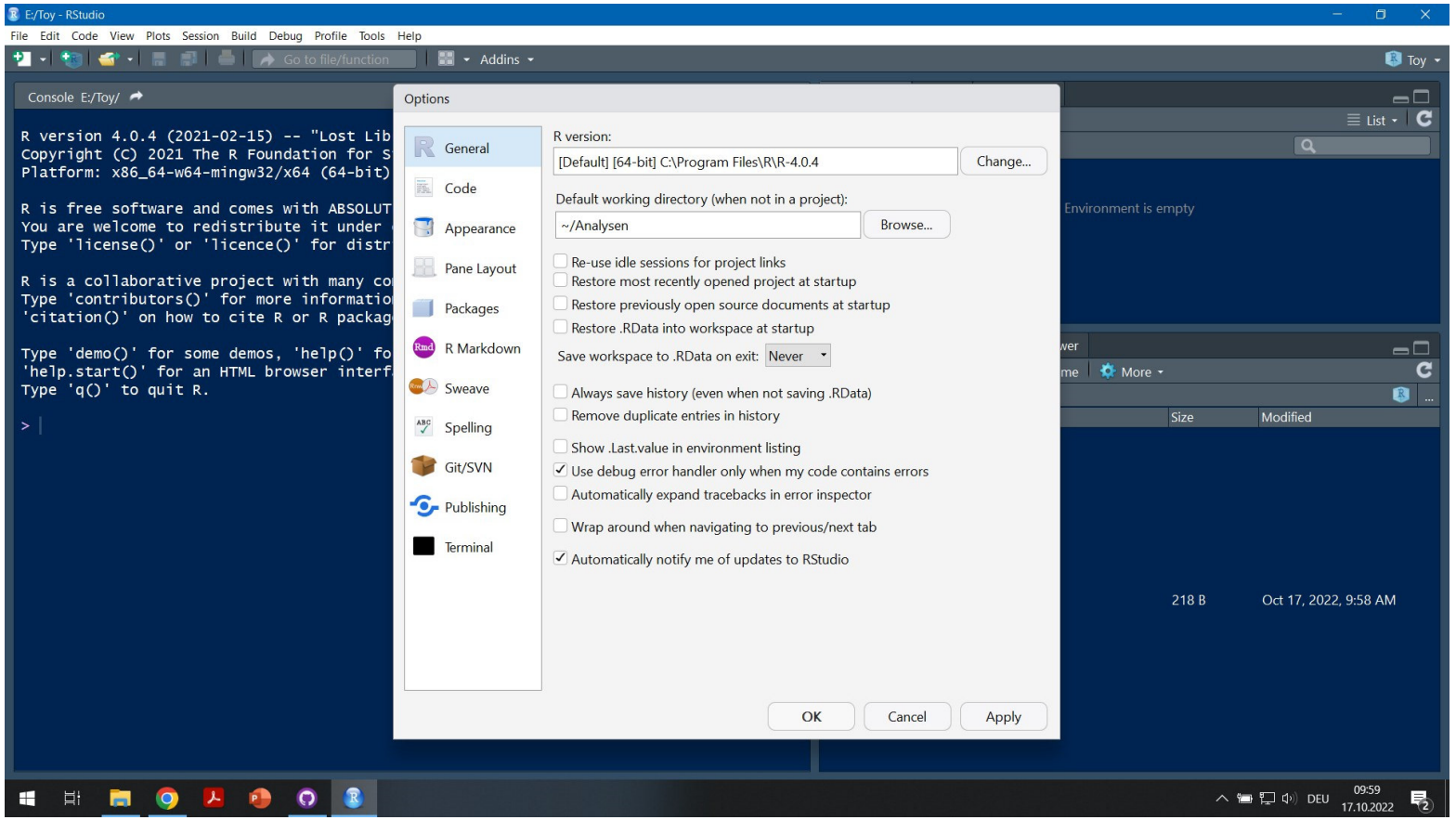
When in a project, try to go to Tools > Version Control > Project Setup.

Try to see whether you can simply enable Git from there.

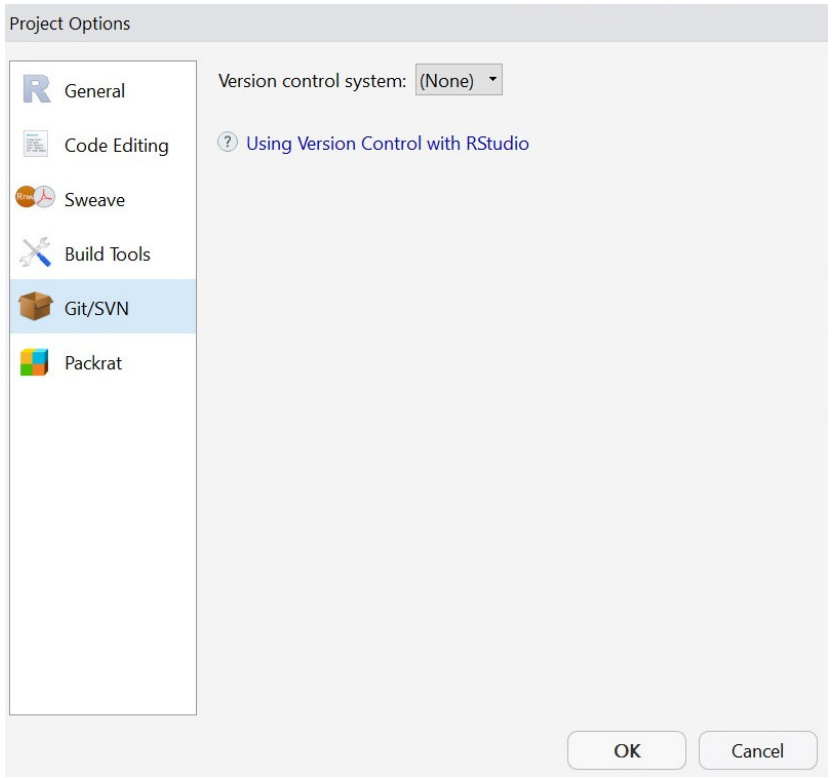
Click on the little button on the Git/SVN tab where it says „None“. Put it to „Git“.

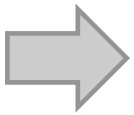
For some folks, this does not work.

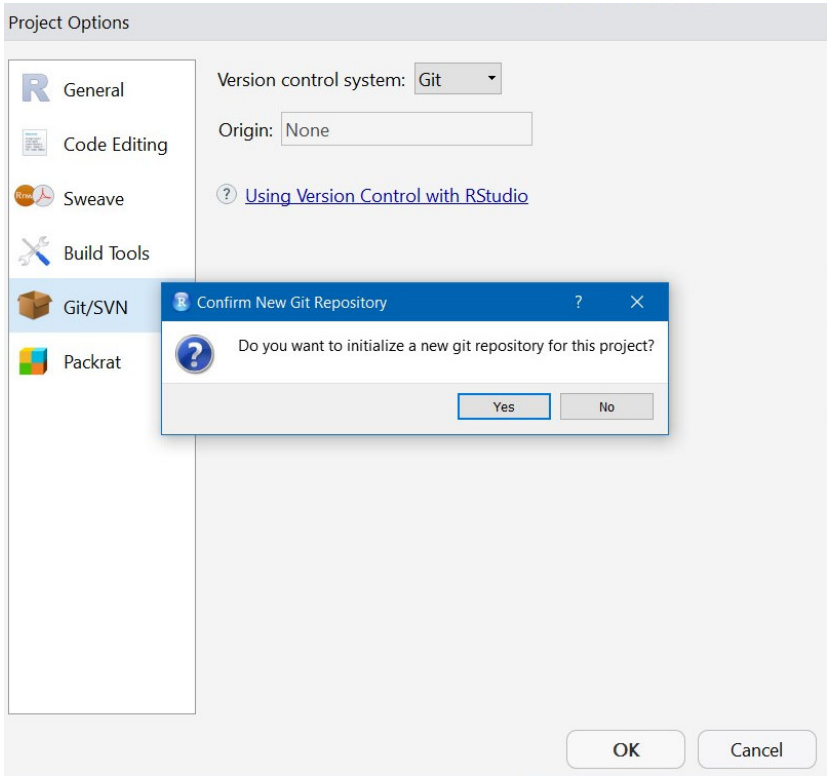
3. Getting R Studio to talk to Git and Github



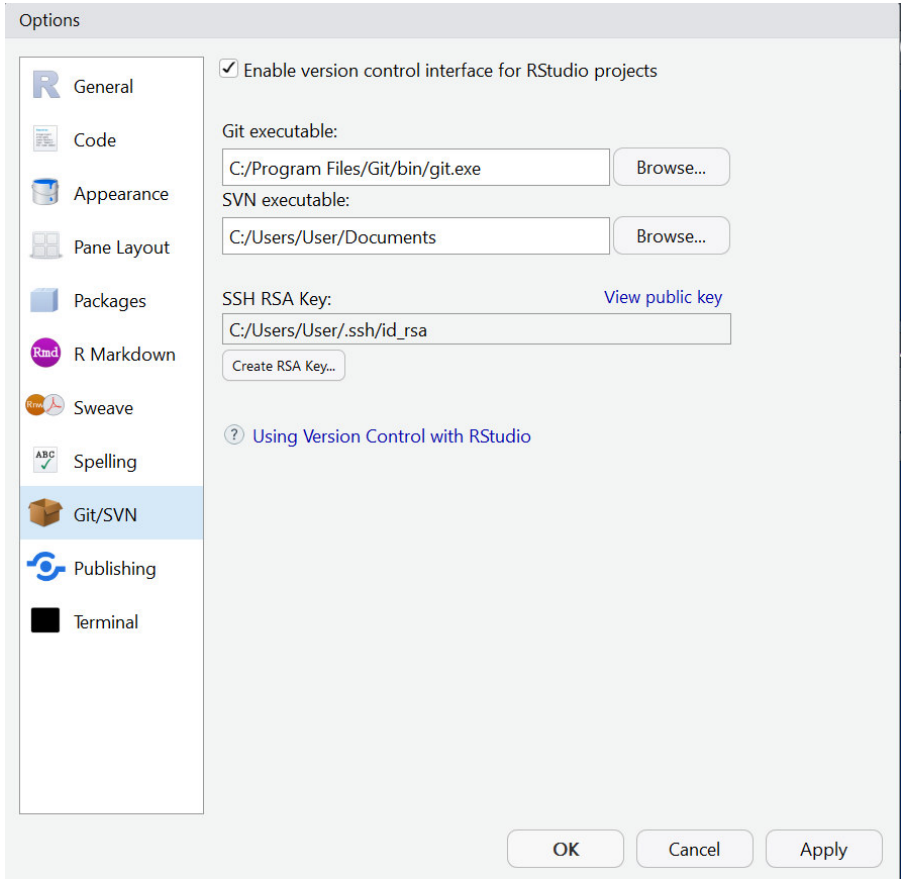
3. Getting R Studio to talk to Git and Github







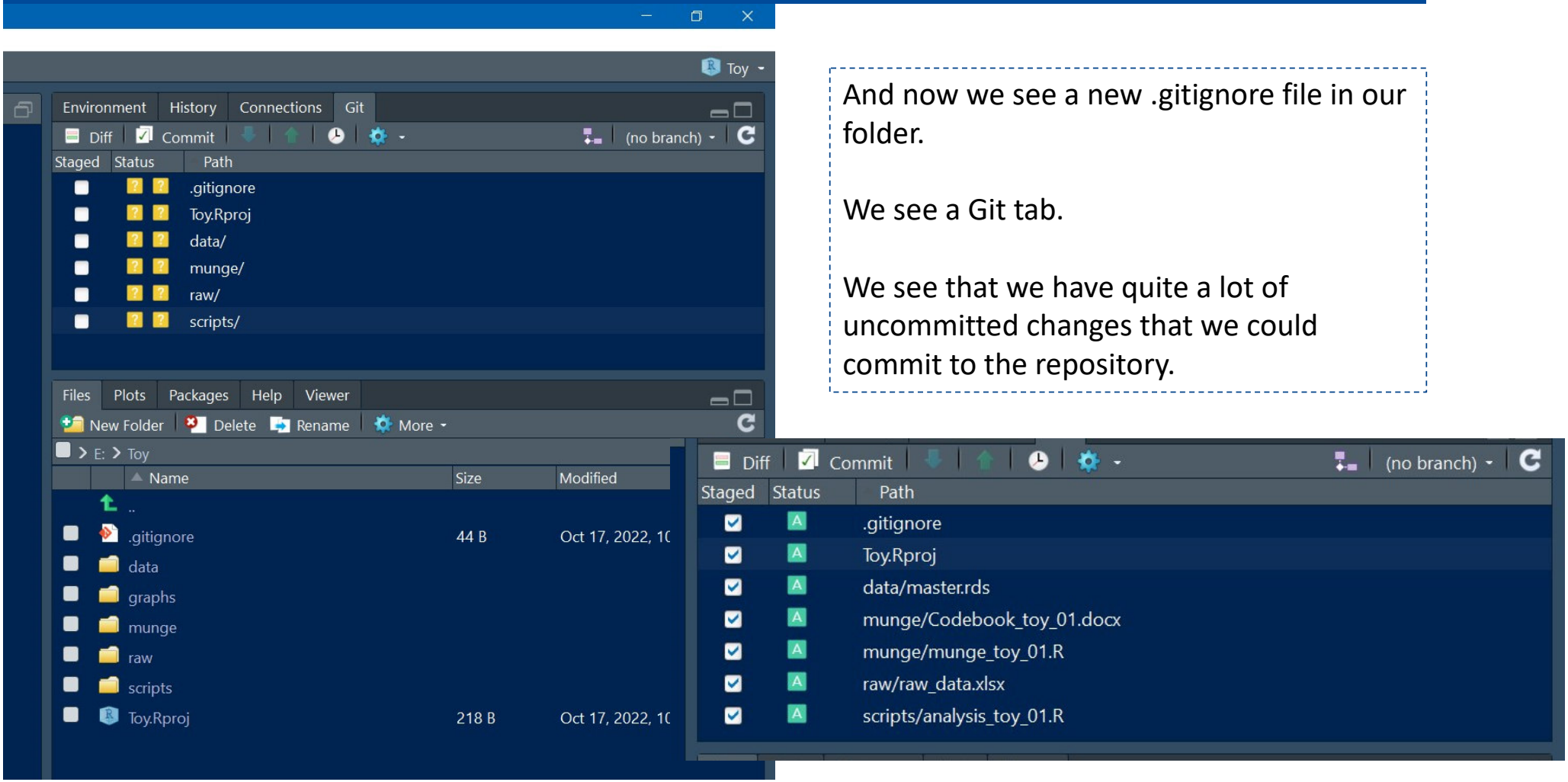
3. Getting R Studio to talk to Git and Github



In Tools > Global Options > Git/SVN

- Tick the box „Enable version control interface for RStudio projects.
- Under Git executable:
  - Browse to the folder in your program files where the git.exe lives.
- SVN executable:
  - Can be left blank or put to the default folder.
  - Needs to have a special SVN programme installed (command line) if Git Bash does not work.
- SSH RSA Key: need to copy over file name from the SSH key passphrase you sit in Git (see earlier slide).

4. Using version control from within R

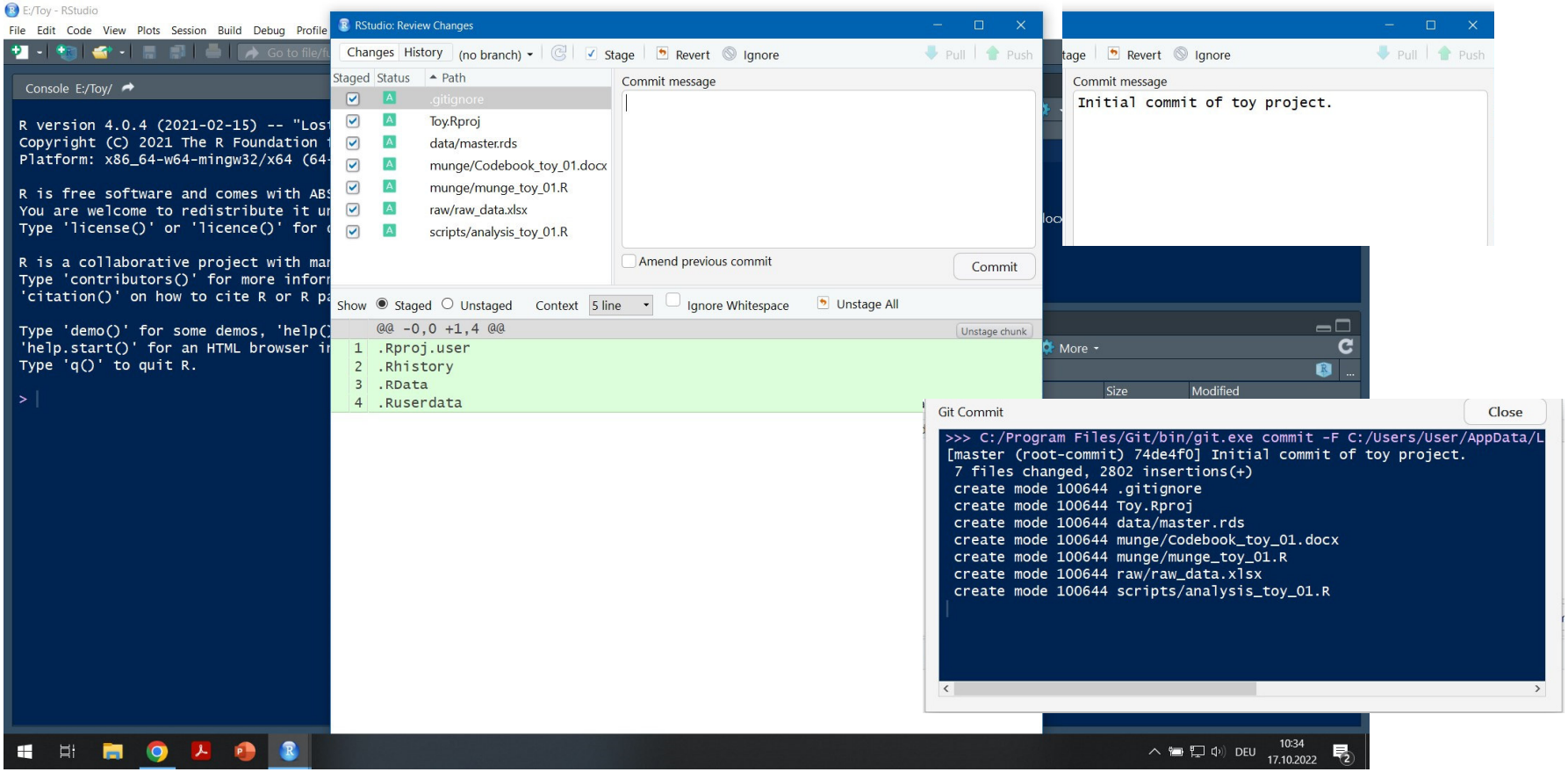


And now we see a new .gitignore file in our folder.

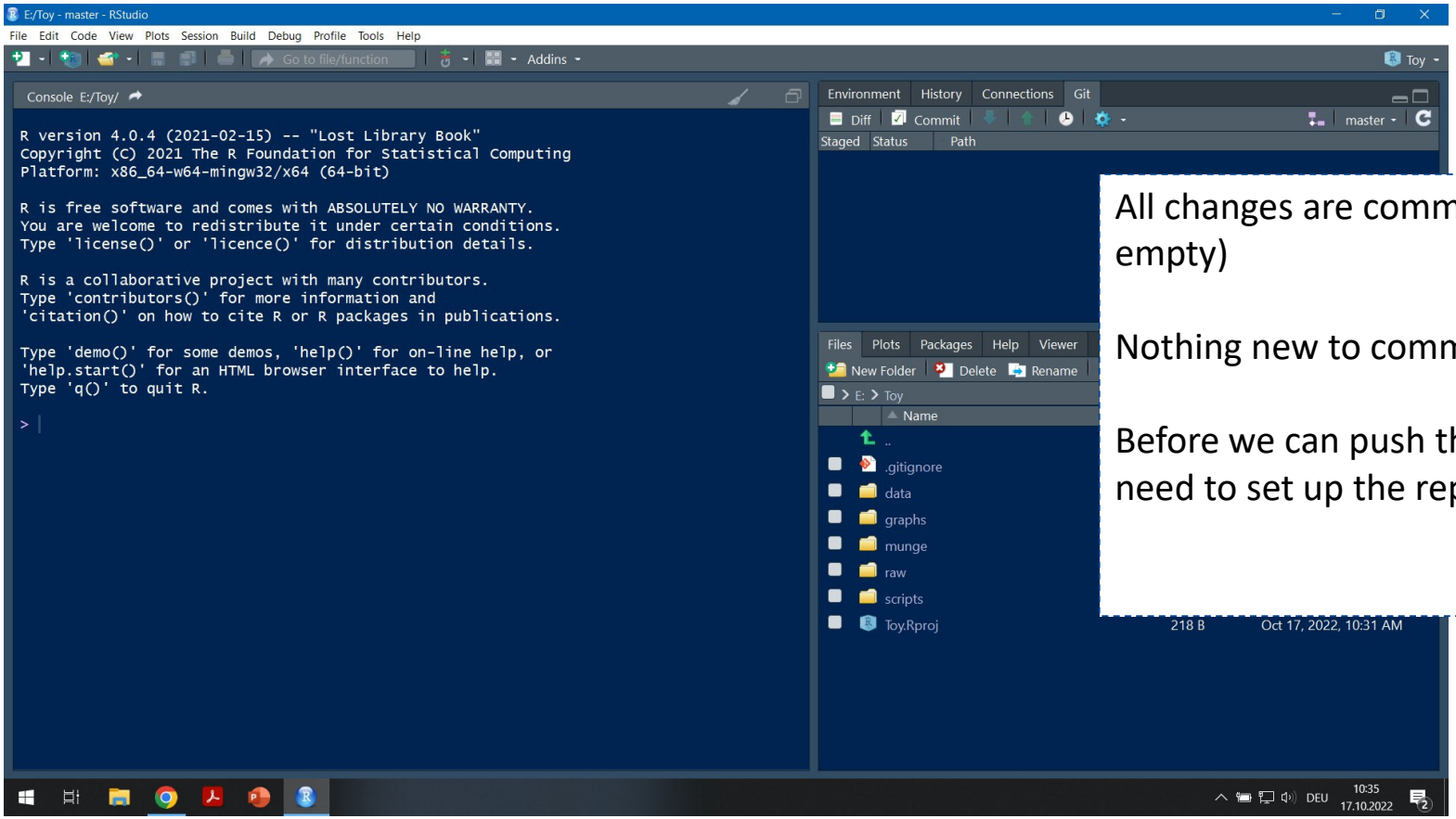
We see a Git tab.

We see that we have quite a lot of uncommitted changes that we could commit to the repository.

4. Using version control from within R



4. Using version control from within R

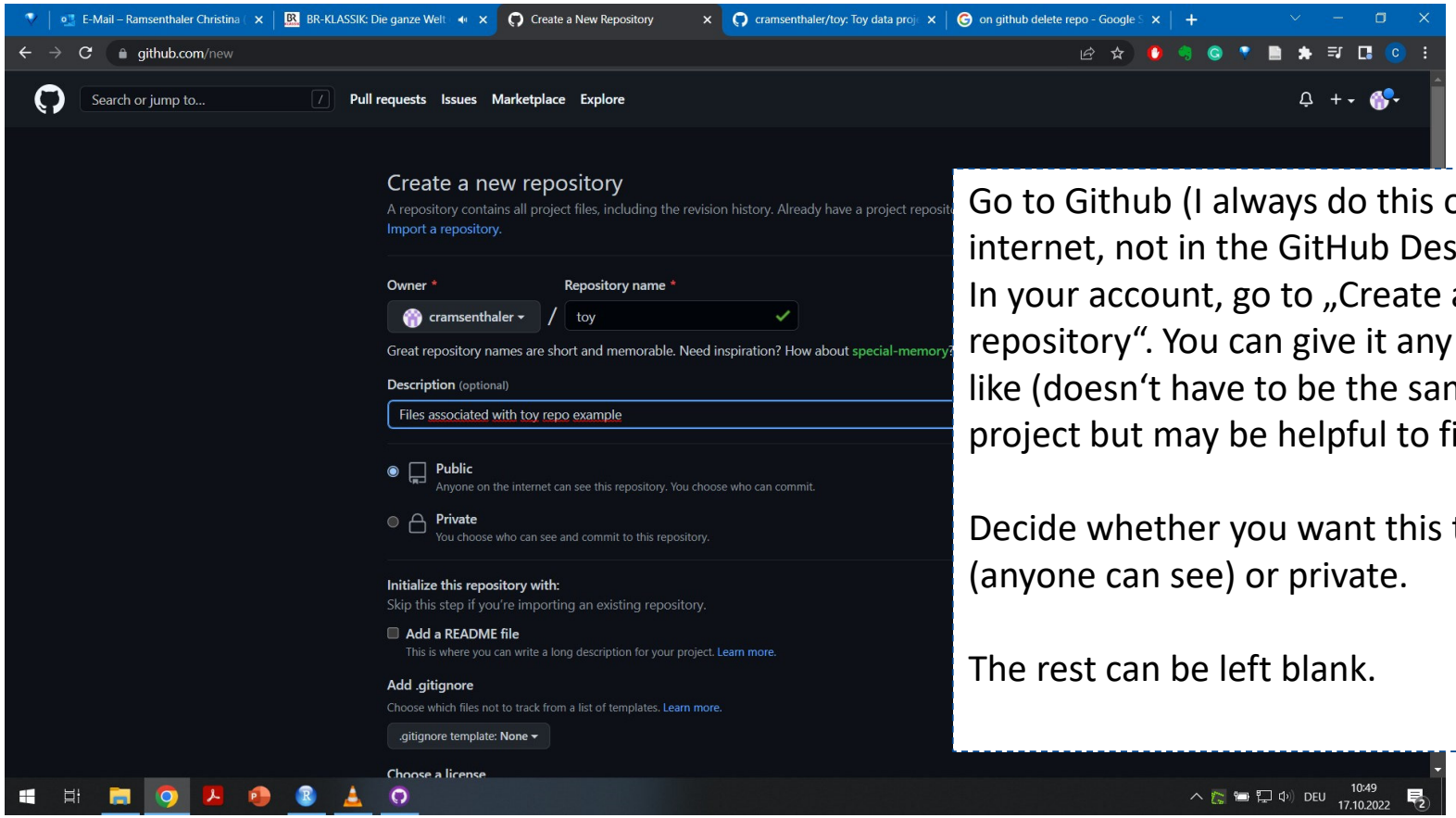


All changes are committed. (Git tab is empty)

Nothing new to commit.

Before we can push this to GitHub, we need to set up the repository on Github.

4. Using version control from within R

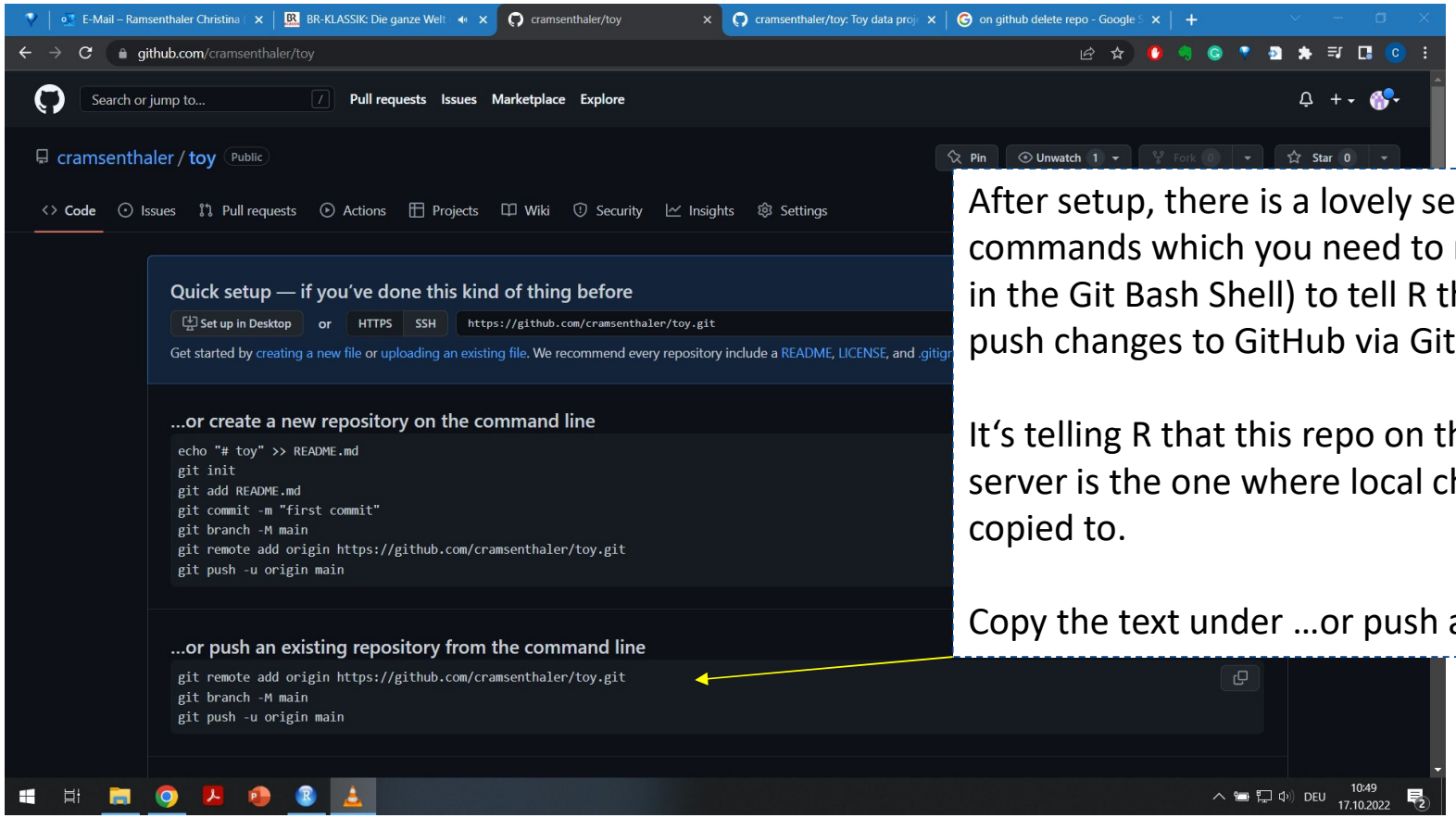


Go to Github (I always do this on the internet, not in the GitHub Desktop app). In your account, go to „Create a new repository“. You can give it any name you like (doesn‘t have to be the same as the R project but may be helpful to find stuff).

Decide whether you want this to be public (anyone can see) or private.

The rest can be left blank.

4. Using version control from within R

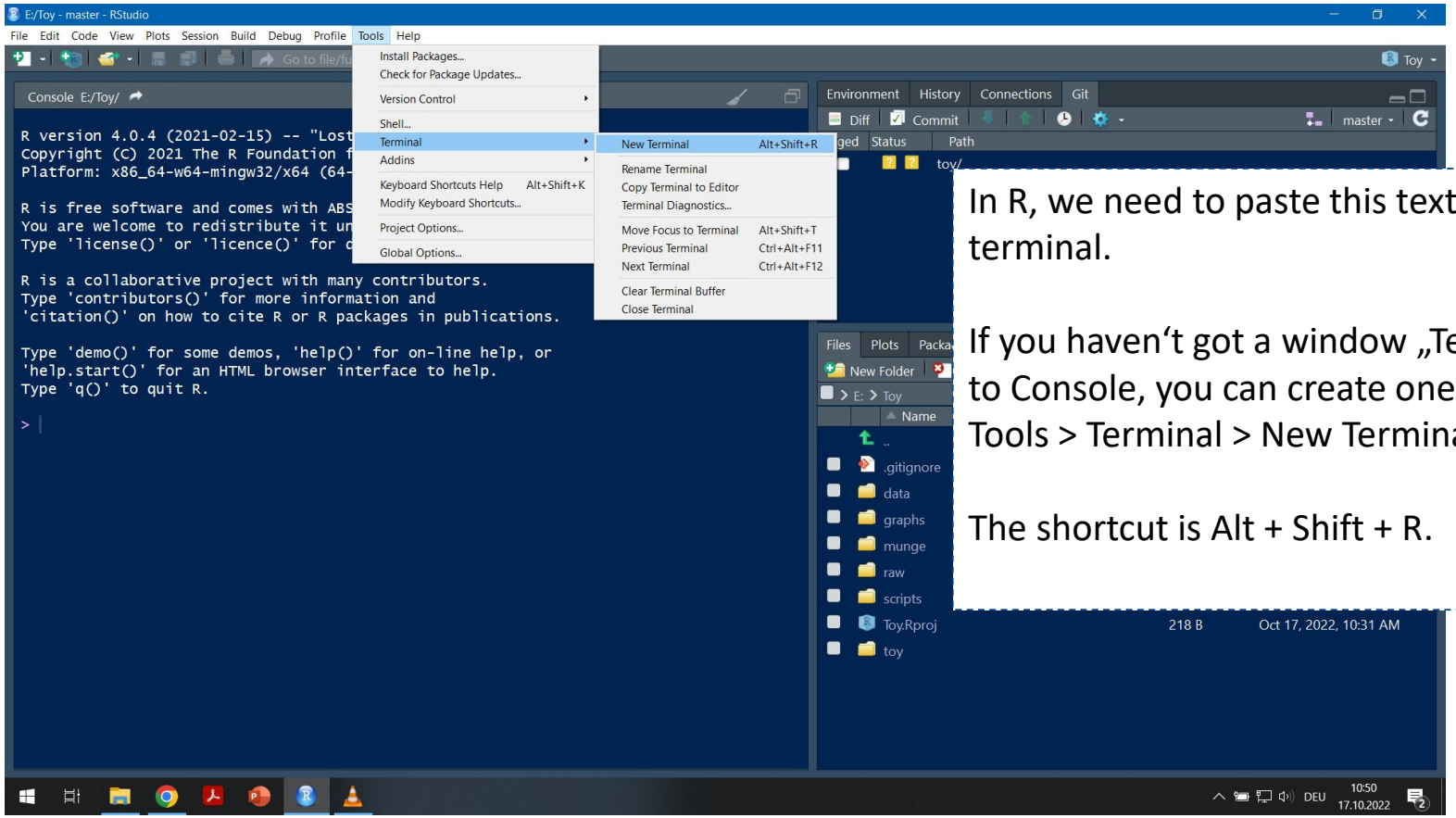


After setup, there is a lovely set of commands which you need to run in R (or in the Git Bash Shell) to tell R that it should push changes to GitHub via Git.

It's telling R that this repo on the GitHub server is the one where local changes are copied to.

Copy the text under …or push an existing…

4. Using version control from within R

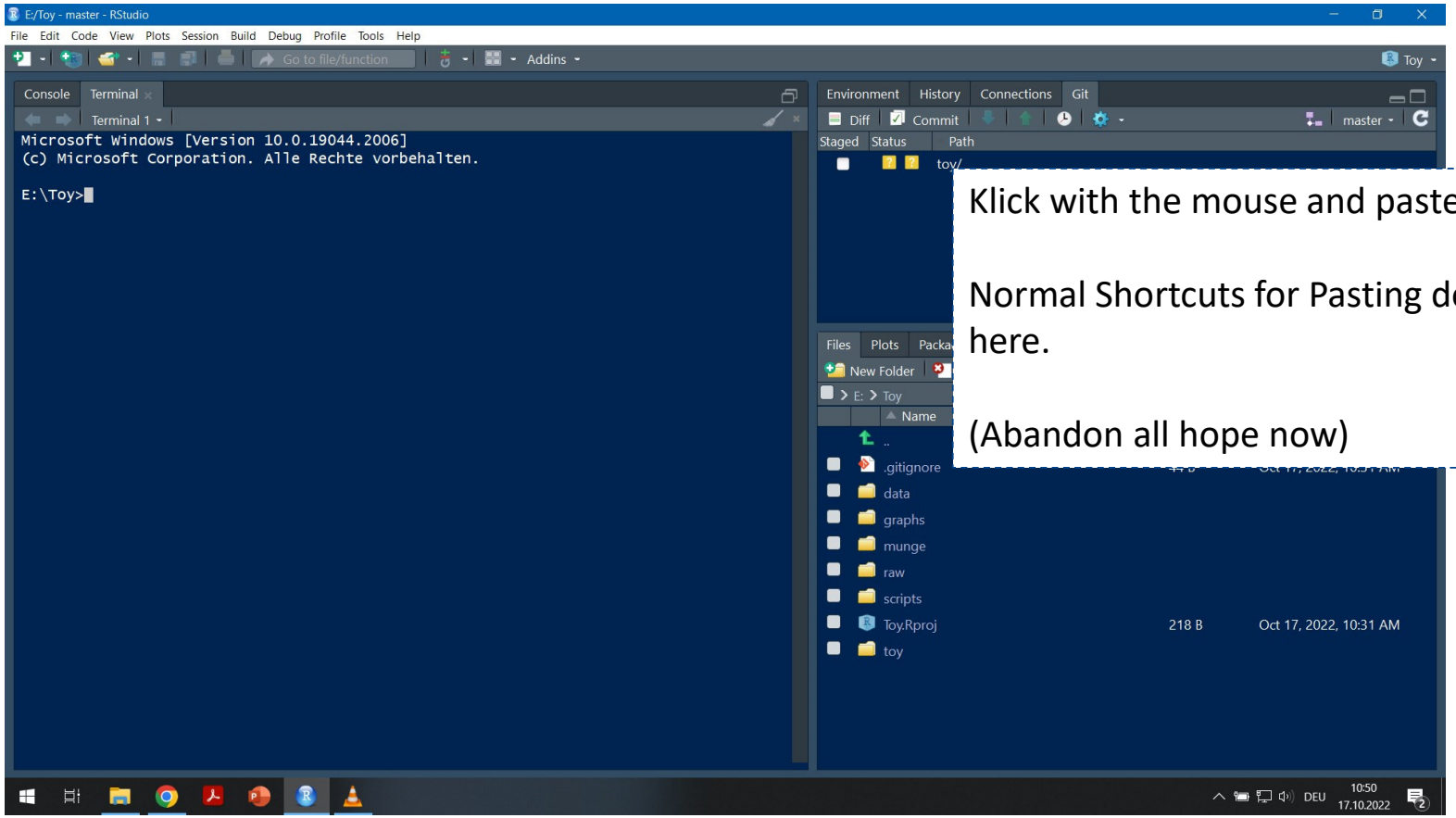


In R, we need to paste this text into a terminal.

If you haven‘t got a window „Terminal“ next to Console, you can create one by going to Tools > Terminal > New Terminal.

The shortcut is Alt + Shift + R.

4. Using version control from within R

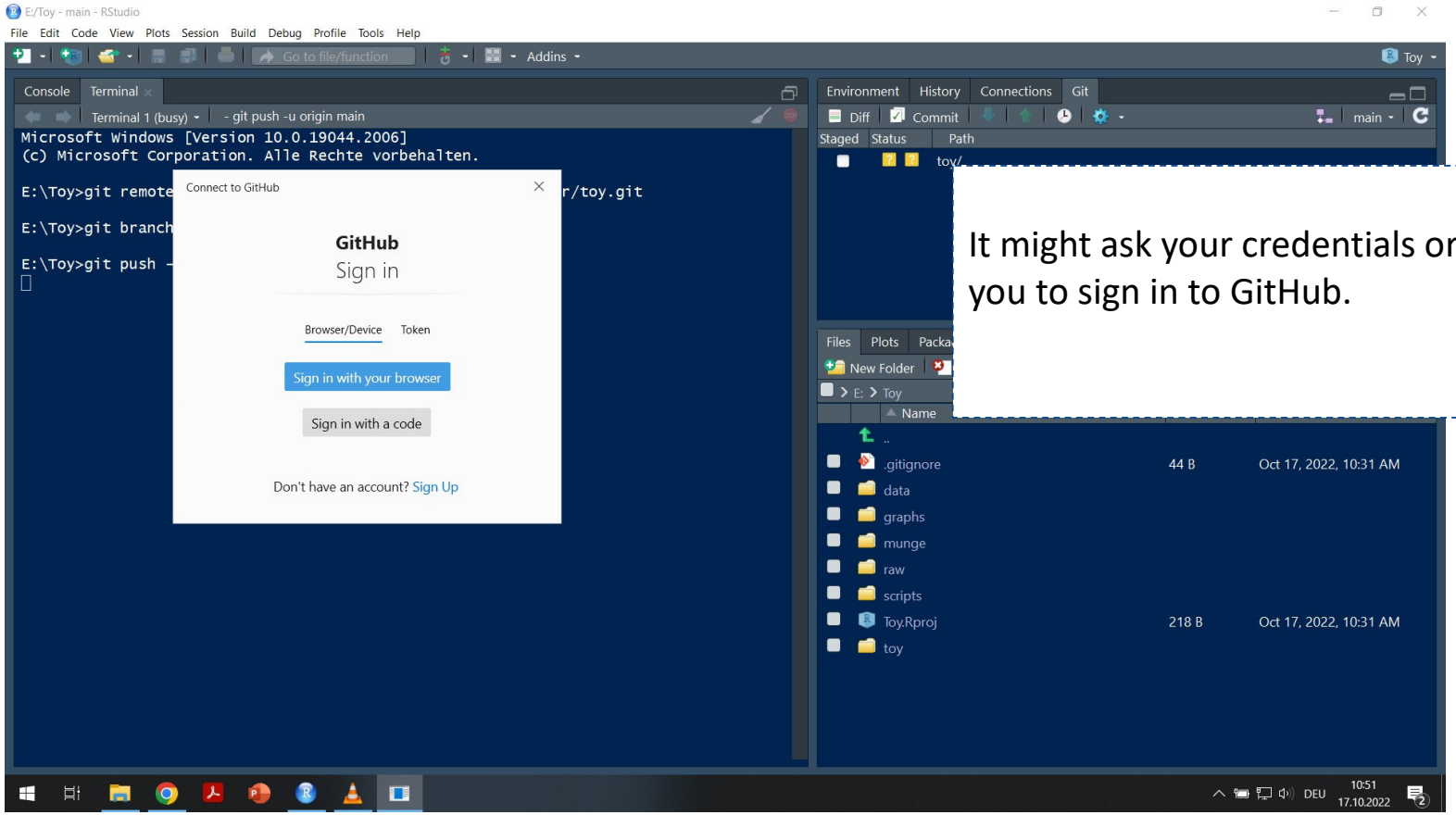


Klick with the mouse and paste the text.

Normal Shortcuts for Pasting do not work here.

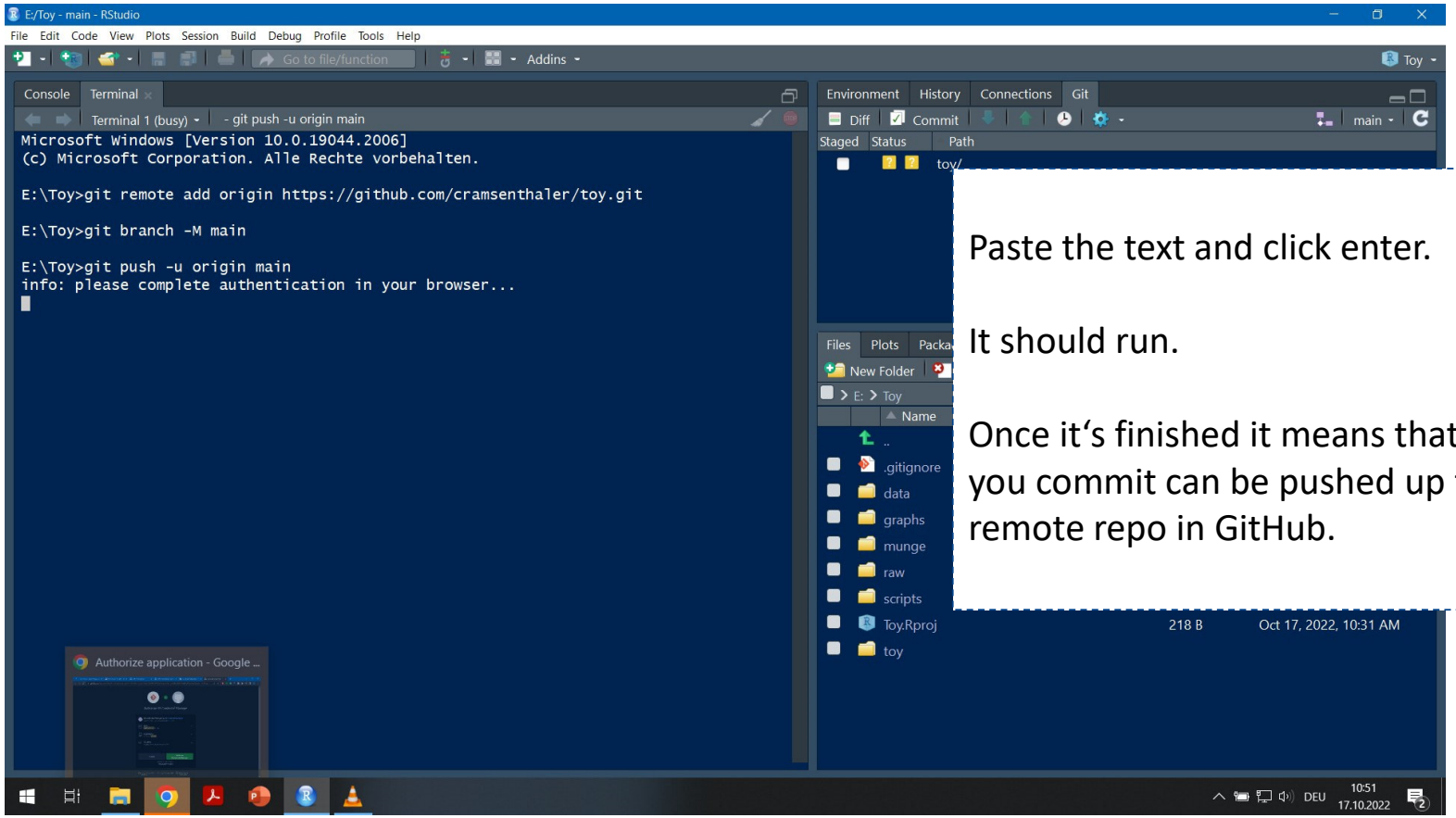
(Abandon all hope now)

4. Using version control from within R



It might ask your credentials or might ask you to sign in to GitHub.

4. Using version control from within R

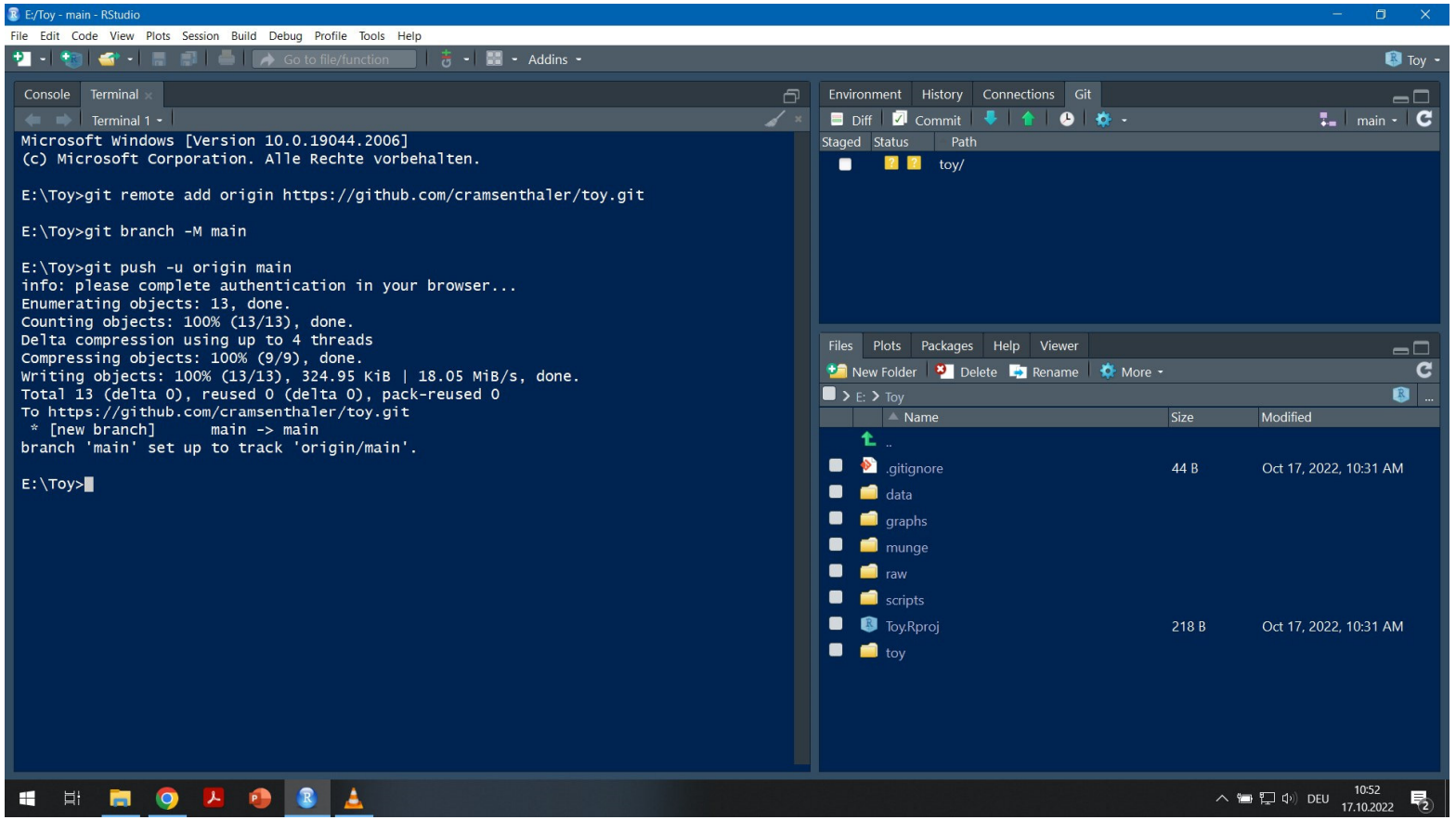


Paste the text and click enter.

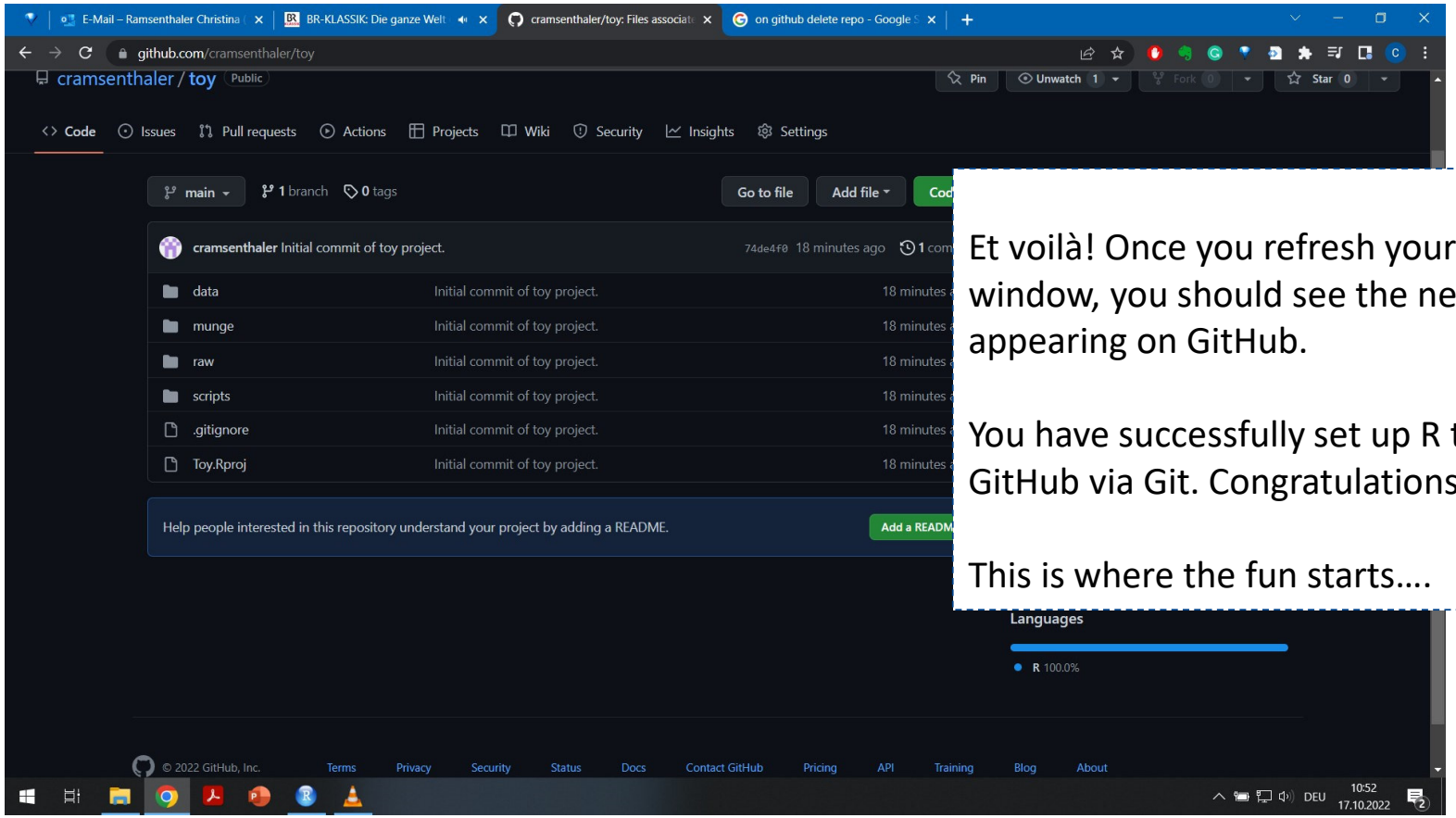
It should run.

Once it‘s finished it means that the changes you commit can be pushed up to the remote repo in GitHub.

4. Using version control from within R



4. Using version control from within R

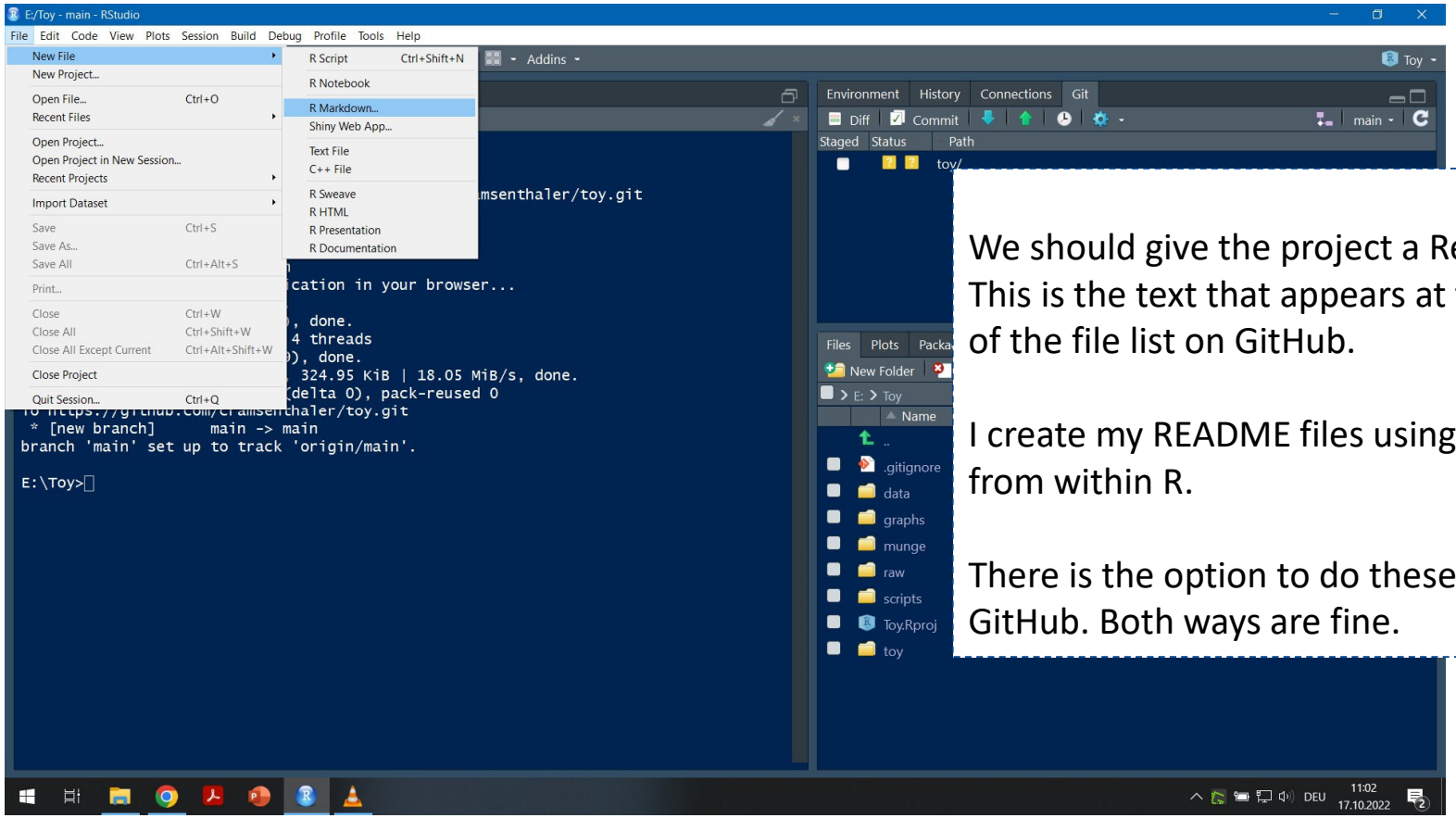


Et voilà! Once you refresh your browser window, you should see the new files appearing on GitHub.

You have successfully set up R to talk to GitHub via Git. Congratulations!

This is where the fun starts….

4. Using version control from within R

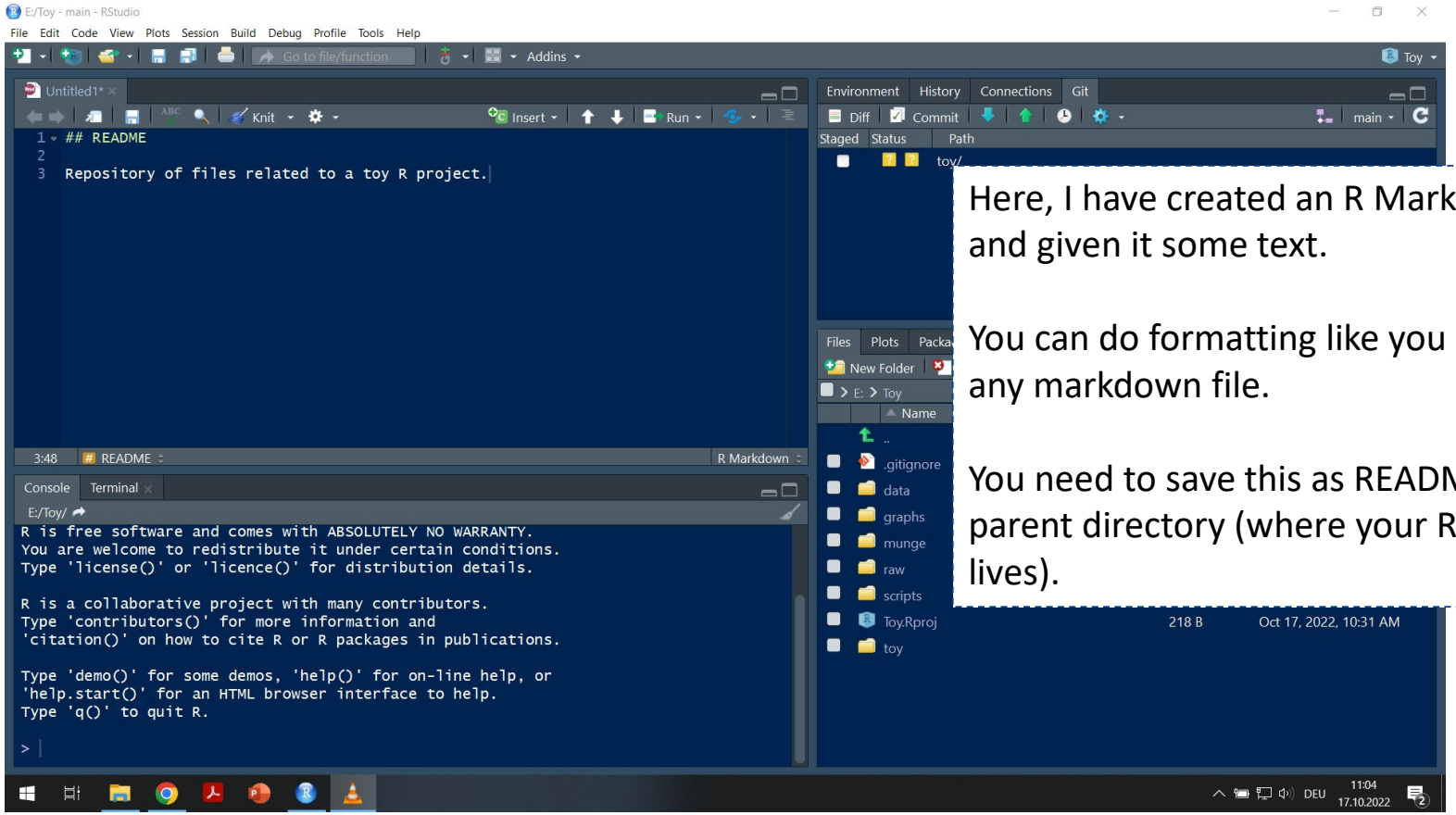


We should give the project a Readme file. This is the text that appears at the bottom of the file list on GitHub.

I create my README files using markdown from within R.

There is the option to do these directly on GitHub. Both ways are fine.

5. Setting up the README file.

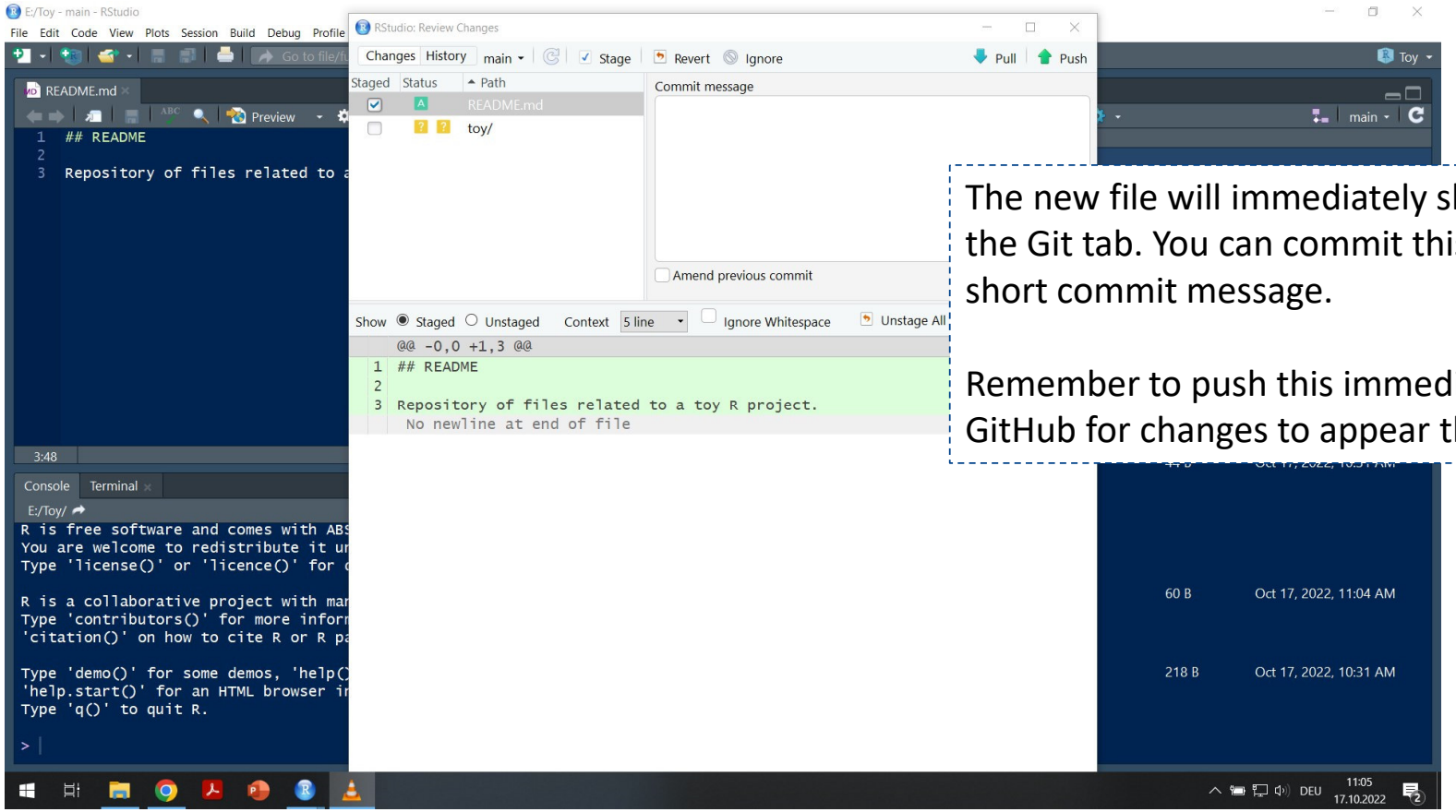


Here, I have created an R Markdown file and given it some text.

You can do formatting like you would do in any markdown file.

You need to save this as README.md in the parent directory (where your R project file lives).

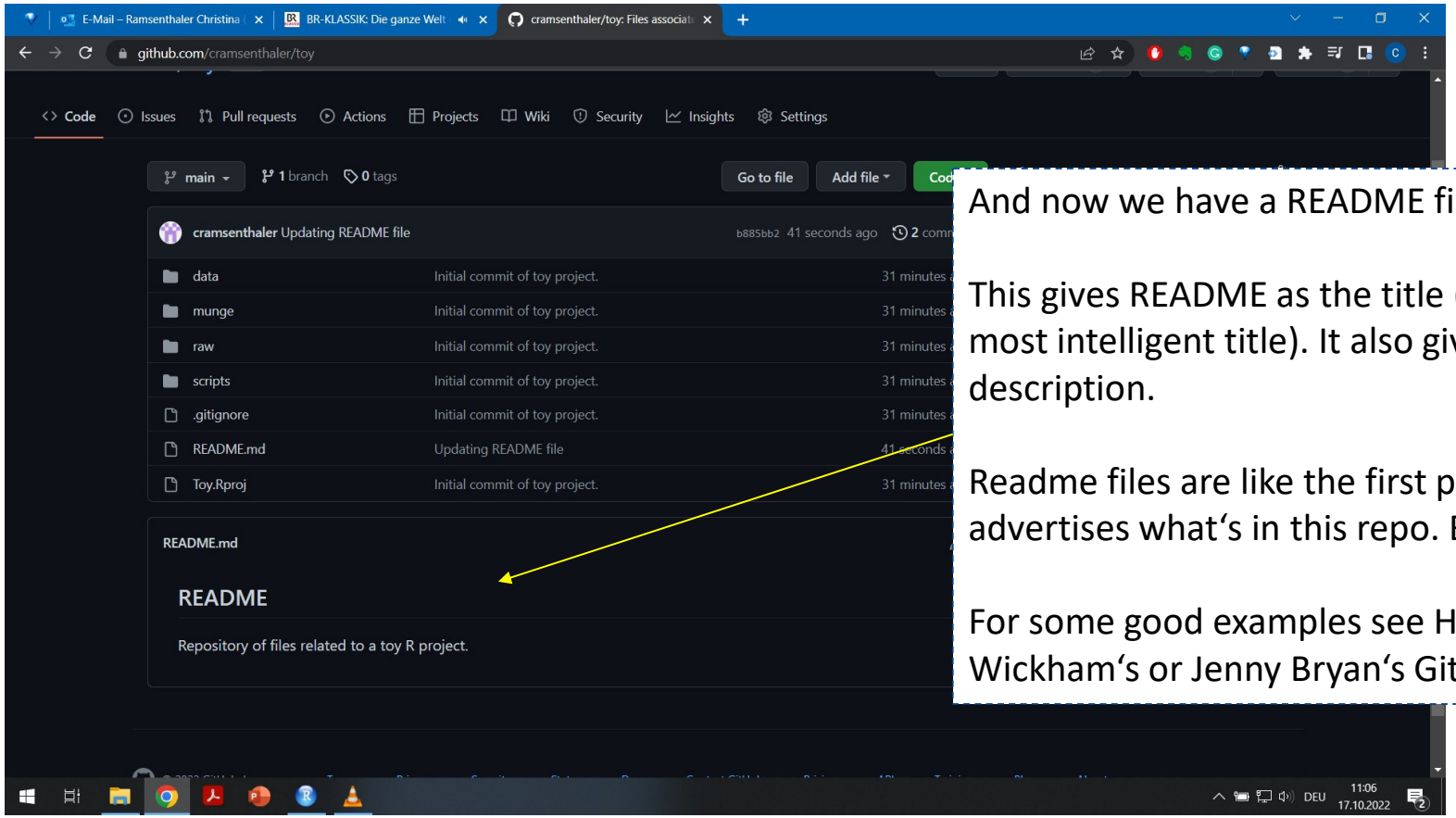
5. Setting up the README file



The new file will immediately show up on the Git tab. You can commit this with a short commit message.

Remember to push this immediately to GitHub for changes to appear there.

5. Setting up the README file



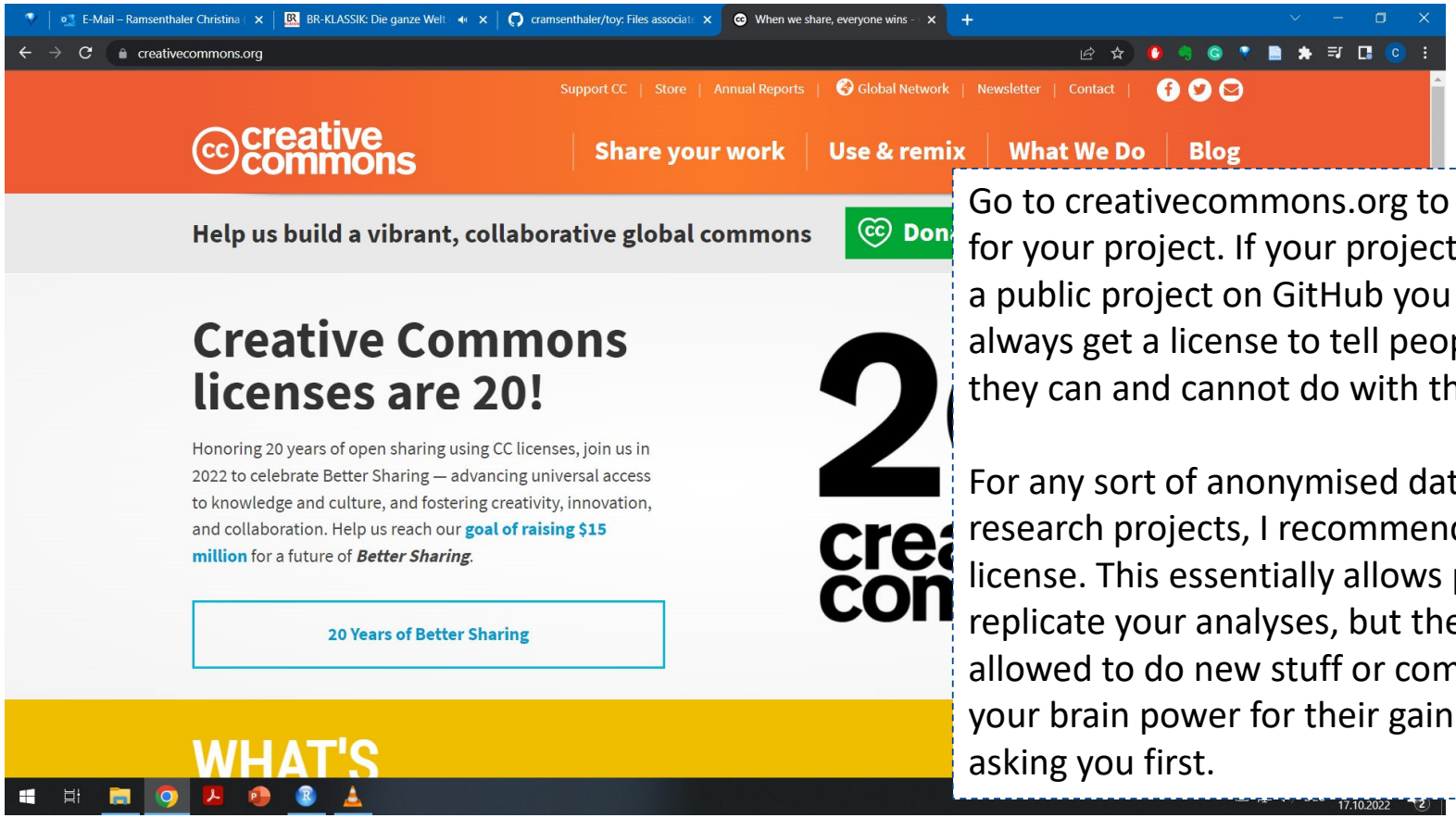
And now we have a README file.

This gives README as the title (not the most intelligent title). It also gives a short description.

Readme files are like the first page that advertises what's in this repo. Be creative.

For some good examples see Hadley Wickham's or Jenny Bryan's GitHubs.

6. Setting the license



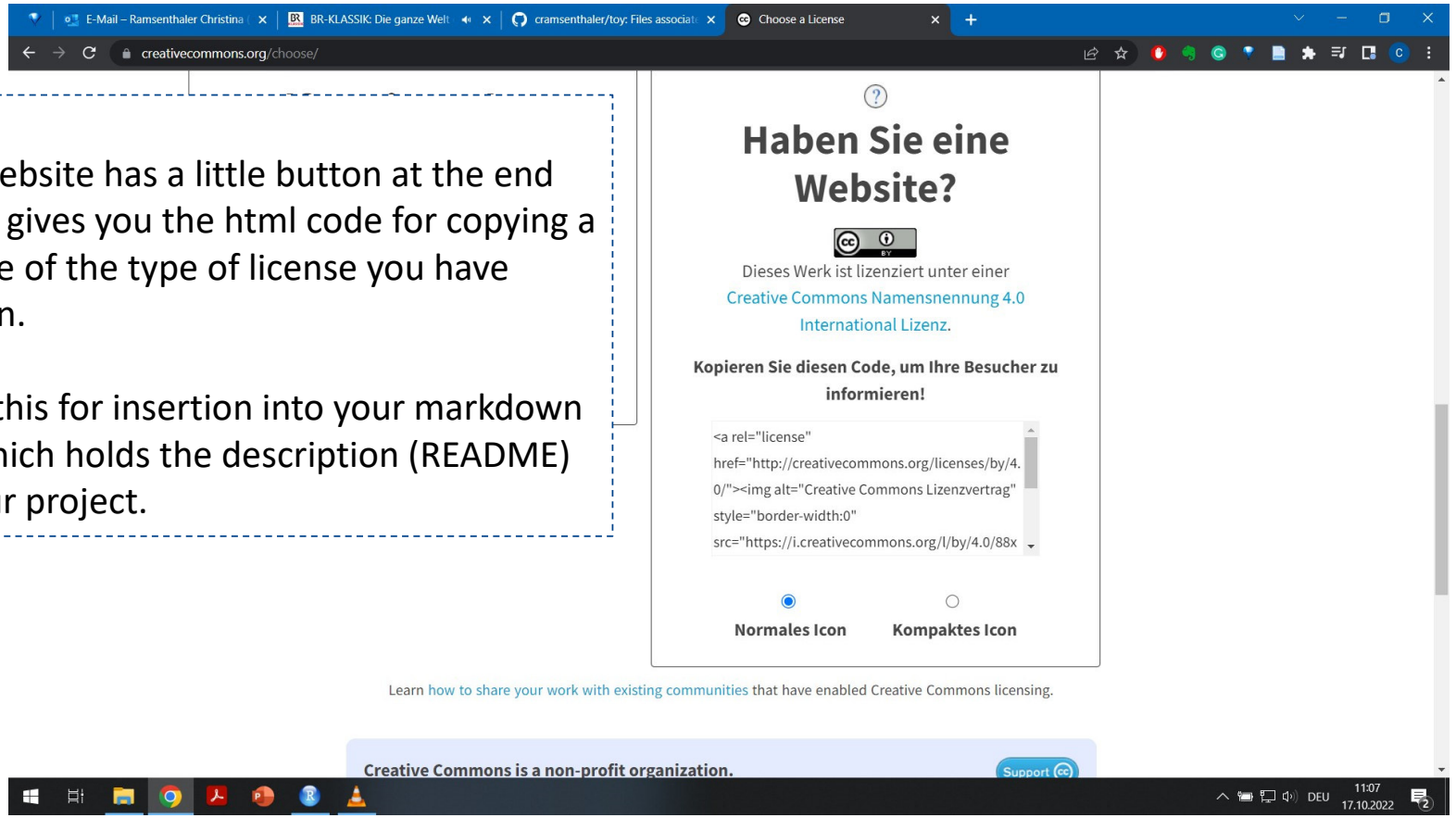
Go to creativecommons.org to get a license for your project. If your project is hosted as a public project on GitHub you should always get a license to tell people what they can and cannot do with this.

For any sort of anonymised data within research projects, I recommend a CC04 license. This essentially allows people to replicate your analyses, but they aren‘t allowed to do new stuff or commercialise your brain power for their gain without asking you first.

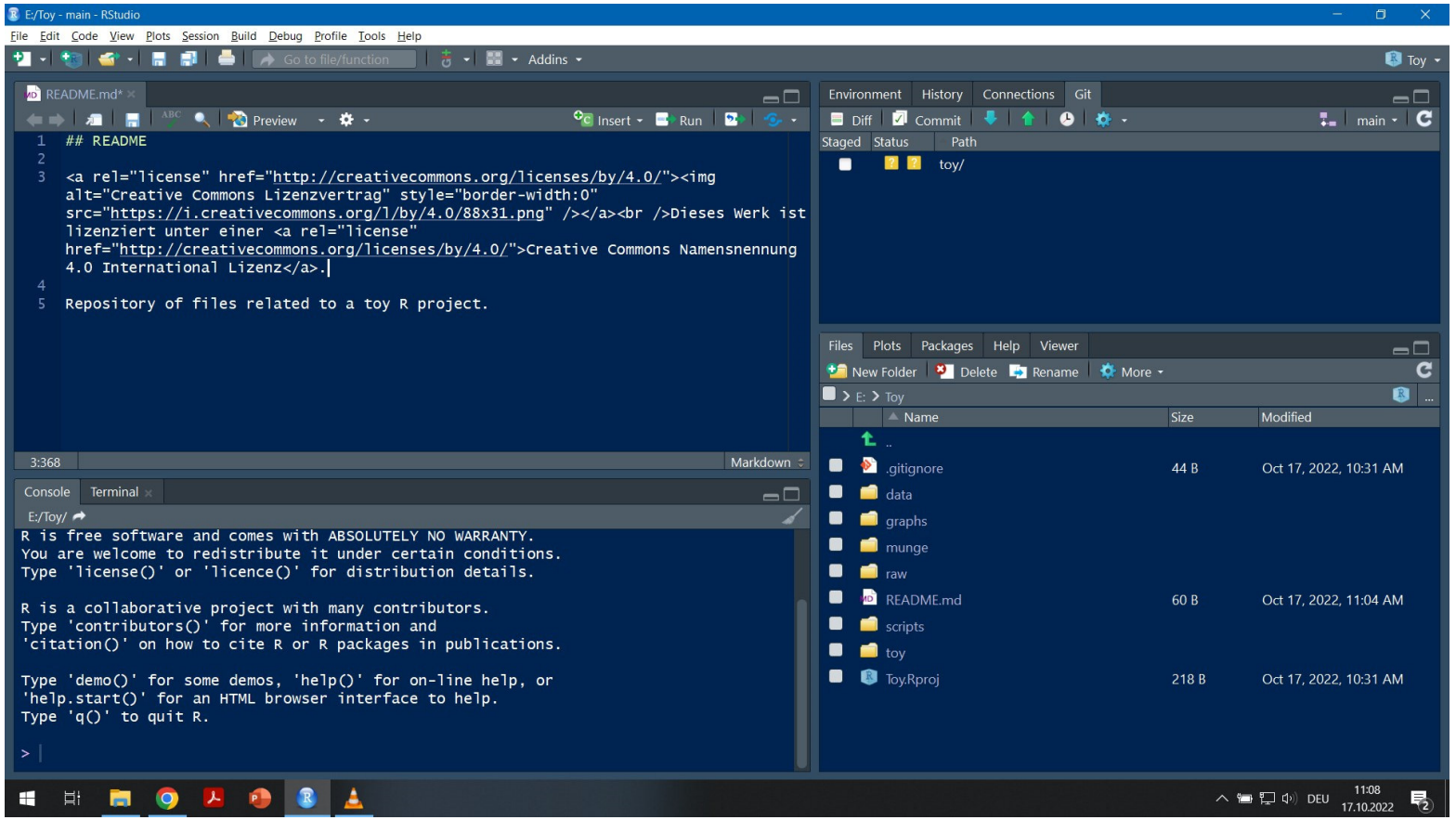
6. Setting the license

The website has a little button at the end which gives you the html code for copying a picture of the type of license you have chosen.

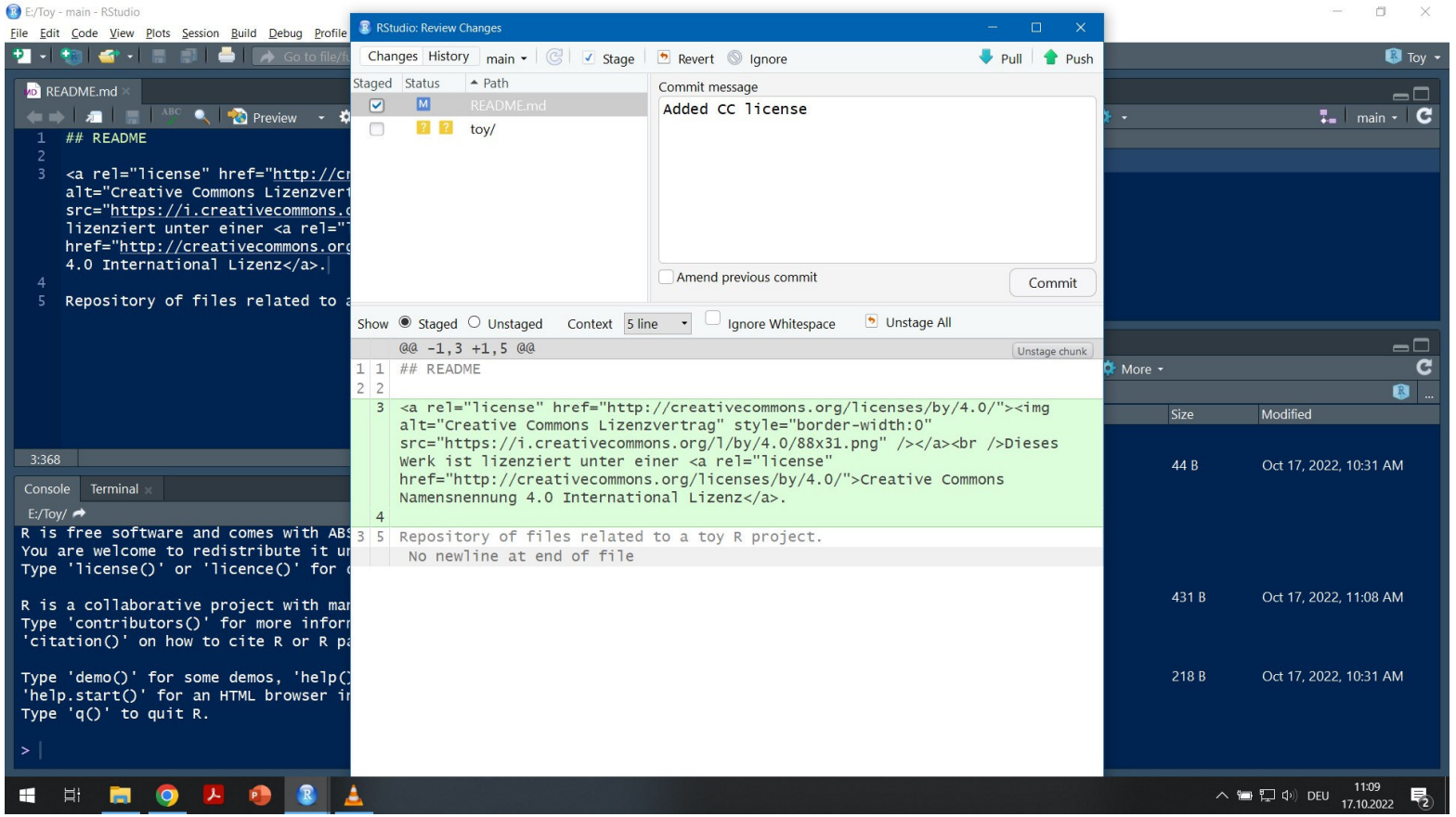
Copy this for insertion into your markdown file which holds the description (README) of your project.



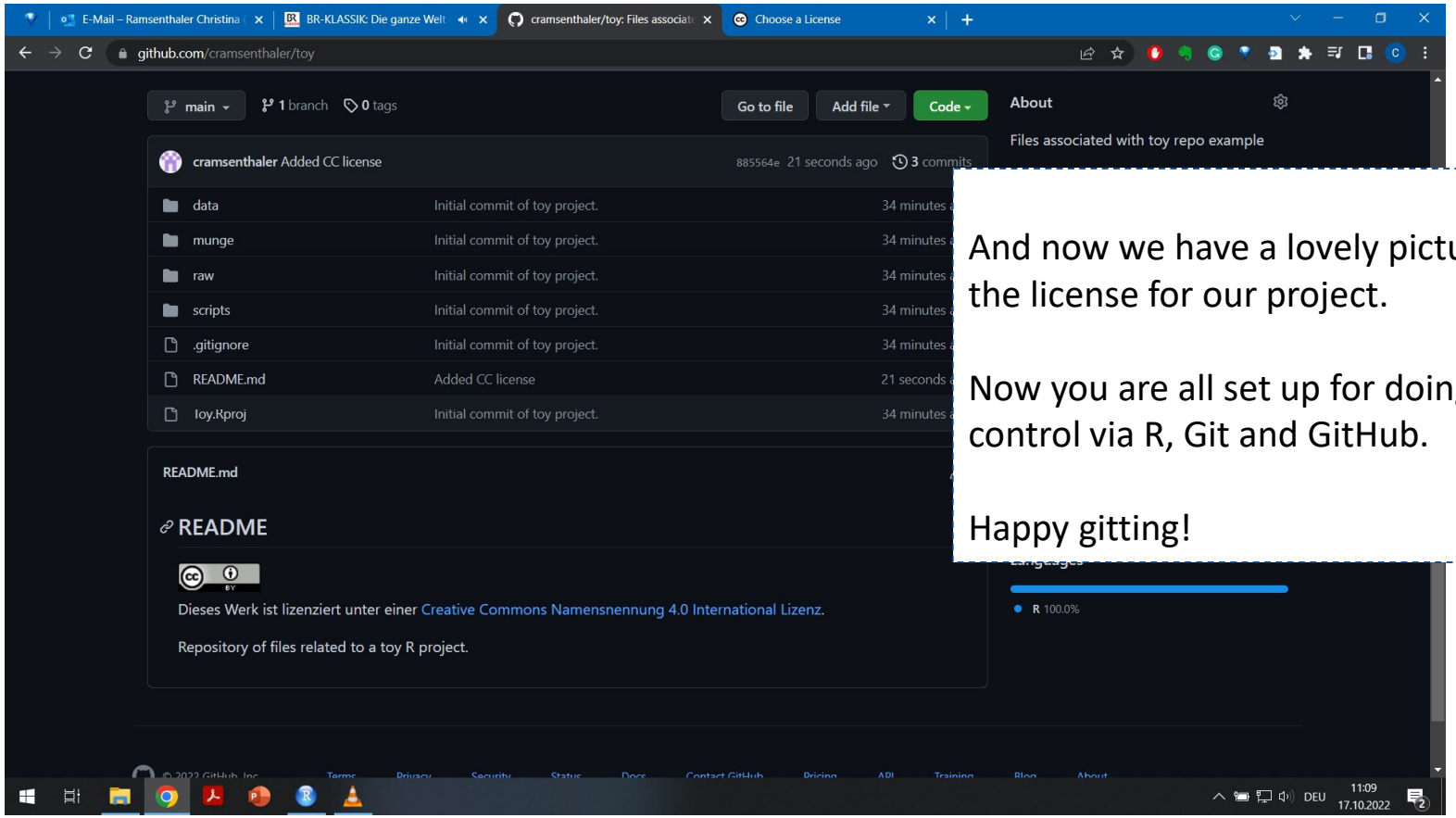
6. Setting the license



6. Setting the license



6. Setting the license

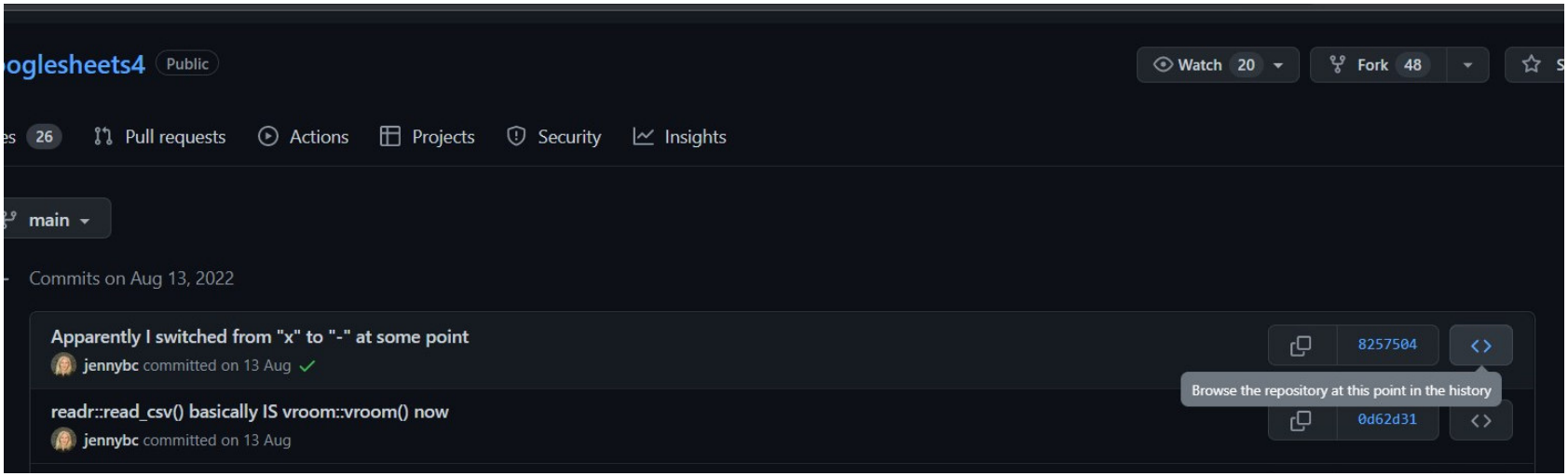


And now we have a lovely picture showing the license for our project.

Now you are all set up for doing version control via R, Git and GitHub.

Happy gitting!

What‘s the purpose of all this?



Dr R – what‘s the purpose of all this?

Well, the purpose is to go back any time to a prior version of your project without the need to saving files under a new name every time. On your GitHub repo, you can browse the complete history of your files associated with this project. You can always go back to these snapshots and restore stuff once things have started to go awry.

How to use version control

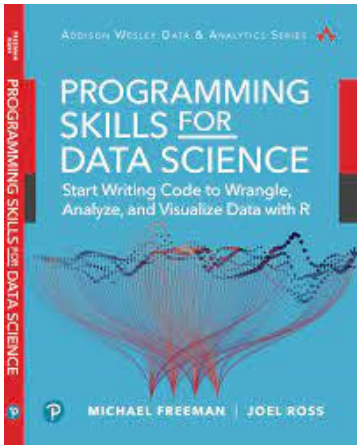
- Remember that committing is not the same as pushing (the midwives in my department would second that statement…).
  - How often should you push to GitHub? Well… that depends. I often do not commit every change. However, I tend to commit once a day when I have worked on that project.
  - When you work on different machines, you want to commit & push each time you finished working on a specific laptop or device.
- The moment you collaborate with colleagues and a project is „hot“ (aka worked on), it might be advisable to commit and push often. Merge conflicts are more easily detected when you don‘t wait till the end of the day.
  - There may be a specific office policy/guideline around how often to commit and push (I have worked in environments where there was one).
- Updating R and RStudio can become a major headache when it breaks the link between Git and RStudio. You might have to run through setup again. Most often, this is due to a bad config file. Consider storing your config file outside of your Windows directory.
- It pays off to understand config files and how config works in RStudio once you work with different GitHub accounts (I work with 3-4, 1 personal & 3 corporate ones).

Help!

- Version control via Git is the nerdiest of nerdy topics. It‘s a world of its own.
- There are a couple of great resources that have helped me:

- Jenny Bryan‘s brilliant bookdown „Happy Git with R“: https://happygitwithr.com/
- The whole documentation on GitHub is brilliant, particularly for folks who want to host webpages via GitHub.
- Michael Freeman’s Programming Skills for Data Science
  - Has several lovely chapters on working with Git
- Several mitp or Hanser books on Git (“Git kurz und gut”)

- Easiest would be to find a person already using version control and apprentice to them



The end!

- Thank you for a wonderful workshop and for your enthusiasm.
- Thank you to the team from FRS for hosting the workshop.

- Please stay in touch. You may always use my email ramsenthalerchristina[a]gmail.com for contacting me.