R Datavisualization 101

Introduction for Term Papers and BA Thesis

Nick Glättli

January 26, 2022

Contents

1 Introduction

The R programing language allows us to do professional data visualziation with very little effort. Since R is not a statistics program in the narrow sense but a programming language, we can apply an almost limitless level of customization to our plots.

In this introduction to datavisualization, I will provide the 101 of R datavisualization for students and starters I mainly focused on how to customize charts. While I cover here a very basic form of R programming, a certain basic level of R programming skills are required, when applying. Particularly data manipulation, which is not covered here, as it sets the groundwork for good data visualization.

Feedback is always welcome, please feel free to send an email to nick.glaettli@gmail.com

1.1 What options do we have?

The R programming language offers many options to display your data. R itself has built in functions to visualize data. Such plots are easy to program and are very useful for work-in-progress visualization. However, when having to display data for term papers or publishing, base plots are not well suited. Particularly since plot customization is rather time and code intensive. Furthermore are base plot inferior in terms of readability and general looks. Hence, R base plot are not the tool of choice for datavisualization, which is why we use ggplot2.

GGplot2 is the most important datavisualization package in the R programming language and is included in the tidyverse package. It has become the state of the art tool to create any plots. Not only has it proven to be superior in simplicity of coding but also in terms of easy customization. GGplot2 will be the main package used here, as almost all plot can be made with this tool. Install and load it via the tidyverse package or ggplot2 itself as follows:

```
#Install via tidyverse
if (!require(tidyverse)) install.packages('tidyverse')
library(tidyverse)

#Install directly
if (!require(ggplot2)) install.packages('ggplot2')
library(ggplot2)
```

There are many further packages that build on ggplot2 and cover the few flaws that the base package has. However, in this short guide, I decided to use additional packages only when absolutely necessary, as they often are helpful only when having reached a more sophisticated level of ggplot2 coding.

1.2 Data and replicability

This short guide was designed so that any code can be copied and is fully functional for all versions of R, that support the packages used. R comes with built in datasets that are well suited to show the basics of R data visualization. Call `data()` to see a full list of all available datasets. In this guide, I mainly used the iris and mtcars data.

2 Basics

2.1 Code

The ggplot2 package provides a very user-friendly way of composing plots. First, we have to call the `ggplot()` function, which tells R that it should make a plot with ggplot2. The brackets of the `ggplot()` function can be filled with information that is used for all layers built later. This can prove to be very effective when using the same data for all layers. When this is the case, we first specify the dataframe from which we want to visualize data. Using the `aes()` argument, the data for each axis can be further

specified. While it is possible to specify more information in the ggplot()‘, I would not recommend to do so, as it can interfere with later added layers. Further information can be added by using the “+” operator.

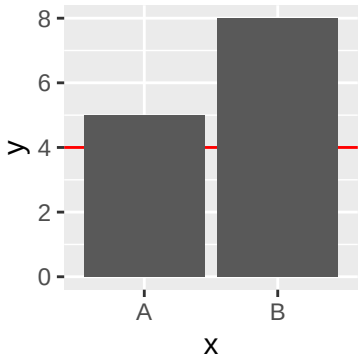
```
ggplot(mtcars, aes(x= mpg, y= hp))+
  geom_point()
```

Geoms build plot types. In the case above, that would be a scatterplot. When not setting the data for all later used commands, the information is specified within the function brackets of the geoms. This allows us to create plots, different data.

2.2 Layers

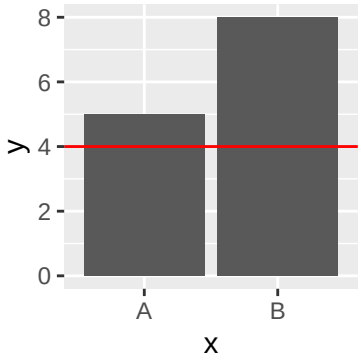
It is crucial to understand that ggplot2 operates layer based. It adds new layers over the existing plot in the order of your code. I will demonstrate this principal by plotting a simple barplot with a horizontal line.

```
#Create data
#Use tibble alternatively: data <- tibble(x,y)
x <- c("A","B")
y <- c(5,8)
data <- data.frame(x,y)


#Plot data
ggplot(data, aes(x= x, y= y))+
  geom_hline(yintercept= 4, color= "red")+
  geom_bar(stat= "identity")
```



Since we told ggplot to build the red line first and then the barplot, the line is now behind the bars. In this case, however, this is not particularly reasonable. To fix this, the code has to be changed slightly. Instead of building the line first and then laying the plot layer over it, the plots have to be built first and the line drawn afterwards. Hence, the `geom_hline` argument (code for horizontal line) has to come after `geom_bar` (code for barplot).

```
ggplot(data, aes(x= x, y= y))+
  geom_bar(stat= "identity")+
  geom_hline(yintercept= 4, color= "red")
```



The red line now lies over the two bars, since the barplots were built first and the line put over the plot. This simple fact about ggplot's system of operating do we have to keep in mind when coding our plots. Every new line of code builds a new layer upon the previous. Often, in a workflow, we do cosmetics as the very last step. Thus it is even more important to keep this in mind. While sometimes, as in the example above, it is favorable to put such cosmetics above the base plot, other times however, such layering looks odd.

3 Plots

As ggplot2 holds almost limitless amounts of geoms that can be combined to create new plots, it is an impossible task. In consequence, I will give here only a brief overview. If you want plotted examples, I recommend the book «ggplot2: elegant graphics for data analysis» by Hadley Wickham, Danielle Navarro, Thomas Lin Pedersen. You can access the book online: https://ggplot2-book.org/

3.1 Plot types

-`geom_boxplot()` creates a boxplot

-`geom_histogram()` creates a histogram

-`geom_density()` creates density plot

-`geom_violin()` creates violin plot

-`geom_errorbar()` creates an errorbar

-`geom_pointrange()` creates a point plot with line range

-`geom_linerange()`creates linrange plot

-`geom_smooth()` fits regression line into data plot

-`geom_bar()` creates barplot

-`geom_line()` creates lineplot

-`geom_point()` creates scatterplot

3.2 Reorder data

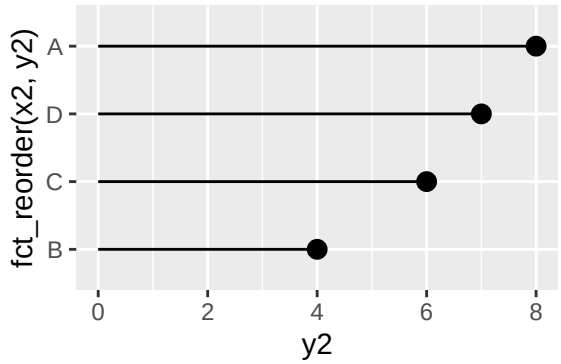
Readability is a crucial criterion for any plot. General good practice in data visualization is to order the data in ascending or descending order. Reordering data can be done in many ways. Unfortunately, ggplot2 itself does not have a function to reorder data. I will show you two options to reorder the data your plots.

1. The tidyverse package also includes the forcats package. To reorder by value, use the `fct_reorder()` function. The first argument tells which value should be reordered, while

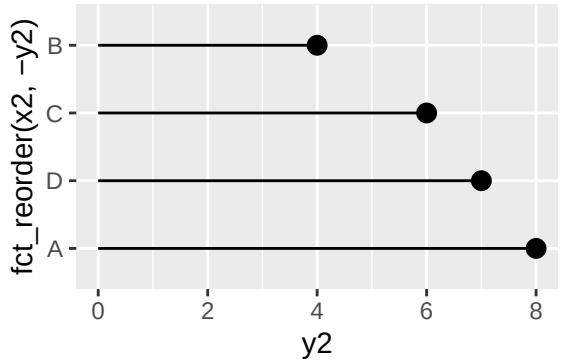
the second argument defines by which values the reordering takes place. By default the function orders the data ascending. To change it to descending, either add the `-` symbol or add `desc()`. The advantage of the forcats package is that it also comes with other reorder options. It is also possible to reorder by the median, adding `.fun="median"` as a third argument. Using the `fc_relevel()` function, it is also possible to define a custom order.

2. Base R also provides a function to reorder data: the `reorder()` function. It works as well as the `fct_reorder()` and also orders by default ascending.

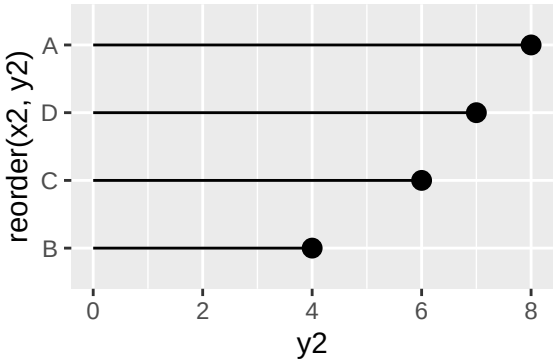
```
#create data
x2 <- c("A","B","C","D")
y2 <- c(8,4,6,7)
data2 <- tibble(x2,y2)

#Reorder Using forcats
#ascending
ggplot(data2, aes(x= fct_reorder(x2, y2),y= y2))+
  geom_segment(aes(xend= x2, yend= 0))+
  geom_point(size=3)+
  coord_flip()
```



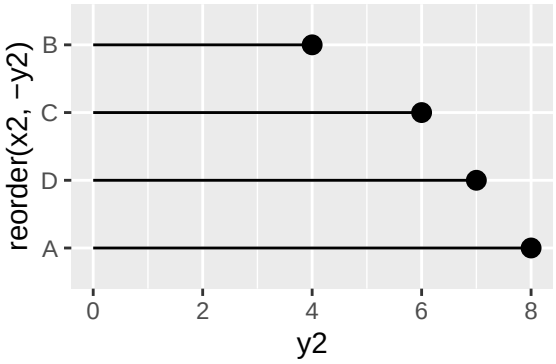

```
#descending
ggplot(data2, aes(x= fct_reorder(x2, -y2),y= y2))+
  geom_segment(aes(xend= x2, yend= 0))+
  geom_point(size=3)+
  coord_flip()
```




```
#Reorder using base R
#ascending
ggplot(data2, aes(x= reorder(x2, y2),y= y2))+
  geom_segment(aes(xend= x2, yend= 0))+
  geom_point(size=3)+
  coord_flip()
```




```
#descending
ggplot(data2, aes(x= reorder(x2, -y2),y= y2))+
  geom_segment(aes(xend= x2, yend= 0))+
  geom_point(size=3)+
  coord_flip()
```



3.3 Common errors

I want to briefly cover two very common errors made by beginners when starting to create their first plots with ggplot2. Usually, these mistakes come from the complexity that lies behind some seemingly easy plot functions. Fortunately it is easy to understand where the flaw lies and how to fix it.

3.3.1 Barplots

It is very common that the first struggles one has with ggplot2, happens when creating barplots. The function to call for barplots is `geom_bar()`. Going back to the simple example data made in subchapter «Layers», most beginners would simply call the `geom_bar()` function and receive the following error.

```
#Falsely coded barplot
ggplot(data, aes(x = x, y = y))+
  geom_bar()
```

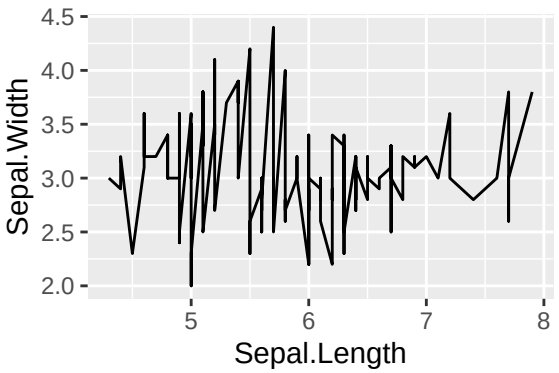
```
## Error: stat_count() can only have an x or y aesthetic.
```

The error message tells us that we can only have a x or y argument, when not further specifying how ggplot2 should aggregate data. By default, the `geom_bar()` counts how often the argument specified on the x- or y-axis, occurs in the data. However, in this case, we do not want to count frequency, as we already assigned values. Consequently, the `geom_bar()` function needs further specification on how to count values. This is why we include the following argument: `stat = "identity"`. When having specified this argument, ggplot2 no longer uses the default `stat_count()` but plots the x- and y-axis according to our dataframe. Another possibility to solve this issue is to use the `geom_col()` function instead.

3.3.2 Line plots

Another common mistake occurs when plotting the mean as a line plot. In many cases, the code and the resulting plot look like this.

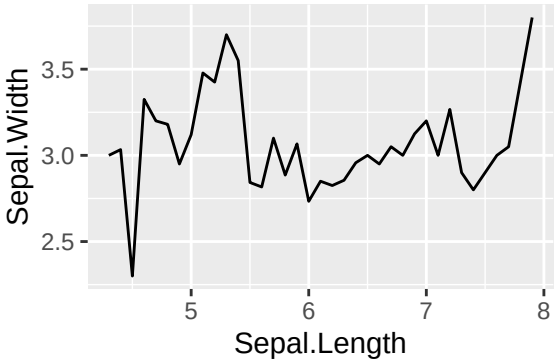
```
#Falsely coded lineplot
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_line()
```



Beginners often misunderstand the functionality of the `geom_line()` function and intuitively assume that it is able to plot the mean. However, this is not the case. The `geom_line()` function (as well as `geom_path()`) connects data points, independently from the data structure. This does not pose any problem when non of the data points overlap on the x-axis. When they do overlap, ggplot2 connects all points, resulting in a straight line. This was what caused the odd look of the plot above. In such a case, we are probably interested in plotting the mean value as a line. There are two possible workarounds to solve this issue.

1. Using base R and/or the dplyr package, the dataframe can be manually manipulated, so that it no longer represents single data points, but the mean value for each corresponding x-axis value. This, however, is code intensive and may result in further issues.
2. Fortunately, ggplot2 offers a very simple workaround: the `stat_summary()` function, which can be used instead of the `geom_line()` function. It allows to summarize your data and define a geom type to be built. First, it has to be defined which data should be summarized in what way. Second, the geom type has to be specified.

```
#Fixed lineplot
ggplot(iris, aes(x= Sepal.Length, y= Sepal.Width))+
  stat_summary(fun = "mean", geom = "line")
```



4 Themes

Themes are the best way to customize your plot and make it not only look tidy but also personalized. The ggplot2 package comes with eight built in themes that you can easily trigger by adding the

`theme_themename()` function. The following options does ggplot offer you:

-`theme_gray()`

-`theme_bw()`

-`theme_classic()`

-`theme_minimal()`

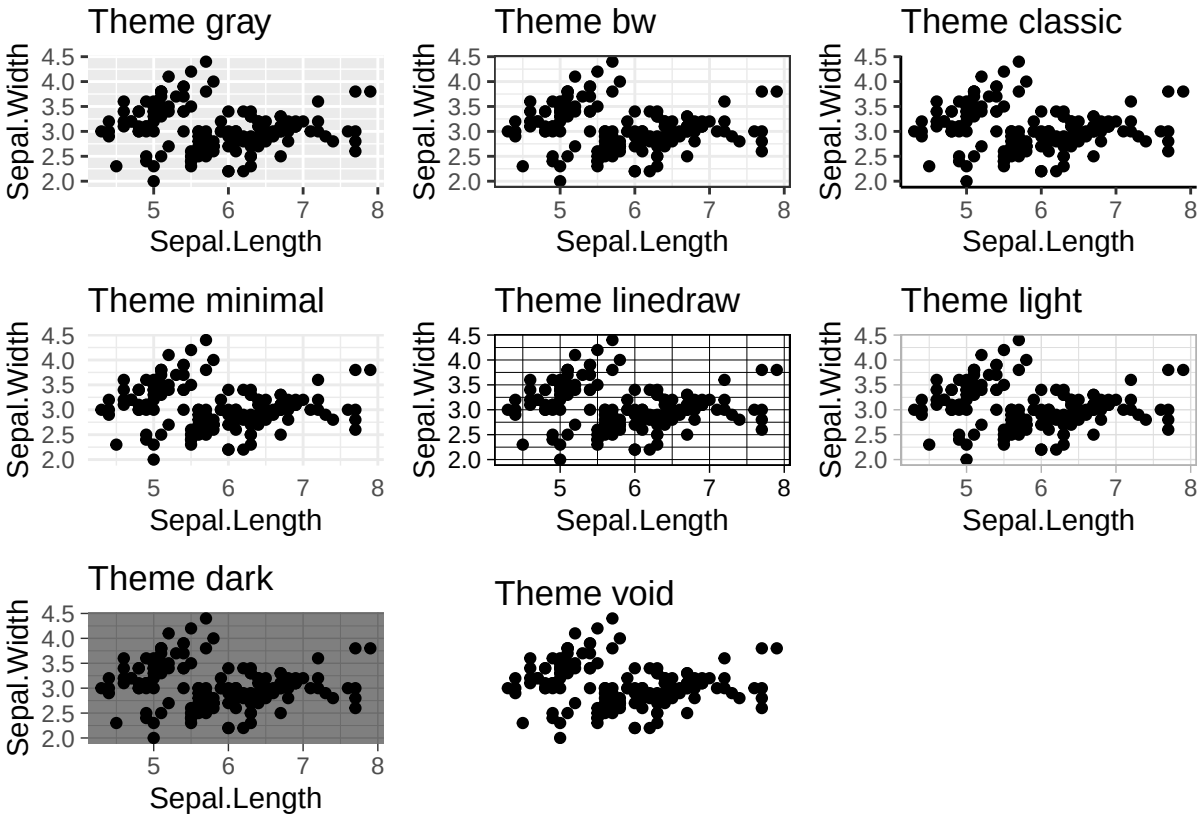
-`theme_linedraw()`

-`theme_light()`

-`theme_dark()`

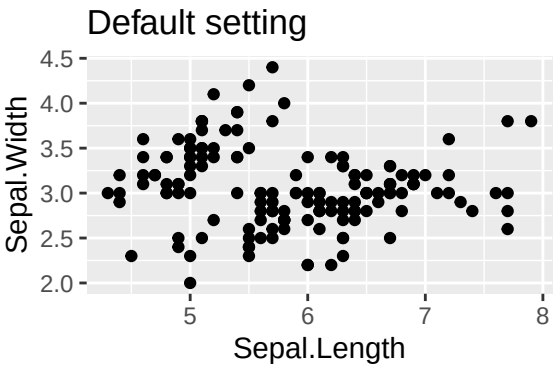
-`theme_void()`

To see the difference between each theme, I created a simple plot and applied each theme.

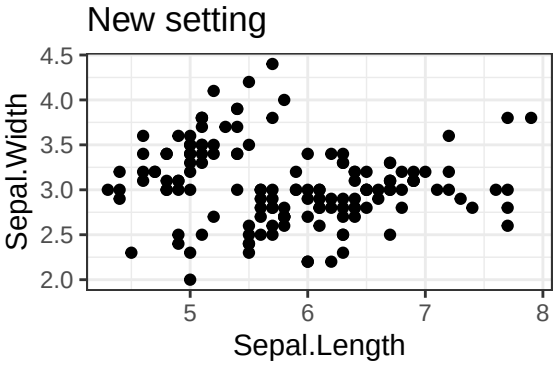


The standard theme that ggplot2 uses is `theme_gray()`. However, if you prefer another theme, you can change this setting easily by calling `set_theme()`. I demonstrate this by setting the default theme to `theme_bw()`.

```
#Default Plot
ggplot(iris, aes(x= Sepal.Length, y= Sepal.Width))+
  geom_point()+
  ggtitle("Default setting")
```




```
#Change setting
theme_set(theme_bw())

#Same Plot with changed default theme
ggplot(iris, aes(x= Sepal.Length, y= Sepal.Width))+
  geom_point()+
  ggtitle("New setting")
```



Furthermore, there are many additional themes that are provided by compatible packages. I particularly recommend my ggthemepark package or the ggtheme package. The ggthemepark package can be installed via Github.

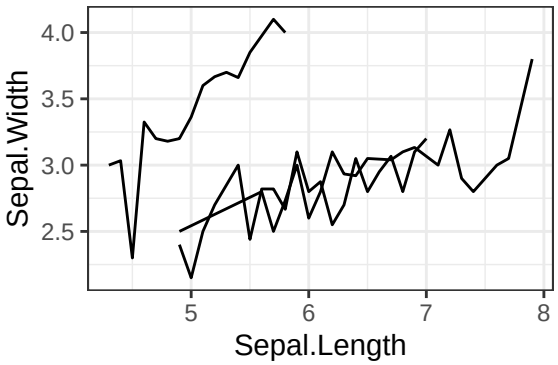
```
library(devtools)
devtools:: install_github("NickGlaettli/ggthemepark")
```

5 Grouping

Often, simply displaying the aggregated data, is not sufficient to gain substantive knowledge from plots. To make them more informative, there is the option of visualizing information about subgroups. Therefore the data used has to be grouped. Using ggplot2, there are various options to do so: Groups can be distinguished by color, or appearance.

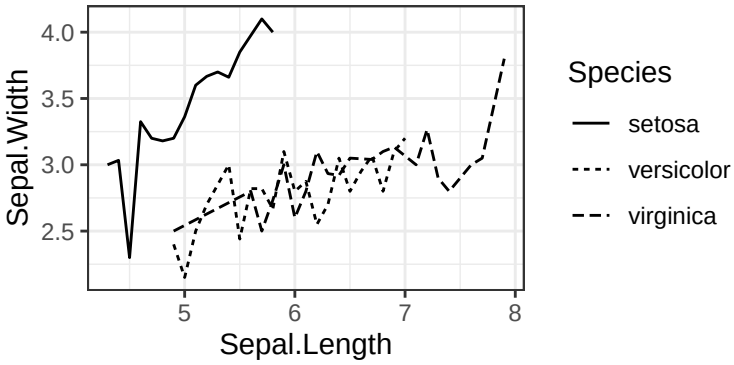
The most simple way is, to specify groups in the `aes()` argument by inserting `group =`. All groupings have to take place inside `aes()`. Adjustments for all groups have to take place outside of this argument.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, group = Species))+
  stat_summary(fun = "mean", geom = "line")
```

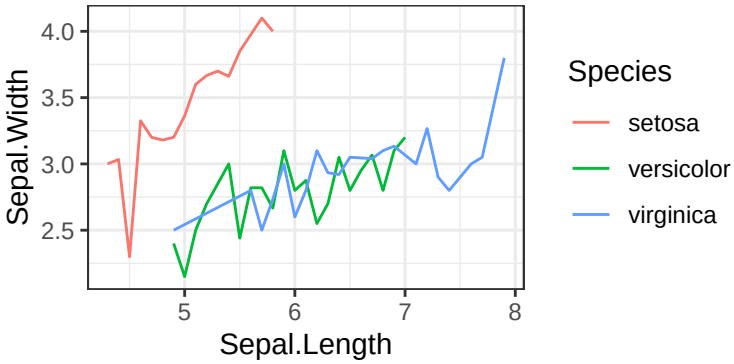


In the plot we can see three lines, representing the values of three species. However, it is still hard to read, as the lines cannot be distinguished. For better readability, there are two main option, that can be used interchangeably or complementary. To distinguish the lines, the linetypes and/or colors can be changed, as you can see in the following code block.

```
#Linetype
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, linetype = Species))+
  stat_summary(fun = "mean", geom = "line")
```




```
#color
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species))+
  stat_summary(fun = "mean", geom = "line")
```



There are six different linetypes:

-dashed

-twodash

-longdash

-dotdash

-dotted

-solid

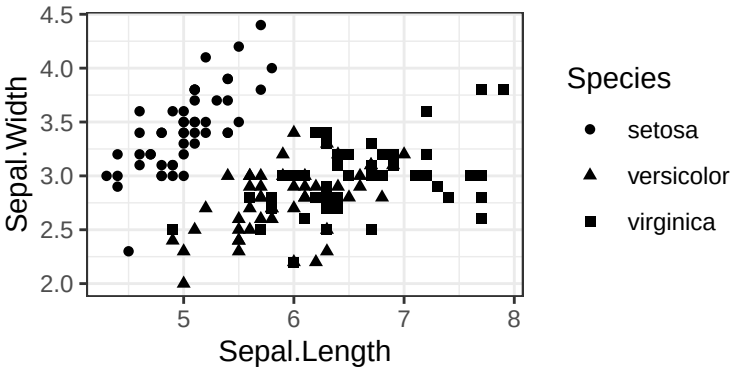
When grouping with linetypes, ggplot2 automatically assigns linetypes. When you are not satisfied with the result, the linetypes can be assigned manually. If all lines should look the same but with a different linetype, the linetype can be specified in the geom function. If it is wished to keep the grouping, then a new function has to be added to the code: `scale_linetype_manual()`

```
#All lines look the same
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  stat_summary(fun = "mean", geom = "line", aes(group= Species), linetype= "longdash")

#Manually assigned
#Option 1
types <- c("dashed","longdash","dotted")
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  stat_summary(fun = "mean", geom = "line", aes(linetype = Species))+
  scale_linetype_manual(values = types)

#Option2
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  stat_summary(fun = "mean", geom = "line", aes(linetype = Species))+
  scale_linetype_manual(values = c("dashed","longdash","dotted"))
```

The same applies to grouping with colors. However, there are too many colors to list them all here. A list can be obtained by running `colors()`. You also have to option to use colors from packages (covered later) or set custom hex-codes or rgb-codes. To set colors manually, you use the `scale_color_manual()` function. For barplots, boxplots etc, the code to group by color differs. When setting `color=` outside the aesthetics, it only changes the border. To change the color of the full geom, the argument `fill=` is used. This is also the argument used to group by color, inside `aes()`.

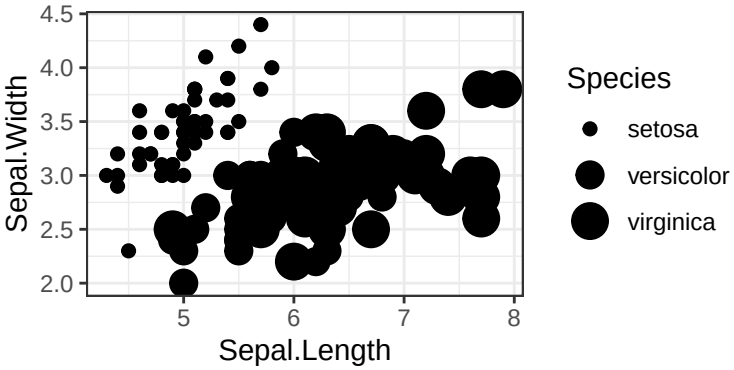
Scatterplots can be grouped by color, of course, but also by shape. To do so, the argument `shape=` has to be called, and used as disucessed above. Another option here, is to group by shape, though in this example not necessarily reasonable, as ggplot2 also nicely warns about in the console.

```
#Grouped by shape
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(shape=Species))
```




```
#Grouped by size
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(size=Species))
```

```
## Warning: Using size for a discrete variable is not advised.
```



As mentioned, can these grouping options be used complementary and for all kinds of plots.

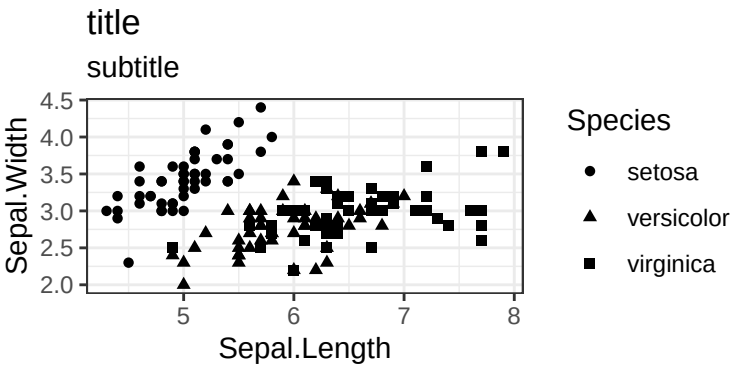
6 Labels

Labels are crucial to ensure high readability of your plots. In this chapter, I will cover how to adjust titles, captions, legends and other labels.

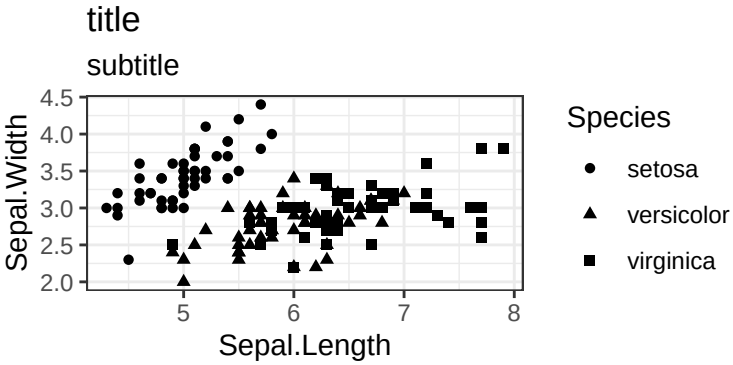
6.1 Titles

Plots can have two titles: title and subtitle. There are two options to add title and subtitle, using the `ggtitle()` or `labs()` function. The functionality of `ggtitle()` is fairly simple. It can contain two arguments, of which the first is the title and the second subtitle. Using `labs()`, on the other hand, is slightly code heavier. Since this function is designed to adjust all sorts labs, first, it has to be specified which kind of lab should be amended. Straight forward the argument for title and subtitle is `title=` and `subtitle=`.

```
#Using ggtitle
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(shape=Species))+
  ggtitle("title","subtitle")
```



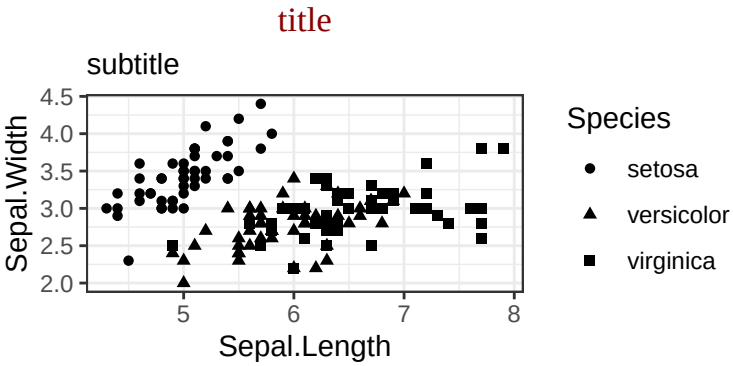

```
#Using labs
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(shape = Species))+
 labs(title = "title", subtitle = "subtitle")
```



Generally, I recommend to use `labs()` over `ggtitle()`, since all labels around the plot can be adjusted with `labs()`, which makes your code much more readable and easy to check for flaws. `ggtitle()` should only be used when the plot code is very simple and no other labels are changed.

The position, as well as general appearance of title and subtitle, is defined in the theme. In order to change to the appearance, the theme has to be amended temporarily. While it is usually quite code intensive, such amendments of the theme can be easily done by setting changes with the `theme()` function. I want to illustrate how to do it by repositioning the title in the plot center and change font, as well as color.

```
#Using ggtitle
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(shape = Species))+
  labs(title = "title", subtitle = "subtitle")+
  theme(
    plot.title = element_text(
      hjust = 0.5,
      family = "serif",
      color = "darkred"))
```



Of course, the title can also be adjusted on the y-axis, using the `vjust=` argument.

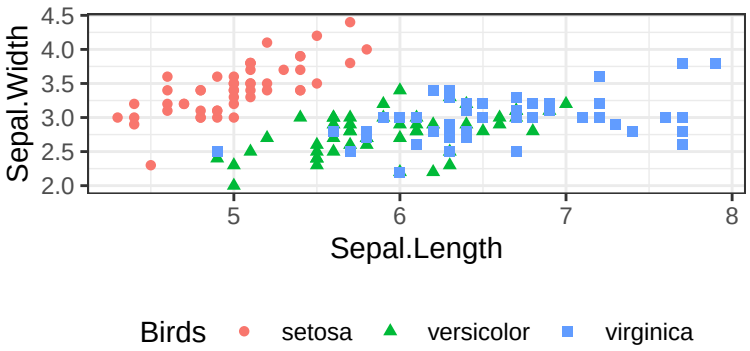
6.2 Legend

By default, the theme is set to display the legend at the right hand side. Hence, to adjust the aesthetics of the legend, the theme has to be amended again. By adding `legend.position =`, the legend can be repositioned. The base options are right, left, top, bottom. To position it at a distinctive position, create a vector for the x- and y-axis value. To remove the legend simply call `legend.position = "none"`. When having legends, the title often has to be changed as it is the variable name by default. There are many ways to do so and which one is best depends on the circumstances.

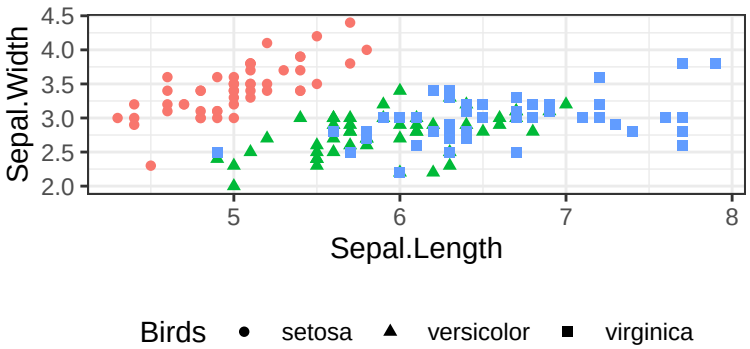
1. When only changing the legend title, it is mostly advised to use the `labs()` function. Inside, the argument used for grouping can be called and renamed. Note that whenever more than one type of grouping was applied, they all have to be renamed. Otherwise, ggplot2 sees them as two different legends

2. When applying further changes to the legend, such as reordering the legends or excluding one, it is preferred to use `guides()` instead. To renmame the function `guide_legend` has to be further added.

```
#Using labs()
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(shape=Species, color=Species))+
  labs(color="Birds", shape= "Birds")+
  theme(legend.position = "bottom")
```



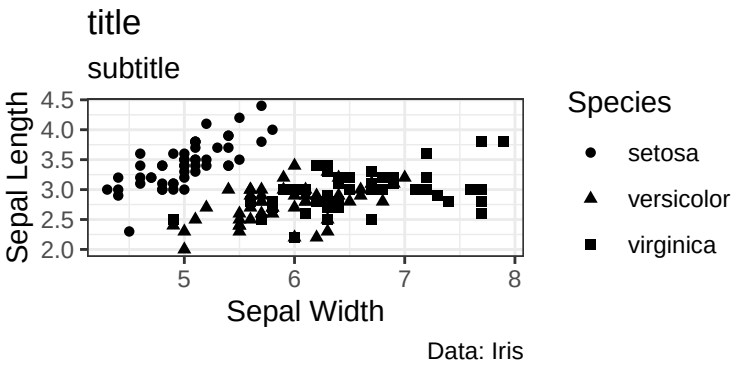

```
#Using guides()
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(shape=Species, color=Species))+
  guides(color= "none", shape= guide_legend("Birds"))+
  theme(legend.position = "bottom")
```



6.3 Axis labels and captions

As mentioned, the `labs()` function is very powerful to adjust all sorts of labels The axis labels can be adjusted in the same manner as title and subtitle by simply specifying `x=` and `y=`. Furthermore, ggplot2 offers the option of adding a caption below the plot, where notes or data sources can be added. To add a caption simply call `caption=`. In the code below I removed the dot from the axis labels and added a caption.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(shape=Species))+
  labs(title = "title", subtitle= "subtitle",
       x = "Sepal Width", y = "Sepal Length",
       caption = "Data: Iris")
```



To remove any axis labs you can either actively set no value in `labs()` or adjust the theme as following. Adjusting themes will be further discussed in this guide.

```
#Remove labels in labs()
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(shape=Species))+
  labs(title = "title", subtitle= "subtitle",
       x = "", y = "",
       caption = "Data: Iris")

#Remove labels in theme()
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(shape=Species))+
  labs(title = "title", subtitle= "subtitle",
  caption = "Data: Iris")+
  theme(
  plot.title = element_blank(),
  axis.title.x = element_blank(),
  axis.title.y = element_blank())
```

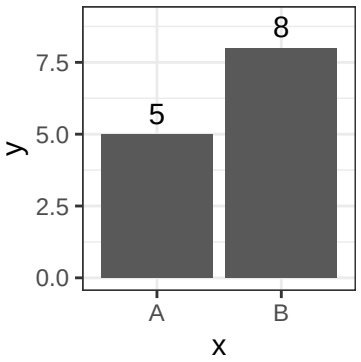
The adjustment of cosmetics and position works exactly as described in the section about titles.

6.4 Label data

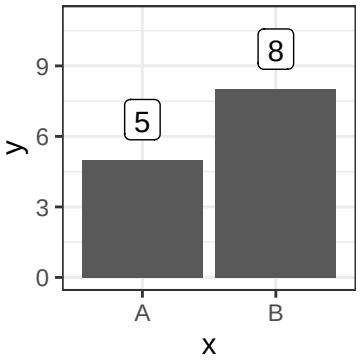
In contrary to the axis labels, data labels are actually geoms. This difference is crucial to understand, as since these labels are geoms, they are again affected by the layer based system of ggplot2. Sometimes it might reasonable to label all or just specific single values in our plot. Fortunately, ggplot2 makes it fairly easy to to do so. Using the simple barplot example again, the bars are labeled by using `geom_text()`. However, ggplot2 needs to know with what data it should label the bars. Hence, inside the `aes()` argument, it has to be specified that it should use the y-axis data. Just for better readability, the placement of the label is adjusted, so that it is not placed half into the bar, but above.

Another option is to use `geom_label()` instead. This geom works basically the same as described before, gives an aesthetically different output, however.

```
#Using geom_text
ggplot(data, aes(x = x, y = y))+
  geom_col()+
  ylim(0,9)+
  geom_text(aes(label= y), vjust = -0.5)
```




```
#Using geom_label
ggplot(data, aes(x = x, y = y))+
  geom_col()+
  ylim(0,11)+
  geom_label(aes(label= y), vjust = -0.5)
```

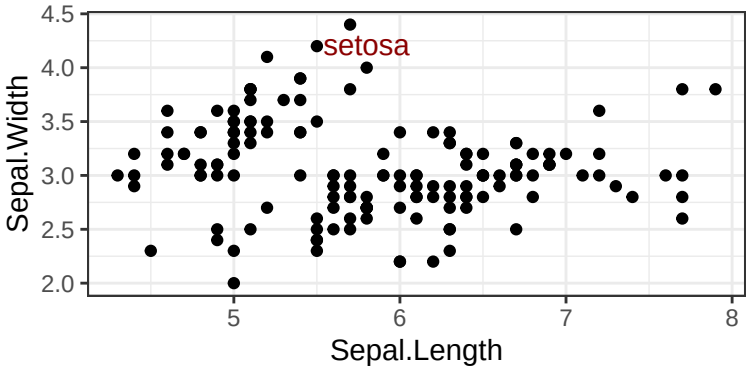


Particularly in situations where barplots have positive and negative values, it might be necessary to adjust positioning dependent on the data point value. In this case, the adjustment can either be defined in a list or by adding a simple condition.

```
ggplot(data, aes(x = x, y = y, ymax = 10))+
  geom_col()+
  geom_label(aes(label= y), vjust = ifelse(data$y > 0, -0.5, 1.5))
```

In some circumstances, it might be reasonable to just label a single data point. There are many ways to tackle this task. You could create a subgroup and specify this group in `aes()`. Also there is the possibility to make a conditional statement. When there is a data point that cannot be identified distinctively, then the position of the label can be chosen freely. I decided to demonstrate a conditional label, by specifying that when points where Sepal.Width equals four, the species should be labeled. In all other cases, the geom text is void.

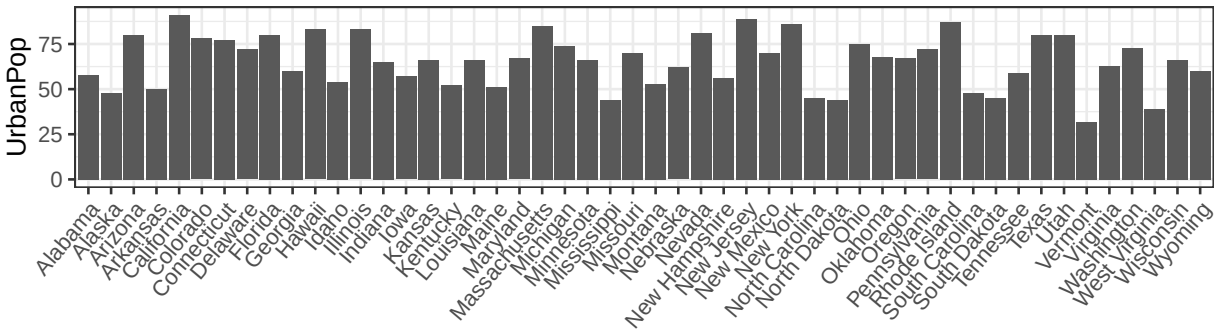
```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()+
  geom_text(aes(label= ifelse(Sepal.Width == 4, as.character(Species), "")),
            vjust = -0.6, color= "darkred")
```



7 Axis and Grid

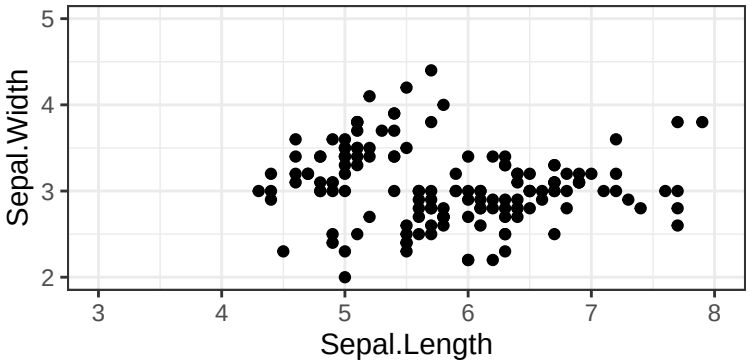
When the values of one axis are characters, it often happens that they overlap. To solve this issue, the angle has to be adjusted. This has to be done with the `theme()` function as well. It has to be specified which axis text should be changed and by which angle.

```
ggplot(USArrests, aes(y= UrbanPop, x= rownames(USArrests)))+
  geom_col()+
  theme(axis.text.x = element_text(angle = 50, hjust = 1),
        axis.title.x = element_blank())
```



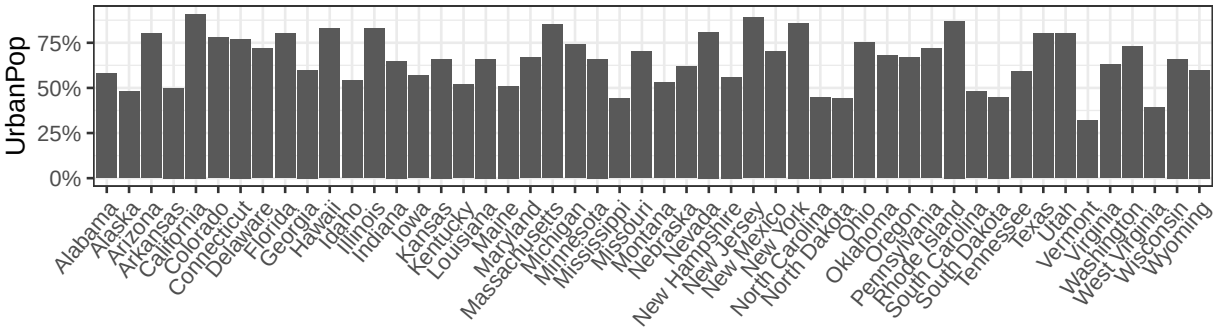
The ggplot2 package automatically sets the axis scales according to the data displayed. In order to make the charts more readable, the limits are set near the minimal and maximal value on each axis. This is usually the first thing beginners want to change. Maybe the interval is too narrow to properly see labeled data, or it is not fit in general. In this case, the axis scales can be adjusted by adding a new line of code. The axis interval can be defined with `xlim()` and `ylim()`. To illustrate, I expanded the y-axis to 5 and the x-axis to 3.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()+
  xlim(3,8)+ ylim(2,5)
```



GGplot2 automatically detects when one of the axis is in a time format and treats it accordingly. As percentage does not come in a distinctive format, the same convenience is not applied in these cases. However, there is a elegant workaround, so that you do not have to transform your data into character to display the percent sign. To amend scales, the function `scale_continous()` has to be called. In this function, the details about the scale are defined. The code `labels = scales::percent_format(scale=1)` transforms the selected scale into percentage. The `scale` argument defines by what factor the values should be multiplied before formatting.

```
ggplot(USArrests, aes(y= UrbanPop, x= rownames(USArrests)))+
  geom_col()+
  scale_y_continuous(labels = scales::percent_format(scale=1))+
  theme(axis.text.x = element_text(angle = 50, hjust = 1),
        axis.title.x = element_blank())
```



Also, the scale can be easily transformed in many other ways, by using the `trans` argument. Here a not complete list of the most important built in transformation arguments:

-log

-log10

-log2

-sqrt

-expt

-reverse

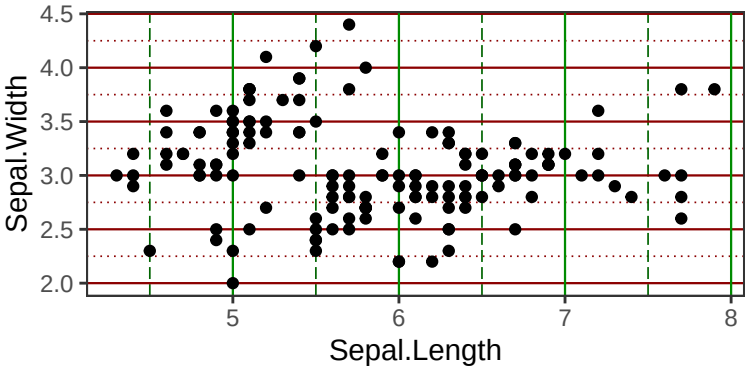
-time

-date

However, another possibility to transform the scale is by calling built in functions, as for example: `scale_x_log10()`

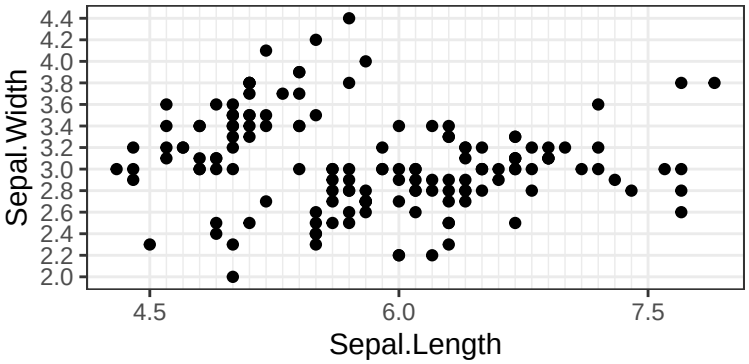
For purposes of readability or general appearance, the plot grid can be adjusted as well. The grid is also part of the theme. Hence, to adjust it, the theme has to be amended by calling the `panel.grid =` argument. There are three aesthetics that can be changed: line width, color and linetype. To adjust every single grid line, simply use `panel.grid =`. However, the major and minor grid lines of each axis can be adjusted separately. I will demonstrate how to do so, by changing color, size and linetype for each grid line separately. Obviously, gridlines can also be removed by calling `element_blank()`.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()+
  theme(panel.grid.major.x = element_line(size = 0.4, linetype = "solid", color = "green4"),
        panel.grid.minor.x = element_line(size = 0.3, linetype = "longdash", color = "darkgreen"),
        panel.grid.major.y = element_line(size = 0.4, linetype = "solid", color = "red4"),
        panel.grid.minor.y = element_line(size = 0.3, linetype = "dotted", color = "darkred"))
```



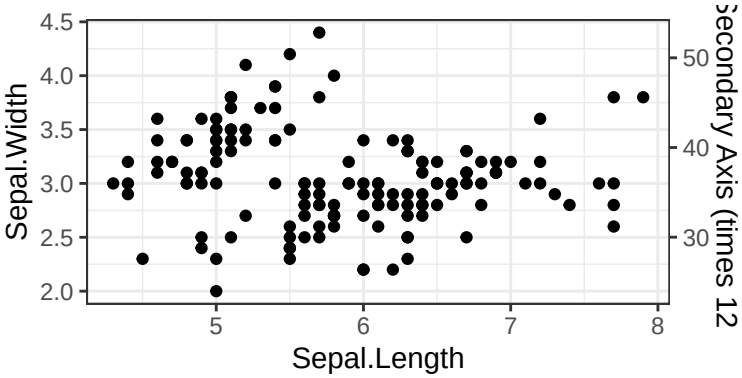
As I mentioned before, ggplot2 automatically adjusts the grid line and axis, according to the underlying data. In some cases it might be needed to adjust this. There are two main options. Either the grid can be defined as a sequence or every single gridline is defined in a list. I will demonstrate both. The frequency of grid lines is actually not defined in the theme. Therefore the function to call is `scale_continous()`. To adjust the major grid, we simply specify `breaks =`, for the minor grid the code is `minor_breaks =`. As mentioned, the first option of adjustment is to change the frequency of gridlines in a certain interval. To do so, we call `seq()` and add three components. The first number is the lower limit, the second the upper limit and the third tells in which frequency the gridlines should be. Important to note here, is that the upper and lower limit does only ally for gridlines. If they do not match with the plot limit, the output lacks of gridlines in some spaces. To set the gridlines manually, instead of calling `seq()`, a simple list is used which defines the places where ggplot2 draws gridlines.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()+
  scale_x_continuous(breaks = c(4.5,6,7.5),
                     minor_breaks = seq(4.5,7.5,0.1))+
  scale_y_continuous(breaks = seq(2,4.5,0.2),
                     minor_breaks = NULL)
```



Fortunately, ggplot2 makes it also very easy to add a second axis. Simply add `sec.axis =` inside the `scale_y_continuous` function to the plot code. Inside, the function `sec_axis()` has to be called. At first sight, the coding might seem complicated. The principal however, is fairly easy. The secondary axis default settings are derived from the primary axis. Hence, when it should show different values, mathematical operations are applied to transform the secondary axis, using the primary axis as the default. The `~.` argument calls the scale from the primary axis. Afterwards, the transformation is applied. The scond argument to be specified is the axis title.

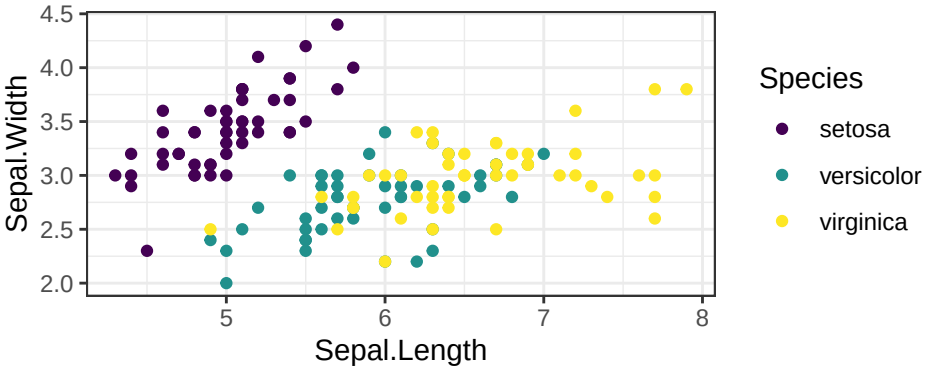
```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()+
  scale_y_continuous(sec.axis = sec_axis( ~. *12, name="Secondary Axis (times 12"))
```



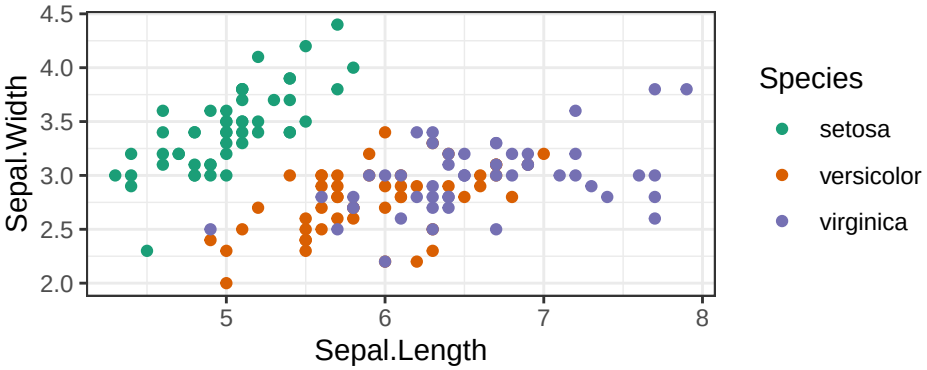
8 Colors

The ggplot2 package comes with its own color palette. A common customization is to use different colors than the package provides by default. I already discussed a simple way of changing the color manually. However, when setting each color manually, it always has to match the length of data displayed. A much more simple and elegant way is to use color palettes. There are many packages that provide additional colors and palettes. To create inclusive plots, I advise to check whether the palette used is colorblind friendly. I recommend two different packages to add color palettes: RColorBrewer and viridis. The viridis package was particularly designed to add colorblind friendly palettes. RColorBrewer, however, does have some palettes that are okay, but not all of them. So take care when using its palettes. Take a look at the palettes both packages provide to decide which one to use. The ggplot2 documenation of RColorBrewer palettes can be found here and the one for viridis here. Conveniently, since the release of ggplot2 3.0.0, ggplot2 already includes the RColorBrewer and viridis palettes. In consequence, there is no more need to actually install these packages anymore. To use these palettes, call `scale_color_viridis_d()` or `scale_color_brewer()`. Both viridis and RColorBrewer come with several palettes. To call them, use `option =` for viridis and `palette =` for RColorBrewer.

```
#viridis
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(colour = Species))+
  scale_color_viridis_d(option = "D")
```




```
#RColorBrewer
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(colour = Species))+
  scale_color_brewer(palette = "Dark2")
```



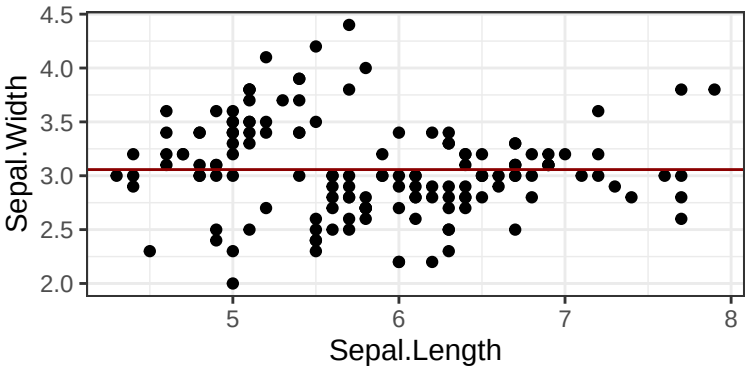
9 Anotation

When having created the base plot, it might be good to add further information. Some possibilities have already been discussed in this guide. It is always possible to add geoms such as errorbars, boxplots, pointrange etc, which might make it better to interpret the displayed data. In this short chapter I will introduce the most important commands to annotate and add additional data.

9.1 Lines

A very common task is to add lines to indicate some sort of threshold, mean or any other information. To create horizontal lines, use the `geom_hline()` and for vertical lines the `geom_vline()` command. Dependent on whether the line is vertical or horizontal, you need to specify the y- or x-intercept by simply calling `xintercept=`/`yintercept=`. If it is wished to continuously draw lines in a certain interval, you can define a interval, as for example: `1:7`. If the line should be dependent on data, simply specify the intercept in the `aes()` argument. In the following code example, I added the mean sepal width to the plot.

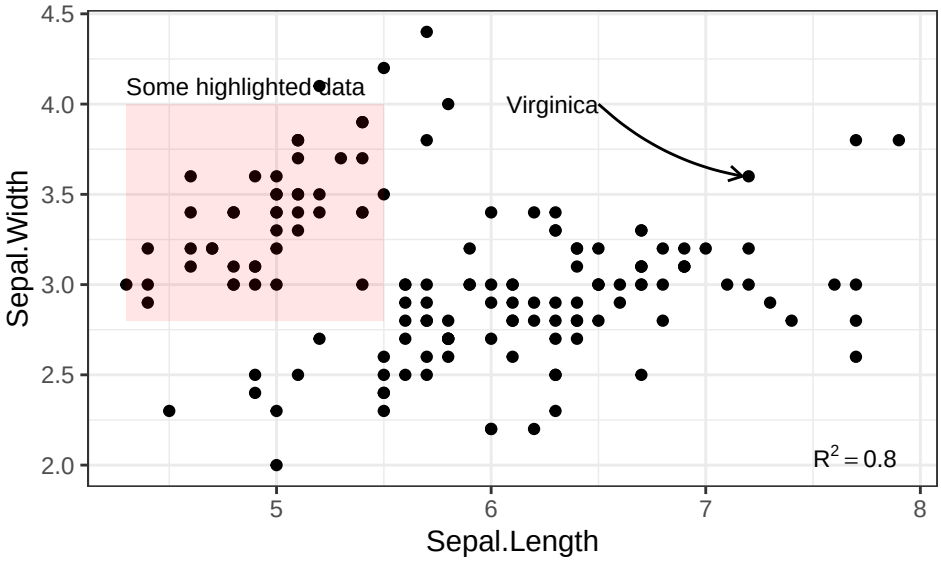
```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()+
  geom_hline(aes(yintercept=mean(Sepal.Width)), color= "darkred")
```



9.2 Annotation layer

I now want to introduce the annotation layer. This special function allows to add geoms that are not dependent on data, but manually defined vectors. It allows to increase readability and add all sorts of additional information. The possibilities are almost limitless. Consequently, it is not possible to go through each of them. In the example below I tried to give a limited overview how to use the annotation layer. First, I manually labeled a datapoint with an arrow pointing at it. Then I highlighted some data and in the bottom left corner, I added an imaginary R-squared value. I generally recommend to use the annotation layer to highlight specifics of your plot and add information.

```r
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()+
  annotate(
    geom = "curve", x = 6.5, y = 4, xend = 7.17, yend = 3.6,
    curvature = 0.15, arrow = arrow(length = unit(2, "mm")))+
  annotate(geom = "text", x = 6.5, y = 4, label= "Virginica", size= 3, hjust = "right")+
  annotate(geom= "rect", xmin = 4.3, xmax = 5.5, ymin = 2.8, ymax = 4, alpha = 0.1, fill = "red")+
  annotate(geom = "text", x = 4.3, y = 4.1, label= "Some highlighted data", size= 3, hjust = "left")+
  annotate(geom = "text", x = 7.5, y = 2.05, label= "R ^ 2 == 0.8", size= 3, hjust = "left", parse =
```



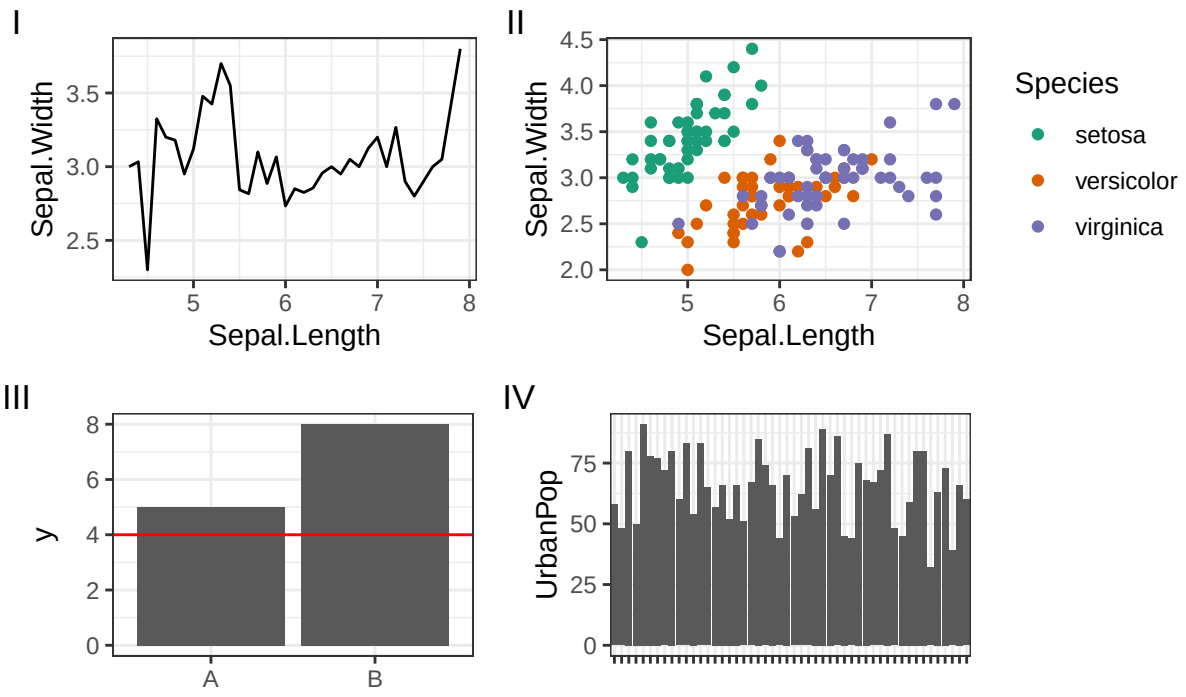
10 Arranging plots

There are several options to combine several plots and arrange them into a single one. Many websites that provide ggplot2 tutorials still use the ggarrange and ggpubr package. However, I recommend the patchwork package, as it generally is more easy to handle and more elegant in code. I will cover plot arrangement only breifly, however, there is a great chapter on the functionalities of patchwork in ggplot2: Elegant Graphics for Data Anlysis. Take a look at it when this guide chapter is not enough for your needs.

The patchwork package is very user-friendly and quite intuitive. To start, the plots need to be saved to the global environment. To combine the plots, we simply add them by using `+` operating sign. There are several customization that can be applied: Using `plot_layout(ncol=)`, the number of columns can be defined. The `/` separates the plots on separate rows, while `|` separates them on seperate columns. To standardize the theme, use `&` and set the wished theme. It is also possible to standardize the scales, using `& scale_continuous(limits = c())`. Title, caption and tags can be added by using the `+ plot_annotation()` function. I tried to show the condensed infromation that was just provided in an example.

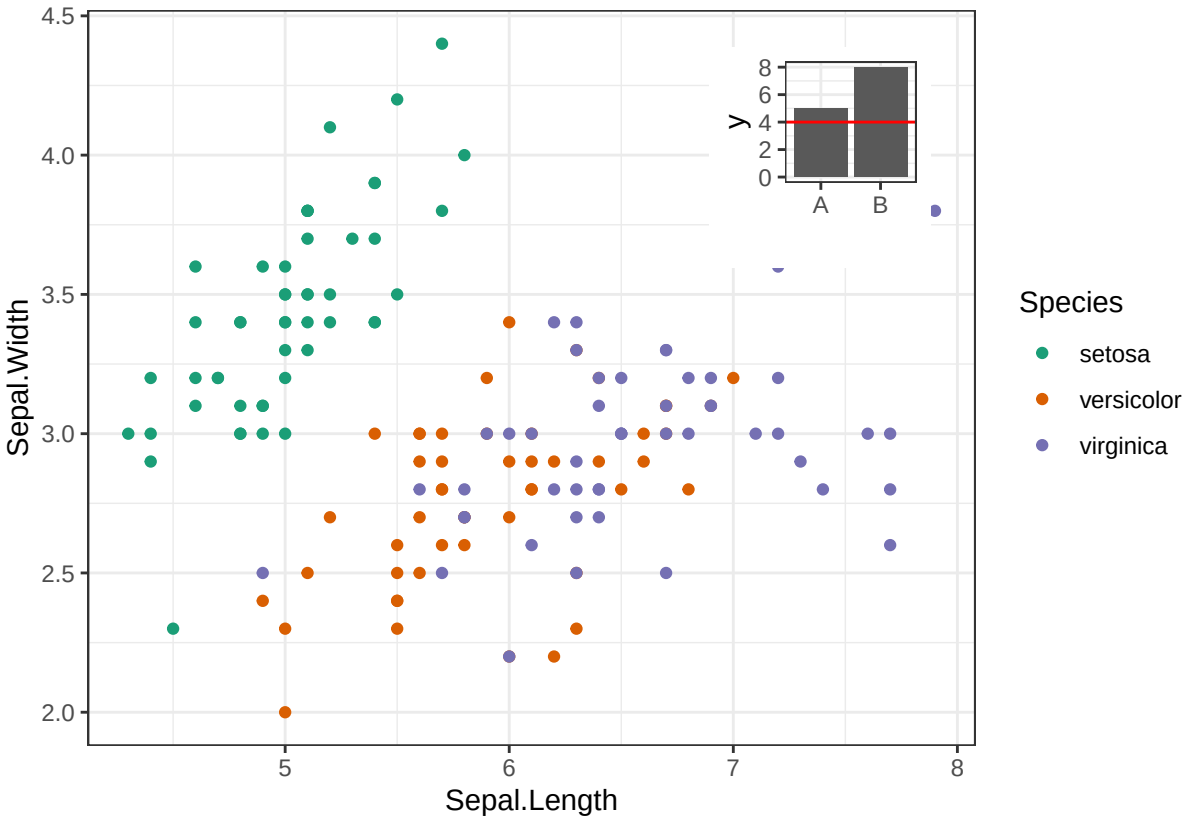
```r
#Create plots
p1 <- ggplot(data, aes(x= x, y= y))+
  geom_bar(stat= "identity")+
  geom_hline(yintercept= 4, color= "red")+
  xlab("")

p2 <- ggplot(iris, aes(x= Sepal.Length, y= Sepal.Width))+
  stat_summary(fun = "mean", geom = "line")

p3 <- ggplot(USArrests, aes(y= UrbanPop, x= rownames(USArrests)))+
```

```
  geom_col()+
  theme(axis.text.x = element_blank(),
        axis.title.x = element_blank())

p4 <- ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(colour = Species))+
  scale_color_brewer(palette = "Dark2")

#load package
library(patchwork)

#arrange plots
(p2+p4)/(p1+p3)+
  plot_annotation(
  title = "Plots of this guide",
  caption = "Source: Base R Datasets",
  tag_levels = "I")
```



The patchwork package furthermore allows to insert a plot in another one. To do so, use the `inset_element()` function. Then define which plot to insert and where.

```
p4 + inset_element(p1, left = 0.7, bottom = 0.65, right = 0.95, top = 0.95)
```



11 Closing statement

I hope this short guide was able to help with some issues or questions you had. The goal was to provide a basic introduction into data visualization in R and to cover the most common tasks and issues beginners might come across. Naturally, the options for further customization are vast. As I wanted to keep the guide short, it might be necessary to look for further help. There are many great websites that help you with all kinds of ggplot2 and R problems. I particularly reccomend: sthda.com, r-charts.com, ggplot2.tidyverse.org, https://www.datanovia.com/en/ and stackoverflow.com