# Data Exploration – Visualization

* Visualization is a great way of starting to get to know your data

**ggplot2**

* ***ggplot2 is an open source plotting system of R***, ***based*** on the ***grammar of graphics***(gg in ggplot stands for grammar of graphics), which tries to take the good parts of base R graphics and lattice graphics (both are graphic packages)
* the ***grammar of graphics*** is a ***school of thought*** saying that ***any graphical representation*** of data can ***be produced from a series of layers***

# Components of the layered grammar

* the layered grammar defines following components of a plot:
  + a **default dataset** (=dataframe) and **set of mappings from variables to aesthetics** ***(parameter tbd)***
  + one or more **layer**(s), with each layer having:
    - one geometric object ***(parameter tbd)***
    - one statistical transformation***(parameter tbd)***
    - one position adjustment***(parameter tbd)***
    - and optionally, one dataset (=dataframe) and set of aesthetic mappings
  + one scale for each aesthetic mapping used (automatically)
  + a coordinate system ***(parameter tbd)***
  + the facet specification ***(parameter tbd)***
* those 7 parameters compose the grammar of graphics: The grammar of graphics is based on the insight that you can uniquely describe any graph as a combination of a dataset, a geom, a set of mappings, a stat, a position adjustment, a coordinate system, and a faceting schema

# Introduction to the Layers

* layers are responsible for creating all the objects that we perceive on the plot
* A layer can be composed of 4 parts:
  + data and set of mapping from variables to aesthetics
  + a geometric object (= geom)
  + a statistical transformation (= stat)
  + a position adjustment (= position)
* hence: layer = **data + mapping + geom + stat + position**
* **a plot** may have **multiple layers**, for example when we overlay a scatterplot (geom\_point) with a smoothed line (geom\_smooth)
* in the layered grammar every part is defined separately
  + this makes it **possible to omit parts from individual specification and rely on defaults**:
    - **if** the **stat** is omitted the geom will supply a default
    - **if geom** is omitted, the **stat** will supply a default
    - if the **mapping** is omitted **the plot default** will be used
* *Yet, minimum individual specifications necessary when doing a plot:* 
  + default data set &set x- and y-axis through mapping variables to aesthetics
    - ***the result will be an empty plot plane***
  + only by adding a geom one gets to display the data

## 2.1 ggplot template:

> ggplot(data = <DATAFRAME I >, aes(<variable name x-Achse   
(= column name of dataframe>, <variable name y-Achse   
(=columnname of dataframe)>) +

required in order to display a plot w/ data

geom\_function I (stat=<STAT\_FUNCTION>, position =   
<POSITION\_FUNCTION>, <AESTHETICS OF THE GEOM (  
e.g. fill = <color>)>) +

geom\_function II(data = <DATAFRAME II>, stat=<STAT\_FUNCTION>,  
position = <POSITION\_FUNCTION>, <AESTHETICS OF THE   
GEOM (e.g. fill = <color>)>) +  
.

Those are add. layer, that can but do not need to be added

They are NOT required

R supplies defaults

.

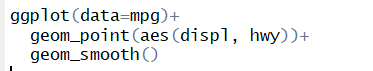
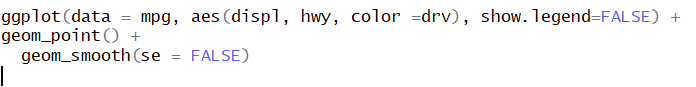
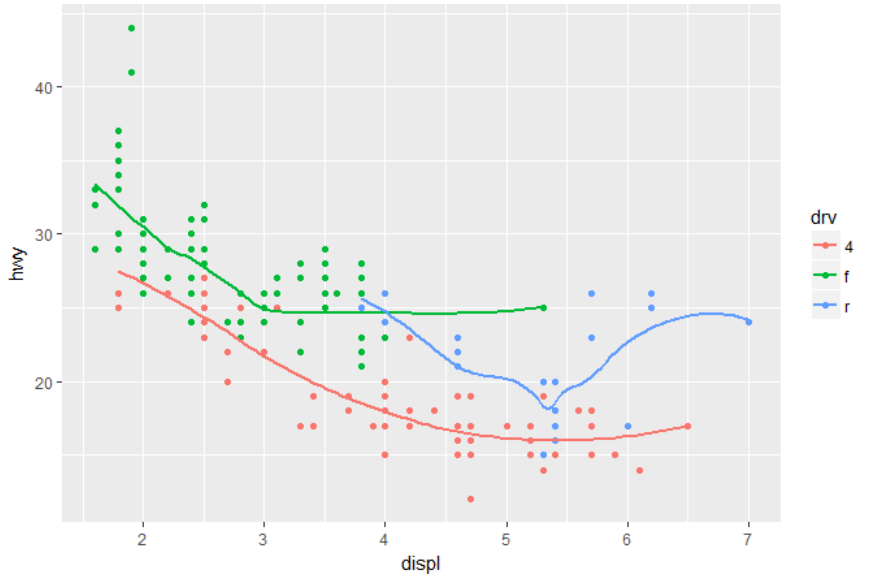
.geom\_function N +

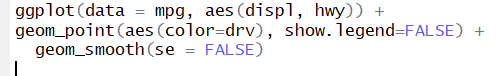
<COORDINATE\_FUNCTION> +

<FACET\_FUNCTION> +

<SCALE\_FUNCTION> +

<THEME\_FUNCTION> +

* **with ggplot 2 we always begin a plot with > ggplot()**
  + **ggplot () creates a coordinate system that we can add layers to**
* **the x-y-mapping argument could also be included in the geom function, but then** ggplot treats it as a **local mapping only** 
  + ggplot uses local mappings to **extend or overwrite** **global mappings** for that **respective layer only**
  + when we, for example, would add geoms, we would have to **add the x-y-mapping** to every add. geom
  + for example, when trying this code:   
    
  + one gets this error message:   
    
* therefore, it is advisable to insert all aesthetic mappings, including the x-y-mappin, in the ggplot\_function
  + R treats every mapping in the ggplot\_function as a **global mapping, that apply to each following layer**
  + so, including as many aesthetic mappings as possible in the ggplot\_function makes updates easier and reduces the risk of inconsistencies
* the concept of local and global mapping is a very important one to understand; see following example:   
   
* as one can see the legend is **not** removed
* this is due to first the global mapping of color to variable “drv” is implemented, which provides a legend by default, and applied to every following layer; after that the global definition to show no legend is applied, which is also applied globally, but **overwritten with the first geom**
* **thus we have to code as follows:**

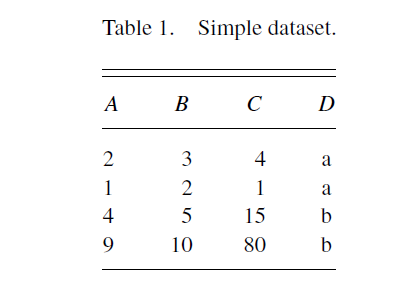
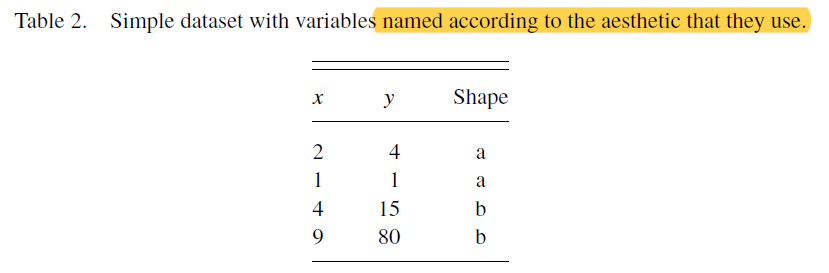




|  |  |
| --- | --- |
| Useful general remarks & Learnings | |
| **Adding a layer** | * „**+**“sign is used * **note I:** the plus-sign must be inserted in the same line as the last layer, so that R “knows” that sth. is added * **note II:** the sequence of the layers does matter! (see boxplot example)s |
| **Adding an argument** | * “,” sign is used * adding another argument or specification in a layer is simply done by using a comma |
| **Name plot** | * in order to not always have a long code displayed over multiple pages, it makes sense to save a code as a named object |
|  |  |
|  |  |
|  |  |
|  |  |

# Data aesthetic mapping

* data: it is important to remember that the data is independent from the other components
  + thus, we can construct a graphic that can be applied to multiple datasets
  + ***data*** are what ***turns*** an ***abstract graphic*** into a ***concrete graphic***
* aesthetic mapping:
  + **aesthetics are all visual properties of the respective geom**
  + aesthetics are basically all the aesthetical aspects of a graphic;   
    all the things that we can perceive on a graph are aesthetics
  + aesthetics include the size, shape or the color of the points
  + in order to display and convey [übermitteln, vermitteln] information about the data, specific aesthetics can be ***mapped*** to specific variables; this process is called ***aesthetic mapping***
    - creating a relationship between a variable and an aesthetic works with **aes(<AESTHETIC MAPPINGS>)** function  
      🡪 all relationships between those two need to be put into this function
    - the aes() function gathers together each of the aesthetic mappings used by a layer and passes them to the layer’s mapping function
  + every geom\_<GEOM> Function offers certain aesthetic mappings
  + some are common among different geoms, others naturally differ between the types of plots:
    - e.g. define x- & y-axis (hence map specific variables to x- & y-axis) is common among geoms (though there are a few exceptions (see cheatsheet))
    - shape or color aesthetic mappings (hence map shape or color to a variable; those differ between geoms
  + once you have mapped an aesthetic ggplot 2 takes care of the rest: ggplot2 will automatically assign a unique level of the aesthetic (e.g. a color) to each unique value of the respective variable; this process is known as ***scaling*** (see. 2. scaling)
  + e.g.:

 🡪 

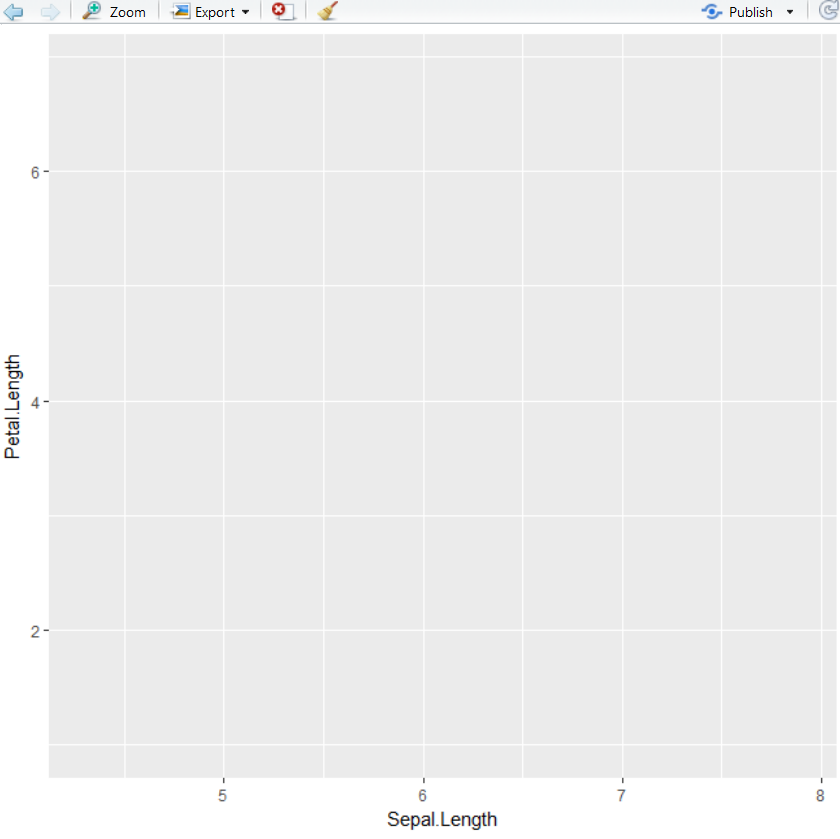
* + - if one would like to draw a scatterplot of Table 1, one could choose to position values of A horizontally (x-position), values of C vertically (y-position) and map categorical variable D to the shape of the points  
      🡪 this is displayed in Table
  + the ***details of the mappings*** are ***described by the scales***

**ggplot () Function:**

**ggplot w/out geom:**

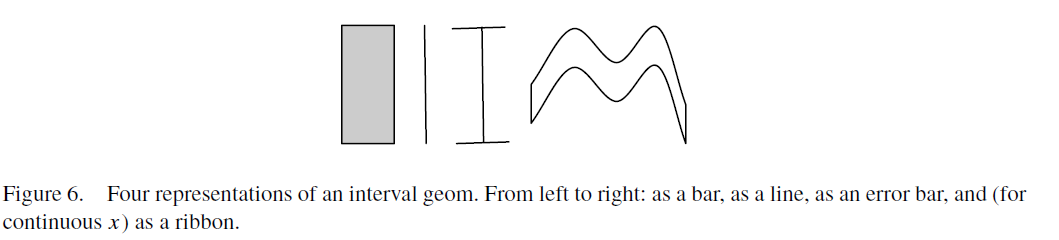
For example: *with Iris data frame (*Iris is a special flower*:*





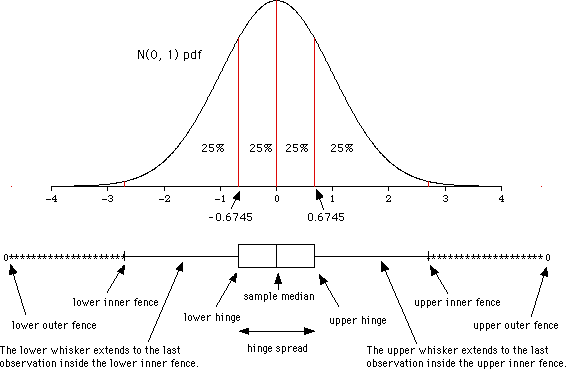
# Introduction to Geometric Object (or geoms)

* a geom is the geometrical object R uses to represent the data
* it determines ***the type of plot*** one creates; for example:
  + using a point geom (geom\_point) will create a scatterplot
  + using a line geom (geom\_line) will create a line plot
  + using a boxplot geom (geom\_boxplot), will create a boxplot
* w/out defining a geom, R has no specification of ***how to display the data frame in the plot***
* most of the geometric objects as we will use them are **functions** as follows:  
  geom\_<GEOM>()
  + within () add. arguments/parameters can be defined (for example aesthetics; see geoms & aesthetics)
* in some rare cases geoms might also be parameters (see stat\_<STAT> Function)
* geoms are basically ***the*** graphical representation of data
* geoms can be classified by their dimensionality: 0d: point, text; 1d: path, line; 2d: polygon (Vieleck), interval (an interval geom can, but must not be rendered 2d; see last illustration in Fig. 6)
* geoms are an abstract component and can be rendered [übersetzen] in different ways

for examples intervals: 

* geoms are mostly of general purpose, but do require certain outputs from a statistic:
  + for example, the boxplot geom requires the position of the upper and lower fences, upper and lower hinges, the middle bar and the outliers
    - any statistic used with the box plot needs to provide these values
    - if not any other stat is used, those values are provided by default from R
* every ***geom has a default statistic*** and ***every statistic a default geom***
  + for example, the ***bin statistic defaults*** to using the ***bar geom*** to produce a histogram
    - note: that does not mean that you can leave them out in the R code
  + overriding these defaults will still produce a valid plot, but it may violate graphical conventions

**Excursus boxplot terminology**

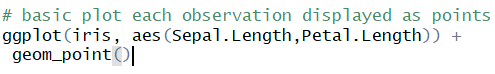


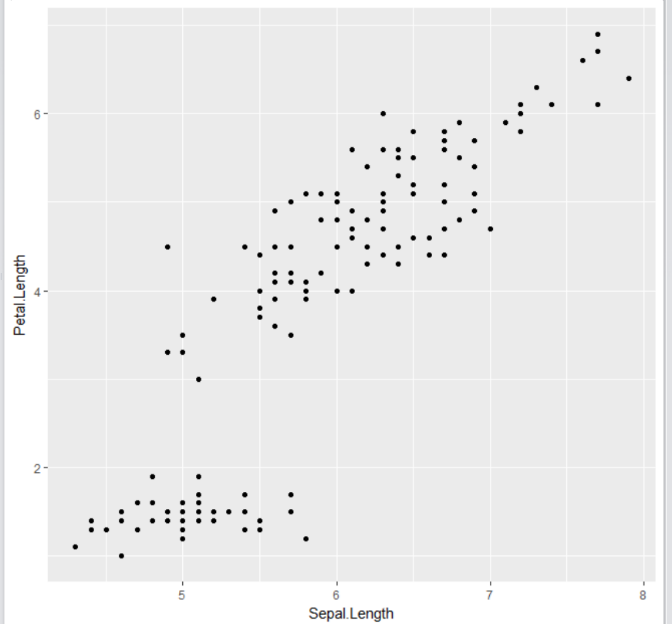
* + lower hinge: 0,25-Quantil (= unteres Quartil)
  + upper hinge: 0,75-Quantil (= oberes Quartil)
  + hinge spread= x 0.75 - x 0.25
  + lower inner fence: x 0.25 - 1.5\*hinge spread
  + upper inner fence: x 0.75 + 1.5\*hinge spread
    - points beyond these fences are valued as potential outliers

## 5.1 Simple examples of using geoms:

**geom\_point:**

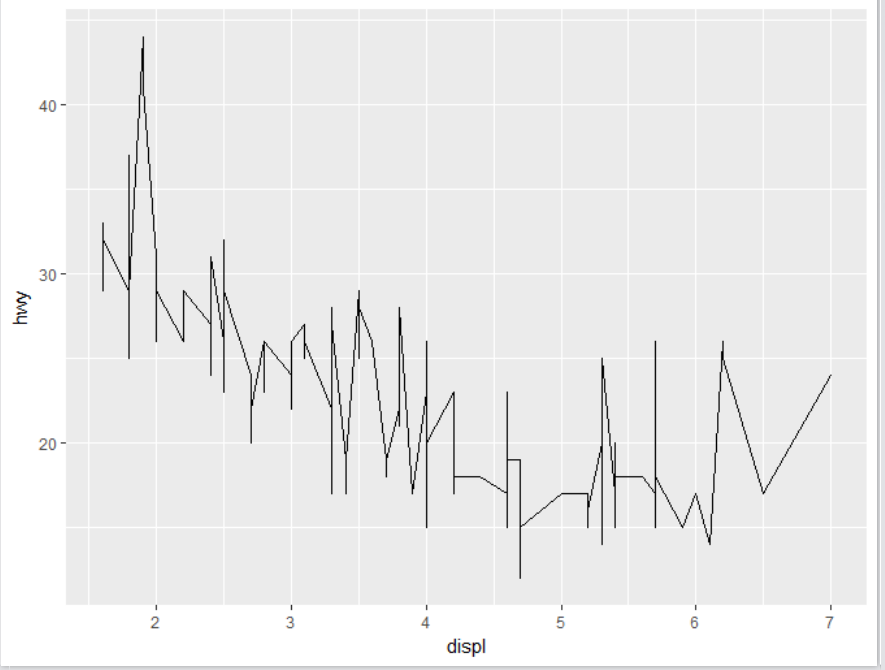
*with Iris data frame:*





**geom\_line: p. 39, R for data science**

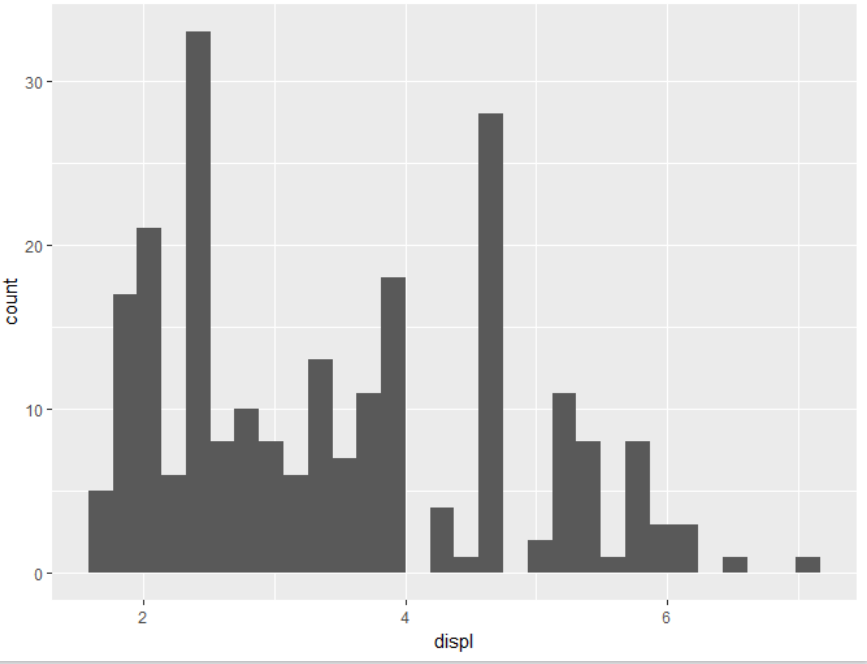




**geom\_histogram:**



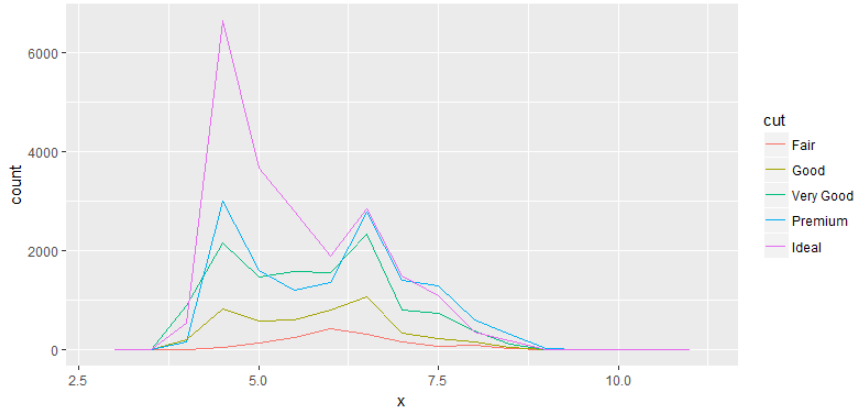
* **note:**
  + you can of course only assign one variable for a histogram
  + R uses count by default



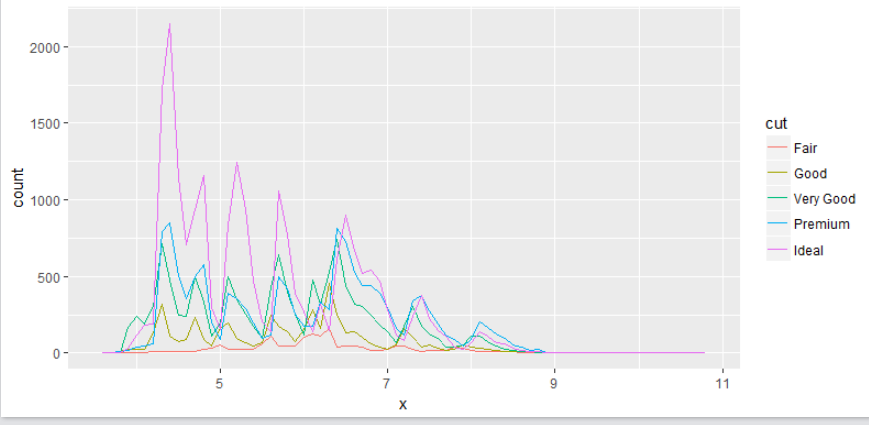
**geom\_freqploy()**

* should be used if one wishes to overlay multiple histograms (hence display frequency of multiple variables) in the same plot
* performs the same calculations as geom\_histogram () but instead of displaying the counts with bars, uses lines instead





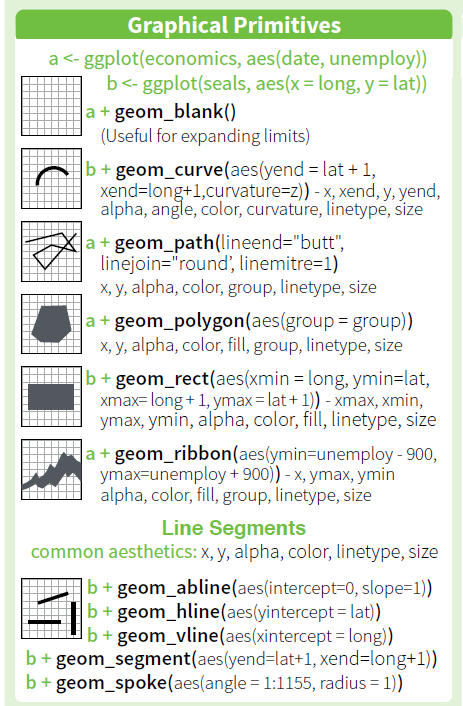
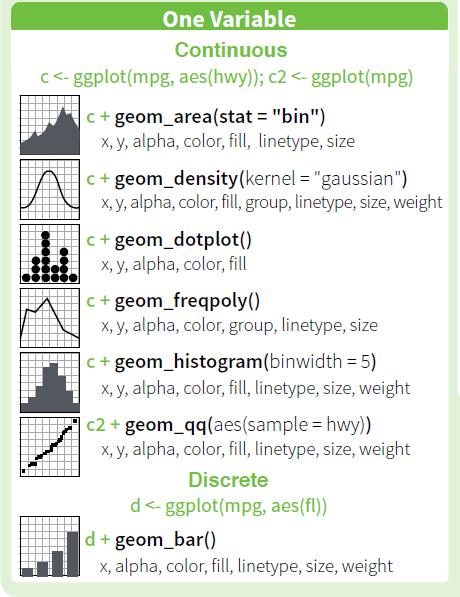


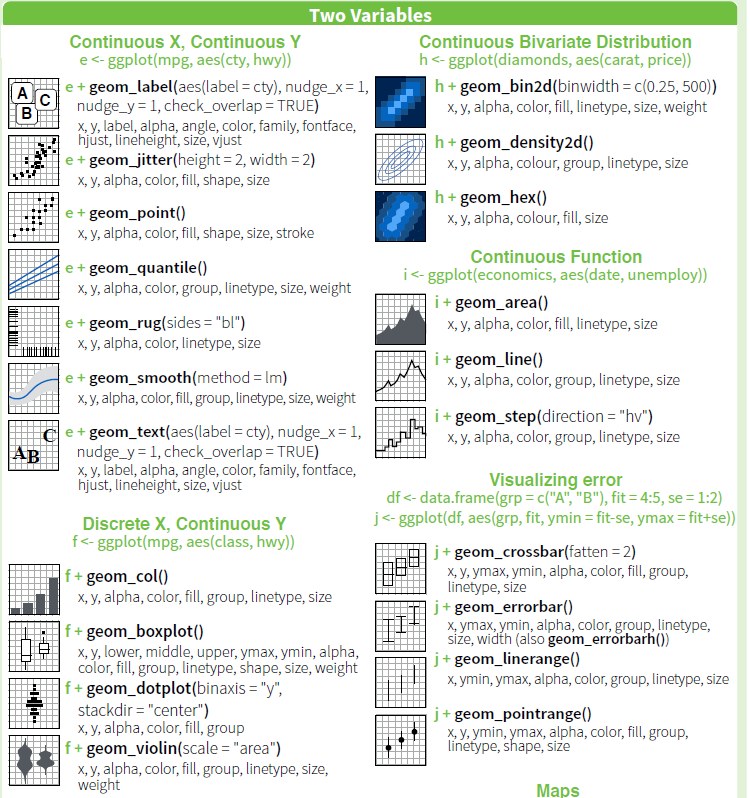


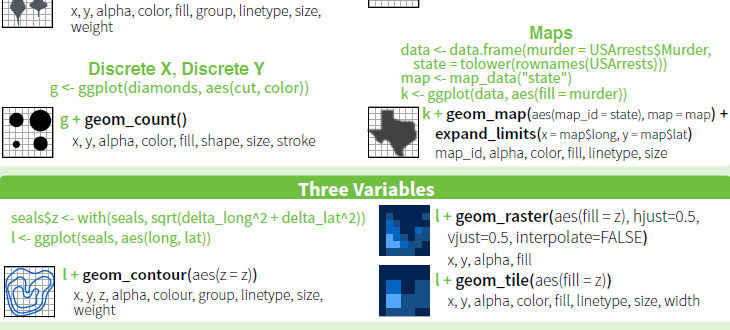
## 5.2 Geoms and aesthetics:

|  |  |
| --- | --- |
| **Purpose** | **Sources** |
| **List of all geoms available** | * cheatsheet above * <http://ggplot2.tidyverse.org/reference/> * > ??geom\_ |
| **List of all aesthetics that can be defined for a specific geom** | * >?geom\_’name of geom’ |

* **every geom comes with a set of aesthetics that can be defined:**



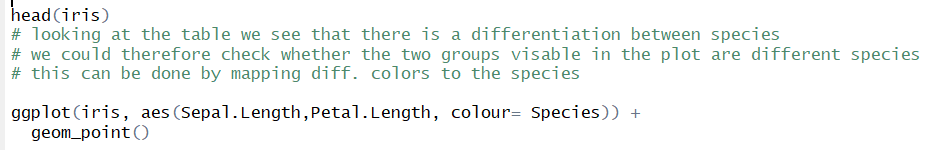


|  |  |
| --- | --- |
| 5.3 General remarks on geoms |  |
| **Geom\_bar** | * by default, R uses the count stat when no stat is added in geom\_bar() |
| **Geom\_histogram** | * the same applies to histograms: by default R uses count stat |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| 5.4 General remarks on aesthetics |  |
| **Learnings** | * One Variable can be mapped to multiple aesthetics (see example 3) * Note however: there is a rule in data visualization, that you should not have any visuals that are redundant * every aesthetic through which the reader does not get any add. info should be avoided |
| **Syntax Aesthetic Mappings** | * die Beispiele verdeutlichen, dass Aesthetics best. Variablen mit Eigenschaften mappen * d.h. die Syntax ist die Folgende:   aes(<Aesthetic> = <Variable>) |
| **Legende Aesthetic Mapping** | * Legenden werden von R automatisch generiert |
|  |  |
| **Color Aesthic** | * when mapping continuous variables w/ color aesthetic in a point geom, a scale is used; not distinct colors * when mapping categorical variables w/ color aesthetic in a point geom, distinct colors are used * when used not for mapping, but manually color only defines the outline of a shape |
| **Alpha Aesthetic** | * controls the transparency of a color: the larger value of a variable, the more solid a color point (see example 4) * most of the time the Alpha Aesthetic isn’t very helpful in separating a dataset, as the human eye has difficulties in quickly separating the diff. nuances |
| **Shape Aesthetic** | * R can only display 6 shapes at a time * thus, don't use shape mapping with variables that have > 6 attributes   + by default, all values of the respective variables left, will get unplotted * the shape aesthetic does also not make sense to map against continuous variabels:   + continuous variables are at least ordinally scaled; hence they have a natural order   + shapes do of course not have a natural order |
| **Size Aesthetic** | * when mapping size aesthetic to continuous variable in a point geom, the size of the points varies between each different value; although the legend only shows a few representative values |
| **Stroke Aesthetic** | * in geom\_point the stroke aesthetic works together with shape aesthetic:   + blackens every shape and largens it |
|  |  |

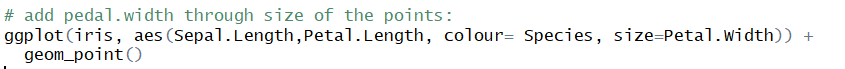
## 5.5 Examples of geoms & aesthics

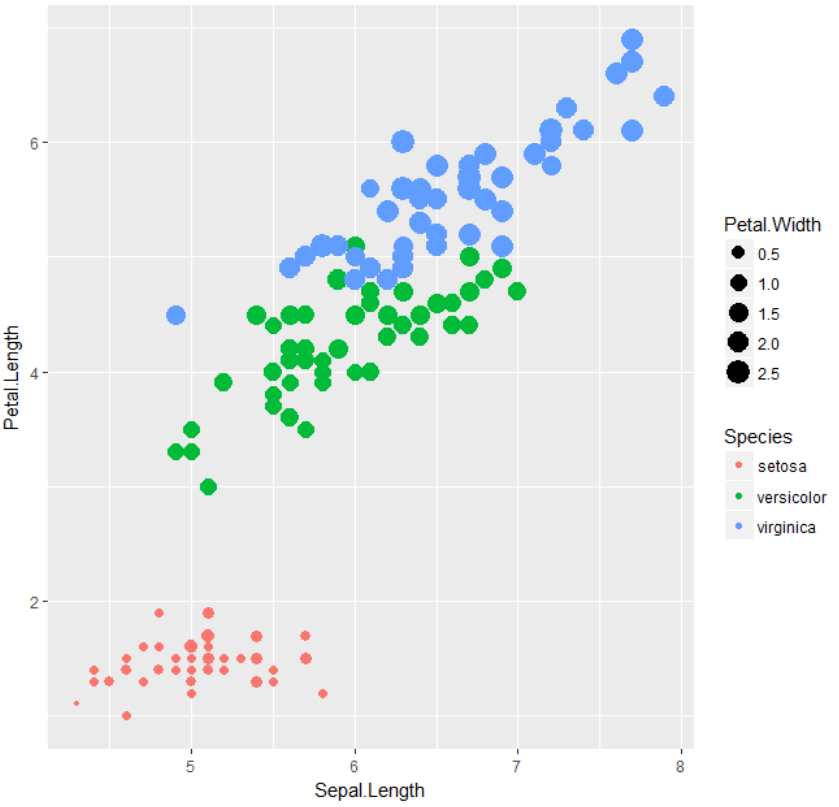
### geom-point\_mapping color-aesthetic to a variable



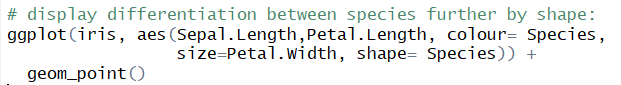


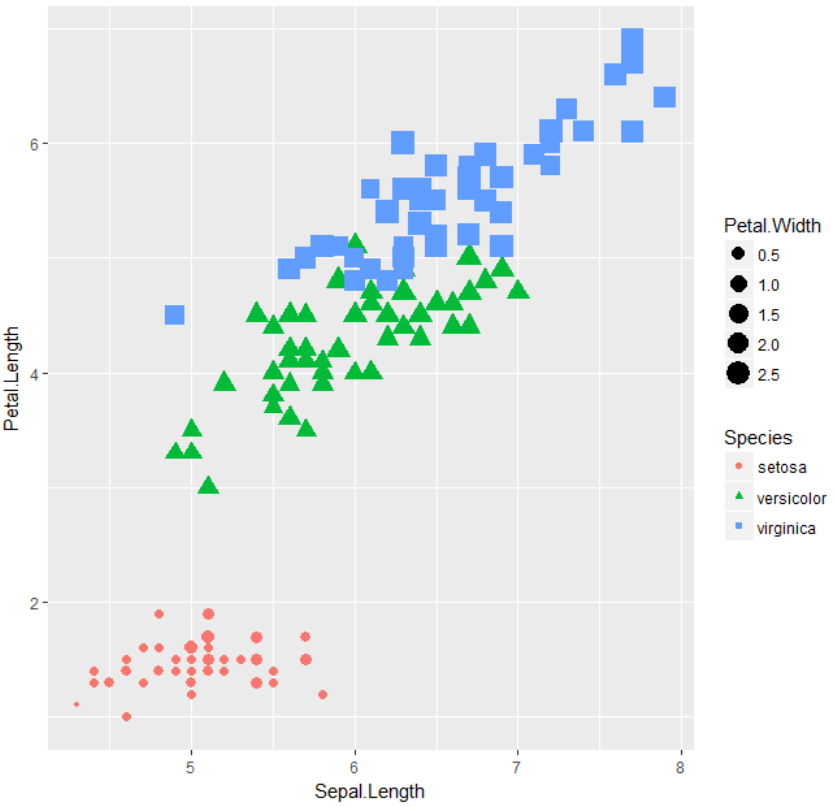
### geom\_point\_mapping size-aesthetic to variable





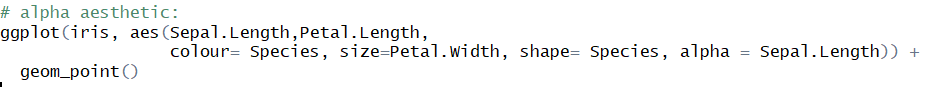
### geom\_point\_mapping shape-aesthetic to variable

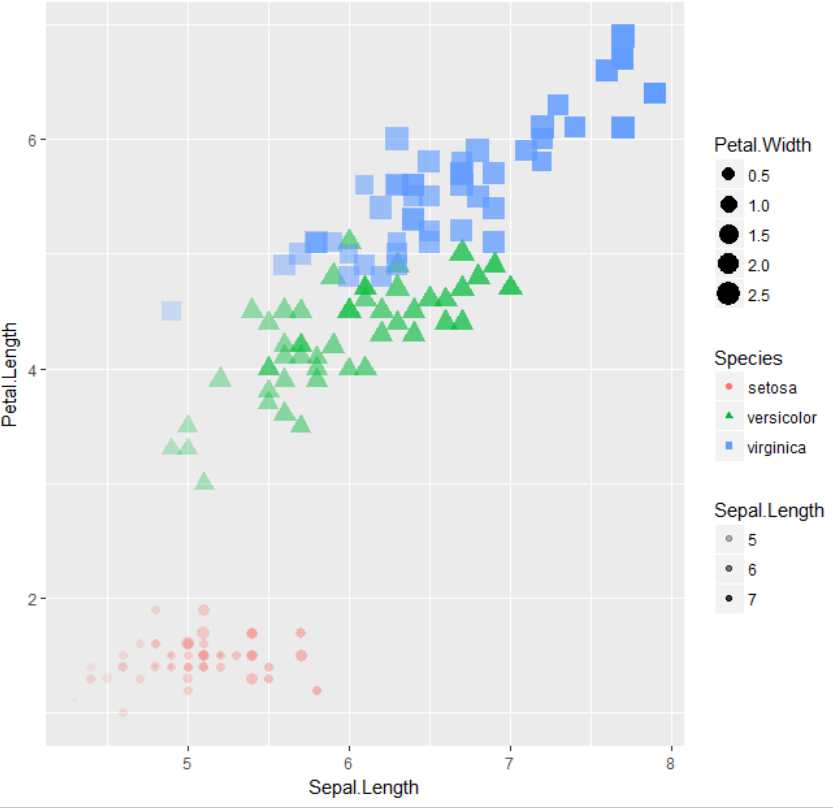




* **note: in this example the reader gets no extra information by displaying the differentiation of species also by shape 🡪 should be avoided**

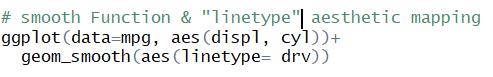
### geom\_point\_mapping alpha-aesthetic to variable

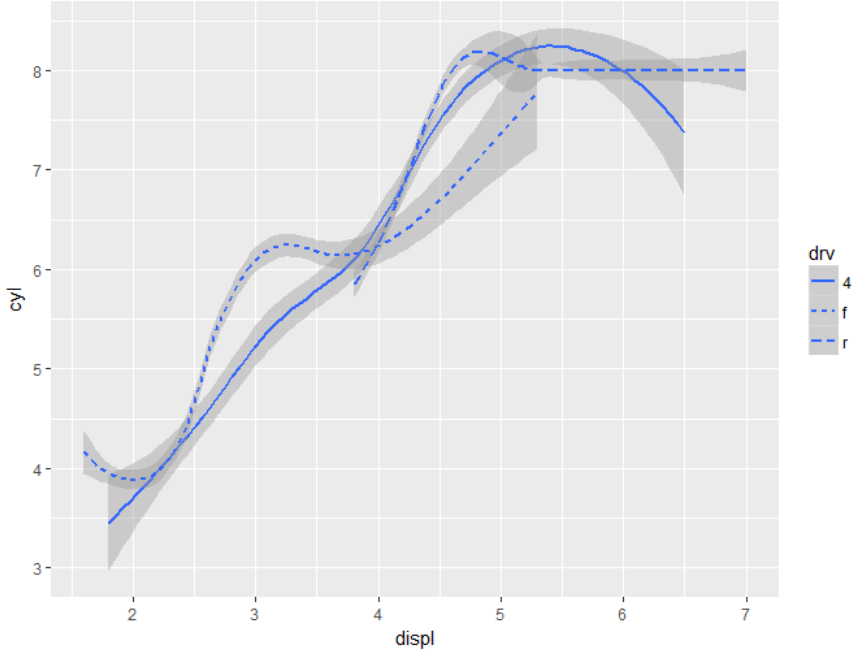




* **note:** as one can see the alpha aesthetic isn’t very helpful in separating the data set  
  🡪 it is rather hard to differentiate which sepal.length is 5 and which is 6  
  🡪 the human eye is not made for such differentiation

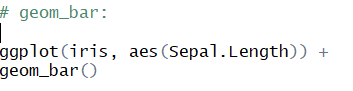
### geom\_smooth\_mapping linetype aesthetic to variable

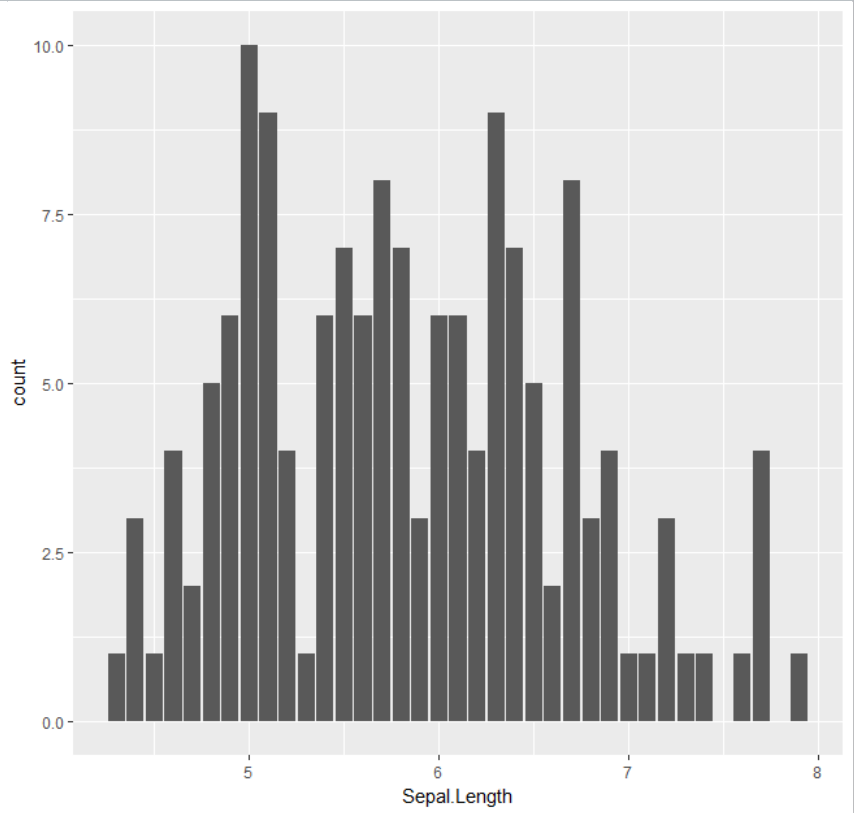




* mapping with aesthetic linetype returns a different line for each attribute of the mapped variable

### geom\_bar\_illustration of count-default

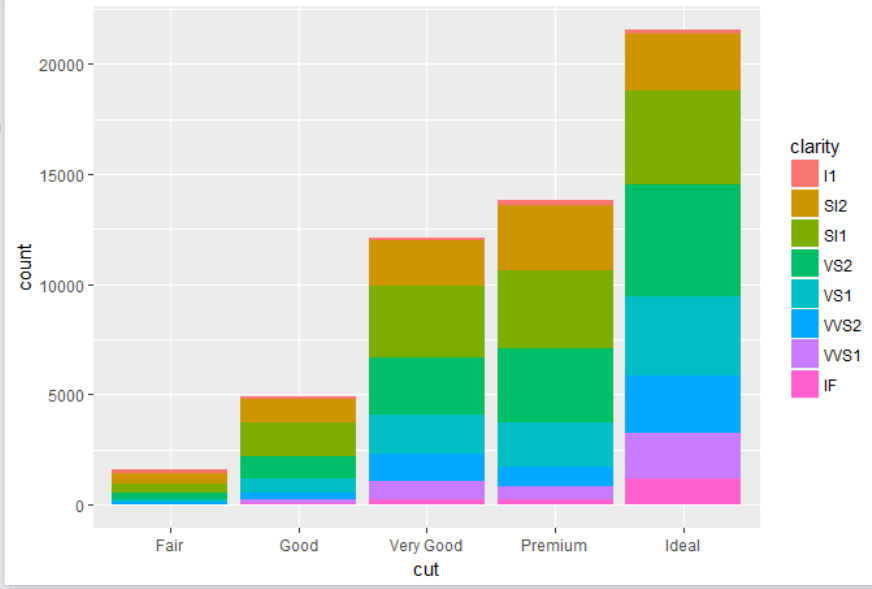




* by default, R uses the *count* stat when creating a bar char, counting the number of observations a variable

### geom\_bar\_mapping *fill* aesthetic to variable *(p. 47, R for Data Science)*

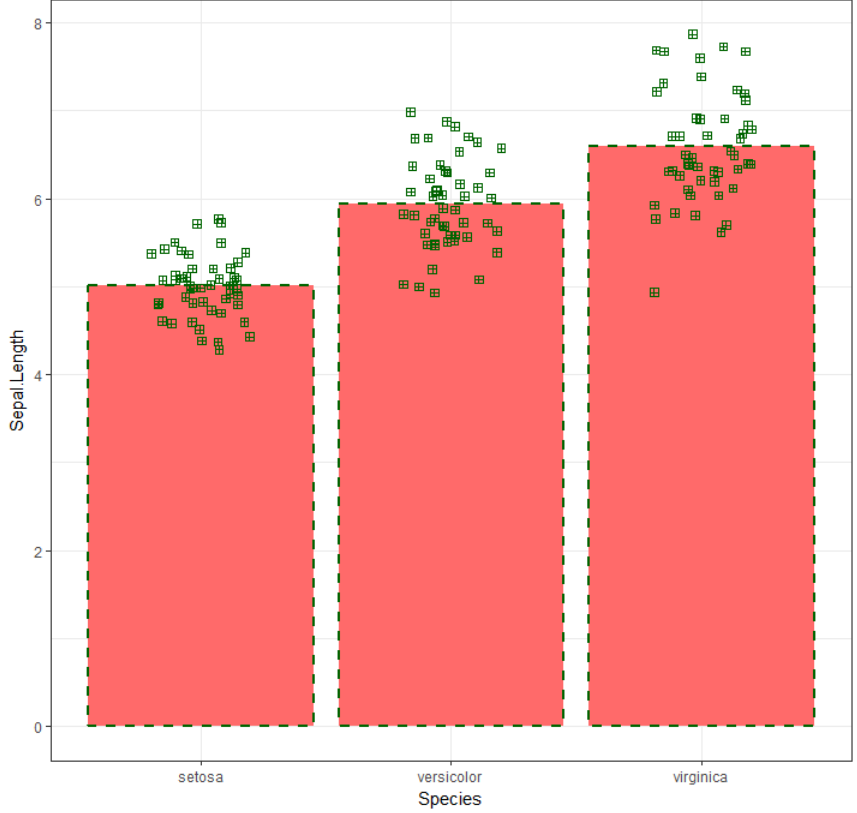




### Theme function:

* R provides certain theme-templates that change the whole design of the plot
* e.g.: theme\_bw, theme\_light, theme\_gray, theme\_dark:

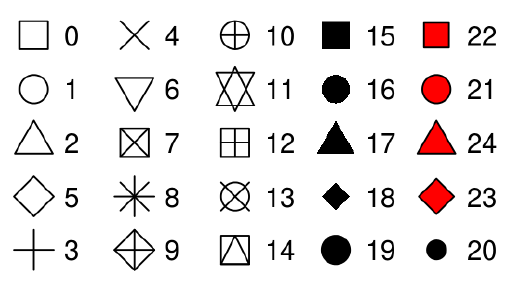




* **if have saved a theme under:** C:\Users\Kim F\Desktop\R&SQL\_for\_Business\_Analytics\3. Zusammenfassungen & Guidelines\ggplot2

## 5.6 Set aesthetics of geoms manually (≠ aesthetic mapping (!)):

* here the aesthetic setting shall not convey any information about a variable (as in aesthetic mappings) but only change the appearance of the plot
* this is done by setting the aesthetic by name as an argument of the geom function
  + thus, it goes ***outside the aes() function, but inside the geom function***
* Syntax: geom\_<GEOM>(<AESTHETIC>= <Attribute>)
* an aesthetic can be determined in 2 ways:
  + pick a value by yourself that makes sense for that attribute
    - e.g.:
    - name of a color as a character string for color or fill aesthetic
    - size of a point in mm for size aesthetic
  + use predetermined values (see sources in table below)
    - e.g.
    - use color code for color or fill aesthetic
    - use number for a specific shape
      * R has 25 built in shapes that are identified by numbers

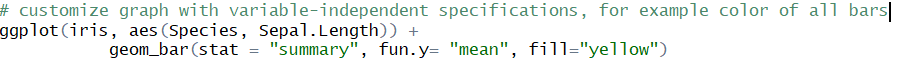


* + - * shapes 0-14: “color” can be changed
      * shapes 15-20: “fill” can be changed
      * shapes 21-24: “fill” and “color” can be changed

|  |  |
| --- | --- |
| **Purpose** | **Sources** |
| **Set of all graphical parameters** | * <https://www.statmethods.net/advgraphs/parameters.html> |
| **Color-codes** | C:\Users\Kim F\Desktop\R&SQL\_for\_Business\_Analytics\3. Zusammenfassungen & Guidelines\ggplot2 |

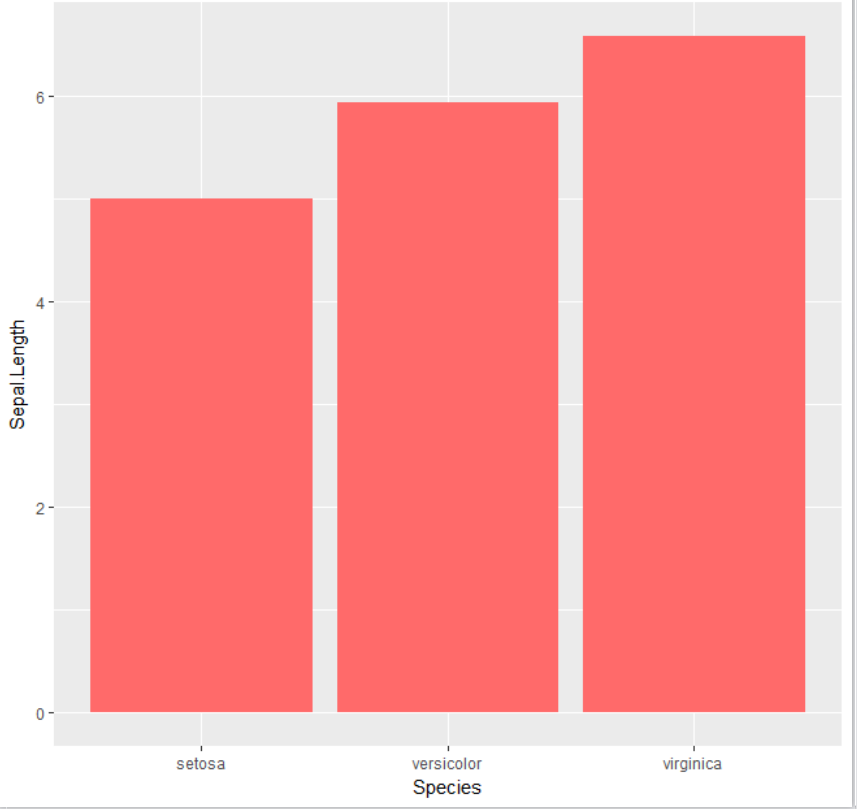
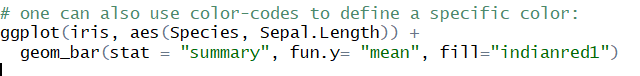
### 5.6.1 Examples for manual setting of aesthetics

#### (1) set color with fill =

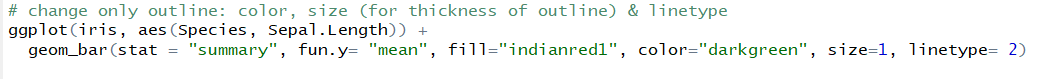


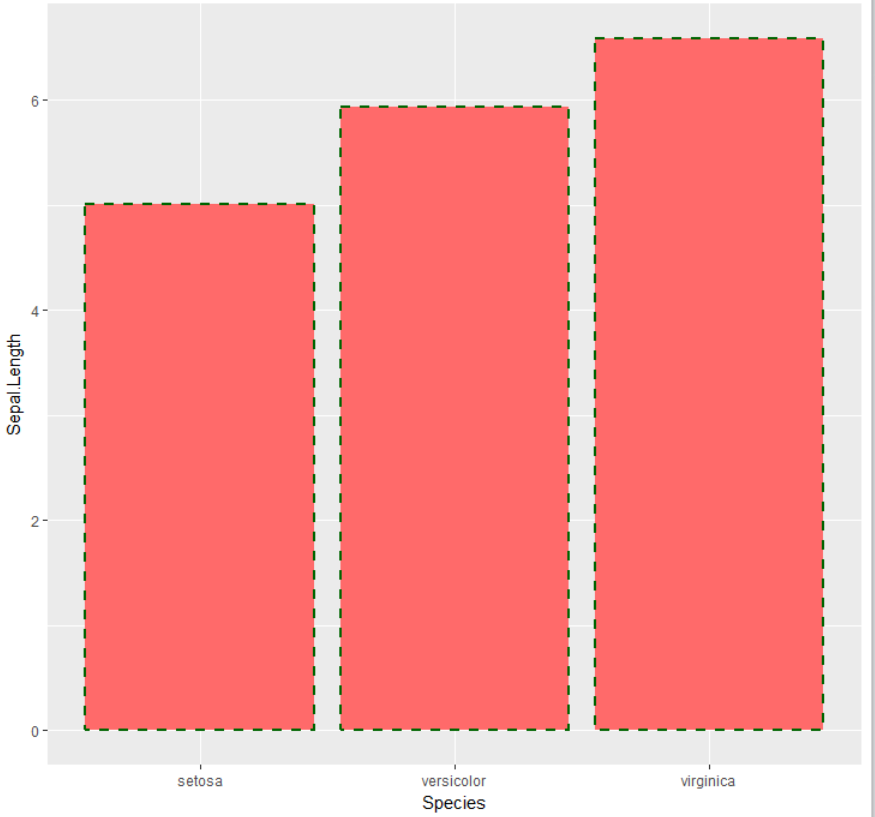


#### (2) set color with fill = and color code

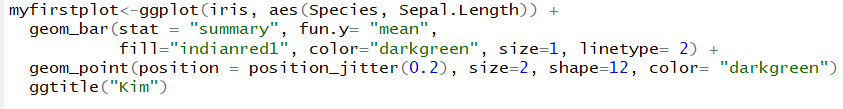


#### (3) set outline with color =





#### (4) Change the title of a plot: ggtitle layer & theme layer

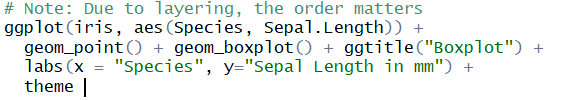
* either. ggtitle () only   
  
  + the title in this case “Kim” is then by default on the left side of the plot
* or ggtitle () & theme ()

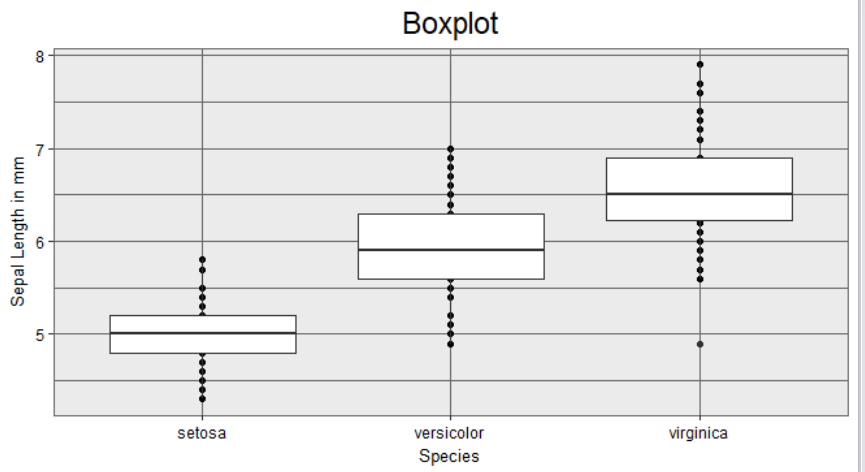


* **puts title in the middle**

#### (5) Change name of axes: labs() function layer

the „theme“ that I am referring to here, is the theme that I have personally created

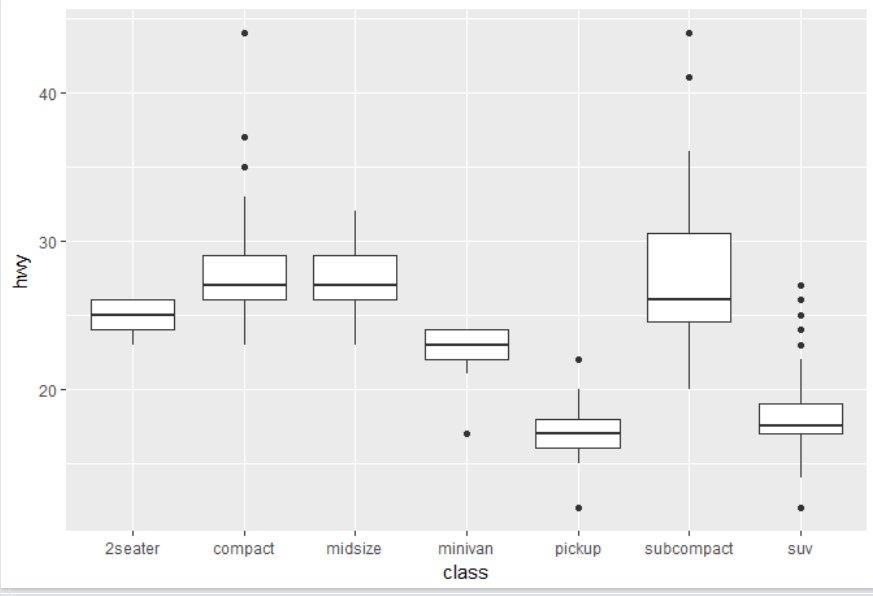




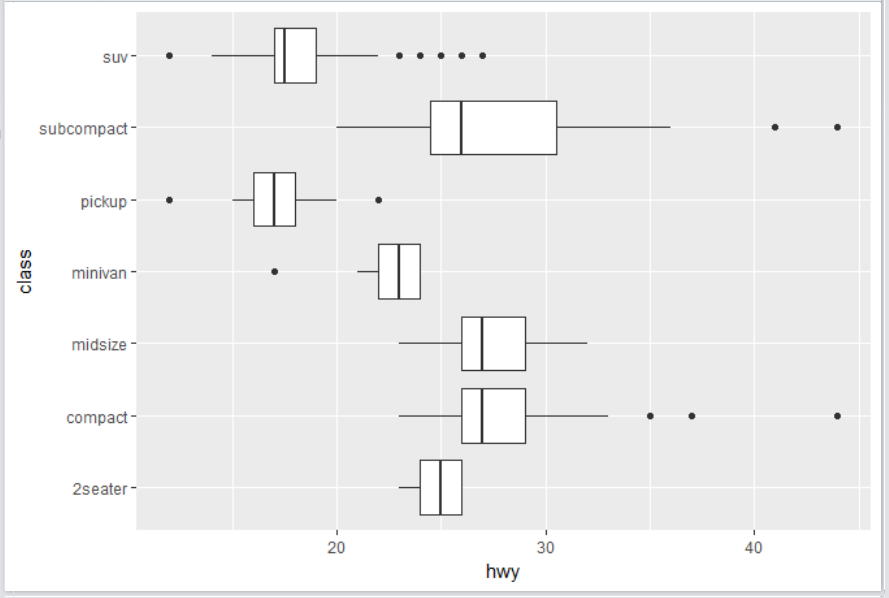
#### (6) Flip of Axis with coord\_flip, e.g. useful for boxplots where x attributes come w/ very long titles:

For example: *p. 51 R for Data Science:*









* note: we **cannot** change the aesthetic mapping of x- and y-axis; **the boxplot still has to be created on continuous variable hwy** 
  + a boxplot is always useful if we want to depict the distribution of a continuous variable, given the attributes of a categorical variable (d.h. für vers. Merkmalsausprägungen)
* therefore we need to use coord.flip, which changes the axes only

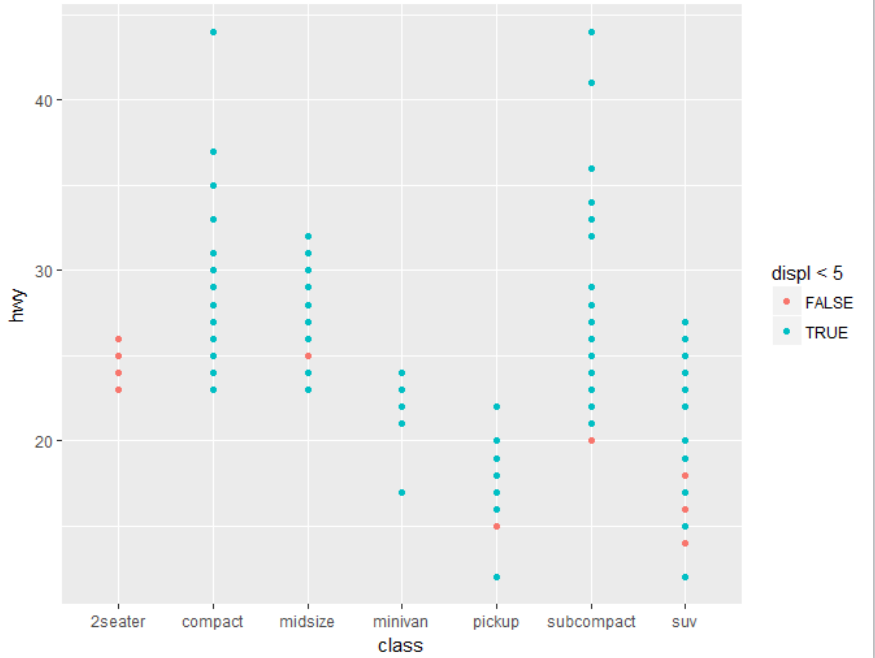
## 5.7 Aesthetics and relational statements:

* Aesthetics can also be mapped to expressions

*Exercise 6, p. 30 (R for Data Science)*

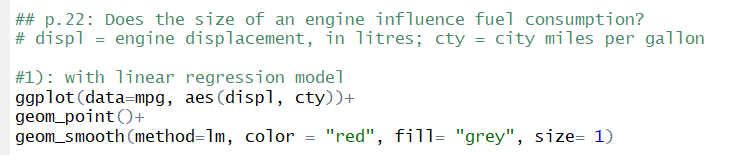


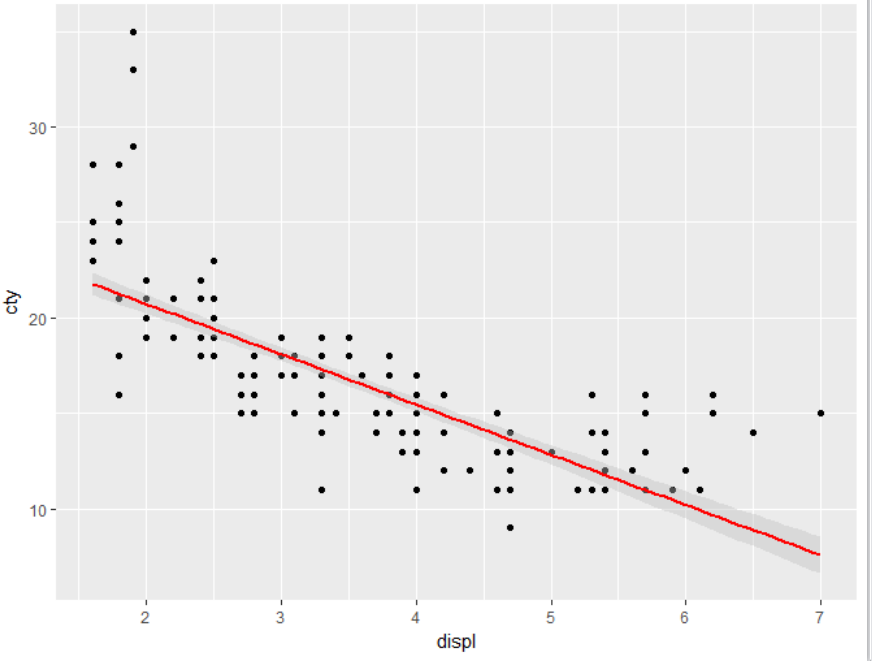
* displ is a variable in the data frame



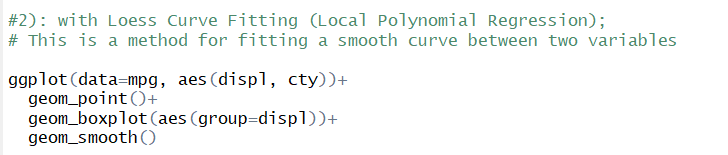
## 5.8 Layer multiple geom\_functions:

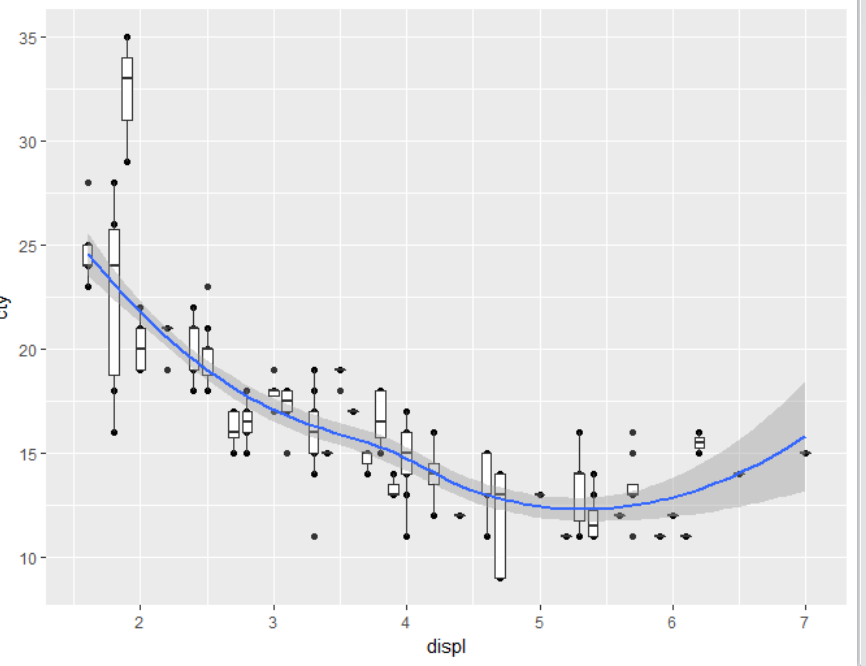
### geom\_point & geom\_smooth\_with linear regression model ‘ (mpg data; p. 22 R for Data Science)



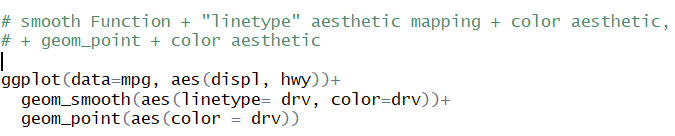


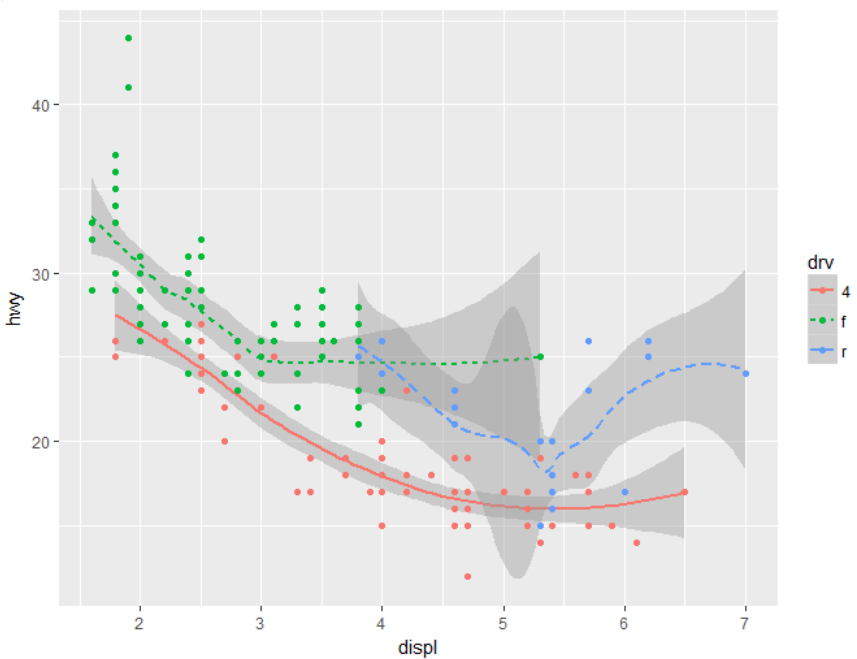
### geom\_point & geom\_smooth\_with Loess Curve Fitting & geom\_boxplot



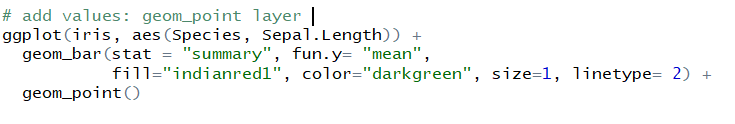


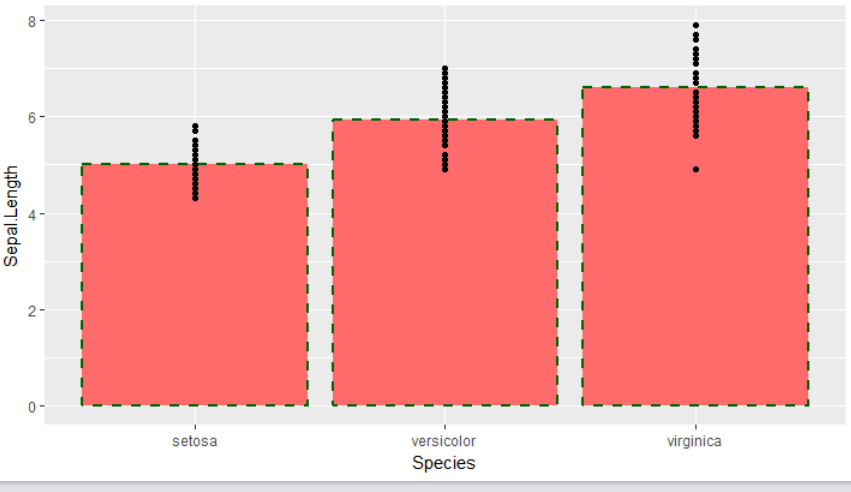
### geom\_smooth\_linetype & color aesthetic, geom\_point color aesthetic





### geom\_bar & geom\_point





### Multiple geoms + filter function

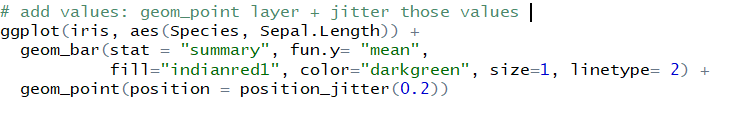


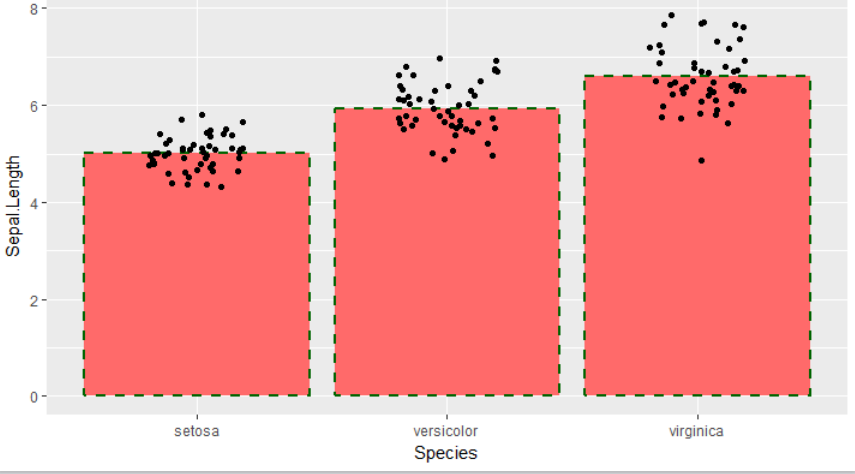
* here geom\_smooth is only applied to a subset of “class” variable, which is “subcompact”
* “se” stands for standard error; se=FALSE means that no standard error should be displayed around the smoothening line

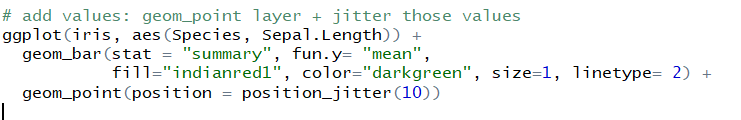
### (7) geom\_bar & geom\_point\_jitter values in geom\_point

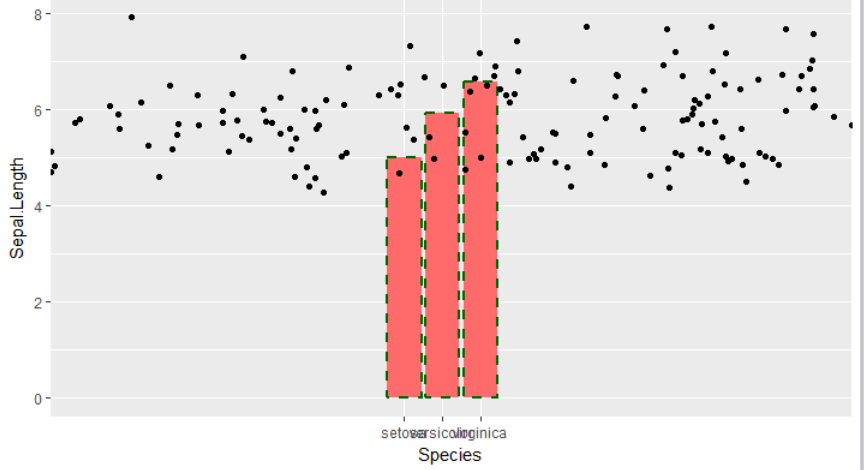
position\_jitter() is a function

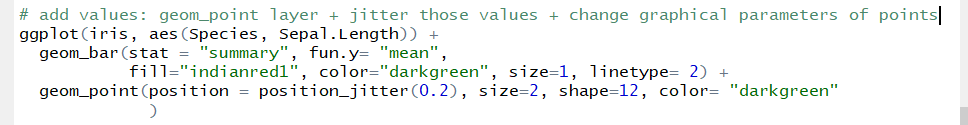
one can define by what degree values shall be jittered



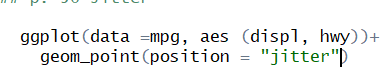








**Alternatively one could shorten the jitter command as follows:**



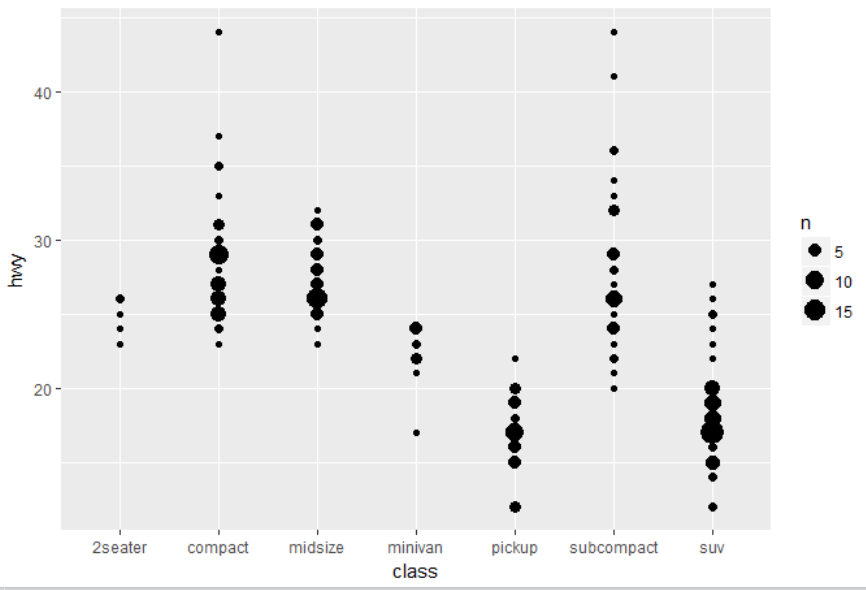
* however, this does not allow for a parameter defining the degree of jittering



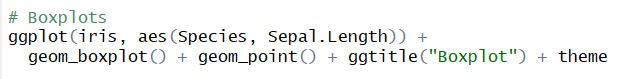
* geom\_jitter Function however, knows width and height parameter with which one can define degree of jittering
* width sets the degree of horizontal jittering, height the degree of vertical jittering

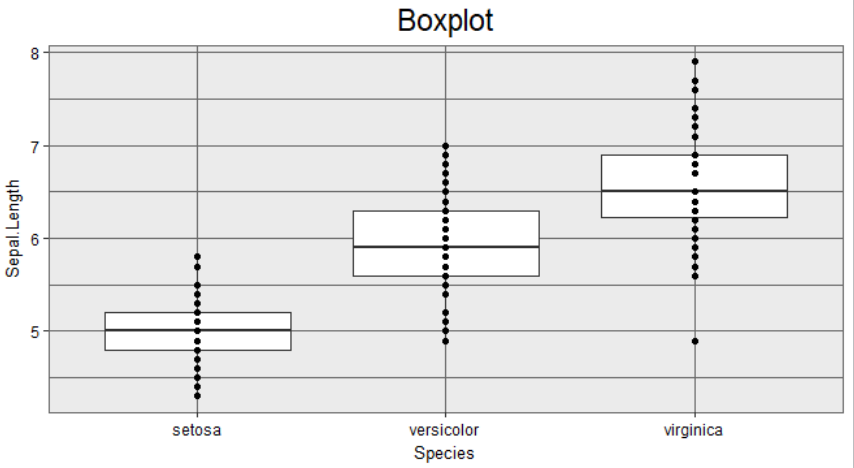
### geom\_point & geom\_count

* This is a variant [geom\_point](http://127.0.0.1:27988/help/library/ggplot2/help/geom_point) that counts the number of observations at each location, then maps the count to point area. It useful when you have discrete data and overplotting.

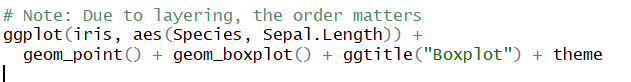
 

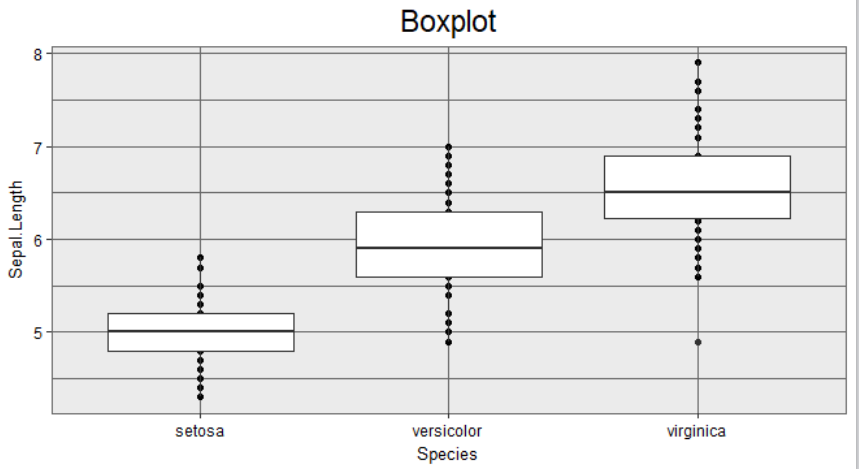
### geom\_boxplot () & geom\_point ():





the „theme“ that I am referring to here, is the theme that I have personally created

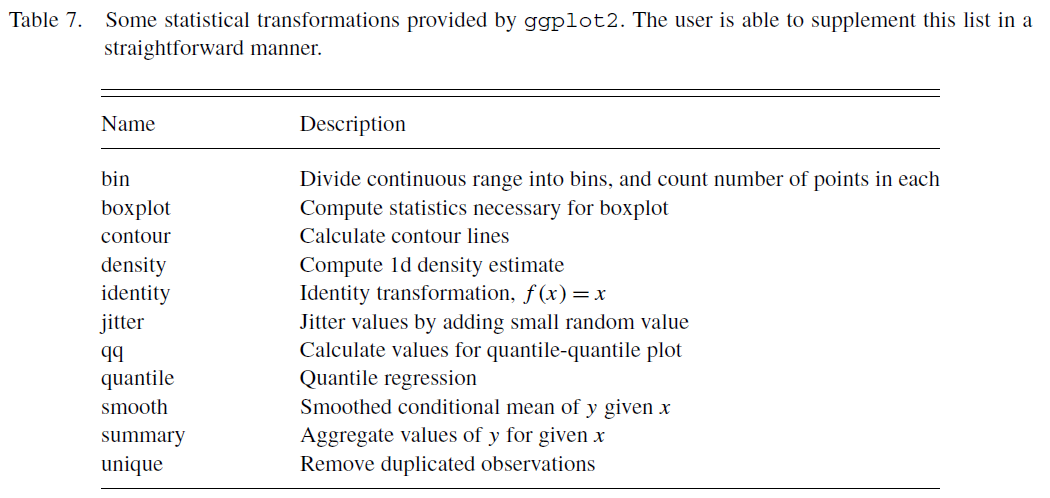




the „theme“ that I am referring to here, is the theme that I have personally created

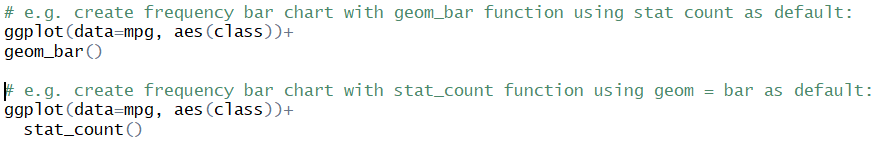
# Statistical transformation (or stat)

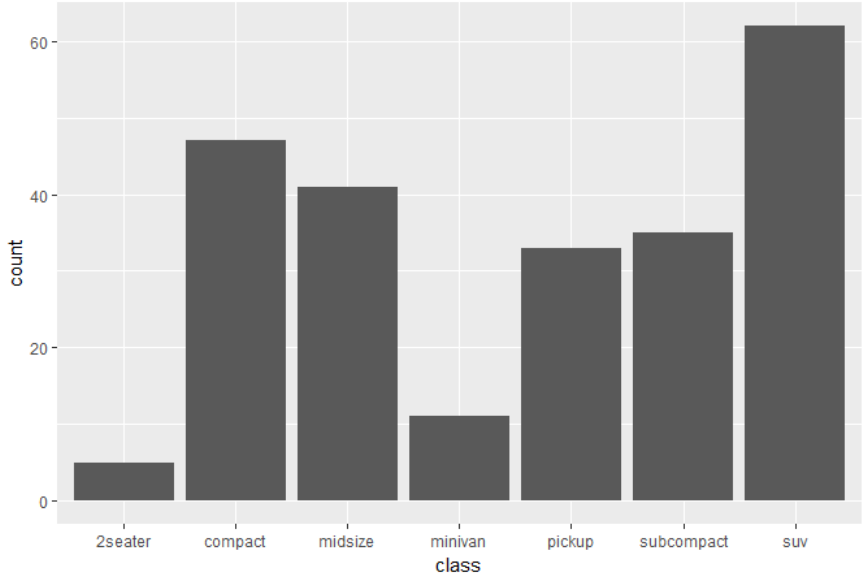
* ***transforms*** the ***data***, typically by summarizing it in some manner, so that it is displayed or displayable in a certain type of plot (hence geom)
  + it is sort of an intermediary between the data and the geom (see Histogram)
* to make sense in a graphical context, a stat must be location-scale invariant [Invarianz ggü. Koordinatentransformationen wie Translation (Verschiebungen), Rotation und Skalierung]:   
  f(x+a) = f(x) + a resp. f(x\*b) = f(x)\*b
* a stat takes a dataset as input and returns a dataset as output
  + it can add new variables to the original dataset



## 6.1 Interchangeability: geom\_function & assigned stat default / stat() function & assigned geom default

* remember: every geom\_function has a default stat and every stat\_function has a default geom
  + thus, the same default plot can be created by using either geom\_function or the stat\_function  
    🡪 they can be used interchangeably
  + for example:

both lead to the following bar chart



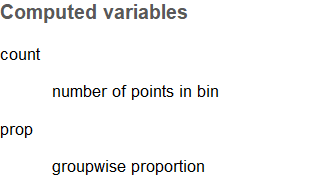
* + the default for a geom\_/stat\_function can be check by using **?geom\_function, ?stat\_function**

|  |  |
| --- | --- |
| 6.2 General remarks on stats |  |
| **Rely on defaults** | * one should check the default stat for a certain plot before integrating a new stat * this can be done using:   + ?geom\_function   for example:   * + - everywhere where an argument is provided, such as with stat=”count”, this stands for the default   + ?stat\_function   For example:     * + - everywhere where an argument is provided, such as with geom=”pointrange”, this stands for the default * for an exhaustive list of all geoms & their default stats, see: http://ggplot2.tidyverse.org/reference/ |
| **How to integrate a stat in ggplot w/ using defaults** | * a stat can be integrated in ggplot by 2 ways: * 1) change default stat of a geom by integrating stat () Function in geom () Function   + thereby you apply the respective stat() Function only for that geom   + Syntax:  geom\_<GEOM>(stat=<STAT> (…), <aesthetic mappings>, <design settings>...)   + e.g. geom\_bar(stat=”summary”) * 2) use stat\_<STAT> () Function as additional layer   + Syntax:  stat\_<STAT>(geom **=** <NAME>, <aesthetic mappins>, <design settings>)   + note: here we *call* [this term is used for every term stated in the parenthesis] a default geom to make a layer equivalent to the geom function   + here one can furthermore use ..<NAME>.. syntax to map stat results to aesthetics:   + examples of stat () Functions: |
| **Stat\_summary** | * summarises the y values for each unique x value * by default, R uses the mean if summary stat is not specified   + this holds for both: the stat\_<STAT> Function, as well as stat=<STAT> parameter * if one wants to use another function than the default, this has to be added * see: <http://ggplot2.tidyverse.org/reference/stat_summary.html#summary-functions> for a list of all SUMMARY FUNCTIONS * see example (7) for an example |
| **Stat\_identity** | * **this is especially useful, when one has calculated some new values in a new dataset** * it plots the actual y values of variable x * this is for example useful, when one wants to overwrite the count default in a bar chart and plot the actual values * for example:       **Note: instead of using the stat\_identity, one can simply use:  geom\_col** |
|  |  |
|  |  |
|  |  |
|  |  |

## Examples stat functions

### geom\_bar & stat function

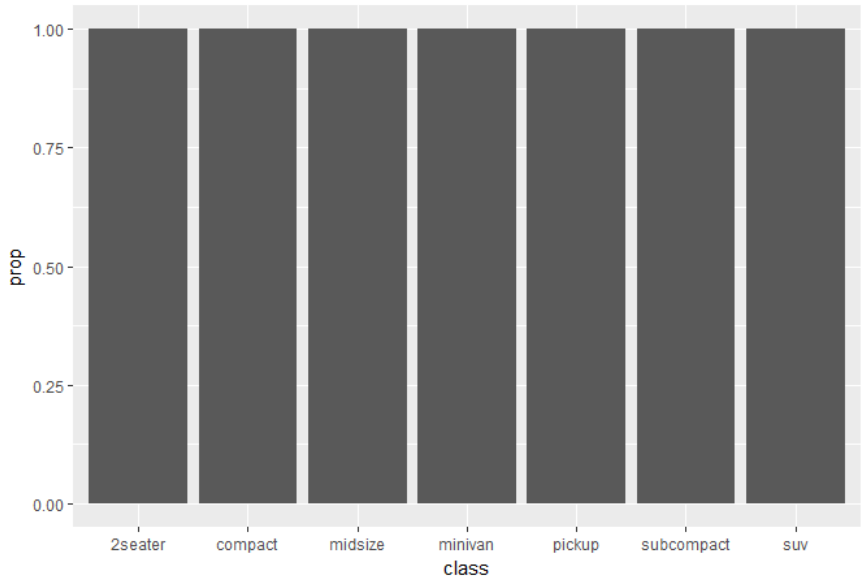
* there might be more than one variable computed by the default stat\_function of a geom
* to see all computed variables use ?geom\_function and go to section “computed variables”
* for example, for geom\_bar you get:



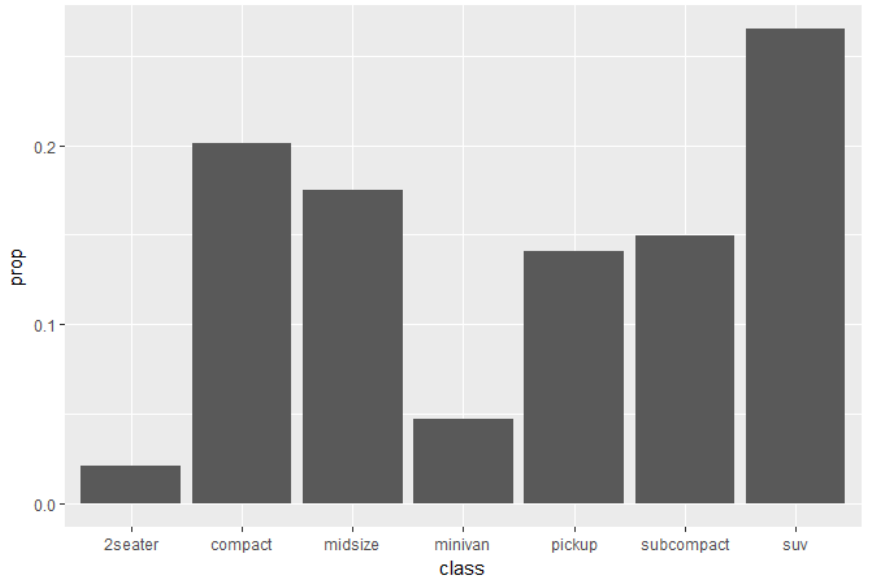
* + count is calculated by default
  + prop (which is rel. frequency) can be calculated as follows:



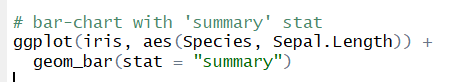
* note the term “group =0”
  + if this term is not added, by default, R takes relative frequency of an attribute of variable x, for that very same attribute only
  + hence rel. frequency is always = 1 and one would get the following plot instead:

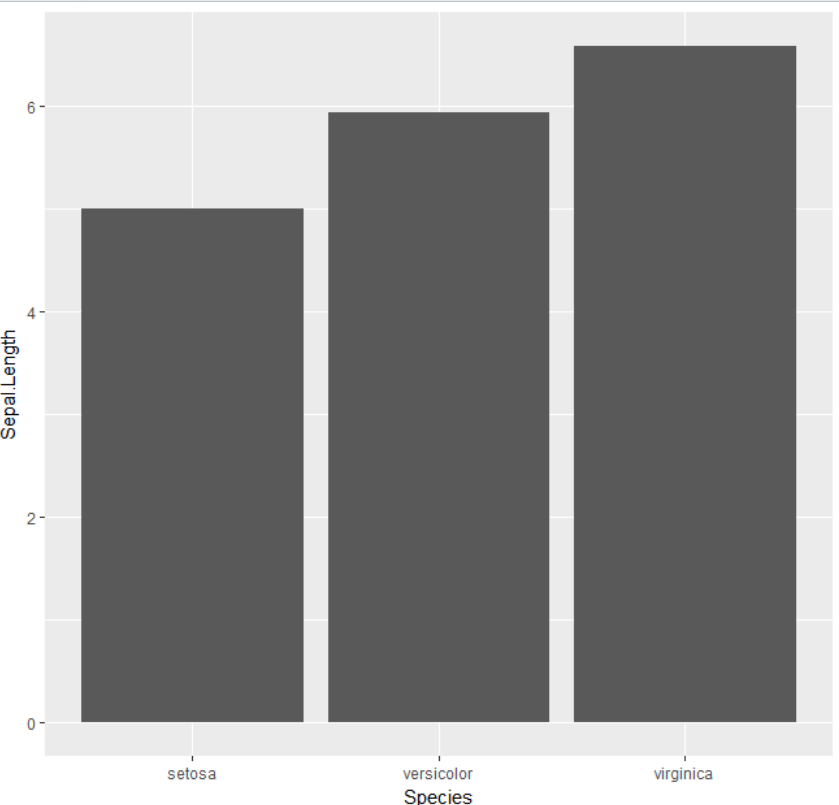
 

* the number 0 in term “group=0” is a dummy variable, hence it has no meaning; any other number can be used as well
* this term is only to indicate for R that it has to take the rel. frequency among all x variables, so that we get the following plot:



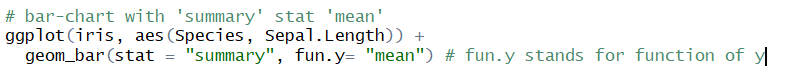
### Example geom\_bar & stat Function with *Iris data frame*

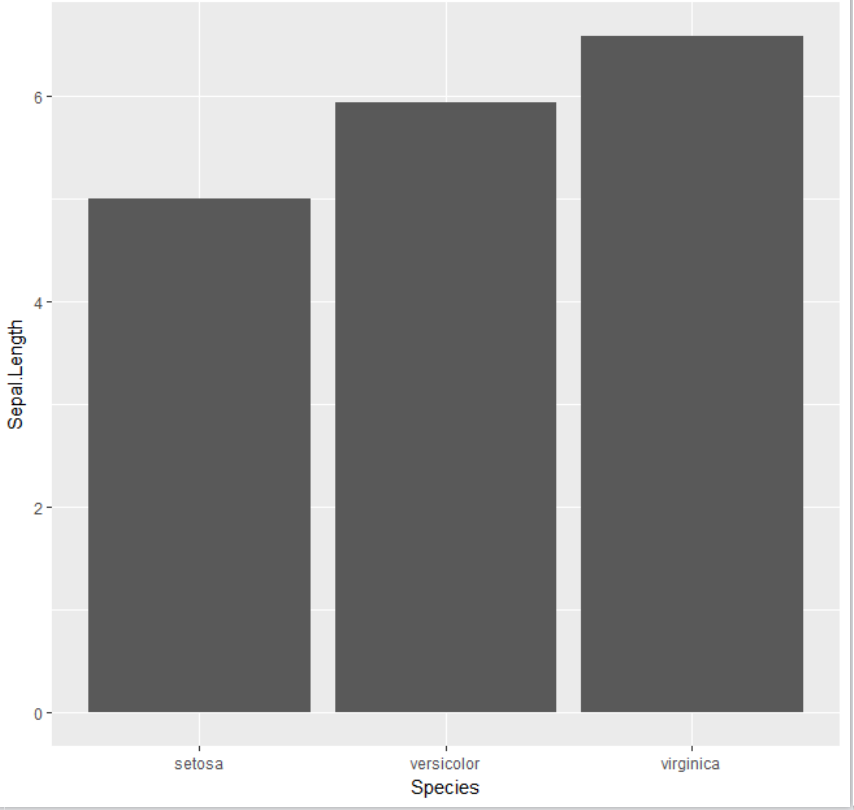




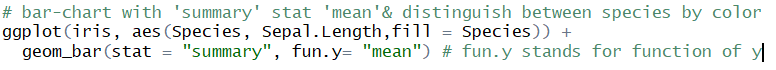
* **if I don’t specify summary stat, by default R uses the mean as a summary stat**

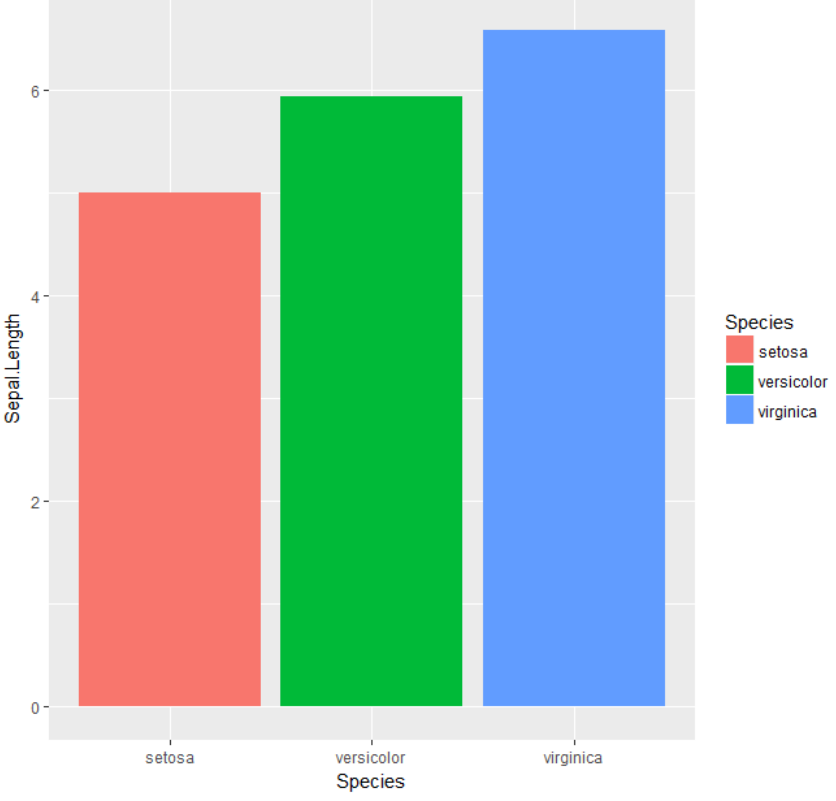
Specification of **summary stat**:

1. mean:   


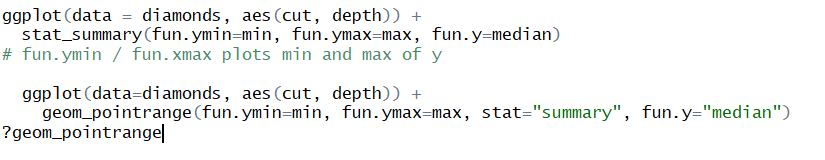


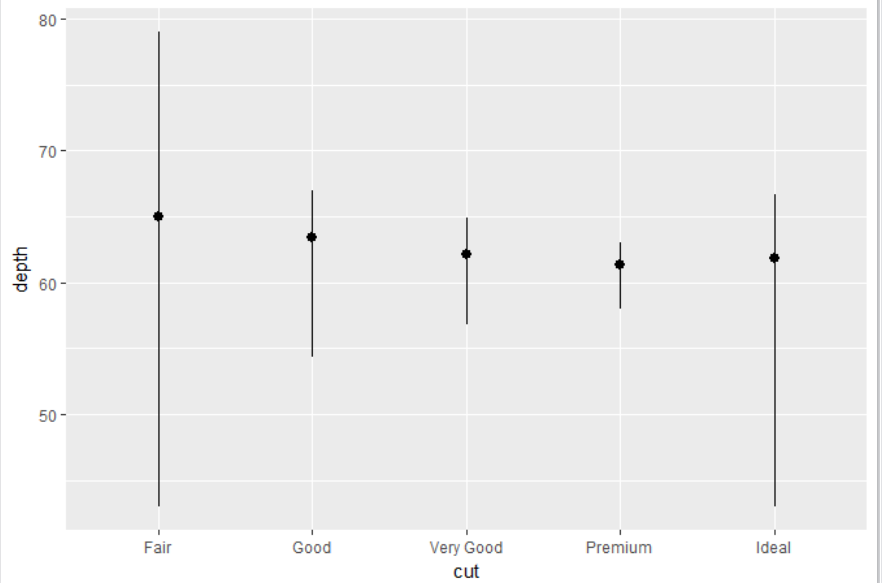
* so what happens her w/ stat and fun. (=function) is that we command to take the mean of the y variable (this is what the functions says) and then “summarize”/aggregate it given x (this is what the summary function says)





### Interchangebility of stat\_Function and geom\_Function: stat\_summary (fun.ymin &max) & geom\_pointrange ( *p. 45 R for Data Science, Exercise 1)*



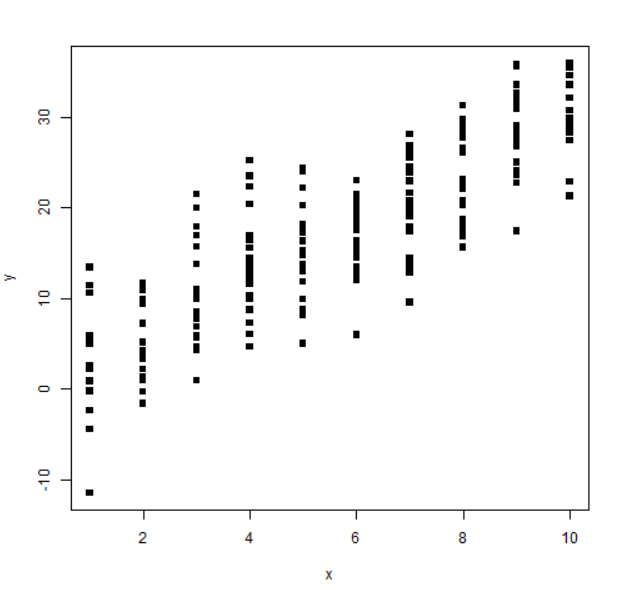


* note that both codes are identical except for the stat definition in the geom\_pointrange code
  + this has to be done, as the summary stat is not a default of the geom pointrage and we want to plot the range of y for every x

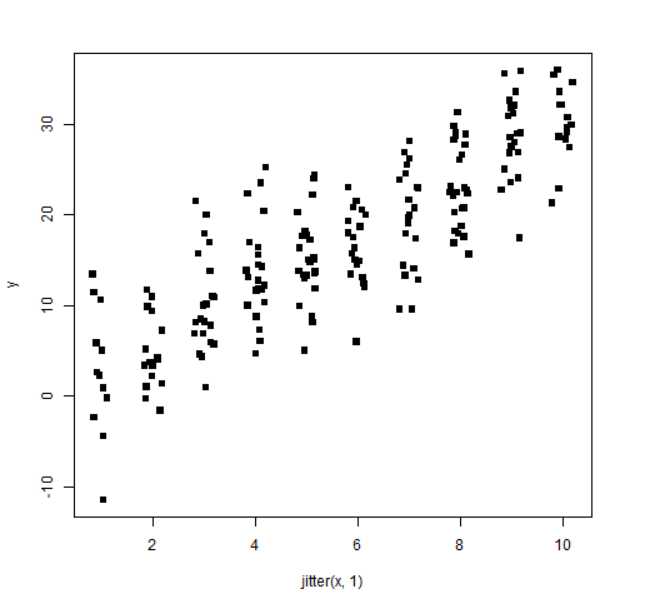
# Position Adjustment

## 7.1 Position Adjustment & Scatterplots:

* by default, geom\_point has position = “identity”, which places the points exactly where they fall on the gird
  + the problem is, that in order to fit a value to the grid, they are automatically rounded
  + this, in turn, increases the overlap of observations/points in the plot
  + the problem of increased overlapping due to rounding is called **“overplotting”**
* overplotting can make it very hard to e.g. find out where the mass of data is
* therefore, sometimes geom’s positions need to be tweaked [optimieren], when otherwise they would obscure each other [sich gg. verdecken]
  + this is done by jitter: in scatterplot we sometimes randomly jitter the points to **reduce overlapping**
    - e.g.:

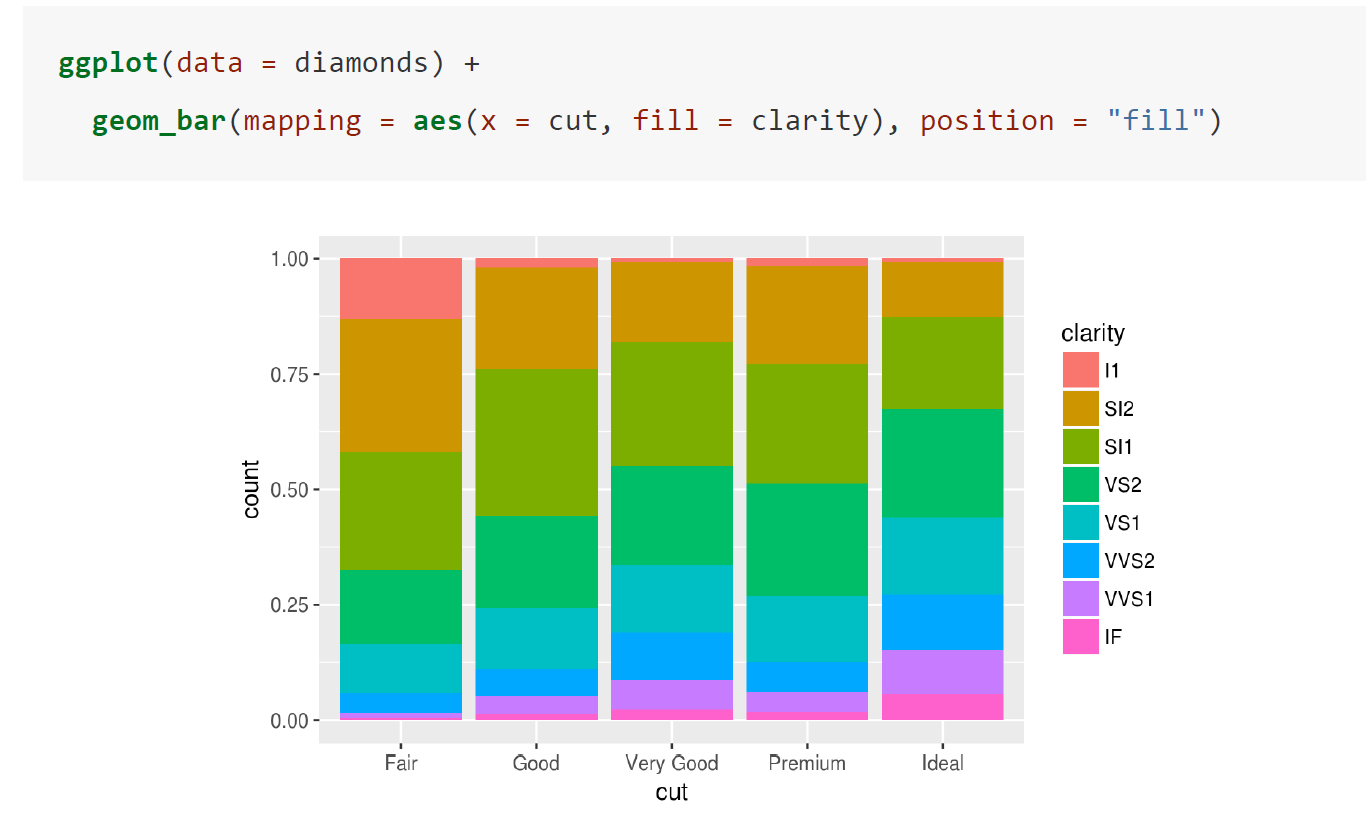


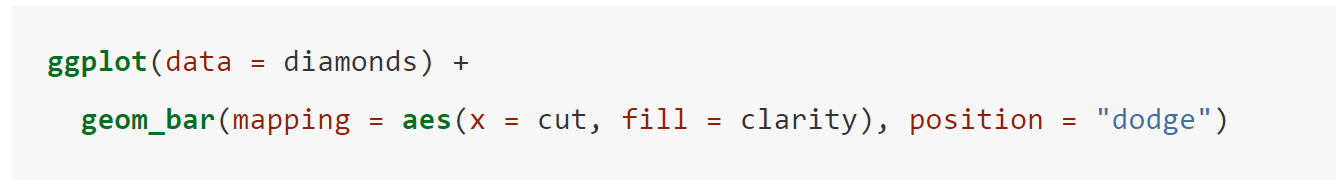
Because the independent variable is only observed at a few levels, it can be difficult to get a sense of the “cloud” of points. We can use jitter to add a little random noise to the data in order to see the cloud more clearly:

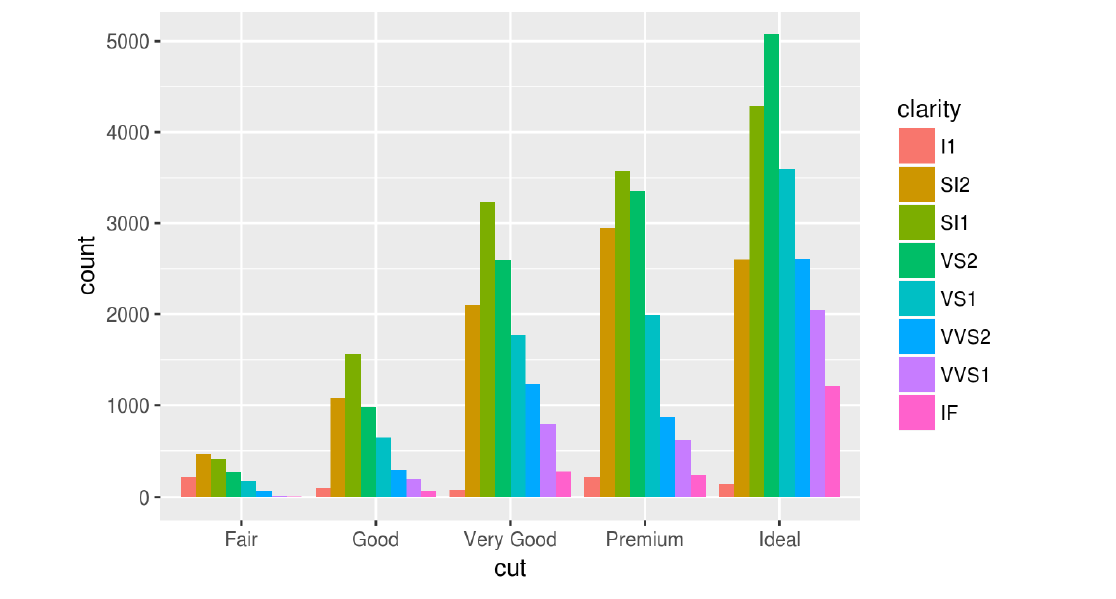


* adding randomness to a scatterplot, makes it less accurate on the one hand, but can reveal important information at the same time: e.g. reveal cluster

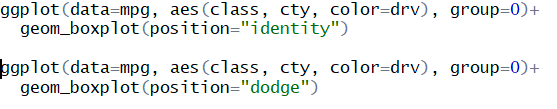
## 7.2 Position Adjustment and Bar Plots

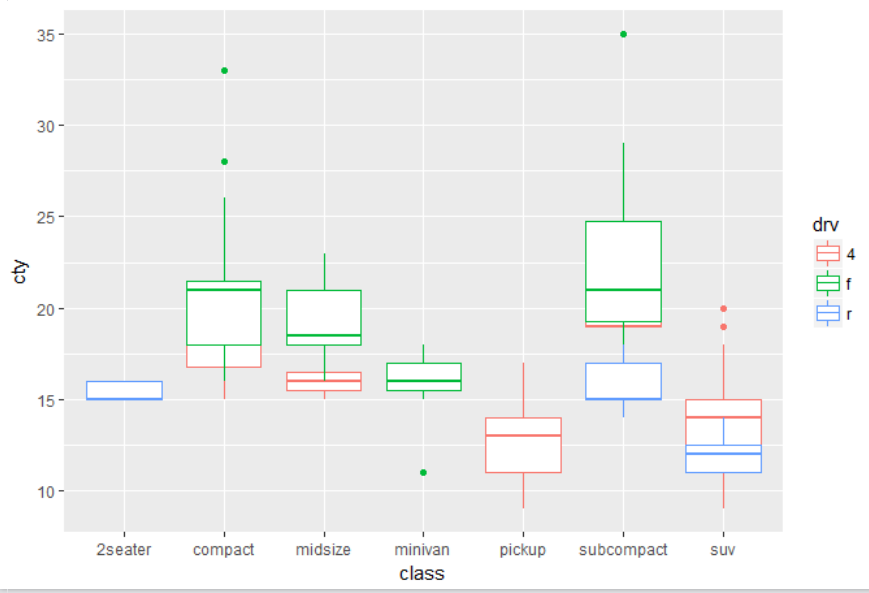
* position = “fill”: makes each set of stacked bars the same height, which is useful for comparison:   
  
* position = “dodge”: places objects directly besides each other:



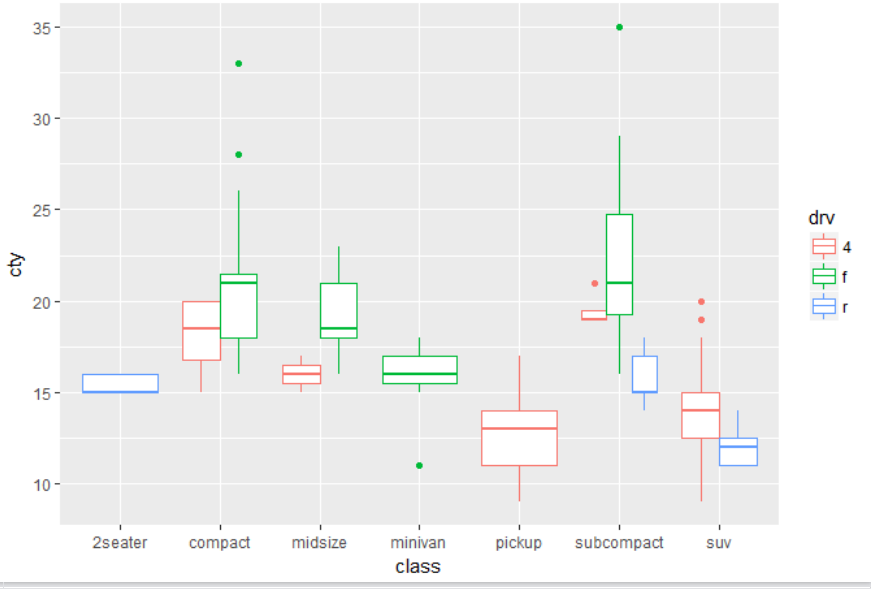


## 7.3 dodge vs. identity in a boxplot:





*position = “identity”*

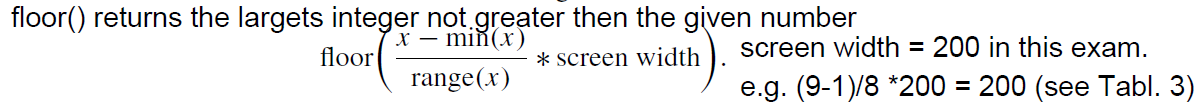


*position = dodge*

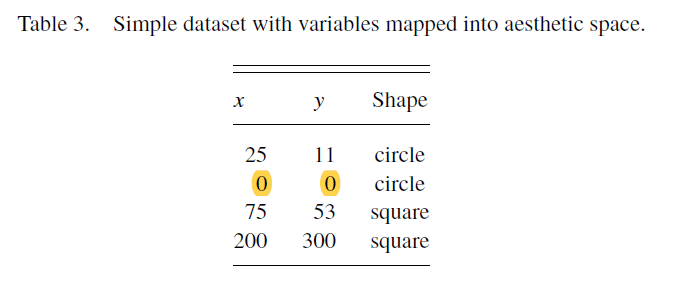
* one can see that in position = identity, those boxplots that rever to the same combination of attributes overlap; this is avoided with position = dogde

# Scales [Skalen]

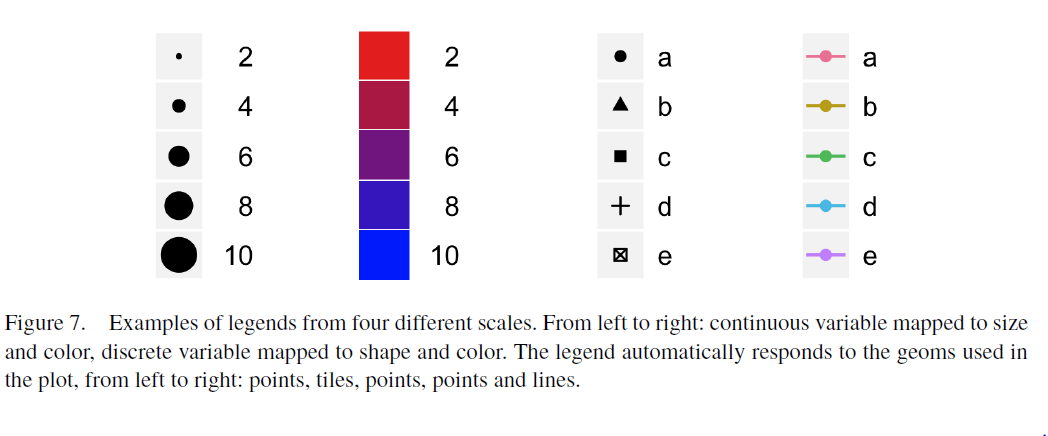
* a scale is a ***function*** ***along with its inverse*** and a ***set of parameters***
  + the inverse function is used to draw a guide so that you can read values from the graph
  + guides are either axes or legends
* a scale ***controls the mapping from data to aesthetic attributes***, which is why we need ***one*** ***scale for each aesthetic property used in a layer*** 
  + during the scaling process ggplot2 ***automatically*** assigns a unique level of aesthetic (e.g. a unique color) to the unique values of a variable
  + so once you have mapped an aesthetic to a variable, ggplot 2 takes care of the rest: it selects a reasonable scale to use with the aesthetic and it constructs a legend that explains the mapping between levels and values
  + for x and y axis ggplot 2 of course does not provide a legend as the axis line itself acts as a legend
* Scaling actually occurs in 3 parts:
  + transforming mapping:
    - scale transformation occurs before stat transformation, so that statistics are computed on the scale-transformed data
    - transformation is only necessary for nonlinear scales, because all statistics are location-scale invariant
  + training:
    - after the statistics are computed, each scale is trained on every faceted dataset (a plot can contain multiple datasets, e.g. raw data and predictions from a model)
    - the training operation combines the ranges of individual datasets (e.g. in table 3 for x: [0,300]) to get the range of the complete data
    - if ***scales were applied locally***, ***comparisons would only be meaningful within a facet***
  + mapping:
    - finally, the scales map the data values into aesthetic values
    - ***data units*** need to be ***converted to aesthetic units***, or more specifically numbers measured in data units need to be converted to ***numbers measured in physical units/pixel coordinates***, hence things that the computer can display:
    - for example, to convert a continuous data value (e.g. as in Table1/ 2) to a horizontal pixel coordinate we need a function as follows:



* + - in ***this*** example the x-position is **scaled to [0, 200]** and the **y-position to [0, 300]**
    - ***these* transformations are the responsibility of scales**



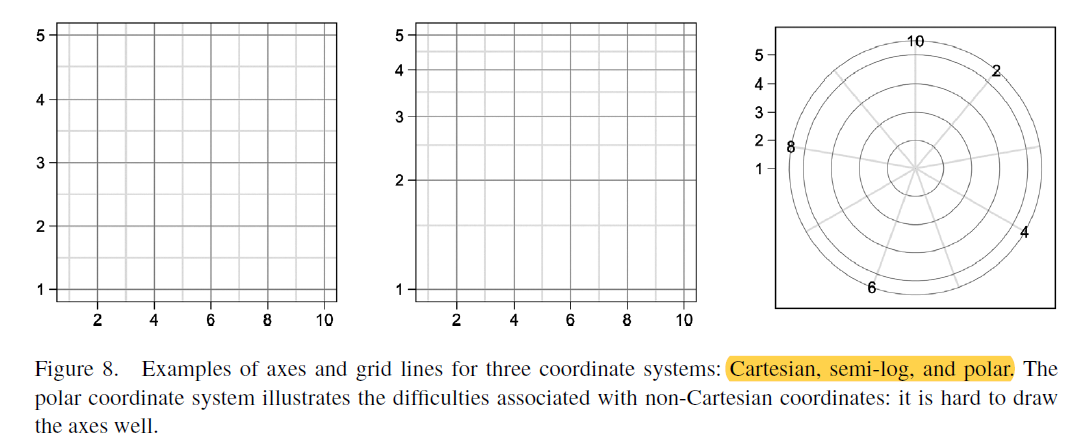
* + - the procedure is similar for other aesthetics, such as shape: in the example above the “a” is mapped to a circle and “b” is mapped to a square
* the legends associated w/ some scales are illustrated in Fig. 7:



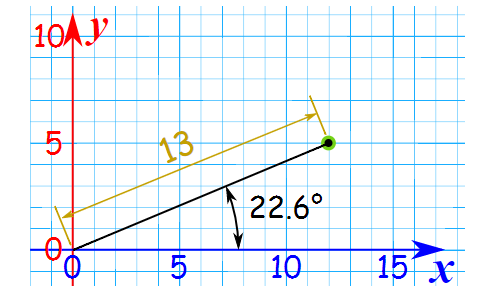
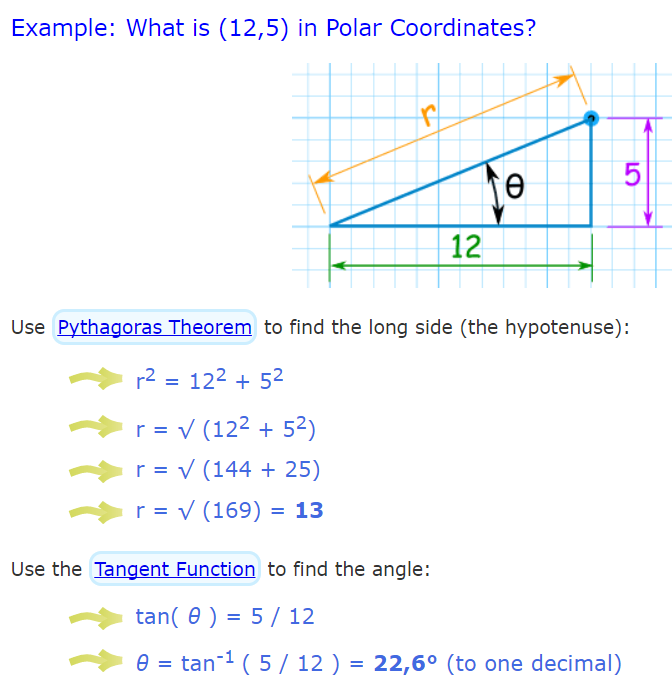
* scales typically ***map from a single variable to a single aesthetic***, but there are exceptions; for example we can create redundant mappings by the same variable to multiple aesthetics which may be useful if a graphic works in both, color and black & white
* scaling is performed before statistical transformation

# Coordination Systems (or coord)

* maps the position of objects onto the plane of the plot
* position is often specified by two coordinates (x,y), but in R any number of coordinates is possible
* the ***cartesian coordinate system*** is the most common coordinate system for 2 dimensions, whereas ***polar coordinates*** and various map projections are used less frequently
* for higher dimensions, we have parallel coordinates (a projective geometry), mosaic plots (a hierarchical coordinate system) and linear projections onto the plane
* coordinate systems affect all position variables simultaneously and differ from scales in that they also change the appearance of the geometric objects
  + for example, in polar coordinates, bar geoms look like segments of a circle
  + additionally, scaling is performed before statistical transformation, whereas coordinate transformations occur afterwards



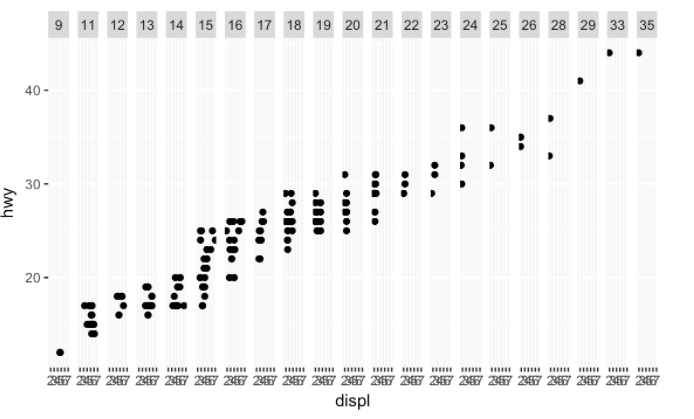
**Excursus polar coordinate system:**

* in contrast to cartesian coordinates, in using polar coordinates we mark a point by how far away it is from the origin and by what angle it has:   
  
* so, to convert from cartesian to polar coordinates and vice versa we use rules of a rectangular triangle
* **to convert from cartesian coordinates (x,y) to polar coordinates (r,** *θ***):**
  + **for r:** Satz des Pythagoras  
    x² + y² = r²
  + **for** *θ:* tan(*θ)  
    tan (θ) = y/x  
    ⬄ tan-1 = (y/x)*
* **to convert from polar coordinates (r,** *θ***) to cartesian coordinates (x,y)  
  for x:** cosinus (*θ*) **cos(***θ***) = x/r**

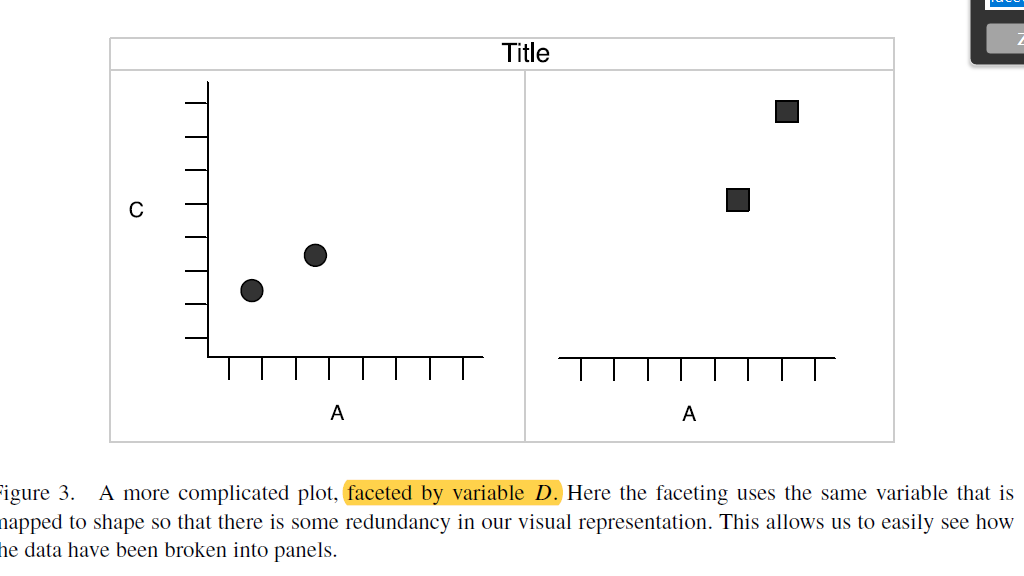
**for y:** sin(*θ*)  
sin(*θ*) **=** y/r

# Faceting

* we have already seen that one way to add additional variables to a plot, is to map them with an aesthetic
* another way, particularly useful for categorical variables is to split a plot into facets
  + a facet is a subplot that each display one subset of the data
* a faceting specification describes which variables should be used to split up the data, and how they should be arranged
  + to **facet a plot** by a **single variable** use: > facet\_wrap(‘formula’)
    - first argument of facet wrap should be a “formula”
      * here “formula” is the name of a specific data structure in R, and is not a synonym for “equation”
      * a formula is created with ~ followed by the name of a variable:
      * > facet\_wrap(~ <VARIABLE>)
      * **the variable that you pass to > facet\_wrap should be discrete**
        + if one uses a continuous variable, R treats this variable as a discrete variable and creates facets for all unique values

for example:  


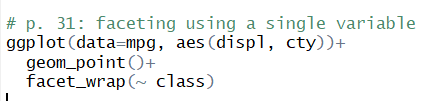
* + - * one can furthermore decide by how many rows/columns one wants to facet using > nrow, > ncol
  + to **facet a plot** on the combination of 2 variables use: > facet\_grid(‘formula’)
    - with 2 variables the formula has following syntax:
    - > facet\_grid(<VARIABLE I> ~ <VARIABLE II>)
      * variable I one the left determines rows, variable II, hence right variable, determines columns of the faceting result
      * one should always put the variable with more unique attributes in the column as it is easier to read when one has more columns but less rows, rather the other way around
    - this formula can also be used, if one prefers to not facet in the rows or columns dimension; then one uses a . rather than a variable:
    - > facet\_grid(. ~ <VARIABLE II>) or   
      > facet\_grid(<VARIABLE I> ~ .)
* faceting vs. aesthetic mapping (e.g. by color):
  + whether one should use faceting or aesthetic mapping to subset a dataset, depends on how wants to proceed further
  + e.g. if one needs to examine each subset in greater detail, faceting is clearer
  + e.g. if one wants to compare subsets, aesthetic mapping might be more useful
  + furthermore, the larger the dataset, the more useful faceting gets
* for example:

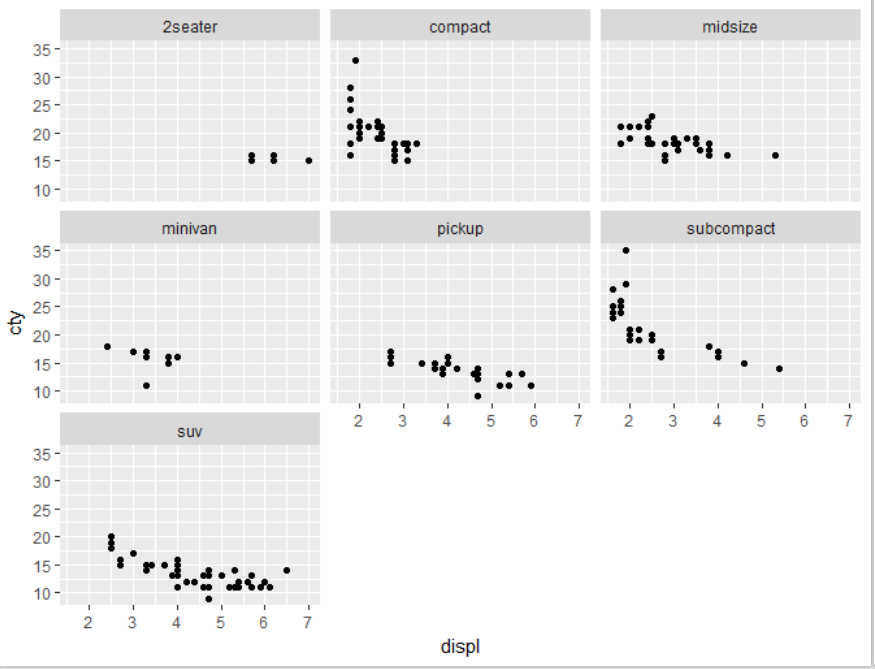


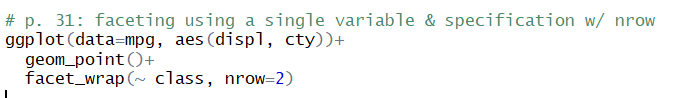
* it is a powerful tool when investigating whether patterns are the same or different across conditions

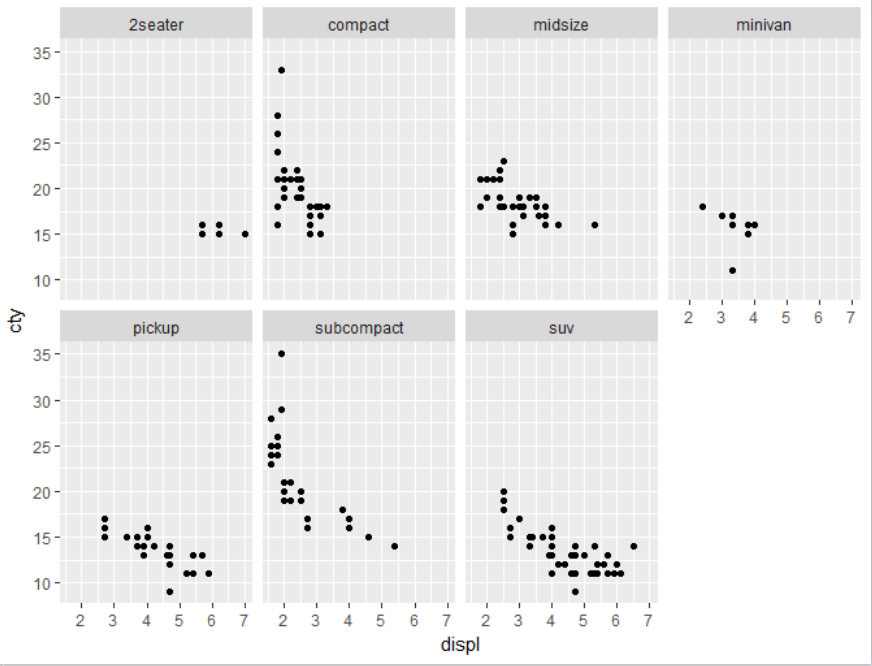
**Examples for faceting:** *p. 31 f. R for Data Science*

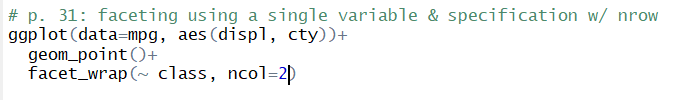
*Faceting w/ single variable:*

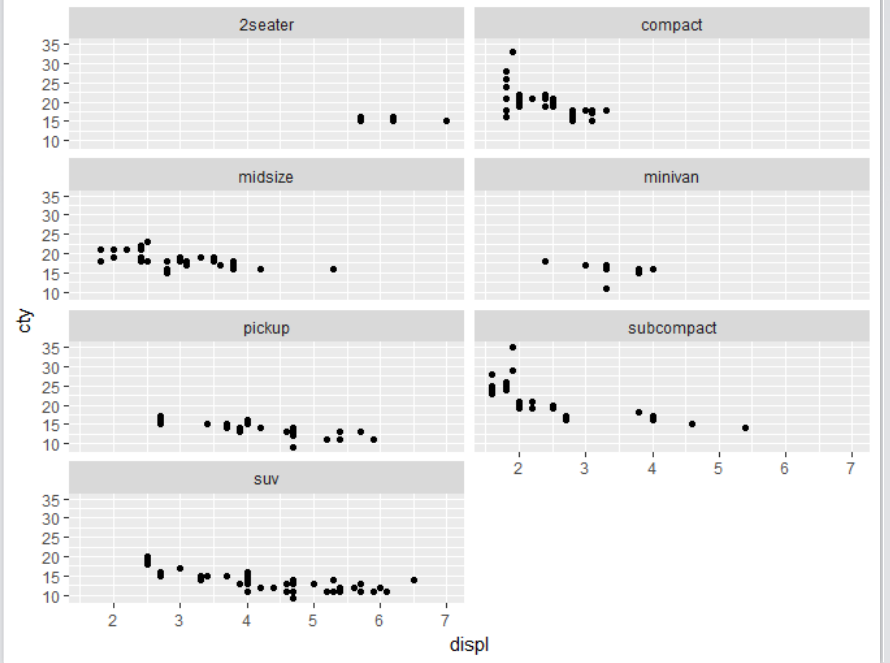




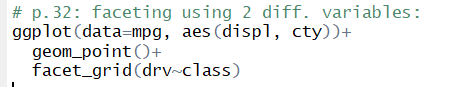


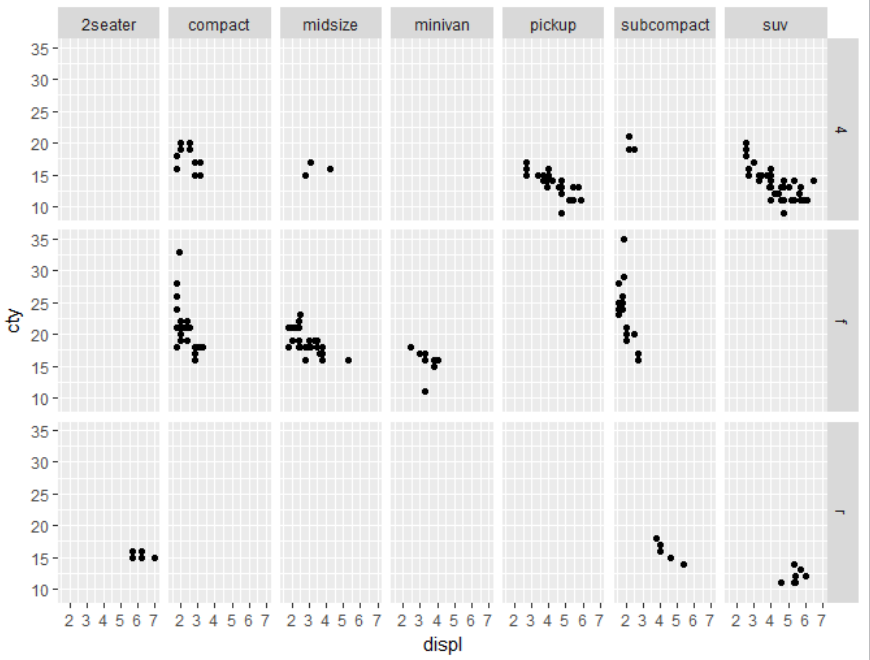




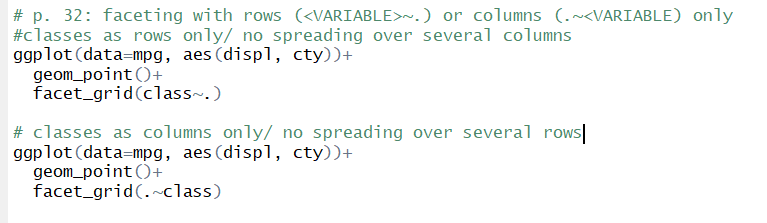


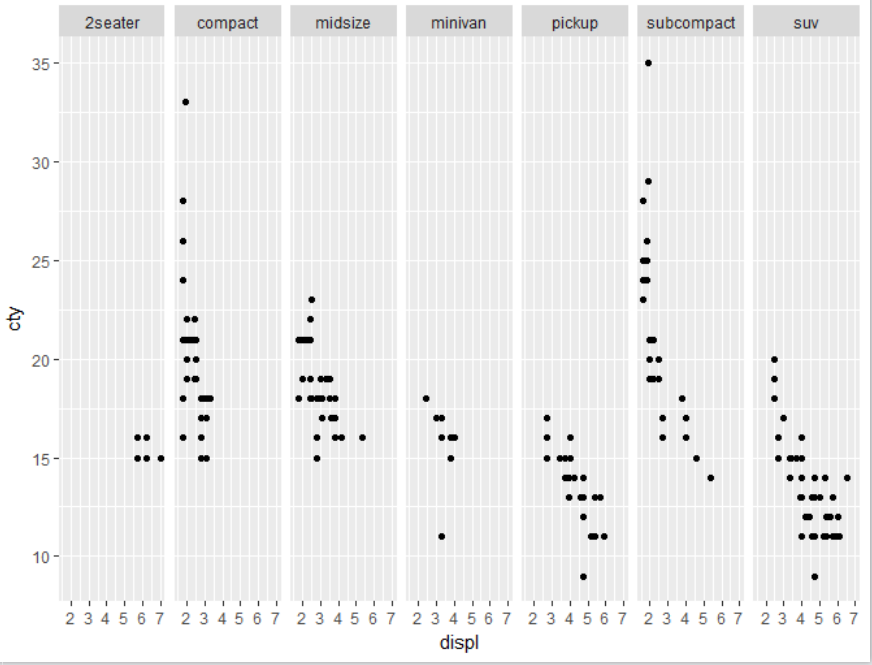
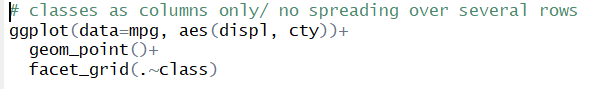
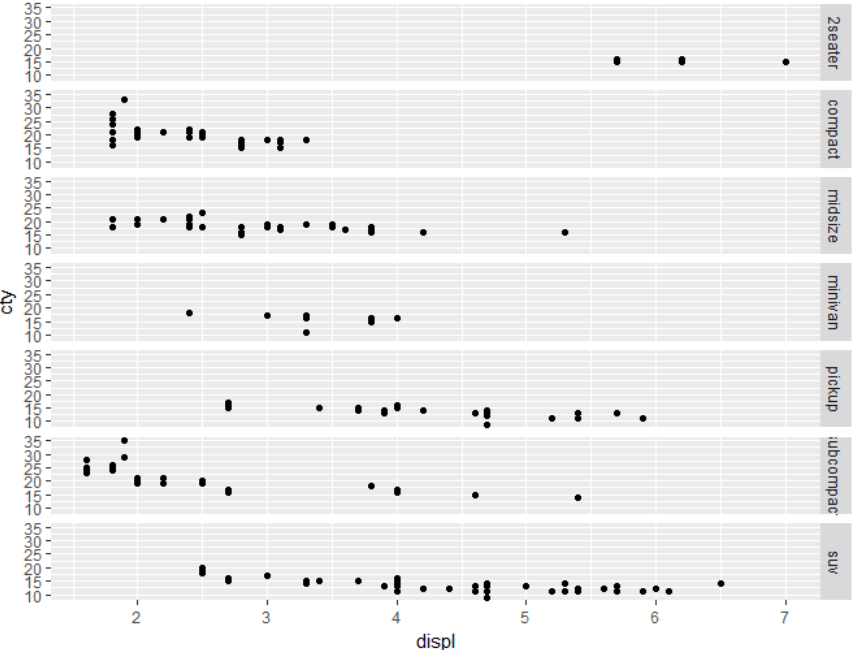
*Faceting w/ 2 variables:*



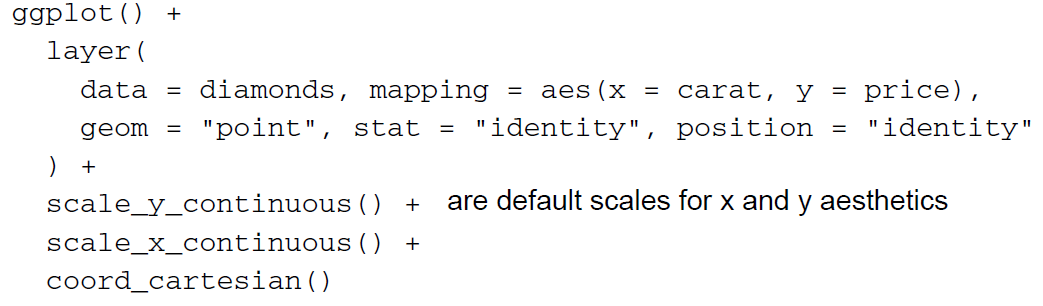
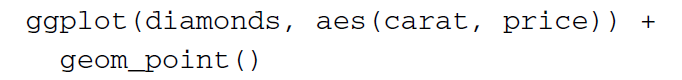
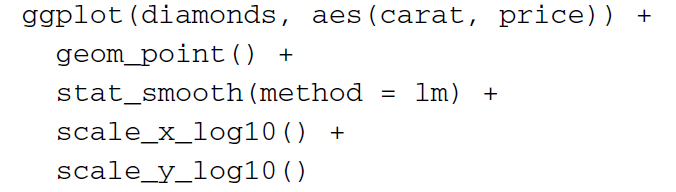
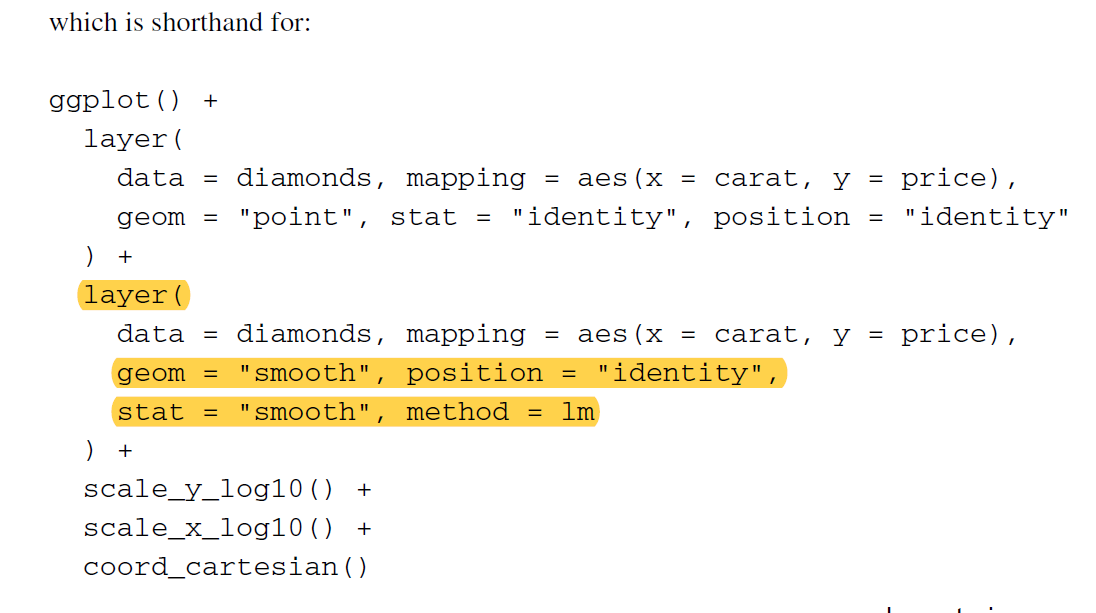


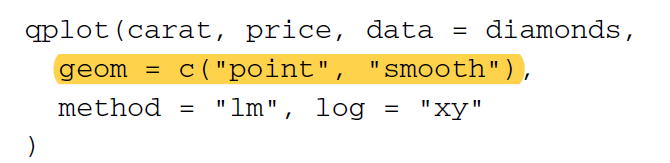
*Faceting w/ rows or columns as end results only:*





# The ggplot syntax & hierarchy of defaults

* the 5 major components of a layer allow us to completely and explicitly describe a wide range of graphics
* **however:** having to describe every component, every time is tiresome
  + **the hierarchy of defaults simplifies this work of making a plot**
* general ggplot syntax (w/out making use of defaults):  
    
  ggplot () +  
  layer (data = ‘name of dataframe’, mapping = aes(x= ‘name of x-variable’, y=’name of y-variable’), geom = “’name of geom”, stat = “’name of stat”, position = “’name of position”)  
  + scale\_y\_’name of scale’ () +  
  scale\_x\_’name of scale’ () +  
  ‘name of coordination system’  
  for example:   
  
* we **start with ggplot()** to create a new plot object
* **we then add all the other components described above:** 
  + a single layer specifying the data, mappings, geom, stat and position
* **yet, intelligent defaults allow us to simplify this specification in a number of ways** 
  + **xor geom & stats**: one needs to specify either geom or stats
    - each geom has a default stat and vice versa
  + **cartesian** coord. system is used by **default**
  + **default scales** will be added depending on aesthetics and type of variable
  + **default** position will be added depending on geom
* so the exemplary code above can be ***reduced to***:  
  ggplot () +  
  layer (data = diamonds, mapping = aes(x= carat, y= price), geom = “point”)
* this can further be reduced to:  
  
  + so typically we will specify a default data set and mapping in the ggplot call and use a short-hand layer such as geom\_point
* **any aesthetics, dataset, scale, stat, position or coord. system explicitly specified in the layer will overwrite the default**
* for example:  
    
  
  + even with many defaults, the explicit grammar syntax can then get verbose [langatmig] and usually spans multiple lines
  + this makes it difficult to rapidly experiment with different plots, which is very important when searching for revealing graphics
* for this reason the grammar of ggplot2 is supplemented with qplot (short for quick plot)
  + qplot makes strong assumptions to reduce the amount of typing needed
  + the qplot functions assumes that multiple layers will use the same data and aesthetic mappings, and **defaults to creating a scatterplot**
  + the exemplary codes could be rewritten as a qplot as follows  
    for simply scatterplot:  
    qplot(carat, price, data = diamonds)

for scatterplot w/ log-transformation:  


* **note:** the geom argument can take a vector of geoms, which are added sequentially to the plot
* **everything else is taken to be a layer to the plot**

# Implications of the layered grammar

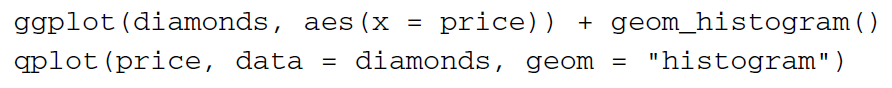
## 1. Histograms

* the histogram is rather special:
  + in general: a histogram is a ***combination of binning stat and bar geom*; and not a single geom itself**
  + it thus maps an aesthetic (bar) to a variable created by a statistic (bin count), and raises some issues regarding parameterization and choice of defaults
* a histogram can be created in 2 ways:
  + with geom “bar”, stats “bin” and position “stack” [stapeln]
    - for example:

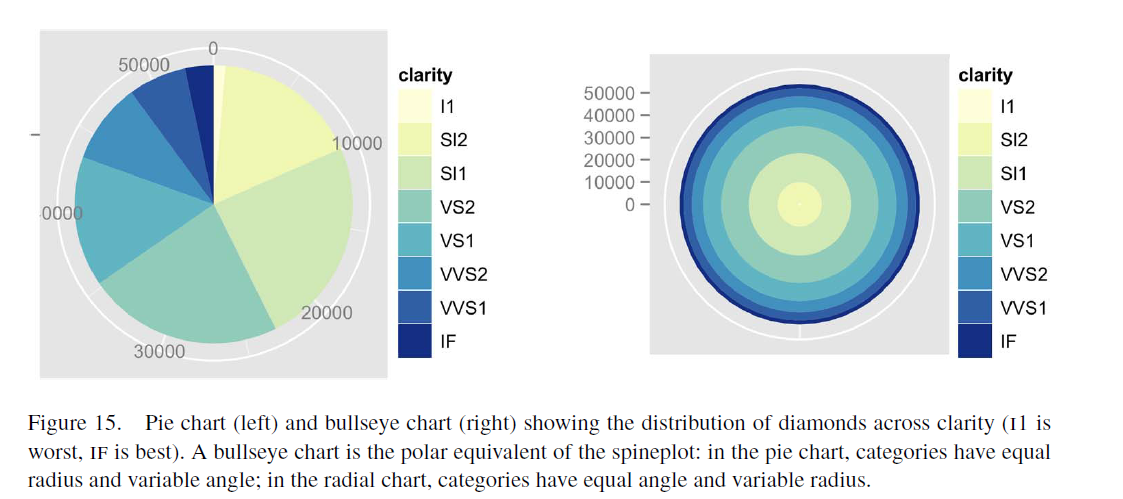


* + the term mapping = aes(y=..count..) relates to the absolute frequency
    - the 2 dots are a visual indicator highlighting that the variable is not present in the original data, but has been computed by the statistic (bin)
    - furthermore, there are other variables produced by the bin statistics that can also be used:  
      aes(y= ..density..):

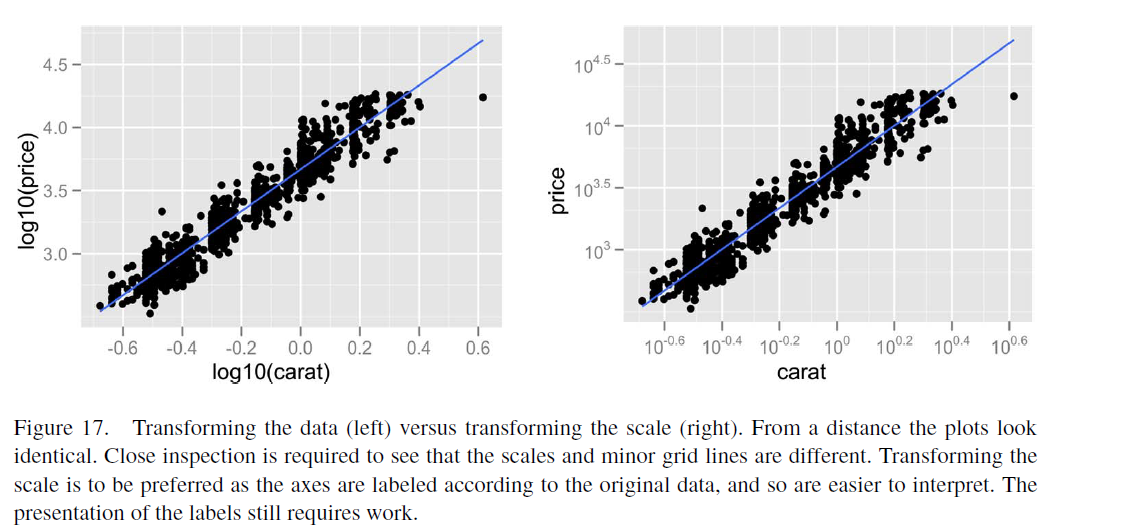
aes(y= ..density../sum(..density..))  
aes(y=..count/max(..count)) [normalisiert 🡪 creates histogram w/ bar range [0,1])

* + or with geom ”histogram”
  + in order to avoid having to specify the same composition of the histogram every time it is created, a histogram geom has been created
    - it is an alias to of the bar geom, with a default bin stat and a default mapping of bar-height to bin count (the latter is indicated by aes(y= ..count..))
  + the same plot can therefore also be created with:   
    or
* the choice of bins/bin width is very important for histograms
  + ggplot 2 alsways uses 30 bins as default, regardless of the data
  + the histogram geom is therefore parameterized by bin width (as opposed to # of bins, as is common elsewhere in R)

## 2. Polar coordinates

* the polar coordinate system is used to produce pie charts and radar plots
* the polar coordinate system parameterizes the 2-dimensional plane in terms of angle *θ* and radius r (for conversions see **Excursus polar coordinate system)**
  + we can choose whether a variable is mapped to angle or radius
* polar coordinate system might be especially useful for cyclical data, because we can aggregate the periods
* in the grammar a pie chart is a stacked bar geom drawn in a polar coordinate system, with height of the bars as being the angle
* in the grammar a bullseye plot is a stacked bar geom drawn in a polar coordinate system, with height of the bars as being the radius
* for example: 
* they are created as follows:   
  pie chart:   
  ggplot(data= diamonds, aes (x=””, fill = clarity)) + geom\_bar(width = 1) + coord\_polar (theta=”y”)  
    
  bullseye chart:   
  ggplot(data= diamonds, aes(x=””, fill = clarity)) + geom\_bar(width = 1) + coord\_polar (theta=”x”)

## 3. Variable transformations and the 3 places in which they can occur

* there are 3 ways to transform values in ggplot2:
  + by transforming the data
  + by transforming the scales
  + by transforming the coordinate system
* transforming the data or scales produces graphs that look very similar, but the axes (and grid lines are different):
* for example:   
  

transformation:   
x 🡪 log10(x)  
-0.6 🡪 log10(-0.6)

log10(y) = x  
⬄ y = 10x

e.g. -0.6 = 10^x  
🡪 math. nicht möglich

* + this is due to when transformation has been done on scales, the stats are done on e.g. log scales with the original data, keeping the original variables
  + when transformation is done on data itself, default linear scale is kept, and stats are -run on this scale   
    🡪 in order to read the correct value, axes are changed
* transforming a coordination system does something quite different
  + the coordinate transformation occurs at the very end of the plotting process and alters the appearance of geoms
* for example: 