Data manipulation and visualization with R

3.2. Data visualization with ggplot2

Introduction

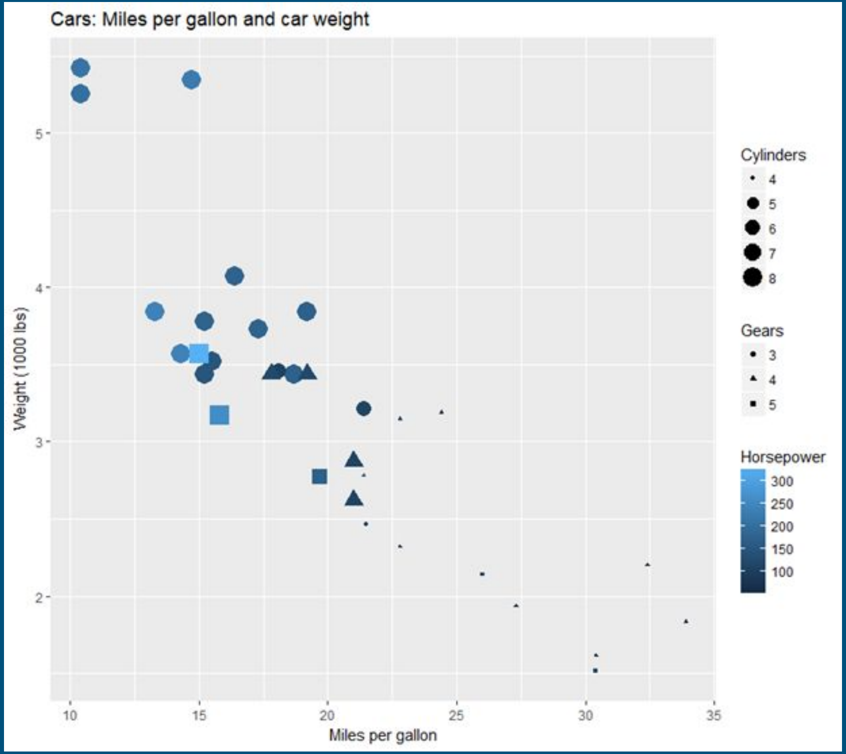
Why ggplot2?

In the last lesson we looked at visualizations with standard R functions for the purpose of data exploration. Now we want to use the package “ggplot2” to create better looking and more versatile graphs for data presentation.

This graph is a good example of how five different layers of information can be shown at once:

- x coordinates: distribution of “miles per gallon”
- y coordinates: distribution of “weight (1000 lbs)”
- Shape of dots: number of gears
- Color of dots: number of horsepowers
- Size of dots: number of cylinders

ggplot2 supports the integration of several layers of information with significantly less effort (and code) than standard R functions.



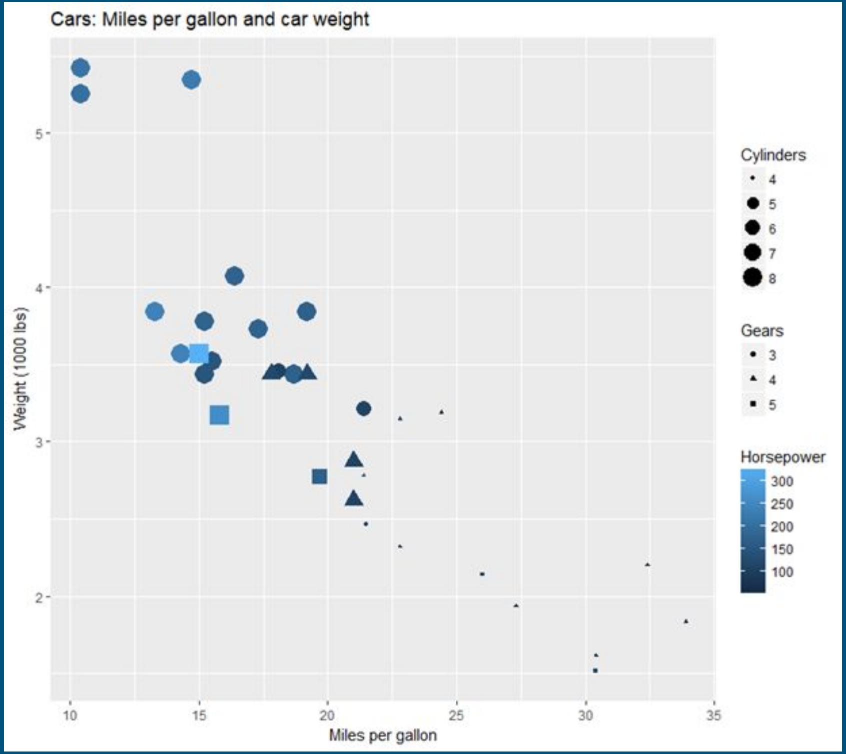
Introduction

What we will learn

This course will give an introduction to the package ggplot2. We will look at the function qplot( ), which is akin to the familiar plot( ) function used for "quick plots".

The function ggplot( ) acts as the base for each graph, but requires a certain "grammar of graphics" (gg) that combines parameters in a similar way words are combined to sentences. Geometries (geom_) are used to create, for example, either a bar plot or a scatter plot from the same data input.

Eventually, we will look at graphical parameters that add visual themes, legends or facets to existing ggplot( ) graphs.



Introduction

Installation guide and online resources

First, we need to install the ggplot2 package. This needs to be done only once.

```
install.package("ggplot2")
```

Next, we add the package to the library. This must be done each session.

```
library("ggplot2")
```

ggplot2 is also part of the package “tidyverse”, which includes “dplyr” as well. Installing the tidyverse package therefore installs ggplot2 automatically.

A comprehensive documentation of ggplot can be found here: https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf.

Furthermore, the “cheat sheet” provides an excellent overview about the most important functions included within this package: https://rstudio.com/wp-content/uploads/2016/11/ggplot2-cheatsheet-2.1.pdf

Quick plots

qplot( )

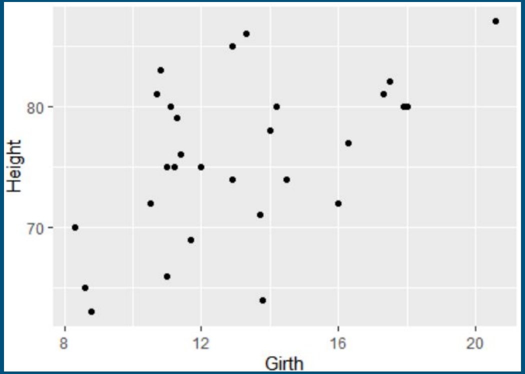
While the ggplot( ) function used in this package brings its own “grammar of graphics”, the qplot( ) (short for “quick plot”) is similar to the plot( ) function we know from standard functions.

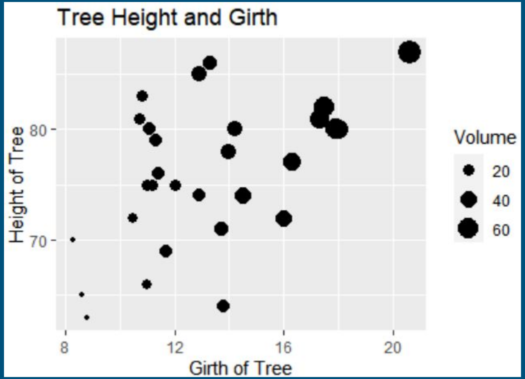
qplot( ) also uses x and y as parameters to create a two-dimensional graph. In this example we specify the dataset as “trees” (a dataset that comes with the package), so the function can find the desired columns without further specification (see first plot):

```
qplot(x=Girth, y=Height, data=trees)
```

The parameters “xlab”, “ylab” and “main” also apply to qplot( ). However, qplot( ) also possesses the parameters “size” and “color” to have a third (or fourth) variable come into play. Notice that the qplot( ) creates a raster background as well as a legend. Even done quickly, the plots are more advanced than the standard plots.

```
qplot(x=Girth, y=Height, data=trees, size=Volume,
  xlab="Girth of Tree", ylab = "Height of Tree", main = "Tree Height and Girth")
```



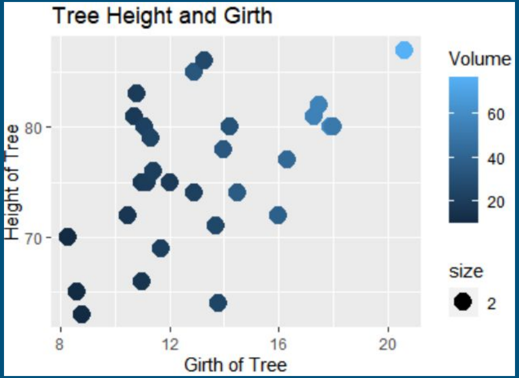


Quick plots

qplot( )

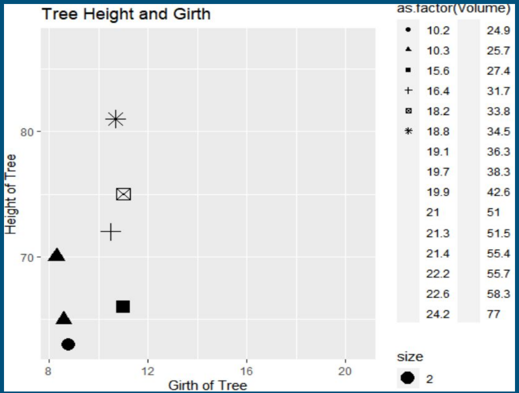
The “color” parameter adds yet another layer of information.

```
qplot(x=Girth, y=Height, color=Volume, data=trees, size=2,
 xlab="Girth of Tree", ylab = "Height of Tree", main = "Tree Height and Girth")
```



The "shape" parameter can change the shape of the dot (similar to “pch” in plot( )). As there is only a limited number of shapes (6), a finite number of entries is needed. For this reason, it is recommended to convert the entry to a factor ("as.factor(x)"). Additionally, choosing a categorical variable with a max of 6 options is recommended, otherwise, a result like the one on the right can be expected.

```
qplot(x=Girth, y=Height, data=trees, size=2, shape=as.factor(Volume),
 xlab="Girth of Tree", ylab = "Height of Tree", main = "Tree Height and Girth")
```



“Grammar of Graphics”

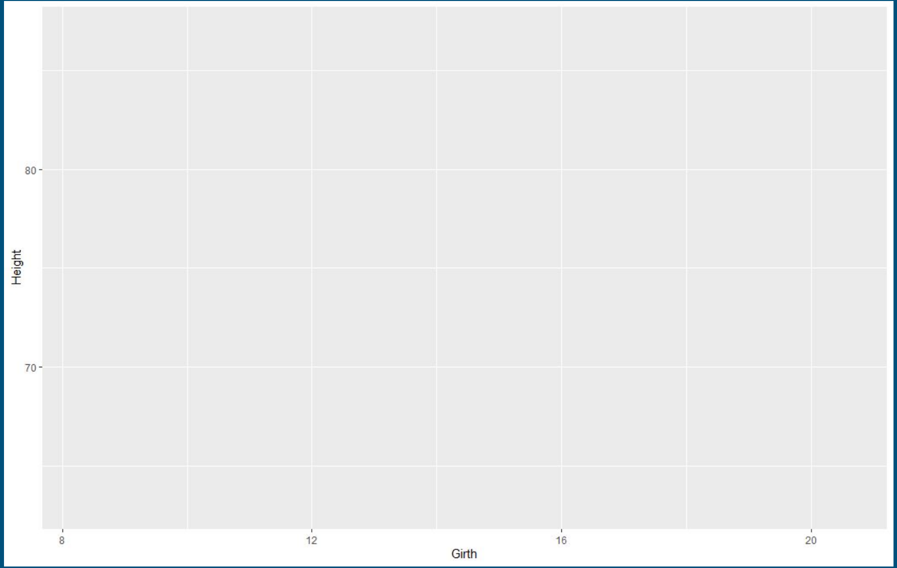
ggplot( )

The package uses a certain “Grammar of Graphics”. A “sentence” usually includes:

ggplot( ) + geom( ) + additional components (themes, facets, labels, …)

First, we look at ggplot( ), which is the main function to build graphics with ggplot2. We need to specify the dataset and the aesthetics (“aes”). Aesthetics are the variables that will be visualized. In this case, we have “Girth” on the x-axis and “Height” on the y-axis. But we will only see an empty plot, as the type of plot is not yet specified by the geom( ).

```
ggplot(data=trees, aes(x=Girth, y=Height))
```



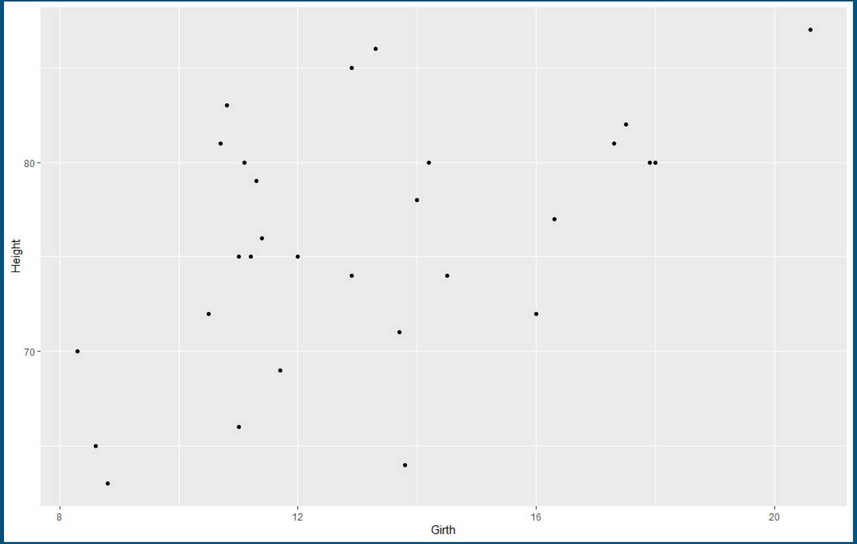
“Grammar of Graphics”

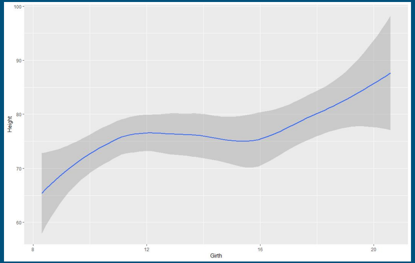
ggplot( ) + geom( )

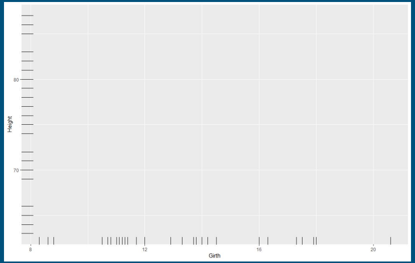
The geoms( ) finally add a type of visualization to the plot. They will be discussed in greater detail in the next chapter. Here are only a few examples using the same aesthetics.

geom_point( ) is the equivalent to a scatter plot created by plot( ). It shows all data points. The plot could be enhanced by adding labels and a title. As the points are not very dense, it might also be useful to increase their “size” a bit. The geom_smooth( ) creates a tendency line, where the gray shaded area indicates uncertainty. The geom_rug( ) only pins the x and y values to their respective axes. This can be useful in addition to a scatter plot or any other plot where individual data points become difficult to make out.

```
ggplot(data=trees, aes(x=Girth, y=Height)) + geom_point()
ggplot(data=trees, aes(x=Girth, y=Height)) + geom_smooth()
ggplot(data=trees, aes(x=Girth, y=Height)) + geom_rug()
```







“Grammar of Graphics”

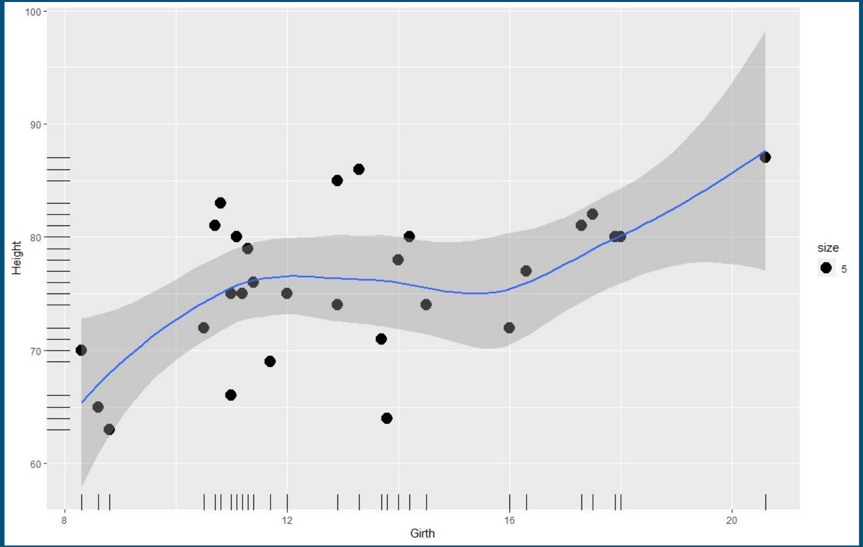
ggplot( ) + geom( ) + geom( ) + ...

The geom( )s can be combined to overlay different layers of information. Here, all three previous plots are combined.

In the code below, we save the ggplot( ) as variable “p”. This makes it easier to combine plots, e.g. p+geom_point( ) or to reuse an essential ggplot( ). By calling “p” in the end, the plot is displayed.

Each geom( ) can have its own aesthetics. Here, we increased the size of the points in geom_point( ). If applied directly to ggplot( ), the size would have affected all geoms( ), creating bold lines and pins, too.

```
p <- ggplot(data=trees, aes(x=Girth, y=Height))
p <- p + geom_point(aes(size=5))
p <- p + geom_smooth()
p <- p + geom_rug()
p
```



“Grammar of Graphics”

ggplot( ) + geom( ) + theme( ) + facets( ).

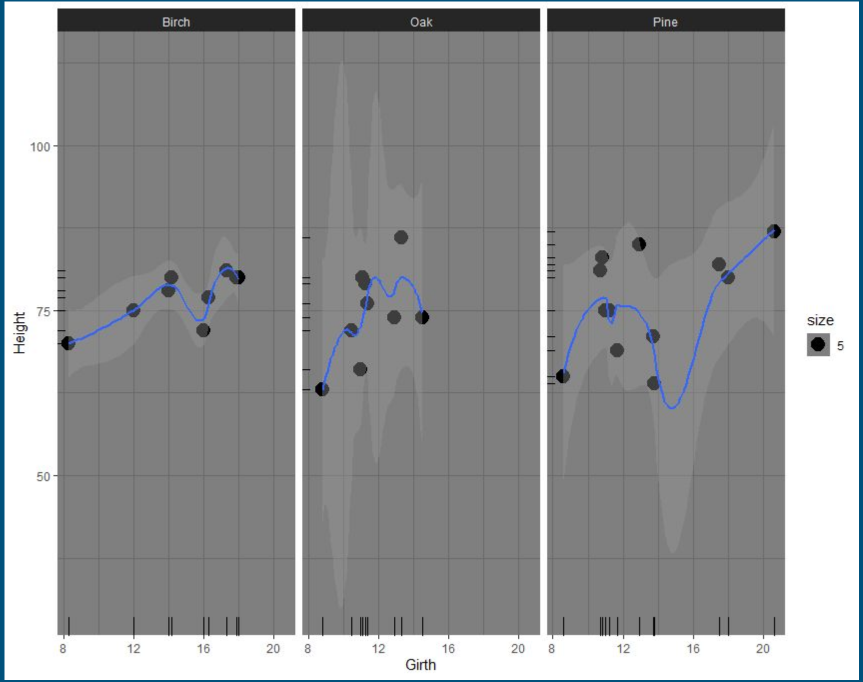
Finally, we add a theme( ) and facets( ).

The theme changes the overall appearance of the plot. In this case we applied a “dark” theme. This is not always the best choice as it might distract from the actual information. The user can also create new themes.

Facets permit comparisons for specified variables. In this case, we created a column with tree species and separated the data by species.

More details on themes and facets will be given later.

```
trees$Species <- sample(c("Birch", "Oak", "Pine"), 31, T)
ggplot(data=trees, aes(x=Girth, y=Height)) +
geom_point(aes(size=5)) + geom_smooth() + geom_rug() +
theme_dark() + facet_grid(. ~Species)
```



geom( )

Overview

In this chapter, we will look at the most useful geoms and give a few examples of how to use them effectively.

geom_point( )	x/y scatter plots
geom_jitter( )	x/y scatter plots
geom_count( )	x/y scatter plots

geom_smooth( )	x/y scatter plots with tendency lines
geom_line( )	x/y scatter plots with connected data points
geom_bin2d( )	x/y heat maps
geom_contour( )	x/y/z heat maps
geom_histogram( )	x frequency charts
geom_area( )	x/y frequency charts
geom_density( )	x frequency charts
geom_bar( )	x, x/y bar plots
geom_boxplot( )	x, x/y box plots
geom_violin( )	x, x/y box plots

geom( )

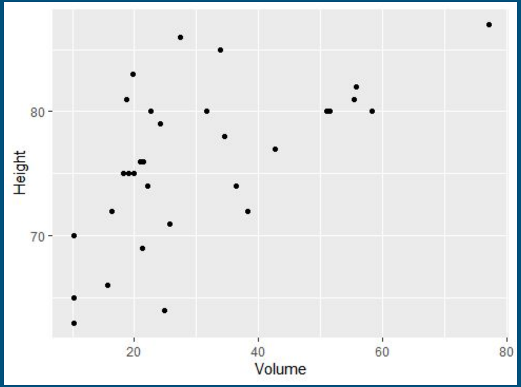
geom_point( )

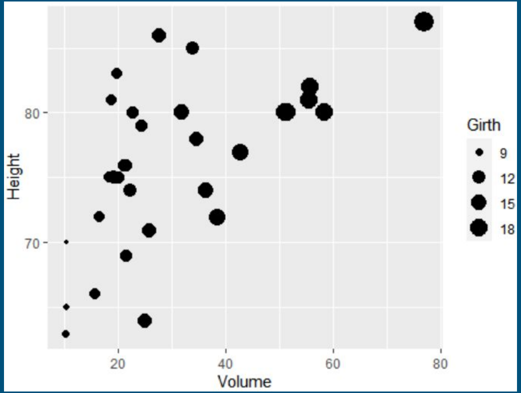
geom_point( ) is the most common way to show the relationship between two variables (correlation). The data type is generally continuous, though categorical variables can also be explored. However, geom_jitter( ), geom_count( ) or geom_bin2d( ) are usually more appropriate for categorical variables. For data with three variables, such as the Height, Girth and Volume of our "trees" dataset, a bubble chart can be used to show the third variable mapped to the size of points, as shown in the second example. Additional layers can be added by "color" or "shape".

Plot Type	Data Type	Dimensions	Purpose
Scatter plot	Continuous X, Continuous Y	2	Showing correlation between 2 variables

```
ggplot(data=trees, aes(x=Volume, y=Height)) + geom_point()

ggplot(data=trees, aes(x=Volume, y=Height, size=Girth)) + geom_point()
```





geom( )

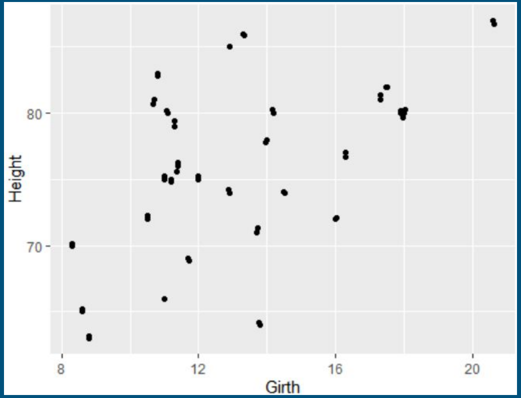
geom_jitter( )

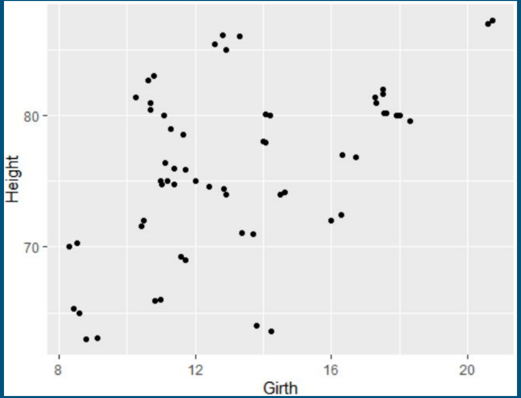
geom_jitter( ) is a variant of geom_point( ) that spreads the data points over a small distance. This is especially useful when many data points overlap, e.g. because there is little variance in the data. The parameters “width” and “height” within the geom_jitter( ) call adjust how much the points spread from their original position. However, this should be avoided when the accuracy of the data is imperative (e.g. physical measurements on an atomic level). In those cases, geom_count( ) might be more appropriate.

Plot Type	Data Type	Dimensions	Purpose
Scatter plot	Continuous X, Continuous Y	2	Allows you to see multiple overplotted data points

```
ggplot(data=trees, aes(x=Girth, y=Height)) + geom_jitter()

ggplot(data=trees, aes(x=Girth, y=Height)) + geom_jitter(width = .5, height = .5)
```

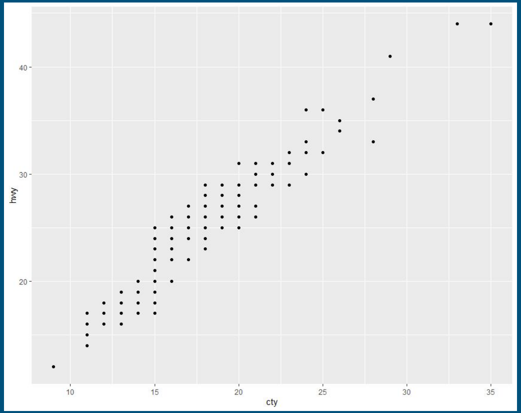




geom( )

geom_count( )

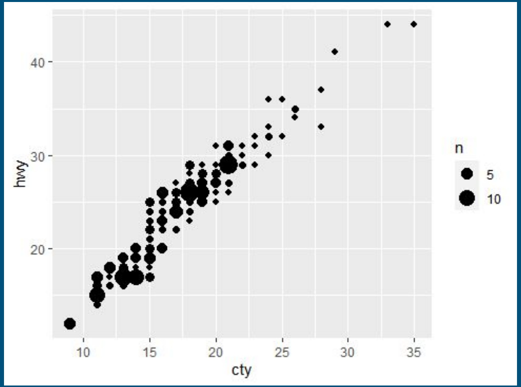
geom_count( ) is a variant of geom_point( ) that counts the number of observations at each point and maps it to that point. It is useful when we have discrete data and overplotting. In this example, we have scatter plot where it is unclear whether points overlap. The geom_count( ) plot, however, shows that many data points are more frequent than others.



Plot Type	Data Type	Dimensions	Purpose
Scatter plot	Continuous	2	Increases circle size when points overlap

```
ggplot(data=mpg, aes(x=cty, y=hwy)) + geom_point()

ggplot(data=mpg, aes(x=cty, y=hwy)) + geom_count()
```



geom( )

geom_smooth( )

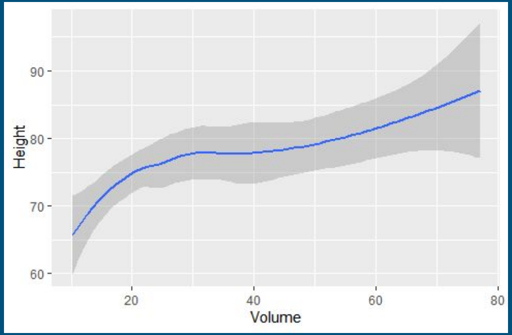
geom_smooth( ) shows a tendency line for an x/y plot. The blue line shows where predicted values would be located, while the gray shaded area indicates uncertainty. The less known data, the higher the uncertainty. Look up the geom( ) in the documentary for further details about the mathematics behind linear models and applicable parameters, as this exceeds this course's scope.

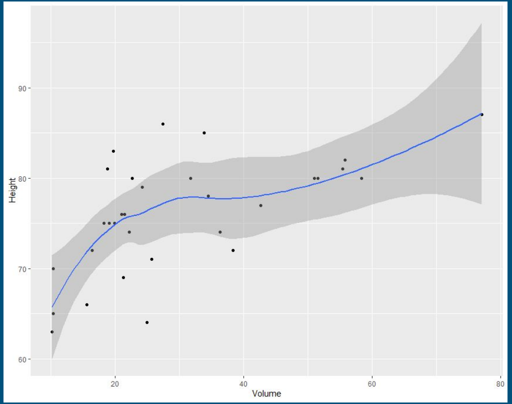
The tendency lines are often combined with scatter plots using geom_point( ) or geom_count( ).

Plot Type	Data Type	Dimensions	Purpose
Line Chart	Continuous Function	2	Showing trends over time

```
ggplot(data=trees, aes(x=Volume, y=Height)) + geom_smooth()

ggplot(data=trees, aes(x=Volume, y=Height)) + geom_smooth() + geom_point()
```





geom( )

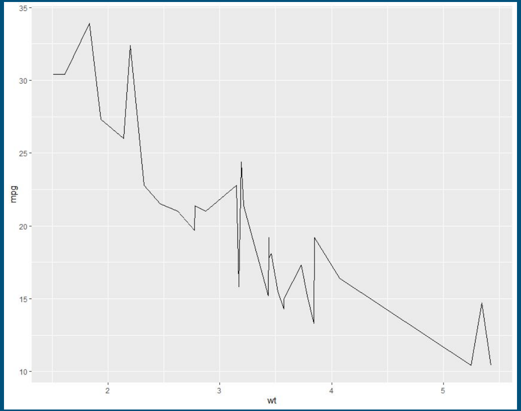
geom_line( )

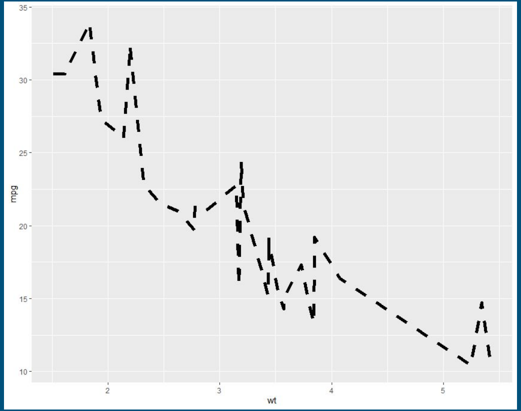
geom_line( ) is mostly used for time series (line charts) to map change over time. It creates a line connecting all data points in the order they appear in the dataset. The optional parameter “linetype” can be changed to e.g. “dashed” or “dotted”. The “size” parameter changes the thickness of the lines, however, this should be used sparingly. It is recommended to change the “size” parameter directly in the geom_line( ) and not as an aesthetic, so it only affects the projection of lines and no other geometries are introduced to the plot.

Plot Type	Data Type	Dimensions	Purpose
Line Chart	Continuous Function	2	Showing trends over time

```
ggplot(data=mtcars, aes(x=wt, y=mpg)) + geom_line()

ggplot(data=mtcars, aes(x=wt, y=mpg)) + geom_line(linetype="dashed", size=2)
```





geom( )

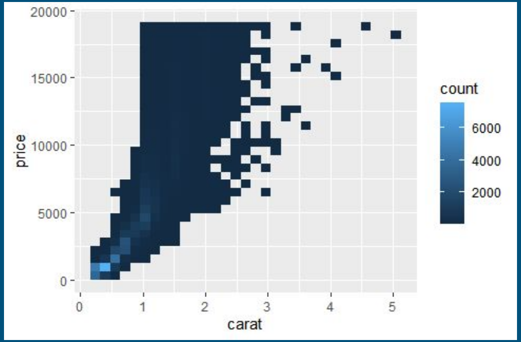
geom_bin2d( )

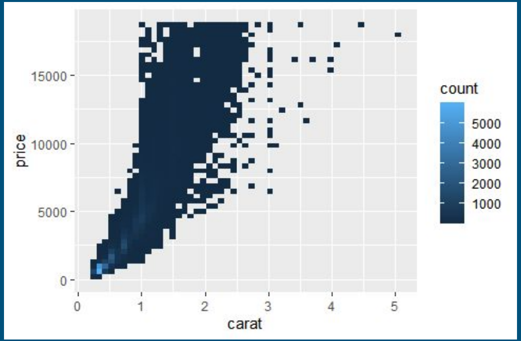
geom_bin2d( ) is a variant of geom_point( ) that counts the number of observations at each point and maps it to that point. It is useful when we have discrete data and overplotting. While geom_count( ) changes the dot size, geom_bin2d( ) indicates higher frequencies with color. You can create a plot where the brighter colors show points that stick out further. The parameter “bins” specifies how many bins per x or y dimension are created. Higher numbers lead to a higher resolution. Bins can be setup with vectors as well: “bin=c(5,50)” creates 5 bins horizontally and 50 vertically.

Plot Type	Data Type	Dimensions	Purpose
2d Histogram and Heatmaps	Continuous bivariate distribution	2	Adding a heatmap of 2d bin counts

```
ggplot(diamonds, aes(carat, price)) + geom_bin2d()

ggplot(diamonds, aes(carat, price)) + geom_bin2d(bins=50)
```





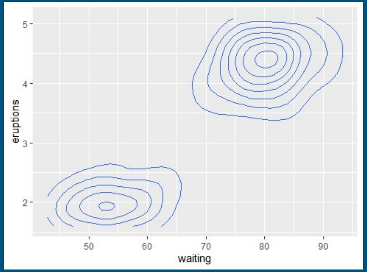
geom( )

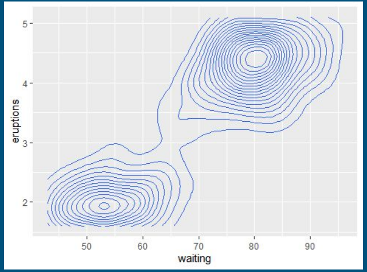
geom_contour( )

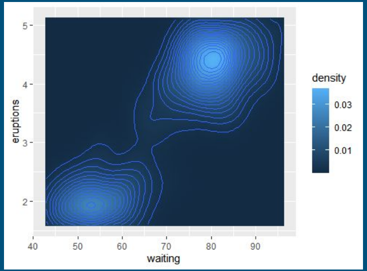
We can use geom_contour( ) to visualize 3d surfaces in 2d. For this visualization, the data must contain only a single row for each unique combination of the variables mapped to the x and y aesthetics. x and y will be shown on a plane from above, while the z variable adds a relief which is shown by lines. You may know this visualization from weather (temperature) or topography (height) maps. "bins" determines how many lines will be shown. A geom_raster( ) using the z variable as a "fill" can be added to visualize an underlying heatmap.

Plot Type	Data Type	Dimensions	Purpose
Contour Plot		3	2d contours of a 3d surface

```
ggplot(data=faithfuld, aes(x=waiting, y=eruptions, z=density)) + geom_contour()
ggplot(faithfuld, aes(x=waiting, y=eruptions, z=density)) + geom_contour(bins=20)
ggplot(faithfuld, aes(x=waiting, y=eruptions, z=density)) + geom_contour(bins=20) + geom_raster(aes(fill=density))
```



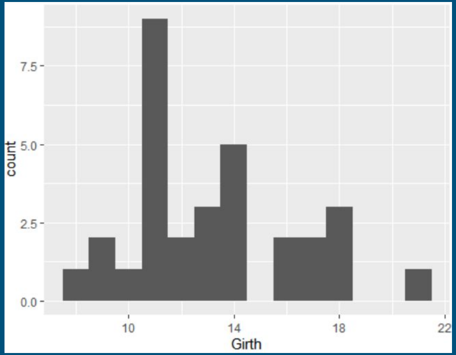




geom( )

geom_histogram( )

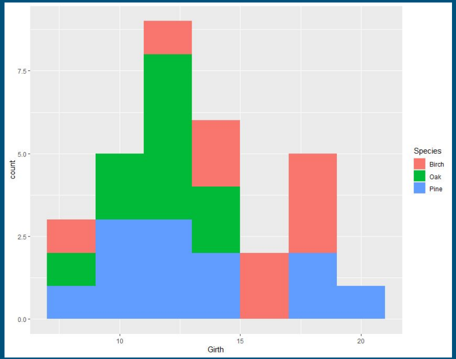
geom_histogram( ) is useful to display the distribution of a data set with a single variable (univariate). A range is divided into a series of intervals, where the “binwidth” parameter indicates how many neighboring x values are grouped into one bin. For example, the second plot includes two x values into every bin, leading to fewer bins in total. The “fill” aesthetic applied to the geom( ) creates stacked bins, thus showing frequencies of different categories within the dataset.



Plot Type	Data Type	Dimensions	Purpose
Histogram	Categorical, continuous	1	Displays categorical differences

```
ggplot(data=trees, aes(x=Girth)) + geom_histogram(binwidth=1)

ggplot(data=trees, aes(x=Girth, )) + geom_histogram(aes(fill=Species), binwidth=2)
```



geom( )

geom_area( )

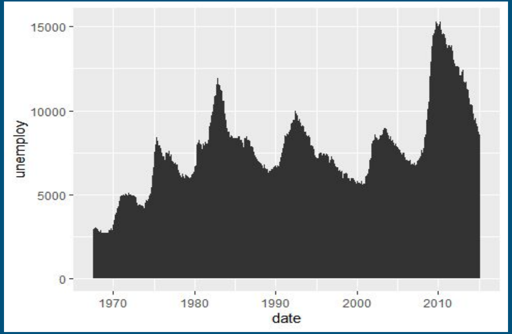
geom_area( ) is typically used to compare how a certain metric performed against a baseline, i.e. stocks, unemployment, etc. The ymin is set to 0. Unlike a histogram that shows the frequency of one x variable, the geom_area( ) uses an actual y variable for the vertical distribution.

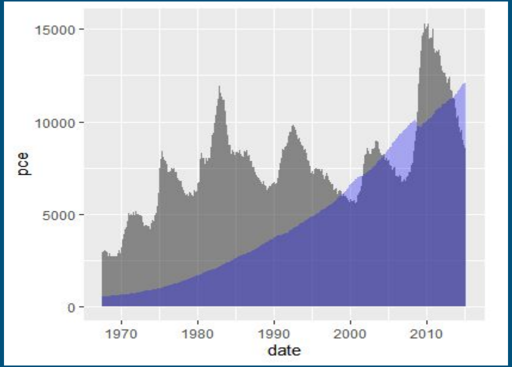
The second example adds two geom_area( ) to the base ggplot( ). Note that one of them uses a different y variable introduced as an aesthetic to only the geom( ). alpha=.3 sets 30% transparency.

Plot Type	Data Type	Dimensions	Purpose
Area Plot	Continuous	1; 2	Connecting observations ordered by “x”

```
ggplot(economics, aes(x=date, y=unemploy)) + geom_area()

ggplot(economics, aes(x=date, y=pce)) + geom_area(aes(y=unemploy, alpha=.3)) +
geom_area(fill="blue", alpha=.3)
```





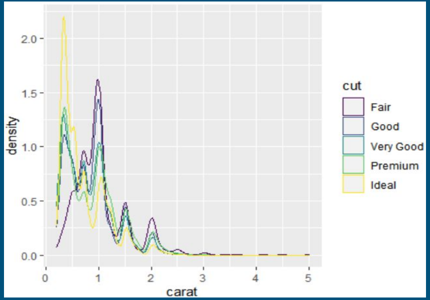
geom( )

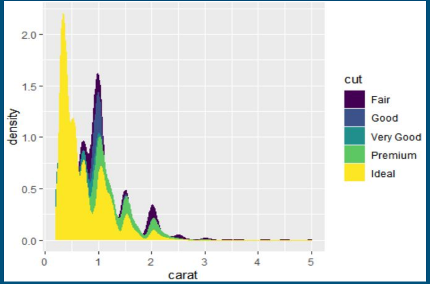
geom_density( )

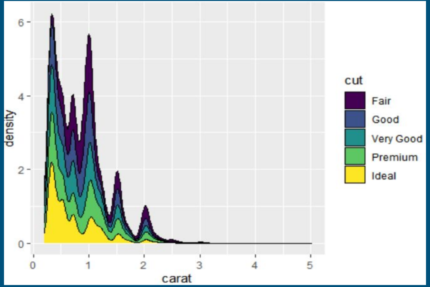
geom_density( ) is similar to a histogram, but is a smoothed version. This is useful for continuous data with an underlying smooth distribution. The plot can contain only outlines or a solidly filled version. In the latter one, however, areas in the foreground might overlap and hide other areas in the background. The "position" parameter in the geom_density( ) call can be used to stack all areas vertically to have a better comparison of each x value and a clearer image of the overall distribution.

Plot Type	Data Type	Dimensions	Purpose
Smoothed Histogram	Continuous	1	Distribution and density estimates

```
ggplot(diamonds, aes(carat, colour = cut)) + geom_density()
ggplot(diamonds, aes(carat, colour = cut, fill = cut)) + geom_density()
ggplot(diamonds, aes(carat, fill = cut)) + geom_density(position = "stack")
```







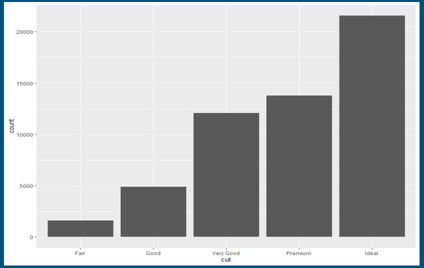
geom( )

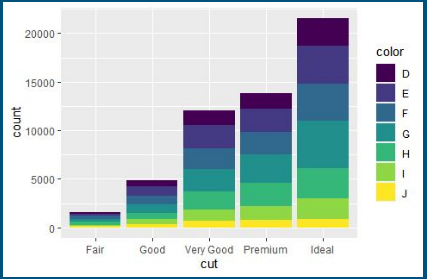
geom_bar( )

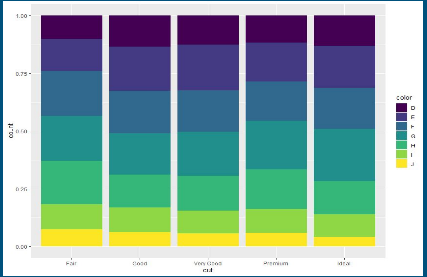
geom_bar( ) creates a bar chart with one variable (x), and the y axis is the frequency of x. These plots are more suitable than histograms when the x variable is categorical, e.g. genders, types or, in this case, diamond cuts. By default, only the frequency of the x variable's categories are presented. If the aesthetic "fill" is given a variable, the bars will be separated into respective shares of the second variable. (The aesthetic "color" only colors the outlines, which, in most cases, is difficult to see.) The "position" parameter can be set to "filled" to give each bar the same height. Now, percentages can be read on the y axis.

Plot Type	Data Type	Dimensions	Purpose
Bar Chart	Categorical	1;2	Deviation, ranking or composition

```
ggplot(diamonds, aes(cut)) + geom_bar()
ggplot(diamonds, aes(cut, fill = color)) + geom_bar()
ggplot(diamonds, aes(cut, fill = color)) + geom_bar(position="fill")
```







geom( )

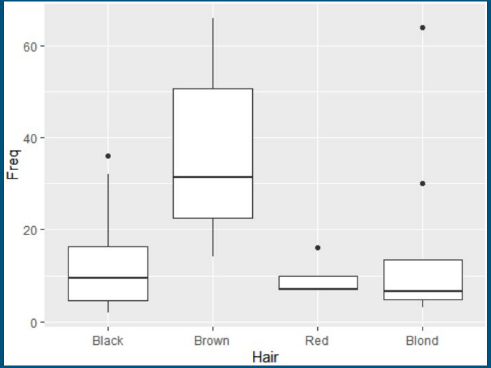
geom_boxplot( )

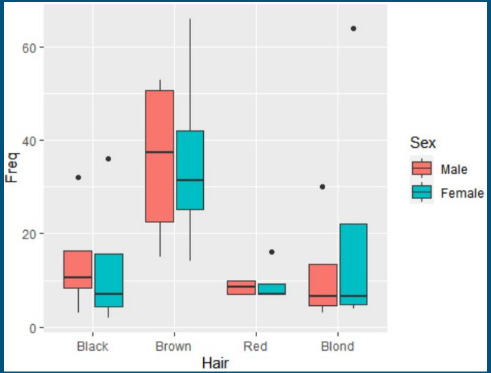
geom_boxplot( ) can be useful to show the distribution, median, range and outliers of a categorical variable. The top of the box is the 75th percentile, and the bottom is 25th percentile, with the dark line being the median. A secondary variable y can be introduced to have the data split. The example above shows the frequencies of hair colors, while the example below also makes a gender split. Long "whiskers" above and below boxes (each indicating 25% of the data) show long tails in the distribution. Outliers are shown with dots.

Plot Type	Data Type	Dimensions	Purpose
Box and Whiskers Plot	Categorical X, Continuous Y	2	Distribution and density estimates, with outliers

```
df <- as.data.frame(HairEyeColor) #input must be a data.frame

ggplot(df, aes(x=Hair, y=Freq)) + geom_boxplot()
ggplot(df, aes(x=Hair, y=Freq, fill=Sex)) + geom_boxplot()
```





geom( )

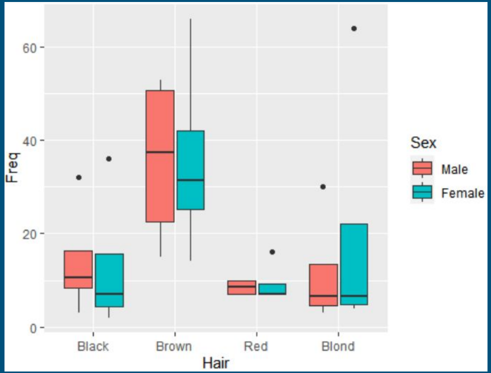
geom_violin( )

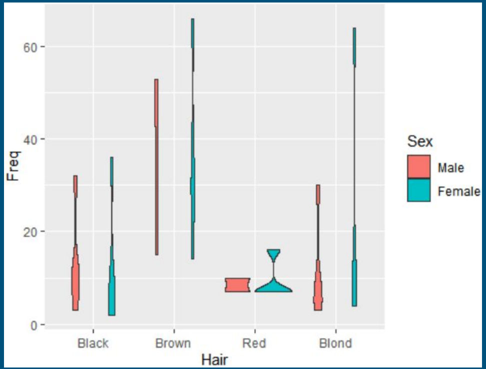
geom_violin( ) is similar to a box plot but with a density distribution. Instead of boxes containing 25% of the data, the graph shows vertically aligned density plots, where "thicker" parts of the shape represent more data in these areas. Short boxes equal thick shapes, stretched boxes equal thin shapes. Often this leads to a violin shape giving the plot its name. Violin plots may be easier to interpret than boxplots for laymen.

Plot Type	Data Type	Dimensions	Purpose
Violin Plot	Categorical X, Continuous Y	2	Compact display of a continuous distribution

```
df <- as.data.frame(HairEyeColor) #input must be a data.frame

ggplot(df, aes(x=Hair, y=Freq, fill=Sex)) + geom_boxplot()
ggplot(df, aes(x=Hair, y=Freq, fill=Sex)) + geom_violin()
```



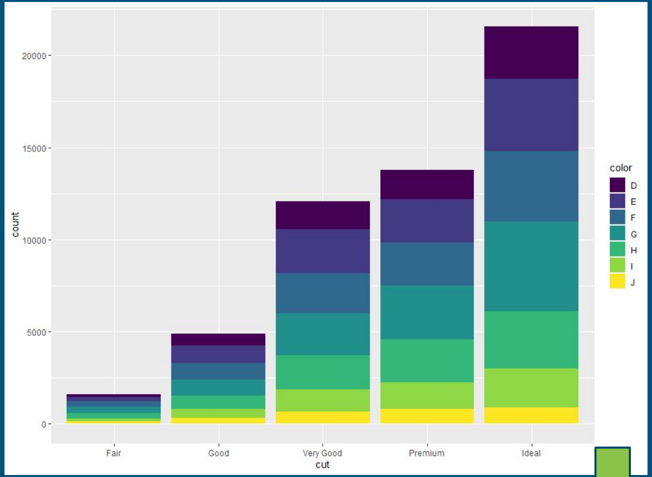


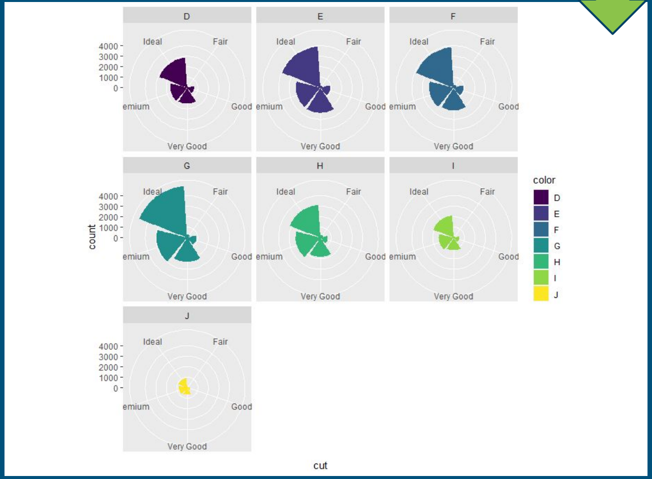
Graphical Parameters

In this chapter we will look at graphical parameters that change the overall appearance of the plot.

We will discuss the following parameters:

- Facets
- Coordinate Systems
- Labels
- Legends
- Themes



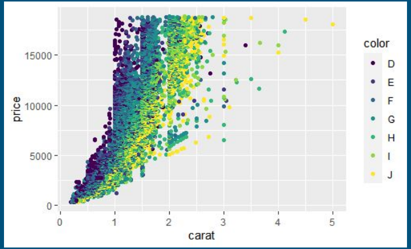


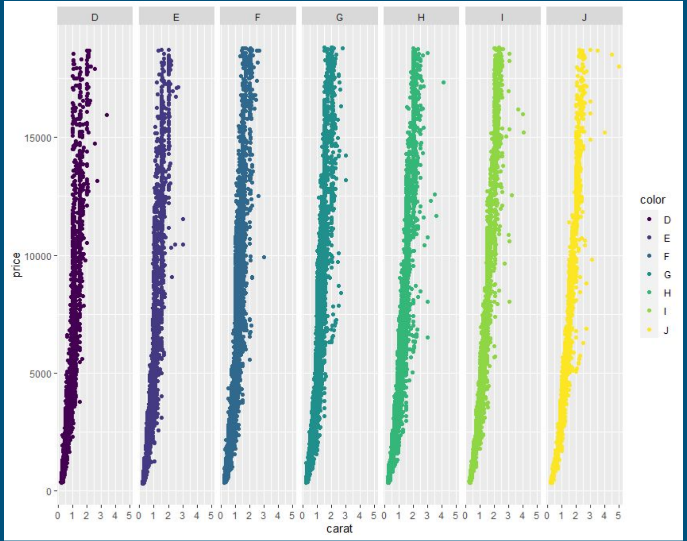
Graphical parameters

Facets

Facets are a key feature of ggplot2. They disentangle data by separating categories of data into individual plots with comparable axes and scales. The example shows a graph where individual data points are difficult to identify because of their quantity and the fact that some points overlap other points. The facet_grid( ) we apply in the second plot separates all data points by the chosen attribute “color”, splitting the previous plot into seven individual ones.

The next slides will explain how to use facet_wrap( ) or facet_grid( ).





```
ggplot(diamonds, aes(carat, price, color= color)) +    geom_point()

ggplot(diamonds, aes(carat, price, color= color)) +    geom_point() + facet_grid(.~color)
```

Graphical parameters

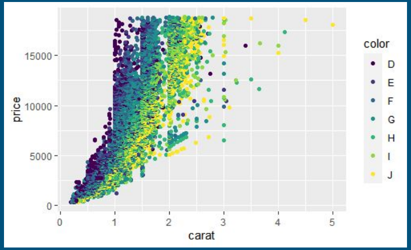
facets_wrap( )

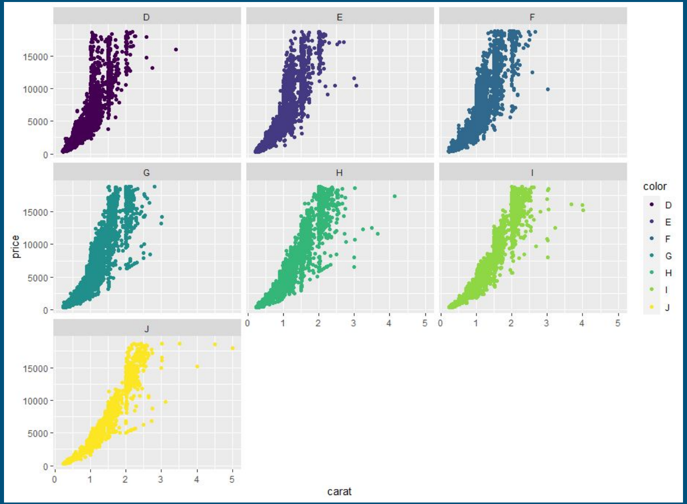
facet_wrap( ) creates an array of facet views. The vars( ) function is needed to subset the data into individual datasets. All views will have the same coordinate system to make them comparable.

The parameters “ncol” and “nrow” can be used to specify how many rows and columns the array will have. By default, three columns will be created filled from left to right and top to bottom.

The variable passed to facet_wrap should be categorical with only a few different categories. A continuous variable with 100 different values would create 100 individual views, all too small to properly inspect. (This might also crash the program.)

```
ggplot(diamonds, aes(carat, price, color= color)) +    geom_point() +
facet_wrap(vars(color))
```





Graphical parameters

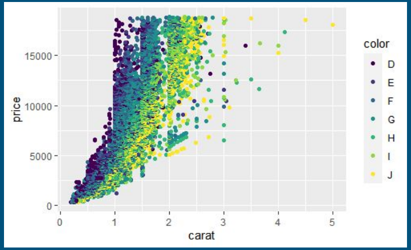
facets_grid( )

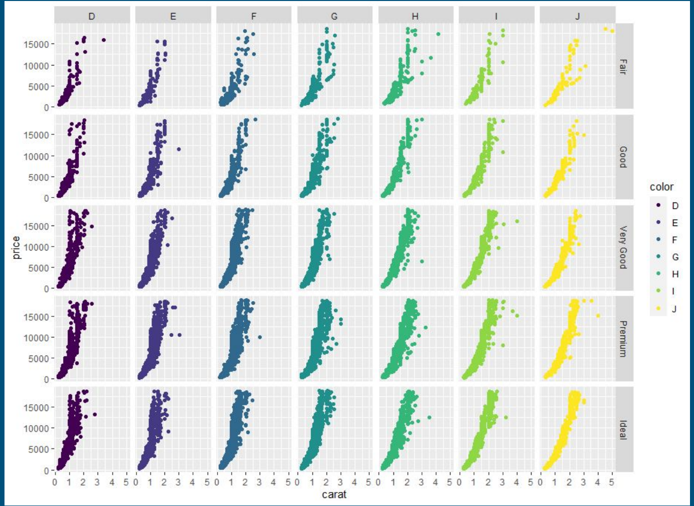
facet_grid( ) creates an array of facet views. While the facet_wrap( ) fills an array in a systematic order, the facet_grid( ) creates a cross table structure.

The facet_grid(y~x) creates a cross table which as many rows as y has categories, and as many columns as x has categories. Here again, the variable should be categorical and, ideally, only have 3-7 categories each. 7 categories in both x and y would create a matrix with 49 individual views. All would be rather small and might require a long time to render.

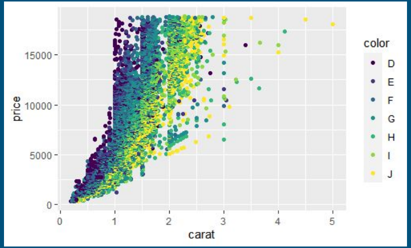
facet_grid(y~.) creates only one column and facet_grid(.~x) creates only one row. Examples are shown on the next slide.

```
ggplot(diamonds, aes(carat, price, color= color)) +    geom_point() +
facet_grid(cut~color)
```

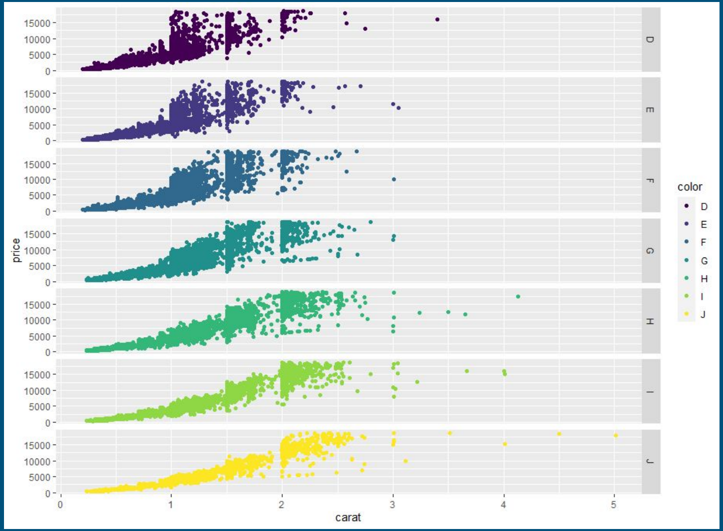




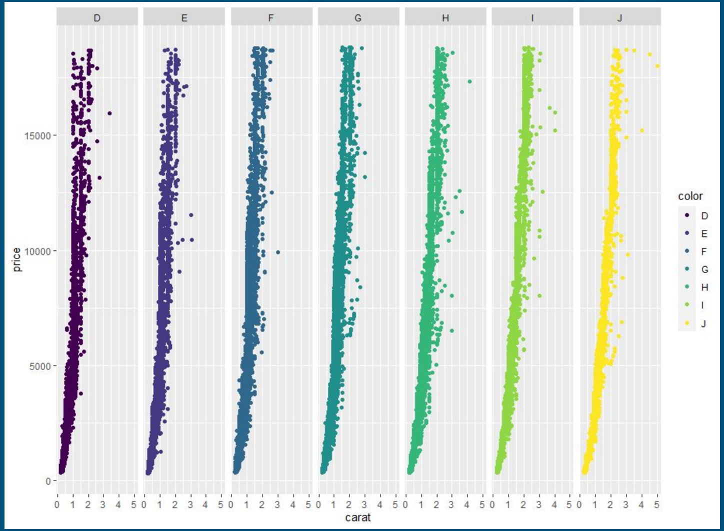
Graphical parameters



facets_grid( )



... + facet_grid(color~.)



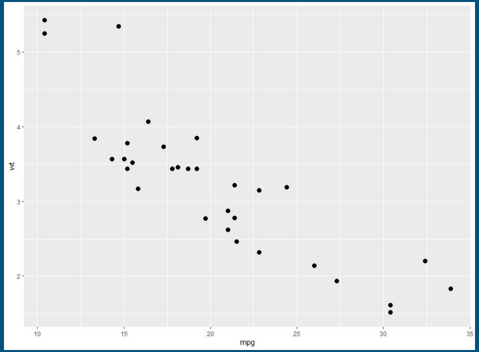
... + facet_grid(.~color)

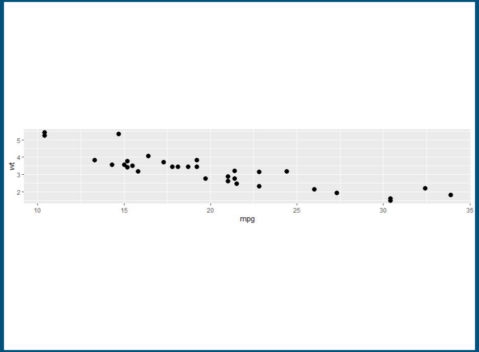
Graphical parameters

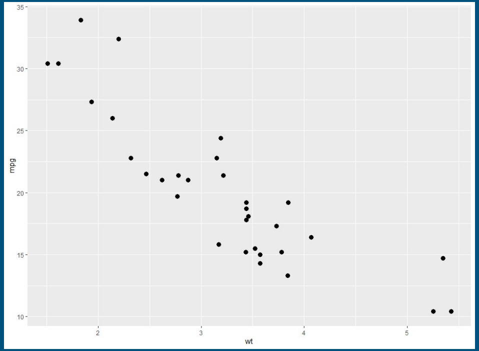
Coordinate systems

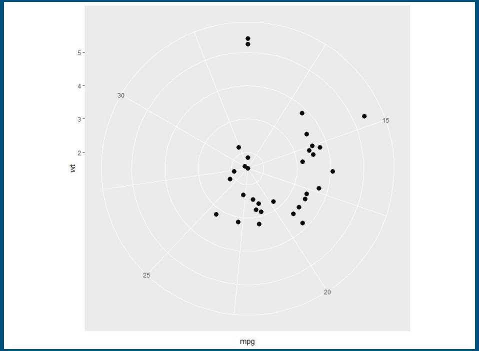
Coordinate systems change the projection of data. coord_cartesian( ) is the standard projection of plot( ) and ggplot( ). coord_fixed( ) creates a plot where the axis are scaled in a similar ratio. In this example x has a range of 25, while y has a range of 5, leading to a 1:5 ratio. coord_flipped( ) exchanges the x and y variables. This can be used to have bar plots vertically aligned. The coord_polar( ) aligns the x axis in a ring. This can also be applied to have star-shaped bar plots. However, polar coordinate systems are less accessible to readers.

```
p <- ggplot(mtcars, aes(mpg, wt)) + geom_point(size=3)
p + coord_cartesian()     # top left
p + coord_fixed()         # top right
p + coord_flip()          # bottom left
p + coord_polar()         # bottom right
```









Graphical parameters

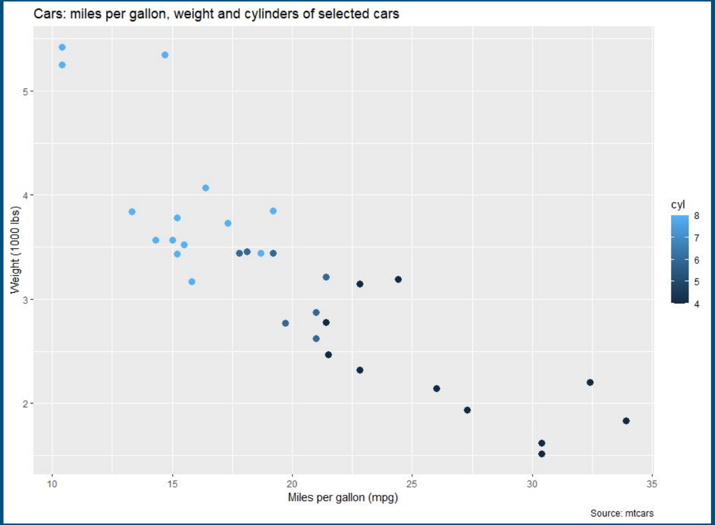
Labels

The labs( ) function adds labels (axes, titles, captions) to the plot.

This is similar to “main” and “x/ylab” in the plot( ) function, however, the parameters are named slightly different. “title” adds a title, while “x” and “y” add their respective axis labels. The input given to the “caption” parameter appears in the bottom right corner of the plot. This can be used to add a source or a note to the plot.

As with plot( ), the user has to decide whether the title and caption should be hard-coded to the plot or whether they will be added in e.g. Microsoft Word as figure caption. Labels for x and y axes should always be given with a precise description of the variables and their units.

```
p <- ggplot(mtcars, aes(mpg, wt, col=cyl)) + geom_point(size=3)
p + labs(title="Cars: miles per gallon, weight and cylinders of selected cars",
       x="Miles per gallon (mpg)", y="Weight (1000 lbs)",  caption="Source: mtcars")
```



Graphical parameters

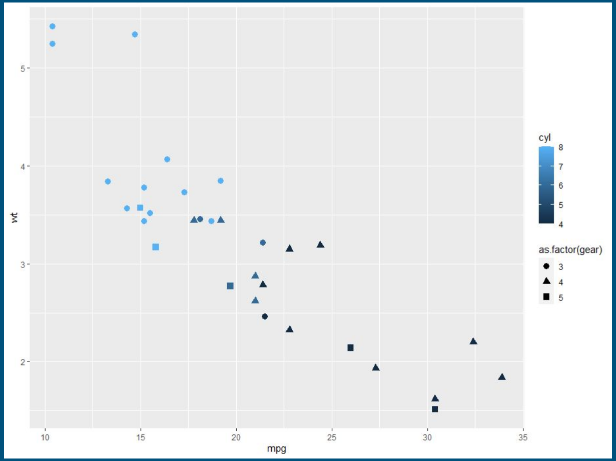
Legends

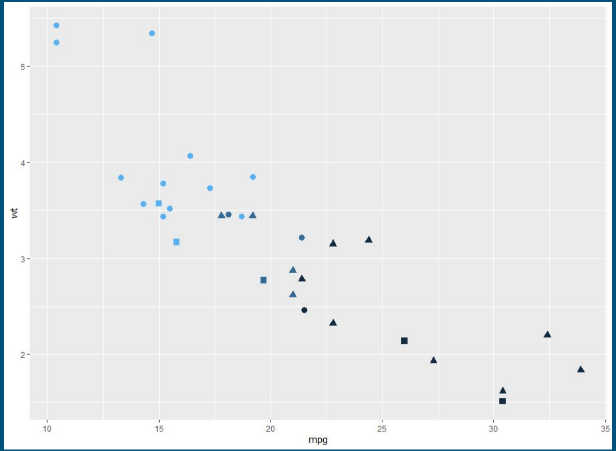
ggplot( ) creates legends automatically for variables added by the parameters “shape”, “color”, “fill” or “size”.

The user has several options to delete the legend or adjust how it is presented. In this example the legend was removed. This is usually not recommended as the plot itself will not be inherently understandable. It might be useful to remove the legend if the plot will be presented amidst other plots where at least one uses the same legend.

```
p <- ggplot(mtcars, aes(mpg, wt, color=cyl, shape=as.factor(gear))) + geom_point(size=3)
p

p + theme(legend.position = "none")
```





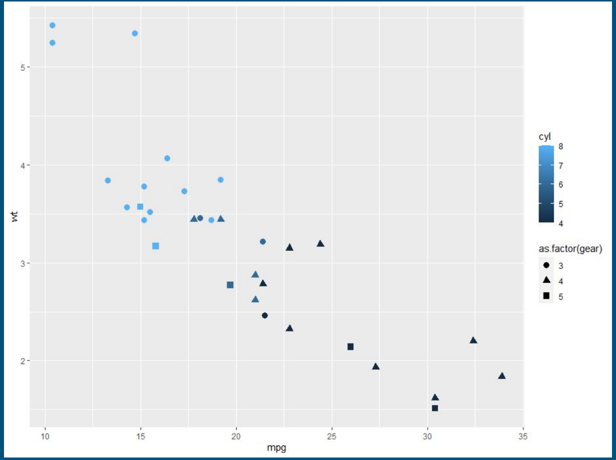
Graphical parameters

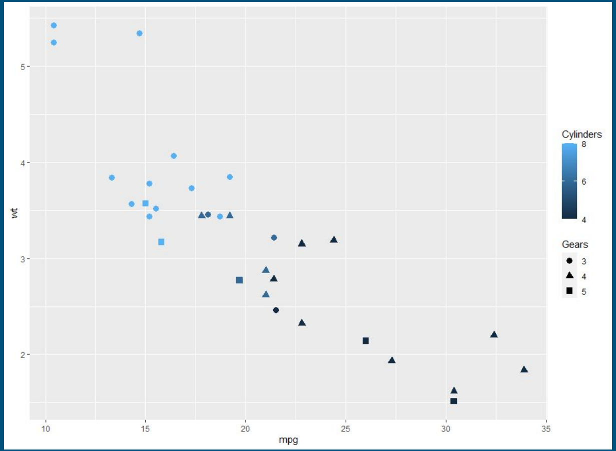
Legends

The legends can be adjusted to show different titles or scales. In the example above, there are a few aspects of the legends that could be improved: (1) “cyl” and “as.factor(gear)” are bad titles, (2) the cylinder scale includes the values “7” and “5” although no car with these properties are in the dataset.

By using scale_PARAMETER_TYPE, we can adjust the legend for the specified parameter. For example, if we have a continuous “fill” parameter, the function “scale_fill_continuous” would be used. The type is usually either “continuous” or “discrete”. The parameter “name” changes the title. A vector passed to “breaks” limits the scale of the legend (here: only including 8, 6 and 4). Similarly, the parameter “labels” adjusts which labels are shown.

```
p <- ggplot(mtcars, aes(mpg, wt, color=cyl, shape=as.factor(gear))) + geom_point(size=3)
p + scale_color_continuous(name="Cylinders", breaks =c(8,6,4) ) +
   scale_shape_discrete(name="Gears")
```





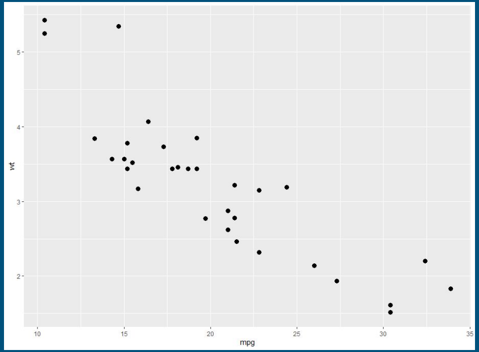
Graphical parameters

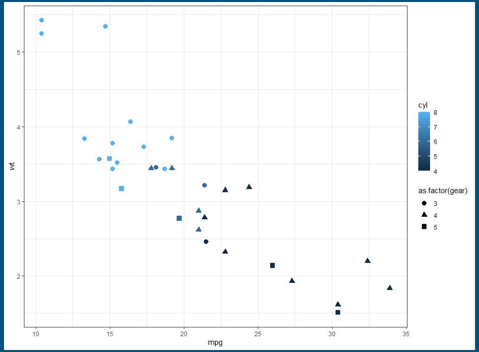
Themes

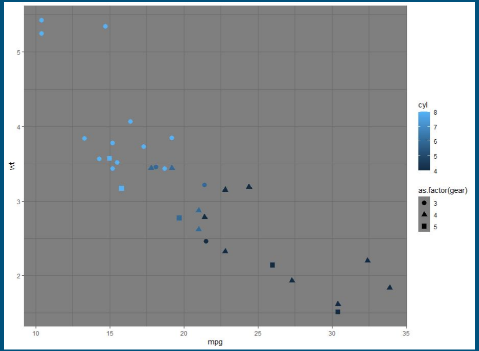
Themes change the overall appearance of a plot. They can be used to define a consistent presentation of several plots. By default, each plot is given a light gray background raster. The theme_bw( ) reduces this to a black/white raster. This saves ink when printing and might make the plot clearer. A theme_dark( ) creates a dark gray background. This should be avoided for prints. The theme_classic( ) only creates axes but no background raster. ?theme gives an overview about further adjustments.

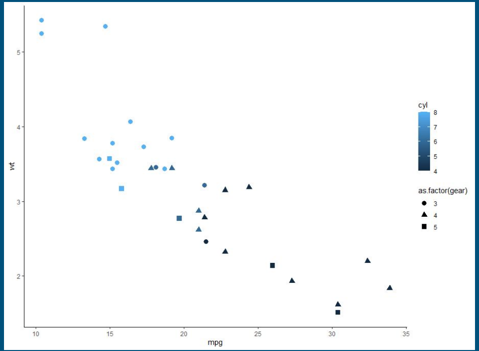
```
p <- ggplot(mtcars, aes(mpg, wt, color=cyl)) + geom_point()

p                       # top left
p + theme_bw()          # top right
p + theme_dark()        # bottom left
p + theme_classic()     # bottom right
```


wt
mpg


wt
mpg
cyl
as.factor(gear)
● 3
▲ 4
■ 5


wt
mpg
cyl
as.factor(gear)
● 3
▲ 4
■ 5


wt
mpg
cyl
as.factor(gear)
● 3
▲ 4
■ 5