Programming for Data Science

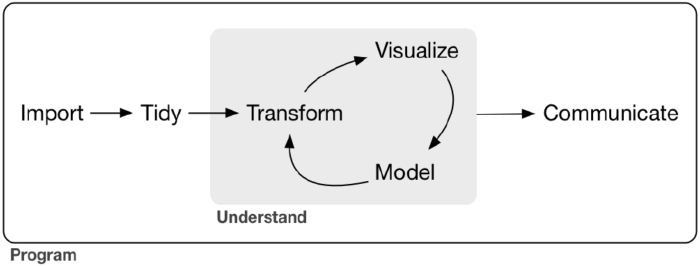
Data visualization in R.

Marco Beccuti
Università degli Studi di Torino
Dipartimento di Informatica

October 2021

Visualization R using ggplot2

- A typical data science project can be sketched as follows:



Visualization in R using ggplot2

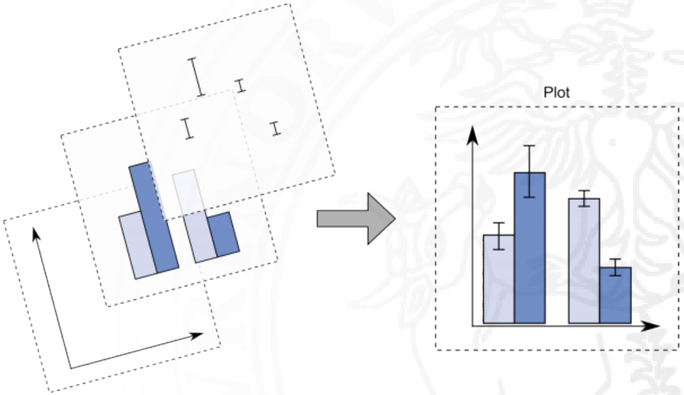
- To draw graphs, you can use a package called **ggplot2** which is more powerful and more versatile than **plot()**
- To install **ggplot2** you have to type:

  > *install.packages("ggplot2")*
- To use it you have to load it:

  > *library(ggplot2)*

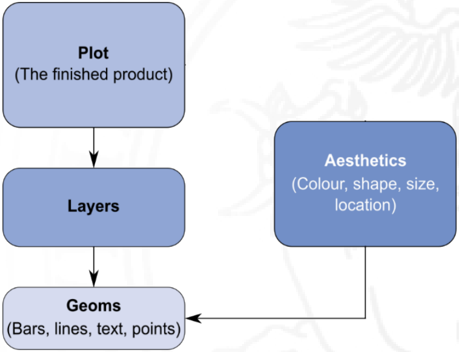
A graph in ggplot2

- A ggplot2 graph is made up of a series of layers.



A graph in ggplot2

- Visual elements are called **geoms** (as in geometric objects: bars, points . . .);
- The appearance and location of **geoms** (size, colours ...) are controlled by the aesthetics properties;
- Variables that you want to plot to referred to as **aes()**.



Geometric objects (geom)

- *geom_bar*(): creates a layer with a bar chart;
- *geom_point*(): creates a layer with a scatterplot;
- *geom_line*(): creates a layer with line plot;
- *geom_smooth*(): creates a layer with a smoothing line showing the trend.
- *geom_histogram*(): creates a layer with a histogram;
- *geom_boxplot*(): creates a layer with a boxplot;
- *geom_text*(): creates a layer with text in it;
- *geom_errorbar*(): creates a layer with error bars in it;
- *geom_hline*() and *geom_vline*(): creates a layer with a user-defined horizontal or vertical line. respectively;
- ...

Between the brackets we need to specify aesthetics:

required: the variable used to create the plot;

optional: plot attributes such as colour, size ...

General command for a ggplot2 graph

ggplot(MyData, aes(var. for x axis, var. for y axis))+geom$_1$()+...+geom$_n$

- Function ggplot() initializes a ggplot object. It can be used to declare the input data frame for a graphic and to specify the set of plot aesthetics common throughout all layers unless specifically overridden.
- Symbol + is used to add layers using geom functions;
- Library ggplot2 works only with a dataframe (i.e. MyData must be a dataframe).

Visualization in ggplot2

- We use a graph to answer the following question:

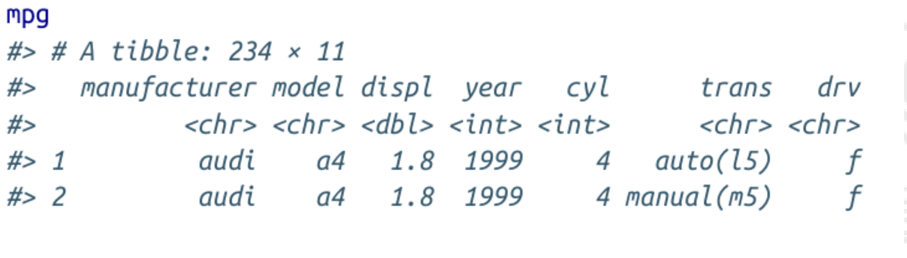
  Do cars with big engines use more fuel than cars with small engines?

  What does the relationship between engine size and fuel efficiently look?

  Is it positive? Negative? Linear? Non linear?

- We can answer using **mpg** data frame containing observation on 38 models of cars;
- A data frame is a rectangular collection of **variables (in the columns) and observations (in the rows)**.

Visualization in ggplot2



```
mpg
#> # A tibble: 234 × 11
#>   manufacturer model displ  year   cyl      trans   drv
#>          <chr> <chr> <dbl> <int> <int>      <chr> <chr>
#> 1         audi    a4   1.8  1999     4   auto(l5)     f
#> 2         audi    a4   1.8  1999     4 manual(m5)     f
```

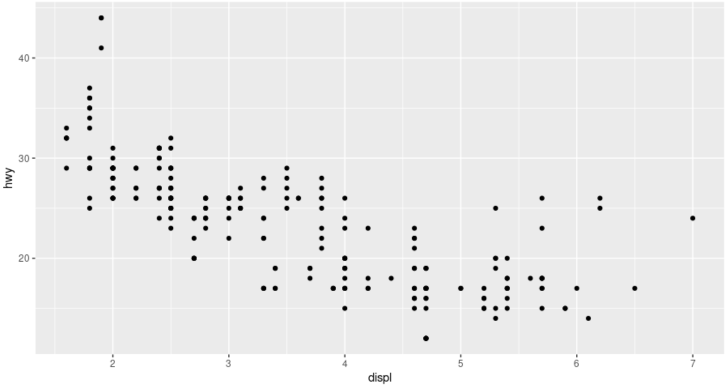
- Among the variables in **mpg** there are:
  - **displ** is the car's engine size in liters;
  - **hwy** is a car's fuel efficiency on the highway. A car with a low fuel efficiency consumes more fuel than a car with a high fuel efficiency when they travel the same distance.
- To learn more about mpg, open its help page by running:

  >?mpg

Creating a ggplot

- To answer we plot **displ** vs **hwy** using the following command:

  $> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) + geom\_point()$



- The plot shows a negative relationship between engine size (**displ**) and fuel efficiency (**hwy**).

Creating a ggplot

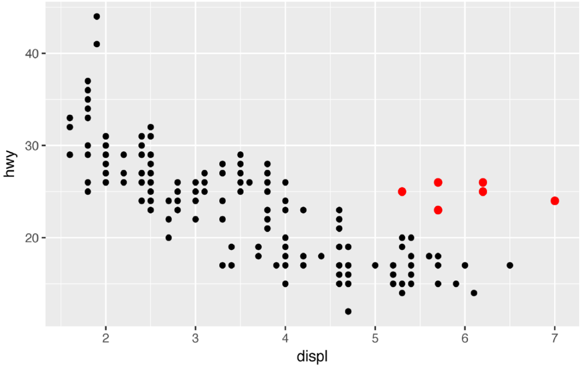
```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) + geom_point()
```

- A plot begins with the function **ggplot()** creating a coordinate system;
- The first argument of **ggplot()** is the dataset to use in the graph;
- The function **geom_point()** adds a layer of points to your plot;
- Each geom function in ggplot2 takes a **mapping** argument defining how variables are mapped to visual properties;

  ```
  > ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy))
  > ggplot(data = mpg) + geom_point(aes(x = displ, y = hwy))
  ```

- The mapping argument is always paired with **aes()**, and the **x** and **y** arguments of **aes()** specify which variables to map to the x-and y-axes.

Aesthetic Mappings



- In the plot, the group of red points seems to fall outside of the linear trend;
- These cars have a higher mileage than you might expect. How can you explain these cars? Let's hypothesize that the cars are **hybrids**.
- The **class** variable classifies cars into groups such as compact, midsize, and SUV. If the outlying points are hybrids, they should be classified as compact cars or, perhaps, subcompact cars (Observe data was collected before hybrid trucks and SUVs).

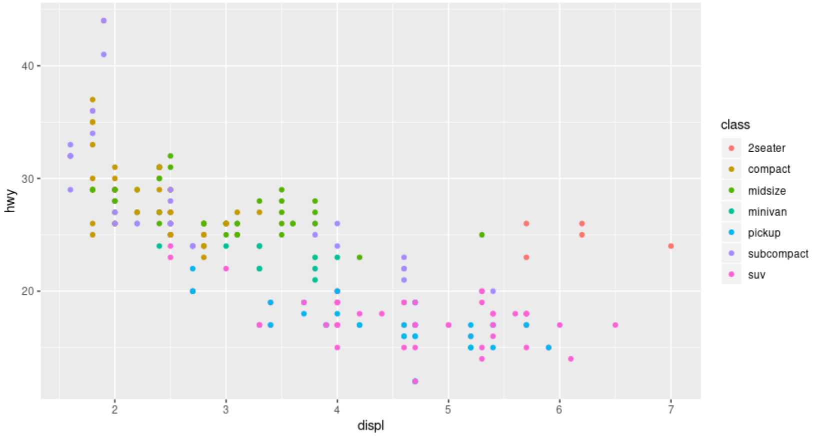
Aesthetic Mappings

- You can add a third variable, like **class**, to a two-dimensional scatterplot by mapping it to an aesthetic;
- An aesthetic is a visual property of the objects in your plot like the size, the shape, or the color of your points;

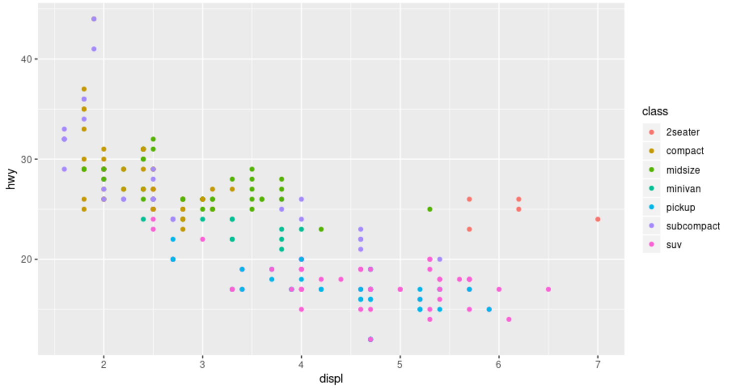
Aesthetic Mappings

- For example, you can map the colors of your points to **class** variable to reveal the class of each car:

  > ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy, color = class))



Aesthetic Mappings

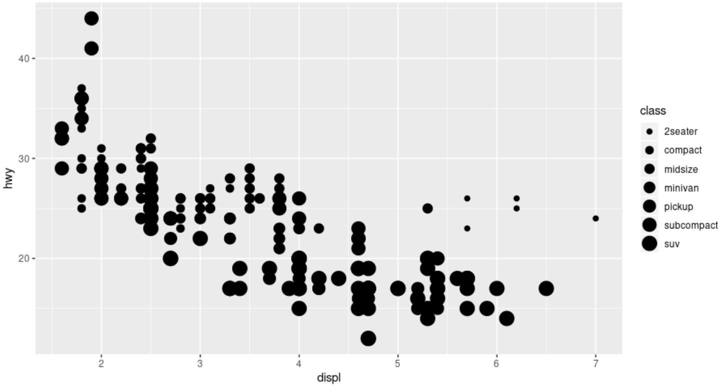


- The colors reveal that many of the unusual points are two-seater cars;
- These cars are sports cars: sports cars have large engines like SUVs, but small bodies like midsize and compact cars.

Aesthetic Mappings

- For example, we could map class to the size aesthetic in the same way. :

  $> ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy, size = class))$

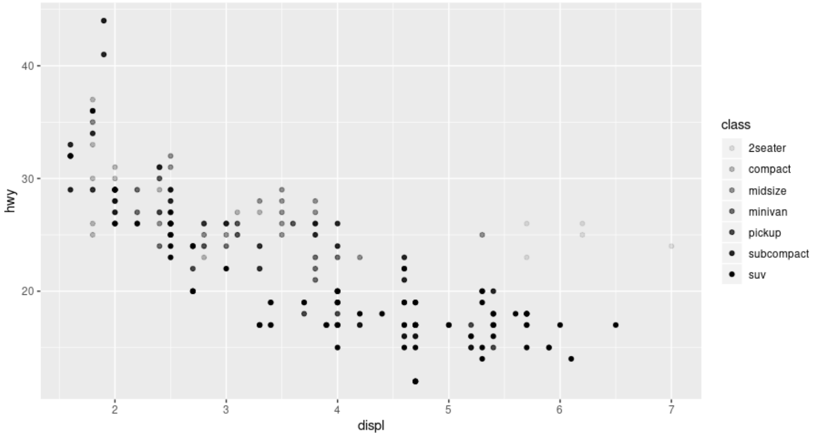


- We get a warning here, because mapping an unordered variable (**class**) to an ordered aesthetic (size) is not a good idea.

Aesthetic Mappings

- For example, we could map **class** to the alpha aesthetic, which controls the transparency of the points, or the shape of the points:

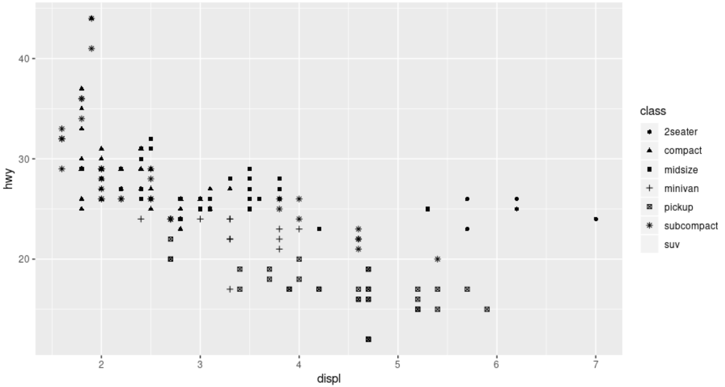
  $> ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy, alpha = class))$



Aesthetic Mappings

- For example, we could map **class** to the alpha aesthetic, which controls the transparency of the points, or the shape of the points:

  $> ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy, shape = class))$

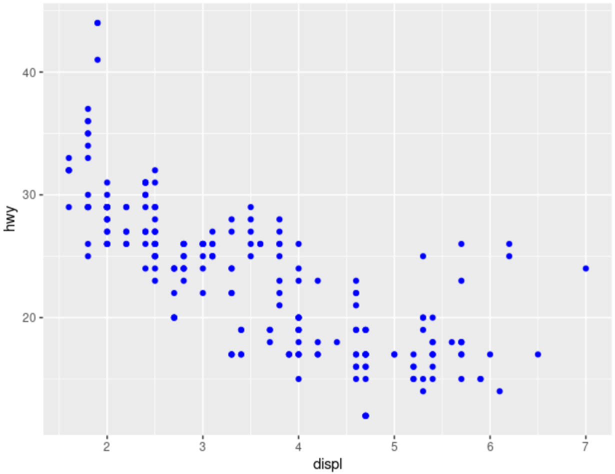


- What happened to the SUVs? ggplot2 will only use six shapes at a time. By default, additional groups will be not plotted.

Aesthetic Mappings

- You can also set the aesthetic properties of your geom manually;
- For example, we can make all of the points in our plot blue:

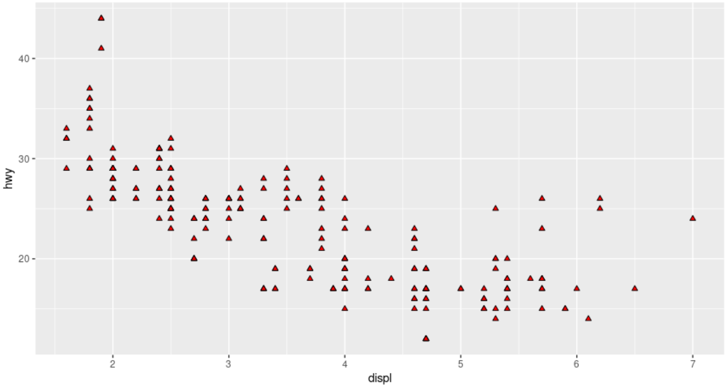
$> ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy), color = \text{“}blue\text{''})$



Aesthetic Mappings

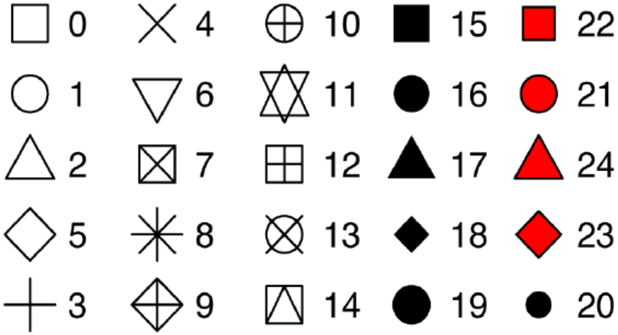
- You can also set the aesthetic properties of your geom manually;
- For example, we can change the point shape as follows:

  $> ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy), shape = 18, fill = "red")$



Aesthetic Mappings

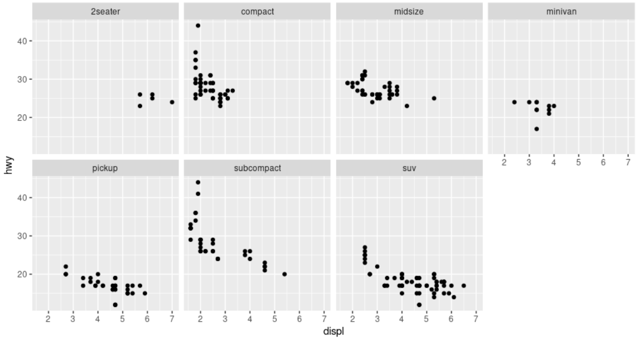
- R has 25 built-in shapes that are identified by numbers.



Facets

- Particularly useful it is to split a plot into subplots according to the values of a categorical variables:

  $> ggplot(data = mpg)+geom\_point(mapping = aes(x = displ, y = hwy))+ facet\_wrap(\sim class, nrow = 2)$

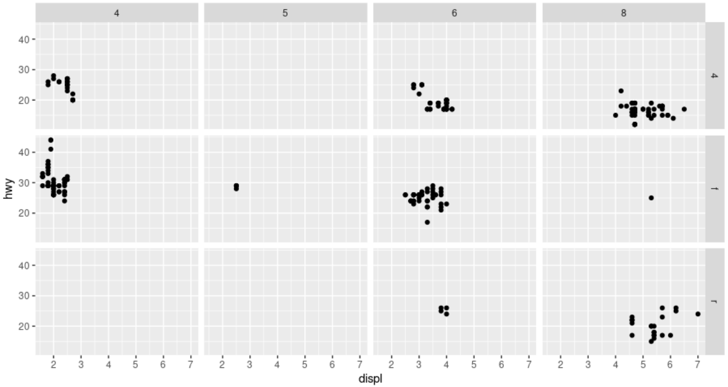


- To facet your plot by a single variable, use **facet_wrap()**. The first argument of **facet_wrap()** is also a formula.

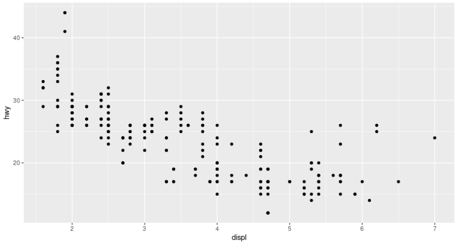
Facets

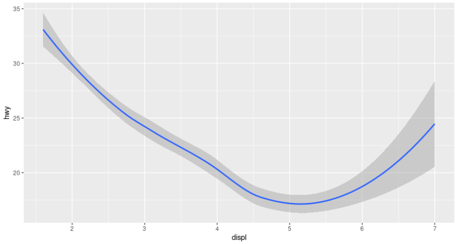
- To facet your plot on the combination of two variables, add **facet_grid()** to your plot call:

  $> ggplot(data = mpg)+geom\_point(mapping = aes(x = displ, y = hwy))+ facet\_grid(drv \sim cyl)$



Geometric objects





- Both plots contain the same x variable and the same y variable, and both describe the same data. But the plots are not identical.
- The plot on the left uses the point geom, and the plot on the right uses the smooth geom, a smooth line fitted to the data.

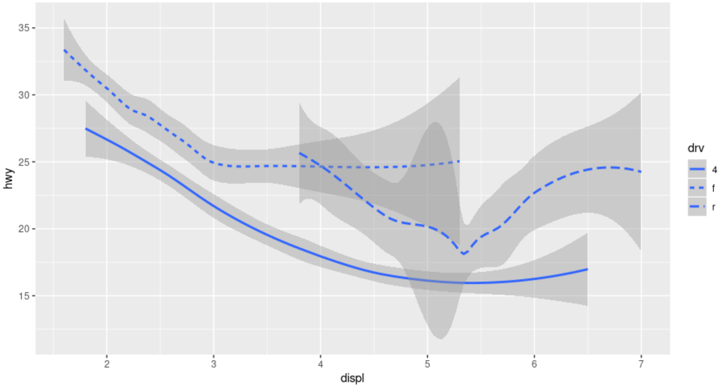
  $> ggplot(data = mpg) + geom\_point(mapping = aes(x = displ, y = hwy))$

  $> ggplot(data = mpg) + geom\_smooth(mapping = aes(x = displ, y = hwy))$

Geometric objects

- **geom_smooth()** will draw a different line, with a different linetype, for each unique value of the variable that you map to linetype:

  $> ggplot(data = mpg) + geom\_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))$

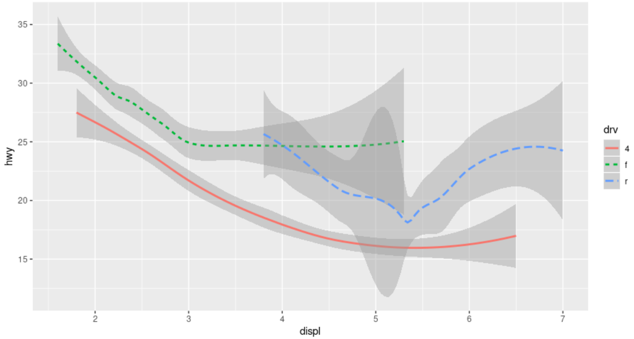


- Here **geom_smooth()** separates the cars into three lines based on their **drv** value, which describes a car's drivetrain.

Geometric objects

- **geom_smooth()** will draw a different line, with a different linetype and color, for each unique value of the variable that you map to linetype and color:

  > ggplot(data = mpg) + geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv, color = drv))

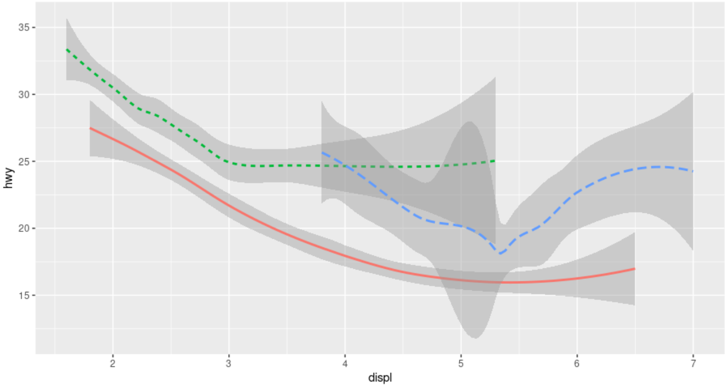


- Here **geom_smooth()** separates the cars into three lines based on their **drv** value, which describes a car's drivetrain.

Geometric objects

- **show.legend** can be used to remove the plot's legend:

  $> ggplot(data = mpg) + geom\_smooth(mapping = aes(x = displ, y = hwy, linetype = drv, color = drv), show.legend = FALSE)$



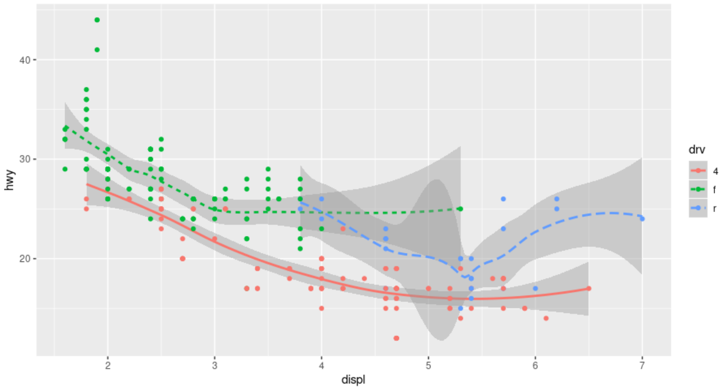
- Here **geom_smooth()** separates the cars into three lines based on their **drv** value, which describes a car's drivetrain.

Geometric objects

- To display multiple geoms in the same plot, add multiple geom functions to **ggplot()**:

  $> my = ggplot(data = mpg) + geom\_smooth(mapping = aes(x = displ, y = hwy, linetype = drv, color = drv))$

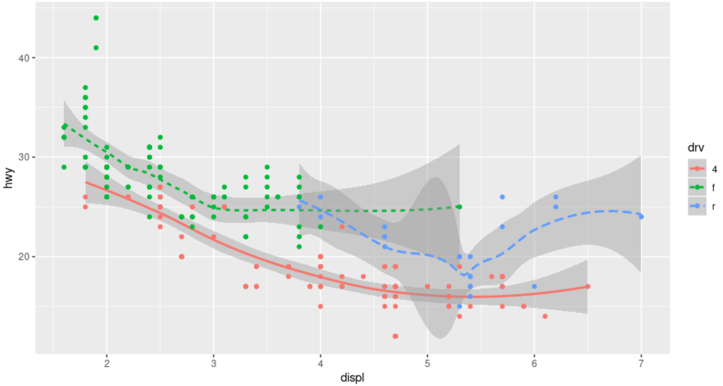
  $> my + geom\_point(mapping = aes(x = displ, y = hwy, color = drv))$



Geometric objects

- The previous example introduces some duplication in our code (i.e. mapping repetition);
- This type of repetition can be avoided by passing a set of mappings to **ggplot()**. These mappings will be treated as global mappings:

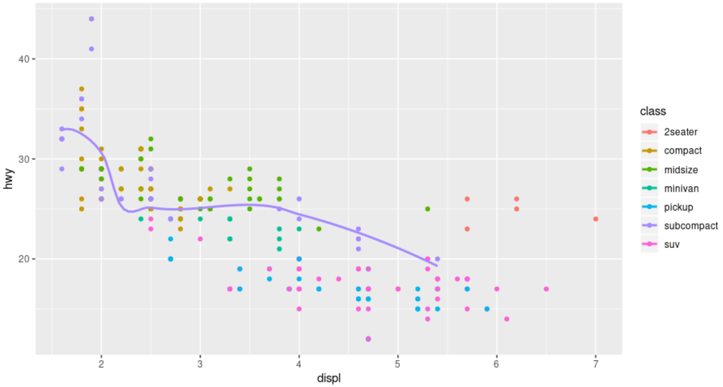
  $> ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) + geom\_point() + geom\_smooth(mapping = aes(linetype = drv))$



Geometric objects

- To specify different data for each layer:

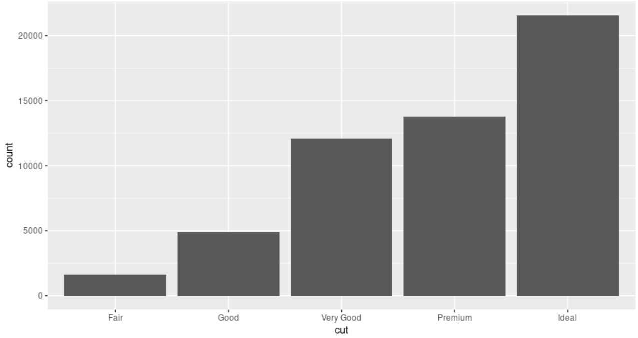
```
> install.packages("tidyverse")
> library(tidyverse)
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color =
class)) + geom_point() + geom_smooth(data = filter(mpg, class ==
"subcompact"), se = FALSE)
```



Statistical Transformations

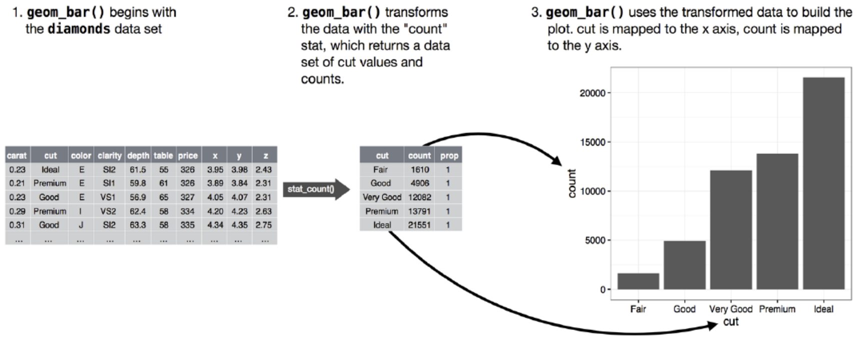
- We consider the **diamonds** dataset in **ggplot2**;
- It contains information about 54,000 diamonds, including the price, carat, color, clarity, and cut of each diamond.
- To shows that more diamonds are available with high-quality cuts than with low quality cuts we used a **bar chart**:

  $> ggplot(data = diamonds) + geom\_bar(mapping = aes(x = cut))$



Statistical Transformations

- Where does **count** come from?
- Many graphs, like scatterplots, plot the raw values of your dataset. Other graphs, like bar charts, calculate new values to plot;
- The algorithm used to calculate new values is called a stat, short for statistical transformation.



1. **geom_bar()** begins with the **diamonds** data set

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...

2. **geom_bar()** transforms the data with the "count" stat, which returns a data set of cut values and counts.

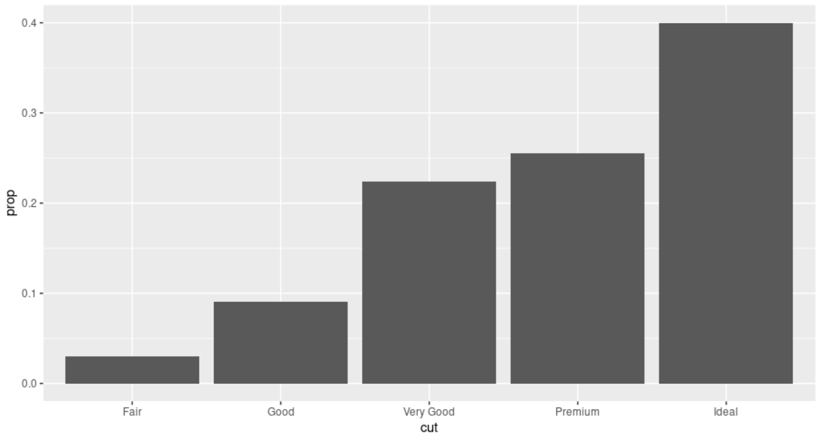
cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1

3. **geom_bar()** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.

Statistical Transformations

- To display a bar chart of proportion, rather than count:

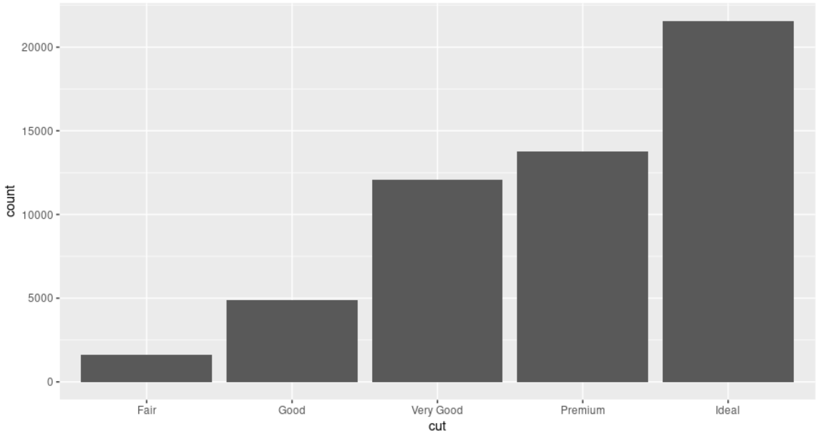
  $> ggplot(data = diamonds) + geom\_bar(mapping = aes(x = cut, y = ..prop.., group = 1))$



Statistical Transformations

- **geom_bar()** uses **stat_count()**;
- You can generally use geoms and stats interchangeably. You can re-create the previous plot using:

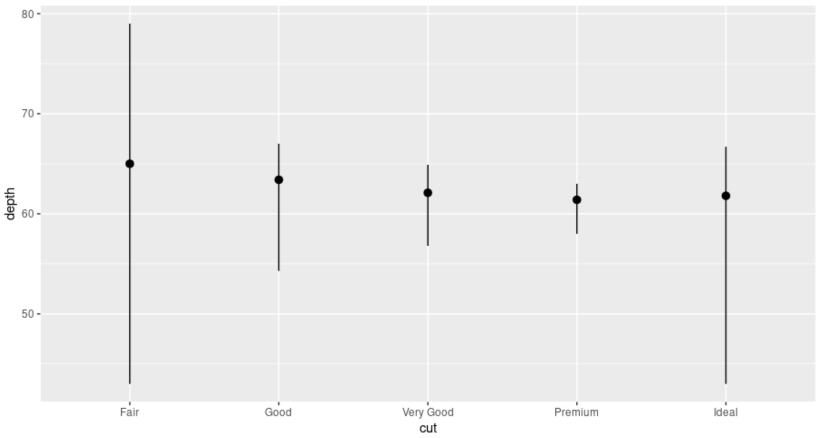
  > ggplot(data = diamonds) + stat_count(mapping = aes(x = cut))



Statistical Transformations

- To visualize a summary of the date **stat_summary()** can be used;
- It summarizes the y values for each unique x value;

> ggplot(data = diamonds) + stat_summary(mapping = aes(x = cut, y = depth), fun.ymin = min, fun.ymax = max, fun.y = median)

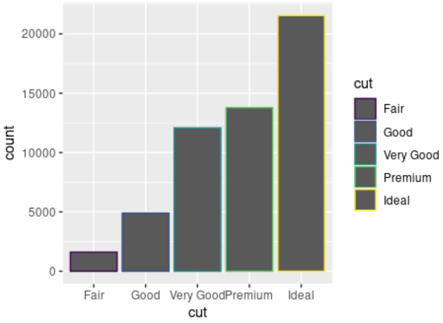


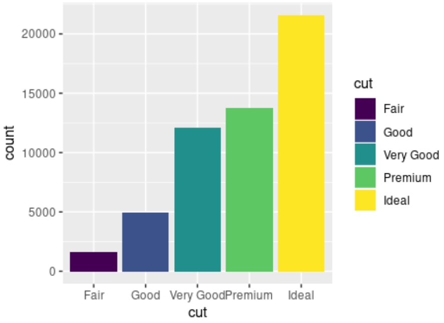
Position Adjustments

- You can color a bar chart using either the **color** or **fill**:

  > ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut, color = cut))

  > ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut, fill = cut))

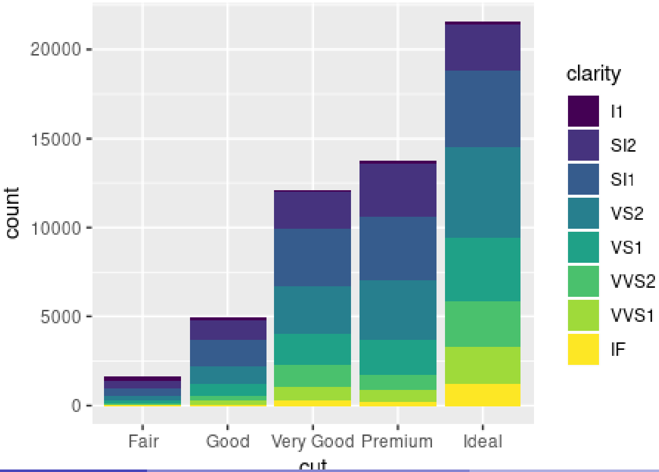




Position Adjustments

- What happens if you map the fill aesthetic to another variable, like *clarity*: the bars are automatically stacked:

  $> ggplot(data = diamonds) + geom\_bar(mapping = aes(x = cut, fill = clarity))$

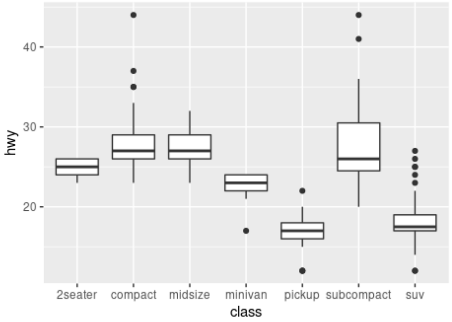


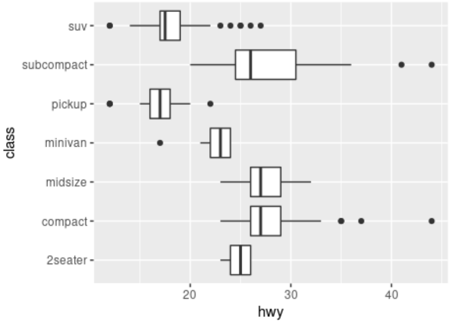
Position Adjustments

- Function **coord_flip()**switches the x- and y-axes. This is useful (for example) if you want horizontal boxplots:

  $> ggplot(data = mpg, mapping = aes(x = class, y = hwy)) + geom\_boxplot()$

  $> ggplot(data = mpg, mapping = aes(x = class, y = hwy)) + geom\_boxplot() + coord\_flip()$

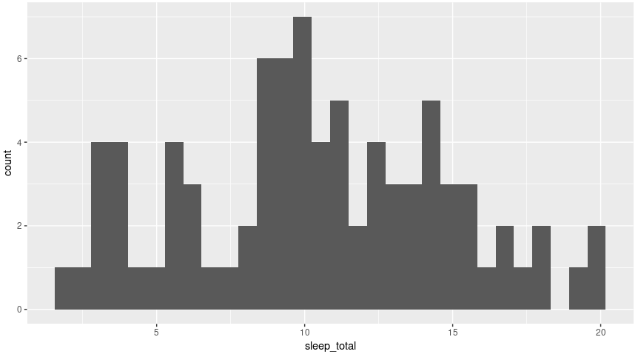




Histogram in ggplot2

- We want to plot into a histogram the hours slept per day (sleep_total)

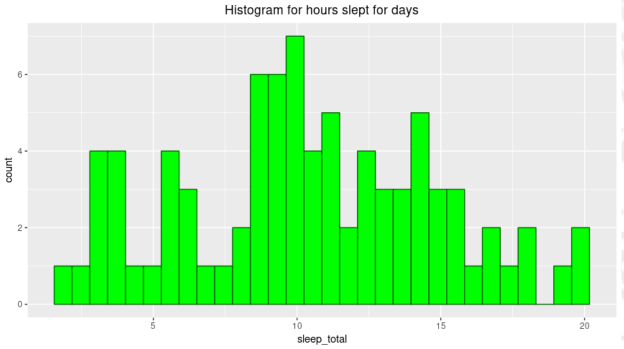
> Hist = ggplot(msleep, aes(sleep_total)) + geom_histogram()



Histogram in ggplot2

How to play with:

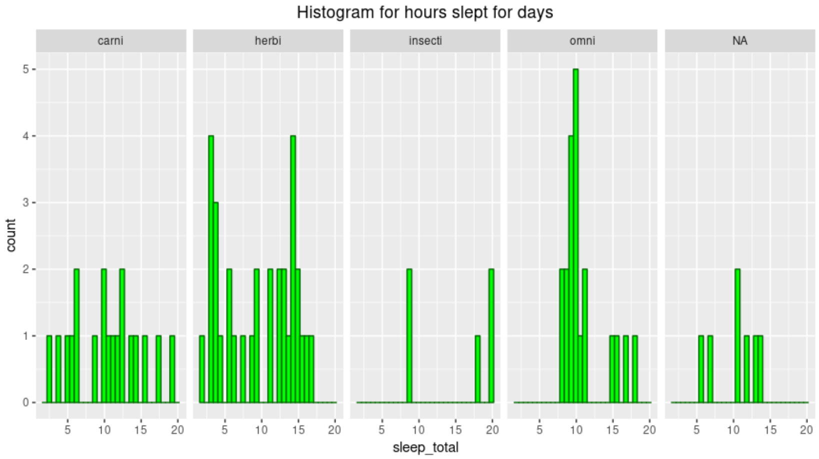
- width of the bins: $+geom\_histogram(binwidth = 0.3)$;
- colour of the bins: $+geom\_histogram(color = "darkgreen", fill = "green")$;
- labels for the axis: $+labs(x = "Count", y = "sleep\_total")$;
- title of the graph: $+ggtitle("Histogram\ for\ hours\ slept\ for\ days")$.
- position of the title: $+theme(plot.title = element\_text(hjust = 0.5))$



Histogram in ggplot2

- How to visualize an histogram for each trophic level (vore) in a single plot

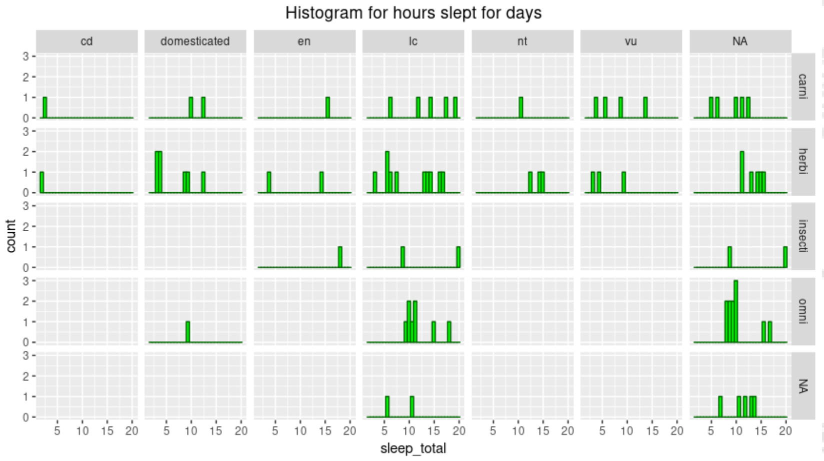
  $> Hist1 = Hist + facet\_grid(\sim vore)$



Histogram in ggplot2

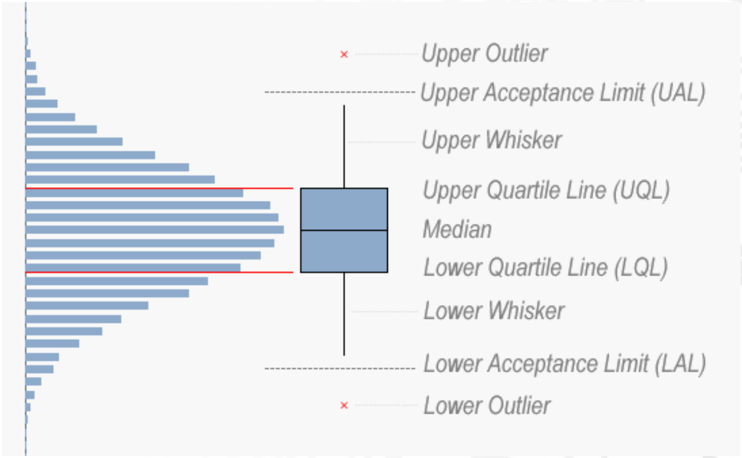
- How to visualize an histogram for each trophic level(vore) and conservation type (conservation) in a single plot

$> Hist1 = Hist + facet\_grid(vore \sim conservation)$



Boxplot in ggplot2

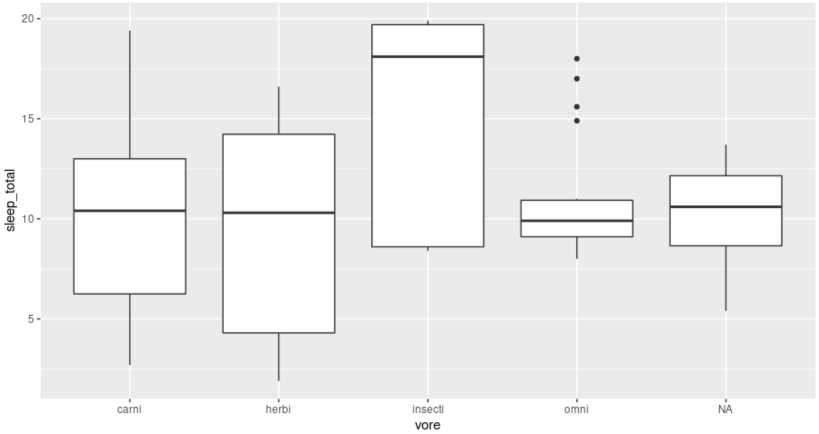
- Boxplot of a variable is a graphical representation based on its quartiles, as well as its smallest and largest values. It attempts to provide a visual shape of the data distribution.


Upper Outlier
Upper Acceptance Limit (UAL)
Upper Whisker
Upper Quartile Line (UQL)
Median
Lower Quartile Line (LQL)
Lower Whisker
Lower Acceptance Limit (LAL)
Lower Outlier

Boxplot in ggplot2

- How to visualize a boxplot for each trophic level (vore) in a single plot

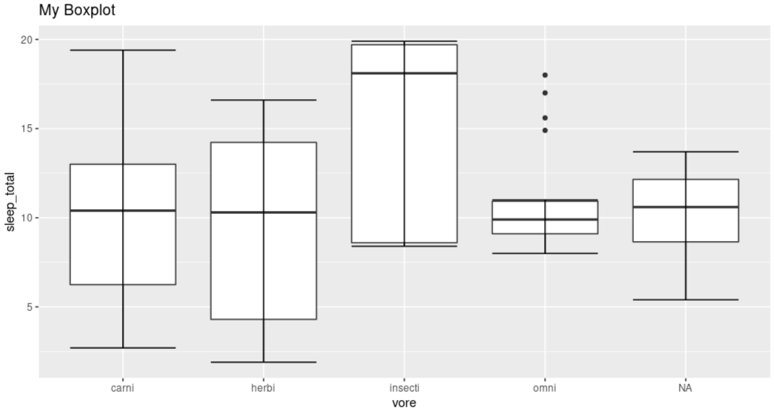
```
> bt = ggplot(msleep, aes(vore, sleep_total)) + geom_boxplot()
```



Boxplot in ggplot2

- to change the look of the whiskers: $+stat\_boxplot(geom = \text{"}errorbar\text{"})$
- to add title $+ggtitle(\text{"}Boxplot\text{"})$

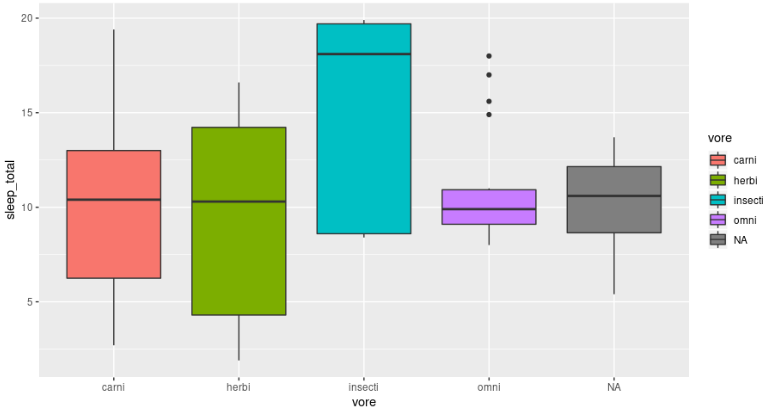
$> bt = bt + stat\_boxplot(geom = \text{"}errorbar\text{"}) + ggtitle(\text{"}My\ Boxplot\text{"})$



Boxplot in ggplot2

- to add color w.r.t. vore:

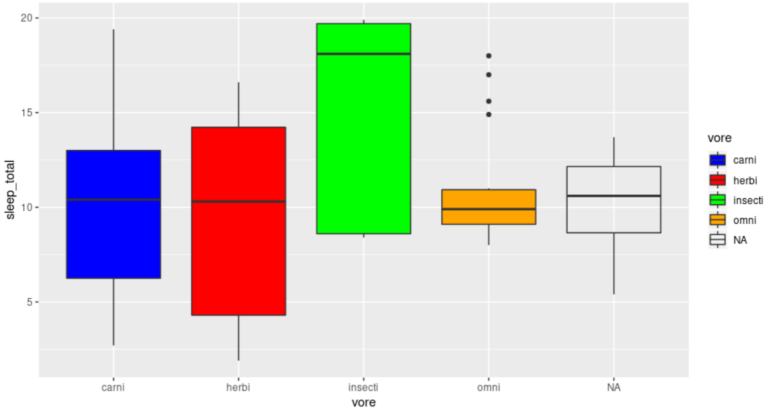
  $> bt = ggplot(msleep, aes(vore, sleep\_total, fill = vore)) + geom\_boxplot()$



Boxplot in ggplot2

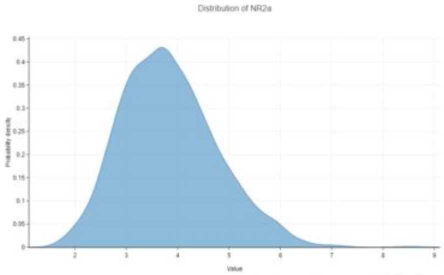
- to personalize the color of each boxes:

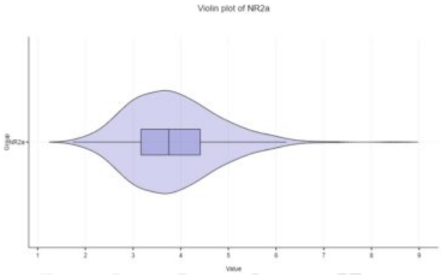
$> bt = bt + scale\_fill\_manual(values = c(\text{"blue"}, \text{"red"}, \text{"green"}, \text{"orange"}))$



Violin plots in ggplot2

- Violin plots are similar to box plots, except that they also show the probability density of the data at different values;
- The shape of a violin plot comes from the data density plot. You just turn that density plot sideway and put it on both sides of the box plot, mirroring each other.

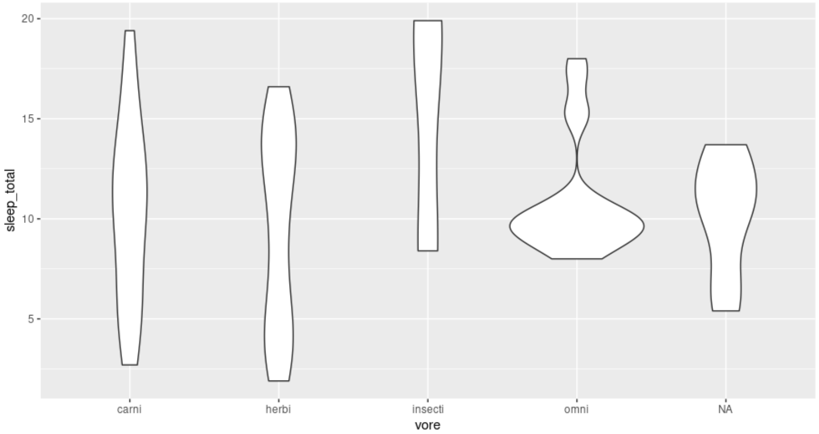




Violin plot in ggplot2

- How to visualize a violin plot for each trophic level (vore) in a single plot

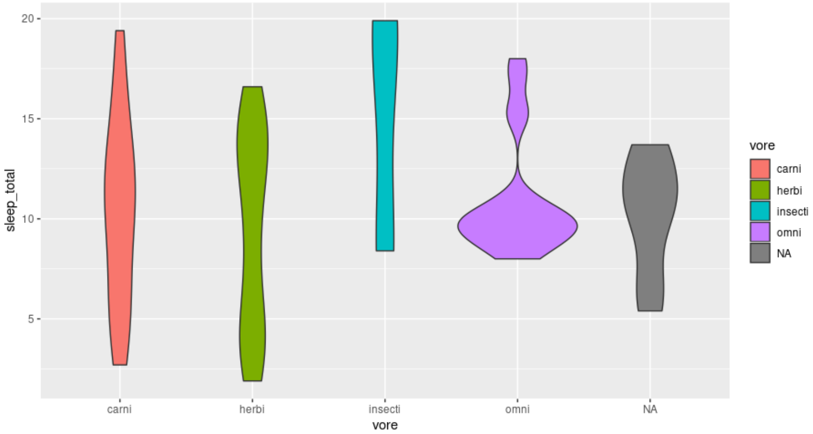
$> vl = ggplot(msleep, aes(vore, sleep\_total)) + geom\_violin()$



Violin plot in ggplot2

- to add color w.r.t. vore:

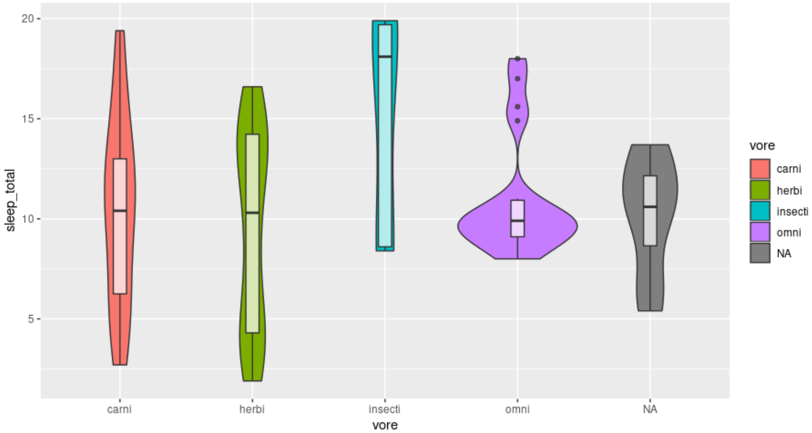
  $> vl = ggplot(msleep, aes(vore, sleep\_total, fill = vore)) + geom\_violin()$



Violin plot in ggplot2

- to add a boxplot inside inside violin plot:

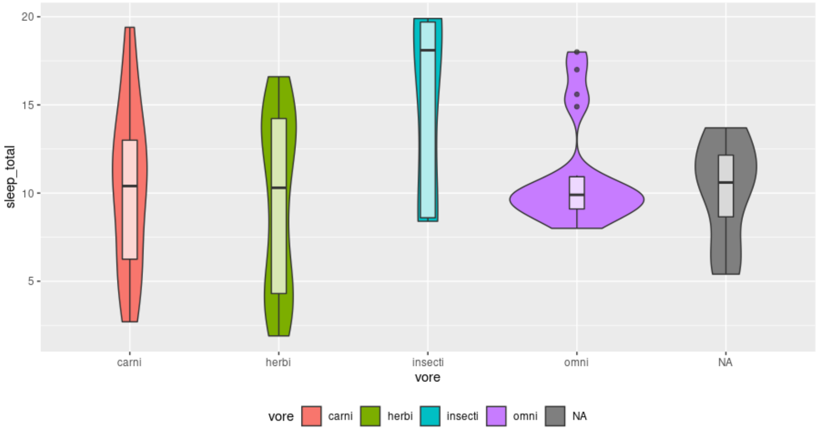
$> vlbox = vl + geom\_boxplot(width = 0.1, fill = \text{“}white\text{''}, alpha = 0.7)$



Violin plot in ggplot2

- to change the legend position:

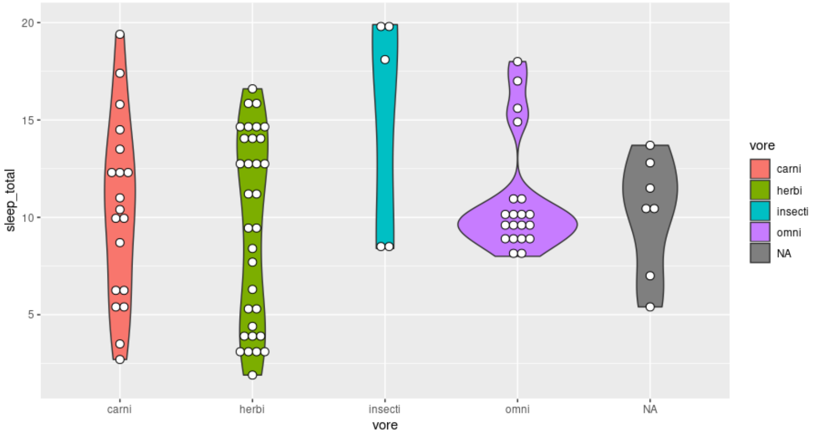
$> vlbox = vlbox + theme(legend.position = \text{“}bottom\text{”})$



Violin plot in ggplot2

- to add points inside the violin plot:

$> vl = vl + geom\_dotplot(binaxis = \text{"}y\text{"}, stackdir = \text{"}center\text{"}, dotsize = 0.7, fill = \text{"}white\text{"})$



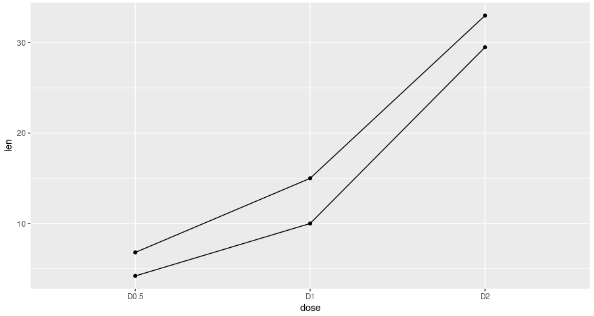
line plot in ggplot2

- Create the following data.frame:

  $> supp = rep(c(\text{“}VC\text{”}, \text{“}OJ\text{”}), each = 3)$
  $> dose = rep(c(\text{“}D0.5\text{”}, \text{“}D1\text{”}, \text{“}D2\text{”}), 2)$
  $> len = c(6.8, 15, 33, 4.2, 10, 29.5)$
  $> df2 = data.frame(supp, dose, len)$

- To create line plot with multiple groups:

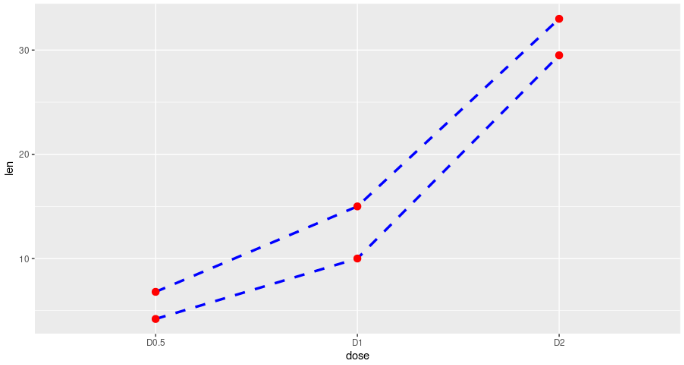
  $> l = ggplot(data = df2, aes(x = dose, y = len, group = supp)) + geom\_line() + geom\_point()$



line plot in ggplot2

- To change line types:

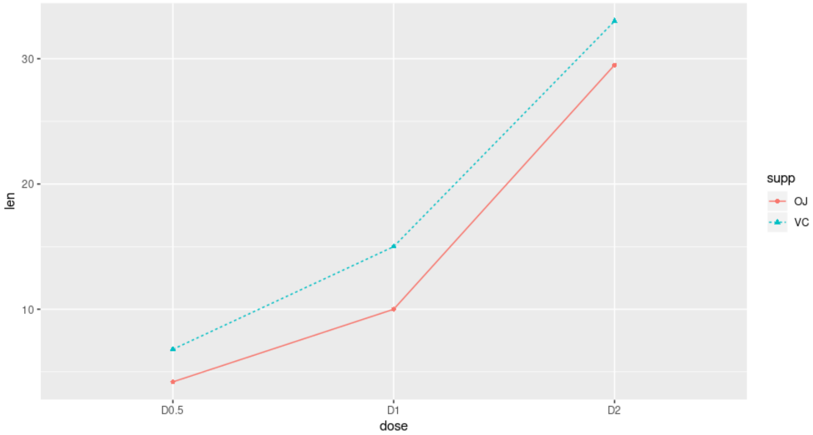
  $> l = ggplot(data = df2, aes(x = dose, y = len, group = supp))$
  $+geom\_line(linetype = “dashed”, color = “blue”, size = 1.2)$
  $+geom\_point(color = “red”, size = 3)$



line plot in ggplot2

- To change line types, line color and point shapes:

$$> l = ggplot(df2, aes(x = dose, y = len, group = supp))$$
$$+geom\_line(aes(linetype = supp, col = supp))$$
$$+geom\_point(aes(shape = supp, col = supp))$$


len
D0.5
D1
D2
dose
supp
OJ
VC