SCwR Visualization Types Summary

Valentin Kodderitzsch

2023-12-27

Introduction

The following document gives a systematic overview of all visualization types covered in the "Statistical Computing with R" course. The classification of the different visualization types is based on my interpretation of their differences.

My main distinction is whether there exists a factor variable or not. A factor variable is a categorical variable containing k unique groups to divide the dataset into. If there is no factor variable then we have only a single group for the entire dataset.

The following table is a hyperlink based index.

Group Comparison	Number of Variables	Variable or Relation Type	Visualization
No	1	Discrete	Histogram
No	1	Continuous	Density Plot
No	2	Discrete	Scatter Plot
No	2	Relationships	Scatter Plot
No	2	Functions	Function Curve
Yes	1	Discrete	Bar Chart
Yes	1	Continuous	Overlapping Density Plots
Yes	2	Relationships	Overlapping Scatter Plots
Yes	2	Functions	Overlapping Function Curves
Yes	2	Correlations	Correlogram
Yes	2	Longitudinal	Overlapping Trajectory Plots

No Groups

In the case that there is no factor variable we are not comparing groups. This is arguably the easier option of the two cases as we only have one group.

1 Variable (No Groups)

Visualizing 1 variable with no groups is arguably the easiest visualization. Here, one more distinction needs to be made between discrete and continuous variables. However, this distinction is often context dependent.

A continuous random variable has an uncountably infinite set of possible values it can take. One example is measured length. In theory, the sepal length of an iris flower for instance is a continuous random variable since the set of possible lengths is uncountably infinite. This means that if we imagine the number line and are given the value of element n, we cannot know the value of the $n+1$th element as there is no defined step size.

On the other hand, a discrete random variable has a countably infinite set of values it can take. Countably infinite means that we know the value of the $n+1$th element, if we are given the n element. This is because the step size is defined, hence the set becomes countable. An example would be the number of pebbles on a beach. If said number was n and we found another pebble, the total count would be updated to $n+1$ as we found one more pebble.

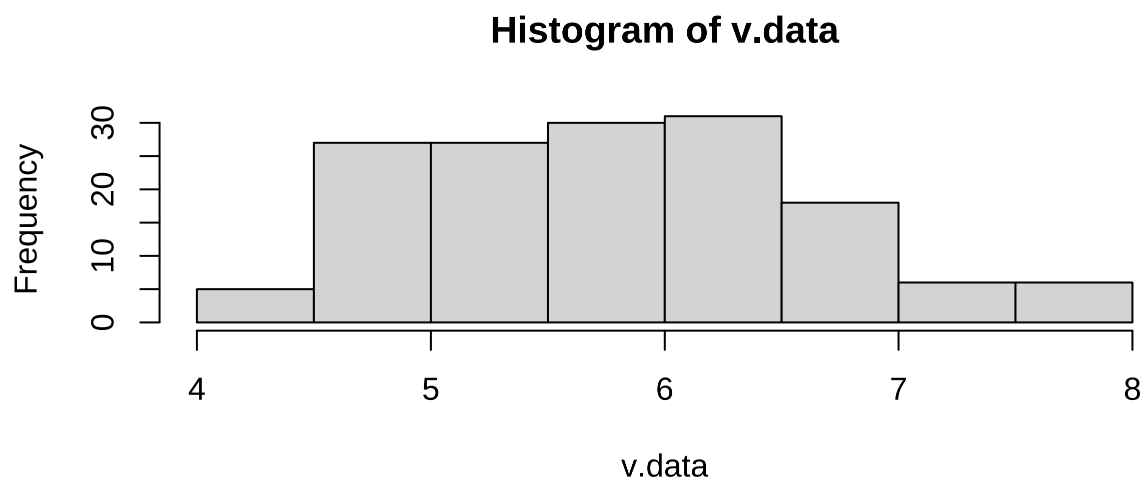
For practical purposes, an uncoutably infinite random variable is often viewed as a countably infinite random variable. Length for instance is often made discrete with a fixed step size of 1 mm, cm, m, etc. to make values easier to work with, e.g. to make computations more (memory) efficient. Additionally, the extra precision is often not needed. E.g. on a practical level if the sepal length of the iris flower is 53.11 mm long, no important information is lost when rounding to 53 mm.

Discrete - 1 Variable (No Groups)

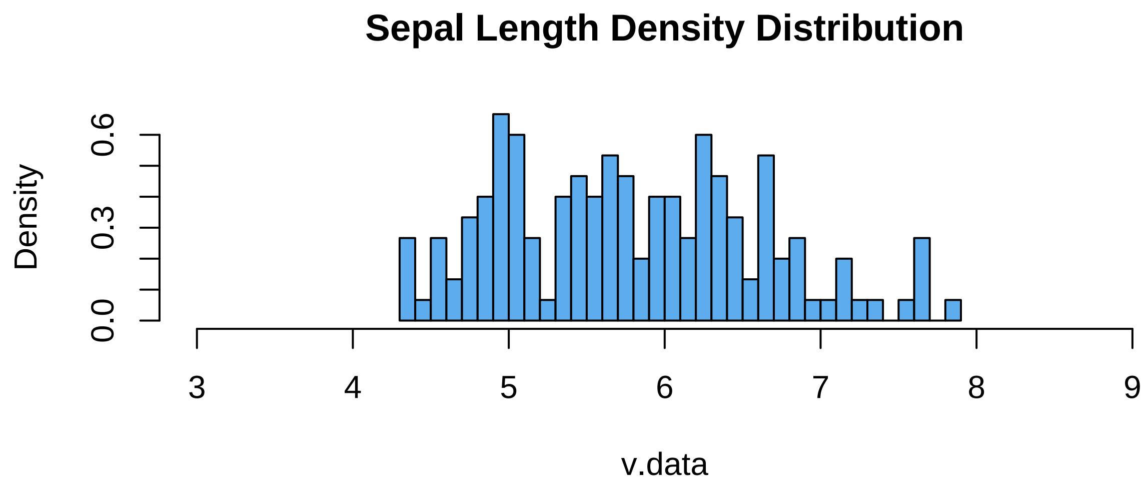
Visualize the sepal length of the iris dataset. Choose the discrete version of length.

→ **Histogram**

```r
# 1 variable = 1 vector
v.data = iris$Sepal.Length

# Histogram with density - basic
hist(v.data)
```



```r
# Histogram with density - fancy
hist(v.data, breaks = 50, main = "Sepal Length Density Distribution", probability = T,
     xlim = c(3, 9), col = 'steelblue2')
```

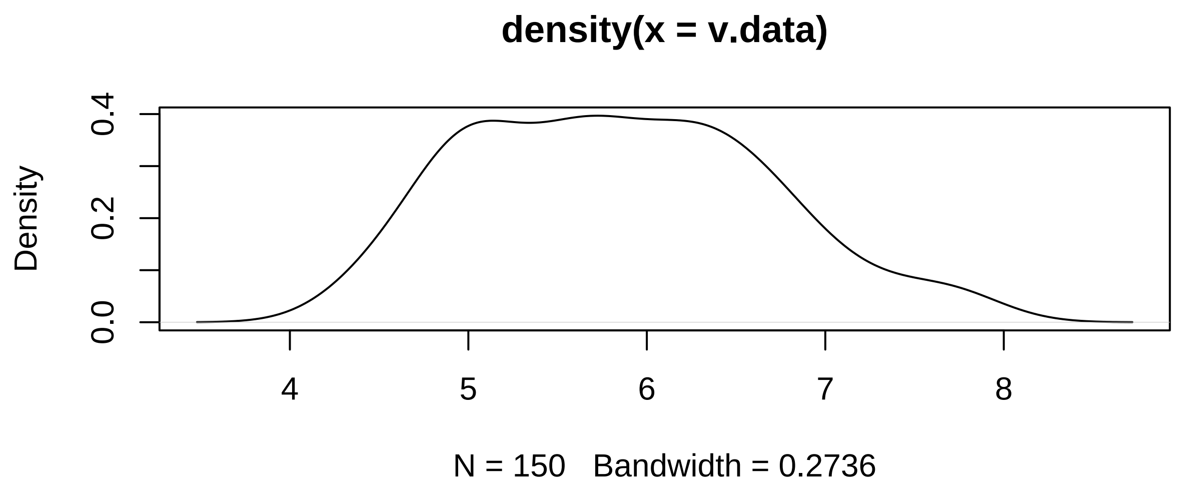


Continious - 1 Variable (No Groups)

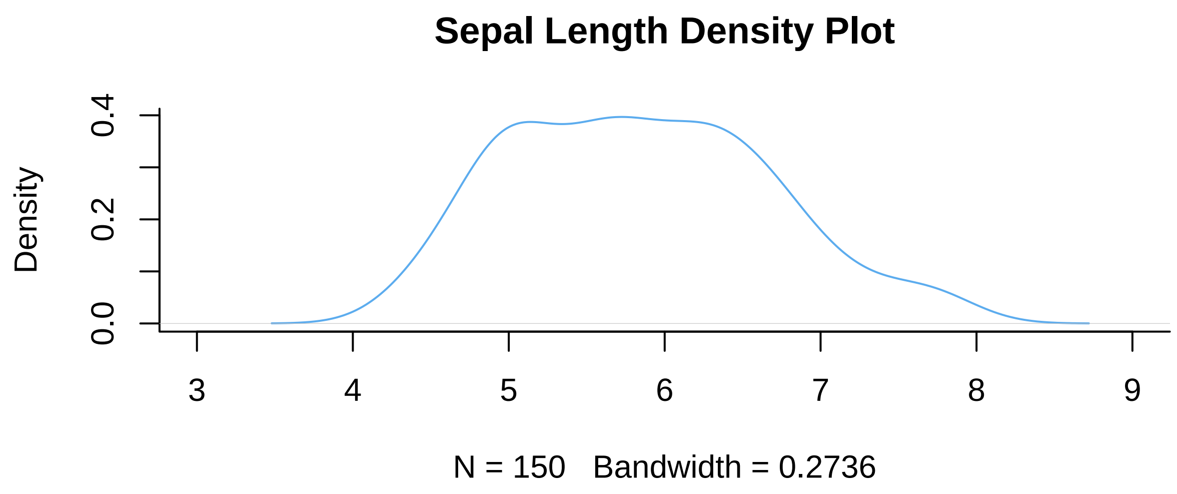
Visualize the sepal length of the iris dataset. Choose the continuous version of length.

→ **Density plot**

```r
# Density plot - basic
plot(density(v.data))
```



```r
# Density plot - fancy
par(bty = 'l')
plot(density(v.data), main = "Sepal Length Density Plot",
     xlim = c(3,9), col = 'steelblue2')
```



The density plot can be viewed as a smooth histogram.

2 Variables (No Groups)

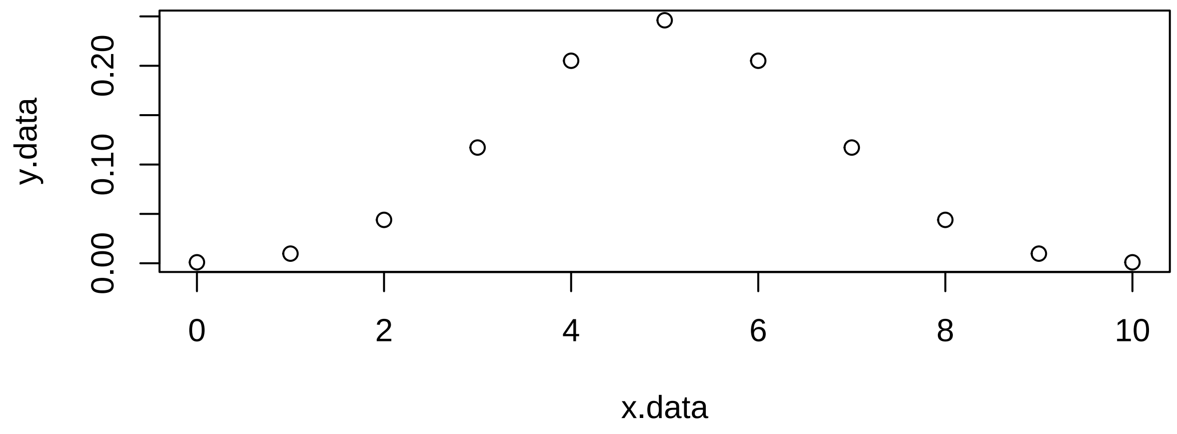
When plotting 2 variables against each other and there are no group comparisons. Another way to think about it is that we need k overlaying plots for k groups. In the "no" group case $k = 1$ as we have exactly one group. Thus, we only need to create a single plot.

Discrete - 2 Variables (No Groups)

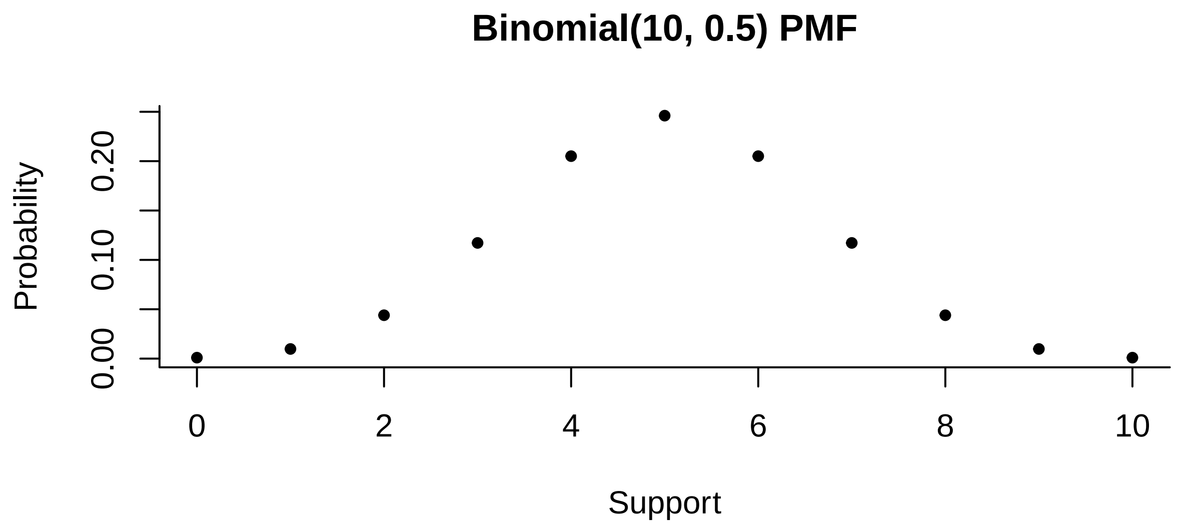
Probability mass functions (PMF) are discrete, non-continuous functions. An example of a PMF is the Binomial distribution.

→ **Scatter plot**

```r
# 2 variables = 1 data frame or 2 separate vectors
x.data = 0:10
y.data = dbinom(x.data, 10, 0.5)

# PMF - basic
plot(y.data ~ x.data)
```



```r
# PMF - fancy
par(bty = 'l')
plot(y.data ~ x.data, xlab = "Support", ylab = 'Probability',
     main = "Binomial(10, 0.5) PMF", pch = 20, cex = 1)
```

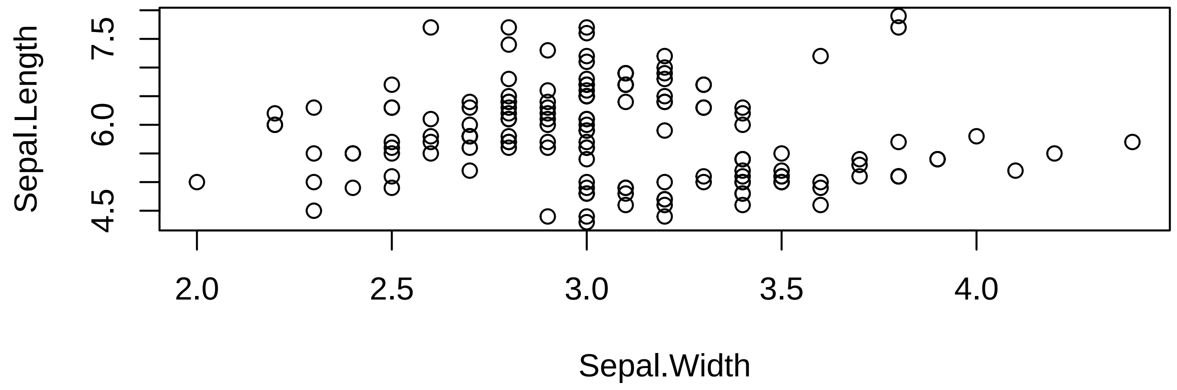

Binomial(10, 0.5) PMF
Probability
0.00
0.10
0.20
0
2
4
6
8
10
Support

Relationships, 2 Variables (No Groups)

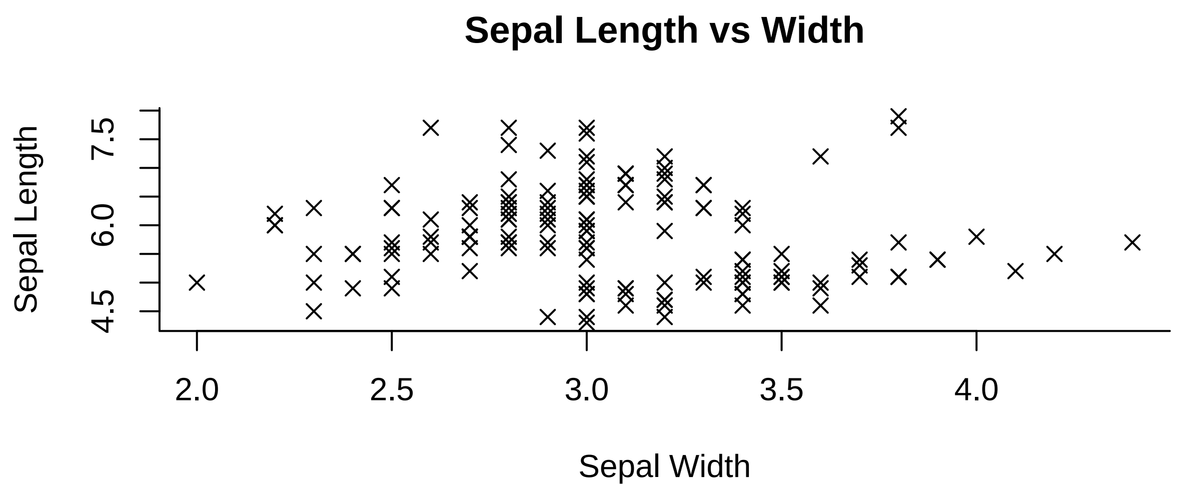
The relationship between 2 variables can be captured using a scatter plot. Said plot captures the correlation between 2 variables. But it is not a function, as the “single valued” property of a function is violated. In other words, the output is not uniquely determined by the input, so there can be multiple y values for the same x value.

→ **Scatter plot**

```r
# Scatter plot - basic
plot(Sepal.Length ~ Sepal.Width, data = iris)
```



```r
# Scatter plot - fancy
par(bty = 'l')
plot(y = iris$Sepal.Length, x = iris$Sepal.Width,
     main = "Sepal Length vs Width", ylab = "Sepal Length",
     xlab = "Sepal Width", pch = 4)
```



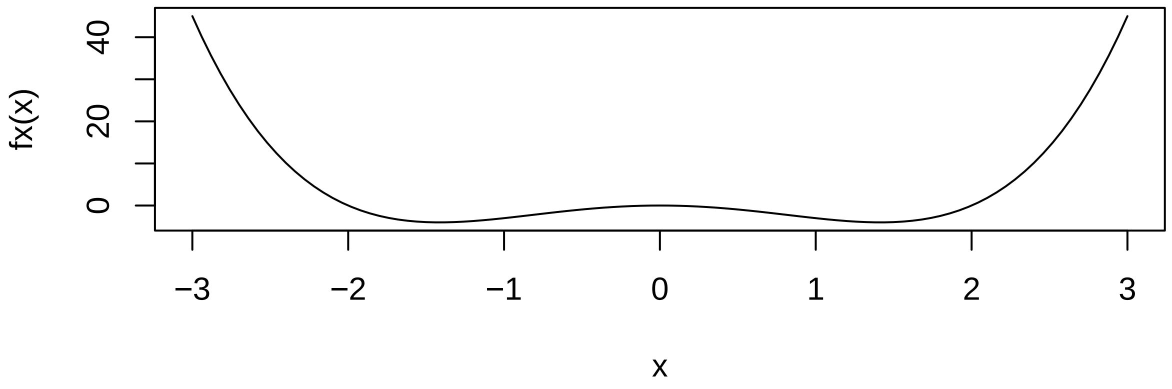
Functions - 2 Variables (No Groups)

Functions can be drawn using the `curve` function. Mathematically, a function f is defined as $f : X \to Y$, where X and Y are both sets. For every element $x \in X$, there must a **unique** $y \in Y$ such that a set of ordered pairs (x, y) can be constructed which is the function.

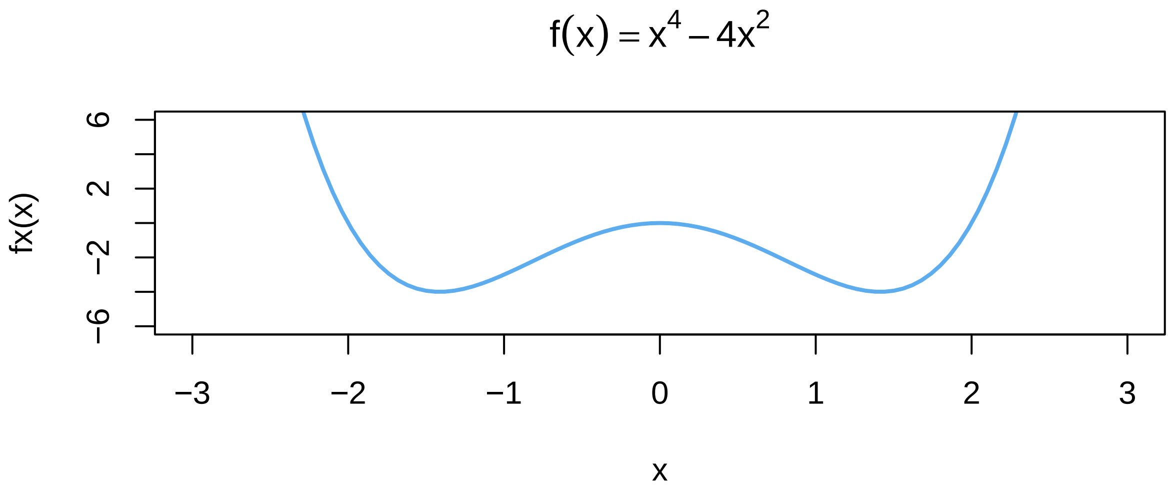
Due to this (strict) definition most observed data cannot be plotted as a function. However, a regression line which is a mathematical function can be "plotted" as a function.

$\rightarrow$ **Function curve**

```
# Define a function
fx = function(x) x^4 - 4*(x^2)

# Function curve - basic
curve(fx, from = -3, to = 3)
```



```
# Function curve - fancy
library(latex2exp)
curve(fx, from = -3, to = 3, ylim = c(-6, 6), xlim = c(-3, 3),
      main = TeX("$f(x)=x^4 - 4x^2$"), col = 'steelblue2', lwd = 2)
```



Comparing Groups

Often when visualizing data we are interested in comparing different groups against each other. This means that there exists another categorical variable which stores the different factors, i.e. groups.

A factor variable usually has a fixed number of unique groups/values. If a factor variable has k groups, then we need to overlap k plots. It makes sense to have a unique color/shape per plot so that the individual factors can be distinguished in the final plot. In most cases there is an automatic way of coloring the labels. If not colors can always be applied to the labels in a manual way. In both cases it is important to add a legend so that the reader knows which color corresponds to which label.

1 Variable (Comparing Groups)

When we have 1 variable only, we often end up computing a "frequency table" as a first step.

Discrete - 1 Variable (Comparing Groups)

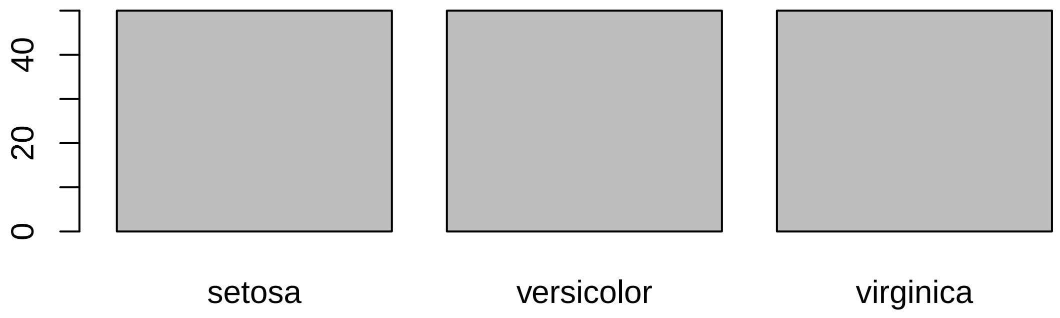
My interpretation of discrete applies to counting occurrences of a certain label, i.e. computing the frequency table. An example would be the number of votes per party.

It is possible to overlay histograms as well for actual discrete variables. An example would be sepal length per iris species. However, this is often more tedious compared to overlaying density plots. Thus, a histogram visualization will be omitted.

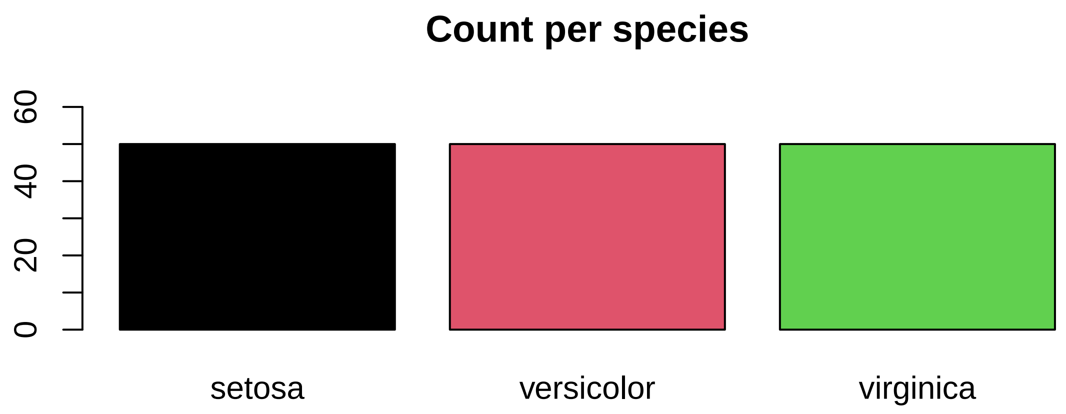
Frequency tables can be visualized as bar charts, pie charts and waffle plots. Personally, I find bar charts to be the most efficient visualisation tool.

→ **Frequency table, bar chart**

```r
# Compute the count per species table
df = table(iris$Species)

# Bar chart - basic
barplot(df)
```



```r
# Bar chart - fancy
barplot(df, col = 1:3, main = "Count per species", ylim = c(0, 60))
```


Count per species
0
20
40
60
setosa
versicolor
virginica

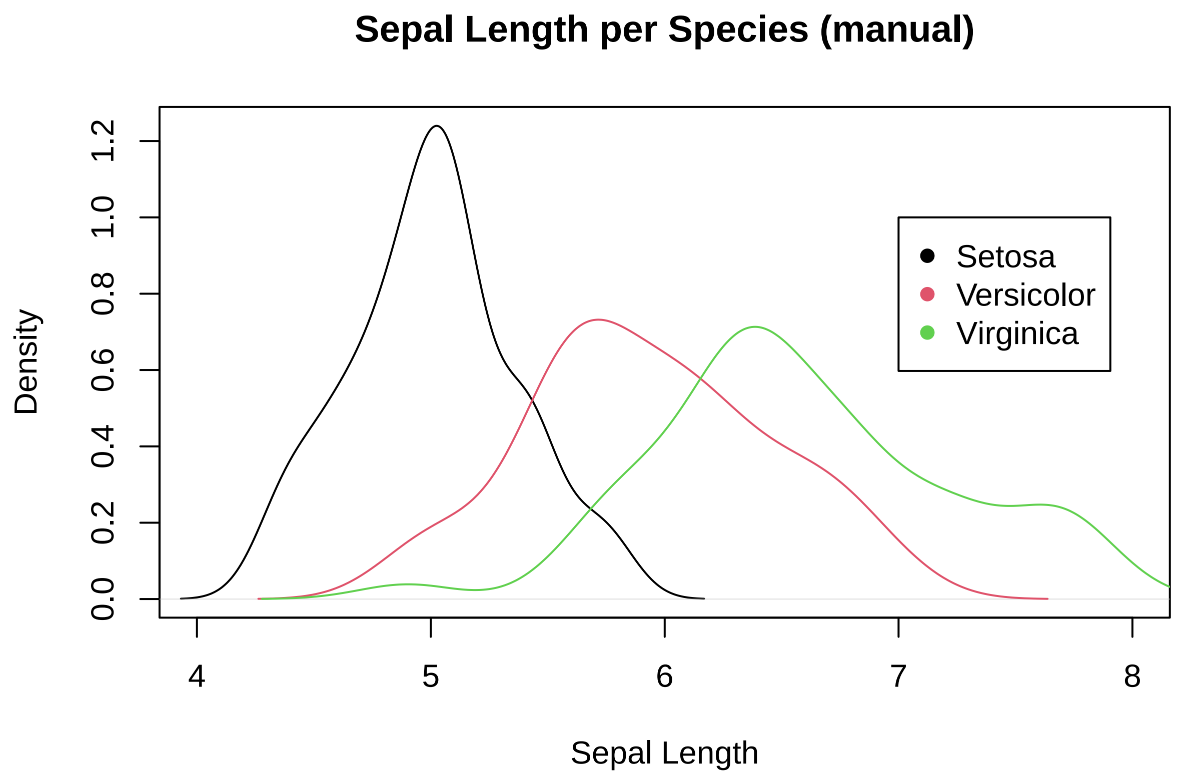
Continious - 1 Variable (Comparing Groups)

Visualizing one continuous variable to compare groups within said variable is very similar to visualizing one group only. The main visualization tool is a density plot which will be overlapped k times. However, it is also possible to create a box plot (shows quantiles) or a violin plot (shows both quantiles & density).

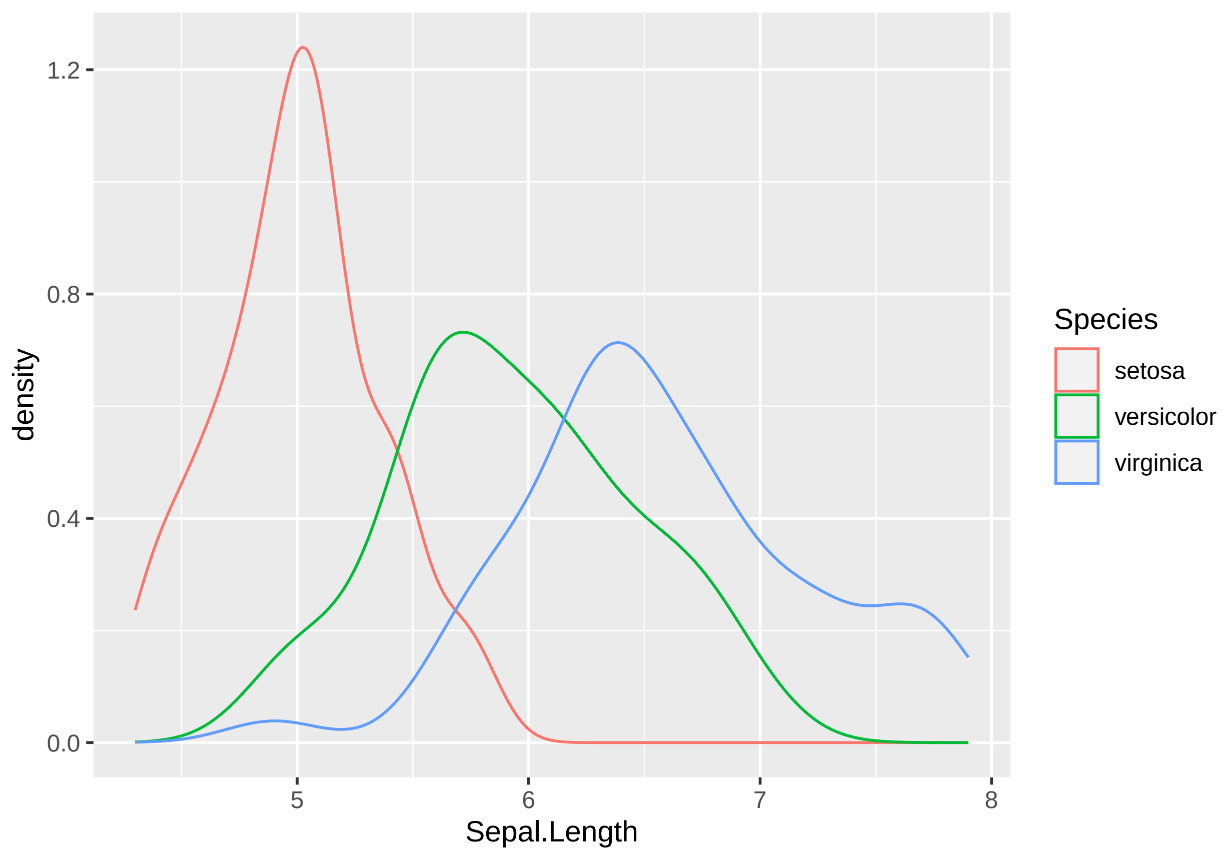
Personally, I believe that a density plot is the most efficient tool as it conveys the same information without being overloaded. However, there is no automated way of overlapping densities unless you use `ggplot2`. Thus, for the manual way you need to subset the dataset per species.

→ **Overlapping Density plot with legend**

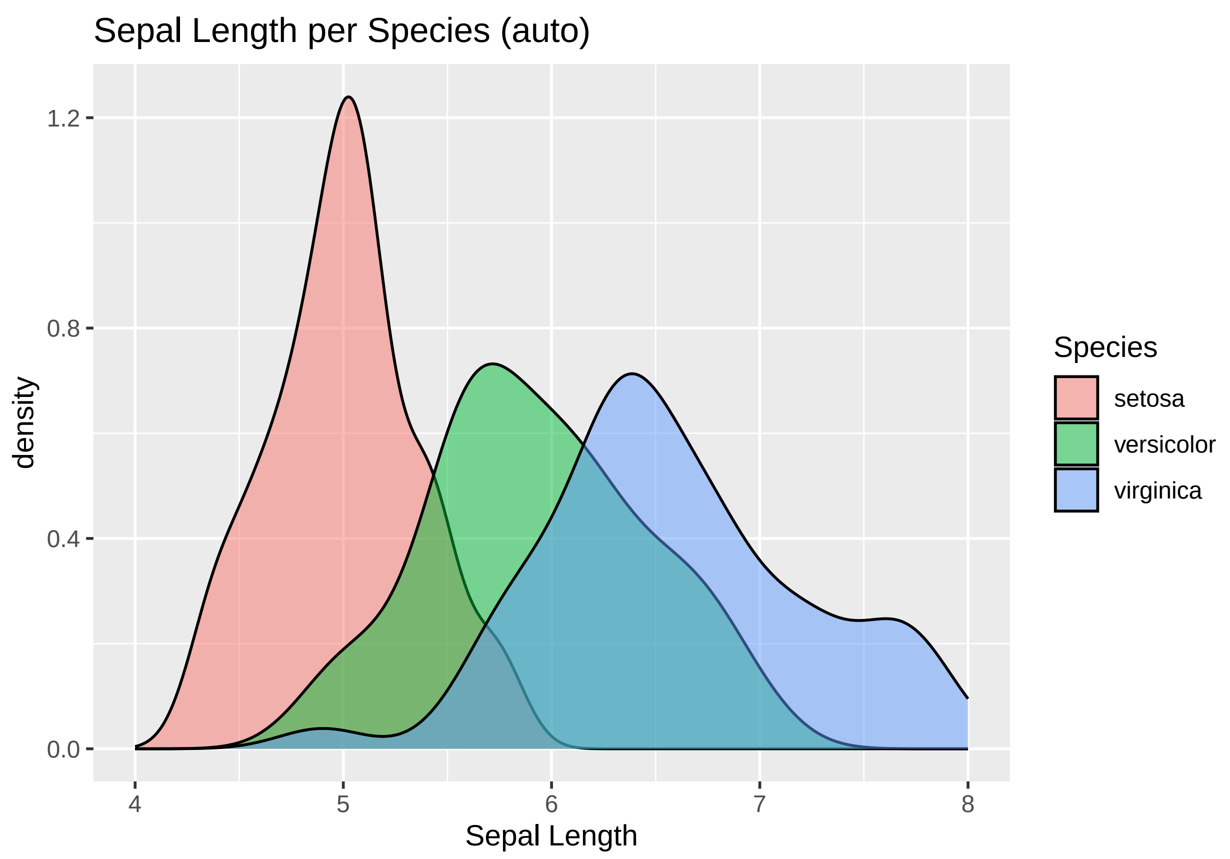
```r
# Subset the dataset per species
library(dplyr)
set = iris |> filter(Species == "setosa") |> select(Sepal.Length, Sepal.Width)
vers = iris |> filter(Species == "versicolor") |> select(Sepal.Length, Sepal.Width)
vir = iris |> filter(Species == "virginica") |> select(Sepal.Length, Sepal.Width)

# Overlapping densities - manual
plot(density(set$Sepal.Length), col = 1, xlim = c(4, 8),
     main = "Sepal Length per Species (manual)", xlab = "Sepal Length")
lines(density(vers$Sepal.Length), col = 2)
lines(density(vir$Sepal.Length), col = 3)
species = c("Setosa", "Versicolor", "Virginica")
legend(x = 7, y = 1, legend = species, col = 1:3, pch = 16)

# Overlapping densities - auto - basic
library(ggplot2)
```



```
ggplot(data = iris,
       mapping = aes(x = Sepal.Length, group = Species, col = Species)) +
  geom_density()
```



```r
# Overlapping densities - auto - fancy
library(ggplot2)
ggplot(data = iris,
       mapping = aes(x = Sepal.Length, group = Species, fill = Species)) +
  geom_density(alpha = 0.5) +
  ggtitle("Sepal Length per Species (auto)") +
  xlab("Sepal Length") +
  xlim(4,8)
```



Sepal Length per Species (auto)
1.2
0.8
0.4
0.0
density
4
5
6
7
8
Sepal Length
Species
setosa
versicolor
virginica

2 Variables (Comparing Groups)

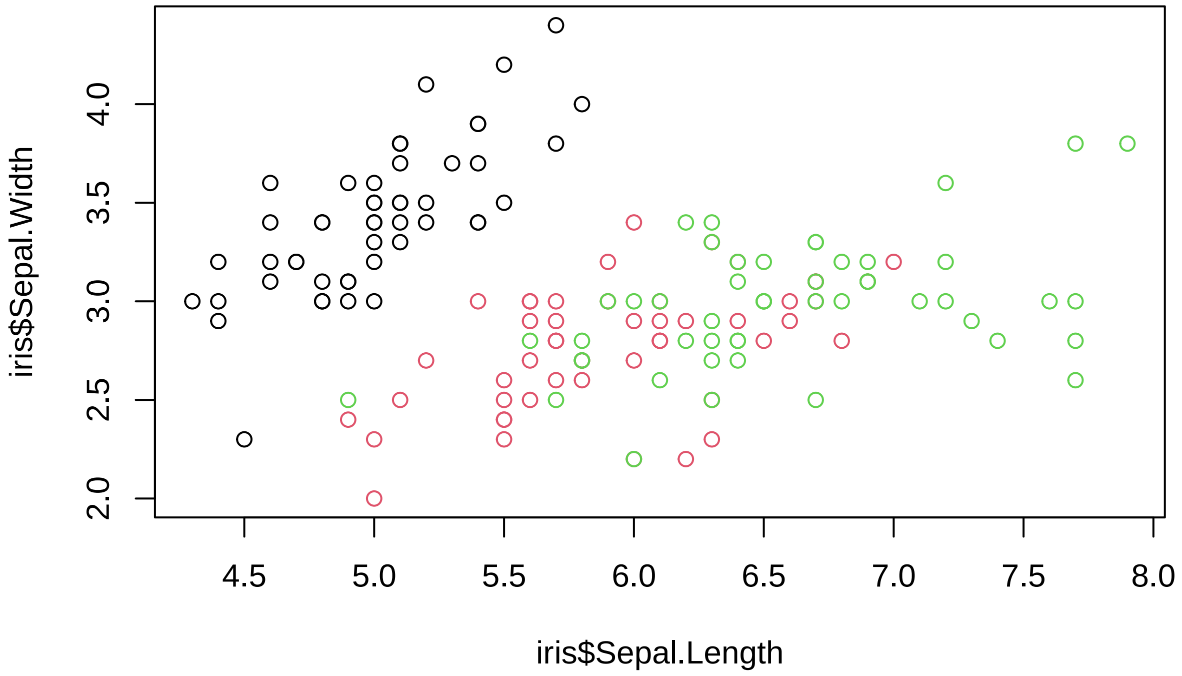
Visualizing 2 variables to compare groups is arguably the most challenging visualization type.

Relationships - 2 Variables (Comparing Groups)

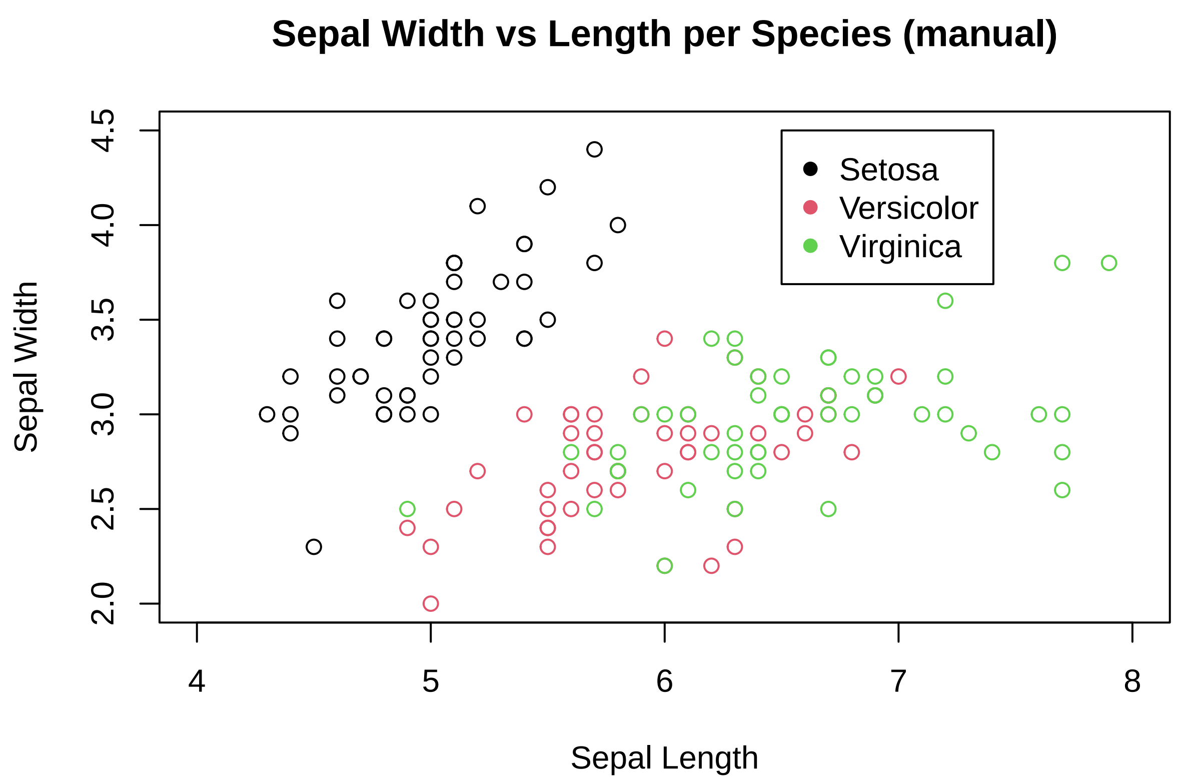
Here we have to overlay k scatter plots for k unique labels. The base R way is the quickest/easiest if you are okay with omitting labels.

→ **Overlapping Scatter plot with legend**

```r
# Overlapping scatter plot - basic -  auto
plot(x = iris$Sepal.Length, y = iris$Sepal.Width, col = iris$Species)
```

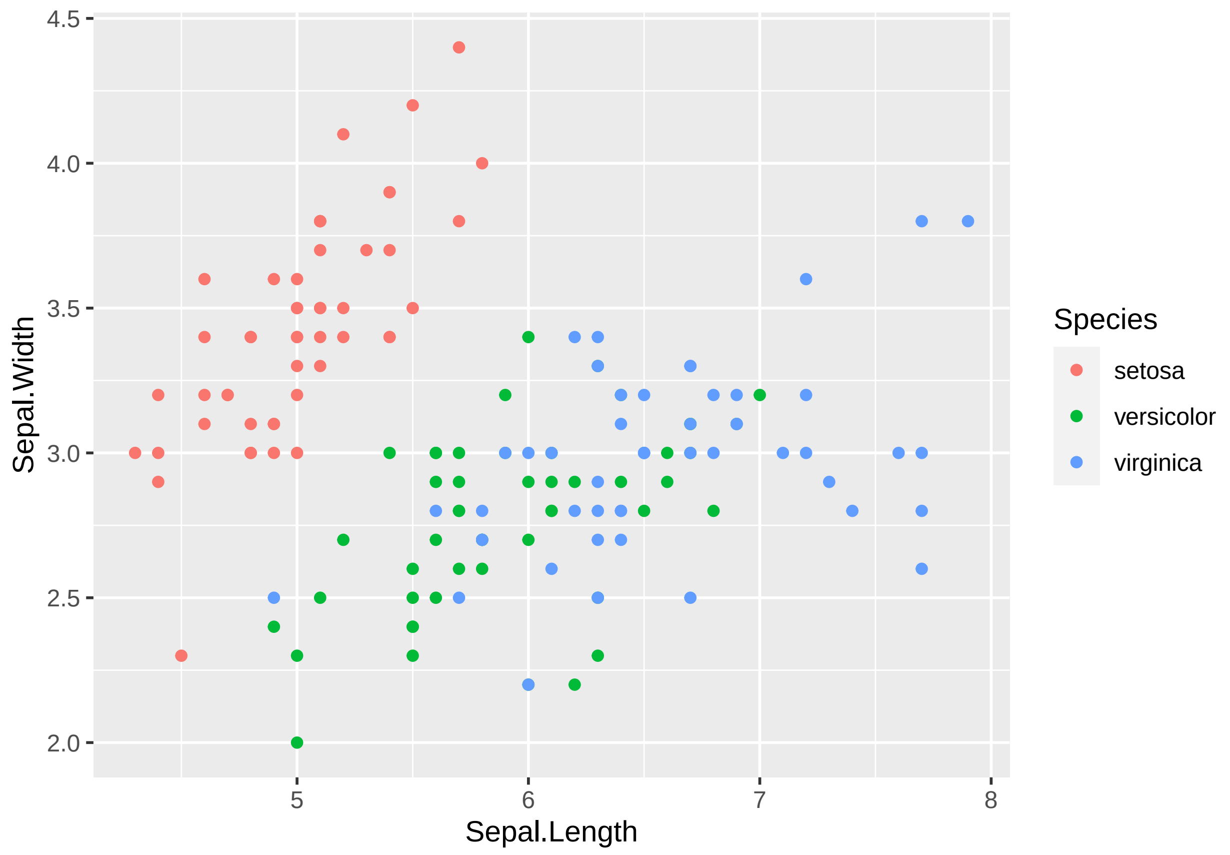


```r
# Overlapping scatter plot - fancy - manual
plot(Sepal.Width ~ Sepal.Length, data = set, col = 1, xlim = c(4, 8), ylim = c(2, 4.5),
     main = "Sepal Width vs Length per Species (manual)",
     ylab = "Sepal Width", xlab = "Sepal Length")
points(Sepal.Width ~ Sepal.Length, data = vers, col = 2)
points(Sepal.Width ~ Sepal.Length, data = vir, col = 3)
legend(x = 6.5, y = 4.5, species, col = 1:3, pch = 16)
```



However, the `ggplot2` way is the most efficient way to create an overlapping scatter plot with an auto-generated legend.

```
# Overlapping scatter plot with legend - basic
ggplot(data = iris,
       mapping = aes(x = Sepal.Length, y = Sepal.Width, group = Species, color = Species)) +
  geom_point()
```

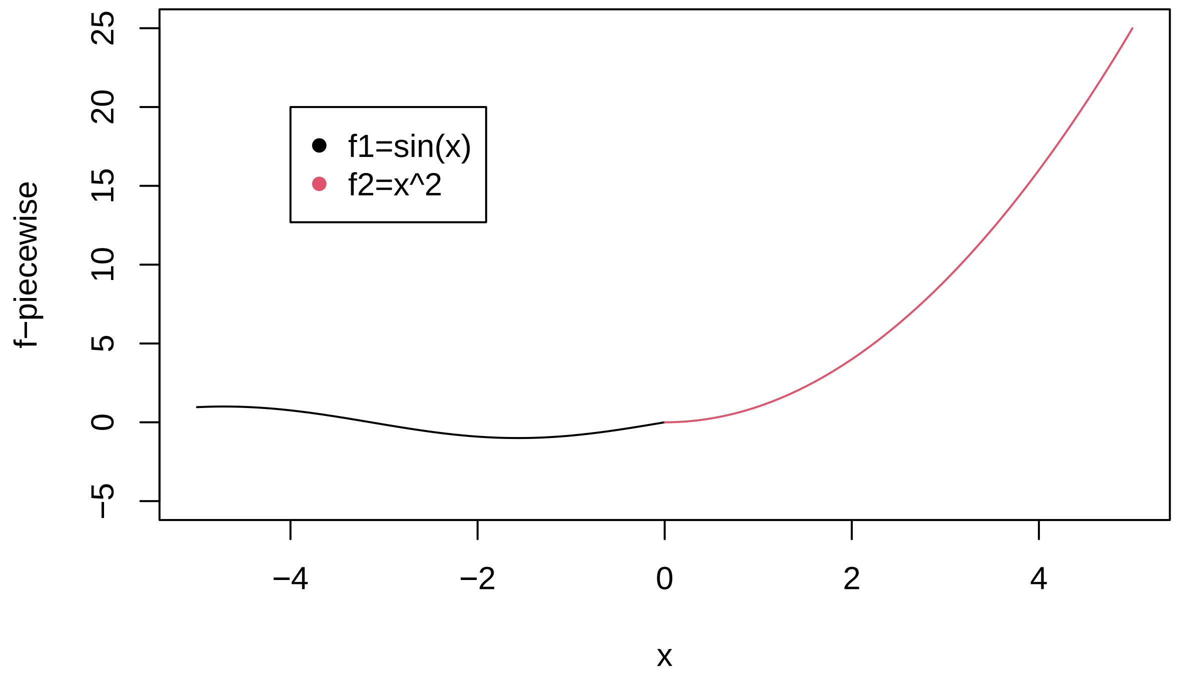

4.5
4.0
3.5
3.0
2.5
2.0
Sepal.Width
5
6
7
8
Sepal.Length
Species
setosa
versicolor
virginica

Functions - 2 Variables (Comparing Groups)

Overlapping functions might appear to be an odd concept at first. However, when considering a piece-wise defined function it can make sense to visualize the different pieces using different colors.

→ **Overlapping Function curves with legend**

```r
f1 = function(x) sin(x)
f2 = function(x) x^2

curve(f1, from = -5, to = 0, col = 1, xlim = c(-5, 5), ylim = c(-5, 25),
      ylab = "f-piecewise")
curve(f2, from = 0, to = 5, add = T, col = 2)
legend(x = -4, y = 20, legend = c('f1=sin(x)', 'f2=x^2'), col = 1:2, pch = 16)
```



Correlogram - 2 Variables (Comparing Groups)

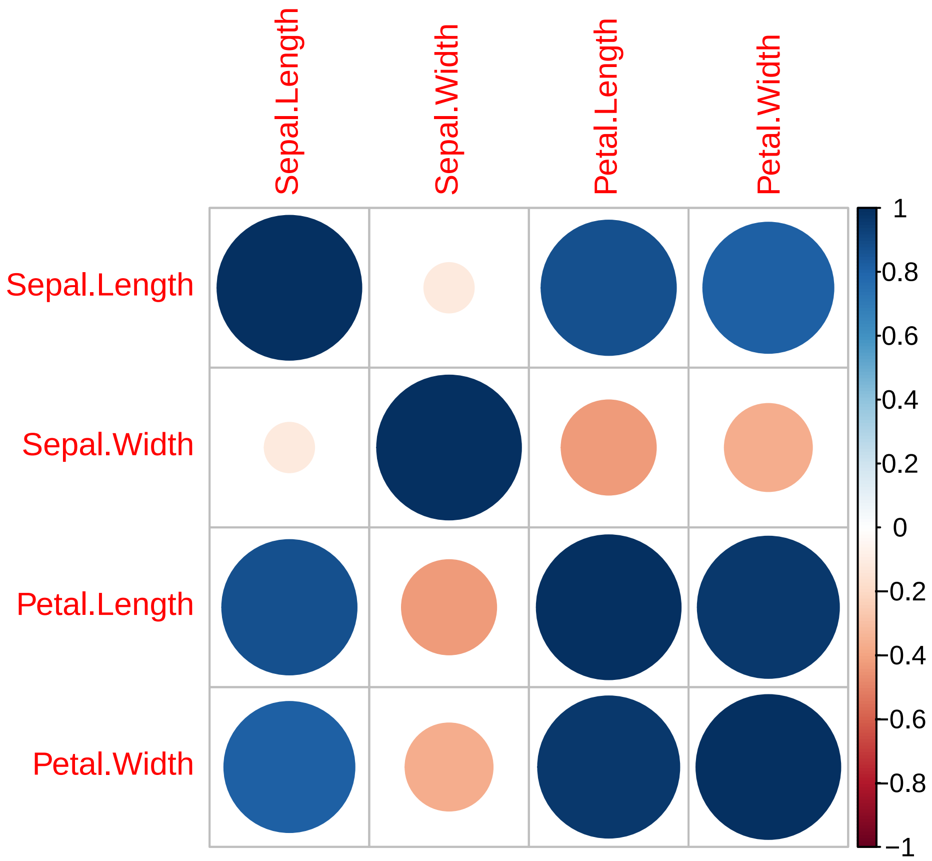
A correlogram is a visualization of a correlation matrix. A correlation can only be computed between 2 numerical variables. However, the correlogram visualizes the the correlation of every variable with every other variable. The colors correspond to the correlation coefficient which is between -1 and 1.

→ **Correlogram**

```r
# Compute the correlation matrix
df_num = iris[-5]
cor.m = cor(df_num)
print(cor.m)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

```r
# Plot the correlation matrix - basic
library(corrplot)
corrplot(cor.m)
```



Longitudinal - 2 Variables (Comparing Groups)

So far all visualizations have been visualizations of "wide" data. Wide data refers to a dataset where each row represents one observation. However, depending on the domain "long" data might be more common. Long data (i.e longitudinal) means that multiple rows represents one observation tracked over time. This is common in medical data, e.g. taking blood measurements of several patients over time.

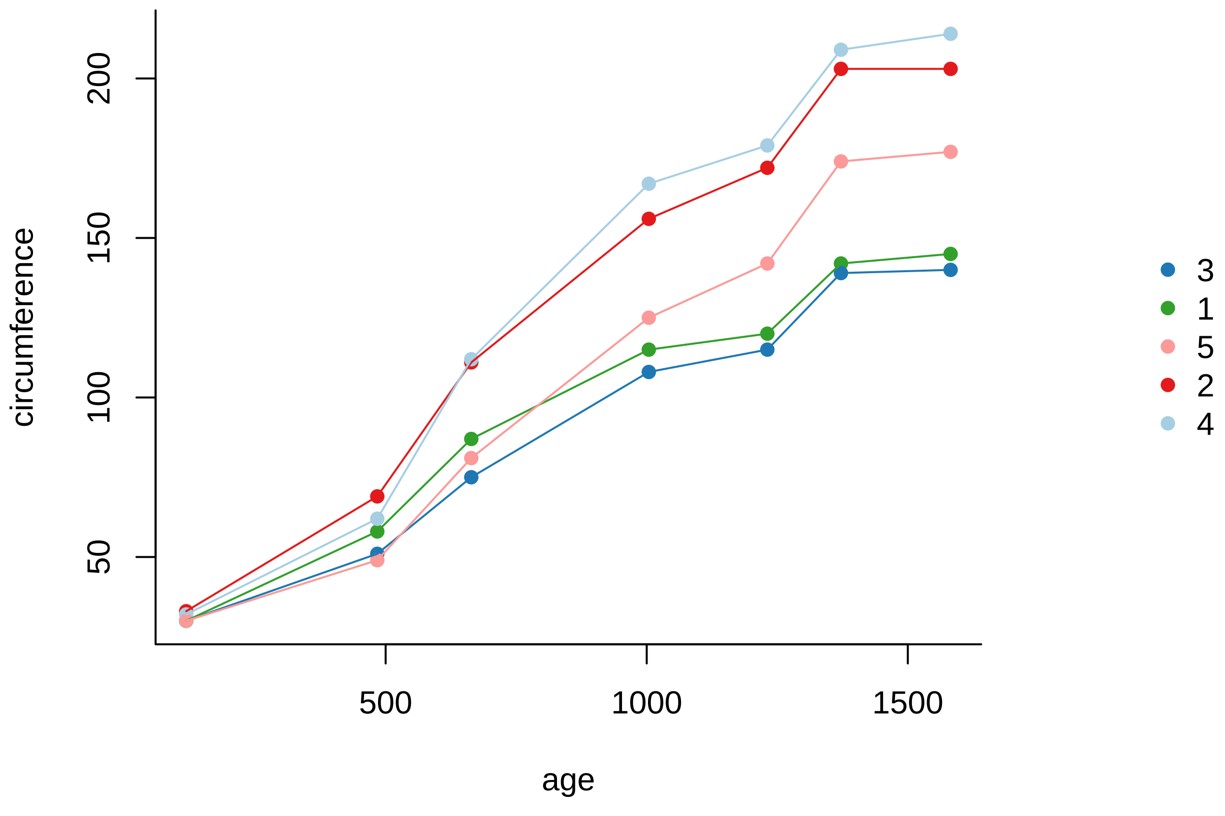
Longitudinal data is best visualized using a trajectory plot. Said plot overlaps the trajectory of *k* individuals. The plot itself is a "connected" scatter plot where *y* represents the measurement of interest, e.g blood sample. The x-axis usually represents time.

→ **Overlapping Trajectory plots with legend**

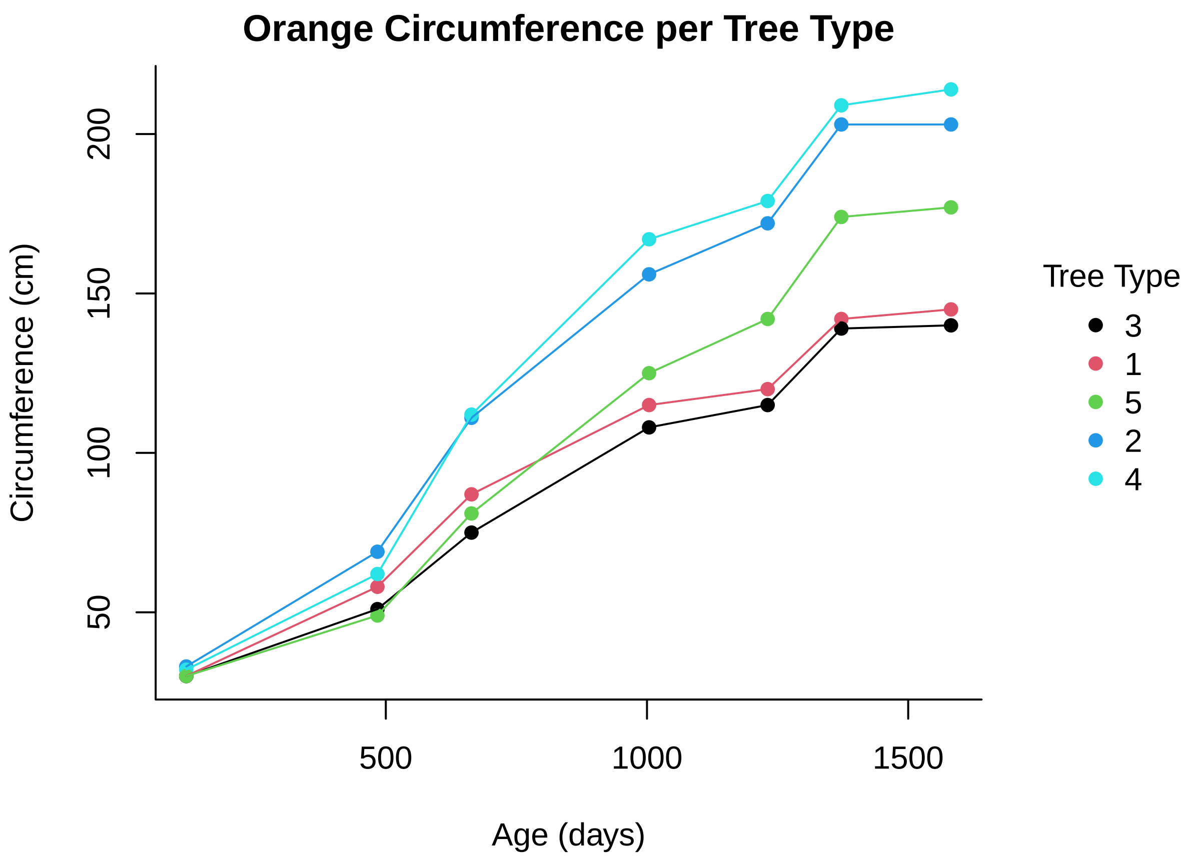
```
# Orange dataset measures the circumference of 6 orange types at 5 time points
head(Orange)
```

```
##   Tree  age circumference
## 1    1  118            30
## 2    1  484            58
## 3    1  664            87
## 4    1 1004           115
## 5    1 1231           120
## 6    1 1372           142
```

```
# Trajectory plot - basic
library(ptmixed)
make.spaghetti(x = age, y = circumference, id = Tree, group = Tree, data = Orange)
```



```r
# Trajectory plot - fancy
library(ptmixed)
make.spaghetti(x = age, y = circumference, id = Tree, group = Tree, data = Orange,
               title = "Orange Circumference per Tree Type", ylab = "Circumference (cm)",
               xlab = "Age (days)", legend.inset = -0.25, col = 1:6,
               legend.title = "Tree Type")
```



It is often necessary to convert between long and wide format which can be done using `pivot_wider` and `pivot_longer` functions from the `tidyr` package.

Final Words

To conclude, I personally believe that there exist 11 different visualization types. Of course there exist many more visualizations, however, I believe that these additional visualizations boil down to the 11 types I have shown above. An example would be box-plots which boil down to be a group comparison of 1 continuous variable, i.e. an overlapping density plot based on my interpretation.