Exploratory data analysis - Graphical summaries

Histograms

Histograms are useful graphical way of representing a data set and allows you to spot patterns that were otherwise not visible in the raw dataset.

Take for example this dataset of randomly generated numbers sampled from a normal distribution:

Table 1: Randomly generated normal dataset

0.8747092	1.0367287	0.8328743	1.3190562	1.0659016	0.8359063	1.0974858	1.1476649	1.1151563	0.9389223
1.3023562	1.0779686	0.8757519	0.5570600	1.2249862	0.9910133	0.9967619	1.1887672	1.1642442	1.1187803
1.1837955	1.1564273	1.0149130	0.6021297	1.1239651	0.9887743	0.9688409	0.7058495	0.9043700	1.0835883
1.2717359	0.9794425	1.0775343	0.9892390	0.7245881	0.9170011	0.9211420	0.9881373	1.2200051	1.1526351
0.9670953	0.9493277	1.1393927	1.1113326	0.8622489	0.8585010	1.0729164	1.1537066	0.9775308	1.1762215
1.0796212	0.8775947	1.0682239	0.7741274	1.2866047	1.3960800	0.9265557	0.7911731	1.1139439	0.9729891
1.4803236	0.9921520	1.1379479	1.0056004	0.8513454	1.0377585	0.6390083	1.2931110	1.0306507	1.4345223
1.0951019	0.8580107	1.1221453	0.8131805	0.7492733	1.0582892	0.9113416	1.0002211	1.0148683	0.8820958
0.8862663	0.9729643	1.2356174	0.6952866	1.1187892	1.0665901	1.2126200	0.9391632	1.0740038	1.0534198
0.8914960	1.2415736	1.2320805	1.1400427	1.3173667	1.1116973	0.7446816	0.8853469	0.7550775	0.9053199

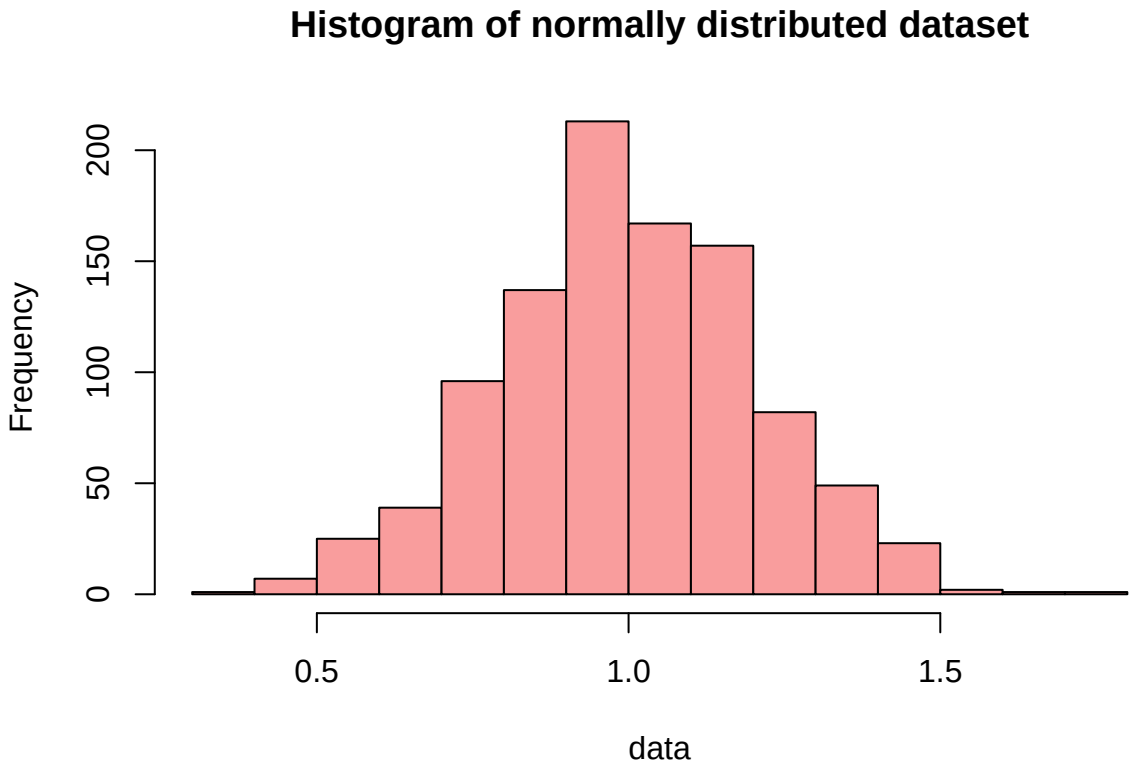
Just staring at the dataset for a while tells us very little. To see something useful, we can plot it as a histogram:

Now this can be done in two ways: 1. Using the **base R** function `hist(data, ...)` 2. Using **ggplot2** to plot the histogram

Base R `hist(data)`

To see all the available arguments that you can pass to the `hist()` function simply write `?hist` in the `R` console

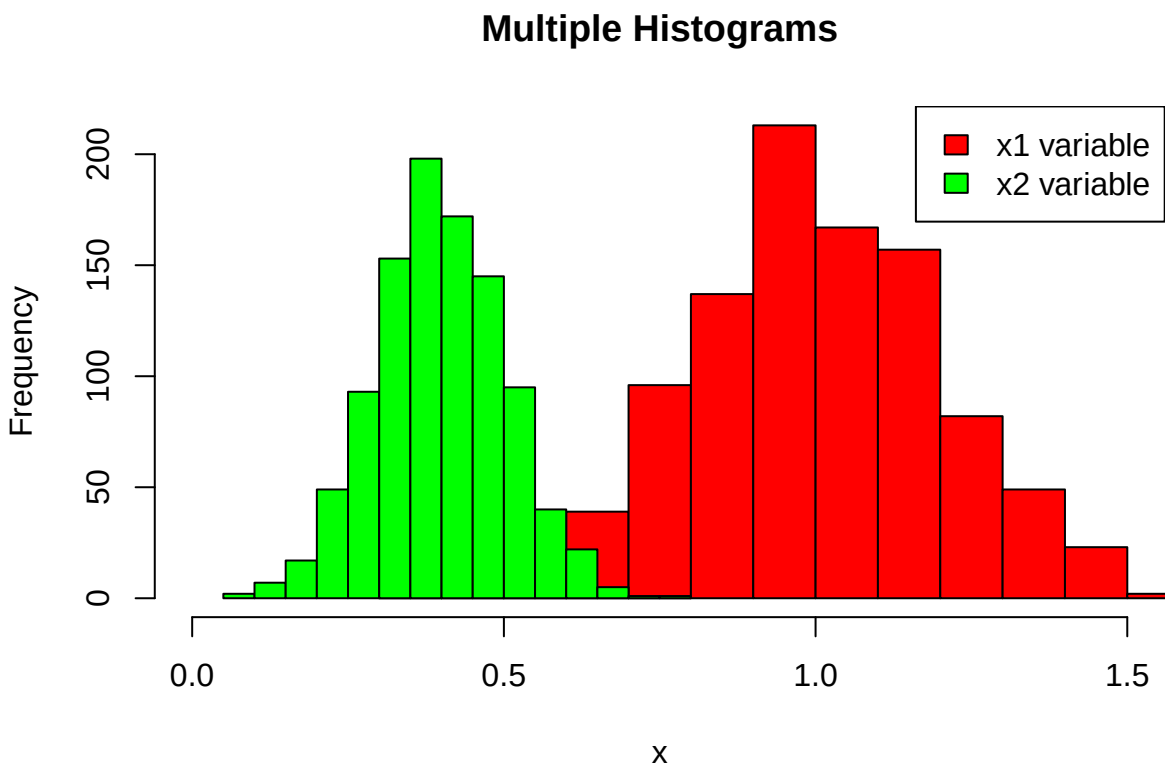
```
set.seed(1)
data <- rnorm(1000, mean = 1, sd = .2)
hist(data, col = "#f99d9d",
  main = "Histogram of normally distributed dataset")
```



Plotting multiple histograms on one plot with `hist(data)`

To plot multiple histograms in with `Base R` one plot one can do as follows:

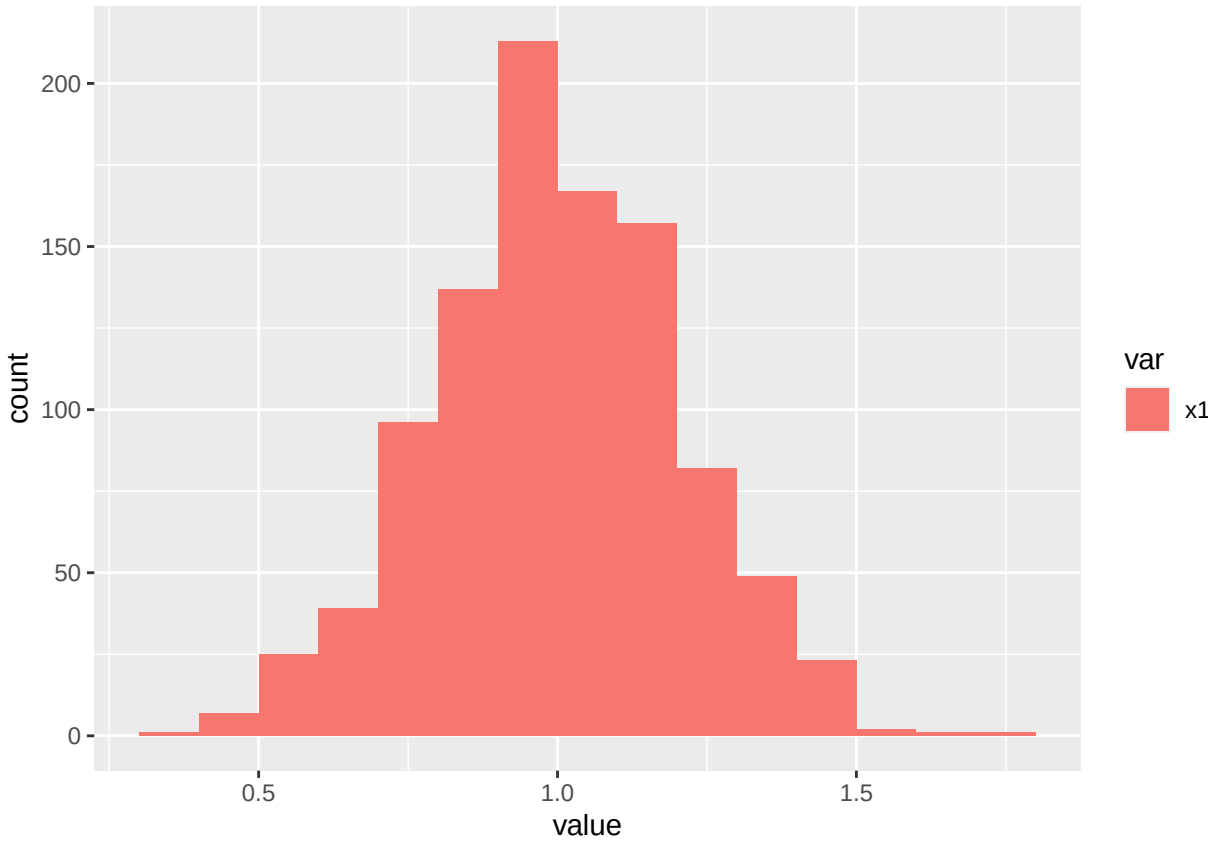
```
#make this example reproducible
set.seed(1)

#define data
x1 <- rnorm(1000, mean = 1, sd = 0.2)
x2 <- rnorm(1000, mean = 0.4, sd = 0.1)

#plot two histograms in same graph
hist(x1, col = "red", xlim = c(0, 1.5),
    main = "Multiple Histograms", xlab = "x")
hist(x2, col = "green", add = TRUE)

#add legend
legend("topright", c("x1 variable", "x2 variable"), fill = c("red", "green"))
```



ggplot2 `ggplot(data)`

Now to use `ggplot` to plot your data you have to first make it into a data frame:

```
set.seed(1)
df <- data.frame(var = rep("x1", 1000),
          value = c(rnorm(1000, mean = 1, sd = .2)))
brx <- pretty(range(df$value),
    n = nclass.Sturges(df$value), min.n = 1)
    # the brx function is only used to make it look like the base R
    # hist function which uses 'Sturges rule' to determine binsize i.e, 'breaks'
    # the default binsize for geom_histogram is 30.
ggplot(df, aes(x = value, fill = var)) + geom_histogram(breaks = brx)
```

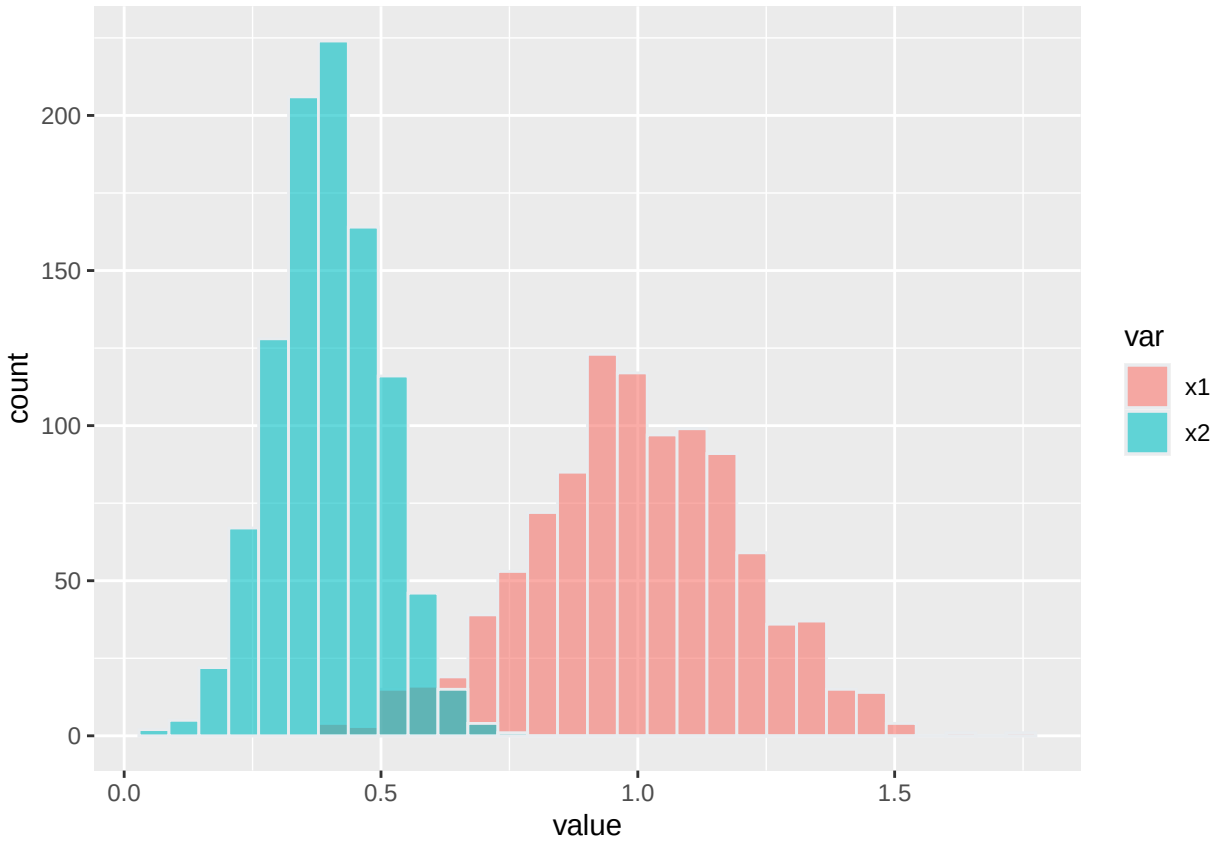


Plotting multiple histograms on one plot with `ggplot(data)`

To plot multiple histograms in with `ggplot2` one plot one can do as follows:

```
#make this example reproducible
set.seed(1)

#create data frame
df <- data.frame(var = c(rep("x1", 1000), rep("x2", 1000)),
                 value = c(rnorm(1000, mean = 1, sd = .2),
                           rnorm(1000, mean = 0.4, sd = 0.1)))

#plot multiple histograms
ggplot(df, aes(x = value, fill = var)) +
  geom_histogram(color = "#e9ecef", alpha = 0.6, position = "identity")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


count
200
150
100
50
0
0.0
0.5
1.0
1.5
value
var
x1
x2

Kernel density estimates

We can graphically represent data in a more variegated plot by a so-called kernel density estimate.

The idea behind the construction of the plot is to "put a pile of sand" around each element of the dataset. At places where the elements accumulate, the sand will pile up. The actual plot is constructed by choosing a *kernel* K and a *bandwidth* h.

The *kernel* K reflects the shape of the piles of sand, whereas the bandwidth is a tuning parameter that determines how wide the piles of sand will be.

Formally, a kernel K is a function $K : \mathbb{R} \to \mathbb{R}$.

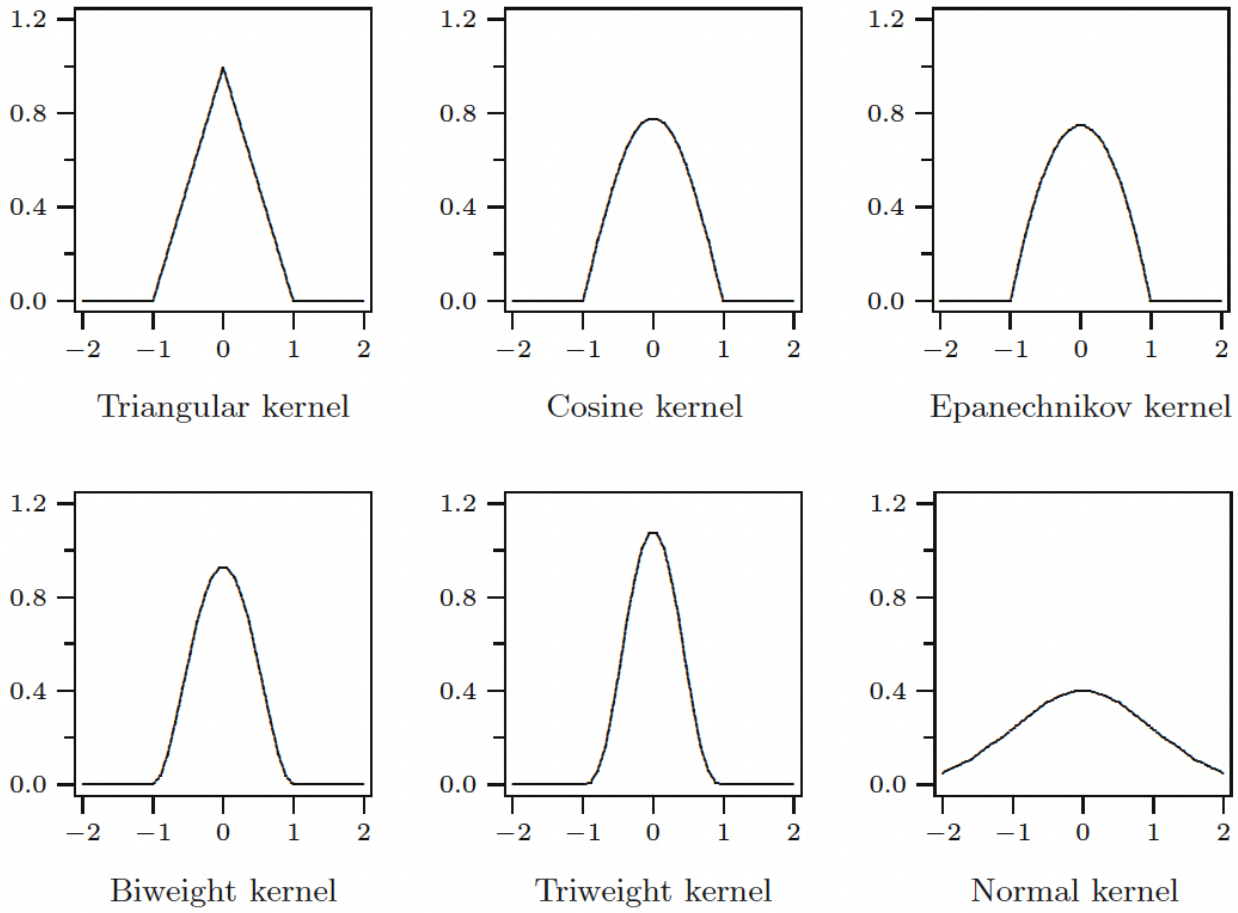


Figure 1: Examples of well-known kernels K

To plot a kernel density estimate you can use either the `densityplot` function from the library `lattice` or you can use the **base R** function `plot` with the parameter `density()`.

For simplicity I will choose to use the **base R** function and using the diamonds dataset from the `UsingR`.

```
library(UsingR)
```

```
kernels <- c("triangular", "cosine", "epanechnikov",
    "biweight", "rectangular", "gaussian")
colors <- brewer.pal(7, "Dark2")
par(mfrow = c(2, 3))
for (kernel in seq_len(length(kernels))) {
```

```
    title <- stringr::str_to_title(kernels[kernel])
    plot(density(diamond$price, bw = 10000, kernel = kernels[kernel]),
    main = paste(title, "kernel"), col = colors[kernel], lwd = 2)
}
```

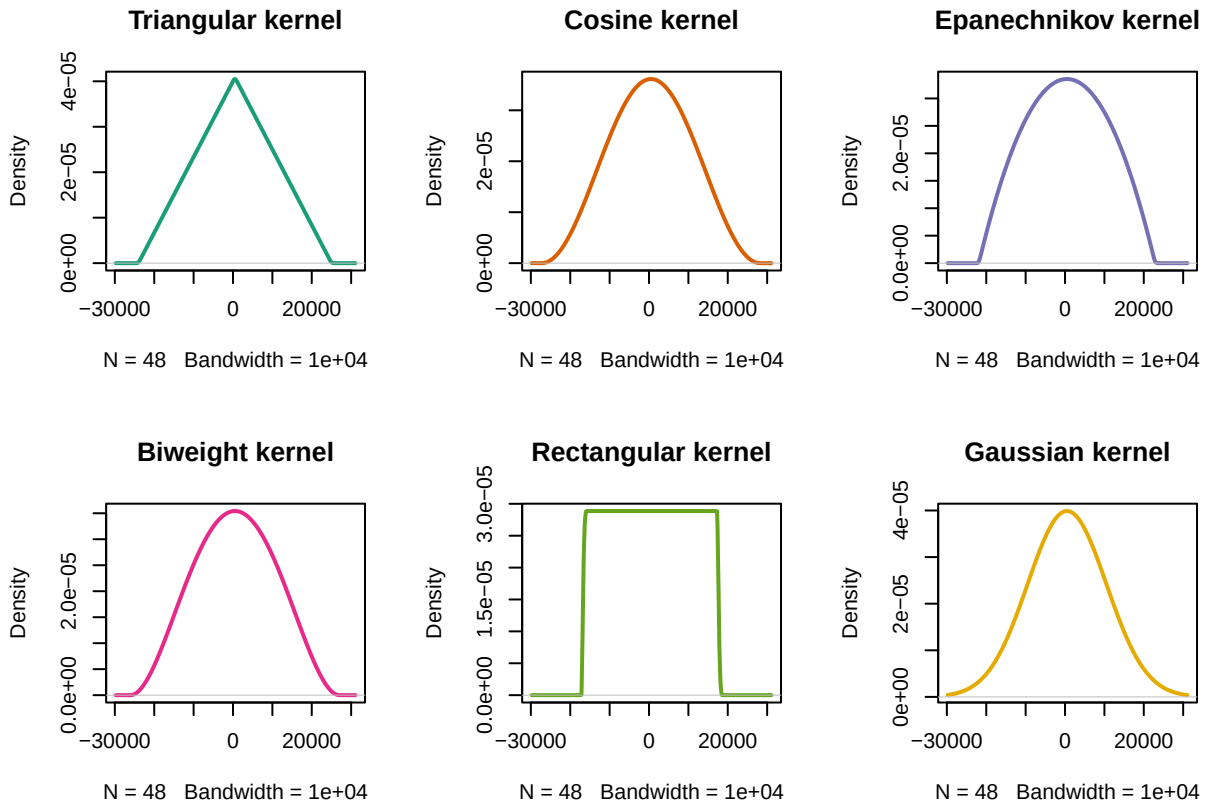


Figure 2: Examples of well-known kernels K in R

Note: R also has the kernel 'optcosine' which is apparently less smooth than 'cosine' see R documentation. It looks almost the same as 'cosine' but with a bit sharper 'hinge' at the beginning of the curve (almost like the 'triangular' kernel). Did not feel it was significant enough to include, but try it for yourself.

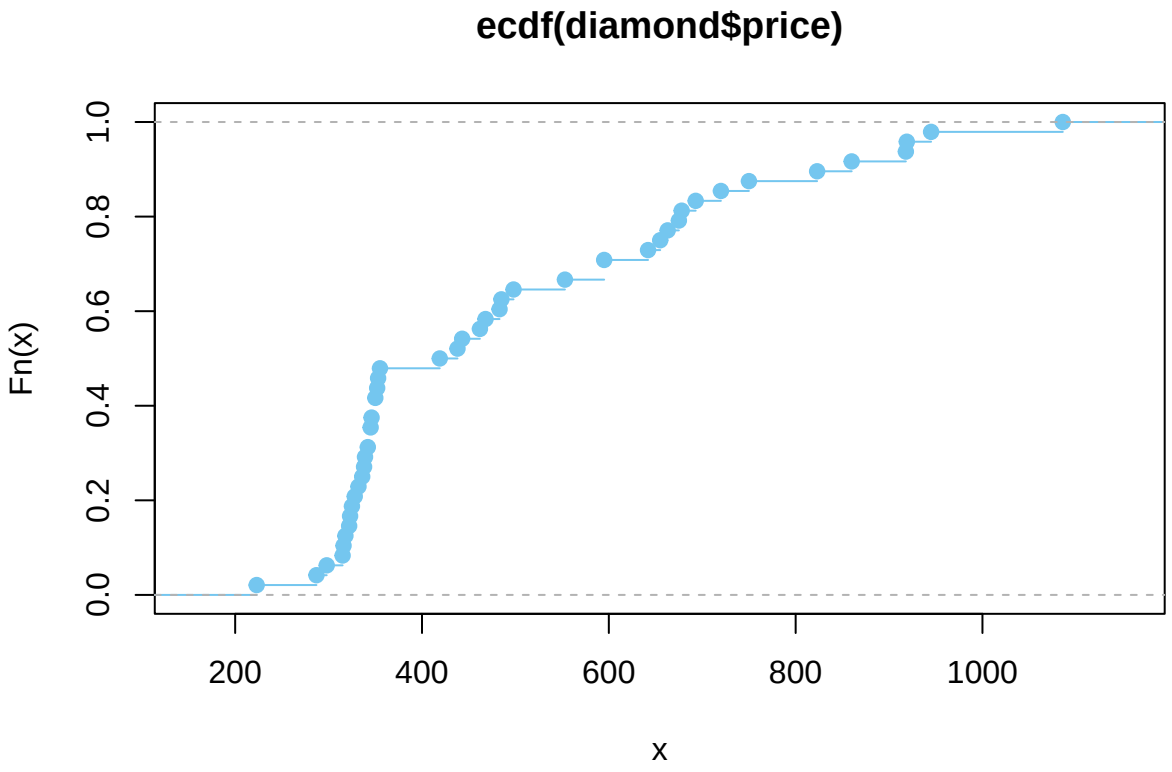
The `bw` argument specifes the bandwith that wil be used to create the kernel density estimate. The `kernel` argument specifies the kernel that will be used

Empirical distribution function

Another way to graphically represent a dataset is to plot the data in a cumulative manner. This can be done using the empirical cumulative distribution function of the data.

In R this can be done like this:

```
plot(ecdf(diamond$price), col = "#74c6ef")
```



Scatterplot

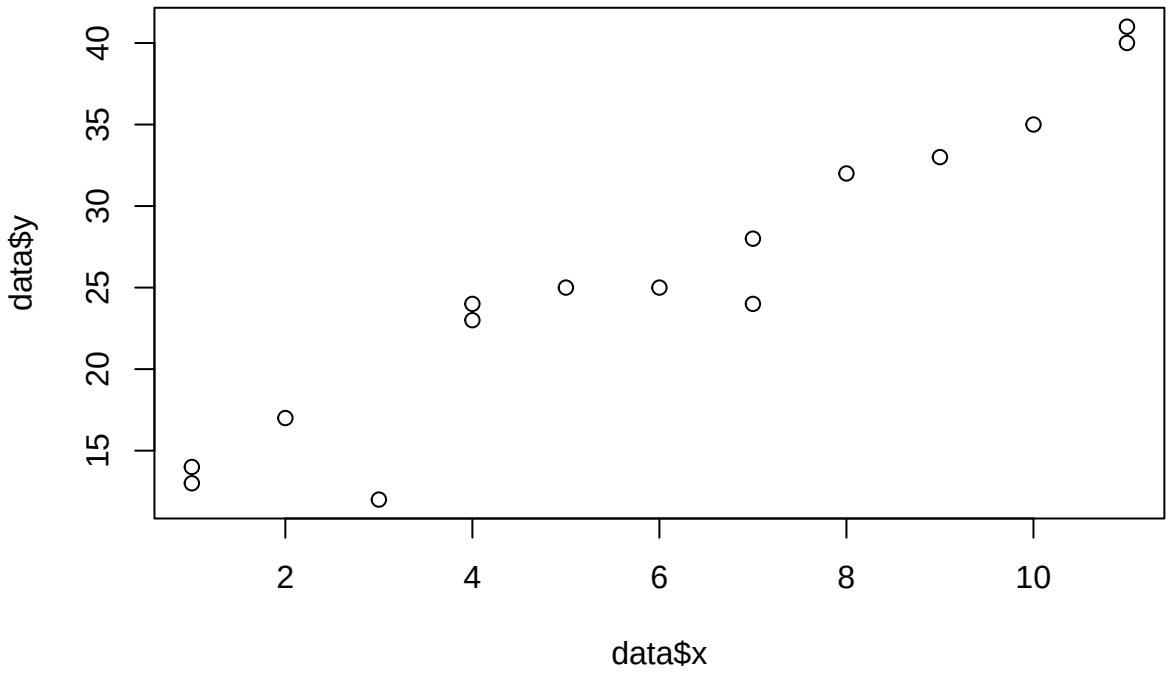
In some situations one wants to investigate the relationship between two or more variables. In the case of two variables x and y, the dataset consists of *pairs of observations*:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

We call such a dataset a bivariate dataset in contrast to the univariate dataset, which consists of observations of one particular quantity. We often like to investigate whether the value of variable y depends on the value of the variable x, and if so, whether we can describe the relation between the two variables. A first step is to take a look at the data, i.e., to plot the points (x_i, y_i) for $i = 1, 2 \ldots, n$. Such a plot is called a scatterplot.

In `R` this can be acheived by simply using the `plot()` function:

```
#create some fake data
data <- data.frame(x = c(1, 1, 2, 3, 4, 4, 5, 6, 7, 7, 8, 9, 10, 11, 11),
                   y = c(13, 14, 17, 12, 23, 24, 25, 25, 24, 28, 32, 33, 35, 40, 41))

#create scatterplot of data
plot(data$x, data$y)
```


data$y
15
20
25
30
35
40
2
4
6
8
10
data$x