Datenanalyse

Abgabe 1

Lukas Mahler (Matr. Nr. 11908553)

01.04.2021

Setup

Loading the library and the data from the loaded library:

```
set.seed(69)
library("StatDA")
```

```
## Warning: package 'StatDA' was built under R version 4.0.4
```

```
## Loading required package: sgeostat
```

```
## Warning: package 'sgeostat' was built under R version 4.0.3
```

```
## Registered S3 method overwritten by 'geoR':
##   method         from
##   plot.variogram sgeostat
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
data(chorizon)
```

Examples:

1st Example:

Histogram:

Histograms: a histogram is a graphical representation of numerical data. The data is divided into several bins that get drawn as rectangles according to the amount of data values contained in the bin. The interval length can get calculated by several distinct formulas, e.g. Sturges or Freedman and Diaconis. Histograms are used to get a broad overview of the given data to further analyze it with other tools.

Histogram for the given data (interval length from Sturges):

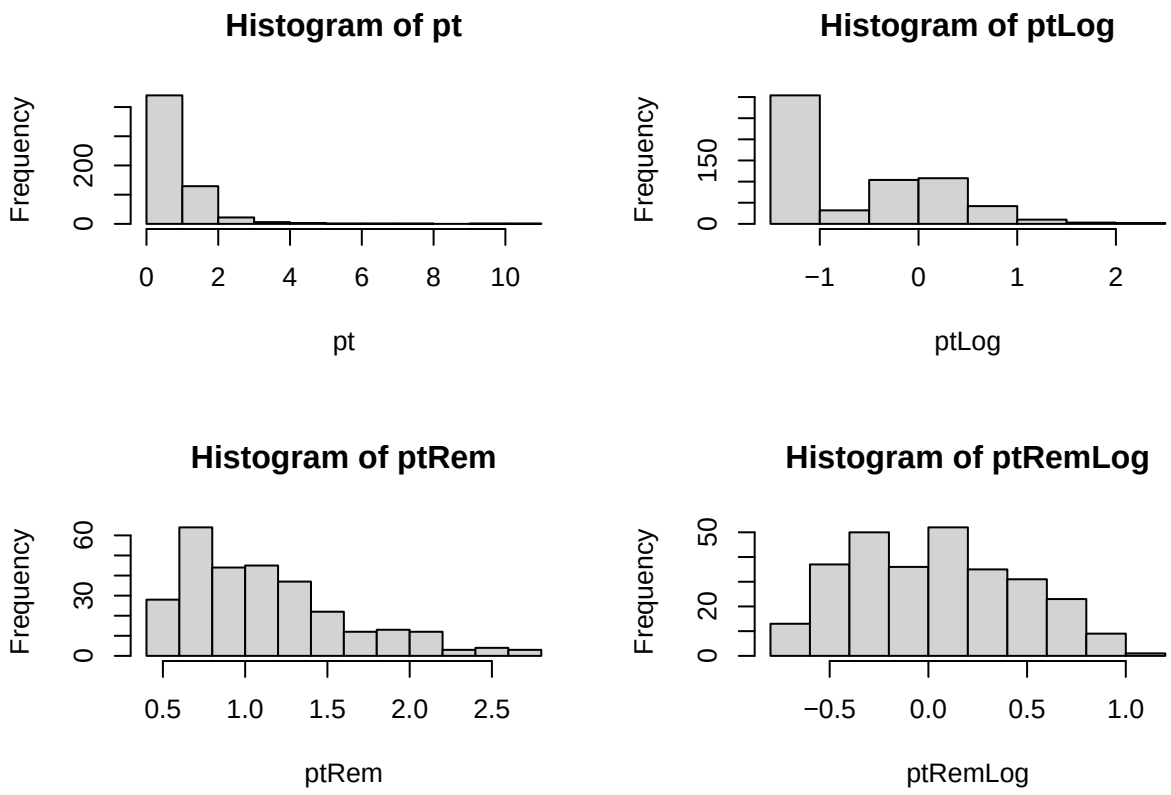
```
pt <- chorizon$Pt
length(pt) # amount of values
```

```
## [1] 605
```

```
length(pt[pt %in% 0.25]) # amount of values = 0.25
```

```
## [1] 304
```

```
ptRem <- pt[!pt %in% 0.25] # Removes the 0.25 values
ptRem <- ptRem[ptRem < 3] # Removes all values above 5
ptLog <- log(pt) # transforms to log()
ptRemLog <- log(ptRem) # transforms to log()
par(mfrow = c (2,2))
hist(pt)
hist(ptLog)
hist(ptRem)
hist(ptRemLog)
```

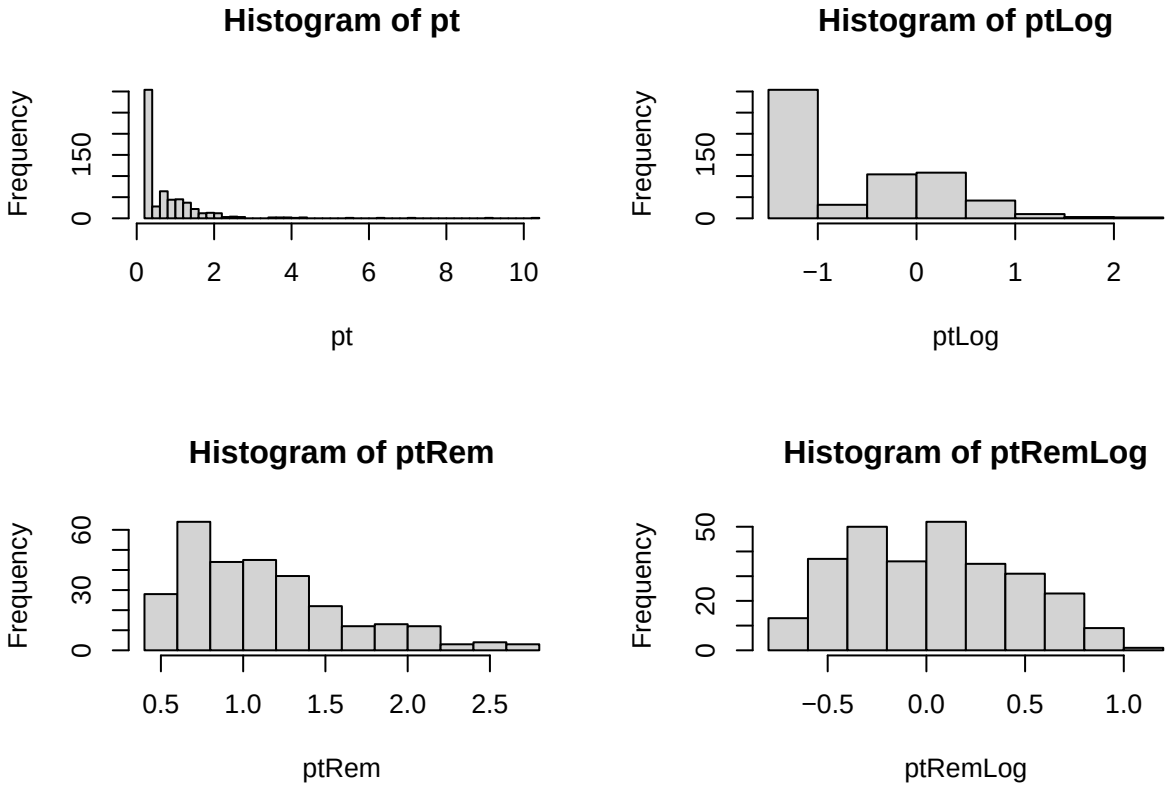


Here one cannot see how the values are distributed because the outliers increase the interval length rapidly. One can only see that most of the values lie between 0 and 2. Outliers range up to over 10.

To better accomodate a normal distribution we can do a log-transformation which gives the dataset more of a normal distribution. The Problems are the heavy outliers on the right side and the many duplicates of the value 0.25. Those get stripped in the variable ptRem. Depending on if the values of 0.25 are legit the transformation to a normalized variable could be helpful. On the other side, the amount of values = 0.25 make it impossible to get a normal distribution.

Histogram for the given data (interval length from Freedman and Diaconis):

```
par(mfrow = c (2,2))
hist(pt, breaks="FD")
hist(ptLog, breaks="FD")
hist(ptRem, breaks="FD")
hist(ptRemLog, breaks="FD")
```

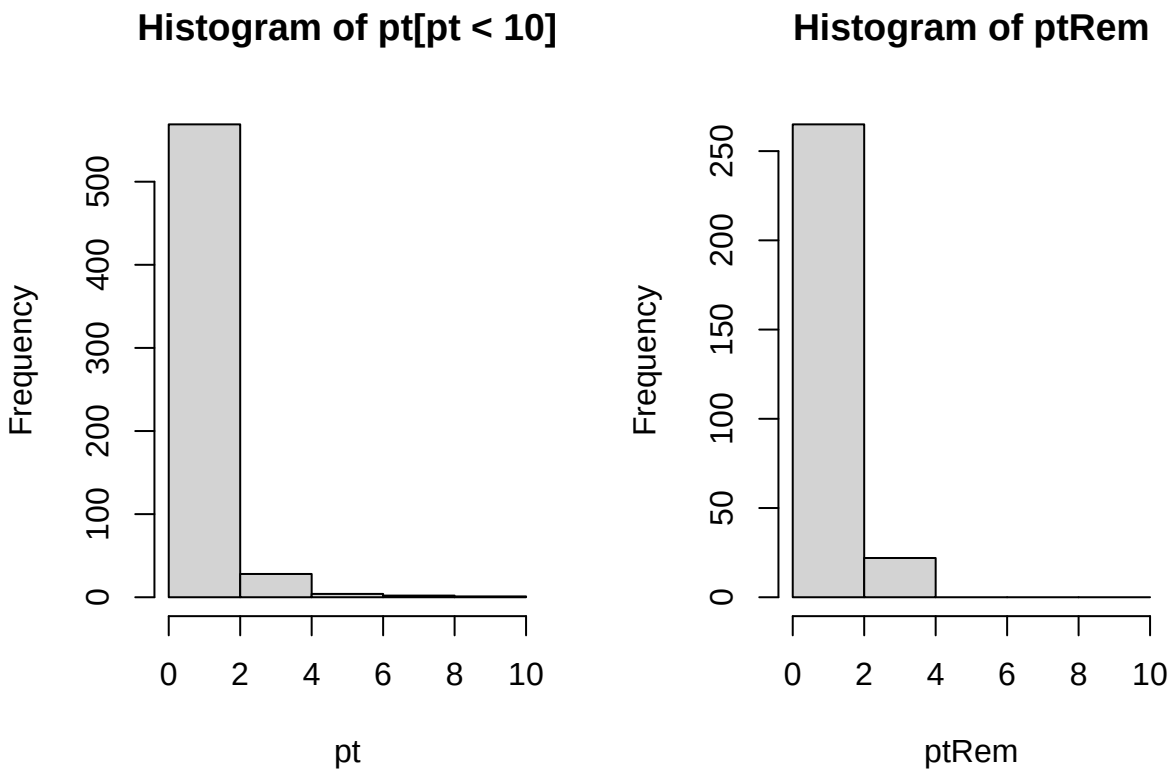


Here we can see the distribution more in depth. The interval length is small enough to see that most of the values not only lie between 0 and 2 but rather that over 300 of the values lie directly in the first interval (almost all having the value of 0.25 as we can see in the of "length(pt[pt %in% 0.25]".

The histogram of ptRemLog lets us see how this data could be transformed almost into a normal distribution (as we will be able to see with qpplot later) but at the cost of deleting several values (of 0.25 and above 3). As it is unlikely that the set has so many errors (more than half the data would be wrong) we will continue using the pt data (the original data without a log transformation).

Histogram for the given data (equidistant interval of 5 classes à length = 2)

```
br <- seq(0,10,2)
par(mfrow = c (1,2))
hist(pt[pt < 10], breaks=br, xlab="pt")
hist(ptRem, breaks=br)
```



The histogram with the Friedman-Diaconis method works best for my data because the outliers increase the width of the intervals which reduces the detail to see any possible distributions. The fixed bin width of 2 is not ideal because important details get lost. Even more details get lost than in the standard interval method from Sturges.

When transforming to log, a potential normal distribution of the values from -0.5 to 0.7 can be spotted but is rather unlikely for the whole set because approximately half of the values is 0.25.

Density approximation:

```
gaussian <- density(pt, kernel= "gaussian")
cosine <- density(pt, kernel="cosine")

gaussian
```

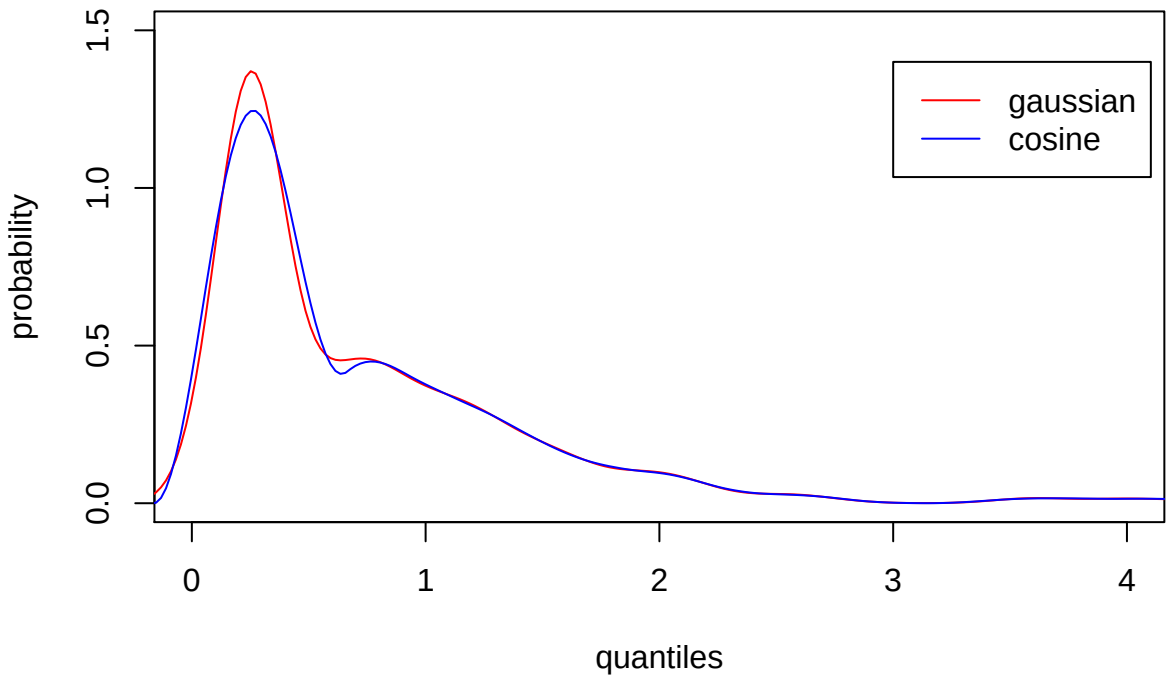
```
##
## Call:
##  density.default(x = pt, kernel = "gaussian")
##
## Data: pt (605 obs.); Bandwidth 'bw' = 0.1486
##
##        x                y
##  Min.   :-0.1958   Min.   :0.0000000
##  1st Qu.: 2.5271   1st Qu.:0.0003976
##  Median : 5.2500   Median :0.0033811
##  Mean   : 5.2500   Mean   :0.0916735
##  3rd Qu.: 7.9729   3rd Qu.:0.0281704
```

```
##  Max.   :10.6958   Max.   :1.3703846
```

```
cosine
```

```
##
## Call:
##  density.default(x = pt, kernel = "cosine")
##
## Data: pt (605 obs.); Bandwidth 'bw' = 0.1486
##
##        x                 y
##  Min.   :-0.1958   Min.   :0.0000000
##  1st Qu.: 2.5271   1st Qu.:0.0002764
##  Median : 5.2500   Median :0.0033182
##  Mean   : 5.2500   Mean   :0.0917264
##  3rd Qu.: 7.9729   3rd Qu.:0.0251358
##  Max.   :10.6958   Max.   :1.2442632
```

```
par(mfrow= c(1,1))

plot(1,1, type="n", xlim=c(0,4), ylim=c(0,1.5), xlab="quantiles", ylab="probability")
lines(gaussian, col="red")
lines(cosine, col="blue")
legend(3,1.4, col = c("red", "blue"), legend=c("gaussian","cosine"),lty=c(1,1))
```

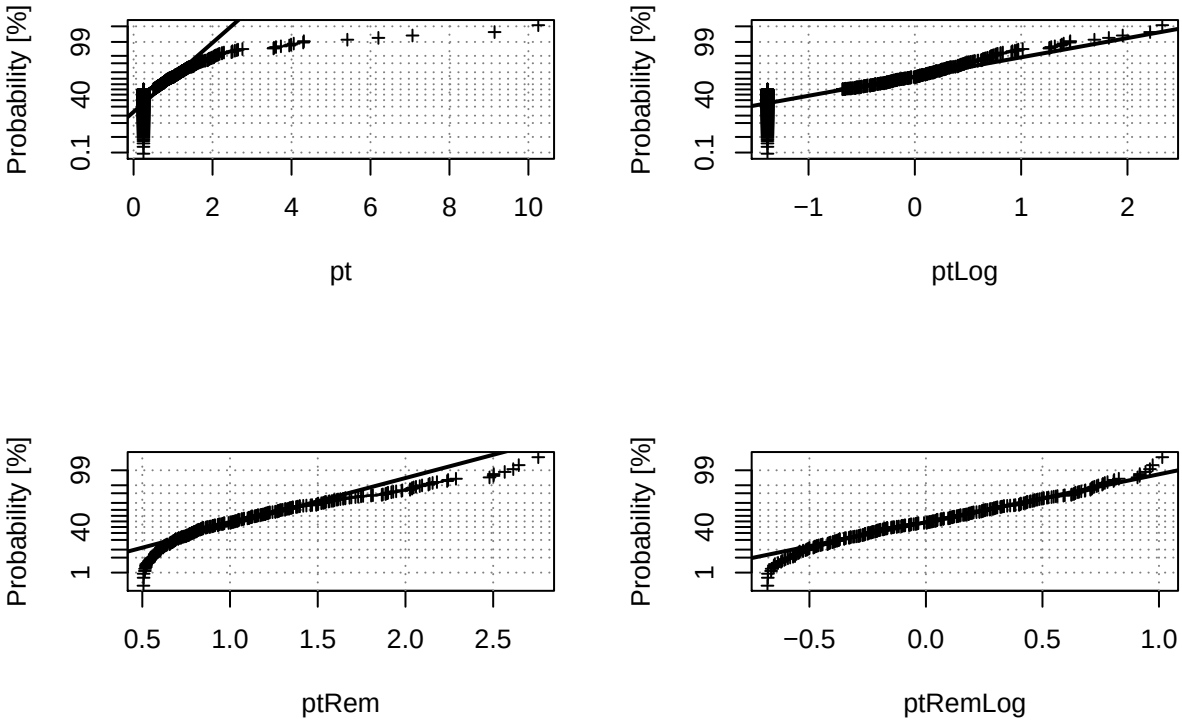


The cosine kernel would be more appropriate because there we can see the 'dip' between the many 0.25 values and the rest of the distribution that follows more of a normal distribution if transformed with log. On the gaussian kernel this fact is not included and would generate more generic datasets where this bump would be

not as strong as in the cosine kernel. On the other hand the gaussian kernel represents the extreme peak at 0.25 way more accurate than the cosine. Depending on the usecase both kernels have their own strengths for this dataset.

Below there are some extra qpplots to show the possibility of a normal distribution in parts of the distribution after a log transformation.

```
par(mfrow= c(2,2))
qpplot.das(pt, xlab = "pt")
qpplot.das(ptLog, xlab = "ptLog")
qpplot.das(ptRem, xlab = "ptRem")
qpplot.das(ptRemLog, xlab = "ptRemLog")
```



2nd Example:

```
library("Ecdat")
```

```
## Warning: package 'Ecdat' was built under R version 4.0.5
```

```
## Loading required package: Ecfun
```

```
## Warning: package 'Ecfun' was built under R version 4.0.5
```

```
##
## Attaching package: 'Ecfun'
```

```
## The following object is masked from 'package:base':
##
##     sign
```

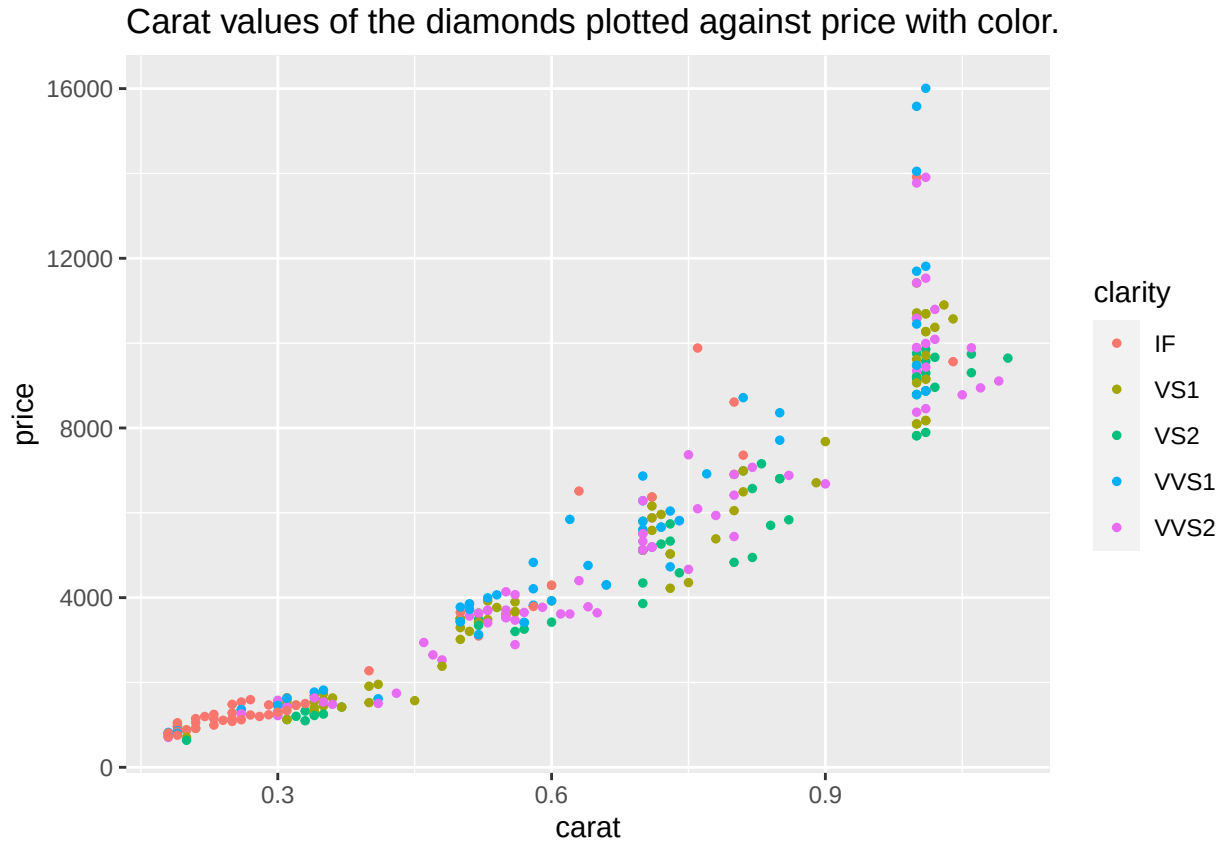
```
##
## Attaching package: 'Ecdat'
```

```
## The following object is masked from 'package:datasets':
##
##     Orange
```

```
data(Diamond)

clarity <- Diamond$clarity
carat <- Diamond$carat
price <- Diamond$price
```

Plotting of price and carat grouped by clarity shows that price and carat correlate. To eliminate this interference, price must be transformed to be independent from carat (weight). Especially the clearer the diamonds get the smaller they are. Since this is the case the clearest diamonds (IF) seem to be the cheapest (because there are simply almost no large diamonds in IF).

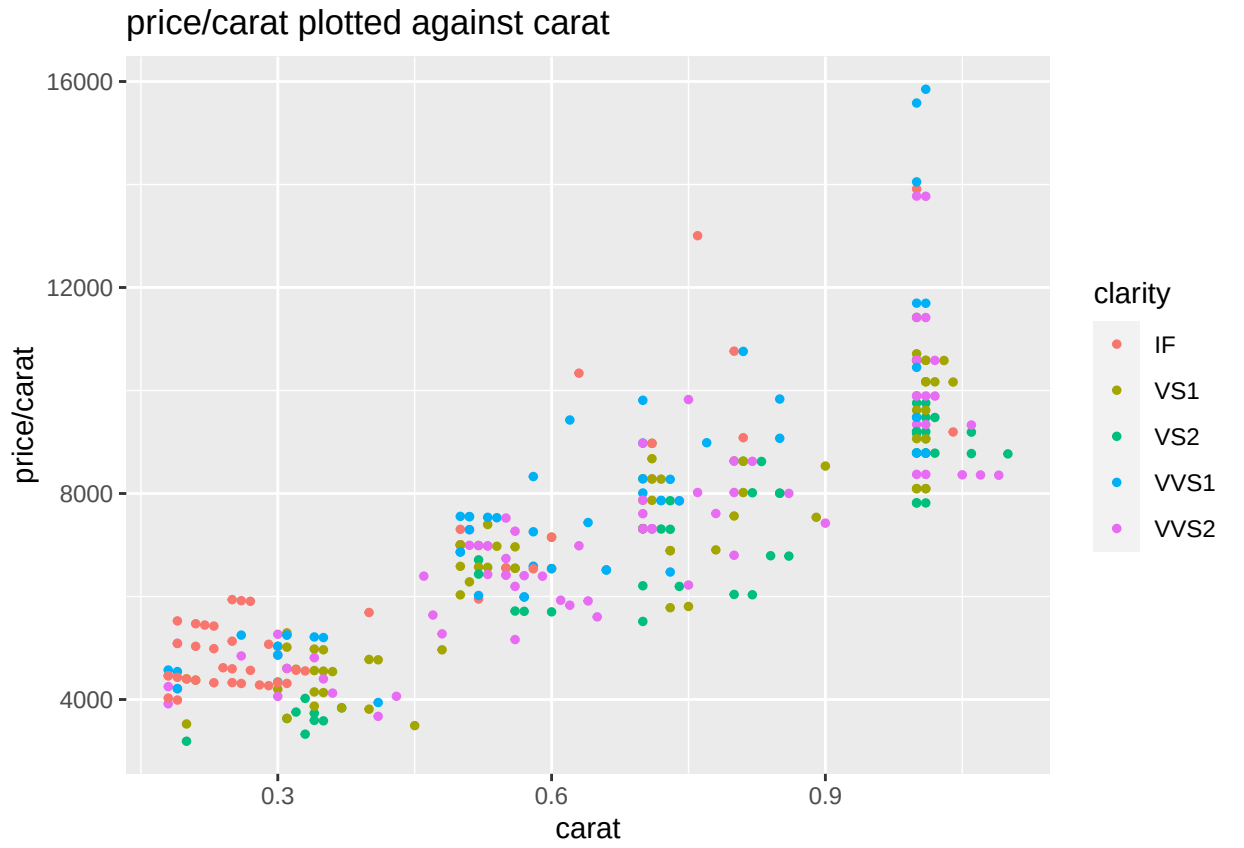
```
# overview of price carat and clarity
par(mfrow = c(1,1))

ggplot(data=Diamond, aes(carat, price,colour=clarity)) +
  geom_point( size=1) +
  ggtitle("Carat values of the diamonds plotted against price with color.")
```



As we can see in the graph above, the price seems to increase quadratically compared to the amount of carats. Now 2 steps need to be done to make price independent of carat: we need to use price/carat as unit since the price goes up with every carat. Secondly we need need to remove the increase of price/carat. This comes

from the lower supply for bigger diamonds. Since very clear diamonds are very rare for bigger sizes, they are way less common for higher carat sizes. To handle this problem we can divide another time by carat and get price/carat per carat which removes the influence of bigger diamonds being worth more per carat.

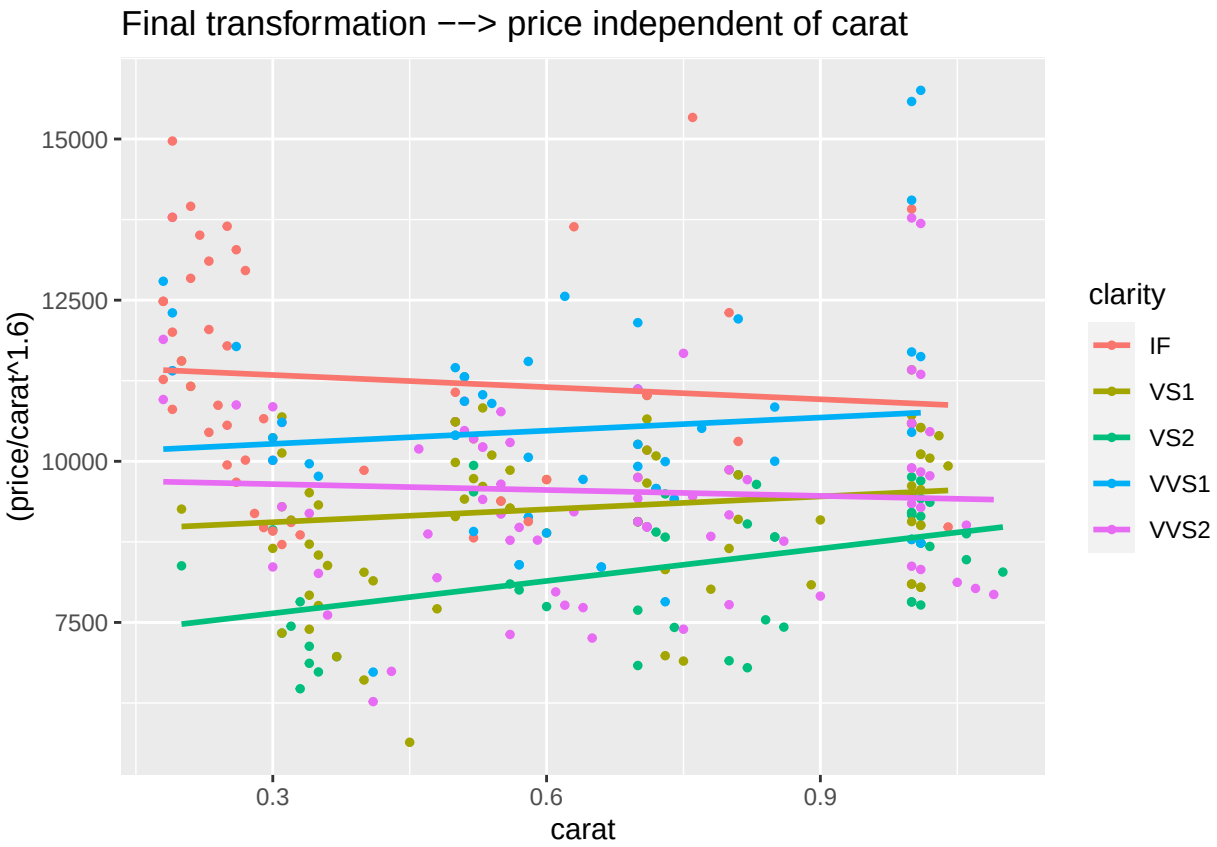
```
ggplot(data=Diamond, aes(carat, price/carat,colour=clarity)) +
  geom_point(size=1) +
  ggtitle("price/carat plotted against carat")
```



Once we use price/carat we see that there is still an increase for bigger diamonds because they are way more rare than the cheaper ones. To remove this influence, we have to divide even more by carat. By doing this, we get the price/carat per caratˆx which removes the increasing value/carat for bigger diamonds. X is in this case at about 0.6, as one can see in the graph below. Below we can see the result of removing the quadratic influence of the weight: The price is now more or less constant over the whole length of carat (as can be seen by the linear regressions).

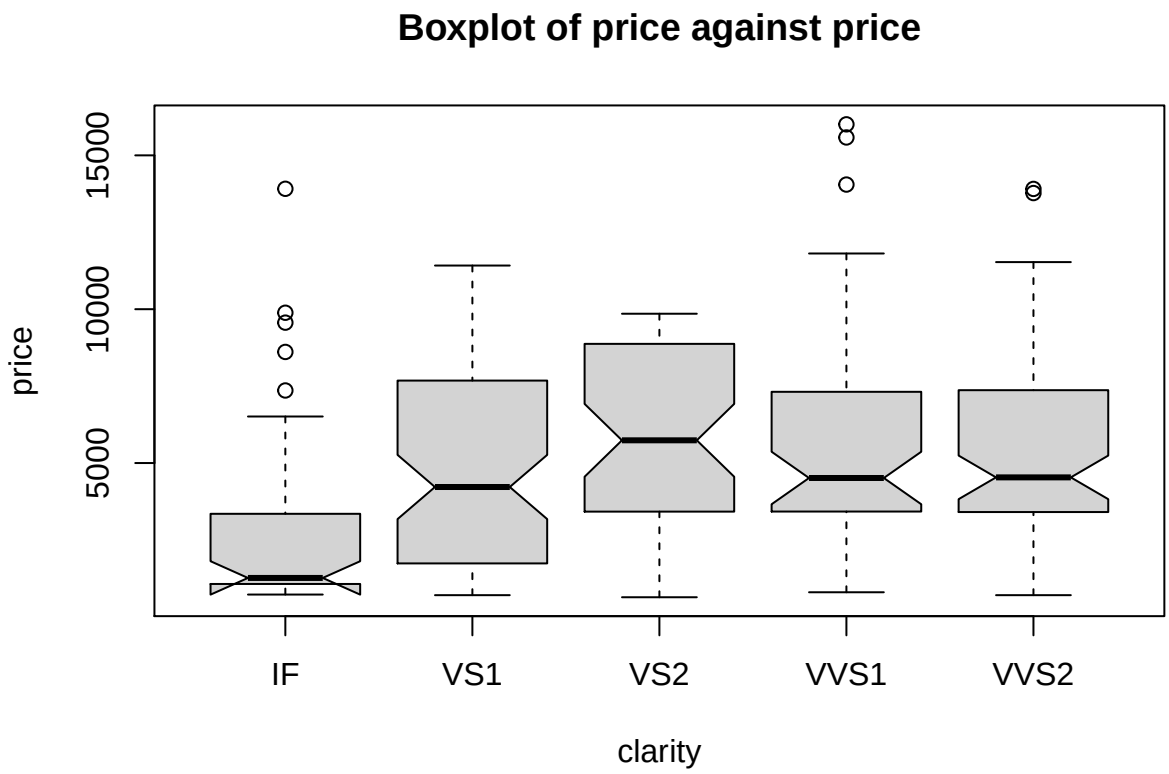
```
ggplot(data=Diamond, aes(carat,(price/caratˆ1.6), colour=clarity)) +
  geom_point( size=1) + geom_smooth(method='lm', se=FALSE) +
  ggtitle("Final transformation --> price independent of carat")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
boxplot(price ~ clarity,data = Diamond,  notch= TRUE, main="Boxplot of price against price")
```

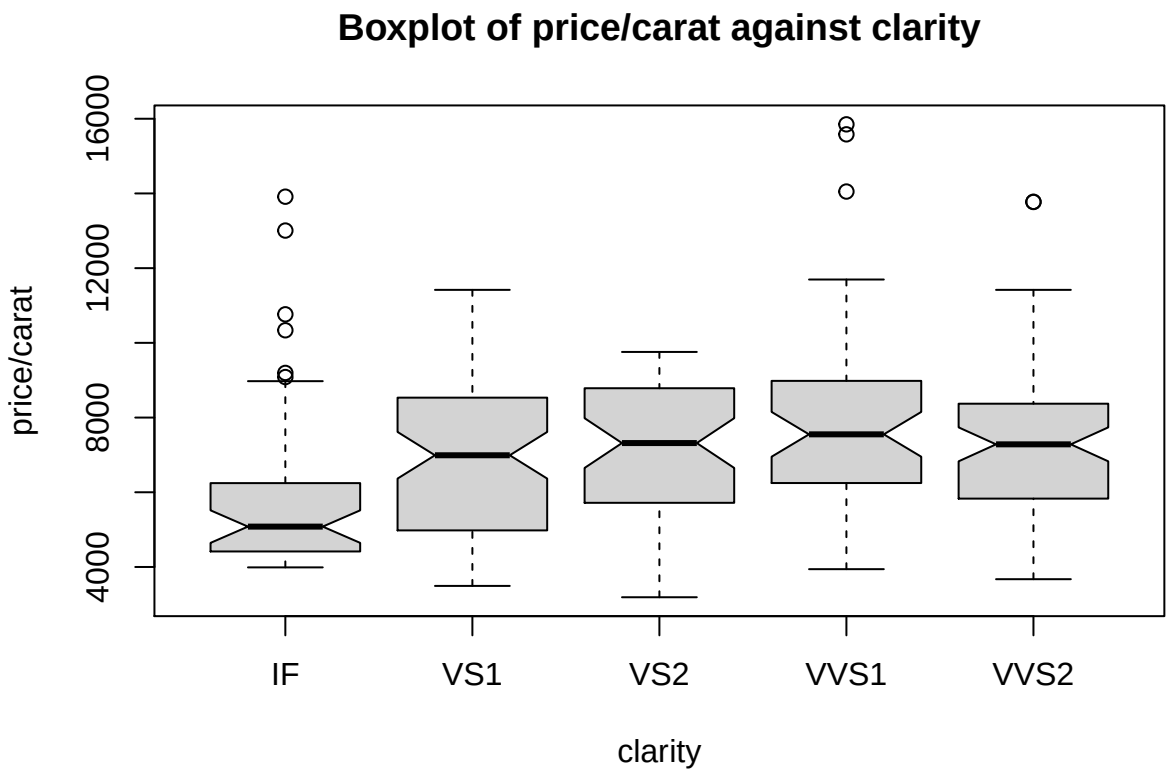
```
## Warning in bxp(list(stats = structure(c(725, 1069.5, 1265.5, 3350, 6512, : some
## notches went outside hinges ('box'): maybe set notch=FALSE
```



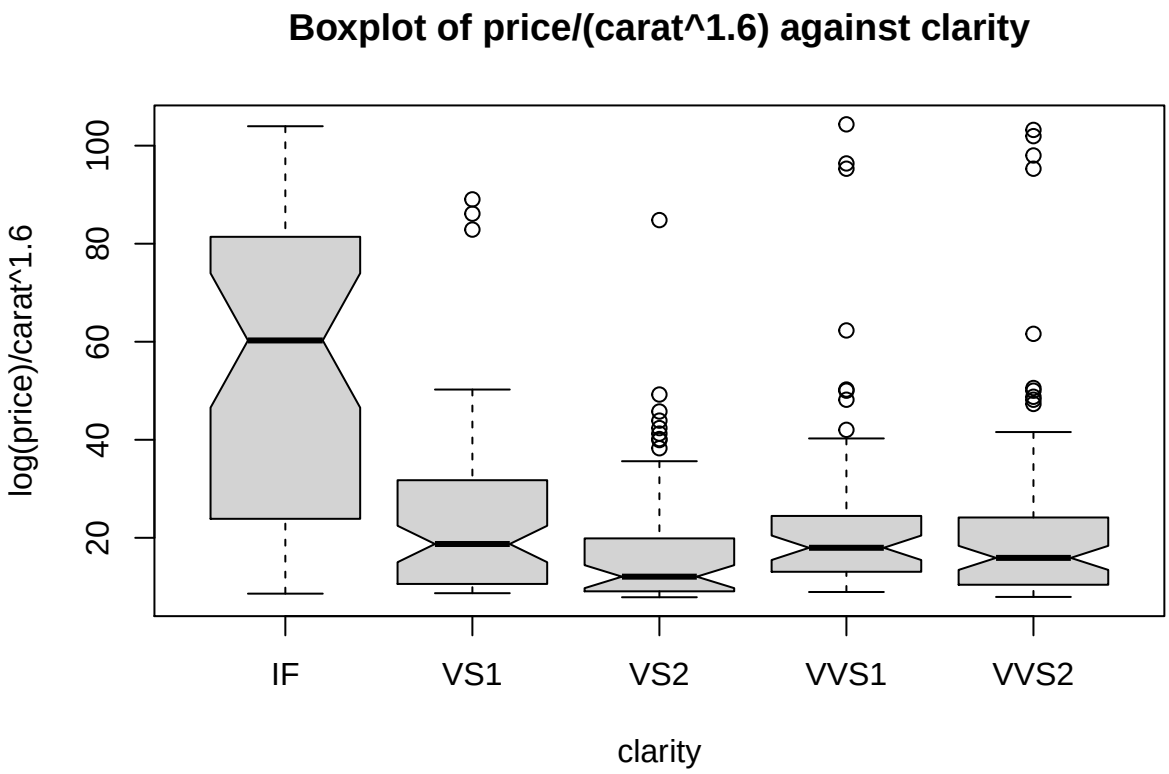
Above we can see the boxplot without any transformation. It seems that diamonds with the clarity (internally flawless) are the cheapest. This comes because IF diamonds with bigger sizes are very rare.

Below we calculate price/carat which still makes us think IF is the cheapest. This comes because bigger diamonds are worth more per carat and IF has no big diamonds. So we divide by carat another time to be independent from the increase of price/carat per carat.

```
boxplot(price/carat~clarity,data = Diamond,  notch= TRUE, main="Boxplot of price/carat against clarity"
```




```
boxplot(log(price)/carat^1.6 ~ clarity, data = Diamond, notch= TRUE, main="Boxplot of price/(carat^1.6)
```



We now clearly see that IF diamonds are significantly more expensive than the other types (as one can see from the notches). The significance interval of IF does not overlap with any other significance interval so it is save to assume that the other types are cheaper than the inertially flawless (IF) ones.

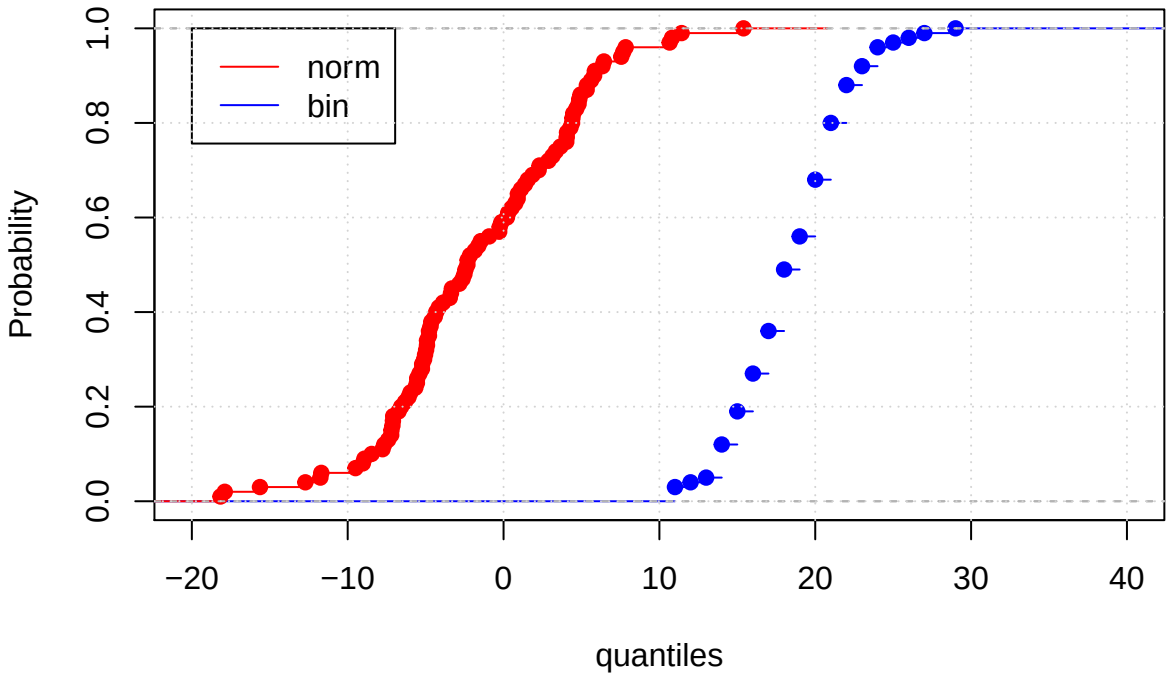
Questions to the boxplot:

The body of the Boxplot contains 50% of the values and is seperated in two quartiles by the median (the value where half of the values is smaller and the other half is bigger) Outside of the bdy on each side are whiskers that show the outer left / right 25% percent of the data. With those 4 quartiles it is possible to estimate the distribution of the values in a very simple way. The notches show the confidence interval for the median (in which interval the median will most likely lie given the sample). If two confidence intervals do not overlap it is very unlikely that both data samples have the same mean.

3rd Example:

Generating the data and drawing the ecdf's of the data sapmples:

```
bin <- rbinom(100, 94, 0.2)
norm <- rnorm(100,-1,7.4)

par(mfrow = c(1,1))
plot(ecdf(bin), col="blue", xlim=c(-20, 40), xlab="quantiles", ylab="Probability", main="Estimated cumul
lines(ecdf(norm), col="red")
legend(-20,1, col = c("red", "blue"), legend=c("norm","bin"),lty=c(1,1))
grid();
```



Difference between both distributions: the binomial distribution moves in steps, it is a discrete distribution (there can be values 20 or 21 but not 20.5 for example). It has a median of about 19 and is symmetric.

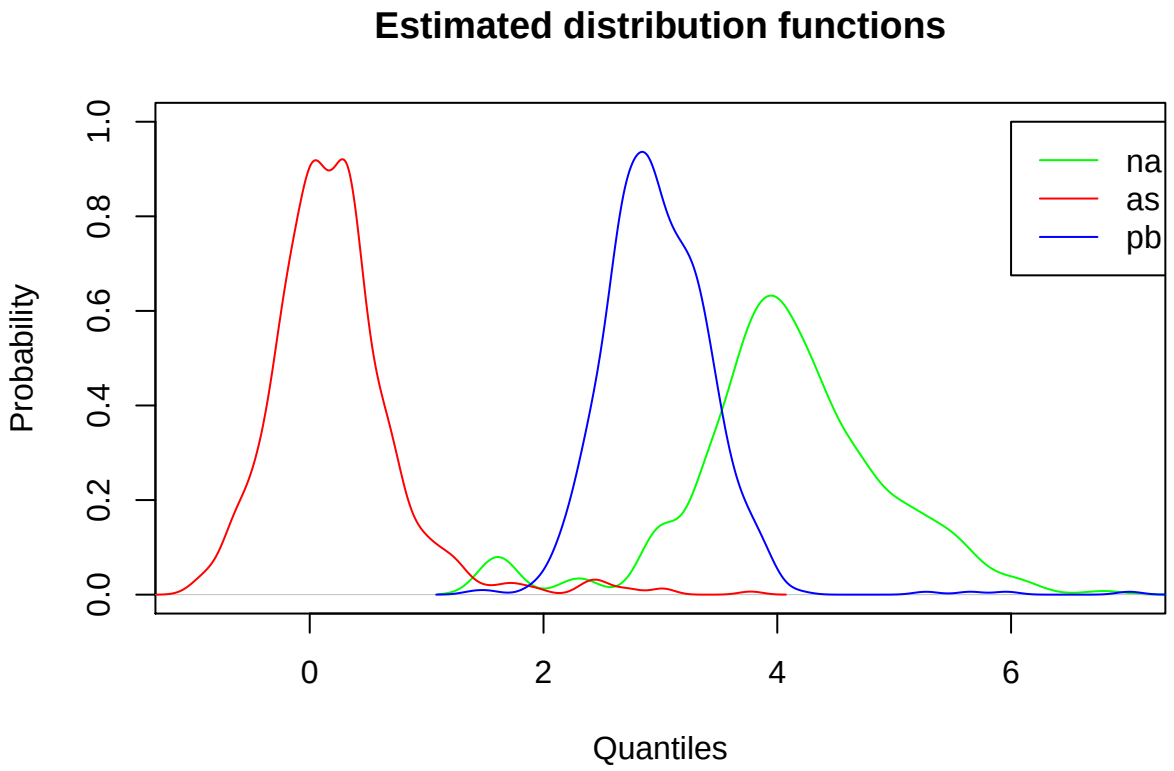
The normal distribution does not move in steps, it is a discrete distribution, it does not go in steps but rather in a continuous flow. It has a median of about -1 and is symmetric.

The empirical distribution function is an estimation of the cumulative distribution function of a given data sample. The values get mapped to their probability according to the schema of the cumulative distribution function. This means that the y-axis of a given point shows the probability that a random value of the distribution is smaller than the given x-value. The y-axis gives the probability (from 0 up to 1) and the x-axis gives the values of the distribution.

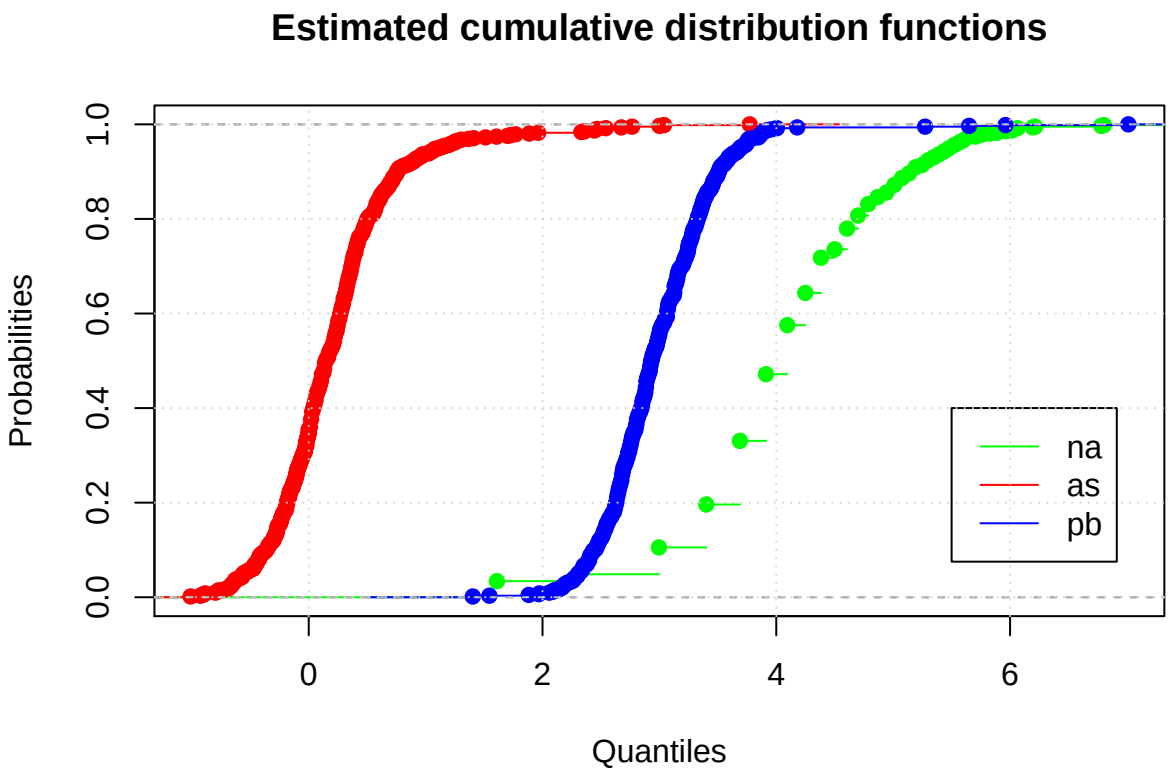
The cdf can be directly generated from the density function and vice versa (via differentiating / integral).

```
data(ohorizon)
na <- ohorizon$Na
as <- ohorizon$As
pb <- ohorizon$Pb

# Log transformations for all variables
na <- log(na)
as <- log(as)
pb <- log(pb)



# density plot
plot(density(na), col="green", xlim=c(-1,7), ylim=c(0,1), main="Estimated distribution functions", xlab=
lines(density(as), col="red")
```

```
lines(density(pb), col="blue")
legend(6,1,legend=c("na","as","pb"), col=c("green","red","blue"), lty=1);
```




```
# cdf plot
plot(ecdf(na), col="green", xlim=c(-1,7), ylim=c(0,1), xlab="Quantiles", ylab="Probabilities", main="Es
lines(ecdf(as), col="red"); lines(ecdf(pb), col="blue")
legend(5.5,0.4,legend=c("na","as","pb"), col=c("green","red","blue"), lty=1)
grid();
```




```
# standard deviations:
sd(na)
```

```
## [1] 0.8660147
```

```
sd(as)
```

```
## [1] 0.5635481
```

```
sd(pb)
```

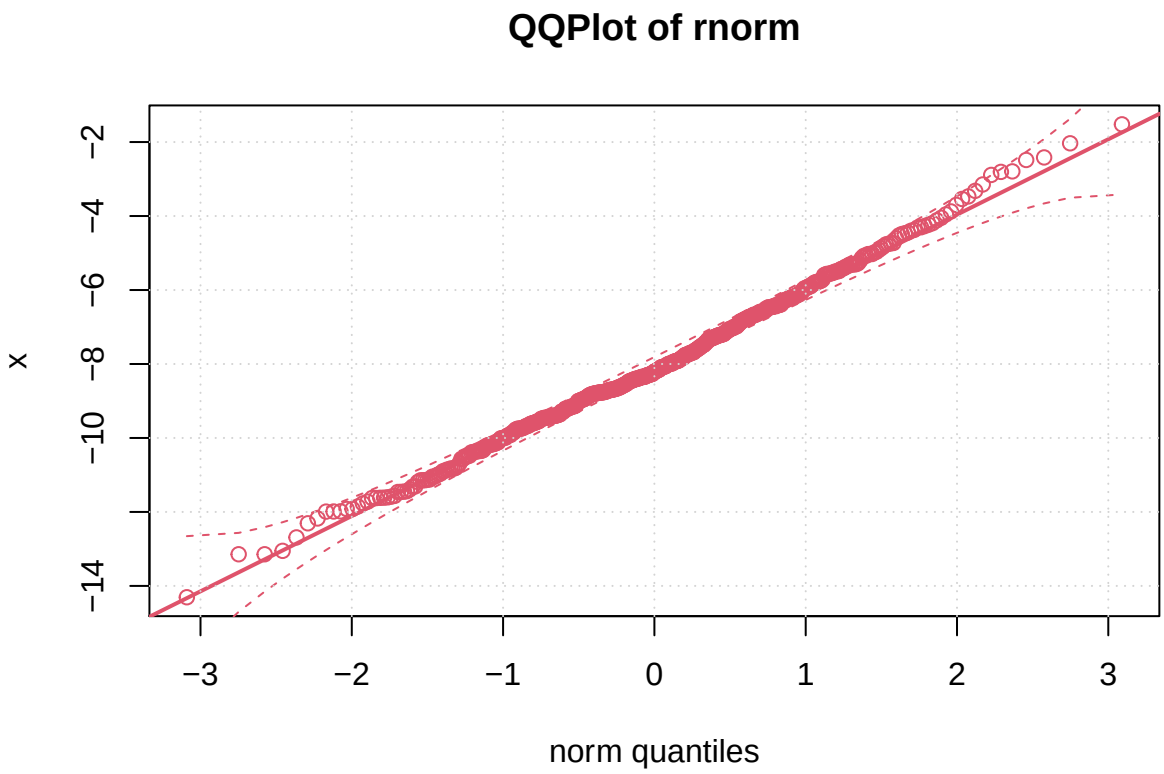
```
## [1] 0.4795077
```

The data got transformed to log since the samples have very high differences in the x-values. With the log transformation the differences are still visible while the readability of the samples is increased. The following description uses the log transformation. As we can see As has a similar distribution as Pb but the mean of As is at about 0.2, the mean of Pb is at about 3. The standard deviation of both as and pb seems to be similar. Na on the other hand has a mean of about 4 and a way higher standard deviation.

All three have outliers in the upper bounds, especially As and Pb. Na has outliers in the lower bounds as well (a heavy tail).

4th Example:

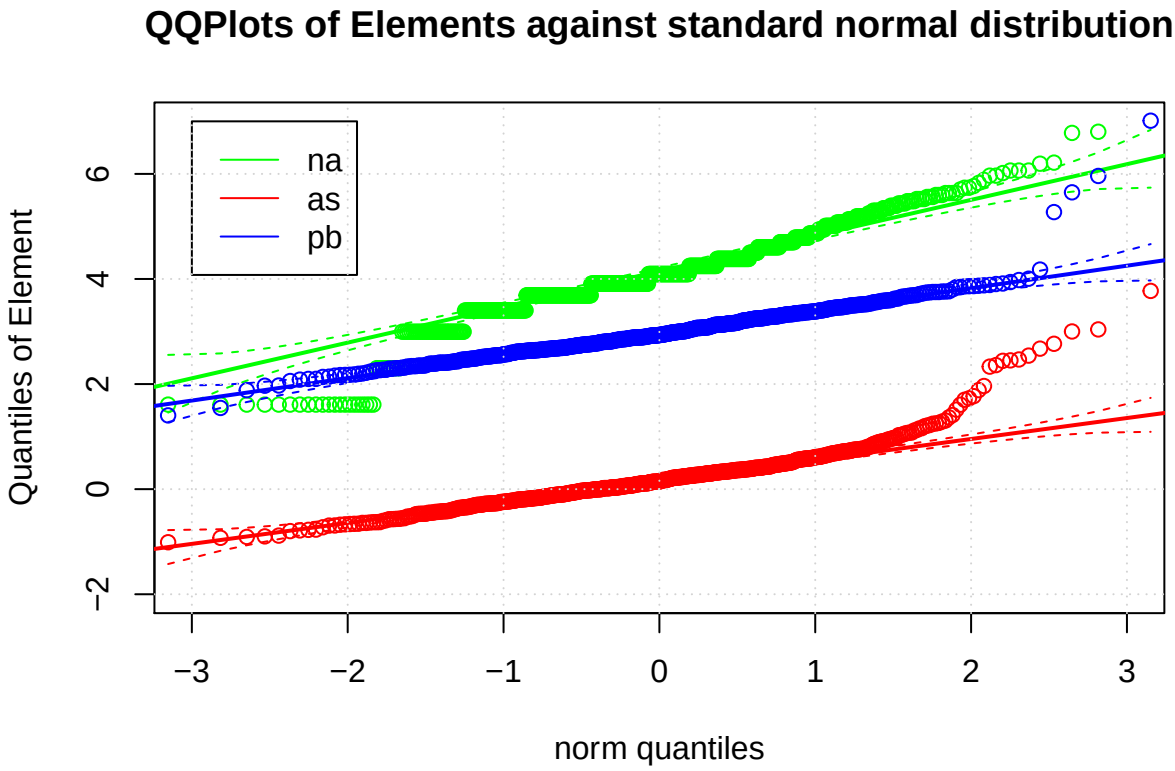
```
x <- rnorm(500, -8, 2.06)
qqplot.das(x, main="QQPlot of rnorm")
grid()
```



In this plot we can see the quantiles of our rnorm distribution (x on the y-axis) plotted against the quantiles of a normal distribution. As we can see the plotted points follow a straight line with random deviations. This means that our x distribution follows the second distribution printed on the x-axis (the standard normal distribution). The dotted lines show the confidence intervals for the quantiles in which most of the points should lie if both distributions are of the same family (here this is the case).

The values of the distributions: the datasample x has a mean of -8, the standard normal distr (x-axis) has a mean of 0. This can be seen via the 0.5 quantile, the most middle one (because the distributions are symmetric). The standard deviation of x ranges at about 2, this can be seen via the quantile for -1 and 1 on the x-axis (1 standard deviation). The amount of values cannot really be calculated by looking at the graph, since the points are hard to count and the emphasis lies on comparing two complete distributions, not single values.

```
qqplot.das(na, ylim=c(-2,7), xlim=c(-3,3), col="green", main="QQPlots of Elements against standard norma
qqplot.das(as, add=TRUE, col="red")
qqplot.das(pb, add=TRUE, col="blue")
legend(-3,7,legend=c("na","as","pb"), col=c("green","red","blue"), lty=1)
grid();
```



As described above the na distribution shows steps. This can come from rounding errors for example. All three samples have systematic deviation from the standard normal distribution in the form of outliers in the upper bounds of the samples. Especially those from As and Pb can be easily be spotted. The Na distribution has strong outliers in the lower bounds as well.

The incline shows the different standard deviations. As and Pb have similar standard deviations, the appear parallel. Na on the other hand has a higher standard deviation, that's why na is more inclined.

As well as the standard deviation the means can be seen as well. That's the offset on the y-axis. As and Pb have a mean difference of about 3 (mean(As) about 0.1, mean(Pb) about 3), the same as in the ecdf plot. Na has a mean of about 4, which is also the same.