R Steinschlag_Datavisualization

Introduction

In this R notebook we are going to explore the data of the "Steinschlag-Challeng, HS19C4". The Idea behind the project is to calculate the probablilty of a death through a rockfall at a slope above a road in Graubünden, Switzerland. The Challenge itself can be found under the following link: Steinschlag Challenge

This notebook is used to visualize the data through numerous plots. The calculation of the probability will be calculated in a simulation and the final findings of this exploration will be published later on a website.

```r
library(tidyverse)
library(grid)
library(gridExtra)
library(chron)
library(psych)
library(knitr)
library(MASS)
library(fitdistrplus)
library(propagate)
library(evd)
library(reticulate)
library(sys)
library(janitor)
library(lubridate)
# Go to Working Directory
getwd()
```

```
## [1] "C:/Users/Roman Studer/Dropbox/01_SG DS/Challenges/Steinschlag/Steinschlag_git_respository/steins
```

```r
# Import Datasets:
out_1 <- read.csv("out_1.csv", sep = ";")
out_2 <- read.csv("out_2.csv", sep = ";")
traffic_density <- read.csv("trafficdensity_per_hour.csv", sep = ";")
```

1. Data transformation

1.1 rename CSV-Colums

```r
out_1 <- rename(out_1, mass = Masse..kg., speed = Geschwindigkeit..m.s., time = Uhrzeit, date = Datum, 
out_2 <- rename(out_2, mass = Masse..kg., speed = Geschwindigkeit..m.s., time = Uhrzeit, date = Datum, 
```

1.2 Transform 'traffic_density' set

The traffic_density data is scaled to 217.0795 percent and needs to be scaled down to 100 percent. Because of that we divide the percentile column by 2.170795

```r
sum(traffic_density$percentile)
```

```
## [1] 217.0795
```

```r
traffic_density <- mutate(traffic_density, percentile = percentile/2.170795 )
sum(traffic_density$percentile)
```

```
## [1] 100
```

1.3 Transform time column into datetime-type

The code below could be used to transform the time column of our dataset into a datetime type.

```
# out_1 <- transform(out_1, time = as.times(time))
# out_2 <- transform(out_1, time = as.times(time))
```

1.4 Combine Datasets

```
#Now that we have two datasets with the same column names we can combine them into one:
data_bind <- rbind(out_1, out_2)
```

1.5 Delete rows with mistakes

In the dataset we see that one event has a mass of zero. Either the rock has a mass below 1 kg or the value is a mistake in the dataset. Because we can't now what is true we are going to work with a subset of our original data that doesn't contain the row with a mass value of zero:

```
rockfall <- subset(data_bind, mass != 0)
```

1.6 Statistic values

Here one can see some statistical values such as mean, standard deviation (sd), median, min and max value.

```
mass <- dplyr::select(rockfall, mass)

stat_data <- dplyr::select(describe(dplyr::select(rockfall, mass, speed, energy)), mean, sd, median, mi
stat_data
```

```
##          mean     sd median  min    max
## mass   463.87 628.13 236.00 3.00 3104.0
## speed   17.93  14.02  10.00 3.60   46.5
## energy  40.45  60.39  19.06 0.46  394.8
```

2. Datavisualization

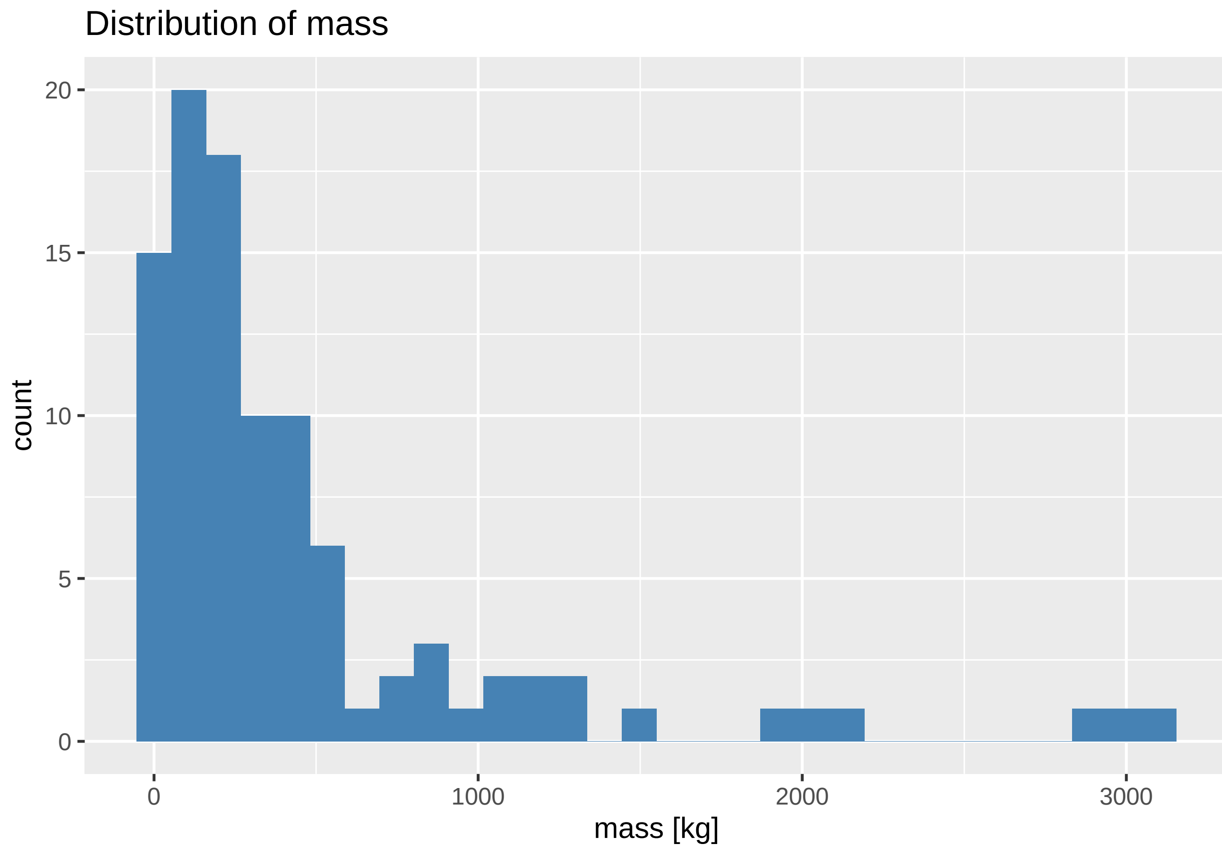
2.1 Histogramms

2.1.1 Mass

The code below plots a histogramm of the "mass"-column in our dataset. Here we see that most of the rocks have a mass below 500kg. This is intersting because we would need more than four rocks of this size to break through the security-net. If we look at other variables we see that we only have two days with this many rockfalls.(The next histogramms are plottet with the same code, just with other variables)

```
ggplot(data = rockfall)+
  # Plot histogram of the mass distribution in the rochfall dataset:
  geom_histogram(mapping = aes(x =mass), fill = "steelblue")+
  labs(title = "Distribution of mass", x='mass [kg]')
```

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

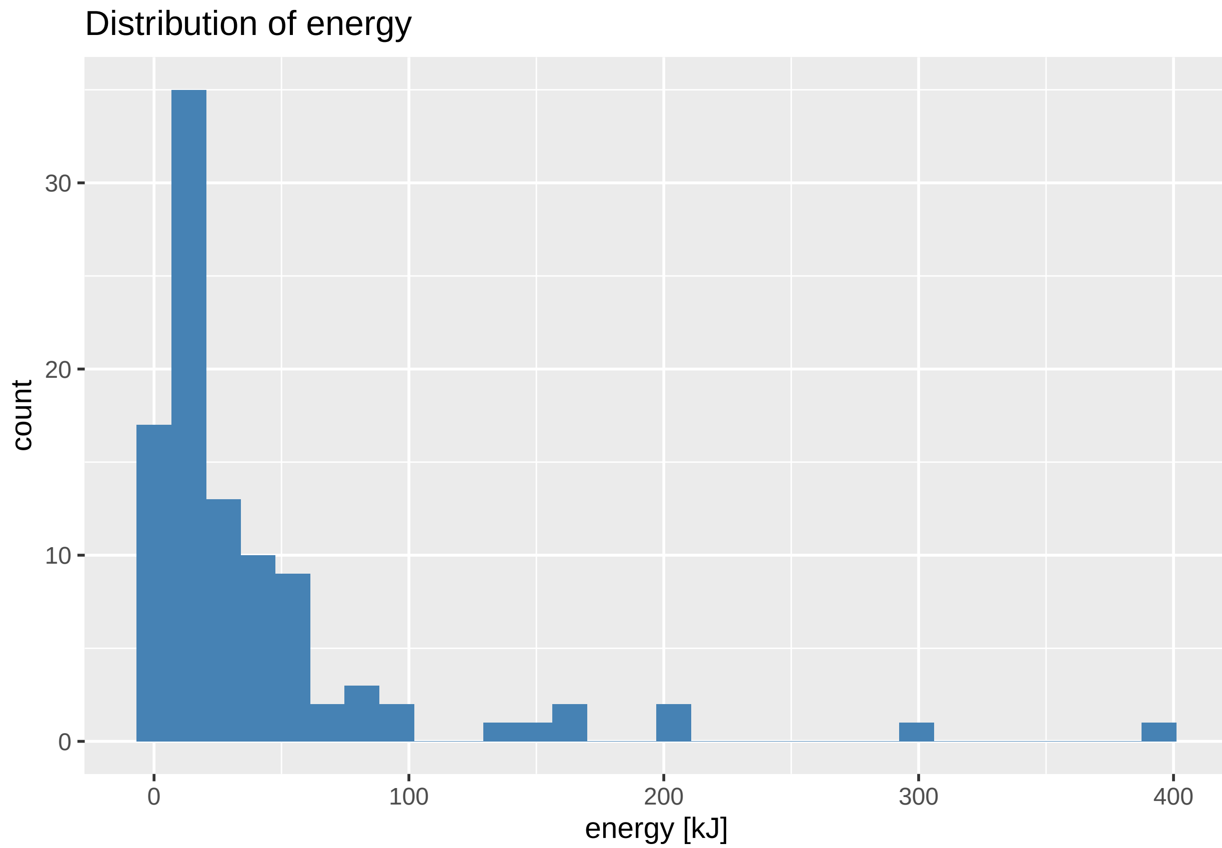


2.1.2 Energy

If we plot the energy (in kilojoule) we can see that a big part of our rocks have a very low energy and we can even see that over 15 have a value of 0 on the plot. This is because we have multiple rocks that don't reach an energy of >10kJ.

```
ggplot(data = rockfall)+
  geom_histogram(mapping = aes(x = energy),fill = "steelblue")+
  labs(title = "Distribution of energy", x='energy [kJ]')
```

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

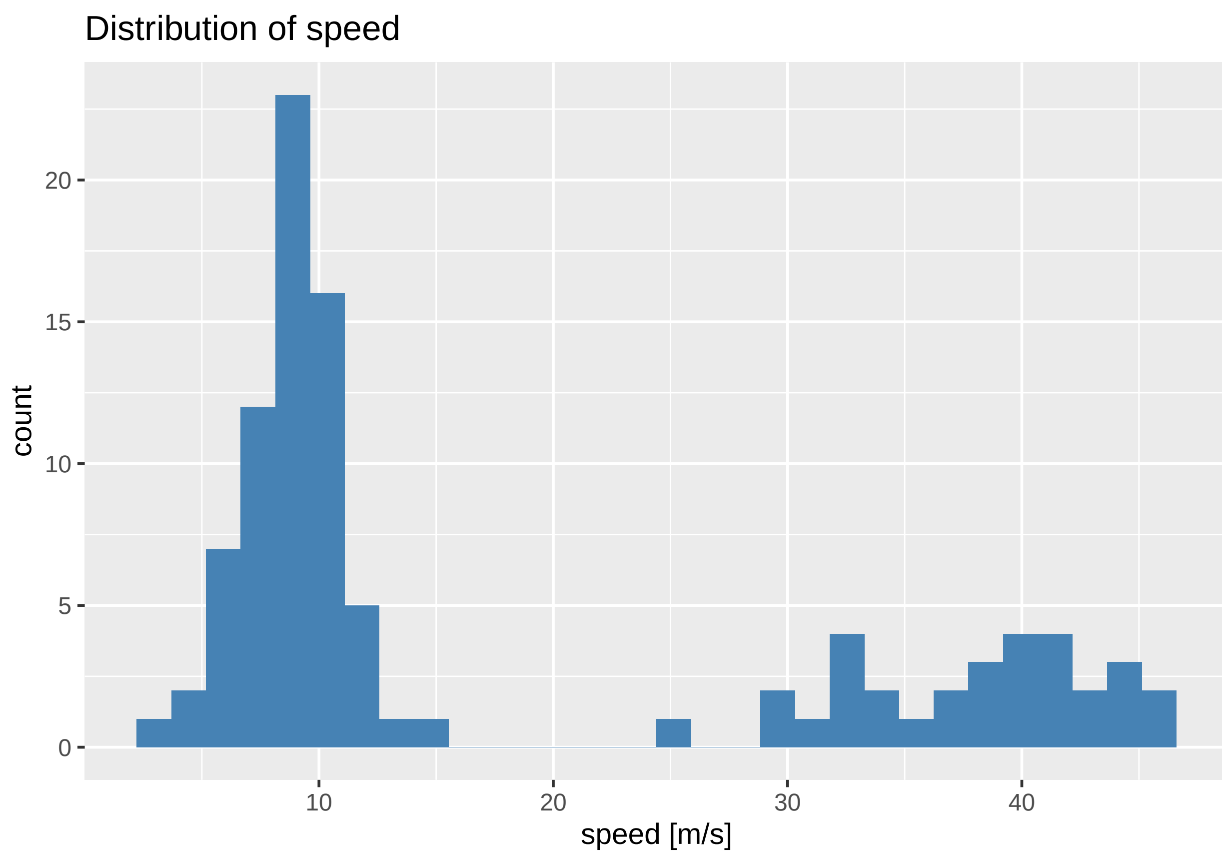


2.1.3 Speed

Here we see that we have a maximum speed of around 46 meters per second. The speed is also split into two groups which indicates shows us the that the two places where rockfalls occur are on a diffrent hight. Later in this notebook I'm going to plot the relationship of the mass and speed.

```
ggplot(data = rockfall)+
  geom_histogram(mapping = aes(x = speed),fill = "steelblue")+
  labs(title = "Distribution of speed", x='speed [m/s]')
```

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

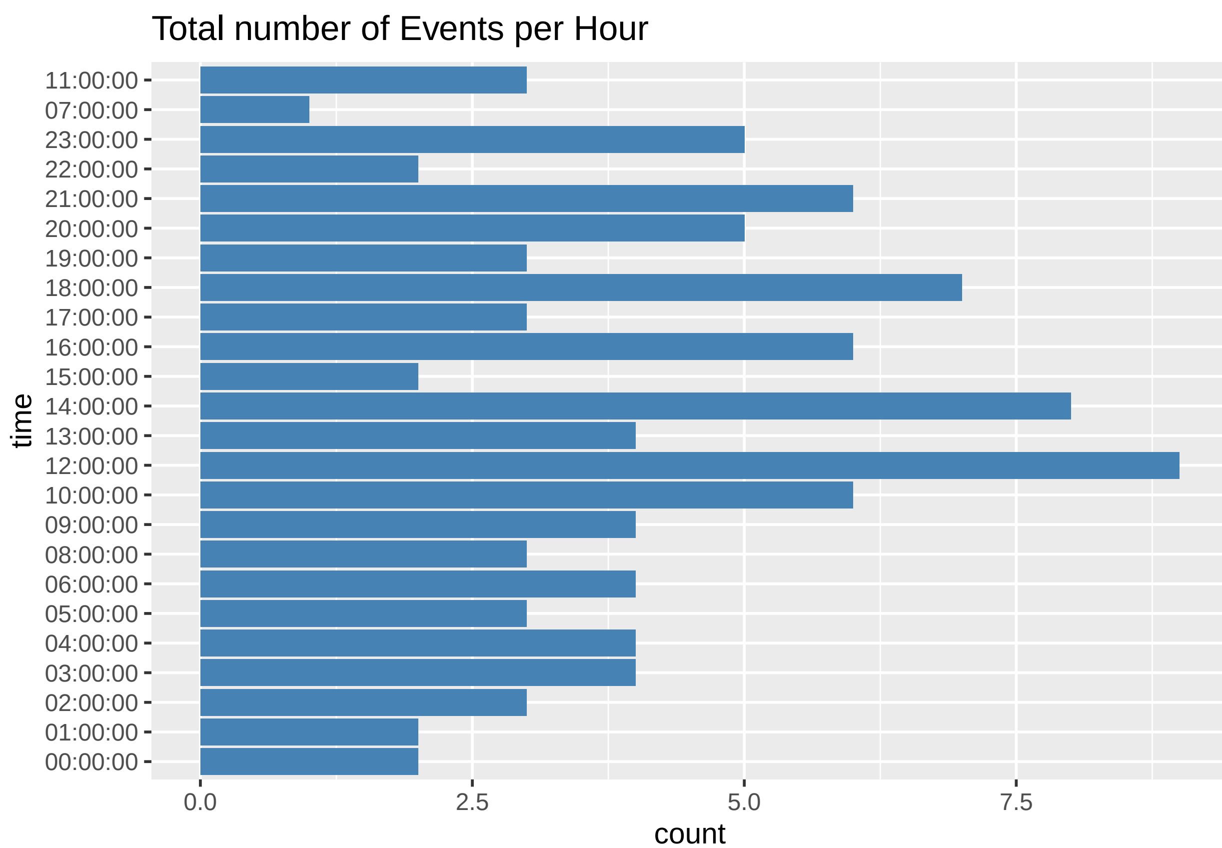


2.1.4 Rockfalls per Hour

As soon as we group the events to the hour they occur we see that we have an increase of rockfalls towards noon and a slower decrease after noon. The most rocks fall at 12 O'clock. At this time the traffic will be high as well, wich means that we should include the traffic density in to our probability calculation.

```r
# count <- select(rockfall, time = n())
ggplot(data = rockfall)+
  geom_histogram(mapping = aes(x = time),fill = "steelblue", stat="count")+
  coord_flip()+
  labs(title = "Total number of Events per Hour")
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



2.2 Graphs

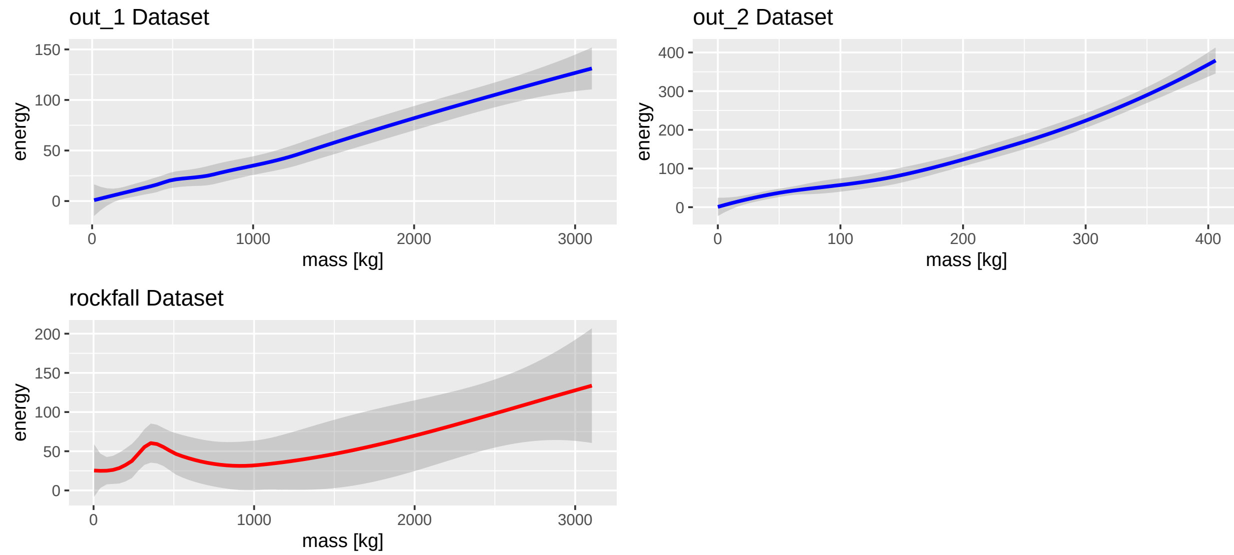
2.2.1 False readouts

Something interesting we found was that the binding of the two datasets can lead to some errors in the datavisualization. The first two graphs below show us how the mass corresponds to the energy of a rockfall. Which seems to be quite linear. This makes sense so far. As soon as we plot the same two variables out of the combined dataset(`rockfall`) we get a spike around 300 to 500 kg. This doesn't seem to make sense. But this tells us that the rocks of the two dataset have to fall from diffrent hights. Because only a greater speed in one dataset would explain the rise of energy in the mass. This also tells us that we can't use the data as a combined dataset and should work with the two seperate datasets. Especially to fit a distribution to the data.

```
ggplot(data = out_1)+
  geom_smooth(mapping = aes(x = mass, y = energy), color = "blue") +
  ggtitle("out_1 Dataset")+
  labs(x='mass [kg]')-> p1

ggplot(data = out_2)+geom_smooth(mapping = aes(x = mass, y = energy), color = "blue") +             ggtitl
  labs(x='mass [kg]')-> p2

ggplot(data = rockfall)+
  geom_smooth(mapping = aes(x = mass, y = energy), color = "red") +
  ggtitle("rockfall Dataset")+
  labs(x='mass [kg]')-> p3
grid.arrange(p1, p2, p3, ncol = 2, nrow = 2)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

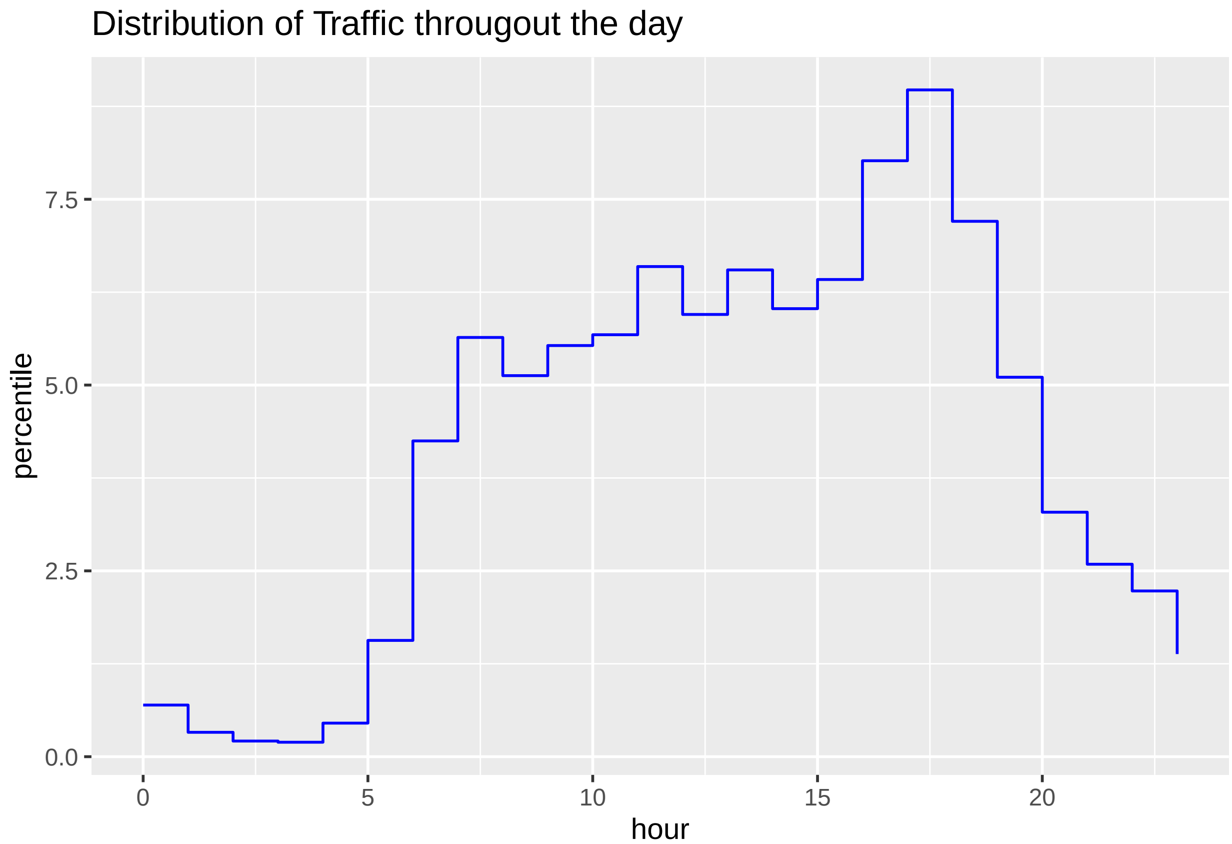
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



2.2.2 Traffic density per hour

The following graph shows the trafiic density on roads in switzerland per hour. This data will be used to calculate the probability of a hit combined with the traffic density. This data has been pulled from the swiss institute for statistics and contains the information of the average of cars passing through per hour. This dataset is from 2015 but is the latest data with this information.

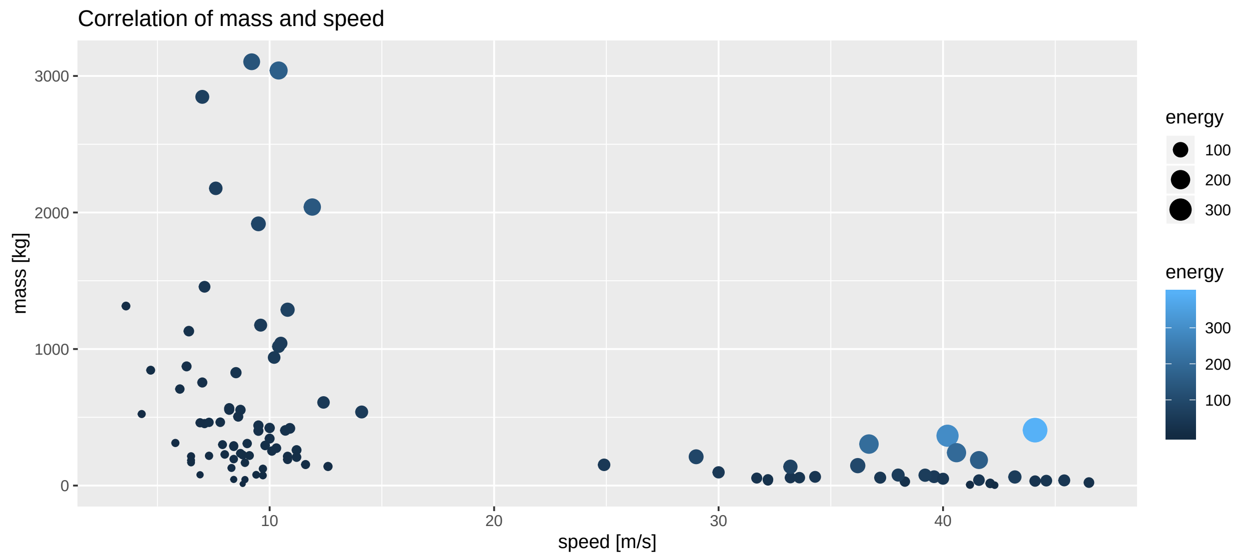
```
ggplot(data = traffic_density)+
  geom_step(mapping = aes(x = hour, y = percentile), color = "blue")+
  ggtitle("Distribution of Traffic througout the day")
```



2.3 Scatterplot

This visualization shows us how the mass and the speed of our rocks corellate. Here we can also see a clear diffrence between the two dataset. (The Energy is also visible as the size and color of the dots. A darker color means less energy.)

```
ggplot(data = rockfall, aes(x = speed, y = mass, size = energy, color = energy ))+
  geom_point()+
  ggtitle("Correlation of mass and speed")+
  labs(x = 'speed [m/s]', y = 'mass [kg]')
```



3. Calculation of the probability of a car getting hit by a rock under the assumption that a rock could break trough the securitynet.

Given:

- Rock breaks through net
- Mass of stone is over two tonnes
- Trafficdensity is 1200 cars per day, without any trafficjam
- Speed of the cars: 16.66 m/s
- Average size of car:
  - Length: 4.4m
  - Width: 1.8m

Assumptions:

- A car getting hit by stone automaticaly causes a deady accident
- Higth of car wont be part of the calculation
- Speed of the rock wont be part of the calculation
- Width of rock is on average about 1 meter

3.1 Probability of Rockfall per Hour

To calculate the expected value at which hour a rockfall takes place we are going to take a look at the Histogram “Rockfalls per Hour” (2.1.4) which shows us the distribution throughout the day. To count the falls per hour we are going to use the function `count` and divide these numbers by 0.99 to get the percentage which also equals the expected value of an event occuring at that hour.

```
exp_rock <- count(rockfall, time)

(exp_rock <- mutate(exp_rock,
                    expected_value = n/0.99/100))
```

```
## # A tibble: 24 x 3
##    time          n expected_value
##    <fct>     <int>          <dbl>
##  1 00:00:00      2         0.0202
##  2 01:00:00      2         0.0202
##  3 02:00:00      3         0.0303
```

```
## 4 03:00:00     4          0.0404
## 5 04:00:00     4          0.0404
## 6 05:00:00     3          0.0303
## 7 06:00:00     4          0.0404
## 8 08:00:00     3          0.0303
## 9 09:00:00     4          0.0404
## 10 10:00:00    6          0.0606
## # ... with 14 more rows
```

3.2 Total Time a Car is in Danger

The next step is to calculate the time a car is in the dangerous zone. We now that the cars drive with 60 km/h and have an average length of 4.4 meters. Because the car can be hit in the front or in the back we need to take double the length plus the width of the stone as our danger zone. Which comes out to be 9.8 meters. With this we calculate the following:

```
#Time to drive a distance of 9.8 meters (2* the length of cars + length of rockd)

speed = 60/3.6 #conversion form km/h to m/s

#time will be calculatet with the formula "distance/speed"
T_single_car = 9.8/speed
print("Time to drive 9.8m :", T_single_car, "seconds")
```

```
## Time to drive 9.8m : 0.588 seconds
```

Now that we have the time a car is in the "danger"-zone we can calculate the total time (seconds in hour) in which cars are in danger of getting hit by a rock breaking though the net. For this we are going to create a new list called "car_per_hour" which will contain the total count of cars driving through, the total time of cars being in danger and the percentage of a car driving through this zone at this hour.

After that we can calculate the expected value of a car getting hit by simply multiplying the probability of a rock falling at this hour and the probability of a car being in the dangerous zone at this hour. We calculate an expected value of 0.009241006 or 0.924% for this event to happen.

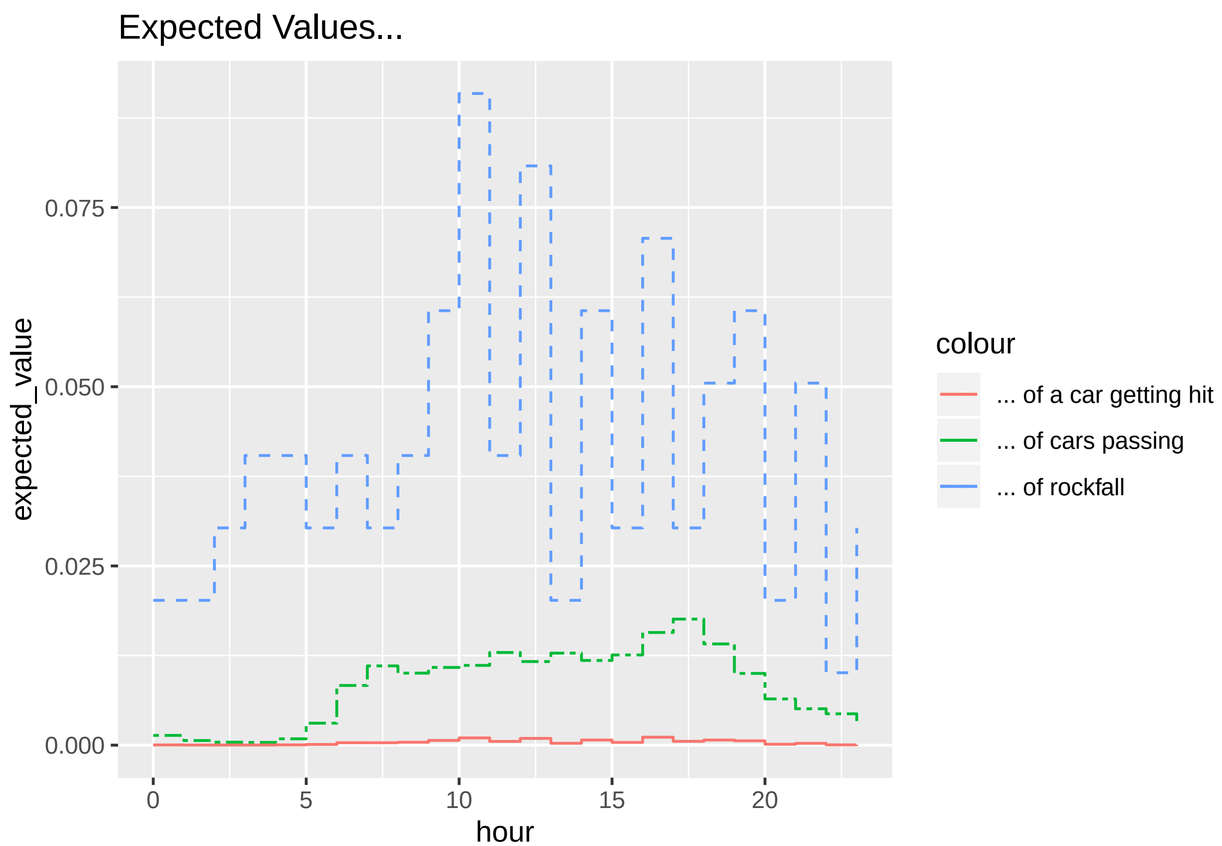
```
# calculate cars passing though per hour
car_per_hour <- mutate(traffic_density,
                       cars_passing = percentile * 12,
                       time_in_danger = cars_passing * 0.588,
                       exp_car = time_in_danger/3600)



(expected_value_car <- mutate(exp_rock, time, n, expected_value,
                          car_passing_hour = car_per_hour$cars_passing,
                          t_car_in_danger = car_per_hour$time_in_danger,
                          exp_car_per_hour = car_per_hour$exp_car,
                          t_exp_per_hour = expected_value * car_per_hour$exp_car))
```

```
## # A tibble: 24 x 7
##   time      n expected_value car_passing_hour t_car_in_danger
##   <fct> <int>          <dbl>            <dbl>           <dbl>
## 1 00:0~     2         0.0202             8.34            4.90
## 2 01:0~     2         0.0202             3.94            2.31
## 3 02:0~     3         0.0303             2.53            1.48
## 4 03:0~     4         0.0404             2.33            1.37
## 5 04:0~     4         0.0404             5.42            3.19
## 6 05:0~     3         0.0303            18.8            11.0
```

```
##  7 06:0~      4          0.0404             51.0            30.0
##  8 08:0~      3          0.0303             67.7            39.8
##  9 09:0~      4          0.0404             61.5            36.2
## 10 10:0~      6          0.0606             66.4            39.0
## # ... with 14 more rows, and 2 more variables: exp_car_per_hour <dbl>,
## #   t_exp_per_hour <dbl>
```

```
total_in_danger <- dplyr::select(expected_value_car, exp_car_per_hour)
```

```
expected_value_car <- mutate(expected_value_car,
hour = traffic_density$hour)

ggplot(data = expected_value_car)+
  geom_step(mapping = aes(x = hour, y = expected_value, color = '... of rockfall'), linetype = "dashed")
  geom_step(mapping = aes(x = hour, y = exp_car_per_hour, color = "... of cars passing"), linetype = "tw
  geom_step(mapping = aes(x = hour, y = t_exp_per_hour, color = "... of a car getting hit"))+
  labs(title='Expected Values...')
```



The expected value of a car getting hit by rockfall at the assumption that a rockfall, capable of breaking through the net, occurs is:

```
# the probability of a car getting hit by a rock incase it breaks through the security net.
e_car_hit <- sum(expected_value_car$t_exp_per_hour)
e_car_hit
```

```
## [1] 0.009241006
```

4. Net-Energy

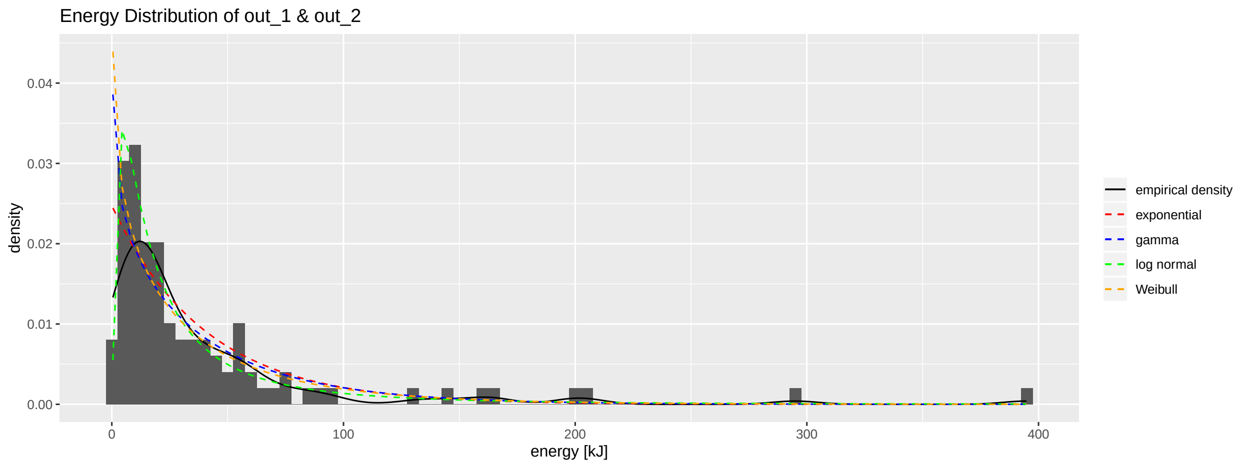
4.1 Net-Energy Distribution

We aim to find an adequate density and distribution function based on the energy of both given data-sets `out_1`" and `out_2`" to be able to calculate the $P(X \geq x|stonefallsintothenet)$ where X is a continuous variable of the calculated energy $energy[kJ] = \frac{0.5*m*v^2}{1000}$. Since there is just one net and the energy distribution of the net doesn't care from wich source the rock falls, both data-sets are considered to evaluate an appropriate density funciton.

To get an imagination of how the data might be distributed, we plot the density of the observed energy with a histogramm.

The comparison of different density functions with the approximated function of the observed energy shows how well each function fit the actual values of the rockfall.

```
e_fit <- fitdistr(rockfall$energy, "exponential")
g_fit <- fitdistr(rockfall$energy, "gamma")
lnrm_fit <- fitdistr(rockfall$energy, "lognormal")
w_fit <- fitdistr(rockfall$energy, "weibull")

ggplot(rockfall, aes(energy)) +
  geom_histogram(aes(y = ..density..), binwidth = 5) +
  geom_line(stat = "density", aes(color = "empirical density", linetype = "empirical density")) +
  ggtitle("Energy Distribution of out_1 & out_2") +
  stat_function(fun = dexp, args = list(e_fit$estimate[1]),
                aes(color = "exponential", linetype = "exponential")) +
  stat_function(fun = dgamma, args = list(g_fit$estimate[1], g_fit$estimate[2]),
                aes(color = "gamma", linetype = "gamma")) +
  stat_function(fun = dlnorm, args = list(lnrm_fit$estimate[1], lnrm_fit$estimate[2]),
                aes(color = "log normal", linetype = "log normal")) +
  stat_function(fun = dweibull, args = list(w_fit$estimate[1], w_fit$estimate[2]),
                aes(color = "Weibull", linetype = "Weibull")) +
  scale_color_manual(name = "",
                     values = c("empirical density" = "black",
                                "exponential" = "red",
                                "gamma" = "blue",
                                "log normal" = "green",
                                "Weibull" = "orange"),
                     breaks = c("empirical density",
                                "exponential",
                                "gamma",
                                "log normal",
                                "Weibull")) +
  scale_linetype_manual(name = "",
                        values = c("empirical density" = "solid",
                                   "exponential" = "dashed",
                                   "gamma" = "dashed",
                                   "log normal" = "dashed",
                                   "Weibull" = "dashed"),
                        breaks = c("empirical density",
                                   "exponential",
                                   "gamma",
                                   "log normal",
                                   "Weibull")) +
  labs(x = "energy [kJ]")
```



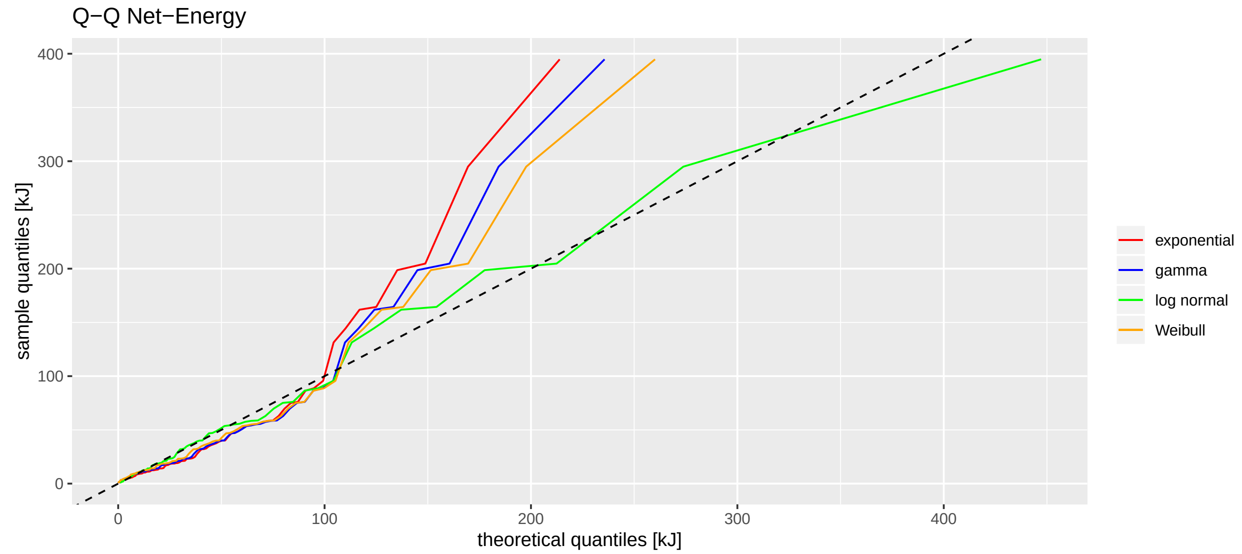
4.2 Net-Energy Density Function Evaluation

Now we have to figure out which distribution fits our energy data best. QQ_plots have the purpose to compare both, theoretical quantiles of the given distribution on the y-axis vs. the actual quantiles of the sample on the x-axis. The more the quantiles match, the closer the data points approach a straight line. The red line placed over the plot describes the line which the data points should approximate if they are distributed perfectly.

We figured out that the log normal distribution seems to have a very good fit on the energy distribution on the net. The comparison shows, that the exponential, gamma, and weibull distribution doesn't fit very well in regards to the extreme values. It is important to mention that each data point does not have a connection as one can see in the QQ-plot. But it helps to get a better overview of the data points.

```
ggplot(rockfall, aes(sample = energy)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qgamma,
          dparams = list(g_fit$estimate[1], g_fit$estimate[2]),
          geom = "line",
          aes(color = "gamma")) +
  stat_qq(distribution = qlnorm,
          dparams = list(lnrm_fit$estimate[1], lnrm_fit$estimate[2]),
          geom = "line",
          aes(color = "log normal")) +
  stat_qq(distribution = qweibull,
          dparams = list(w_fit$estimate[1], w_fit$estimate[2]),
          geom = "line",
          aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("exponential" = "red",
                                "gamma" = "blue",
                                "log normal" = "green",
                                "Weibull" = "orange"),
                     breaks = c("exponential",
                                "gamma",
                                "log normal",
                                "Weibull")) +
```

```r
  labs(title = "Q-Q Net-Energy", x = "theoretical quantiles [kJ]", y = "sample quantiles [kJ]")
```

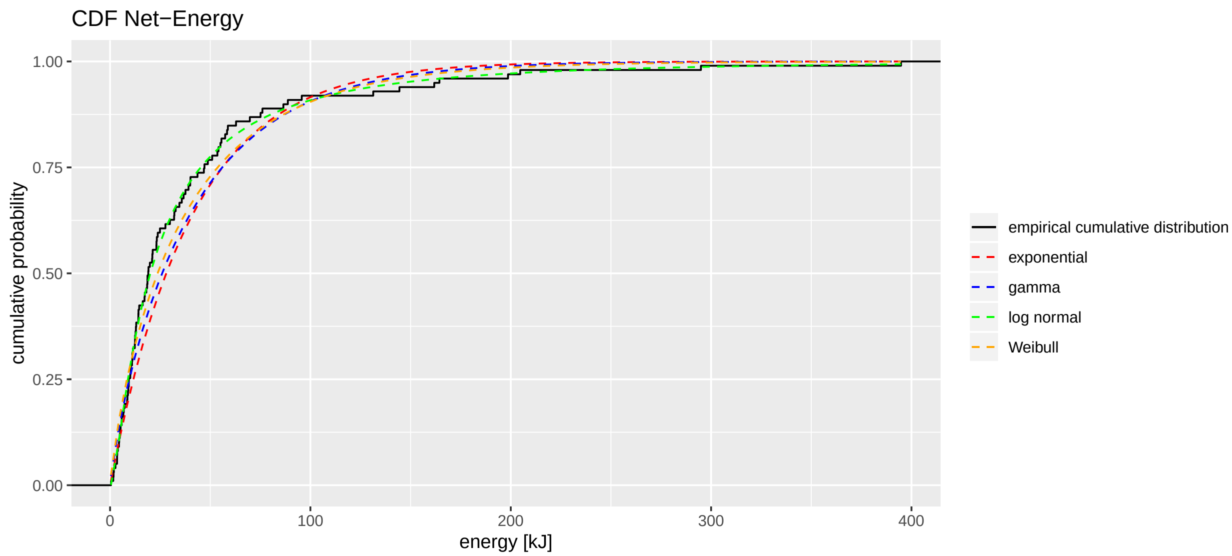


4.3 Net-Energy Cumulative Distribution Function

Here one can see that the cumulative distribution of the log normal function matches the empiric values best.

```r
ggplot(rockfall, aes(energy)) +
  stat_ecdf(aes(color = "empirical cumulative distribution",
                linetype = "empirical cumulative distribution")) +
  stat_function(fun = pexp, args = list(e_fit$estimate),
                aes(color = "exponential", linetype = "exponential")) +
  stat_function(fun = pgamma, args = list(g_fit$estimate[1], g_fit$estimate[2]),
                aes(color = "gamma", linetype = "gamma")) +
  stat_function(fun = plnorm, args = list(lnrm_fit$estimate[1], lnrm_fit$estimate[2]),
                aes(color = "log normal", linetype = "log normal")) +
  stat_function(fun = pweibull, args = list(w_fit$estimate[1], w_fit$estimate[2]),
                aes(color = "Weibull", linetype = "Weibull")) +
  scale_color_manual(name = "",
                     values = c("empirical cumulative distribution" = "black",
                                "exponential" = "red",
                                "gamma" = "blue",
                                "log normal" = "green",
                                "Weibull" = "orange"),
                     breaks = c("empirical cumulative distribution",
                                "exponential",
                                "gamma",
                                "log normal",
                                "Weibull")) +
  scale_linetype_manual(name = "",
                        values = c("empirical cumulative distribution" = "solid",
                                   "exponential" = "dashed",
                                   "gamma" = "dashed",
                                   "log normal" = "dashed",
                                   "Weibull" = "dashed"),
                        breaks = c("empirical cumulative distribution",
                                   "exponential",
```

```
                                      "gamma",
                                      "log normal",
                                      "Weibull")) +
  labs(title = "CDF Net-Energy", x = "energy [kJ]", y = "cumulative probability")
```

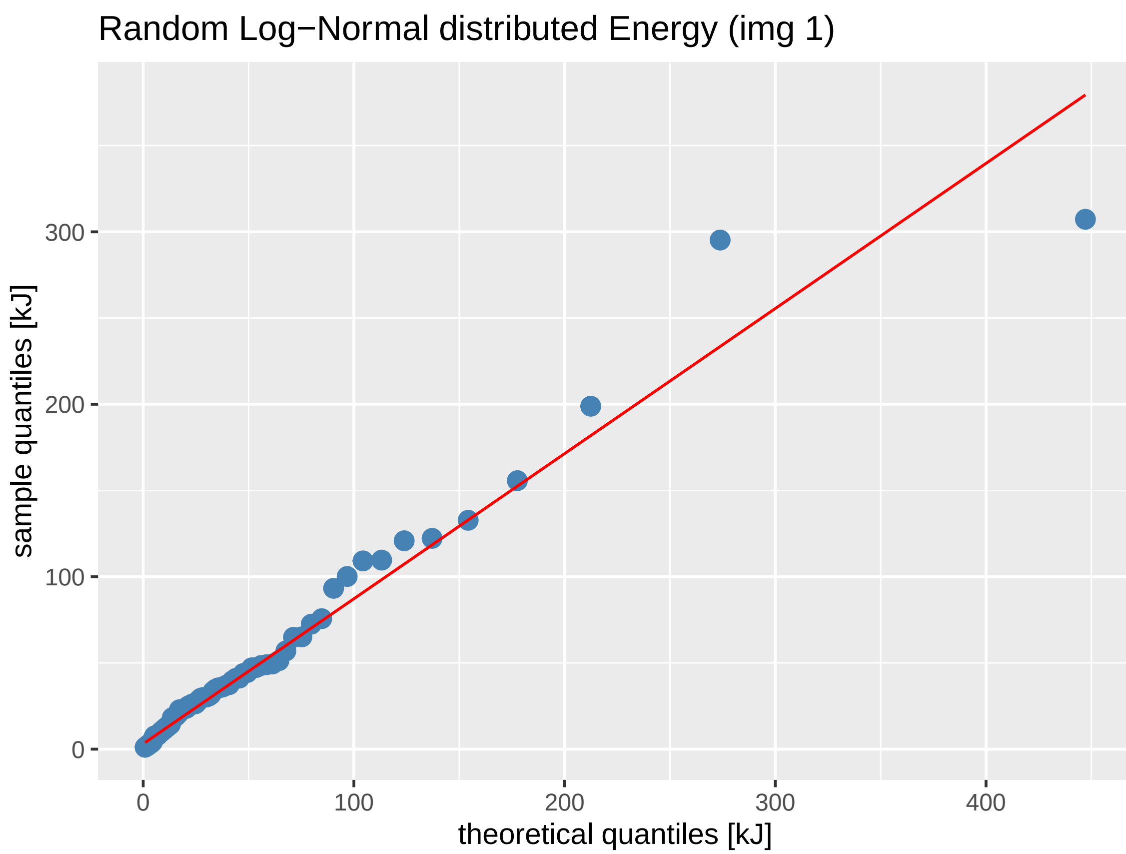


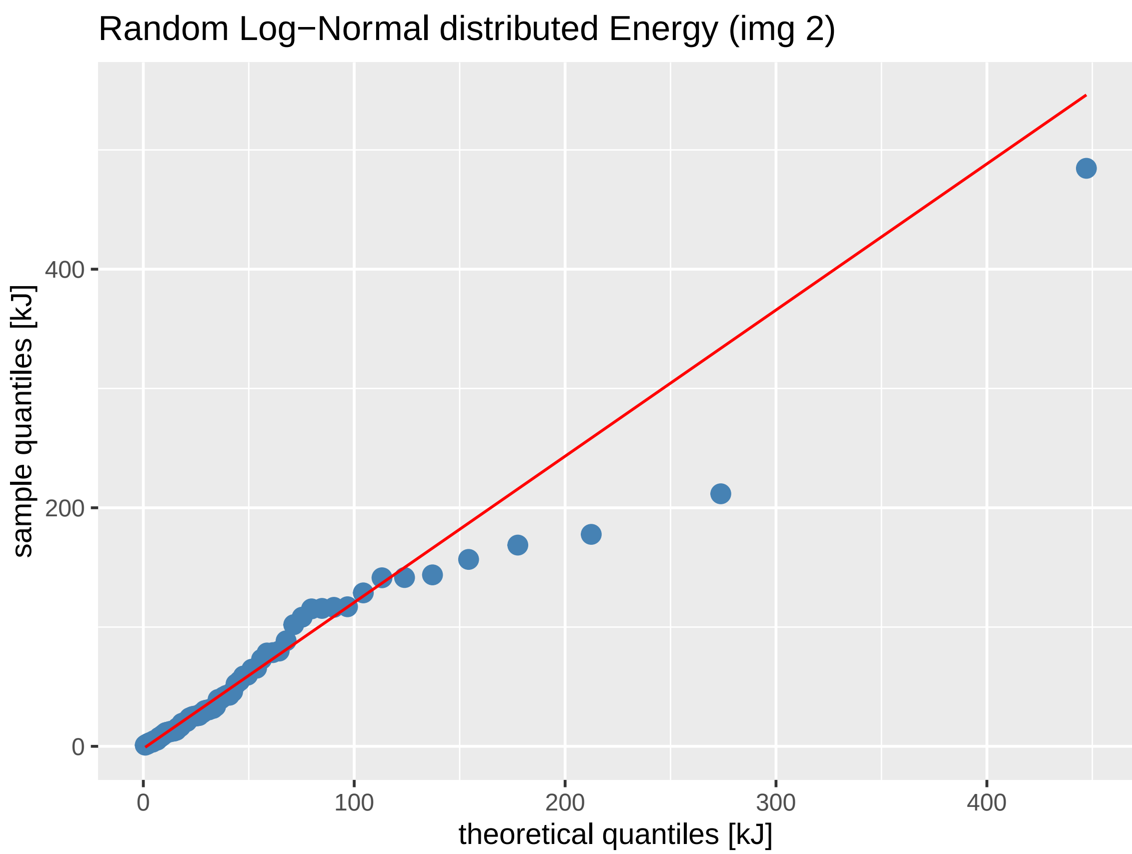
4.4 Net-Energy Log-Norm Distribution Validation

To be sure, we test the sample if it is log normal distributed. We assume (H_0) that our data is log normal distributed with significant level 95%. H_1 is that the energy of the data set is not log normal distributed. Therefore, we compare 20 smples of random generated log normal distributed quantiles vs. the theoretical quantiles. If there is a similar QQ-plot compared to the plot with the actual energy data, then we can assume that the energy of the rockfall is log-normal distributed.

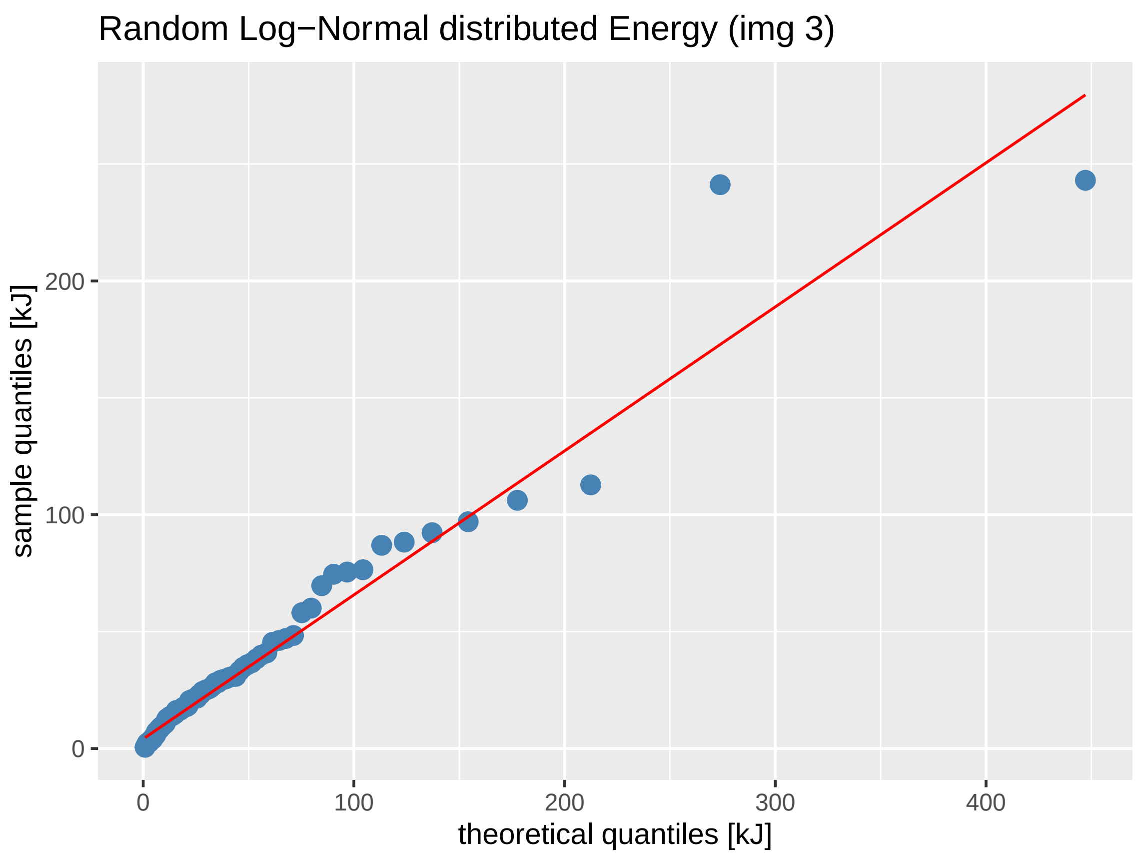
The comparison of the artificial created data sample and the data set of the energy let us assume, not to deny H_0. For this reason, we can transform the energy values with the $ln(x)$ to get a normal distribution in order to compute the probability with the normal distribution function.

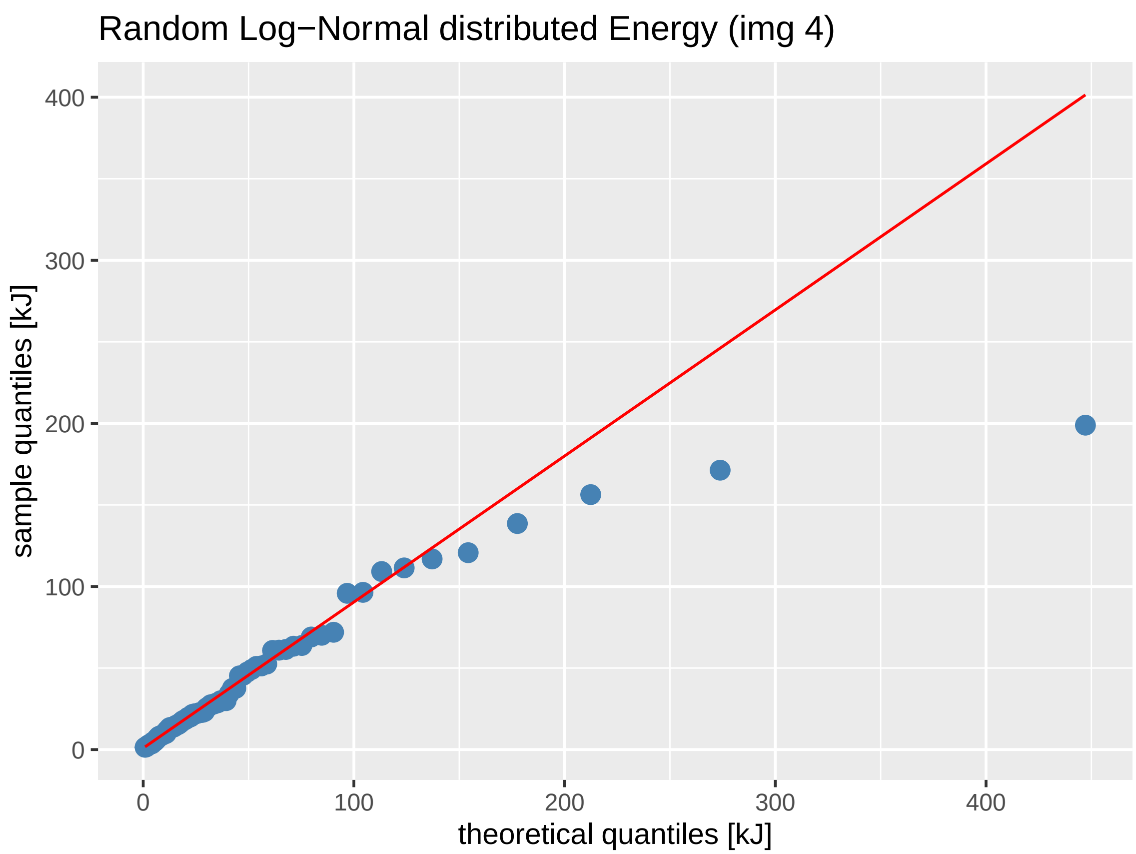
```
rows <- 20
cols <- length(rockfall$energy)
random_dlnorm_matrix <- matrix(nrow = rows, ncol = cols)
for (i in 1:rows){
  random_dlnorm_matrix[i,] <- c(rlnorm(cols, meanlog = lnrm_fit$estimate[1], sdlog = lnrm_fit$estimate[
}
for (j in 1:rows){
  p <- ggplot2::qplot(sample = random_dlnorm_matrix[j,], geom = "blank") +
    stat_qq(distribution = qlnorm, dparams = list(lnrm_fit$estimate[1], lnrm_fit$estimate[2]), color =
    stat_qq_line(distribution = qlnorm, dparams = list(lnrm_fit$estimate[1], lnrm_fit$estimate[2]), col
    labs(title = paste0("Random Log-Normal distributed Energy (img ", j, ")"), x = "theoretical quantil
  print(p)
}
```

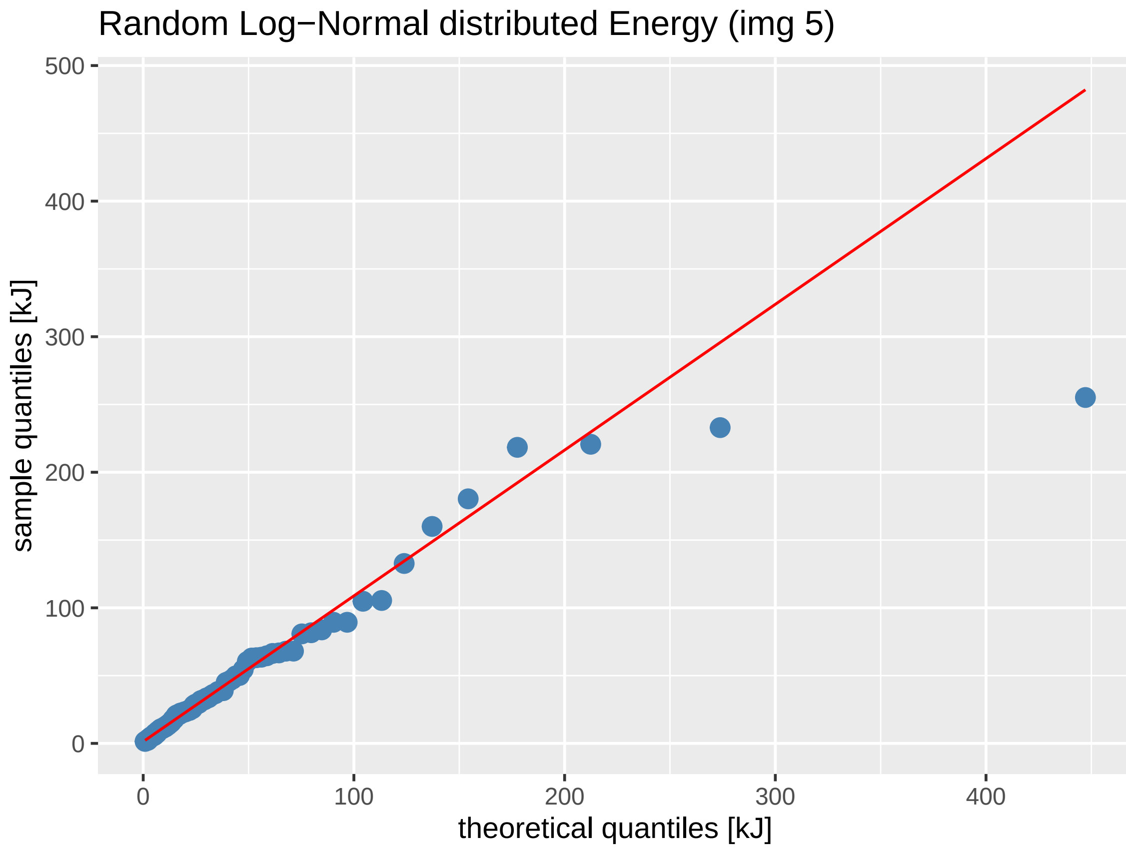

Random Log−Normal distributed Energy (img 1)
sample quantiles [kJ]
theoretical quantiles [kJ]
0
100
200
300
0
100
200
300
400

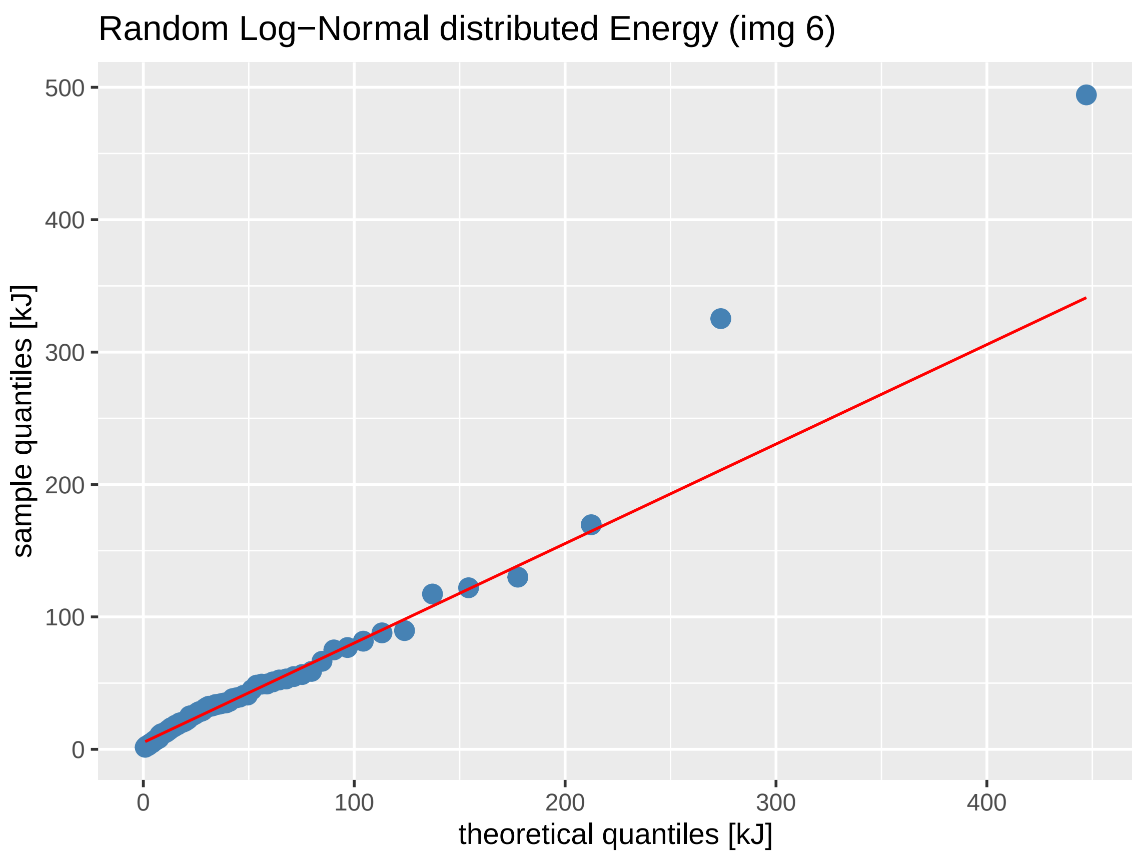


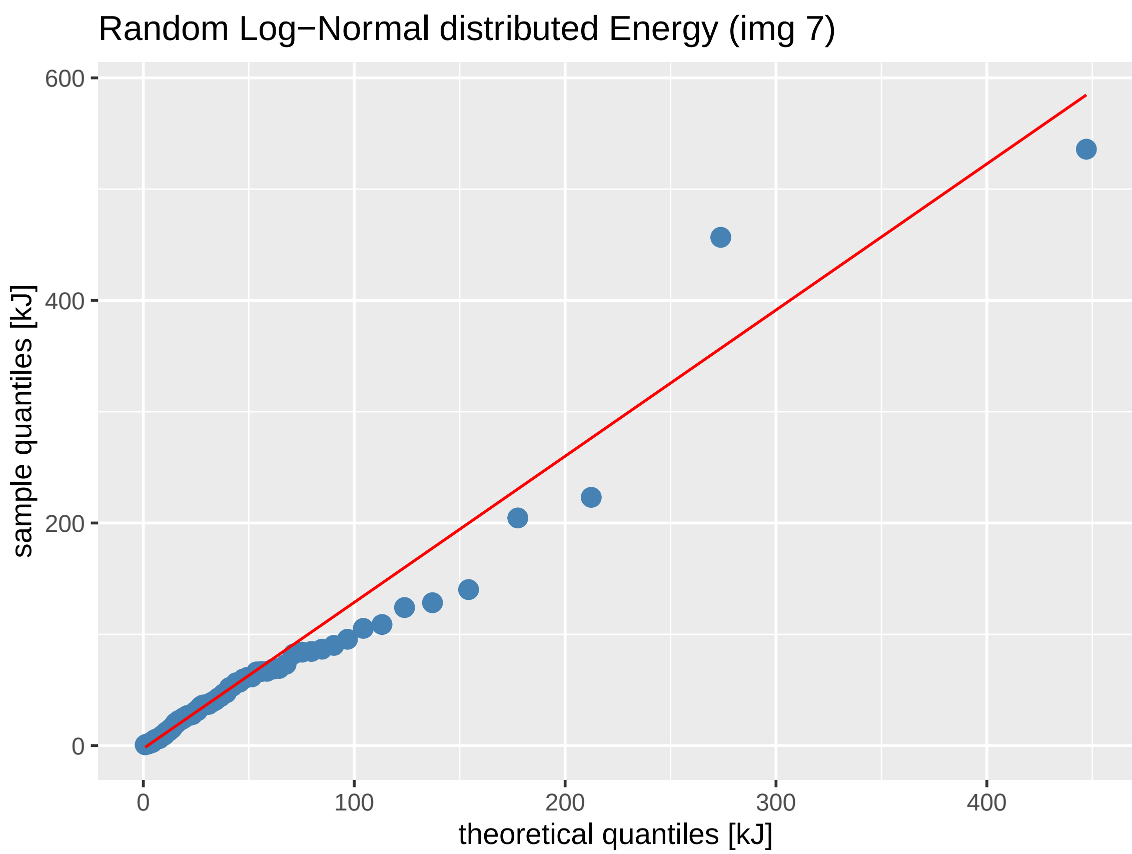
Random Log−Normal distributed Energy (img 2)
sample quantiles [kJ]
400
200
0
0
100
200
300
400
theoretical quantiles [kJ]

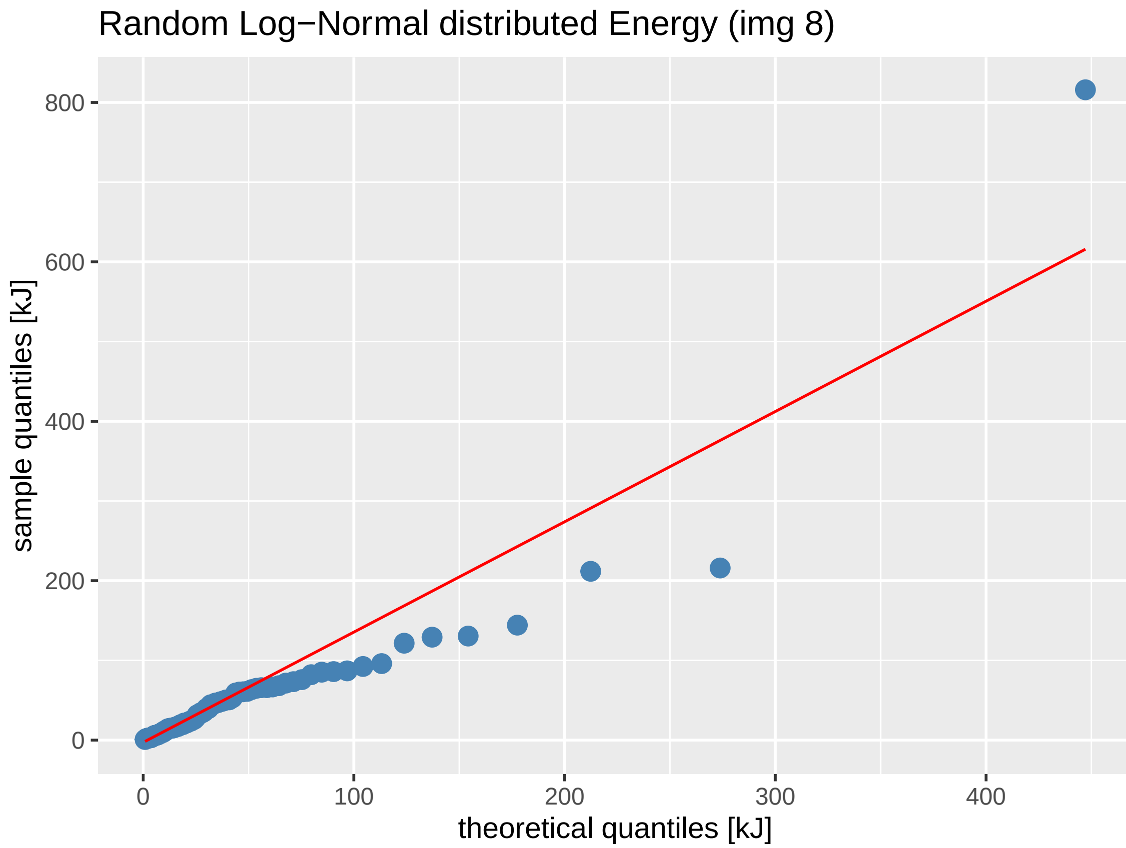

Random Log−Normal distributed Energy (img 3)
sample quantiles [kJ]
theoretical quantiles [kJ]
0
100
200
0
100
200
300
400

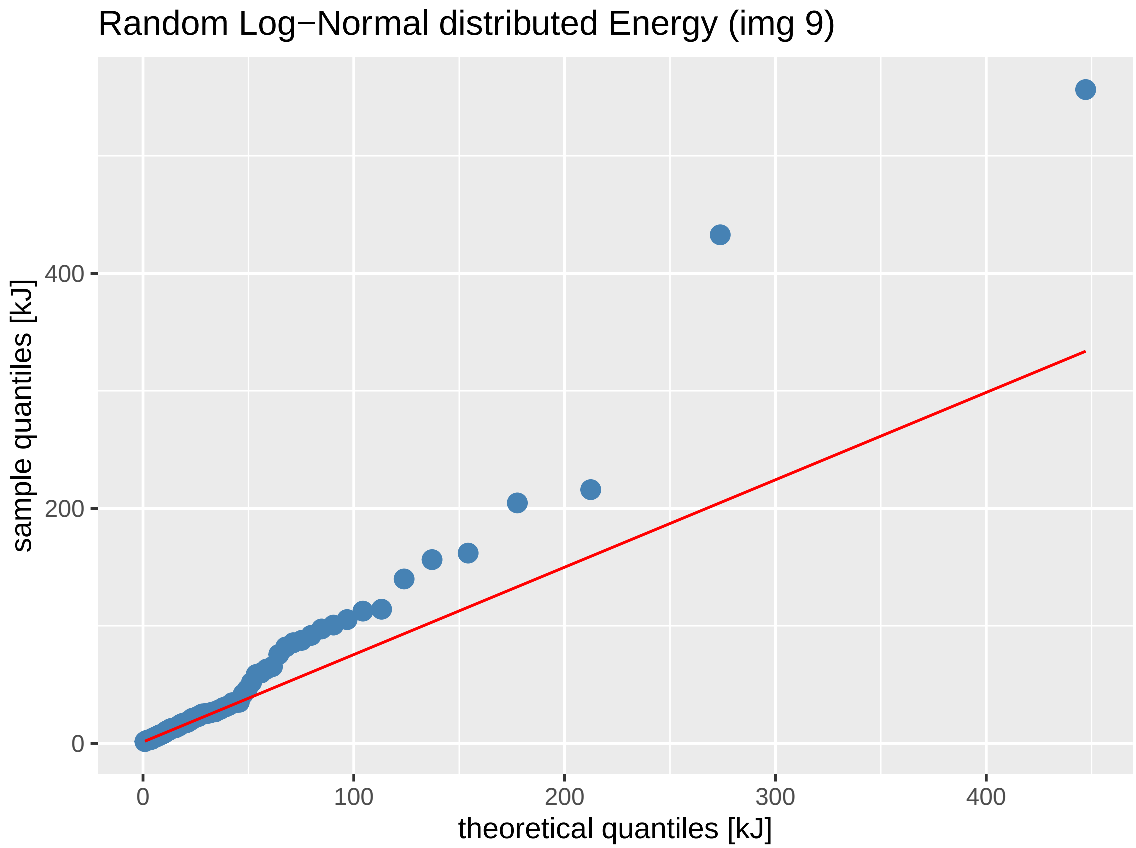

Random Log−Normal distributed Energy (img 4)
sample quantiles [kJ]
theoretical quantiles [kJ]
0
100
200
300
400
0
100
200
300
400

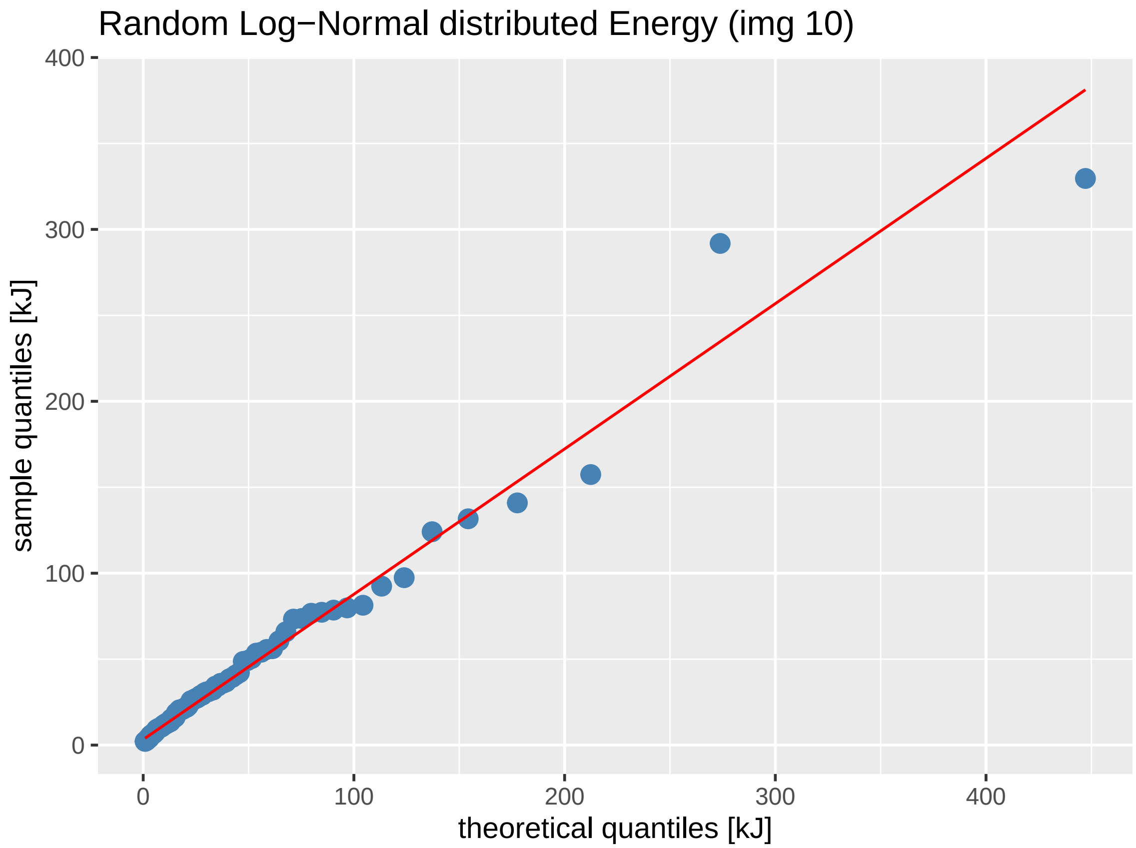

Random Log−Normal distributed Energy (img 5)
sample quantiles [kJ]
theoretical quantiles [kJ]
0
100
200
300
400
500

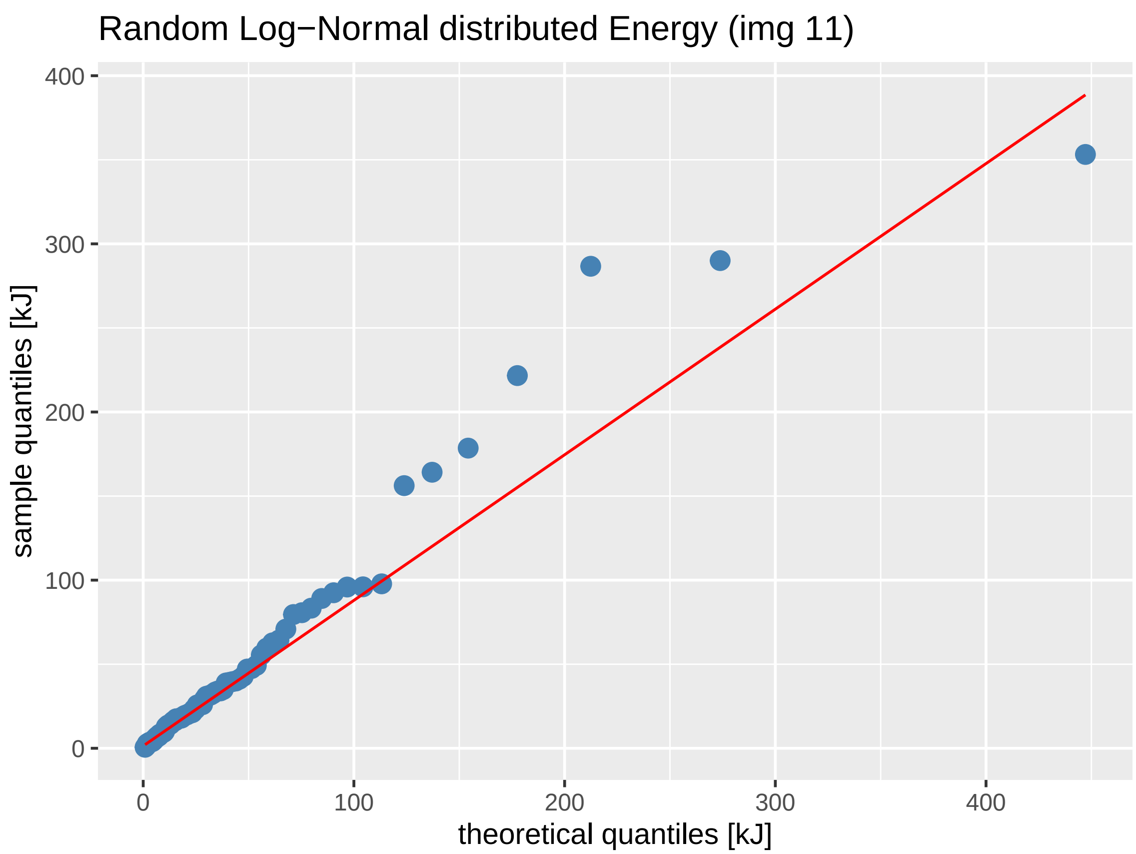

Random Log−Normal distributed Energy (img 6)
sample quantiles [kJ]
theoretical quantiles [kJ]
0
100
200
300
400
500

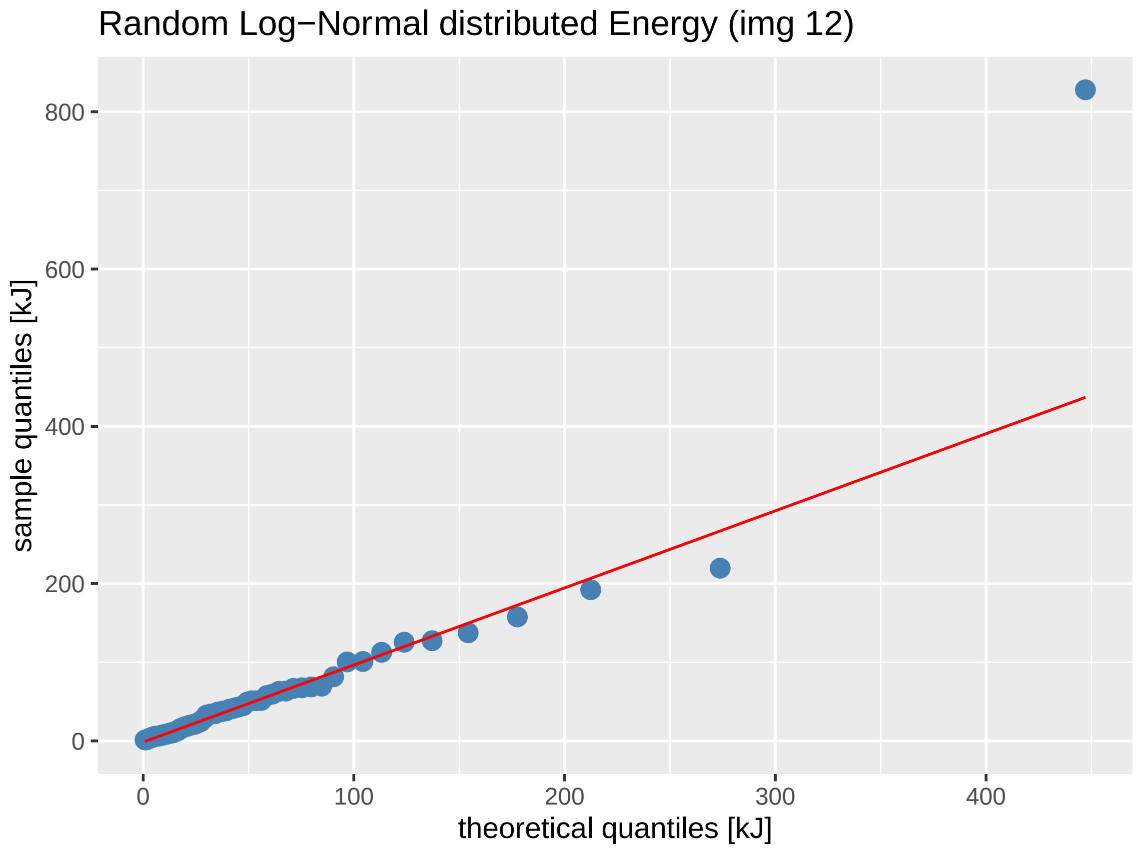

Random Log−Normal distributed Energy (img 7)
sample quantiles [kJ]
600
400
200
0
0
100
200
300
400
theoretical quantiles [kJ]

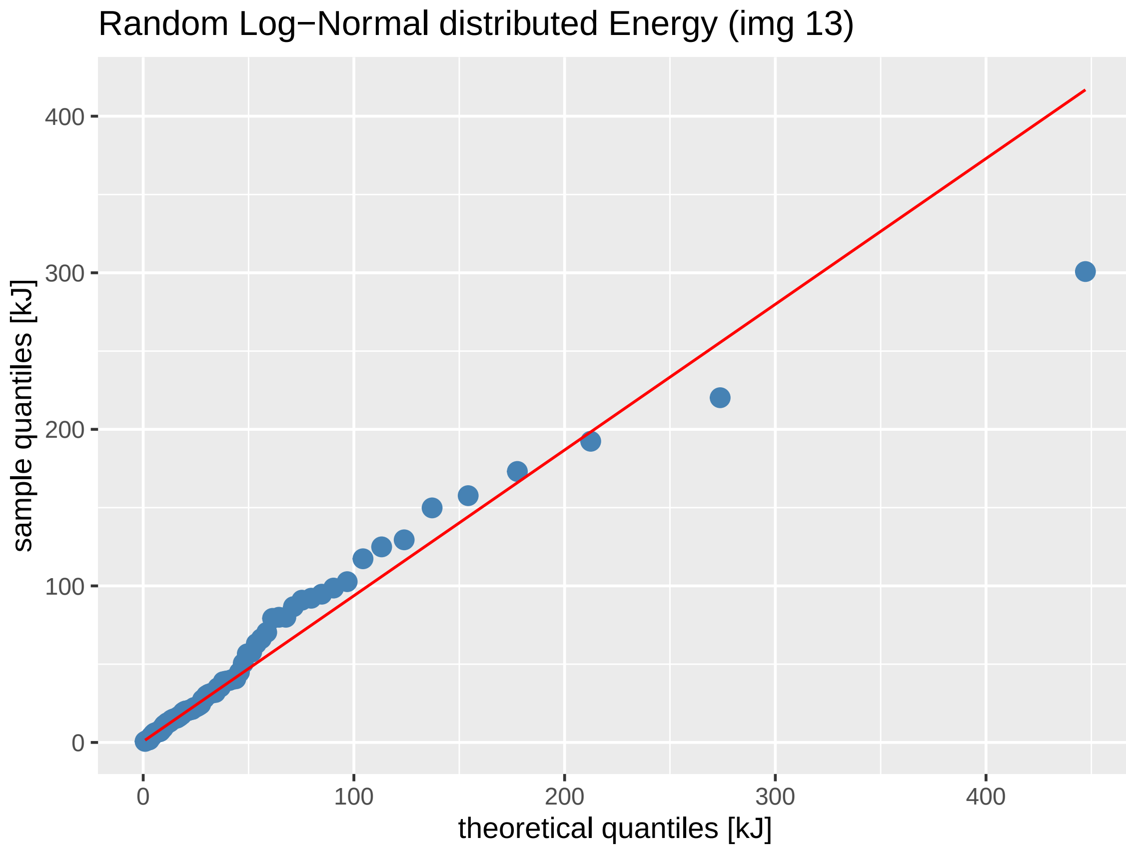

Random Log−Normal distributed Energy (img 8)
sample quantiles [kJ]
800
600
400
200
0
0
100
200
300
400
theoretical quantiles [kJ]


Random Log−Normal distributed Energy (img 9)
sample quantiles [kJ]
400
200
0
0
100
200
300
400
theoretical quantiles [kJ]

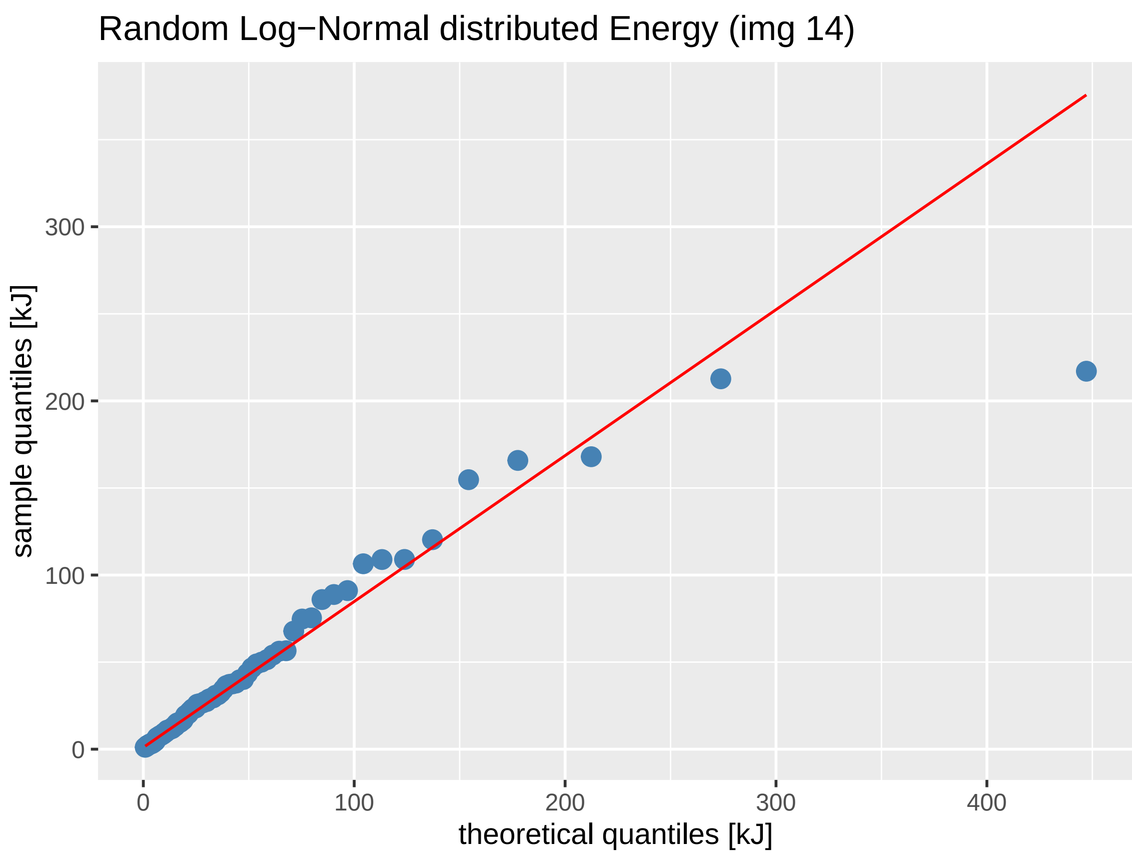

Random Log−Normal distributed Energy (img 10)
400
300
200
100
0
sample quantiles [kJ]
0
100
200
300
400
theoretical quantiles [kJ]

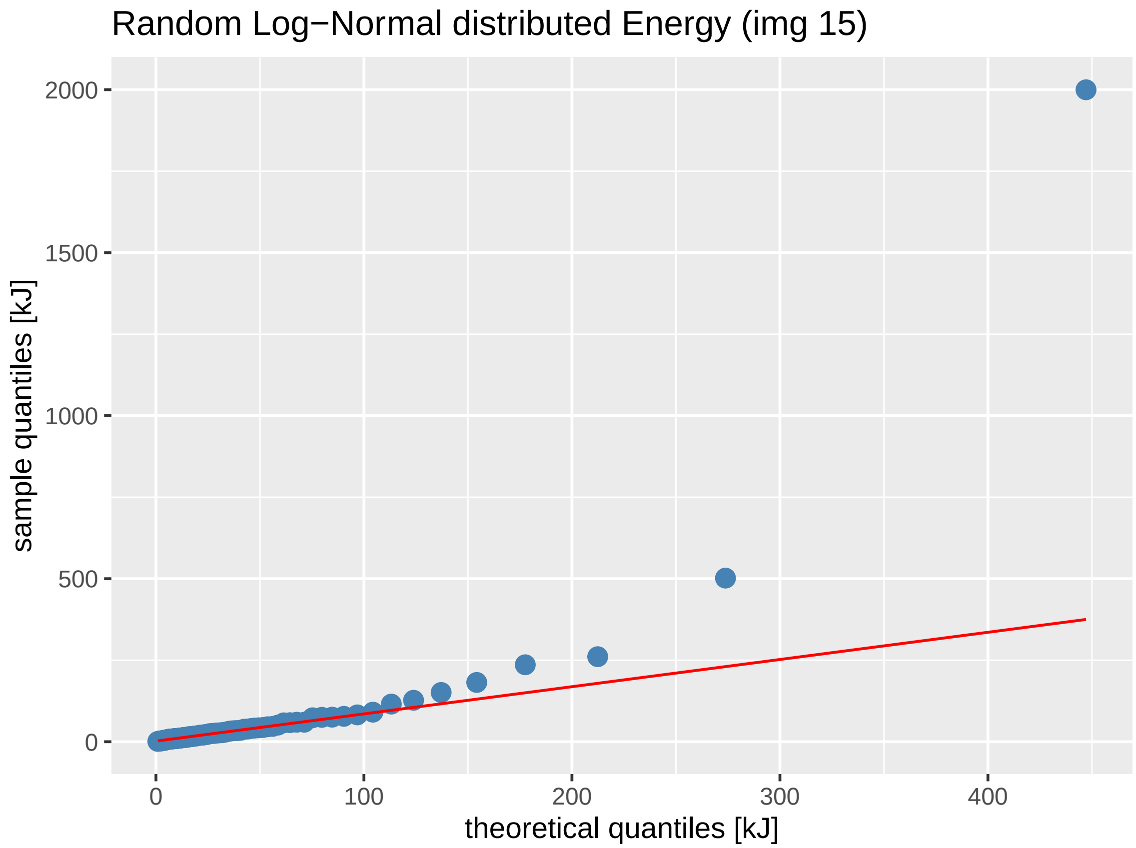

Random Log−Normal distributed Energy (img 11)
sample quantiles [kJ]
400
300
200
100
0
0
100
200
300
400
theoretical quantiles [kJ]

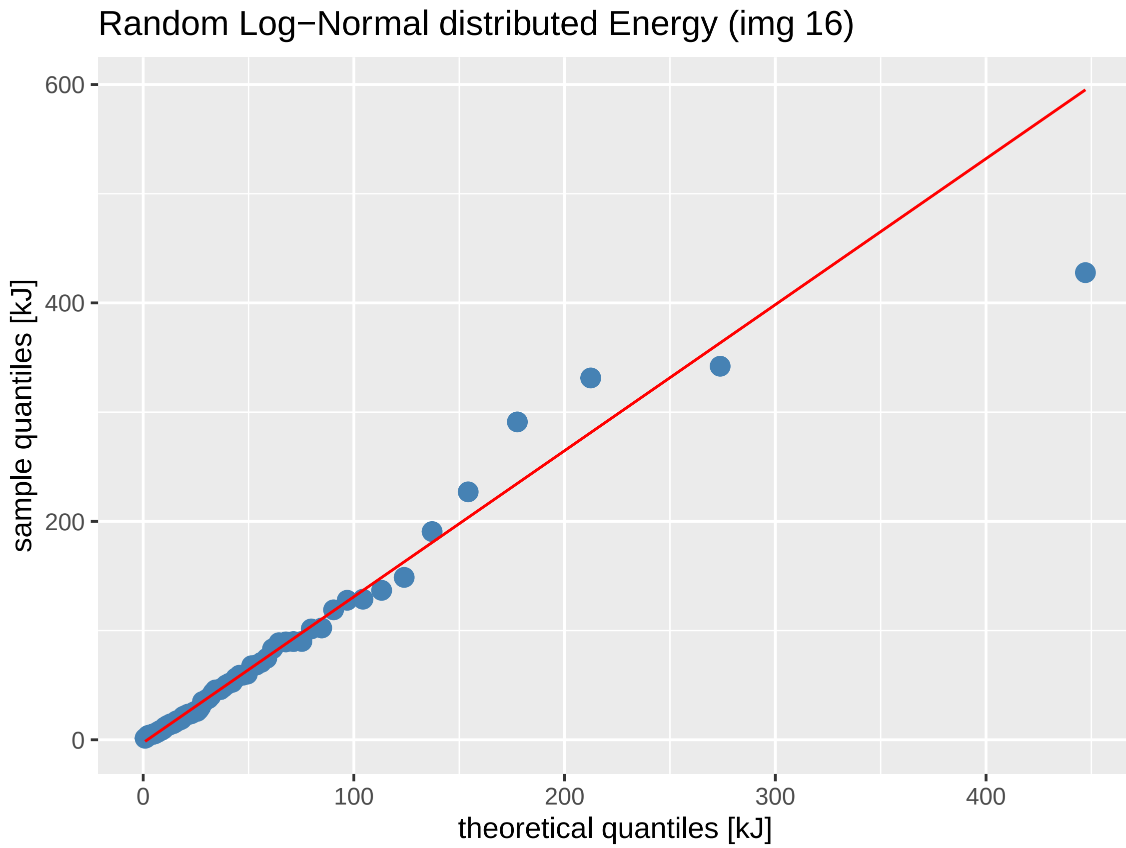

Random Log−Normal distributed Energy (img 12)
800
600
400
200
0
sample quantiles [kJ]
0
100
200
300
400
theoretical quantiles [kJ]

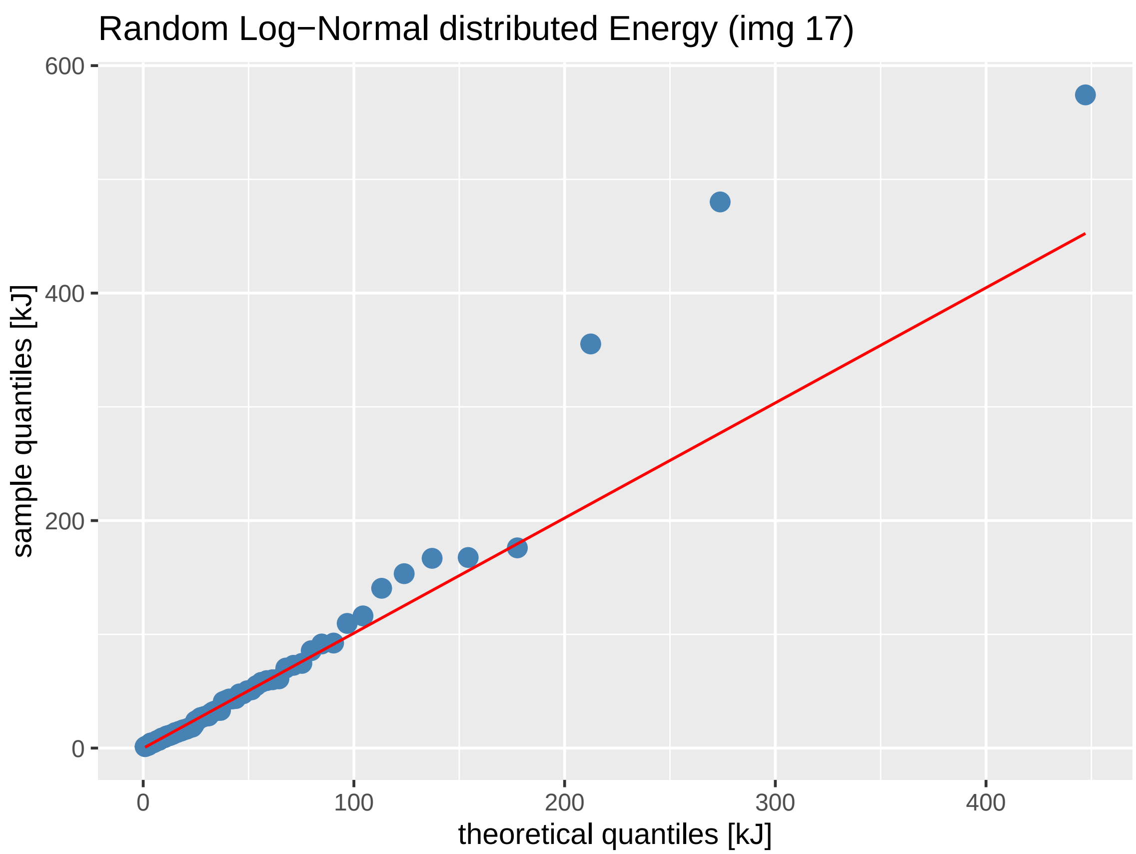


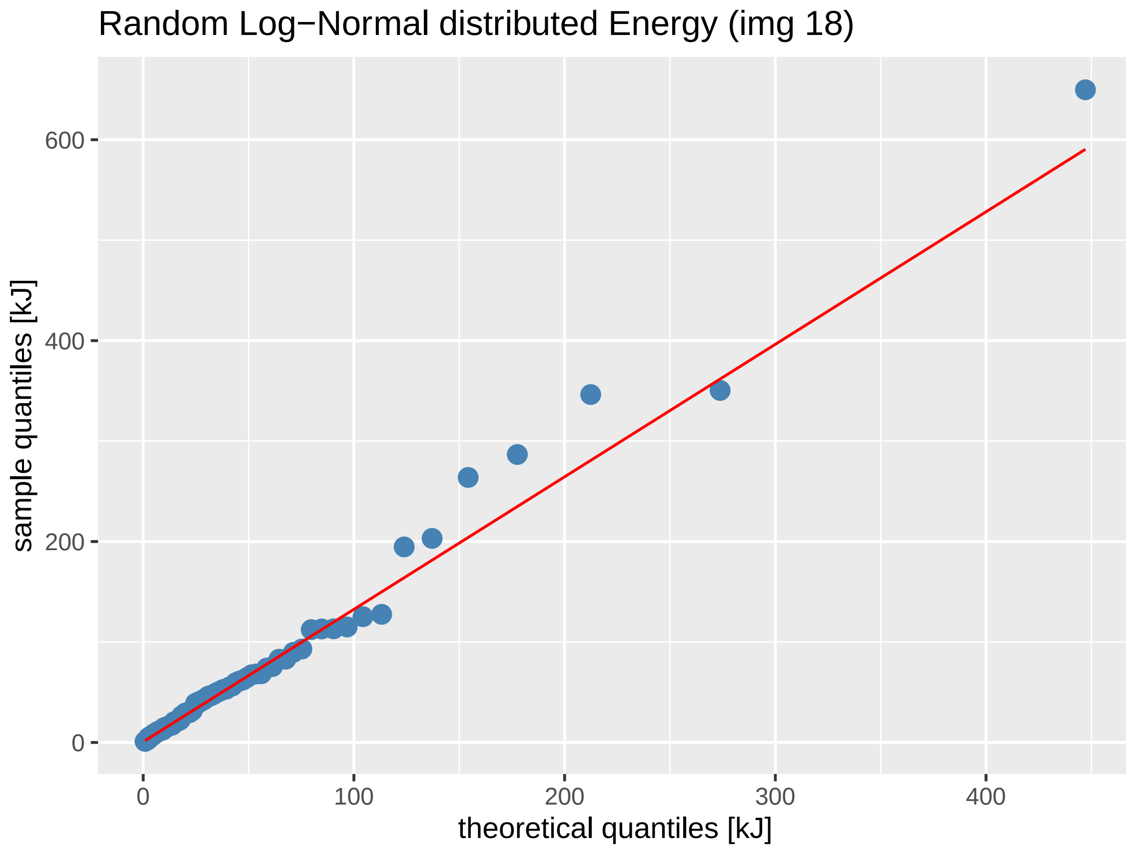
Random Log−Normal distributed Energy (img 13)
sample quantiles [kJ]
theoretical quantiles [kJ]
0
100
200
300
400

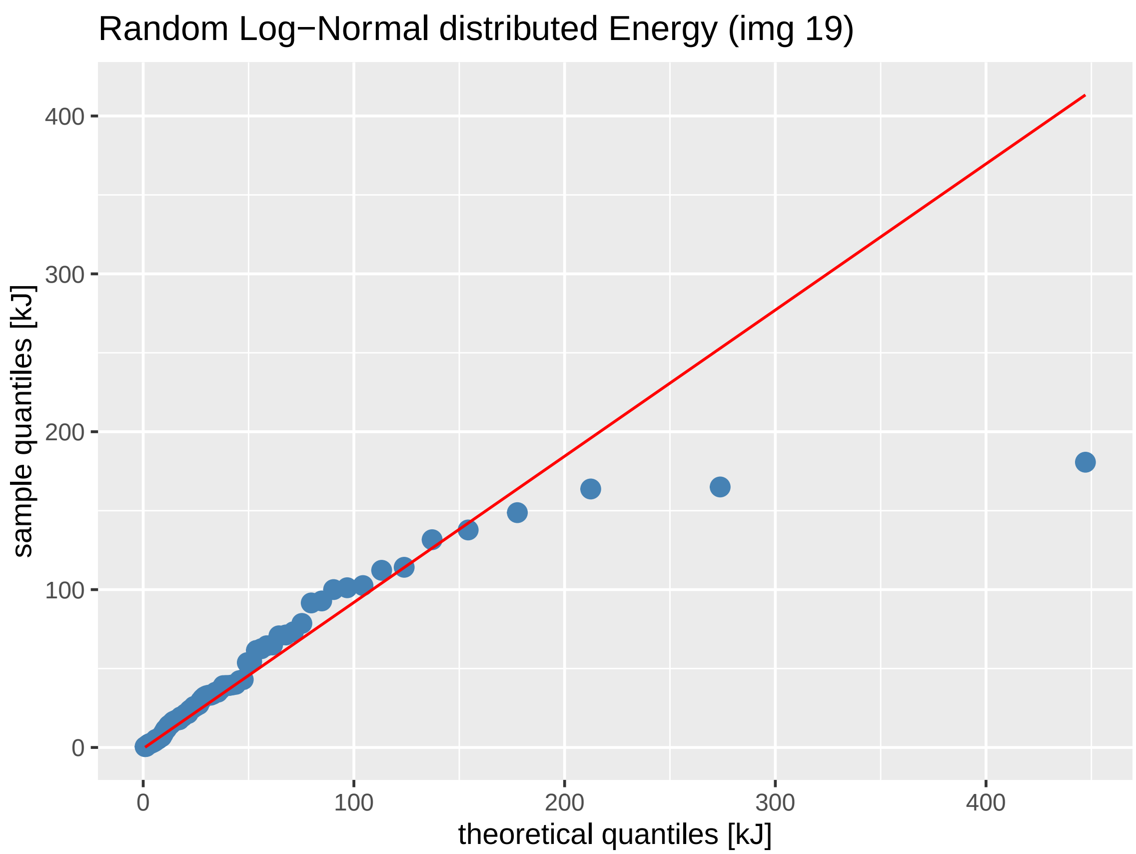

Random Log−Normal distributed Energy (img 14)
sample quantiles [kJ]
theoretical quantiles [kJ]
0
100
200
300
0
100
200
300
400

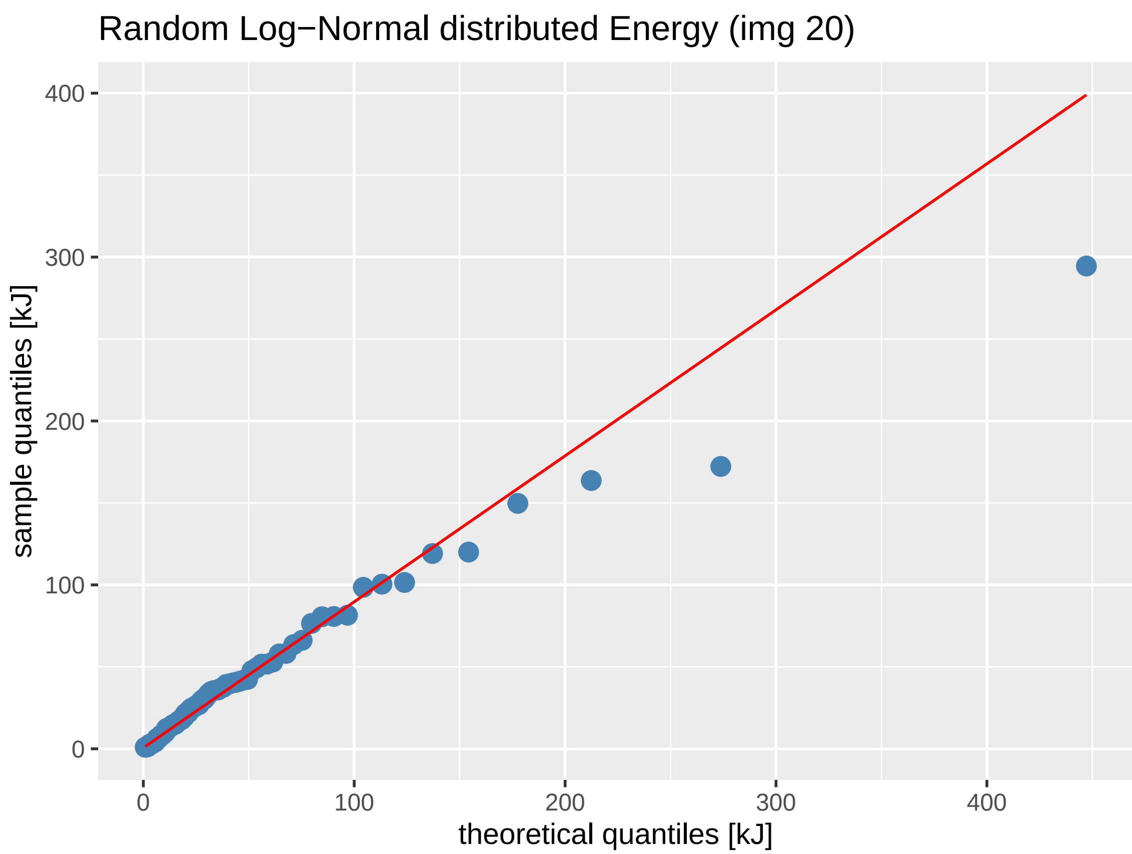

Random Log−Normal distributed Energy (img 15)
2000
1500
1000
500
0
sample quantiles [kJ]
0
100
200
300
400
theoretical quantiles [kJ]


Random Log−Normal distributed Energy (img 16)
sample quantiles [kJ]
theoretical quantiles [kJ]
0
200
400
600
0
100
200
300
400


Random Log−Normal distributed Energy (img 17)
600
400
200
0
sample quantiles [kJ]
0
100
200
300
400
theoretical quantiles [kJ]


Random Log−Normal distributed Energy (img 18)
600
400
200
0
sample quantiles [kJ]
0
100
200
300
400
theoretical quantiles [kJ]


Random Log−Normal distributed Energy (img 19)
sample quantiles [kJ]
theoretical quantiles [kJ]
0
100
200
300
400
0
100
200
300
400



4.5 Probability of a Rockfall in a certain Energy Interval

Assuming to have a log normal distribution, we are now able to calculate the probability $P(500 \leq x \leq 1000|A)$ and $P(x \geq 1000|A)$ of an interval where A is the condition that a rock falls into the net. $P(0 \leq x < 500|A)$ is not interesting since a rock with energy $x < 500$ is never strong enough to break through the net. First we compute the natural logarithm of each value on the rockfall energy set $ln(x_i)$ To be able to calculate the probability, the log normal distribution needs *meanlog* and the *sdlog* of the distribution, which can be calculated from the logarithmized values of the energy data set.

Now we can take the cumulative distribution function of normal distribution $F(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi} * \sigma} e^{-\frac{1}{2}(\frac{t-\mu}{\sigma})} dt$ if we transform each energy values with the natural logarithm. To calculate e.g. $P(x \geq 1000|A)$ we are now able to calculate the area between 0 and $ln(1000)$ with the normal distribution if we take $\mu = meanlog$, $\sigma = sdlog$ and then subtract the value from 1 to get the area after $1000kJ$.

```
#fitDistr(rockfall$energy, nbin = 99, plot = "qq") ~5min
prob_lnorm <- stats::plnorm(c(500, 1000), meanlog = lnrm_fit$estimate[1], sdlog = lnrm_fit$estimate[2],
prob_lnorm_500_to_1000 <- (prob_lnorm[2] - prob_lnorm[1])
prob_lnorm_1000 <- (1 - prob_lnorm[2])
print(paste('P(500 < x < 1000 | A) =', prob_lnorm_500_to_1000))
```

```
## [1] "P(500 < x < 1000 | A) = 0.00325148586454382"
```

```
print(paste('P(1000 < x | A) =', prob_lnorm_1000))
```

```
## [1] "P(1000 < x | A) = 0.000599808701256865"
```

```
day_mass <- 0
```

5. Mass on the Net Distribution

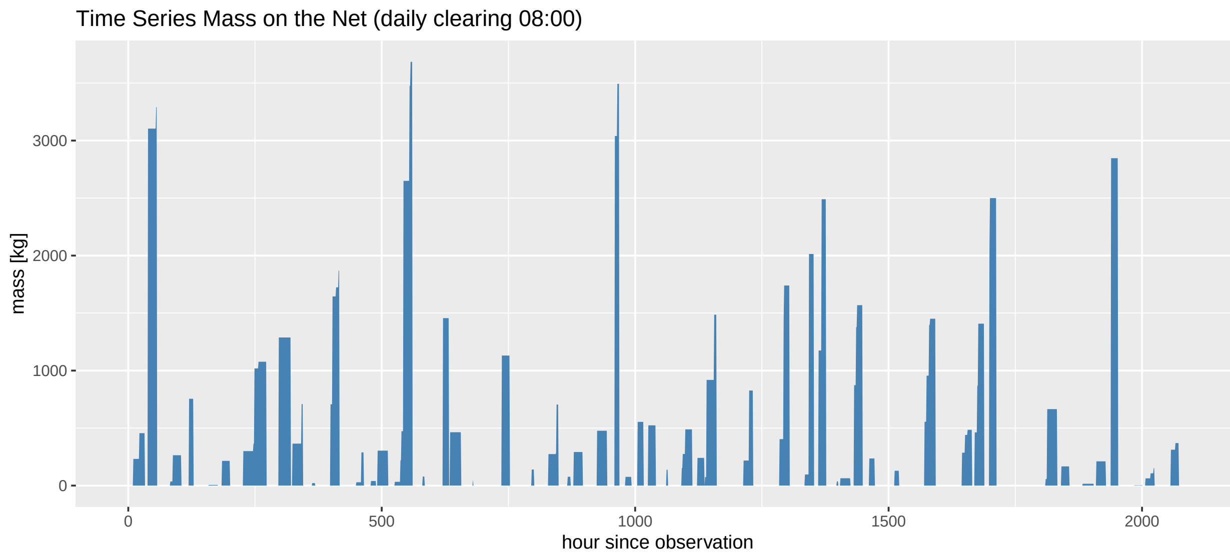
5.1 Time Series Mass in Net Creation

Furthermore, we want to know the probability that the net has rock-weights equals or more than 2000 kg $P(x \geq 2000|A)$. Therefore, a new dataframe must be created with the cumulative weight for each day and hour in the net. We assume, that the net was cleared every day at 08:00.

```
rockfall_ordered <- rockfall
rockfall_ordered$date <- base::as.Date(rockfall$date, format = "%d.%m.%Y")
rockfall_ordered <- rockfall_ordered[order(rockfall_ordered$date),]
num_of_days <- difftime(rockfall_ordered$date[base::nrow(rockfall_ordered)], rockfall_ordered$date[1], u
day_first_observation <- rockfall_ordered$date[1]
day_times <- c("00:00:00", "01:00:00", "02:00:00", "03:00:00", "04:00:00", "05:00:00", "06:00:00", "07:0
clearing_time <- "08:00:00"

#initialize first value of vector to ensure correct data type, must be sliced afterwards
day <- c(day_first_observation)
time <- c("00:00:00")
mass <- c(rockfall_ordered$mass[1])


#loop through every day since the first day of observation
for (i in 0:num_of_days){
  today <- day_first_observation + (i)
  selected_frame_day <- dplyr::filter(rockfall_ordered, rockfall_ordered$date == today)
  #loop through every hour of the day and add each observed weight of a day
  for (hour in day_times){
    if (hour == clearing_time){
      day_mass = 0
    }
    selected_frame_hour <- dplyr::filter(selected_frame_day, selected_frame_day$time == hour)
    if (base::nrow(selected_frame_hour) > 0){
      day_mass = day_mass + base::sum(selected_frame_hour[3]) #sum if the vector contains multiple valu
    }
    day <- c(day, today)
    time <- c(time, hour)
    mass <- c(mass, day_mass)
  }
}
net_weigt <- base::data.frame(day, time, mass)
net_weigt <- net_weigt[-1,] #delete first row from the frame
```

This is the load of the net of each hour since the beginning of the observation.

```
hours_since_observation <- c(1:nrow(net_weigt))
ggplot2::ggplot(data = net_weigt) +
  geom_area(mapping = aes(x = hours_since_observation, y = mass), fill="steelblue") +
  labs(title = "Time Series Mass on the Net (daily clearing 08:00)", x = "hour since observation", y =
```

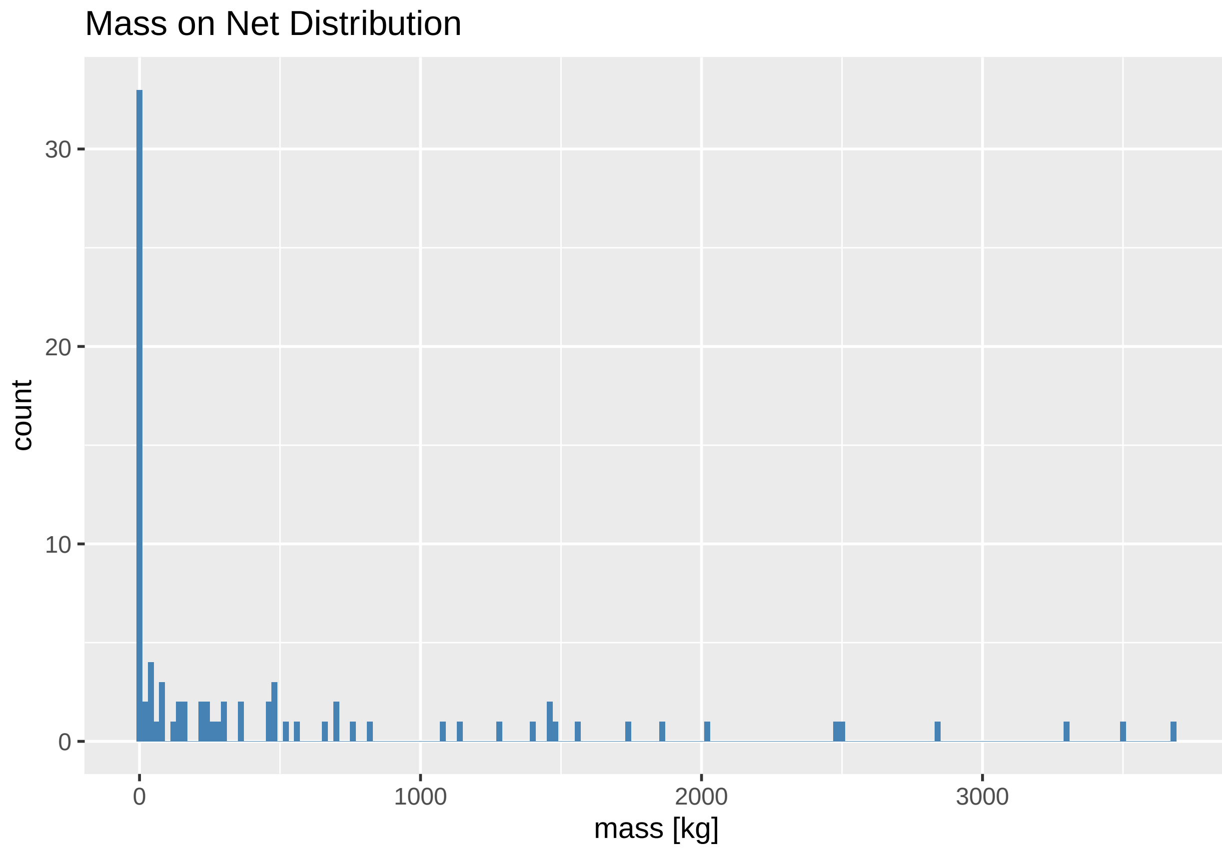


5.2 Mass on the Net Distribution

Now we aim to find a distribution funciton which fits the actual mass data on the net best. In addition, we need the data to be independant. Therefore, to be on the safe side, we just considered the max weight before clearing.

After several tries we did not find out a distribution that fits the weigt on the net well. We decided to try another approach to calculate an appropriate probability that a rock falls through the net.

```
#gets the last cumulative load on the safety net
index_of_clearing_time <- base::match(c(clearing_time), day_times)
if (index_of_clearing_time == 1){
  index_of_clearing_time = length(day_times)
} else {
  index_of_clearing_time = index_of_clearing_time -1
}
cumulative_weight_per_period <- dplyr::filter(net_weigt, time == day_times[(index_of_clearing_time)])

ggplot2::ggplot(data = cumulative_weight_per_period, mapping = aes(x = mass)) +
  geom_histogram(mapping = aes(y = ..count..), binwidth = 20, fill = "steelblue") +
  labs(title = "Mass on Net Distribution", x = 'mass [kg]')
```

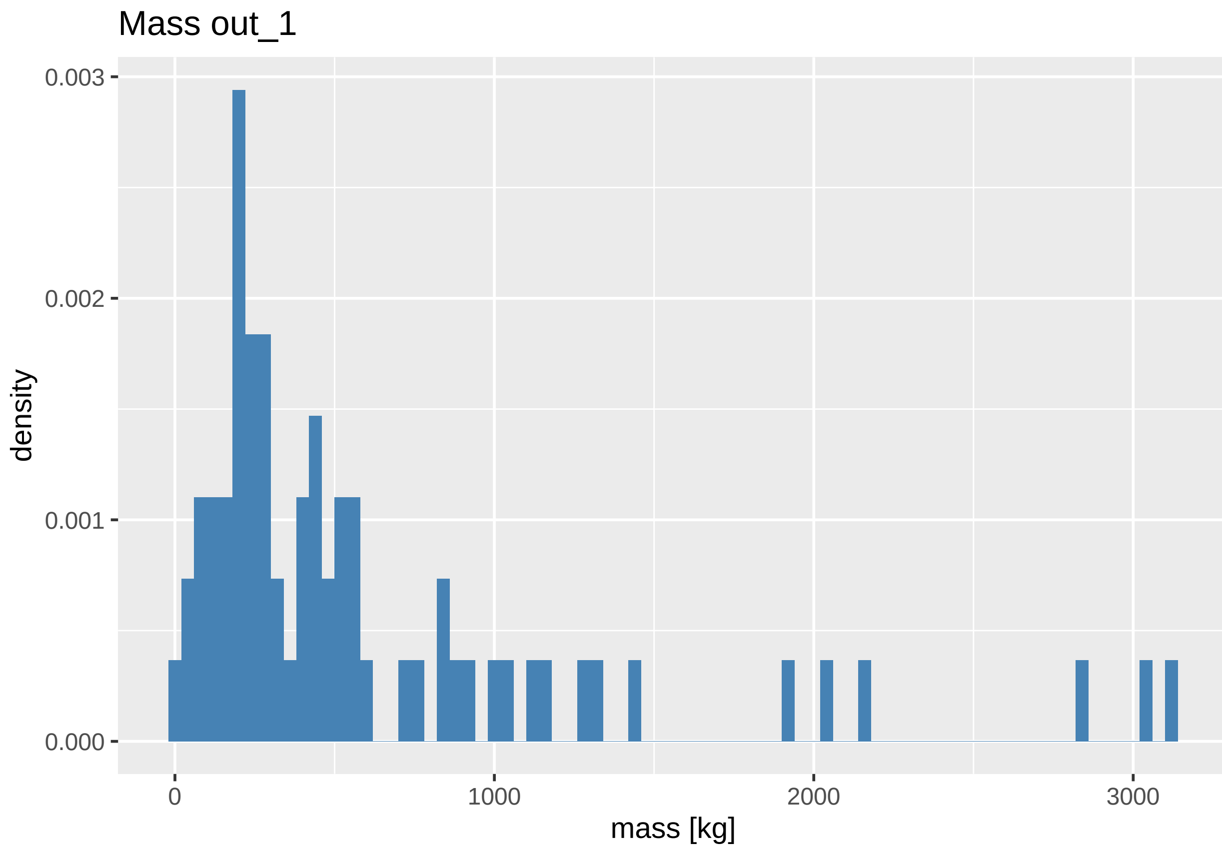


6. Mass & Speed Distribution

6.1 Mass Distribution out_1

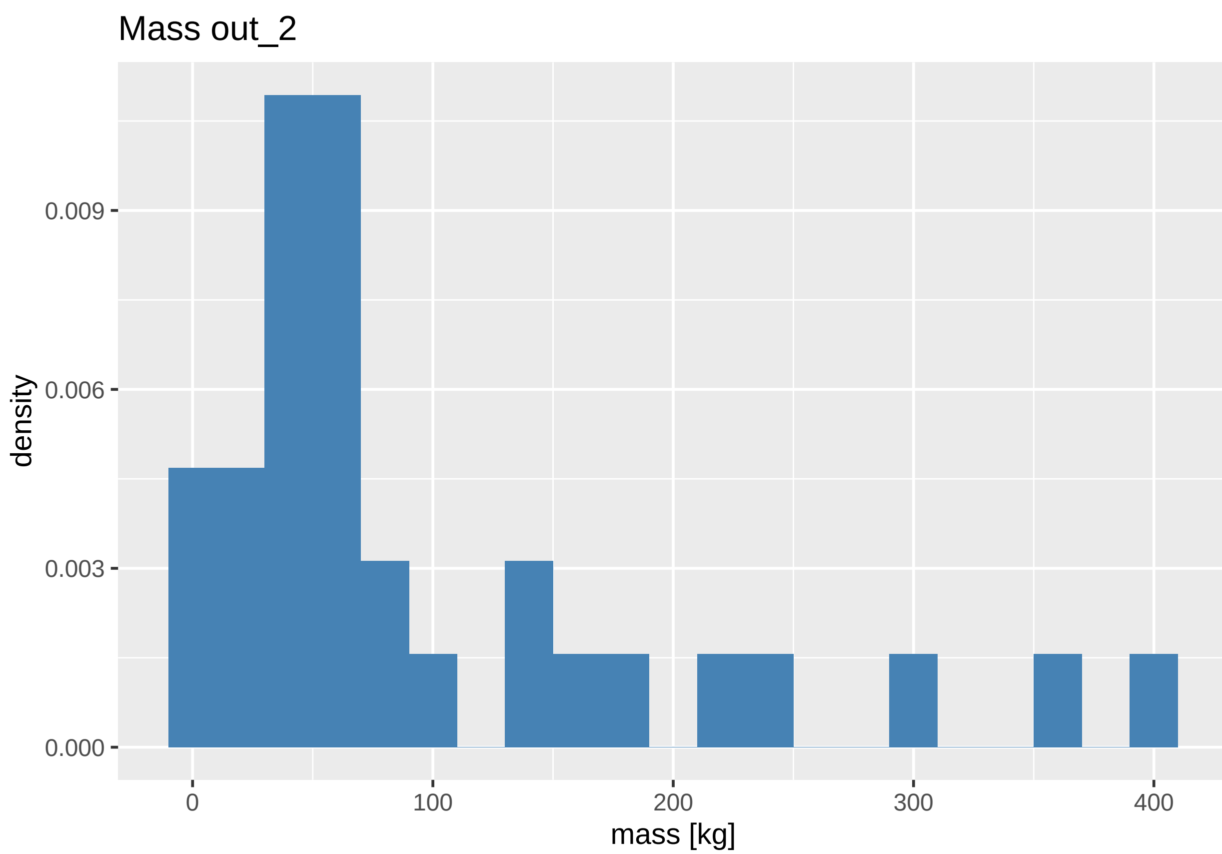
Our next approach is to get the probability with the help of a Monte-Carlo simulation. Therefore, we have to find out the distributions of the weight and speed of a rockfall of each data set `out_1` and `out_2`. In addition, we need the time from an event (rockfall) to the next to set up the simulation.

```
ggplot(out_1, aes(mass)) +
  geom_histogram(aes(y = ..density..), binwidth = 40, fill = "steelblue") +
  labs(title = "Mass out_1", x = "mass [kg]")
```



6.2 Mass Distribution out_2

```
out_2$mass <- base::replace(out_2$mass, out_2$mass == 0, 1) #replace 0 values of the observation to be
ggplot(out_2, aes(mass)) +
  geom_histogram(aes(y = ..density..), binwidth = 20, fill = "steelblue") +
  labs(title = "Mass out_2", x = "mass [kg]")
```

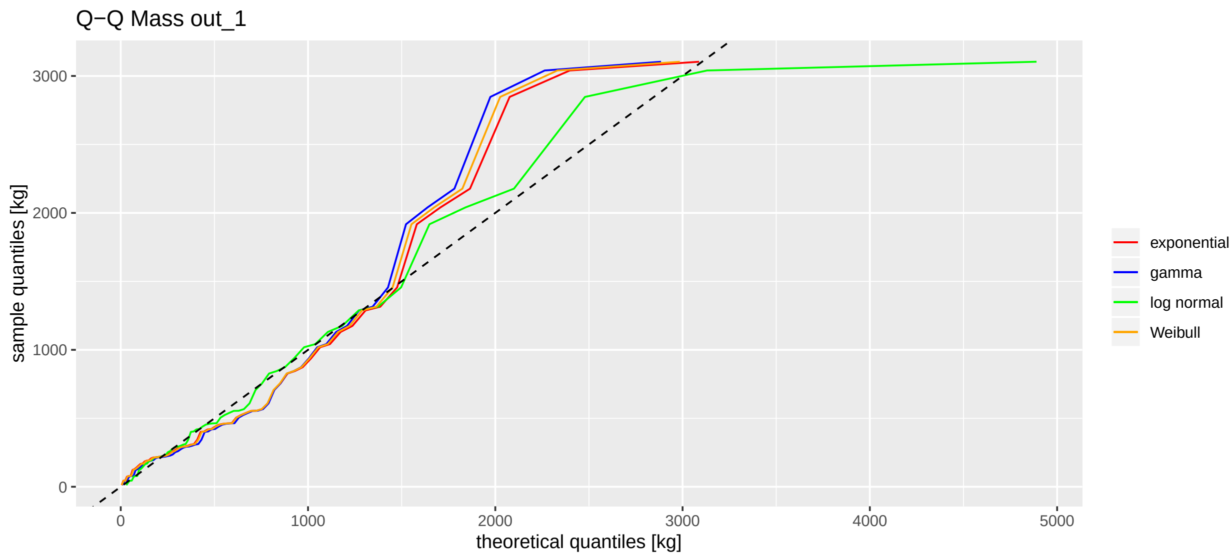


6.3 Mass out_1 Density Function Evaluation

Now we check what distribution fits the mass of `out_1` best. Therefore, we compare exponential, gamma, log-normal and weibull. We can see that the log-normal distribution fits our data best except of the last data point. Even if the last data point is completely not correct estimated, the log normal distribution esitmated the weight higher then the actual weight was, which was estimated more cautiously.

```
e_fit_mass_out_1 <- fitdistr(out_1$mass, "exponential")
g_fit_mass_out_1 <- fitdistr(out_1$mass, "gamma")
lnrm_fit_mass_out_1 <- fitdistr(out_1$mass, "lognormal")
w_fit_mass_out_1 <- fitdistr(out_1$mass, "weibull")

ggplot(out_1, aes(sample = mass)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit_mass_out_1$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qgamma,
          dparams = list(g_fit_mass_out_1$estimate[1], g_fit_mass_out_1$estimate[2]),
          geom = "line",
          aes(color = "gamma")) +
  stat_qq(distribution = qlnorm,
          dparams = list(lnrm_fit_mass_out_1$estimate[1], lnrm_fit_mass_out_1$estimate[2]),
          geom = "line",
          aes(color = "log normal")) +
  stat_qq(distribution = qweibull,
```

```
          dparams = list(w_fit_mass_out_1$estimate[1], w_fit_mass_out_1$estimate[2]),
          geom = "line",
          aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("exponential" = "red",
                                "gamma" = "blue",
                                "log normal" = "green",
                                "Weibull" = "orange"),
                     breaks = c("exponential",
                                "gamma",
                                "log normal",
                                "Weibull")) +
  labs(title = "Q-Q Mass out_1", x = "theoretical quantiles [kg]", y = "sample quantiles [kg]")
```

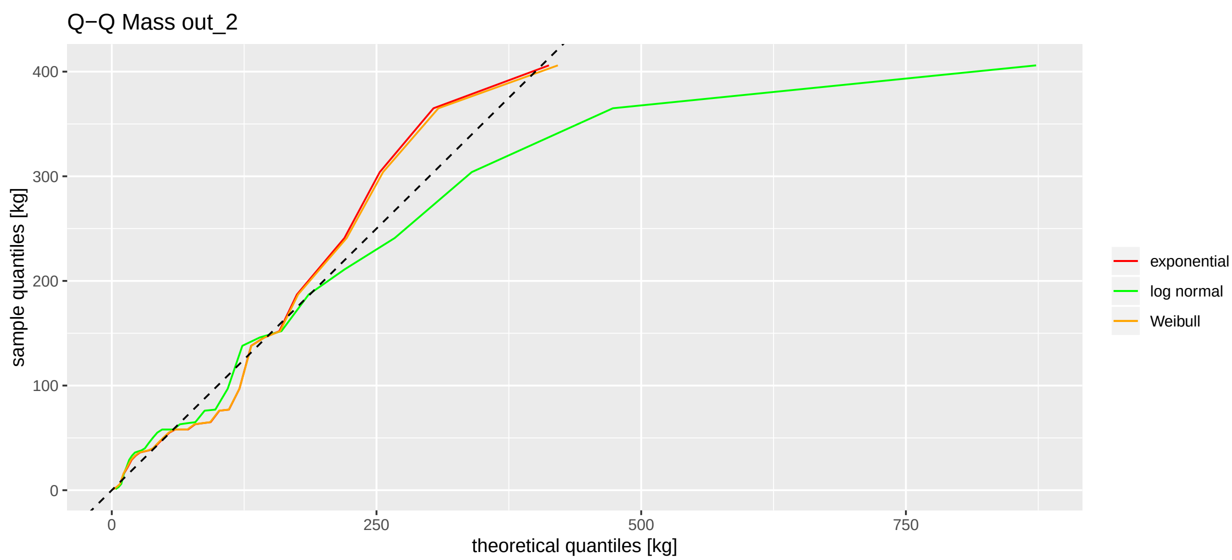


6.4 Mass out__2 Density Function Evaluation

Now we aim to find an appropriate distribution funciton for the mass of `out_2`. Therefore, we compare exponential, lognormal and weibull. In regards to the result of the QQ-plot, we decided to consider the exponential distribution function, which is pretty similar distributed compared to the weibull function.

```
e_fit_mass_out_2 <- fitdistr(out_2$mass, "exponential")
lnrm_fit_mass_out_2 <- fitdistr(out_2$mass, "lognormal")
w_fit_mass_out_2 <- fitdistr(out_2$mass, "weibull")

ggplot(out_2, aes(sample = mass)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit_mass_out_2$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qlnorm,
          dparams = list(lnrm_fit_mass_out_2$estimate[1], lnrm_fit_mass_out_2$estimate[2]),
          geom = "line",
          aes(color = "log normal")) +
  stat_qq(distribution = qweibull,
          dparams = list(w_fit_mass_out_2$estimate[1], w_fit_mass_out_2$estimate[2]),
```

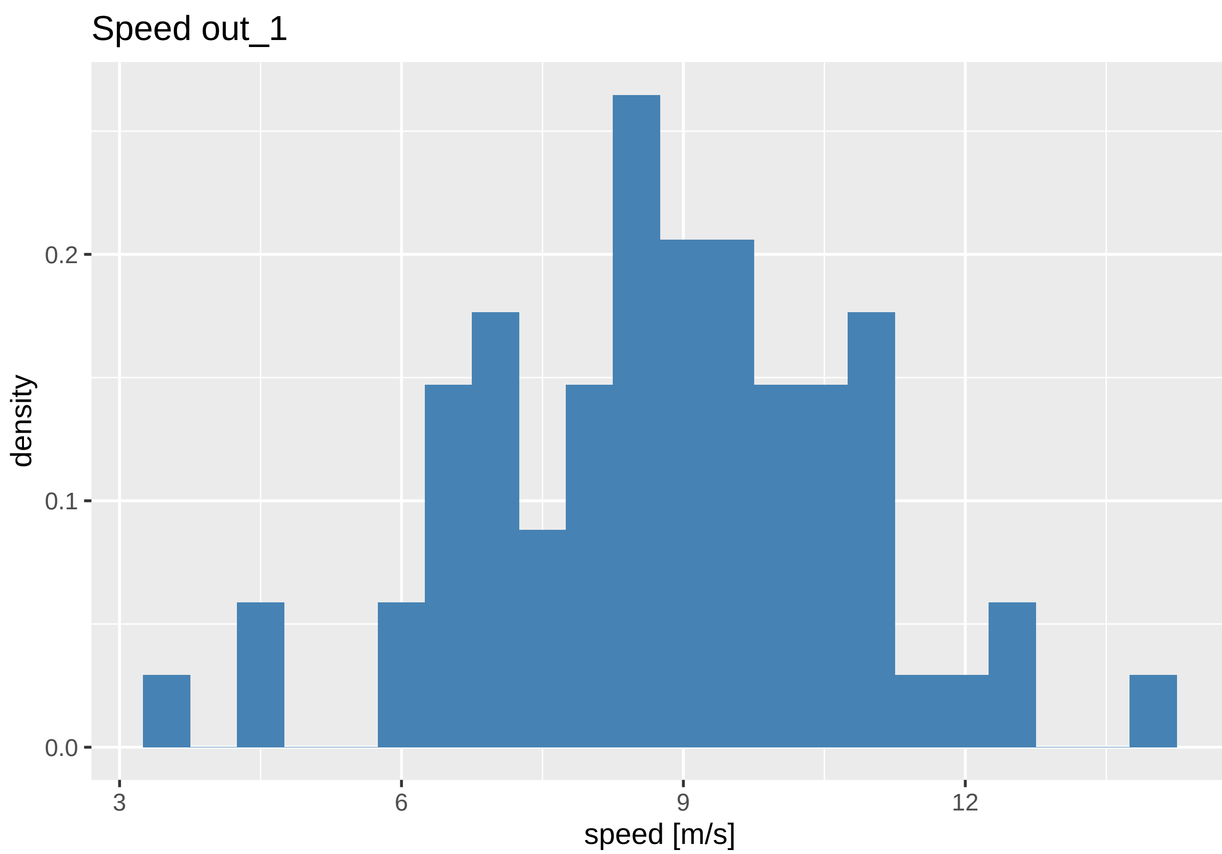
```
          geom = "line",
          aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("exponential" = "red",
                                "gamma" = "blue",
                                "log normal" = "green",
                                "Weibull" = "orange"),
                     breaks = c("exponential",
                                "gamma",
                                "log normal",
                                "Weibull")) +
  labs(title = "Q-Q Mass out_2", x = "theoretical quantiles [kg]", y = "sample quantiles [kg]")
```



6.5 Speed Distribution out_1

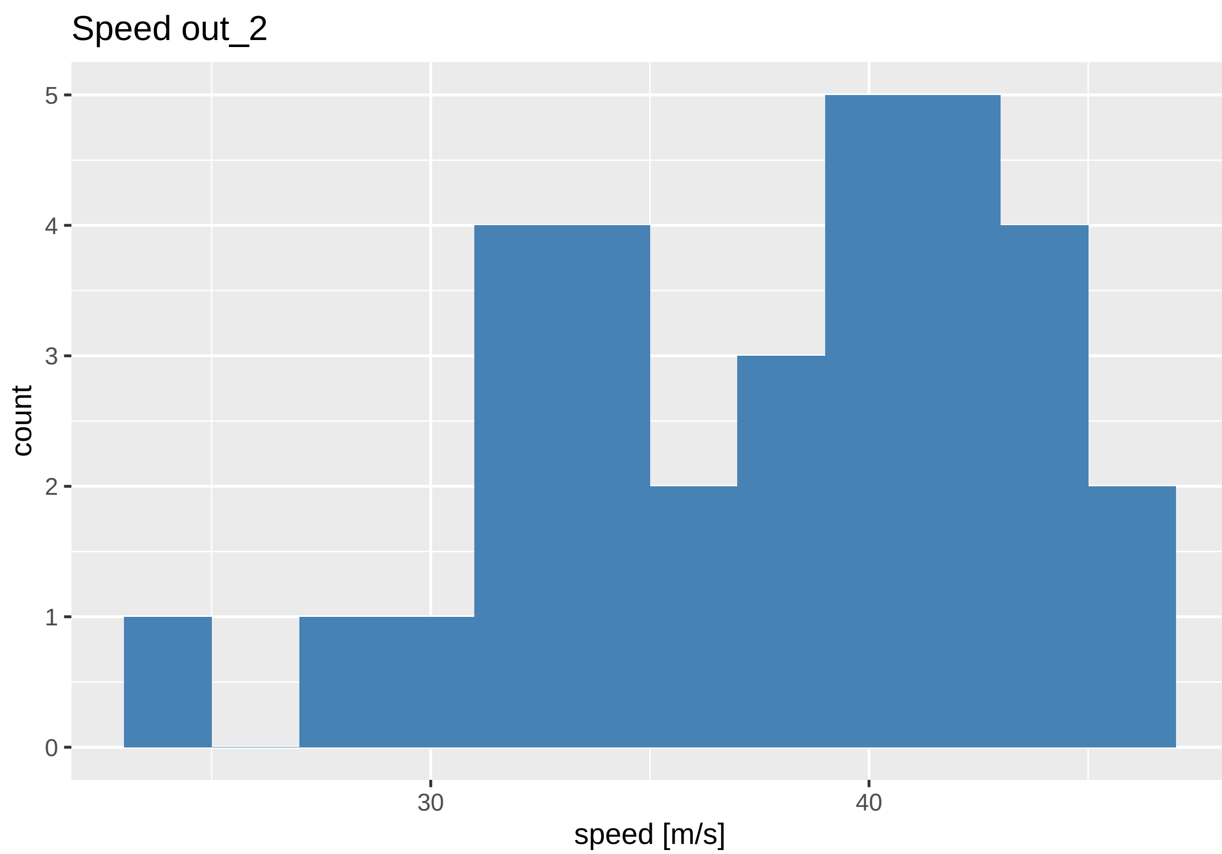
Now we are interested in how the speed of `out_1` and `out_2` is distributed.

```
ggplot(out_1, aes(speed)) +
  geom_histogram(aes(y = ..density..), binwidth = 0.5, fill = "steelblue") +
  labs(title = "Speed out_1", x = "speed [m/s]")
```



6.6 Speed Distribution out_2

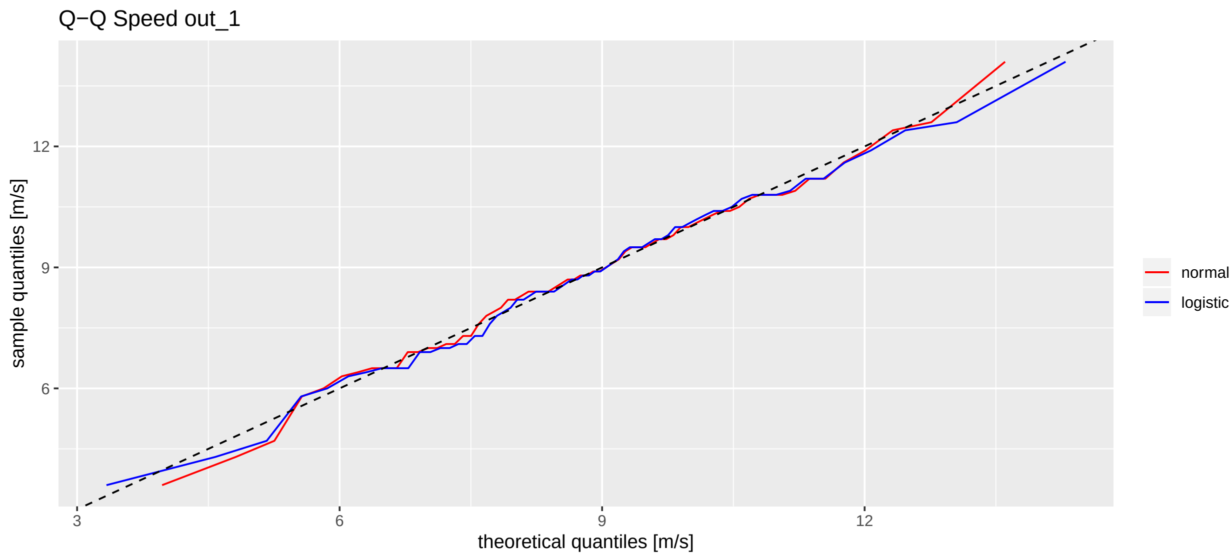
```
ggplot(out_2, aes(speed)) +
  geom_histogram(aes(y = ..count..), binwidth = 2, fill = "steelblue") +
  labs(title = "Speed out_2", x = "speed [m/s]")
```



6.7 Speed out_1 Density Function Evaluation

We can assume to have a normal distribution in regards to the histogram of `out_1`. For this reason, we just check, if the speed data fits a normal or logistic distribution. In the QQ-plot we can see that both, normal and logistic distribution fits the speed of `out_1` very well. We continue with the assumtion of having a normal distribution.

```
nrm_fit_speed_out_1 <- fitdistr(out_1$speed, "normal")
logis_fit_speed_out_1 <- fitdistr(out_1$speed, "logistic")

ggplot(out_1, aes(sample = speed)) +
  stat_qq(distribution = qnorm,
          dparams = list(nrm_fit_speed_out_1$estimate[1], nrm_fit_speed_out_1$estimate[2]),
          geom = "line",
          aes(color = "normal")) +
  stat_qq(distribution = qlogis,
          dparams = list(logis_fit_speed_out_1$estimate[1], logis_fit_speed_out_1$estimate[2]),
          geom = "line",
          aes(color = "logistic")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("normal" = "red",
                                "logistic" = "blue"),
                     breaks = c("normal",
                                "logistic")) +
  labs(title = "Q-Q Speed out_1", x = "theoretical quantiles [m/s]", y = "sample quantiles [m/s]")
```



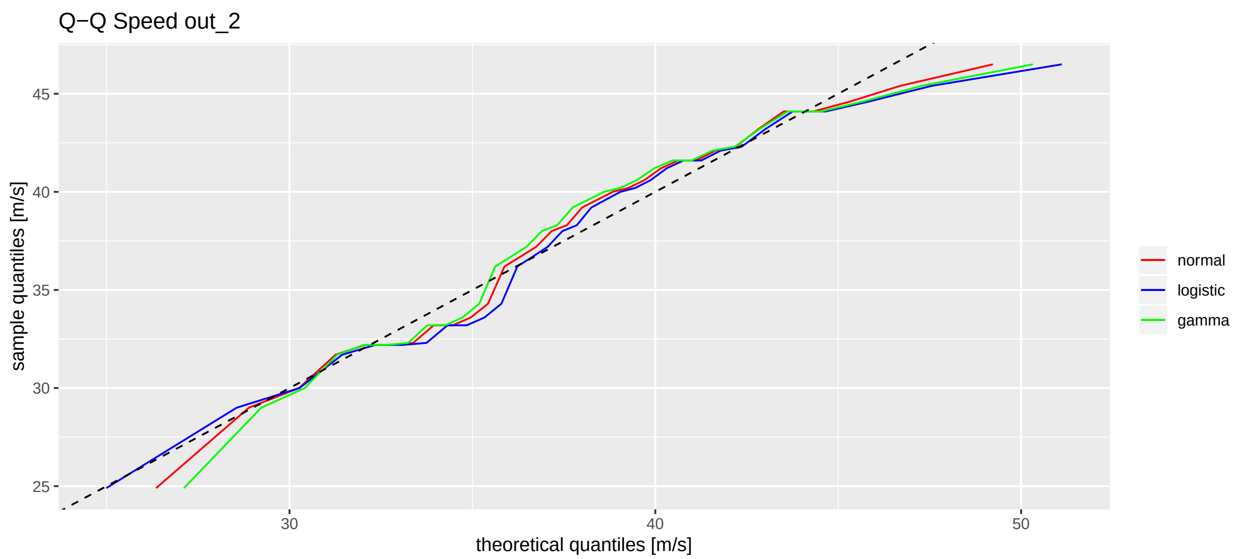
6.8 Speed out_2 Density Function Evaluation

To estimate the speed of the sample `out_2`, we set up another QQ-plot with possible distributions like normal, logistic and gamma distribution.

Here, it is not very clear which distribution fits best or even if a selected distribution is appropriate. We assume that our speed data of `out_2` could be normal distributed. Therefore, we test, if with the given μ and σ a normal distribution is possible.

```
nrm_fit_speed_out_2 <- fitdistr(out_2$speed, "normal")
logis_fit_speed_out_2 <- fitdistr(out_2$speed, "logistic")
gamma_fit_speed_out_2 <- fitdistr(out_2$speed, "gamma")

ggplot(out_2, aes(sample = speed)) +
  stat_qq(distribution = qnorm,
          dparams = list(nrm_fit_speed_out_2$estimate[1], nrm_fit_speed_out_2$estimate[2]),
          geom = "line",
          aes(color = "normal")) +
  stat_qq(distribution = qlogis,
          dparams = list(logis_fit_speed_out_2$estimate[1], logis_fit_speed_out_2$estimate[2]),
          geom = "line",
          aes(color = "logistic")) +
  stat_qq(distribution = qgamma,
          dparams = list(gamma_fit_speed_out_2$estimate[1], gamma_fit_speed_out_2$estimate[2]),
          geom = "line",
          aes(color = "gamma")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("normal" = "red",
                                "logistic" = "blue",
                                "chi-squared" = "purple",
                                "gamma" = "green"),
                     breaks = c("normal",
                                "logistic",
                                "chi-squared",
                                "gamma")) +
```

```
labs(title = "Q-Q Speed out_2", x = "theoretical quantiles [m/s]", y = "sample quantiles [m/s]")
```



6.9 Speed out_2 Gaussian Distribution Validation

7. Investigation of the amount of rockfalls

We know the date and the time of any rockfalls. Out of that we can calculate what is the average amount of rockfalls in a day or even in a hour. First we need a proper format for the time calculation. We have now a accurate timestamp vor every rockfall. Than we can subtract the previous Timestamp from every timestampt to get the time between the rockfalls.

```
# Zone 1
out_1
```

```
##          date     time mass speed     energy
## 1  01.01.2019 09:00:00  194   8.4   6.844320
## 2  01.01.2019 21:00:00  224   8.8   8.673280
## 3  02.01.2019 14:00:00 3104   9.2 131.361280
## 4  04.01.2019 15:00:00  228   8.0   7.296000
## 5  05.01.2019 23:00:00  755   7.0  18.497500
## 6  08.01.2019 16:00:00  215   6.5   4.541875
## 7  10.01.2019 10:00:00  300   7.9   9.361500
## 8  11.01.2019 08:00:00 1019  10.4  55.107520
## 9  13.01.2019 08:00:00 1288  10.8  75.116160
## 10 15.01.2019 05:00:00  344  10.0  17.200000
## 11 17.01.2019 14:00:00  707   6.0  12.726000
## 12 17.01.2019 18:00:00  938  10.2  48.794760
## 13 18.01.2019 01:00:00   79   6.9   1.880595
## 14 20.01.2019 03:00:00  260  11.2  16.307200
## 15 23.01.2019 08:00:00  220   9.1   9.109100
## 16 23.01.2019 10:00:00  253  10.1  12.904265
## 17 23.01.2019 14:00:00 2177   7.6  62.871760
## 18 24.01.2019 02:00:00  827   8.5  29.875375
## 19 24.01.2019 04:00:00  208  11.2  13.045760
## 20 25.01.2019 04:00:00   79   9.4   3.490220
## 21 26.01.2019 20:00:00 1456   7.1  36.698480
```

```
## 22 27.01.2019 10:00:00  464   7.8   14.114880
## 23 31.01.2019 16:00:00 1131   6.4   23.162880
## 24 03.02.2019 03:00:00  140  12.6   11.113200
## 25 04.02.2019 12:00:00  274  10.3   14.534330
## 26 05.02.2019 03:00:00   12   8.8    0.464640
## 27 05.02.2019 04:00:00  419  10.9   24.890695
## 28 06.02.2019 14:00:00  292   8.4   10.301760
## 29 08.02.2019 12:00:00  169   6.5    3.570125
## 30 08.02.2019 12:00:00  308   9.0   12.474000
## 31 09.02.2019 23:00:00 3040  10.4  164.403200
## 32 10.02.2019 04:00:00  454   7.1   11.443070
## 33 11.02.2019 20:00:00  554   8.7   20.966130
## 34 12.02.2019 17:00:00  524   4.3    4.844380
## 35 15.02.2019 13:00:00  123   9.7    5.786535
## 36 15.02.2019 18:00:00  214  10.8   12.480480
## 37 17.02.2019 09:00:00   74   9.7    3.481330
## 38 17.02.2019 12:00:00  845   4.7    9.333025
## 39 18.02.2019 03:00:00  567   8.2   19.062540
## 40 20.02.2019 13:00:00  218   7.3    5.808610
## 41 21.02.2019 00:00:00  609  12.4   46.819920
## 42 23.02.2019 12:00:00  404  10.7   23.126980
## 43 23.02.2019 20:00:00 1042  10.5   57.440250
## 44 23.02.2019 21:00:00  294   9.8   14.117880
## 45 25.02.2019 22:00:00 1917   9.5   86.504625
## 46 26.02.2019 17:00:00 1175   9.6   54.144000
## 47 26.02.2019 23:00:00 1315   3.6    8.521200
## 48 01.03.2019 15:00:00  873   6.3   17.324685
## 49 01.03.2019 19:00:00  505   8.6   18.674900
## 50 01.03.2019 21:00:00  191  10.8   11.139120
## 51 02.03.2019 21:00:00  236   8.7    8.931420
## 52 04.03.2019 23:00:00  129   8.3    4.443405
## 53 07.03.2019 10:00:00  555   8.2   18.659100
## 54 07.03.2019 14:00:00  401   9.5   18.095125
## 55 07.03.2019 19:00:00  440   9.5   19.855000
## 56 10.03.2019 12:00:00  286   8.4   10.090080
## 57 10.03.2019 18:00:00  154  11.6   10.361120
## 58 11.03.2019 13:00:00  463   7.3   12.336635
## 59 11.03.2019 20:00:00  539  14.1   53.579295
## 60 12.03.2019 18:00:00 2040  11.9  144.442200
## 61 12.03.2019 19:00:00  460   6.9   10.950300
## 62 17.03.2019 12:00:00  185   6.5    3.908125
## 63 17.03.2019 12:00:00  422  10.0   21.100000
## 64 18.03.2019 16:00:00  167   8.9    6.614035
## 65 22.03.2019 18:00:00 2847   7.0   69.751500
## 66 26.03.2019 00:00:00   44   8.9    1.742620
## 67 26.03.2019 06:00:00   45   8.4    1.587600
## 68 27.03.2019 16:00:00  312   5.8    5.247840
```

```r
zone_1_Time <- out_1 %>%
  transmute(dateTime = as.POSIXct(paste(date, time), format = "%d.%m.%Y %H:%M:%S"))

zone_1_Time
```

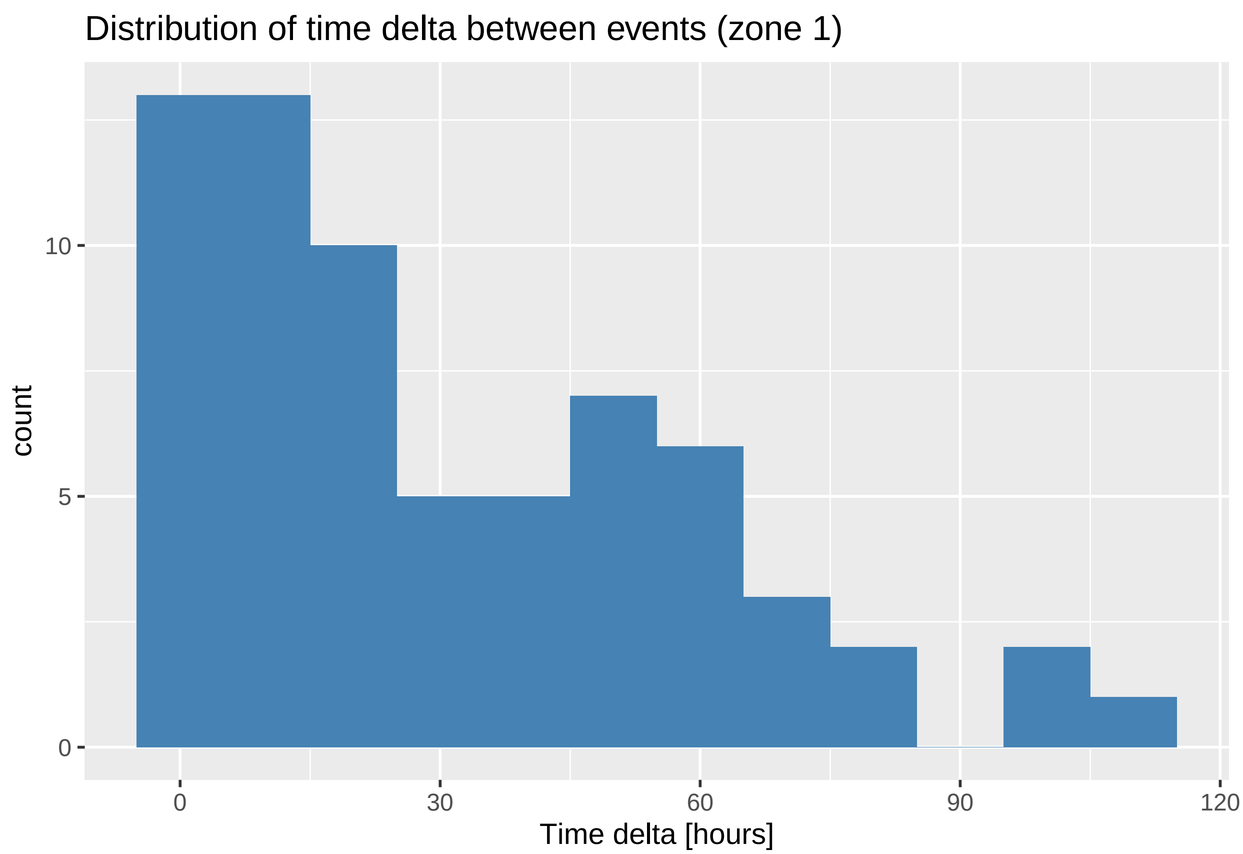
```
##                 dateTime
## 1  2019-01-01 09:00:00
```

```
## 2  2019-01-01 21:00:00
## 3  2019-01-02 14:00:00
## 4  2019-01-04 15:00:00
## 5  2019-01-05 23:00:00
## 6  2019-01-08 16:00:00
## 7  2019-01-10 10:00:00
## 8  2019-01-11 08:00:00
## 9  2019-01-13 08:00:00
## 10 2019-01-15 05:00:00
## 11 2019-01-17 14:00:00
## 12 2019-01-17 18:00:00
## 13 2019-01-18 01:00:00
## 14 2019-01-20 03:00:00
## 15 2019-01-23 08:00:00
## 16 2019-01-23 10:00:00
## 17 2019-01-23 14:00:00
## 18 2019-01-24 02:00:00
## 19 2019-01-24 04:00:00
## 20 2019-01-25 04:00:00
## 21 2019-01-26 20:00:00
## 22 2019-01-27 10:00:00
## 23 2019-01-31 16:00:00
## 24 2019-02-03 03:00:00
## 25 2019-02-04 12:00:00
## 26 2019-02-05 03:00:00
## 27 2019-02-05 04:00:00
## 28 2019-02-06 14:00:00
## 29 2019-02-08 12:00:00
## 30 2019-02-08 12:00:00
## 31 2019-02-09 23:00:00
## 32 2019-02-10 04:00:00
## 33 2019-02-11 20:00:00
## 34 2019-02-12 17:00:00
## 35 2019-02-15 13:00:00
## 36 2019-02-15 18:00:00
## 37 2019-02-17 09:00:00
## 38 2019-02-17 12:00:00
## 39 2019-02-18 03:00:00
## 40 2019-02-20 13:00:00
## 41 2019-02-21 00:00:00
## 42 2019-02-23 12:00:00
## 43 2019-02-23 20:00:00
## 44 2019-02-23 21:00:00
## 45 2019-02-25 22:00:00
## 46 2019-02-26 17:00:00
## 47 2019-02-26 23:00:00
## 48 2019-03-01 15:00:00
## 49 2019-03-01 19:00:00
## 50 2019-03-01 21:00:00
## 51 2019-03-02 21:00:00
## 52 2019-03-04 23:00:00
## 53 2019-03-07 10:00:00
## 54 2019-03-07 14:00:00
## 55 2019-03-07 19:00:00
```

```
## 56 2019-03-10 12:00:00
## 57 2019-03-10 18:00:00
## 58 2019-03-11 13:00:00
## 59 2019-03-11 20:00:00
## 60 2019-03-12 18:00:00
## 61 2019-03-12 19:00:00
## 62 2019-03-17 12:00:00
## 63 2019-03-17 12:00:00
## 64 2019-03-18 16:00:00
## 65 2019-03-22 18:00:00
## 66 2019-03-26 00:00:00
## 67 2019-03-26 06:00:00
## 68 2019-03-27 16:00:00
```

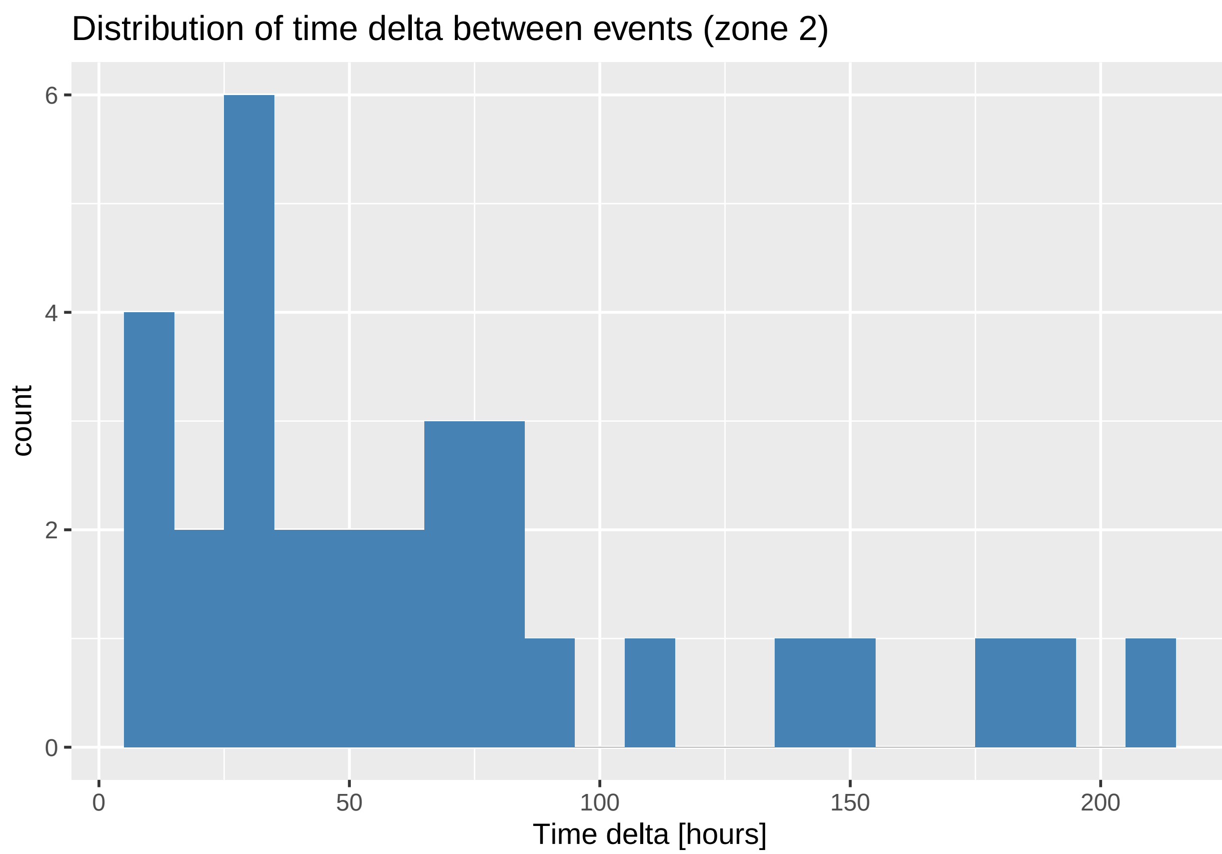
```
zone_1_TimeDiff <- zone_1_Time %>%
  mutate(
    diff_secs = as.numeric(dateTime - lag(dateTime))+1,
    diff_mins =  as.numeric(diff_secs)/60,
    diff_hours =  as.numeric(diff_secs)/3600,
    diff_days =  as.numeric(diff_secs)/ 86400)

#Zone 2
zone_2_Time <- out_2 %>%
  transmute(dateTime = as.POSIXct(paste(date, time), format = "%d.%m.%Y %H:%M:%S"))


zone_2_TimeDiff <- zone_2_Time %>%
  mutate(
    diff_hours = as.numeric (dateTime - lag(dateTime)) +1,
    diff_mins = diff_hours * 60,
    diff_secs = diff_hours * 3600,
     diff_days = diff_secs/ 86400)

zone_1_TimeDiff %>%
  ggplot() +
  geom_histogram(aes(x = diff_hours), binwidth = 10, fill = "steelblue")+
  labs(title='Distribution of time delta between events (zone 1)', x = 'Time delta [hours]')
```

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```




```
zone_2_TimeDiff %>%
  ggplot() +
  geom_histogram(aes(x = diff_hours), binwidth = 10, fill = "steelblue")+
  labs(title='Distribution of time delta between events (zone 2)', x = 'Time delta [hours]')
```

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```




```
head(zone_1_TimeDiff)
```

```
##              dateTime diff_secs diff_mins diff_hours diff_days
## 1 2019-01-01 09:00:00        NA        NA         NA        NA
## 2 2019-01-01 21:00:00     43201  720.0167   12.00028 0.5000116
## 3 2019-01-02 14:00:00     61201 1020.0167   17.00028 0.7083449
## 4 2019-01-04 15:00:00    176401 2940.0167   49.00028 2.0416782
## 5 2019-01-05 23:00:00    115201 1920.0167   32.00028 1.3333449
## 6 2019-01-08 16:00:00    234001 3900.0167   65.00028 2.7083449
```

```
head(zone_2_TimeDiff)
```

```
##              dateTime diff_hours diff_mins diff_secs diff_days
## 1 2019-01-01 09:00:00         NA        NA        NA        NA
## 2 2019-01-03 06:00:00         46      2760    165600 1.9166667
## 3 2019-01-04 10:00:00         29      1740    104400 1.2083333
## 4 2019-01-07 14:00:00         77      4620    277200 3.2083333
## 5 2019-01-11 06:00:00         89      5340    320400 3.7083333
## 6 2019-01-11 16:00:00         11       660     39600 0.4583333
```

```
#Der erste Wert muss gelöscht werden. Denn aus der Berechnung der Zeitdiffertenz ergibt sich die Annham

sum(is.na(zone_1_TimeDiff))
```

```
## [1] 4
```

```
#hat 1 NA Wert
sum(is.na(zone_2_TimeDiff))
```

```
## [1] 4
```

```r
#hat 1 NA Wert

zone_1_TimeDiff <- na.omit(zone_1_TimeDiff)
zone_2_TimeDiff <- na.omit(zone_2_TimeDiff)

sum(is.na(zone_1_TimeDiff))
```

```
## [1] 0
```

```r
#hat 0 NA Wert
sum(is.na(zone_2_TimeDiff))
```

```
## [1] 0
```

```r
#hat 0 NA Wert

#Ergebniss :
# - Eine Merkmalsausprägung musste gelöscht werden, das nun nur noch der Zeitunterschied untersucct wi

zone_1_TimeDiff
```

```
##                 dateTime diff_secs    diff_mins   diff_hours    diff_days
## 2  2019-01-01 21:00:00     43201 7.200167e+02 1.200028e+01 5.000116e-01
## 3  2019-01-02 14:00:00     61201 1.020017e+03 1.700028e+01 7.083449e-01
## 4  2019-01-04 15:00:00    176401 2.940017e+03 4.900028e+01 2.041678e+00
## 5  2019-01-05 23:00:00    115201 1.920017e+03 3.200028e+01 1.333345e+00
## 6  2019-01-08 16:00:00    234001 3.900017e+03 6.500028e+01 2.708345e+00
## 7  2019-01-10 10:00:00    151201 2.520017e+03 4.200028e+01 1.750012e+00
## 8  2019-01-11 08:00:00     79201 1.320017e+03 2.200028e+01 9.166782e-01
## 9  2019-01-13 08:00:00    172801 2.880017e+03 4.800028e+01 2.000012e+00
## 10 2019-01-15 05:00:00    162001 2.700017e+03 4.500028e+01 1.875012e+00
## 11 2019-01-17 14:00:00    205201 3.420017e+03 5.700028e+01 2.375012e+00
## 12 2019-01-17 18:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 13 2019-01-18 01:00:00     25201 4.200167e+02 7.000278e+00 2.916782e-01
## 14 2019-01-20 03:00:00    180001 3.000017e+03 5.000028e+01 2.083345e+00
## 15 2019-01-23 08:00:00    277201 4.620017e+03 7.700028e+01 3.208345e+00
## 16 2019-01-23 10:00:00      7201 1.200167e+02 2.000278e+00 8.334491e-02
## 17 2019-01-23 14:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 18 2019-01-24 02:00:00     43201 7.200167e+02 1.200028e+01 5.000116e-01
## 19 2019-01-24 04:00:00      7201 1.200167e+02 2.000278e+00 8.334491e-02
## 20 2019-01-25 04:00:00     86401 1.440017e+03 2.400028e+01 1.000012e+00
## 21 2019-01-26 20:00:00    144001 2.400017e+03 4.000028e+01 1.666678e+00
## 22 2019-01-27 10:00:00     50401 8.400167e+02 1.400028e+01 5.833449e-01
## 23 2019-01-31 16:00:00    367201 6.120017e+03 1.020003e+02 4.250012e+00
## 24 2019-02-03 03:00:00    212401 3.540017e+03 5.900028e+01 2.458345e+00
## 25 2019-02-04 12:00:00    118801 1.980017e+03 3.300028e+01 1.375012e+00
## 26 2019-02-05 03:00:00     54001 9.000167e+02 1.500028e+01 6.250116e-01
## 27 2019-02-05 04:00:00      3601 6.001667e+01 1.000278e+00 4.167824e-02
## 28 2019-02-06 14:00:00    122401 2.040017e+03 3.400028e+01 1.416678e+00
## 29 2019-02-08 12:00:00    165601 2.760017e+03 4.600028e+01 1.916678e+00
## 30 2019-02-08 12:00:00         1 1.666667e-02 2.777778e-04 1.157407e-05
## 31 2019-02-09 23:00:00    126001 2.100017e+03 3.500028e+01 1.458345e+00
## 32 2019-02-10 04:00:00     18001 3.000167e+02 5.000278e+00 2.083449e-01
## 33 2019-02-11 20:00:00    144001 2.400017e+03 4.000028e+01 1.666678e+00
```

```
## 34 2019-02-12 17:00:00     75601 1.260017e+03 2.100028e+01 8.750116e-01
## 35 2019-02-15 13:00:00    244801 4.080017e+03 6.800028e+01 2.833345e+00
## 36 2019-02-15 18:00:00     18001 3.000167e+02 5.000278e+00 2.083449e-01
## 37 2019-02-17 09:00:00    140401 2.340017e+03 3.900028e+01 1.625012e+00
## 38 2019-02-17 12:00:00     10801 1.800167e+02 3.000278e+00 1.250116e-01
## 39 2019-02-18 03:00:00     54001 9.000167e+02 1.500028e+01 6.250116e-01
## 40 2019-02-20 13:00:00    208801 3.480017e+03 5.800028e+01 2.416678e+00
## 41 2019-02-21 00:00:00     39601 6.600167e+02 1.100028e+01 4.583449e-01
## 42 2019-02-23 12:00:00    216001 3.600017e+03 6.000028e+01 2.500012e+00
## 43 2019-02-23 20:00:00     28801 4.800167e+02 8.000278e+00 3.333449e-01
## 44 2019-02-23 21:00:00      3601 6.001667e+01 1.000278e+00 4.167824e-02
## 45 2019-02-25 22:00:00    176401 2.940017e+03 4.900028e+01 2.041678e+00
## 46 2019-02-26 17:00:00     68401 1.140017e+03 1.900028e+01 7.916782e-01
## 47 2019-02-26 23:00:00     21601 3.600167e+02 6.000278e+00 2.500116e-01
## 48 2019-03-01 15:00:00    230401 3.840017e+03 6.400028e+01 2.666678e+00
## 49 2019-03-01 19:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 50 2019-03-01 21:00:00      7201 1.200167e+02 2.000278e+00 8.334491e-02
## 51 2019-03-02 21:00:00     86401 1.440017e+03 2.400028e+01 1.000012e+00
## 52 2019-03-04 23:00:00    180001 3.000017e+03 5.000028e+01 2.083345e+00
## 53 2019-03-07 10:00:00    212401 3.540017e+03 5.900028e+01 2.458345e+00
## 54 2019-03-07 14:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 55 2019-03-07 19:00:00     18001 3.000167e+02 5.000278e+00 2.083449e-01
## 56 2019-03-10 12:00:00    234001 3.900017e+03 6.500028e+01 2.708345e+00
## 57 2019-03-10 18:00:00     21601 3.600167e+02 6.000278e+00 2.500116e-01
## 58 2019-03-11 13:00:00     68401 1.140017e+03 1.900028e+01 7.916782e-01
## 59 2019-03-11 20:00:00     25201 4.200167e+02 7.000278e+00 2.916782e-01
## 60 2019-03-12 18:00:00     79201 1.320017e+03 2.200028e+01 9.166782e-01
## 61 2019-03-12 19:00:00      3601 6.001667e+01 1.000278e+00 4.167824e-02
## 62 2019-03-17 12:00:00    406801 6.780017e+03 1.130003e+02 4.708345e+00
## 63 2019-03-17 12:00:00         1 1.666667e-02 2.777778e-04 1.157407e-05
## 64 2019-03-18 16:00:00    100801 1.680017e+03 2.800028e+01 1.166678e+00
## 65 2019-03-22 18:00:00    352801 5.880017e+03 9.800028e+01 4.083345e+00
## 66 2019-03-26 00:00:00    280801 4.680017e+03 7.800028e+01 3.250012e+00
## 67 2019-03-26 06:00:00     21601 3.600167e+02 6.000278e+00 2.500116e-01
## 68 2019-03-27 16:00:00    122401 2.040017e+03 3.400028e+01 1.416678e+00
```

```
zone_2_TimeDiff
```

```
##               dateTime diff_hours diff_mins diff_secs diff_days
## 2  2019-01-03 06:00:00         46      2760    165600 1.9166667
## 3  2019-01-04 10:00:00         29      1740    104400 1.2083333
## 4  2019-01-07 14:00:00         77      4620    277200 3.2083333
## 5  2019-01-11 06:00:00         89      5340    320400 3.7083333
## 6  2019-01-11 16:00:00         11       660     39600 0.4583333
## 7  2019-01-14 11:00:00         68      4080    244800 2.8333333
## 8  2019-01-16 02:00:00         40      2400    144000 1.6666667
## 9  2019-01-18 06:00:00         53      3180    190800 2.2083333
## 10 2019-01-19 17:00:00         36      2160    129600 1.5000000
## 11 2019-01-20 22:00:00         30      1800    108000 1.2500000
## 12 2019-01-21 11:00:00         14       840     50400 0.5833333
## 13 2019-01-22 21:00:00         35      2100    126000 1.4583333
## 14 2019-01-29 07:00:00        155      9300    558000 6.4583333
## 15 2019-02-06 02:00:00        188     11280    676800 7.8333333
## 16 2019-02-10 20:00:00        115      6900    414000 4.7916667
## 17 2019-02-14 05:00:00         82      4920    295200 3.4166667
```

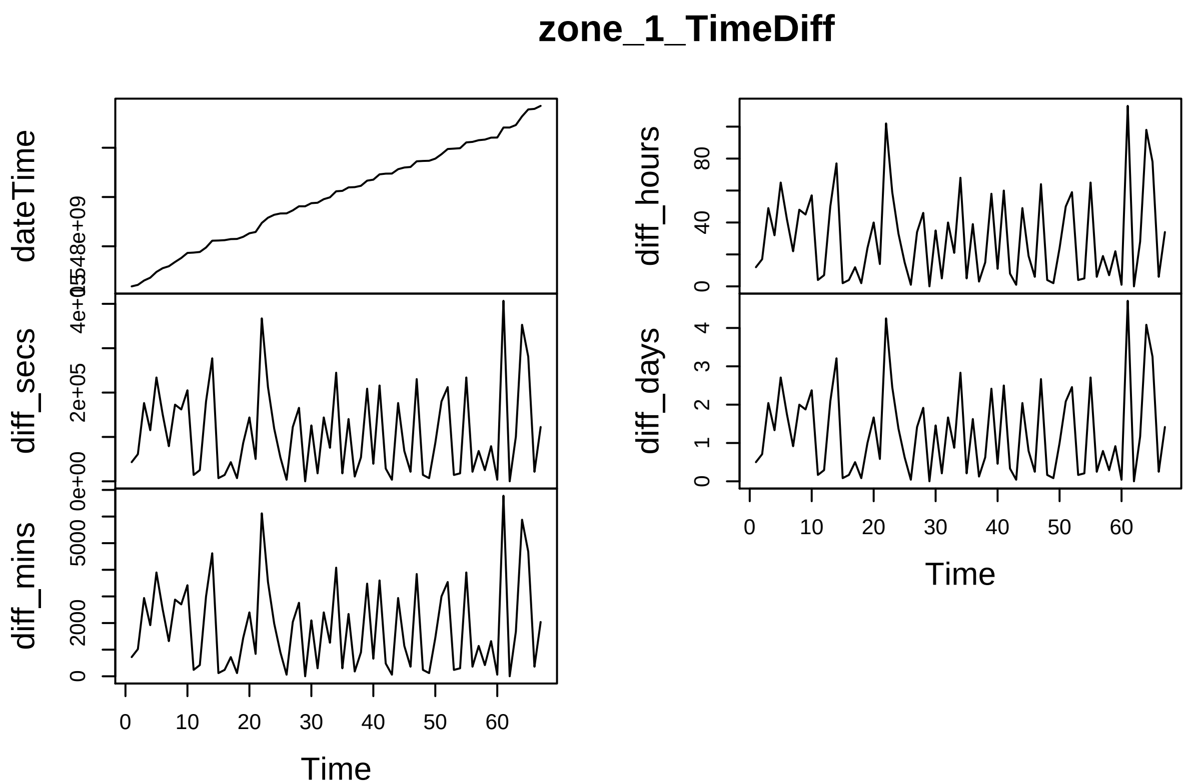
```
## 18 2019-02-15 11:00:00         31      1860    111600 1.2916667
## 19 2019-02-16 18:00:00         32      1920    115200 1.3333333
## 20 2019-02-25 14:00:00        213     12780    766800 8.8750000
## 21 2019-02-28 05:00:00         64      3840    230400 2.6666667
## 22 2019-02-28 12:00:00          8       480     28800 0.3333333
## 23 2019-03-07 21:00:00        178     10680    640800 7.4166667
## 24 2019-03-10 16:00:00         68      4080    244800 2.8333333
## 25 2019-03-10 23:00:00          8       480     28800 0.3333333
## 26 2019-03-11 18:00:00         20      1200     72000 0.8333333
## 27 2019-03-17 09:00:00        136      8160    489600 5.6666667
## 28 2019-03-20 10:00:00         74      4440    266400 3.0833333
## 29 2019-03-21 13:00:00         28      1680    100800 1.1666667
## 30 2019-03-24 16:00:00         76      4560    273600 3.1666667
## 31 2019-03-25 14:00:00         23      1380     82800 0.9583333
## 32 2019-03-28 01:00:00         60      3600    216000 2.5000000
```

7.1 Difference betewen the zones

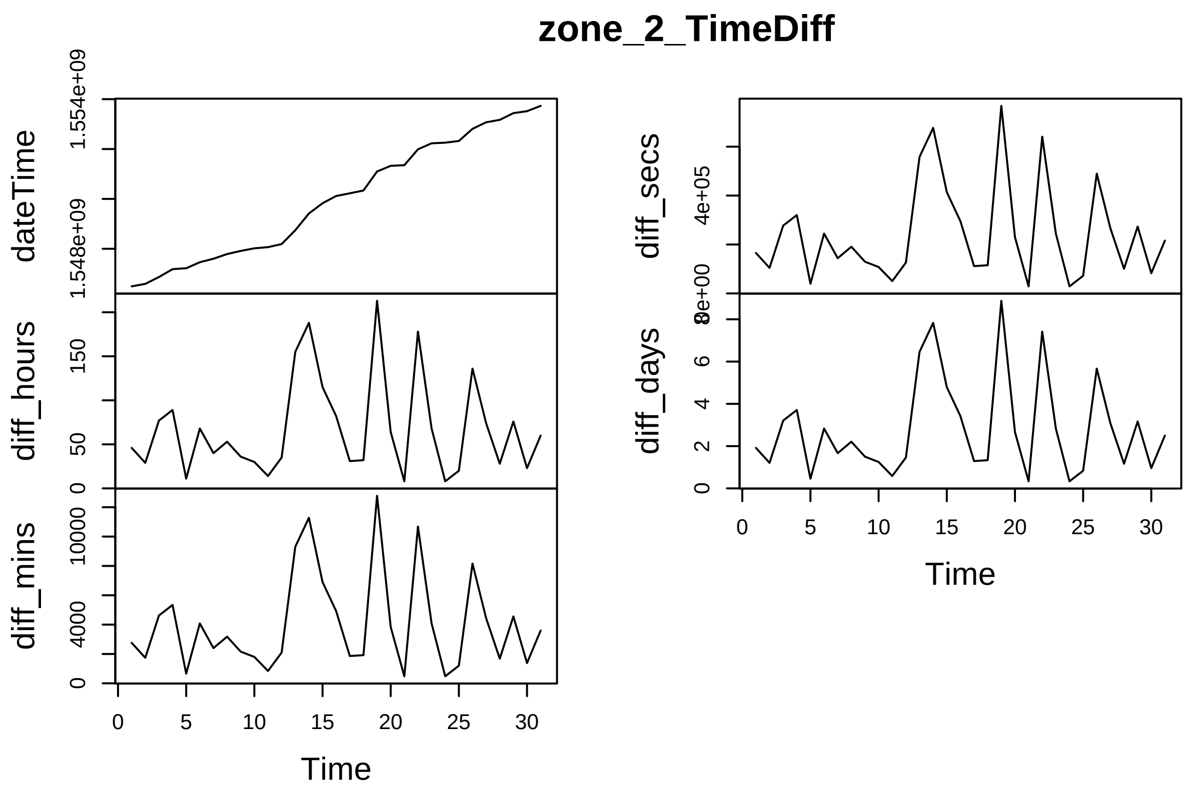
It shows that there is a big difference between the two datasets. - The analysis of the time between events has a probability calculation. The cut of the distances is 30.55 hours with stretching from 0 to 133 hours. From the histogram it can be seen that there is a steep rise at the beginning that reaches its maxium at c.a 10 hours and then flattens again. The maximum time intervals between each end is 113 hours, now the question is whether this is also the maximum. I suppose the maximum of the distances is one year, because this is the time because after this time comes the new network.

- On the one hand this merging of "the time in which nothing happened" distorts the result, on the other hand these values are all at 0 anyway. The goal of a time series is to determine a certain distribution by smoothing and co. This is not useful because many values are at 0. Time series are only meaningful if the Y-value never goes to 0.
- Time unit must be checked, perhaps there is a time unit that outputs a symmetricteling. Time units have been extended, dataset now has the time distance between the achievements in seconds, minutes, hours and days.
- Class instead of mass. I could divide the time unit more and more. For the optimal class size there are different approaches. One of them is the root of the number of events. The events are in record 1= 67. In record 2 they are 31.
  - Class division 1 = 8.18 -> 8
  - Class division 2 = 5.56 -> 6
- Time interval was divided into classes of days. To keep the optimal class width the binwith is set to 0.5. Now there are 10 classes. Which in my opinion is a good class instead of mass.
- Time interval was divided into "regular to each other" classes. Now only the probability has to be calculated.

```r
plot.ts(zone_1_TimeDiff)
```



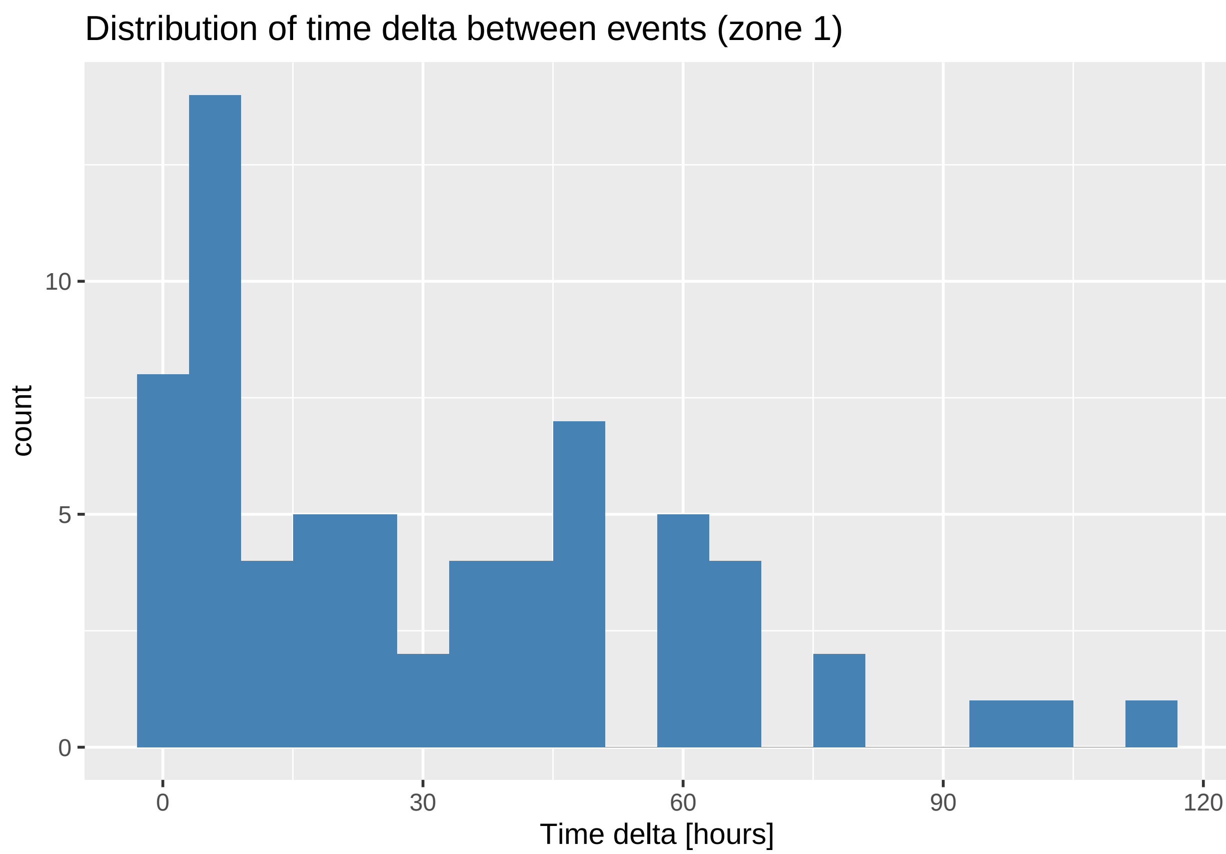

```
plot.ts(zone_2_TimeDiff)
```




```r
# hier ist jede gemesene Stunde eine Klasse. Ansatz: Klasse statt Masse
# Doch zuerst wird est mal die Diff_hour untersucht

stat_data_3 <- dplyr::select(describe(dplyr::select(zone_1_TimeDiff, diff_days,diff_hours, diff_mins, di

stat_data_4 <- dplyr::select(describe(dplyr::select(zone_2_TimeDiff, diff_days,diff_hours, diff_mins, di

stat_zone_new2 <- rbind(stat_data_3,stat_data_4)
stat_zone_new2
```

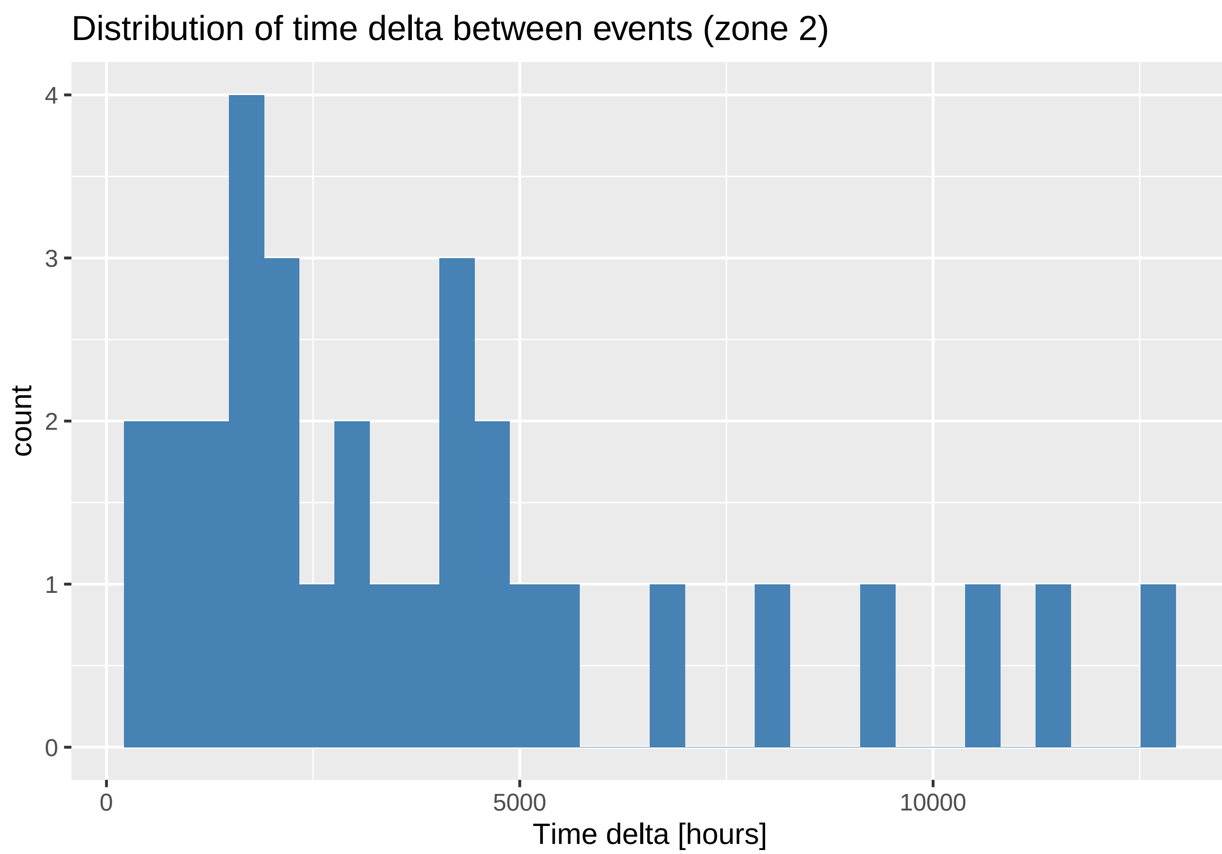
```
##                   mean        sd    median      min       max
## diff_days         1.27      1.16      0.92     0.00      4.71
## diff_hours       30.55     27.75     22.00     0.00    113.00
## diff_mins      1833.15   1664.94   1320.02     0.02   6780.02
## diff_secs    109989.06  99896.50  79201.00     1.00 406801.00
## diff_days1        2.81      2.30      2.21     0.33      8.88
## diff_hours1      67.32     55.18     53.00     8.00    213.00
## diff_mins1     4039.35   3310.72   3180.00   480.00  12780.00
## diff_secs1   242361.29 198643.44 190800.00 28800.00 766800.00
```

```r
zone_1_TimeDiff %>%
  ggplot() +
  geom_histogram(aes(x = diff_hours), fill = "steelblue", binwidth = 6)+
  labs(title='Distribution of time delta between events (zone 1)', x = 'Time delta [hours]')
```




```r
zone_2_TimeDiff %>%
  ggplot() +
  geom_histogram(aes(x = diff_mins), fill = "steelblue")+
  labs(title='Distribution of time delta between events (zone 2)', x = 'Time delta [hours]')
```

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



7.2 The probability of the time difference

- The fitting of the probability distribution does not depend on the time magnitude and therefore also not on the Binwith.
- It's hard to say which distribution to take.

7.2.1 Result of zone 1

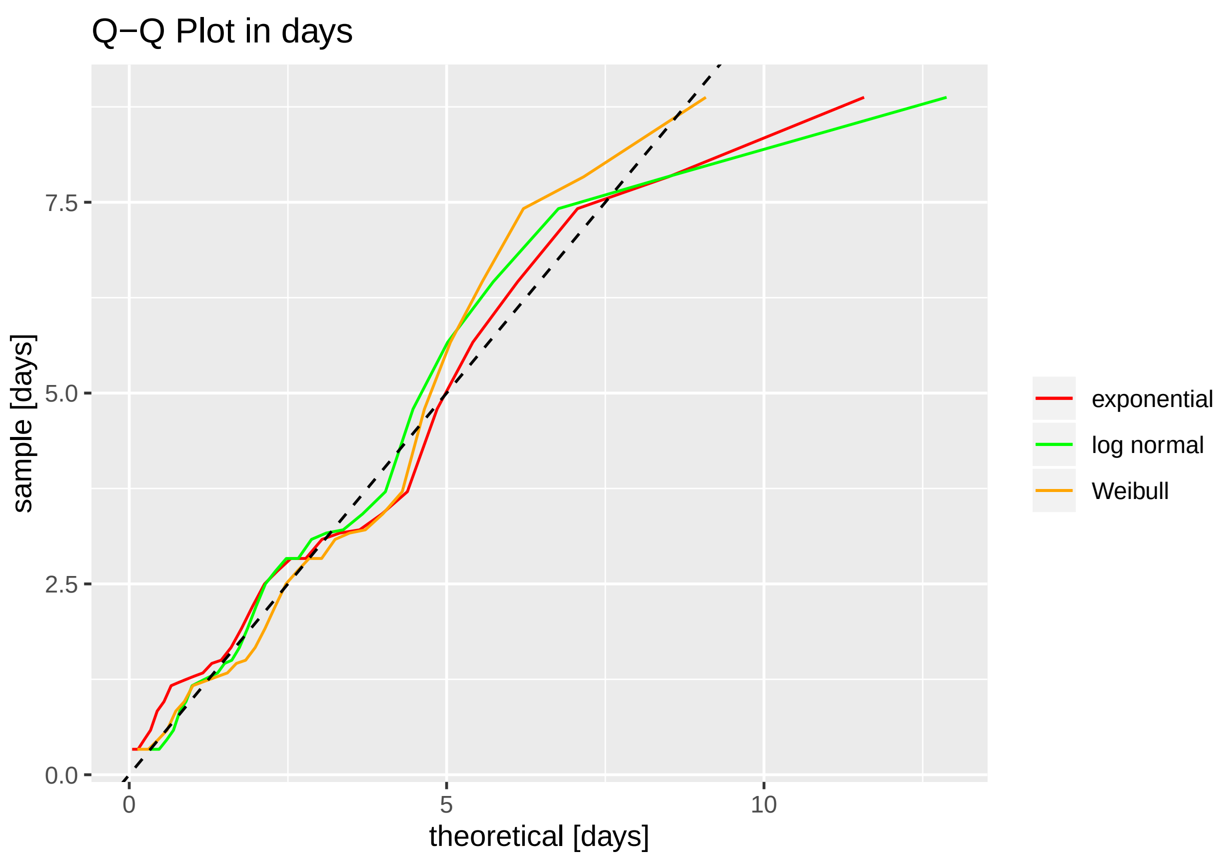
The best fit is the "log normal" distribution, because this distribution is clearly the best at the beginning and even in the area where all distributions begin to become inaccurate it is still best. Only at the last extreme points is the log distribution the worst.

```
# Untersuchung welche Zeiteinheit am besten passt.
# GGplot für Tage

e_fit_mass_zone_2_d <- fitdistr(zone_2_TimeDiff$diff_days, "exponential")
lnrm_fit_mass_zone_2_d <- fitdistr(zone_2_TimeDiff$diff_days, "lognormal")
w_fit_mass_zone_2_d <- fitdistr(zone_2_TimeDiff$diff_days, "weibull")
```

```
## Warning in densfun(x, parm[1], parm[2], ...): NaNs wurden erzeugt
```

```
ggplot(zone_2_TimeDiff, aes(sample = diff_days)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit_mass_zone_2_d$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qlnorm,
```

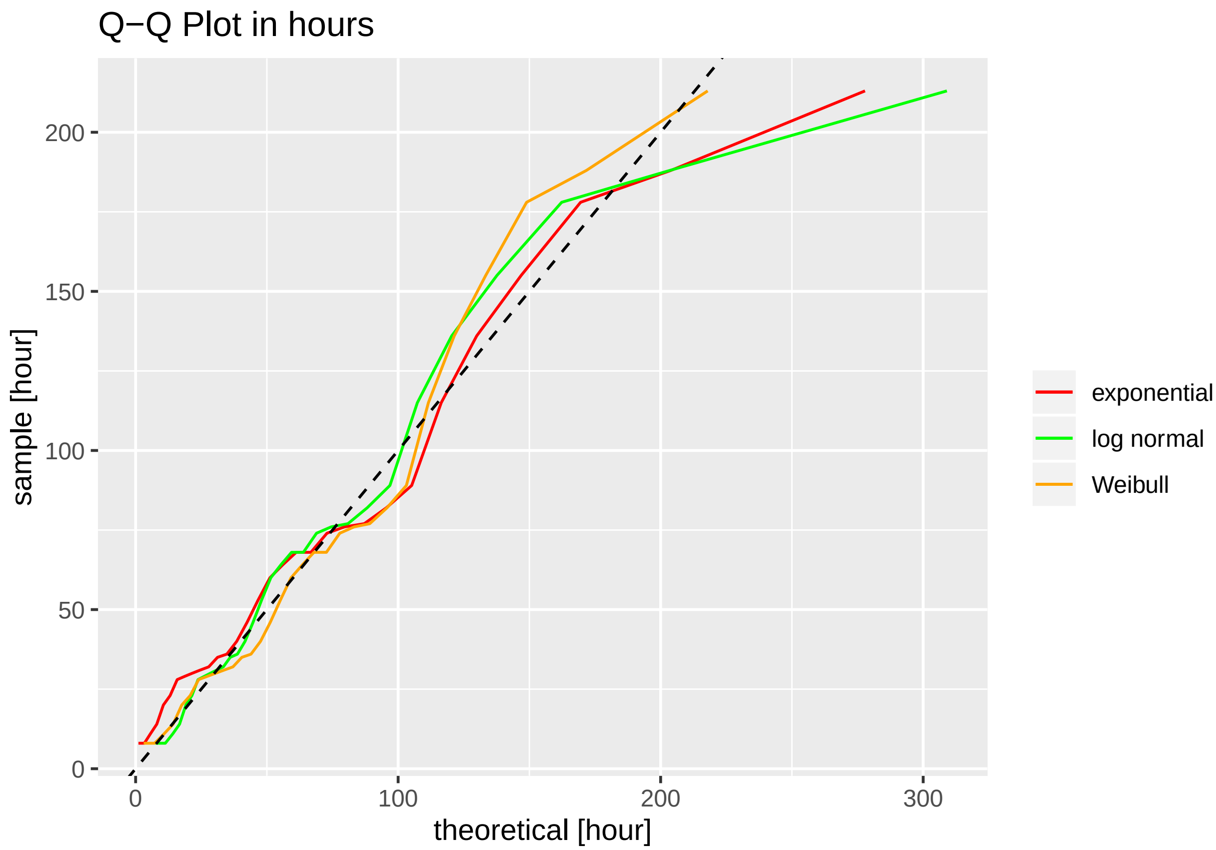
```
         dparams = list(lnrm_fit_mass_zone_2_d$estimate[1], lnrm_fit_mass_zone_2_d$estimate[2]),
         geom = "line",
         aes(color = "log normal")) +
  stat_qq(distribution = qweibull,
         dparams = list(w_fit_mass_zone_2_d$estimate[1], w_fit_mass_zone_2_d$estimate[2]),
         geom = "line",
         aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                    values = c("exponential" = "red",
                              # "gamma" = "blue",
                               "log normal" = "green",
                               "Weibull" = "orange"),
                    breaks = c("exponential",
                               #"gamma",
                               "log normal",
                               "Weibull")) +
  labs(title = "Q-Q Plot in days", x = 'theoretical [days]', y = 'sample [days]')
```




```
#GGplot für Stunden

e_fit_mass_zone_2_h <- fitdistr(zone_2_TimeDiff$diff_hours, "exponential")
lnrm_fit_mass_zone_2_h <- fitdistr(zone_2_TimeDiff$diff_hours, "lognormal")
w_fit_mass_zone_2_h <- fitdistr(zone_2_TimeDiff$diff_hours, "weibull")
```

```
## Warning in densfun(x, parm[1], parm[2], ...): NaNs wurden erzeugt
```

```r
ggplot(zone_2_TimeDiff, aes(sample = diff_hours)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit_mass_zone_2_h$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qlnorm,
          dparams = list(lnrm_fit_mass_zone_2_h$estimate[1], lnrm_fit_mass_zone_2_h$estimate[2]),
          geom = "line",
          aes(color = "log normal")) +
  stat_qq(distribution = qweibull,
          dparams = list(w_fit_mass_zone_2_h$estimate[1], w_fit_mass_zone_2_h$estimate[2]),
          geom = "line",
          aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("exponential" = "red",
                                "log normal" = "green",
                                "Weibull" = "orange"),
                     breaks = c("exponential",
                                "log normal",
                                "Weibull")) +
  labs(title = "Q-Q Plot in hours", x = 'theoretical [hour]', y = 'sample [hour]')
```

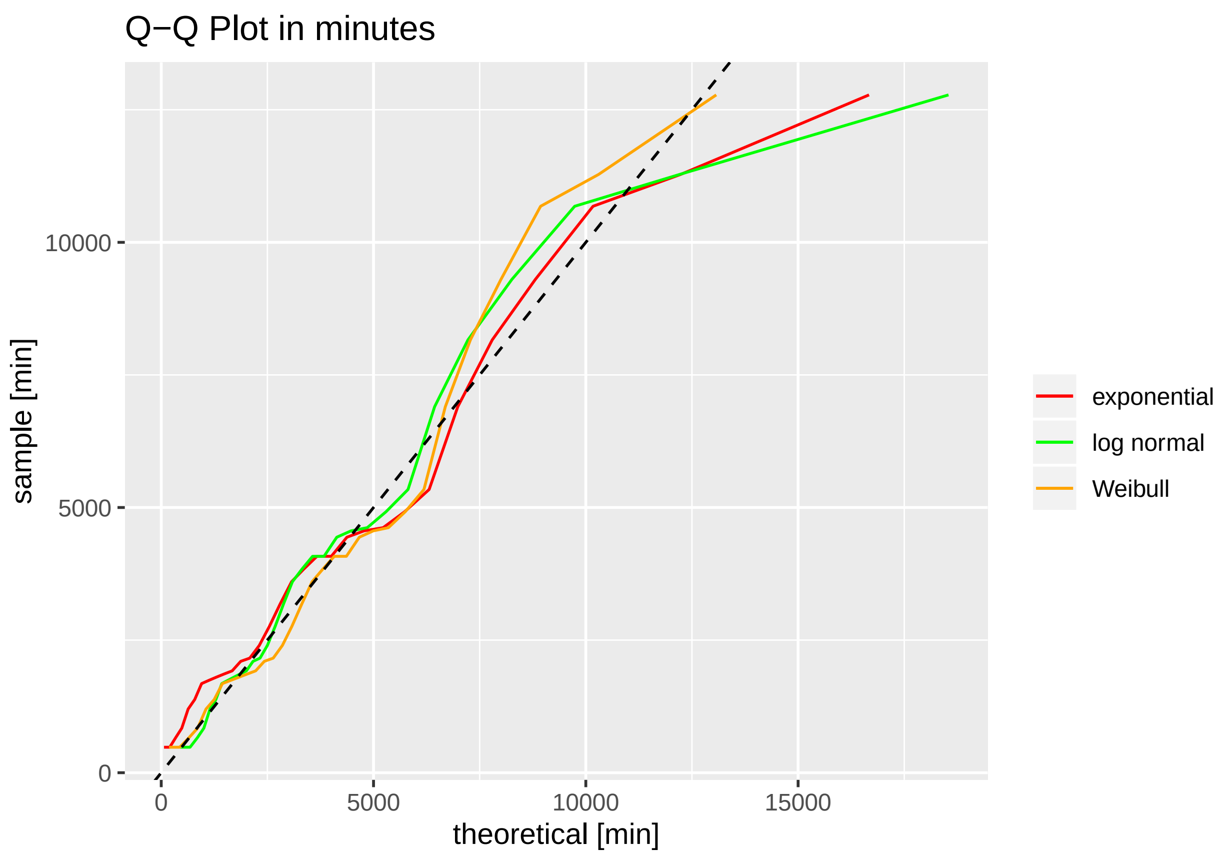



```r
# QQ-plot für Minuten
```

```r
e_fit_mass_zone_2_m <- fitdistr(zone_2_TimeDiff$diff_mins, "exponential")
lnrm_fit_mass_zone_2_m <- fitdistr(zone_2_TimeDiff$diff_mins, "lognormal")
w_fit_mass_zone_2_m <- fitdistr(zone_2_TimeDiff$diff_mins, "weibull")
```

```
## Warning in densfun(x, parm[1], parm[2], ...): NaNs wurden erzeugt
```

```r
ggplot(zone_2_TimeDiff, aes(sample = diff_mins)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit_mass_zone_2_m$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qlnorm,
          dparams = list(lnrm_fit_mass_zone_2_m$estimate[1], lnrm_fit_mass_zone_2_m$estimate[2]),
          geom = "line",
          aes(color = "log normal")) +
  stat_qq(distribution = qweibull,
          dparams = list(w_fit_mass_zone_2_m$estimate[1], w_fit_mass_zone_2_m$estimate[2]),
          geom = "line",
          aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("exponential" = "red",
                                "log normal" = "green",
                                "Weibull" = "orange"),
                     breaks = c("exponential",
                                "log normal",
                                "Weibull")) +
  labs(title = "Q-Q Plot in minutes", x = 'theoretical [min]', y = 'sample [min]')
```

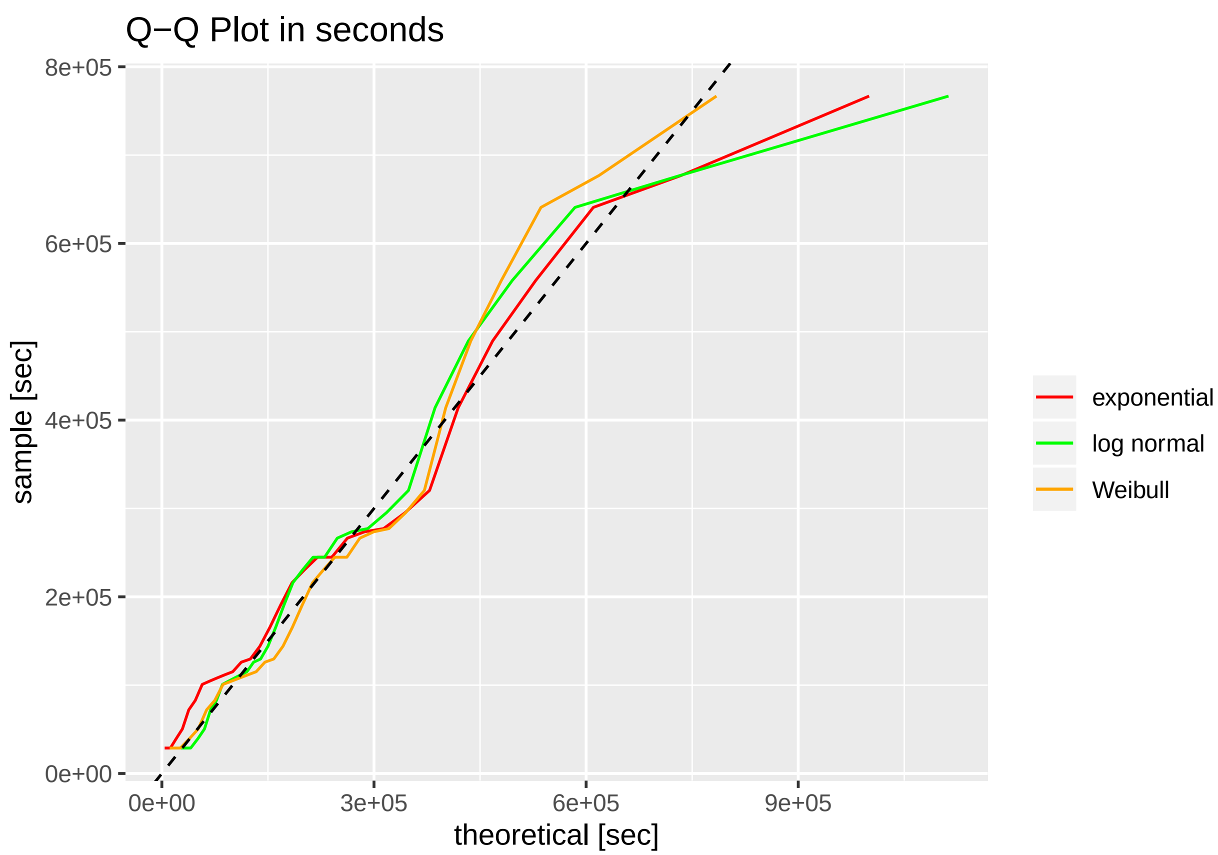



```
# QQ-plot für Minuten

e_fit_mass_zone_2_s <- fitdistr(zone_2_TimeDiff$diff_secs, "exponential")
lnrm_fit_mass_zone_2_s <- fitdistr(zone_2_TimeDiff$diff_secs, "lognormal")
w_fit_mass_zone_2_s <- fitdistr(zone_2_TimeDiff$diff_secs, "weibull")
```

```
## Warning in densfun(x, parm[1], parm[2], ...): NaNs wurden erzeugt
```

```
ggplot(zone_2_TimeDiff, aes(sample = diff_secs)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit_mass_zone_2_s$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qlnorm,
          dparams = list(lnrm_fit_mass_zone_2_s$estimate[1], lnrm_fit_mass_zone_2_s$estimate[2]),
          geom = "line",
          aes(color = "log normal")) +
  stat_qq(distribution = qweibull,
          dparams = list(w_fit_mass_zone_2_s$estimate[1], w_fit_mass_zone_2_s$estimate[2]),
          geom = "line",
          aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("exponential" = "red",
                                "log normal" = "green",
                                "Weibull" = "orange"),
                     breaks = c("exponential",
```

```
                                   "log normal",
                                   "Weibull")) +
  labs(title = "Q-Q Plot in seconds",  x = 'theoretical [sec]', y = 'sample [sec]')
```




```
gDistTimeZ2 <- fitdistr(zone_2_TimeDiff$diff_mins, "log-normal")

gDistTimeZ2$estimate[1]
```

```
##  meanlog
## 7.965341
```

```
gDistTimeZ2$estimate[2]
```

```
##   sdlog
## 0.86983
```

7.2.2 Result of zone 1

The best fit is the "exponential" distribution.

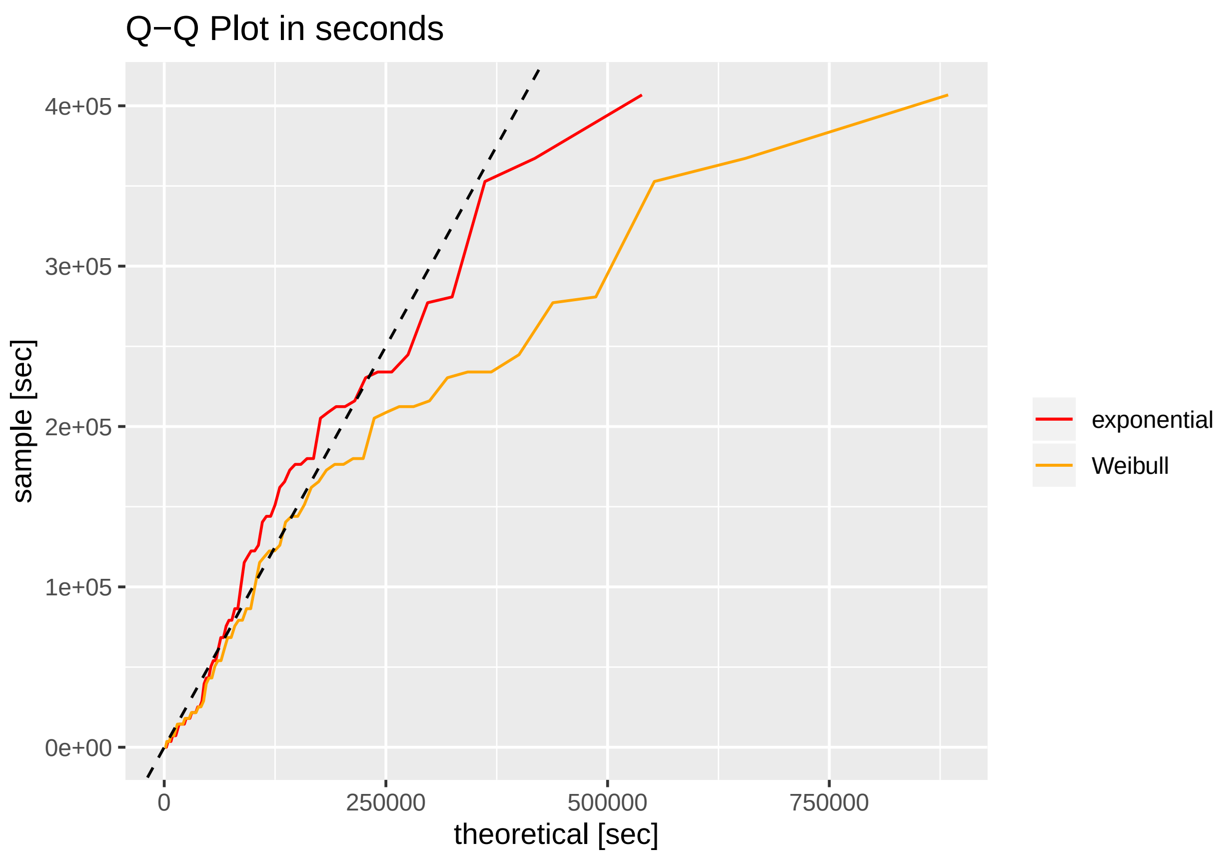
```
# QQ-plot für Sekunden

zone_1_TimeDiff
```

```
##                 dateTime diff_secs    diff_mins   diff_hours    diff_days
## 2  2019-01-01 21:00:00     43201 7.200167e+02 1.200028e+01 5.000116e-01
## 3  2019-01-02 14:00:00     61201 1.020017e+03 1.700028e+01 7.083449e-01
## 4  2019-01-04 15:00:00    176401 2.940017e+03 4.900028e+01 2.041678e+00
```

```
## 5  2019-01-05 23:00:00    115201 1.920017e+03 3.200028e+01 1.333345e+00
## 6  2019-01-08 16:00:00    234001 3.900017e+03 6.500028e+01 2.708345e+00
## 7  2019-01-10 10:00:00    151201 2.520017e+03 4.200028e+01 1.750012e+00
## 8  2019-01-11 08:00:00     79201 1.320017e+03 2.200028e+01 9.166782e-01
## 9  2019-01-13 08:00:00    172801 2.880017e+03 4.800028e+01 2.000012e+00
## 10 2019-01-15 05:00:00    162001 2.700017e+03 4.500028e+01 1.875012e+00
## 11 2019-01-17 14:00:00    205201 3.420017e+03 5.700028e+01 2.375012e+00
## 12 2019-01-17 18:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 13 2019-01-18 01:00:00     25201 4.200167e+02 7.000278e+00 2.916782e-01
## 14 2019-01-20 03:00:00    180001 3.000017e+03 5.000028e+01 2.083345e+00
## 15 2019-01-23 08:00:00    277201 4.620017e+03 7.700028e+01 3.208345e+00
## 16 2019-01-23 10:00:00      7201 1.200167e+02 2.000278e+00 8.334491e-02
## 17 2019-01-23 14:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 18 2019-01-24 02:00:00     43201 7.200167e+02 1.200028e+01 5.000116e-01
## 19 2019-01-24 04:00:00      7201 1.200167e+02 2.000278e+00 8.334491e-02
## 20 2019-01-25 04:00:00     86401 1.440017e+03 2.400028e+01 1.000012e+00
## 21 2019-01-26 20:00:00    144001 2.400017e+03 4.000028e+01 1.666678e+00
## 22 2019-01-27 10:00:00     50401 8.400167e+02 1.400028e+01 5.833449e-01
## 23 2019-01-31 16:00:00    367201 6.120017e+03 1.020003e+02 4.250012e+00
## 24 2019-02-03 03:00:00    212401 3.540017e+03 5.900028e+01 2.458345e+00
## 25 2019-02-04 12:00:00    118801 1.980017e+03 3.300028e+01 1.375012e+00
## 26 2019-02-05 03:00:00     54001 9.000167e+02 1.500028e+01 6.250116e-01
## 27 2019-02-05 04:00:00      3601 6.001667e+01 1.000278e+00 4.167824e-02
## 28 2019-02-06 14:00:00    122401 2.040017e+03 3.400028e+01 1.416678e+00
## 29 2019-02-08 12:00:00    165601 2.760017e+03 4.600028e+01 1.916678e+00
## 30 2019-02-08 12:00:00         1 1.666667e-02 2.777778e-04 1.157407e-05
## 31 2019-02-09 23:00:00    126001 2.100017e+03 3.500028e+01 1.458345e+00
## 32 2019-02-10 04:00:00     18001 3.000167e+02 5.000278e+00 2.083449e-01
## 33 2019-02-11 20:00:00    144001 2.400017e+03 4.000028e+01 1.666678e+00
## 34 2019-02-12 17:00:00     75601 1.260017e+03 2.100028e+01 8.750116e-01
## 35 2019-02-15 13:00:00    244801 4.080017e+03 6.800028e+01 2.833345e+00
## 36 2019-02-15 18:00:00     18001 3.000167e+02 5.000278e+00 2.083449e-01
## 37 2019-02-17 09:00:00    140401 2.340017e+03 3.900028e+01 1.625012e+00
## 38 2019-02-17 12:00:00     10801 1.800167e+02 3.000278e+00 1.250116e-01
## 39 2019-02-18 03:00:00     54001 9.000167e+02 1.500028e+01 6.250116e-01
## 40 2019-02-20 13:00:00    208801 3.480017e+03 5.800028e+01 2.416678e+00
## 41 2019-02-21 00:00:00     39601 6.600167e+02 1.100028e+01 4.583449e-01
## 42 2019-02-23 12:00:00    216001 3.600017e+03 6.000028e+01 2.500012e+00
## 43 2019-02-23 20:00:00     28801 4.800167e+02 8.000278e+00 3.333449e-01
## 44 2019-02-23 21:00:00      3601 6.001667e+01 1.000278e+00 4.167824e-02
## 45 2019-02-25 22:00:00    176401 2.940017e+03 4.900028e+01 2.041678e+00
## 46 2019-02-26 17:00:00     68401 1.140017e+03 1.900028e+01 7.916782e-01
## 47 2019-02-26 23:00:00     21601 3.600167e+02 6.000278e+00 2.500116e-01
## 48 2019-03-01 15:00:00    230401 3.840017e+03 6.400028e+01 2.666678e+00
## 49 2019-03-01 19:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 50 2019-03-01 21:00:00      7201 1.200167e+02 2.000278e+00 8.334491e-02
## 51 2019-03-02 21:00:00     86401 1.440017e+03 2.400028e+01 1.000012e+00
## 52 2019-03-04 23:00:00    180001 3.000017e+03 5.000028e+01 2.083345e+00
## 53 2019-03-07 10:00:00    212401 3.540017e+03 5.900028e+01 2.458345e+00
## 54 2019-03-07 14:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 55 2019-03-07 19:00:00     18001 3.000167e+02 5.000278e+00 2.083449e-01
## 56 2019-03-10 12:00:00    234001 3.900017e+03 6.500028e+01 2.708345e+00
## 57 2019-03-10 18:00:00     21601 3.600167e+02 6.000278e+00 2.500116e-01
## 58 2019-03-11 13:00:00     68401 1.140017e+03 1.900028e+01 7.916782e-01
```

```
## 59 2019-03-11 20:00:00     25201 4.200167e+02 7.000278e+00 2.916782e-01
## 60 2019-03-12 18:00:00     79201 1.320017e+03 2.200028e+01 9.166782e-01
## 61 2019-03-12 19:00:00      3601 6.001667e+01 1.000278e+00 4.167824e-02
## 62 2019-03-17 12:00:00    406801 6.780017e+03 1.130003e+02 4.708345e+00
## 63 2019-03-17 12:00:00         1 1.666667e-02 2.777778e-04 1.157407e-05
## 64 2019-03-18 16:00:00    100801 1.680017e+03 2.800028e+01 1.166678e+00
## 65 2019-03-22 18:00:00    352801 5.880017e+03 9.800028e+01 4.083345e+00
## 66 2019-03-26 00:00:00    280801 4.680017e+03 7.800028e+01 3.250012e+00
## 67 2019-03-26 06:00:00     21601 3.600167e+02 6.000278e+00 2.500116e-01
## 68 2019-03-27 16:00:00    122401 2.040017e+03 3.400028e+01 1.416678e+00
```

```r
e_fit_mass_zone_1_s <- fitdistr(zone_1_TimeDiff$diff_secs, "exponential")
w_fit_mass_zone_1_s <- fitdistr(zone_1_TimeDiff$diff_secs, "weibull")

ggplot(zone_1_TimeDiff, aes(sample = diff_secs)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit_mass_zone_1_s$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qweibull,
          dparams = list(w_fit_mass_zone_1_s$estimate[1], w_fit_mass_zone_1_s$estimate[2]),
          geom = "line",
          aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("exponential" = "red",
                                "Weibull" = "orange"),
                     breaks = c("exponential",
                                "Weibull")) +
  labs(title = "Q-Q Plot in seconds",x = 'theoretical [sec]', y = 'sample [sec]')
```




```r
show(zone_1_TimeDiff)
```

```
##                dateTime diff_secs    diff_mins   diff_hours    diff_days
## 2  2019-01-01 21:00:00     43201 7.200167e+02 1.200028e+01 5.000116e-01
## 3  2019-01-02 14:00:00     61201 1.020017e+03 1.700028e+01 7.083449e-01
## 4  2019-01-04 15:00:00    176401 2.940017e+03 4.900028e+01 2.041678e+00
## 5  2019-01-05 23:00:00    115201 1.920017e+03 3.200028e+01 1.333345e+00
## 6  2019-01-08 16:00:00    234001 3.900017e+03 6.500028e+01 2.708345e+00
## 7  2019-01-10 10:00:00    151201 2.520017e+03 4.200028e+01 1.750012e+00
## 8  2019-01-11 08:00:00     79201 1.320017e+03 2.200028e+01 9.166782e-01
## 9  2019-01-13 08:00:00    172801 2.880017e+03 4.800028e+01 2.000012e+00
## 10 2019-01-15 05:00:00    162001 2.700017e+03 4.500028e+01 1.875012e+00
## 11 2019-01-17 14:00:00    205201 3.420017e+03 5.700028e+01 2.375012e+00
## 12 2019-01-17 18:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 13 2019-01-18 01:00:00     25201 4.200167e+02 7.000278e+00 2.916782e-01
## 14 2019-01-20 03:00:00    180001 3.000017e+03 5.000028e+01 2.083345e+00
## 15 2019-01-23 08:00:00    277201 4.620017e+03 7.700028e+01 3.208345e+00
## 16 2019-01-23 10:00:00      7201 1.200167e+02 2.000278e+00 8.334491e-02
## 17 2019-01-23 14:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 18 2019-01-24 02:00:00     43201 7.200167e+02 1.200028e+01 5.000116e-01
## 19 2019-01-24 04:00:00      7201 1.200167e+02 2.000278e+00 8.334491e-02
## 20 2019-01-25 04:00:00     86401 1.440017e+03 2.400028e+01 1.000012e+00
## 21 2019-01-26 20:00:00    144001 2.400017e+03 4.000028e+01 1.666678e+00
## 22 2019-01-27 10:00:00     50401 8.400167e+02 1.400028e+01 5.833449e-01
## 23 2019-01-31 16:00:00    367201 6.120017e+03 1.020003e+02 4.250012e+00
## 24 2019-02-03 03:00:00    212401 3.540017e+03 5.900028e+01 2.458345e+00
```

```
## 25 2019-02-04 12:00:00    118801 1.980017e+03 3.300028e+01 1.375012e+00
## 26 2019-02-05 03:00:00     54001 9.000167e+02 1.500028e+01 6.250116e-01
## 27 2019-02-05 04:00:00      3601 6.001667e+01 1.000278e+00 4.167824e-02
## 28 2019-02-06 14:00:00    122401 2.040017e+03 3.400028e+01 1.416678e+00
## 29 2019-02-08 12:00:00    165601 2.760017e+03 4.600028e+01 1.916678e+00
## 30 2019-02-08 12:00:00         1 1.666667e-02 2.777778e-04 1.157407e-05
## 31 2019-02-09 23:00:00    126001 2.100017e+03 3.500028e+01 1.458345e+00
## 32 2019-02-10 04:00:00     18001 3.000167e+02 5.000278e+00 2.083449e-01
## 33 2019-02-11 20:00:00    144001 2.400017e+03 4.000028e+01 1.666678e+00
## 34 2019-02-12 17:00:00     75601 1.260017e+03 2.100028e+01 8.750116e-01
## 35 2019-02-15 13:00:00    244801 4.080017e+03 6.800028e+01 2.833345e+00
## 36 2019-02-15 18:00:00     18001 3.000167e+02 5.000278e+00 2.083449e-01
## 37 2019-02-17 09:00:00    140401 2.340017e+03 3.900028e+01 1.625012e+00
## 38 2019-02-17 12:00:00     10801 1.800167e+02 3.000278e+00 1.250116e-01
## 39 2019-02-18 03:00:00     54001 9.000167e+02 1.500028e+01 6.250116e-01
## 40 2019-02-20 13:00:00    208801 3.480017e+03 5.800028e+01 2.416678e+00
## 41 2019-02-21 00:00:00     39601 6.600167e+02 1.100028e+01 4.583449e-01
## 42 2019-02-23 12:00:00    216001 3.600017e+03 6.000028e+01 2.500012e+00
## 43 2019-02-23 20:00:00     28801 4.800167e+02 8.000278e+00 3.333449e-01
## 44 2019-02-23 21:00:00      3601 6.001667e+01 1.000278e+00 4.167824e-02
## 45 2019-02-25 22:00:00    176401 2.940017e+03 4.900028e+01 2.041678e+00
## 46 2019-02-26 17:00:00     68401 1.140017e+03 1.900028e+01 7.916782e-01
## 47 2019-02-26 23:00:00     21601 3.600167e+02 6.000278e+00 2.500116e-01
## 48 2019-03-01 15:00:00    230401 3.840017e+03 6.400028e+01 2.666678e+00
## 49 2019-03-01 19:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 50 2019-03-01 21:00:00      7201 1.200167e+02 2.000278e+00 8.334491e-02
## 51 2019-03-02 21:00:00     86401 1.440017e+03 2.400028e+01 1.000012e+00
## 52 2019-03-04 23:00:00    180001 3.000017e+03 5.000028e+01 2.083345e+00
## 53 2019-03-07 10:00:00    212401 3.540017e+03 5.900028e+01 2.458345e+00
## 54 2019-03-07 14:00:00     14401 2.400167e+02 4.000278e+00 1.666782e-01
## 55 2019-03-07 19:00:00     18001 3.000167e+02 5.000278e+00 2.083449e-01
## 56 2019-03-10 12:00:00    234001 3.900017e+03 6.500028e+01 2.708345e+00
## 57 2019-03-10 18:00:00     21601 3.600167e+02 6.000278e+00 2.500116e-01
## 58 2019-03-11 13:00:00     68401 1.140017e+03 1.900028e+01 7.916782e-01
## 59 2019-03-11 20:00:00     25201 4.200167e+02 7.000278e+00 2.916782e-01
## 60 2019-03-12 18:00:00     79201 1.320017e+03 2.200028e+01 9.166782e-01
## 61 2019-03-12 19:00:00      3601 6.001667e+01 1.000278e+00 4.167824e-02
## 62 2019-03-17 12:00:00    406801 6.780017e+03 1.130003e+02 4.708345e+00
## 63 2019-03-17 12:00:00         1 1.666667e-02 2.777778e-04 1.157407e-05
## 64 2019-03-18 16:00:00    100801 1.680017e+03 2.800028e+01 1.166678e+00
## 65 2019-03-22 18:00:00    352801 5.880017e+03 9.800028e+01 4.083345e+00
## 66 2019-03-26 00:00:00    280801 4.680017e+03 7.800028e+01 3.250012e+00
## 67 2019-03-26 06:00:00     21601 3.600167e+02 6.000278e+00 2.500116e-01
## 68 2019-03-27 16:00:00    122401 2.040017e+03 3.400028e+01 1.416678e+00
```

```r
# Ergebniss 1 : Die Verteilung die am besten passt ist die exponentialverteilung.

gDistTimeZ1 <- fitdistr(zone_2_TimeDiff$diff_mins, "exponential")
gDistTimeZ1$estimate[1]
```

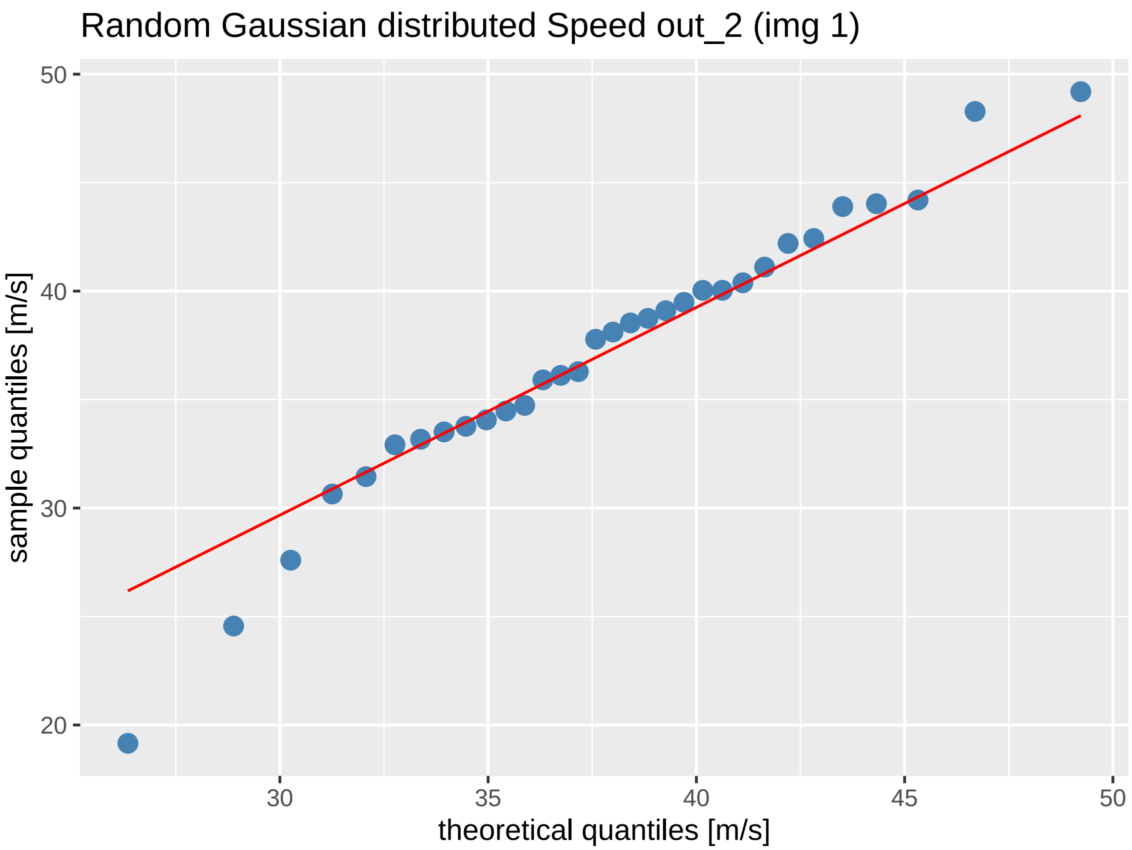
```
##         rate
## 0.0002475643
```

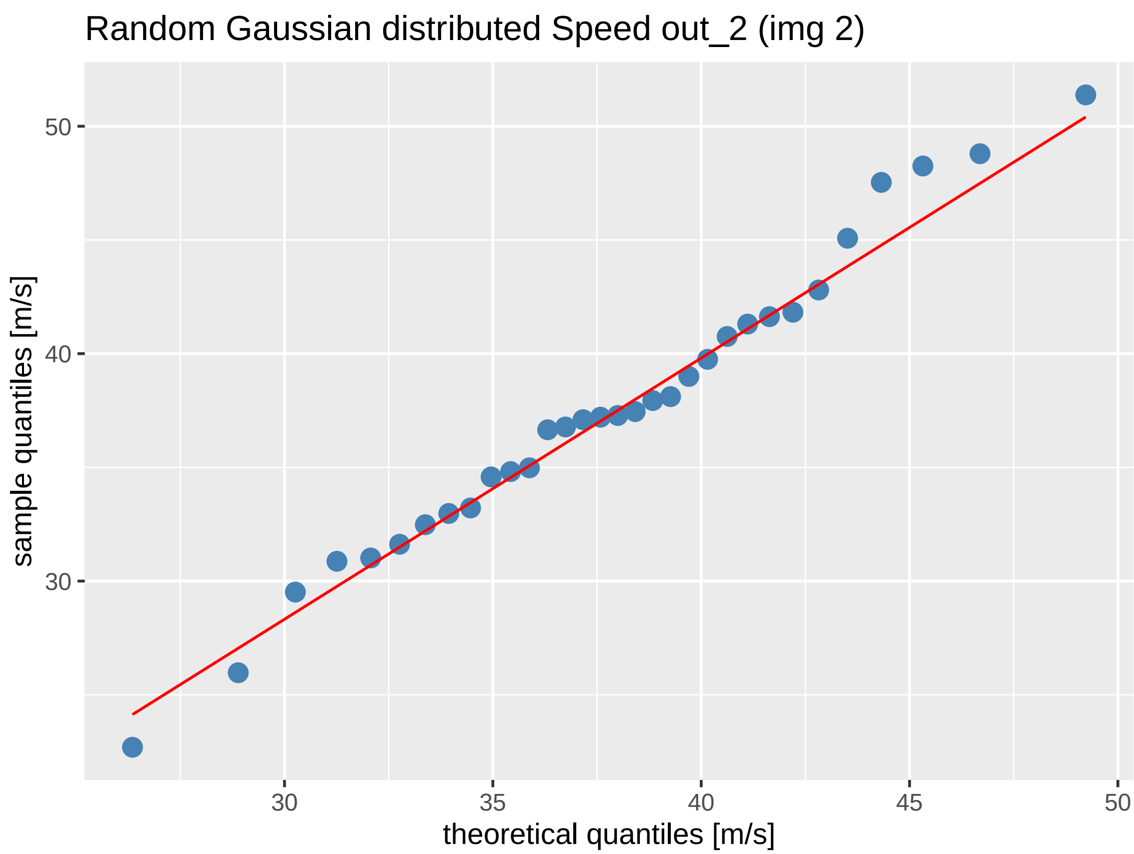
8. Montecarlo

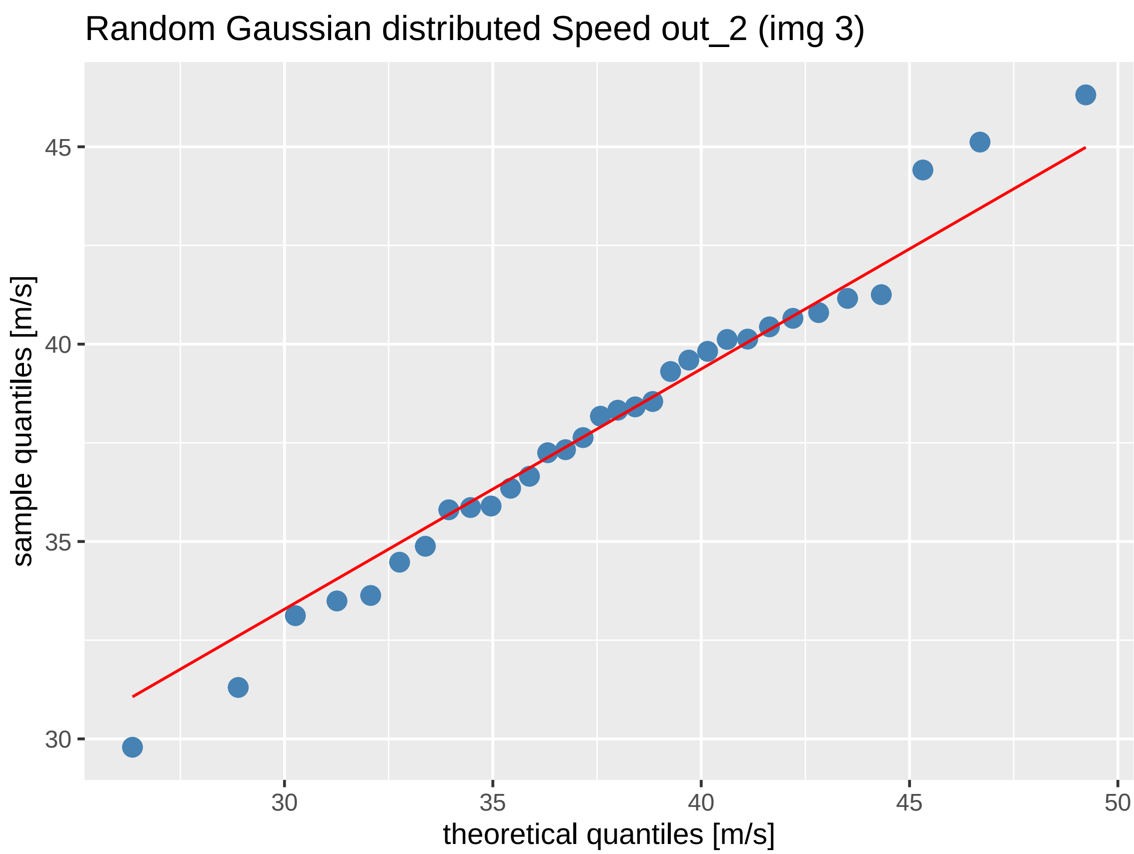
Because the previous test is ambiguous, we test if the sample could be normal distributed. We assume (H_0) that the speed of `out_2` is normal distributed with significant level 95%. H_1 is that the speed of `out_2` of the data set is not normal distributed. Therefore, we compare 20 smples of random generated normal distributed quantiles vs. the theoretical quantiles. If there is a similar QQ-plot compared to the plot with the actual speed data, then we can assume that the speed of out_2 is normal distributed.

The comparison of the artificial created data sample and the speed data `out_2` let us assume, not to deny H_0. In addition, it is important to note that most of the speed values have not been estimated significantly differently with the normal distribution.

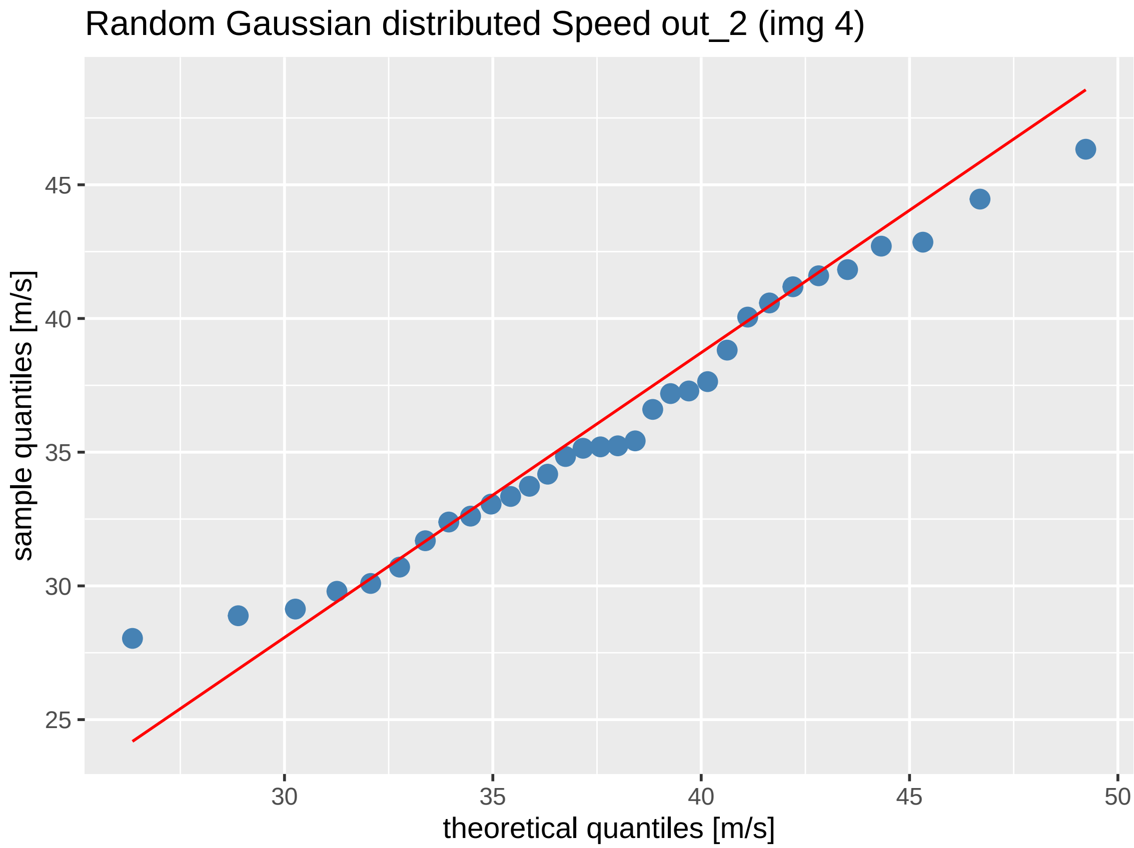
```
rows <- 20
cols <- length(out_2$speed)
random_dnorm_matrix <- matrix(nrow = rows, ncol = cols)
for (i in 1:rows){
  random_dnorm_matrix[i,] <- c(rnorm(cols, mean = nrm_fit_speed_out_2$estimate[1], sd = nrm_fit_speed_ou
}

for (j in 1:rows){
  p <- ggplot2::qplot(sample = random_dnorm_matrix[j,], geom = "blank", xlab = "theoretical quantiles",
    stat_qq(distribution = qnorm, dparams = list(nrm_fit_speed_out_2$estimate[1], nrm_fit_speed_out_2$es
    stat_qq_line(distribution = qnorm, dparams = list(nrm_fit_speed_out_2$estimate[1], nrm_fit_speed_ou
    labs(title = paste0("Random Gaussian distributed Speed out_2 (img ", j, ")"), x = "theoretical quan
 print(p)
}
```

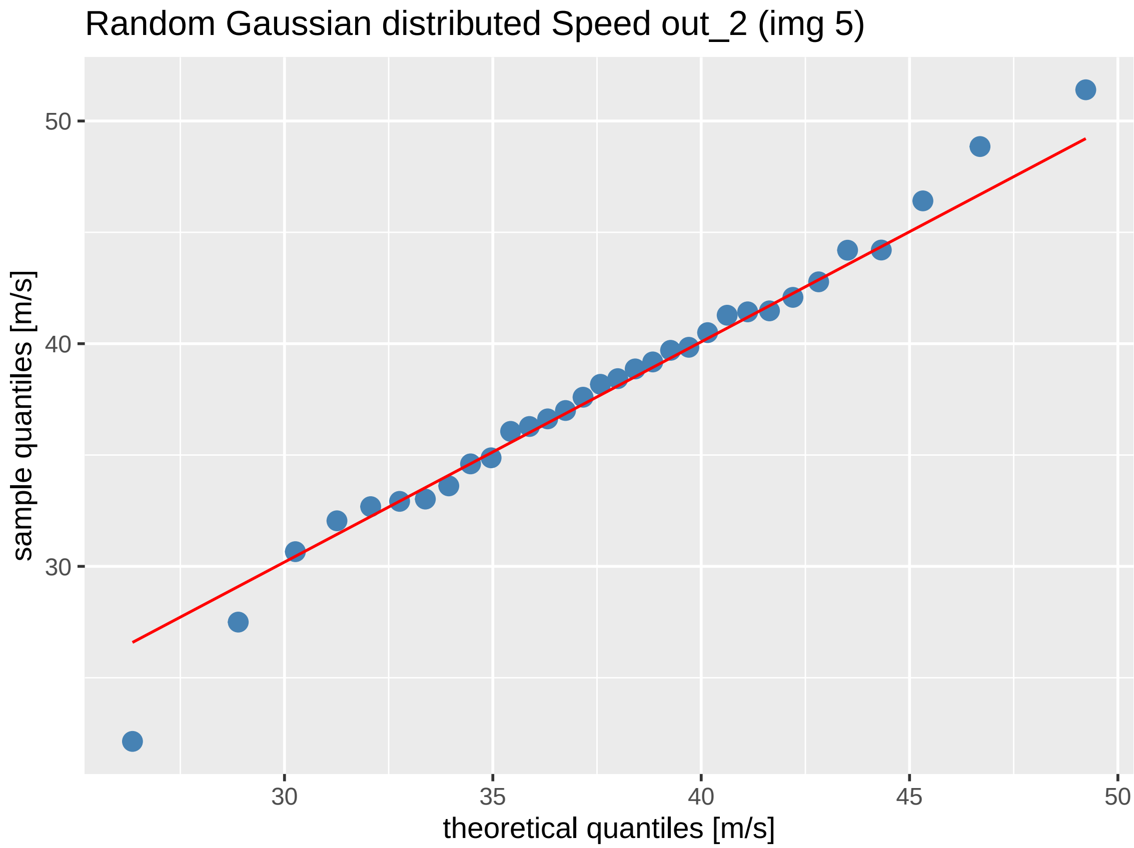


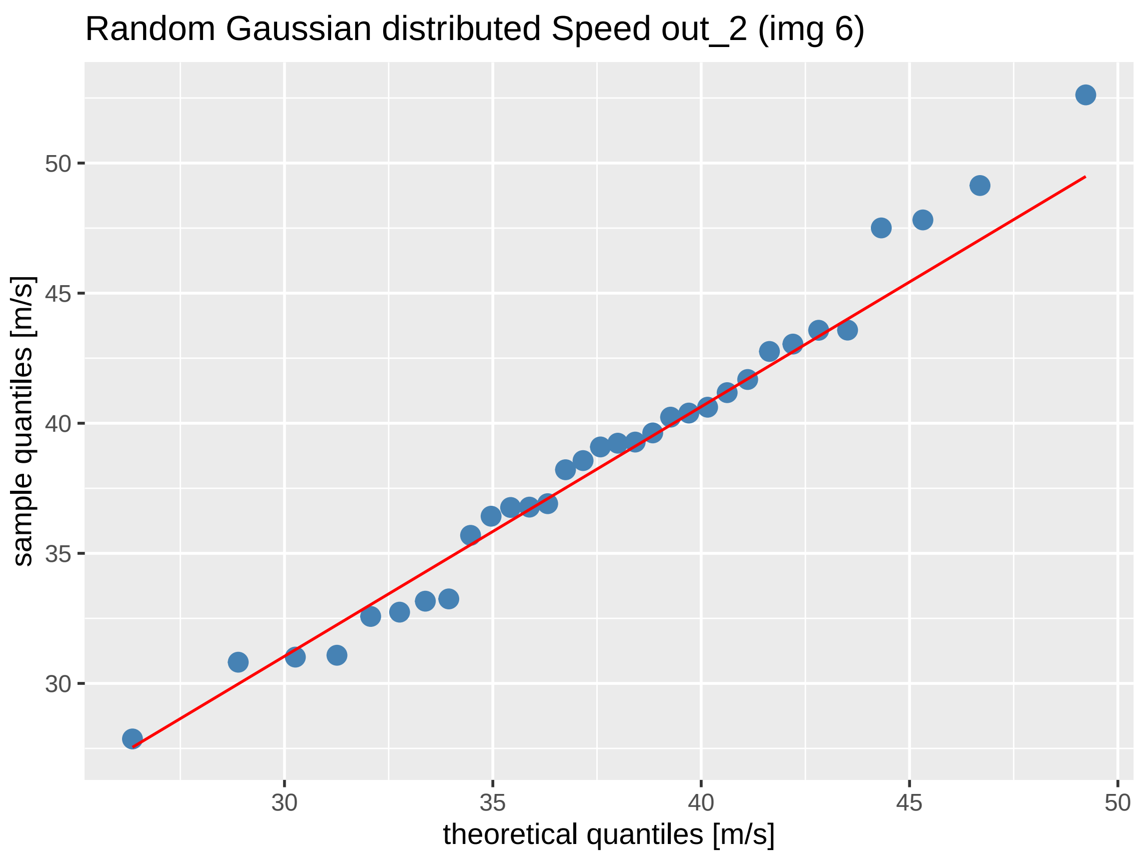

Random Gaussian distributed Speed out_2 (img 2)
sample quantiles [m/s]
50
40
30
30
35
40
45
50
theoretical quantiles [m/s]



Random Gaussian distributed Speed out_2 (img 3)
45
40
35
30
sample quantiles [m/s]
30
35
40
45
50
theoretical quantiles [m/s]

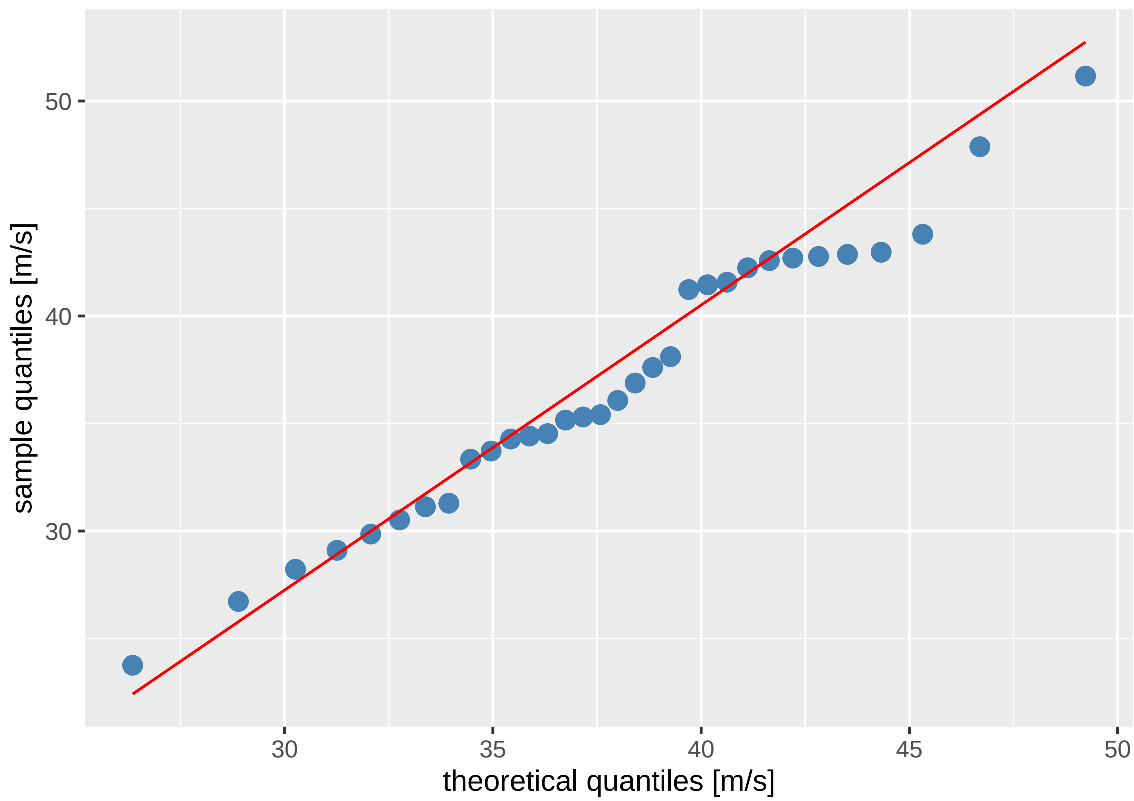

Random Gaussian distributed Speed out_2 (img 4)
45
40
35
30
25
sample quantiles [m/s]
30
35
40
45
50
theoretical quantiles [m/s]

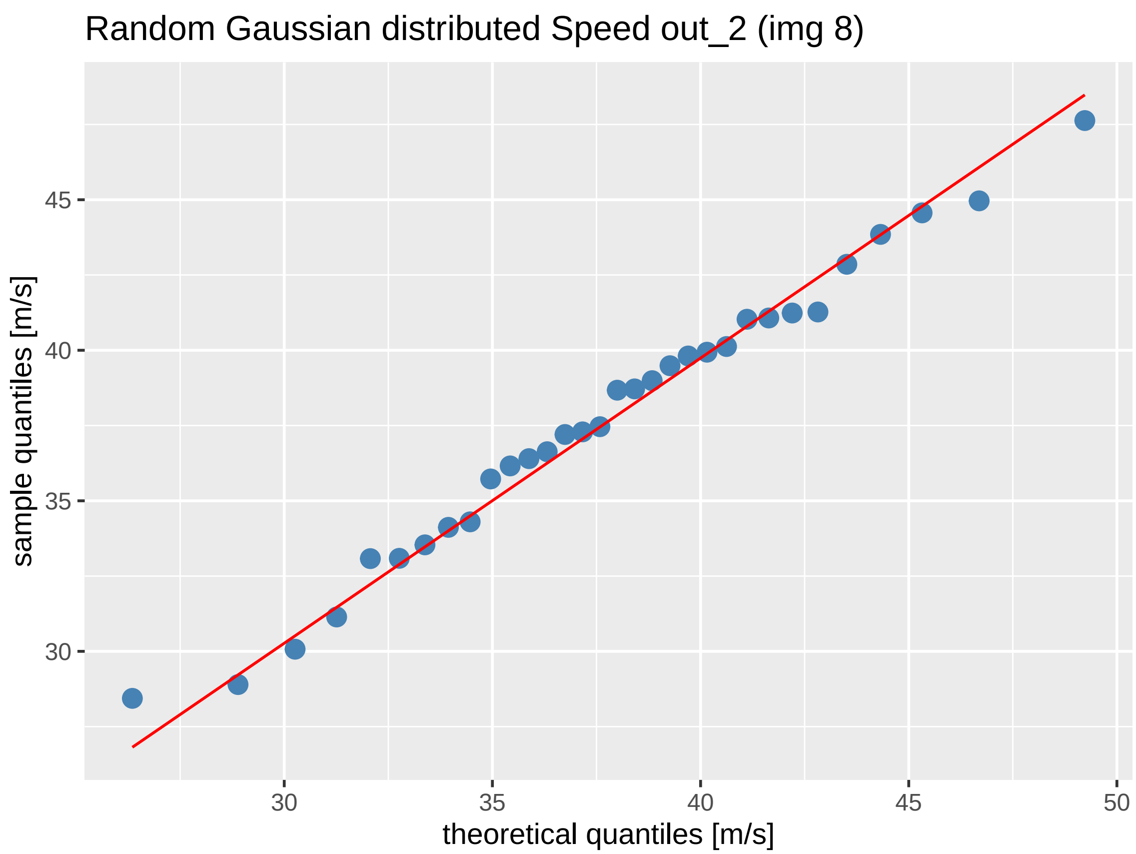

Random Gaussian distributed Speed out_2 (img 5)
sample quantiles [m/s]
50
40
30
30
35
40
45
50
theoretical quantiles [m/s]

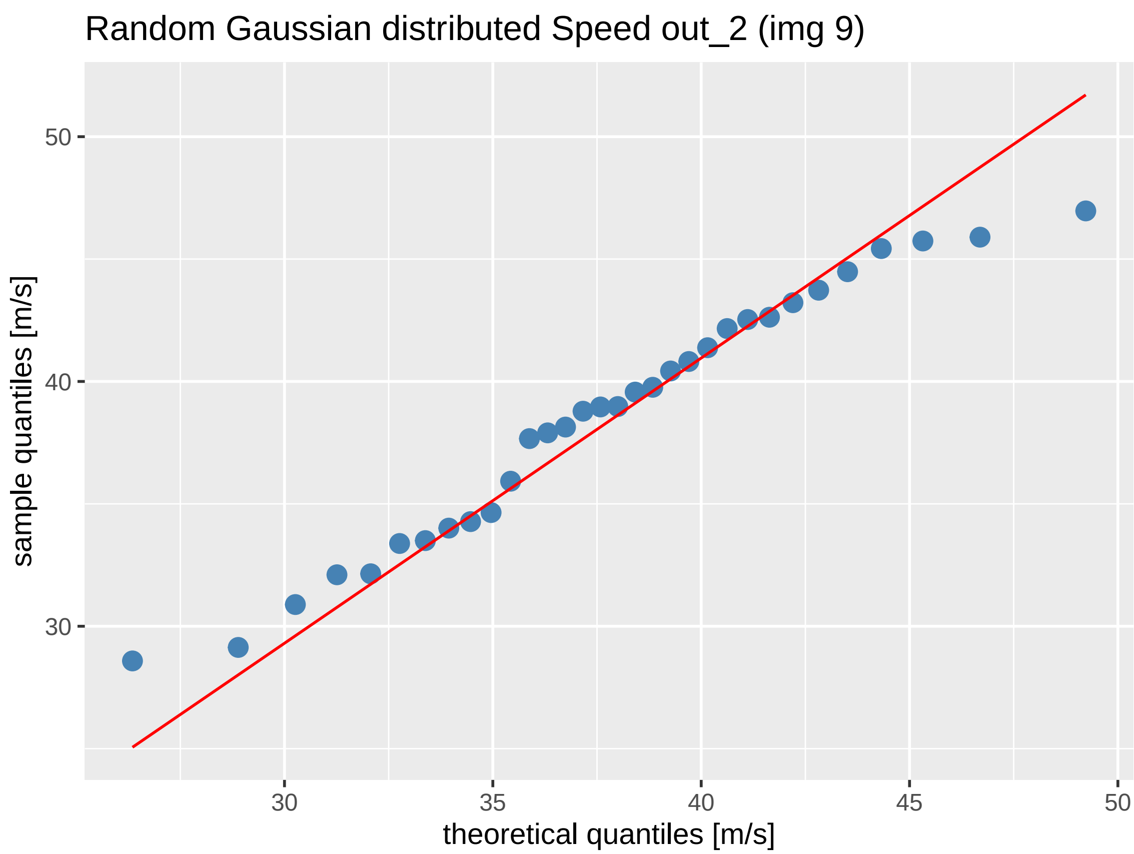


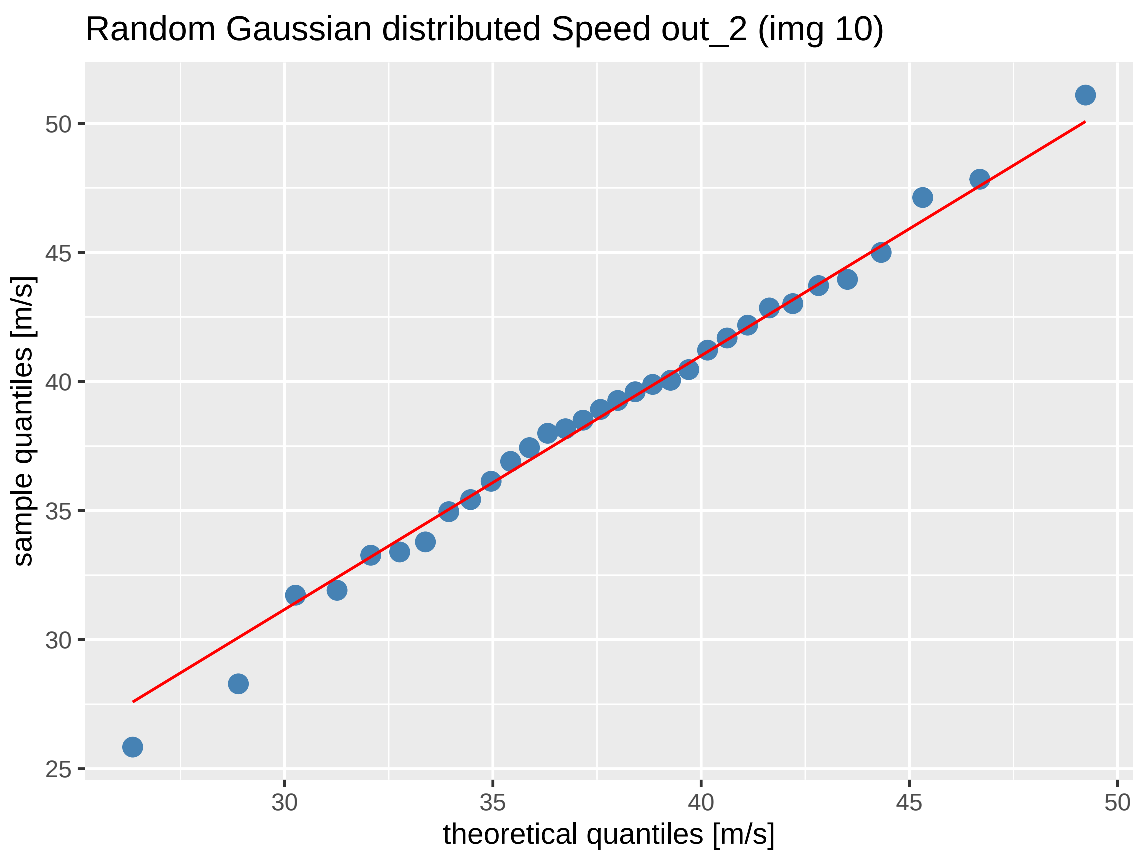
Random Gaussian distributed Speed out_2 (img 6)
sample quantiles [m/s]
theoretical quantiles [m/s]
30
35
40
45
50

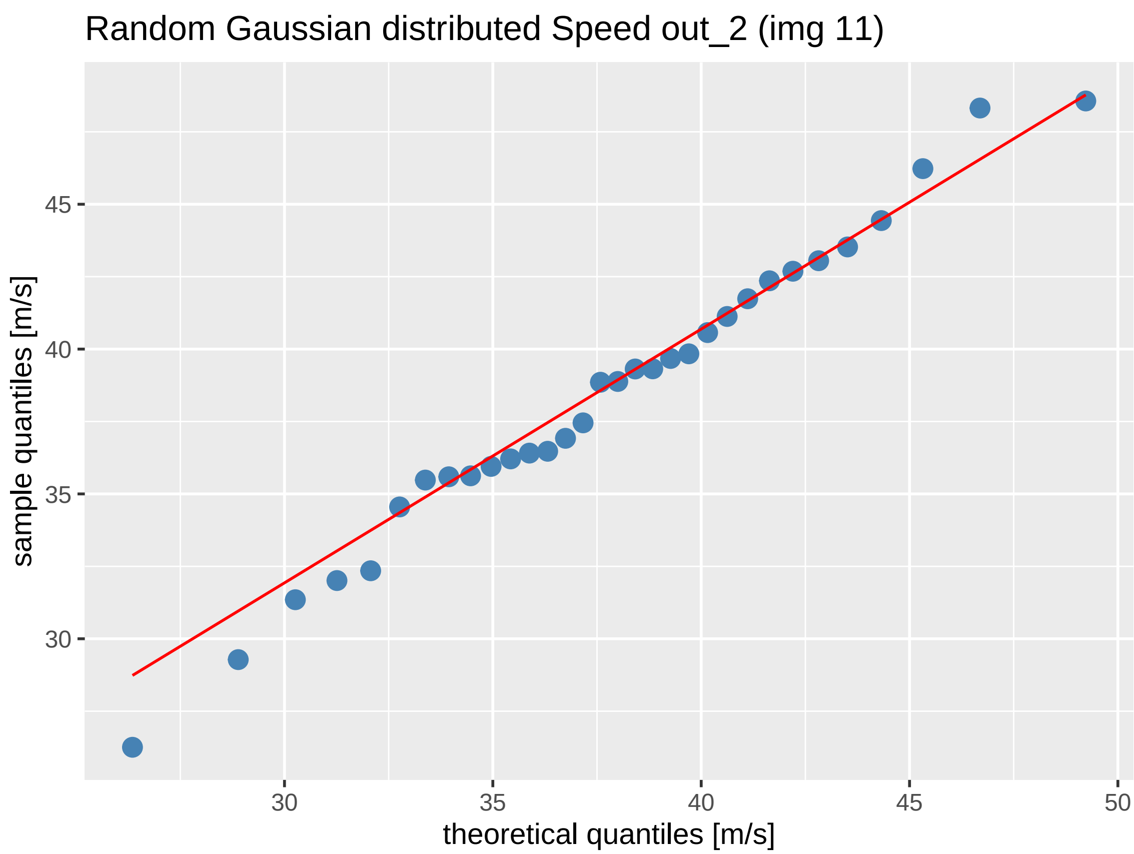
Random Gaussian distributed Speed out_2 (img 7)

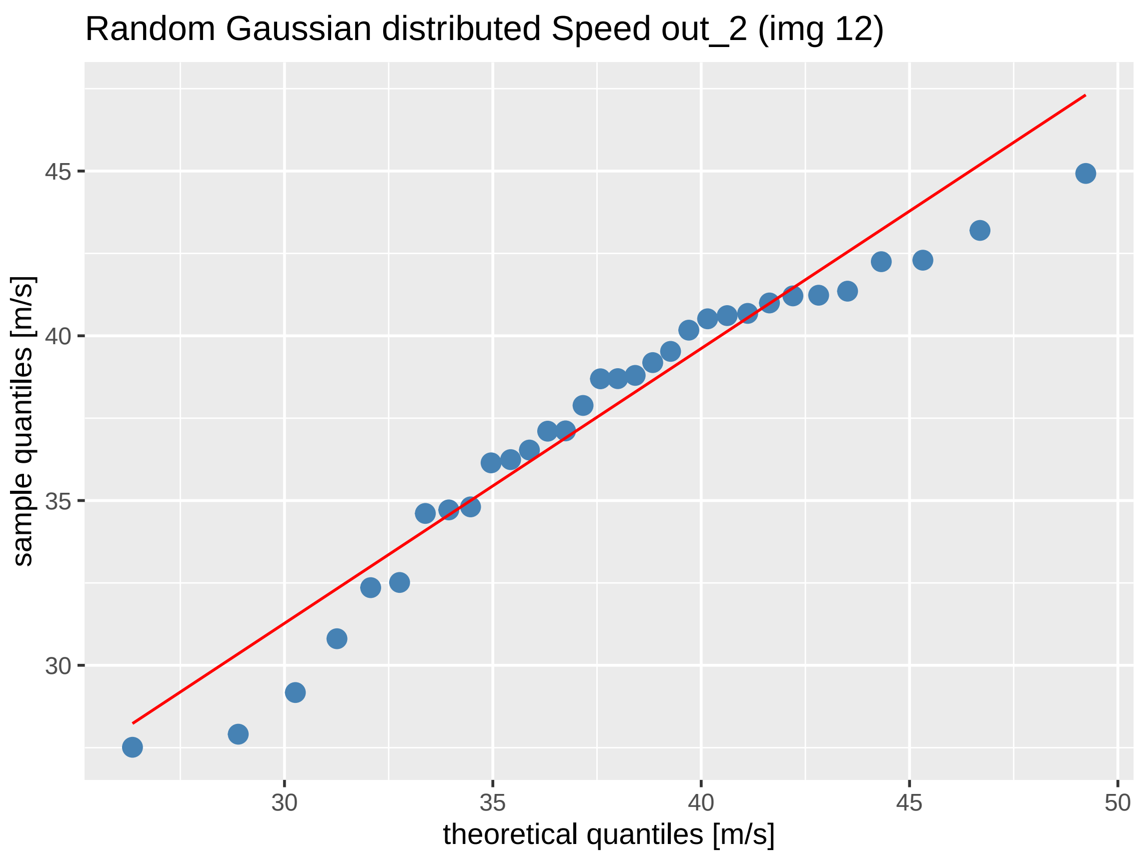


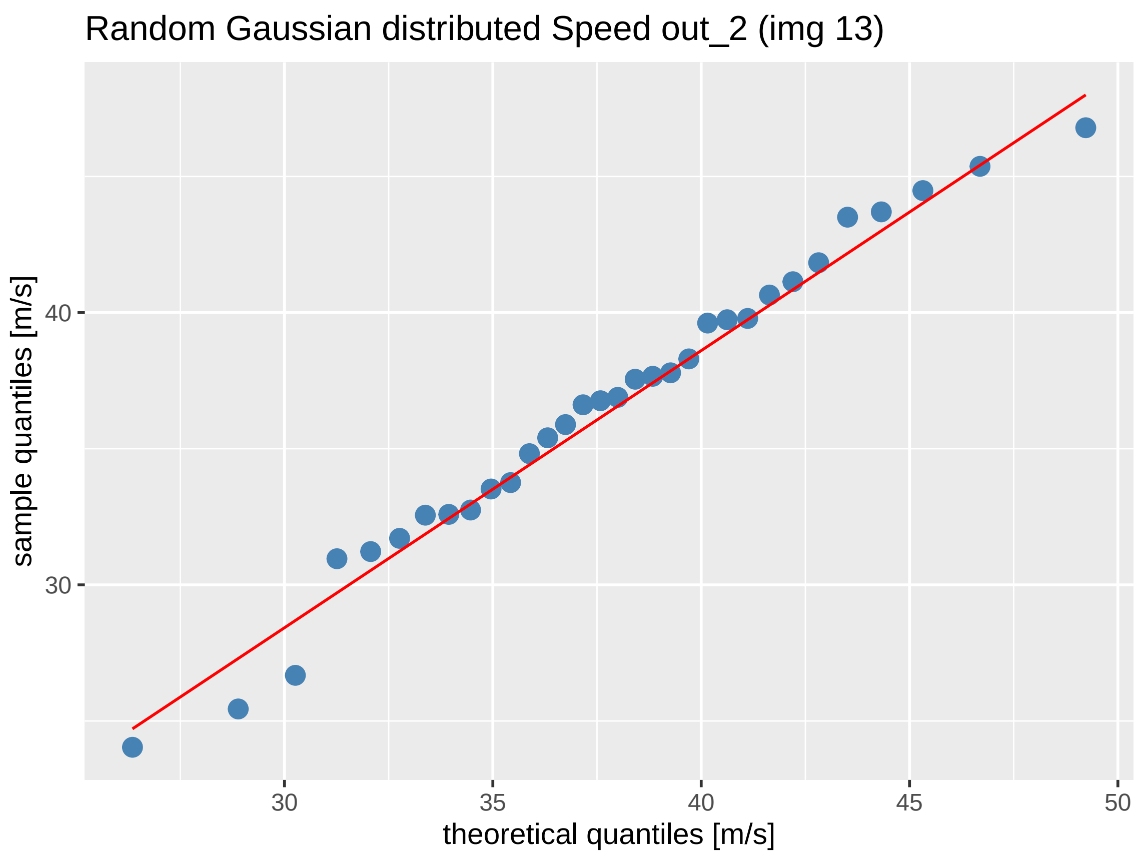

Random Gaussian distributed Speed out_2 (img 8)
sample quantiles [m/s]
45
40
35
30
30
35
40
45
50
theoretical quantiles [m/s]

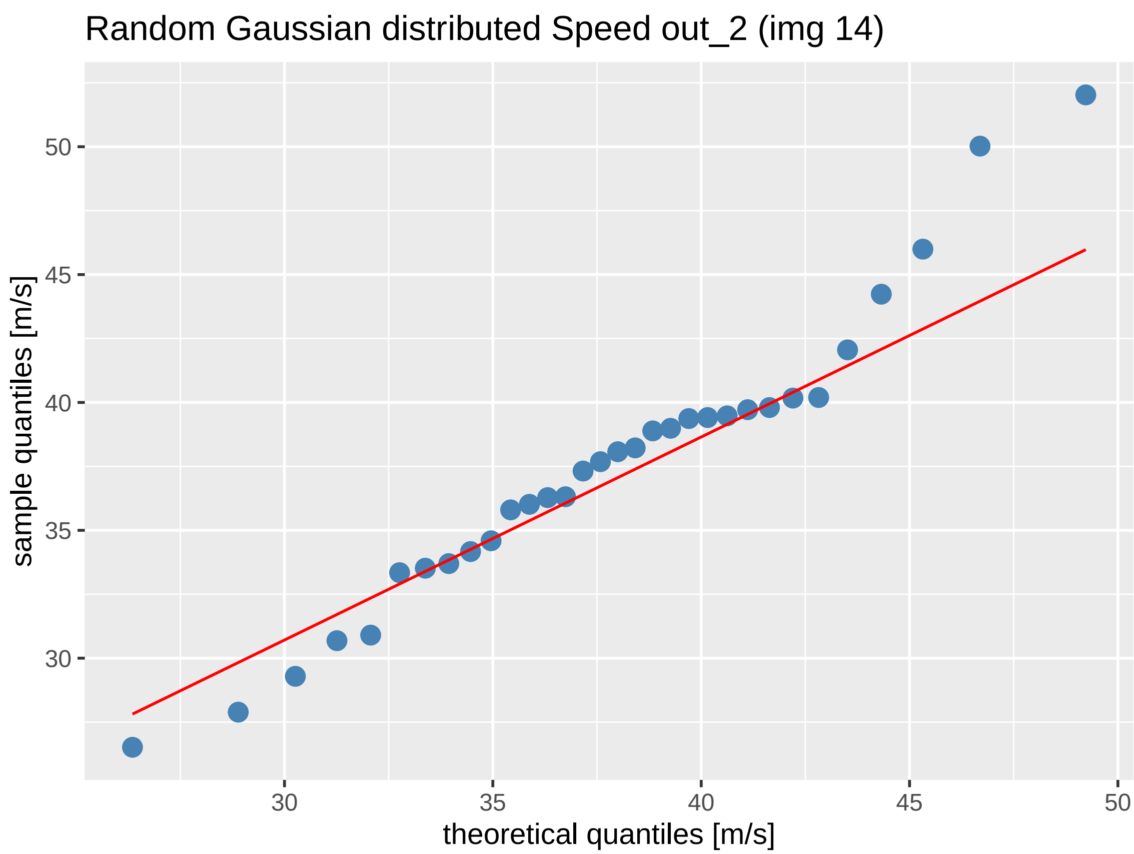

Random Gaussian distributed Speed out_2 (img 9)
sample quantiles [m/s]
50
40
30
30
35
40
45
50
theoretical quantiles [m/s]

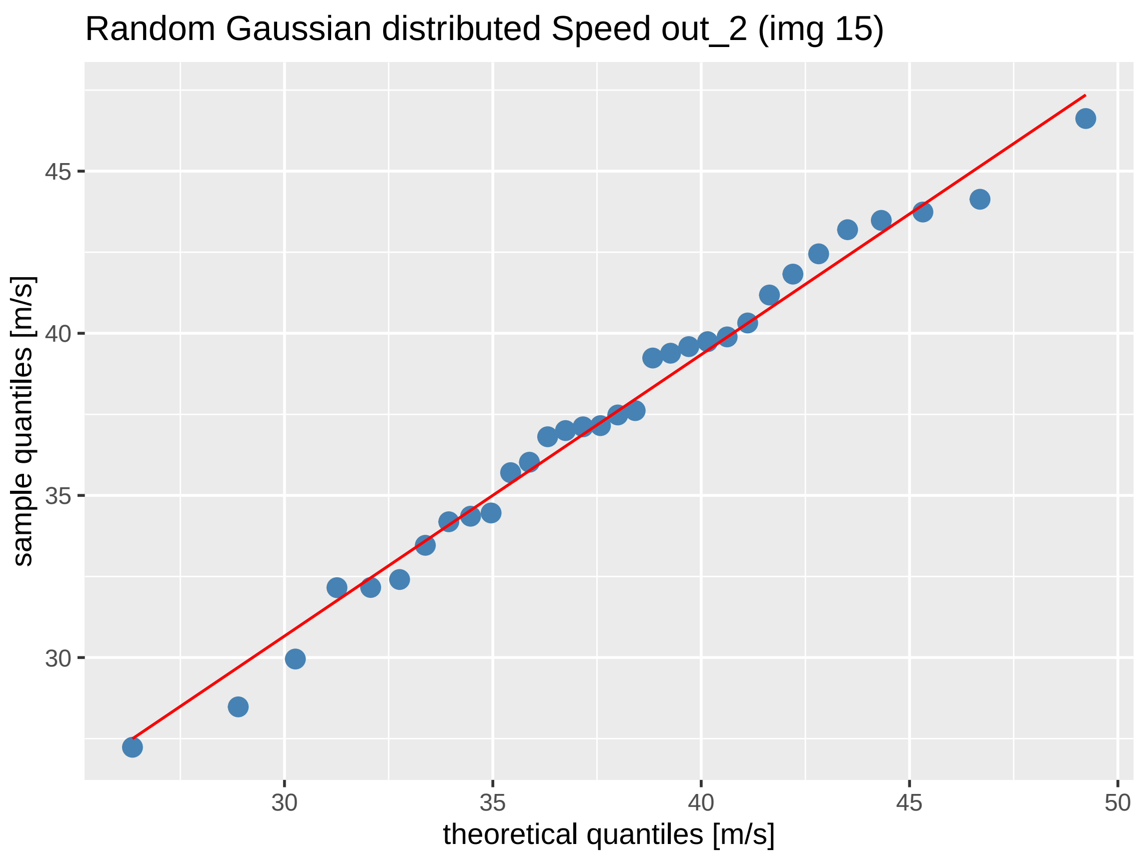

Random Gaussian distributed Speed out_2 (img 10)
sample quantiles [m/s]
theoretical quantiles [m/s]
25
30
35
40
45
50
30
35
40
45
50

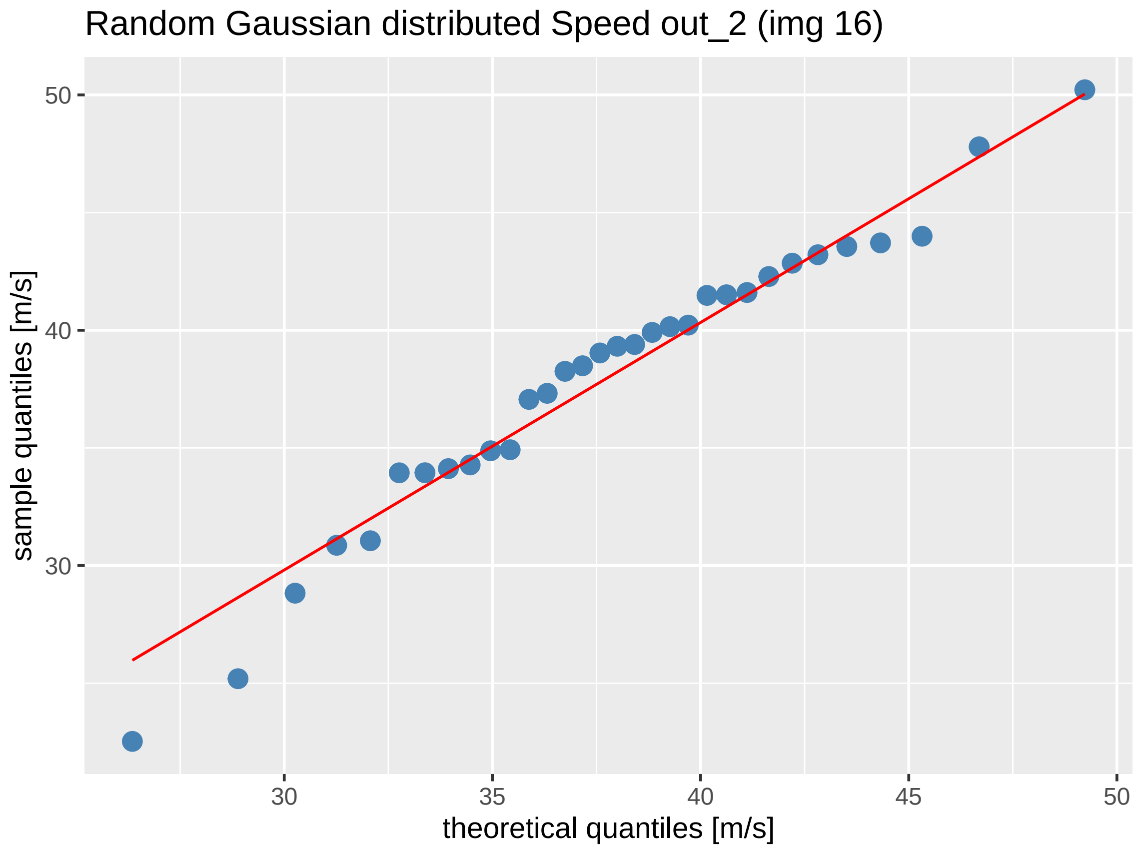

Random Gaussian distributed Speed out_2 (img 11)
sample quantiles [m/s]
theoretical quantiles [m/s]
30
35
40
45
50

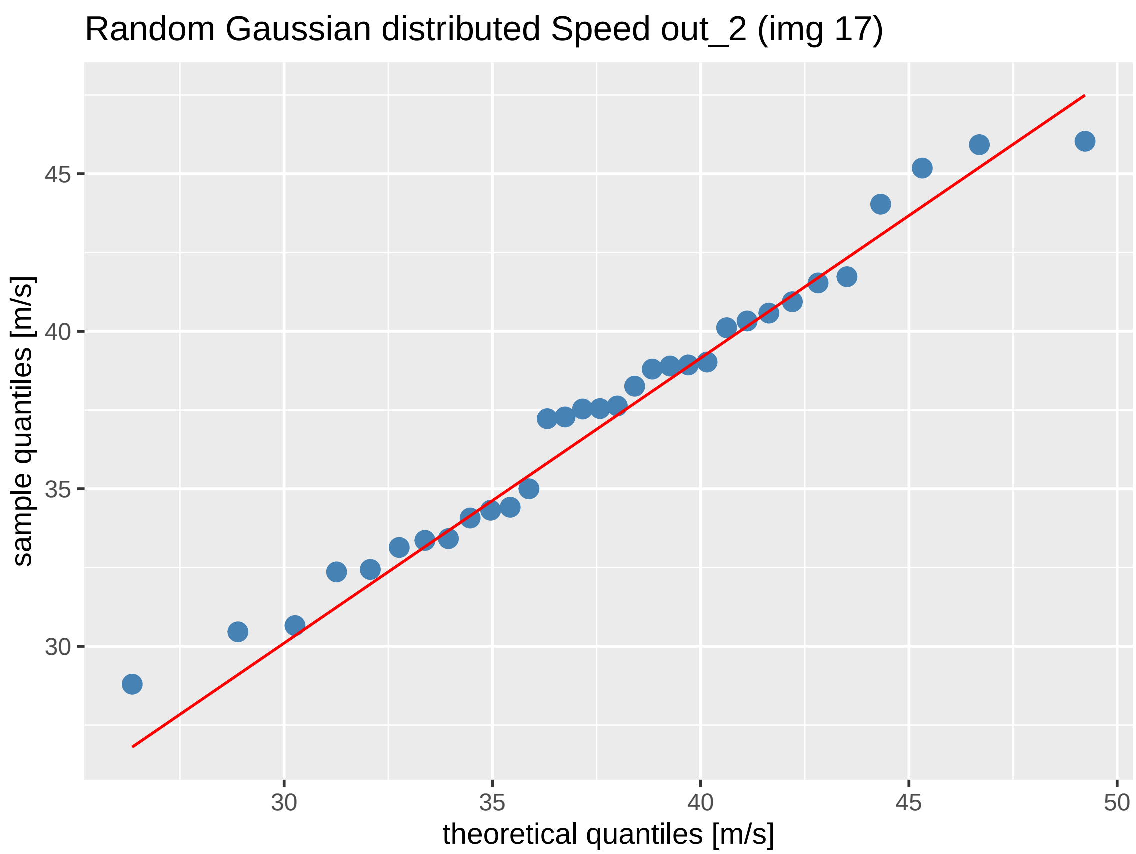

Random Gaussian distributed Speed out_2 (img 12)
sample quantiles [m/s]
45
40
35
30
30
35
40
45
50
theoretical quantiles [m/s]

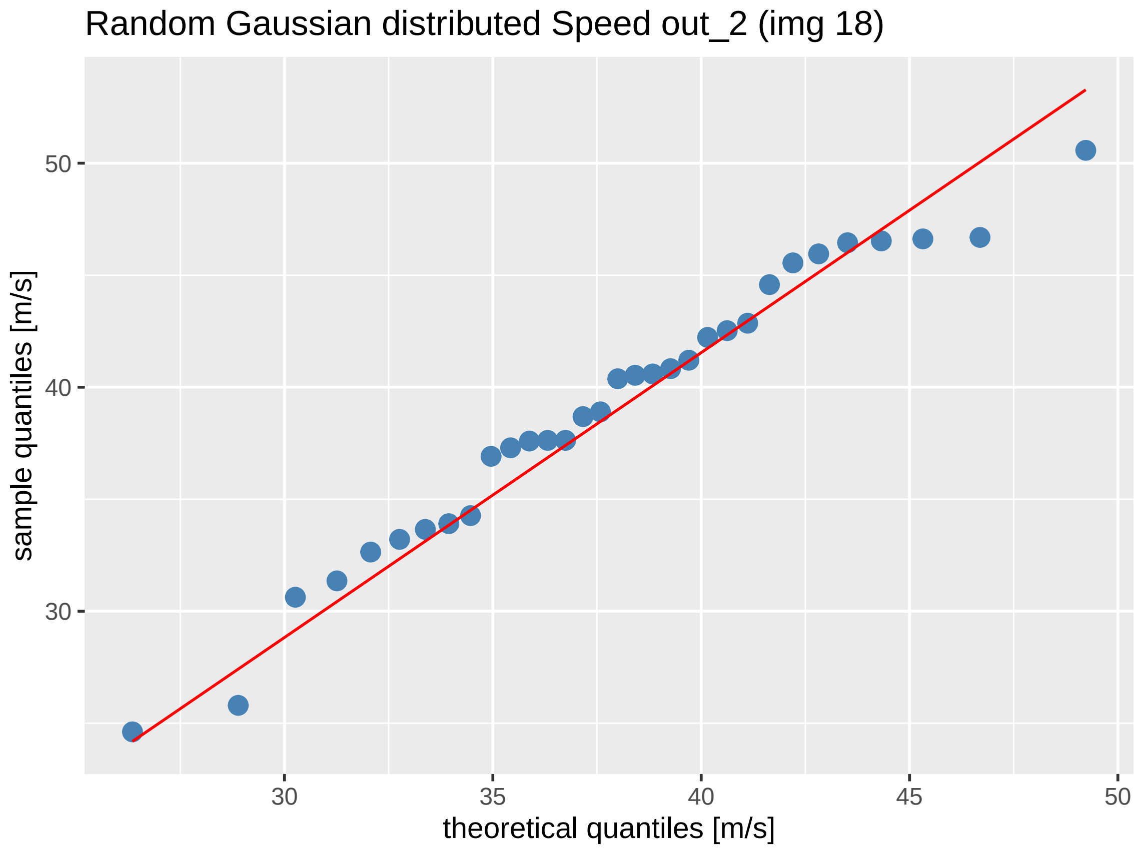

Random Gaussian distributed Speed out_2 (img 13)
sample quantiles [m/s]
40
30
30
35
40
45
50
theoretical quantiles [m/s]

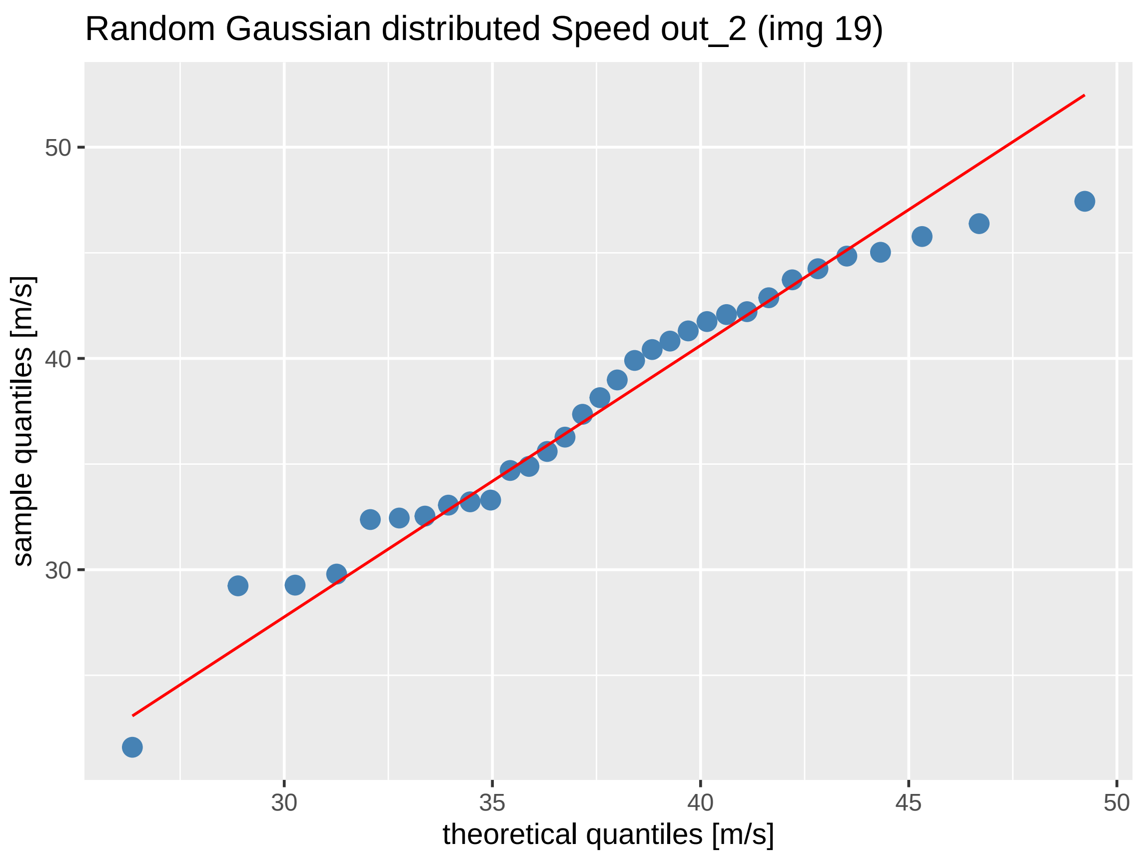

Random Gaussian distributed Speed out_2 (img 14)
50
45
40
35
30
sample quantiles [m/s]
30
35
40
45
50
theoretical quantiles [m/s]

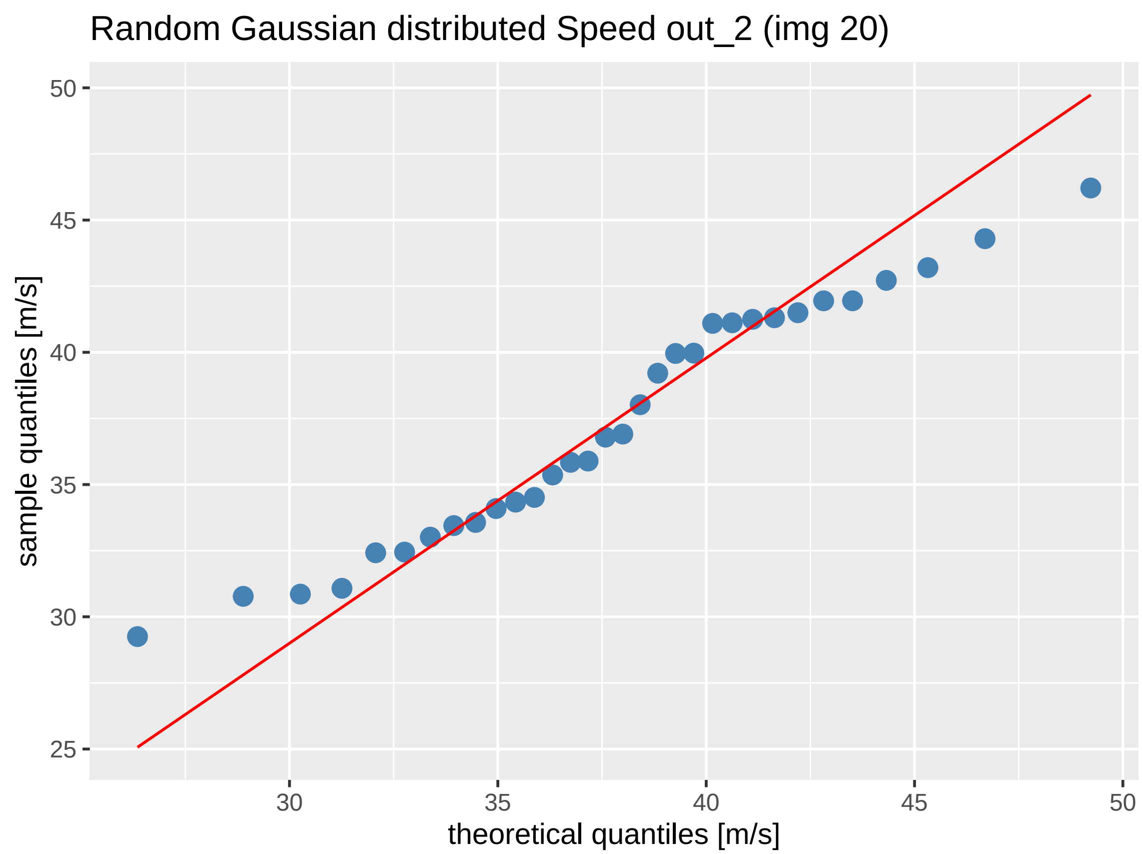

Random Gaussian distributed Speed out_2 (img 15)
sample quantiles [m/s]
theoretical quantiles [m/s]
30
35
40
45
50


Random Gaussian distributed Speed out_2 (img 16)
sample quantiles [m/s]
theoretical quantiles [m/s]
50
40
30
30
35
40
45
50


Random Gaussian distributed Speed out_2 (img 17)
sample quantiles [m/s]
45
40
35
30
30
35
40
45
50
theoretical quantiles [m/s]


Random Gaussian distributed Speed out_2 (img 18)
sample quantiles [m/s]
theoretical quantiles [m/s]
50
40
30
30
35
40
45
50


Random Gaussian distributed Speed out_2 (img 19)
50
40
30
sample quantiles [m/s]
30
35
40
45
50
theoretical quantiles [m/s]



9. Monte-Carlo Simulation

To get $P(S \geq 1)$ where \$s = \$ 'num of rocks through the net in a 1-year time frame', we set up a Monte-Carlo simulation to count, how many times a rock has fallen through the net considering numerous one-year simulations. In addition, we set up multiple simulations in order to compare them with each other. We assume, that the street will get closed after a rock fell through the net. Therefore, the simulation of one year will not continue to get probably another rockfall.

The simulation also takes into account the probability of a car accident, i.e. the probability that a car will be hit by a stone if it falls through the net. To this end, we at the Federal Statistical Office (BfS) have taken into account the driving density per hour on Swiss roads. We assume 3 different cases:

1. all 1600 cars are equally distributed on the day
2. we assume the highest traffic density over every hour of the day according to BfS
3. we consider the hour in which the stone falls down and the traffic density of the BfS

The result of the third variant should be taken with caution, as we take into account the time intervals between two events for the simulation, and not the time of day itself.

In order to obtain appropriate values for a rockfall, we obtain a random value corresponding to the distribution of the individual sources in terms of mass and speed, from which the energy can be calculated $energy[kJ] = 0.5 * m * v^2$. If a stone falls into the net with more than 1000 kJ, then the rock falls onto the street. If rockfall has between 500 and 1000 kJ, then the stability of the net depends on the weight that is already in the net. It breaks through when the weight is more than 2000 kg. We assume that on every day, the net will be cleared on 08:00.

```
rock_event <- function(speed, mass){
  broken <- FALSE
```

```
  energy <- (0.5 * mass * speed^2) / 1000
  if (energy >= 1000){
    broken <- TRUE
  } else if (energy >= 500){
    if (mass_in_net >= 2000){
      broken <- TRUE
    }
  }
  if (broken){
    num_rock_through_net <<- num_rock_through_net + 1
    events_current_year <<- events_current_year + 1
  }
  mass_in_net <<- mass_in_net + mass
  return(broken)
}

net_clearing_process <- function(now, event_time_delta, clearing_time_minute, day_minutes){
  current_time <- now %% day_minutes
  event_time = current_time + event_time_delta
  if (current_time < clearing_time_minute){
    #day_time before clearing_time
    if (event_time >= clearing_time_minute){
      #clearing_time lies between current_time & event_time
      mass_in_net <<- 0
    }
  } else {
    #day_time after clearing_time
    if (event_time >= (day_minutes + clearing_time_minute))
      #clearing_time lies between current_time & event_time
      mass_in_net <<- 0
  }
}

car_hitting <- function(event_time, day_minutes){
  current_time <- event_time %% day_minutes
  #current_hour in integer, 8 O'clock = 08
  current_hour <- base::floor(current_time / 60)
  if (current_hour == 0) {
    current_hour = 24
  }
  #runif generates a random number between 0 and 1. If the number is below the expected value the car g
  hit_number <- runif(1, min = 0, max = 1)
  #Simulation with traffic data of the institute of statistics:
  if (hit_number <= total_in_danger$exp_car_per_hour[current_hour]) {
     car_hit_stat <<- car_hit_stat + 1
  }

  #Simulation with evenly distributed traffic:
  if (hit_number <= 0.008166) {
    car_hit_even <<- car_hit_even + 1
  }

  #Simulation with max traffic througout the day:
```

```
  if (hit_number <= 0.0175843437) {
    car_hit_max <<- car_hit_max + 1
  }

}
#create dataframe for probability counter
pro_counter <<- NULL


monte_carlo_rockfall <- function(num_of_years, clearing_time_hour, simulation_id){

  day_minutes <- 24 * 60
  year_minutes <- day_minutes * 365
  clearing_time_minute <- clearing_time_hour * 60



  for (year in 1:num_of_years){
    current_year <<- year
    events_current_year <<- 0
    mass_in_net <<- 0
    now <- 0
    time_to_next_event_out_1 <- rexp(1, rate = gDistTimeZ1$estimate[1])
    time_to_next_event_out_2 <- rlnorm(1, meanlog = gDistTimeZ2$estimate[1], sdlog = gDistTimeZ2$estima
    while (now < year_minutes){

      if (time_to_next_event_out_1 < time_to_next_event_out_2){
        net_clearing_process(now, time_to_next_event_out_1, clearing_time_minute, day_minutes)
        time_to_next_event_out_2 <- time_to_next_event_out_2 - time_to_next_event_out_1
        now <- now + time_to_next_event_out_1
        time_to_next_event_out_1 <- rexp(1, rate = gDistTimeZ1$estimate[1])
        speed_1 <- rnorm(1, mean = nrm_fit_speed_out_1$estimate[1], sd = nrm_fit_speed_out_1$estimate[2
        mass_1 <- rlnorm(1, meanlog = lnrm_fit_mass_out_1$estimate[1], sdlog = lnrm_fit_mass_out_1$esti
        broken <- rock_event(speed_1, mass_1)
        if (broken){
          car_hitting(now, day_minutes)
          break
        }

      } else {
        net_clearing_process(now, time_to_next_event_out_2, clearing_time_minute, day_minutes)
        time_to_next_event_out_1 <- time_to_next_event_out_1 - time_to_next_event_out_2
        now <- now + time_to_next_event_out_2
        time_to_next_event_out_2 <- rlnorm(1, meanlog = gDistTimeZ2$estimate[1], sdlog = gDistTimeZ2$es
        speed_2 <- rnorm(1, mean = nrm_fit_speed_out_2$estimate[1], sd = nrm_fit_speed_out_2$estimate[2
        mass_2 <- rexp(1, rate = e_fit_mass_out_2$estimate[1])
        broken <- rock_event(speed_2, mass_2)
        if (broken){
          car_hitting(now, day_minutes)
          break
        }
      }
    }
```

```
    yearly_prob_rock_through_net_mc <- num_rock_through_net / year
    yearly_prob_car_hit_stat_mc <- car_hit_stat / year
    yearly_prob_car_hit_even_my <- car_hit_even / year
    yearly_prob_car_hit_max_mc <- car_hit_max / year


    pro_counter <<- rbind(pro_counter, data.frame(current_year, events_current_year, num_rock_through_n
  }
  prob_rock_through_net_mc <- num_rock_through_net / num_of_years
  prob_car_hit_stat_mc <- car_hit_stat / num_of_years
  prob_car_hit_even_mc <- car_hit_even / num_of_years
  prob_car_hit_max_mc <- car_hit_max / num_of_years

  file_pro_counter <- paste0('./RData/pro_counter_', num_of_years,'_years.rda')
  save(pro_counter, file = file_pro_counter)

  log_text <- paste('simulation_id:', simulation_id,
                    '\nrock through net: ', num_rock_through_net,
                    '\nprobability rock through net: ', prob_rock_through_net_mc,
                    '\ncar hit stat:', car_hit_stat, '(Calculated with traffic data of the swiss institu
                    '\nprobability car hit stat:', prob_car_hit_stat_mc,
                    '\ncar hit even:', car_hit_even, '(Calculated with an evenly distributed traffic)',
                    '\nprobability car hit even:', prob_car_hit_even_mc,
                    '\ncar hit max:', car_hit_max, '(Calculated with maximum traffic at all times)',
                    '\nprobability car hit max:', prob_car_hit_max_mc, '\n\n')

  cat(log_text, file = "./Log/simulation.log", append = TRUE)

  result_frame <- data.frame('simulation_id' = simulation_id,
                       'num_rock_through_net' = num_rock_through_net,
                       'prob_rock_through_net_mc' = prob_rock_through_net_mc,
                       'car_hit_stat' = car_hit_stat,
                       'prob_car_hit_stat_mc' = prob_car_hit_stat_mc,
                       'car_hit_even' = car_hit_even,
                       'prob_car_hit_even_mc' = prob_car_hit_even_mc,
                       'car_hit_max' = car_hit_max,
                       'prob_car_hit_max_mc' = prob_car_hit_max_mc,
                       'num_of_years' = num_of_years)
  return(result_frame)

}

simulation_controller <- function(num_of_simulations, num_of_years, clearing_time_hour){
  result <- data.frame('simulation_id' = integer(),
                       'num_rock_through_net' = integer(),
                       'prob_rock_through_net_mc' = double(),
                       'car_hit_stat' = integer(),
                       'prob_car_hit_stat_mc' = double(),
                       'car_hit_even' = integer(),
                       'prob_car_hit_even_mc' = double(),
                       'car_hit_max' = integer(),
                       'prob_car_hit_max_mc' = double(),
                       'num_of_years' = integer())
```

```r
  file_path <- paste0('./RData/monte_carlo_rockfall_', num_of_simulations, '_simulations_', num_of_years
  save(result, file = file_path)
  for (simulation in 1:num_of_simulations){
    simulation_result <- monte_carlo_rockfall(num_of_years, clearing_time_hour, simulation)
    load(file = file_path)
    result <- rbind(result, simulation_result)
    save(result, file = file_path)

    #reset global variables for next simulation
    car_hit_stat <<- 0
    car_hit_even <<- 0
    car_hit_max <<- 0
    num_rock_through_net <<- 0
  }
}

car_hit_stat <- 0
car_hit_even <- 0
car_hit_max <- 0
num_rock_through_net <- 0
mass_in_net <- 0
load(file = './RData/DeltaT1Exponanitial.RData')
load(file = './RData/DeltaT2LogNormal.RData')


#simulation_controller(num_of_simulations = 1, num_of_years = 100, clearing_time_hour = 8)
```

10. Simulation Results

10.1 Result Table

`simulation_id` → 1 - 100 simulations of 100,000 one year runs

`num_rock_through_net` → number of rocks through net in one simulation

`prob_rock_through_net_mc` → $P(S \geq 1)$

`car_hit_stat` → number of car hits in one simulation respecting the day-time of breakthrough and day-time traffic density statistics of the BfS

`prob_car_hit_stat_mc` → $P(B = 1)$ where $b =$ 'car hit during one year' considering `car_hit_stat`

`car_hit_even` → number of car hits in one simulation assuming the traffic density is uniform distributed

`prob_car_hit_even_mc` → $P(E = 1)$ where $e =$ 'car hit during one year' considering `car_hit_even`

`car_hit_max` → number of car hits in one simulation assuming to have max traffic density respecting the statistics of the BfS

`prob_car_hit_max_mc` → $P(M = 1)$ where $m =$ 'car hit during one year' considering `car_hit_max`

`num_of_years` → number of simulated same years in one simulation

```r
load(file = './RData/monte_carlo_rockfall_100_simulations_1e+05_years.rda')
result
```

```
##     simulation_id num_rock_through_net prob_rock_through_net_mc
## 1               1                 1702                  0.01702
## 2               2                 1614                  0.01614
```

```
## 3                3               1561               0.01561
## 4                4               1638               0.01638
## 5                5               1657               0.01657
## 6                6               1517               0.01517
## 7                7               1594               0.01594
## 8                8               1589               0.01589
## 9                9               1644               0.01644
## 10              10               1687               0.01687
## 11              11               1509               0.01509
## 12              12               1558               0.01558
## 13              13               1584               0.01584
## 14              14               1649               0.01649
## 15              15               1600               0.01600
## 16              16               1656               0.01656
## 17              17               1644               0.01644
## 18              18               1624               0.01624
## 19              19               1642               0.01642
## 20              20               1612               0.01612
## 21              21               1600               0.01600
## 22              22               1554               0.01554
## 23              23               1703               0.01703
## 24              24               1645               0.01645
## 25              25               1622               0.01622
## 26              26               1611               0.01611
## 27              27               1682               0.01682
## 28              28               1625               0.01625
## 29              29               1676               0.01676
## 30              30               1516               0.01516
## 31              31               1587               0.01587
## 32              32               1629               0.01629
## 33              33               1623               0.01623
## 34              34               1634               0.01634
## 35              35               1694               0.01694
## 36              36               1590               0.01590
## 37              37               1596               0.01596
## 38              38               1635               0.01635
## 39              39               1636               0.01636
## 40              40               1599               0.01599
## 41              41               1571               0.01571
## 42              42               1648               0.01648
## 43              43               1586               0.01586
## 44              44               1570               0.01570
## 45              45               1611               0.01611
## 46              46               1633               0.01633
## 47              47               1555               0.01555
## 48              48               1575               0.01575
## 49              49               1571               0.01571
## 50              50               1621               0.01621
## 51              51               1617               0.01617
## 52              52               1627               0.01627
## 53              53               1634               0.01634
## 54              54               1637               0.01637
## 55              55               1626               0.01626
## 56              56               1648               0.01648
```

```
## 57            57                1705                0.01705
## 58            58                1572                0.01572
## 59            59                1690                0.01690
## 60            60                1665                0.01665
## 61            61                1636                0.01636
## 62            62                1609                0.01609
## 63            63                1631                0.01631
## 64            64                1638                0.01638
## 65            65                1608                0.01608
## 66            66                1534                0.01534
## 67            67                1636                0.01636
## 68            68                1604                0.01604
## 69            69                1604                0.01604
## 70            70                1652                0.01652
## 71            71                1656                0.01656
## 72            72                1683                0.01683
## 73            73                1583                0.01583
## 74            74                1558                0.01558
## 75            75                1649                0.01649
## 76            76                1611                0.01611
## 77            77                1616                0.01616
## 78            78                1613                0.01613
## 79            79                1600                0.01600
## 80            80                1648                0.01648
## 81            81                1630                0.01630
## 82            82                1659                0.01659
## 83            83                1592                0.01592
## 84            84                1644                0.01644
## 85            85                1636                0.01636
## 86            86                1691                0.01691
## 87            87                1686                0.01686
## 88            88                1657                0.01657
## 89            89                1605                0.01605
## 90            90                1639                0.01639
## 91            91                1559                0.01559
## 92            92                1619                0.01619
## 93            93                1554                0.01554
## 94            94                1629                0.01629
## 95            95                1665                0.01665
## 96            96                1656                0.01656
## 97            97                1595                0.01595
## 98            98                1634                0.01634
## 99            99                1646                0.01646
## 100          100                1571                0.01571
##     car_hit_stat prob_car_hit_stat_mc car_hit_even prob_car_hit_even_mc
## 1             18              0.00018           15              0.00015
## 2             12              0.00012           20              0.00020
## 3             16              0.00016           17              0.00017
## 4             11              0.00011           15              0.00015
## 5             11              0.00011           11              0.00011
## 6              9              0.00009            9              0.00009
## 7             10              0.00010           13              0.00013
## 8             13              0.00013           11              0.00011
## 9             15              0.00015           14              0.00014
```

```
## 10          9        0.00009       10        0.00010
## 11         10        0.00010        9        0.00009
## 12         15        0.00015       15        0.00015
## 13         17        0.00017       15        0.00015
## 14          9        0.00009       12        0.00012
## 15         16        0.00016       20        0.00020
## 16         14        0.00014       11        0.00011
## 17         13        0.00013       18        0.00018
## 18          9        0.00009       12        0.00012
## 19         10        0.00010       12        0.00012
## 20         10        0.00010        9        0.00009
## 21          9        0.00009       11        0.00011
## 22         17        0.00017       15        0.00015
## 23         10        0.00010       14        0.00014
## 24         21        0.00021       21        0.00021
## 25         13        0.00013       17        0.00017
## 26         19        0.00019       20        0.00020
## 27         11        0.00011        8        0.00008
## 28         15        0.00015       14        0.00014
## 29         15        0.00015       12        0.00012
## 30          8        0.00008        9        0.00009
## 31         14        0.00014       13        0.00013
## 32         13        0.00013       12        0.00012
## 33         12        0.00012       13        0.00013
## 34         11        0.00011       18        0.00018
## 35         18        0.00018       15        0.00015
## 36          9        0.00009       14        0.00014
## 37          7        0.00007        7        0.00007
## 38         14        0.00014       14        0.00014
## 39         15        0.00015       15        0.00015
## 40         13        0.00013       18        0.00018
## 41          7        0.00007        6        0.00006
## 42          8        0.00008       12        0.00012
## 43         14        0.00014       17        0.00017
## 44          6        0.00006       10        0.00010
## 45          9        0.00009       12        0.00012
## 46         14        0.00014        9        0.00009
## 47          9        0.00009       12        0.00012
## 48         16        0.00016       15        0.00015
## 49         11        0.00011       14        0.00014
## 50         17        0.00017       13        0.00013
## 51         14        0.00014        9        0.00009
## 52         18        0.00018       19        0.00019
## 53         11        0.00011       13        0.00013
## 54         11        0.00011        8        0.00008
## 55         12        0.00012       16        0.00016
## 56         13        0.00013       13        0.00013
## 57         11        0.00011       15        0.00015
## 58         10        0.00010        8        0.00008
## 59         14        0.00014       14        0.00014
## 60          8        0.00008       11        0.00011
## 61         10        0.00010       10        0.00010
## 62         14        0.00014       12        0.00012
## 63         14        0.00014       11        0.00011
```

```
## 64            15              0.00015           17              0.00017
## 65            18              0.00018           19              0.00019
## 66            12              0.00012           13              0.00013
## 67            11              0.00011           10              0.00010
## 68            12              0.00012            8              0.00008
## 69            16              0.00016           15              0.00015
## 70            12              0.00012           18              0.00018
## 71            14              0.00014           14              0.00014
## 72             9              0.00009           11              0.00011
## 73            11              0.00011           14              0.00014
## 74            10              0.00010            5              0.00005
## 75            14              0.00014           15              0.00015
## 76            13              0.00013           15              0.00015
## 77            11              0.00011           10              0.00010
## 78            13              0.00013           11              0.00011
## 79            14              0.00014           17              0.00017
## 80            12              0.00012           15              0.00015
## 81            14              0.00014           13              0.00013
## 82            12              0.00012           16              0.00016
## 83            10              0.00010           11              0.00011
## 84            15              0.00015           17              0.00017
## 85            16              0.00016           18              0.00018
## 86            13              0.00013           13              0.00013
## 87            13              0.00013           16              0.00016
## 88            13              0.00013           17              0.00017
## 89            11              0.00011           17              0.00017
## 90            12              0.00012           18              0.00018
## 91            10              0.00010           13              0.00013
## 92             9              0.00009            9              0.00009
## 93            13              0.00013           14              0.00014
## 94            13              0.00013           14              0.00014
## 95            12              0.00012           16              0.00016
## 96             8              0.00008           11              0.00011
## 97            14              0.00014           14              0.00014
## 98             8              0.00008           10              0.00010
## 99             8              0.00008           13              0.00013
## 100           10              0.00010           14              0.00014
##     car_hit_max prob_car_hit_max_mc num_of_years
## 1            37             0.00037        1e+05
## 2            36             0.00036        1e+05
## 3            32             0.00032        1e+05
## 4            29             0.00029        1e+05
## 5            32             0.00032        1e+05
## 6            25             0.00025        1e+05
## 7            26             0.00026        1e+05
## 8            26             0.00026        1e+05
## 9            32             0.00032        1e+05
## 10           24             0.00024        1e+05
## 11           19             0.00019        1e+05
## 12           30             0.00030        1e+05
## 13           36             0.00036        1e+05
## 14           26             0.00026        1e+05
## 15           41             0.00041        1e+05
## 16           24             0.00024        1e+05
```

```
## 17              32              0.00032         1e+05
## 18              28              0.00028         1e+05
## 19              29              0.00029         1e+05
## 20              24              0.00024         1e+05
## 21              24              0.00024         1e+05
## 22              36              0.00036         1e+05
## 23              34              0.00034         1e+05
## 24              39              0.00039         1e+05
## 25              39              0.00039         1e+05
## 26              32              0.00032         1e+05
## 27              26              0.00026         1e+05
## 28              27              0.00027         1e+05
## 29              29              0.00029         1e+05
## 30              18              0.00018         1e+05
## 31              35              0.00035         1e+05
## 32              29              0.00029         1e+05
## 33              23              0.00023         1e+05
## 34              32              0.00032         1e+05
## 35              34              0.00034         1e+05
## 36              29              0.00029         1e+05
## 37              22              0.00022         1e+05
## 38              32              0.00032         1e+05
## 39              29              0.00029         1e+05
## 40              25              0.00025         1e+05
## 41              16              0.00016         1e+05
## 42              30              0.00030         1e+05
## 43              34              0.00034         1e+05
## 44              25              0.00025         1e+05
## 45              23              0.00023         1e+05
## 46              27              0.00027         1e+05
## 47              26              0.00026         1e+05
## 48              29              0.00029         1e+05
## 49              34              0.00034         1e+05
## 50              34              0.00034         1e+05
## 51              23              0.00023         1e+05
## 52              35              0.00035         1e+05
## 53              27              0.00027         1e+05
## 54              22              0.00022         1e+05
## 55              27              0.00027         1e+05
## 56              26              0.00026         1e+05
## 57              28              0.00028         1e+05
## 58              26              0.00026         1e+05
## 59              29              0.00029         1e+05
## 60              22              0.00022         1e+05
## 61              24              0.00024         1e+05
## 62              24              0.00024         1e+05
## 63              25              0.00025         1e+05
## 64              32              0.00032         1e+05
## 65              29              0.00029         1e+05
## 66              23              0.00023         1e+05
## 67              29              0.00029         1e+05
## 68              28              0.00028         1e+05
## 69              41              0.00041         1e+05
## 70              40              0.00040         1e+05
```

```
## 71           29          0.00029       1e+05
## 72           22          0.00022       1e+05
## 73           31          0.00031       1e+05
## 74           29          0.00029       1e+05
## 75           33          0.00033       1e+05
## 76           32          0.00032       1e+05
## 77           28          0.00028       1e+05
## 78           31          0.00031       1e+05
## 79           30          0.00030       1e+05
## 80           30          0.00030       1e+05
## 81           28          0.00028       1e+05
## 82           32          0.00032       1e+05
## 83           23          0.00023       1e+05
## 84           31          0.00031       1e+05
## 85           31          0.00031       1e+05
## 86           30          0.00030       1e+05
## 87           29          0.00029       1e+05
## 88           28          0.00028       1e+05
## 89           32          0.00032       1e+05
## 90           29          0.00029       1e+05
## 91           29          0.00029       1e+05
## 92           19          0.00019       1e+05
## 93           27          0.00027       1e+05
## 94           34          0.00034       1e+05
## 95           25          0.00025       1e+05
## 96           24          0.00024       1e+05
## 97           32          0.00032       1e+05
## 98           17          0.00017       1e+05
## 99           30          0.00030       1e+05
## 100          34          0.00034       1e+05
```

10.2 Statistical Characteristics

Information about the data collected by running the simulation 100 times with 100,000 years of simulation each. The values are rounded to two decimal places. Therefore, all `prob_*` are not presented appropriate. Details about the specific probabilities are shown in the next section.

```r
#describeFast(result)
psych::describe(result[-1]) %>%
  dplyr::select(-vars, -n)
```

```
##                                mean    sd    median   trimmed   mad
## num_rock_through_net        1620.36 42.18   1625.50   1621.24 37.81
## prob_rock_through_net_mc       0.02  0.00      0.02      0.02  0.00
## car_hit_stat                  12.33  2.98     12.00     12.21  2.97
## prob_car_hit_stat_mc           0.00  0.00      0.00      0.00  0.00
## car_hit_even                  13.38  3.35     13.50     13.38  3.71
## prob_car_hit_even_mc           0.00  0.00      0.00      0.00  0.00
## car_hit_max                   28.79  5.09     29.00     28.74  4.45
## prob_car_hit_max_mc            0.00  0.00      0.00      0.00  0.00
## num_of_years              100000.00  0.00 100000.00 100000.00  0.00
##                                 min       max range  skew kurtosis   se
## num_rock_through_net        1509.00   1705.00   196 -0.31    -0.08 4.22
## prob_rock_through_net_mc       0.02      0.02     0 -0.31    -0.08 0.00
## car_hit_stat                   6.00     21.00    15  0.32    -0.27 0.30
```

```
## prob_car_hit_stat_mc          0.00      0.00     0  0.32    -0.27 0.00
## car_hit_even                  5.00     21.00    16 -0.02    -0.45 0.34
## prob_car_hit_even_mc          0.00      0.00     0 -0.02    -0.45 0.00
## car_hit_max                  16.00     41.00    25  0.04     0.01 0.51
## prob_car_hit_max_mc           0.00      0.00     0  0.04     0.01 0.00
## num_of_years             100000.00 100000.00     0   NaN      NaN 0.00
```

```
#data doesn't show probability correctly (rounded to 0.00)
```

10.3 Probabilities

Here we present the statistical characteristics of the probabilities we got from the simulations.

```
probs <- matrix(c(mean(result$prob_rock_through_net_mc), sd(result$prob_rock_through_net_mc), median(re
colnames(probs) <- c('mean', 'sd', 'median', 'min', 'max')
rownames(probs) <- c('prob_rock_through_net_mc', 'prob_car_hit_stat_mc', 'prob_car_hit_even_mc', 'prob_
probs <- as.table(probs)
probs
```

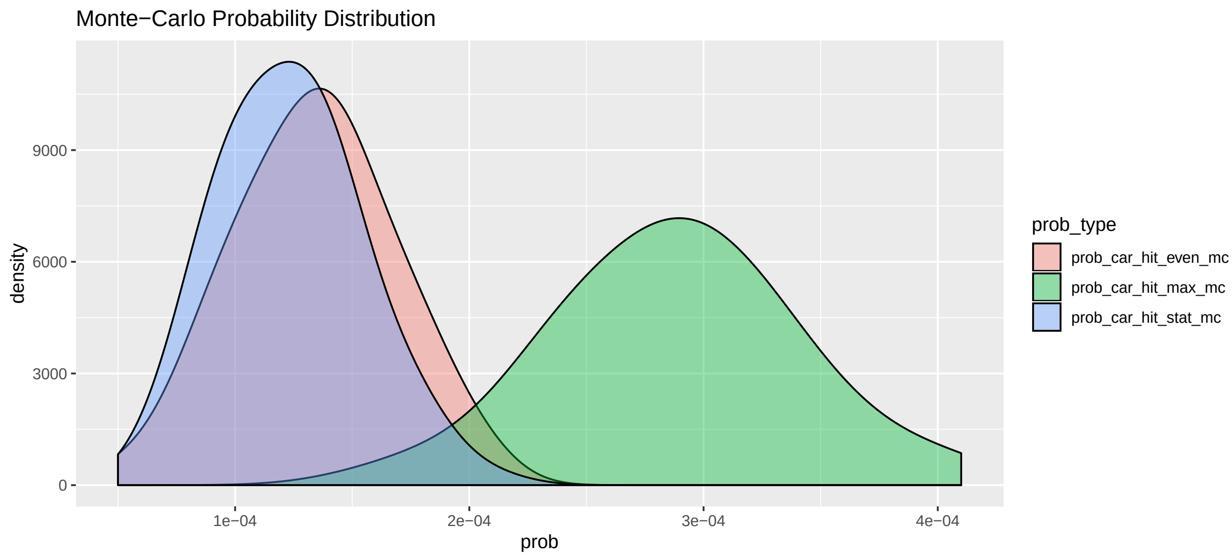
```
##                                  mean           sd       median
## prob_rock_through_net_mc 1.620360e-02 4.217868e-04 1.625500e-02
## prob_car_hit_stat_mc     1.233000e-04 2.978221e-05 1.200000e-04
## prob_car_hit_even_mc     1.338000e-04 3.350803e-05 1.350000e-04
## prob_car_hit_max_mc      2.879000e-04 5.087706e-05 2.900000e-04
##                                   min          max
## prob_rock_through_net_mc 1.509000e-02 1.705000e-02
## prob_car_hit_stat_mc     6.000000e-05 2.100000e-04
## prob_car_hit_even_mc     5.000000e-05 2.100000e-04
## prob_car_hit_max_mc      1.600000e-04 4.100000e-04
```

The mean and median of the probability that a vehicle will be hit is above the predefined threshold 1.e-04 in all three cases.

For security reasons, we strongly recommend to close the road section!

10.4 Simulation Probability Distribution

```
result_dist <- result %>%
  dplyr::select(prob_car_hit_stat_mc, prob_car_hit_even_mc, prob_car_hit_max_mc) %>%
  tidyr::gather(key = 'prob_type', value = 'prob')

ggplot(data = result_dist, mapping = aes(x = prob, group = prob_type, fill = prob_type)) +
  geom_density(adjust = 1.5, alpha = 0.4) +
  labs(title = 'Monte-Carlo Probability Distribution')
```



10.5 Car Hit Prob Simulated vs. Calculated

In a final step we calculate $P(E = 1|S)$ to compare the calculated vs. the simulated value, where we assume that the traffic density is uniform distributed. We calculate it with the mean of `prob_rock_through_net_mc`.

```
prob_car_hit <- mean(result$prob_rock_through_net_mc) * e_car_hit
prob_car_hit
```

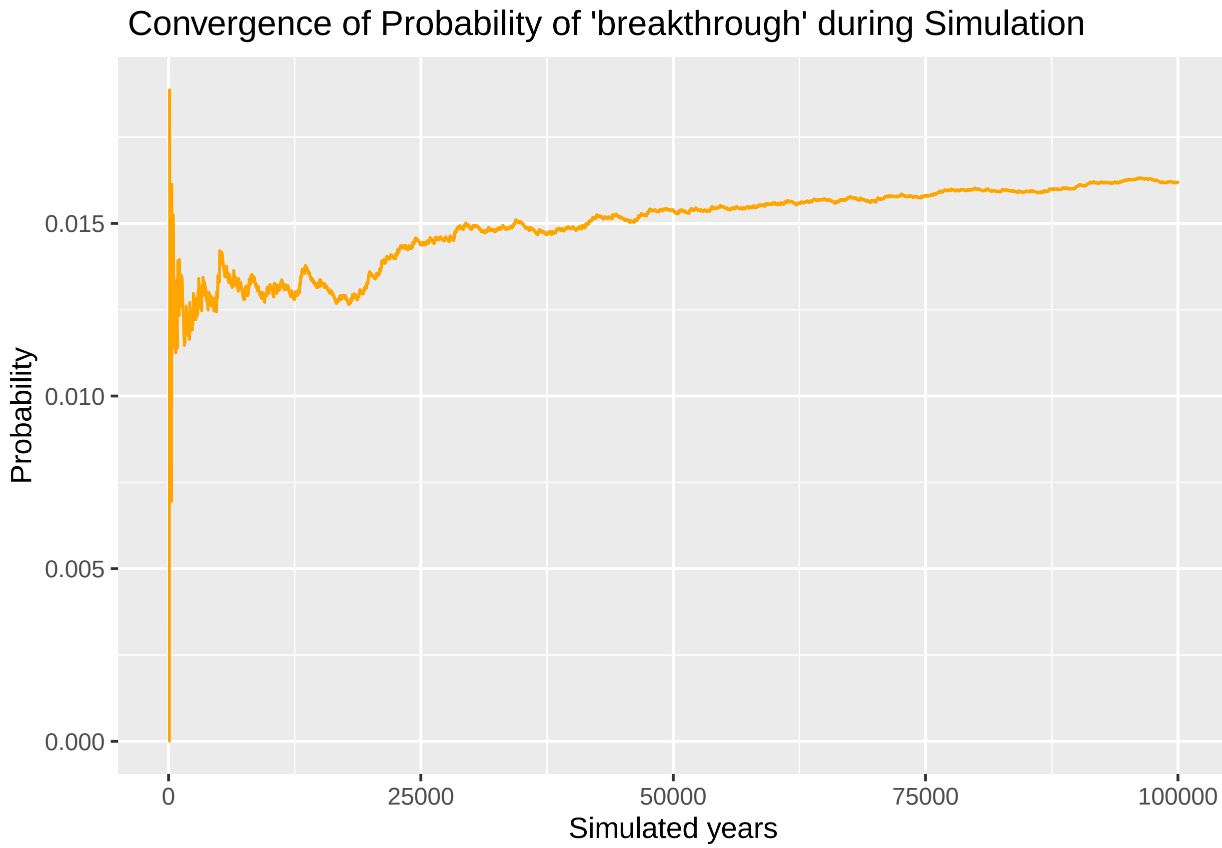
```
## [1] 0.0001497376
```

We can see, that the simulated (1.3380e-04) and calculated (1.4974e-04) probabilites approach each other. Both values are clearly above the pre-defined threshold of 1.e-04.

10.6 Convergence of Probability during Simulation

It is important to know if the monte-carlo simulation converges to a certain value. The below plot shows how the probability levelled of at about 0.017 or 1.7% after about 500 thousend simulated years.

```
#plot convergence of probability as time passes on
load(file = './RData/pro_counter_1e+05_years.rda')

ggplot(data = pro_counter)+
  geom_line(mapping = aes(x = pro_counter$current_year,  y = pro_counter$yearly_prob_rock_through_net_mc
  labs(title = " Convergence of Probability of 'breakthrough' during Simulation", x = "Simulated years"
```



11. Conclusion

By calculating the probability of a rockfall with deadly concequences and also simulating 10 Million years to get a probability of a car getting hit by a rock as well, we can conclude that the probability of a death is to high. The probability is in all cases (calculated and simulated) above the given reference value of 1.e-04. For this reason we advise to close the main road in Schiers until the safety nets are completely replaced, to inform the population about the decision and to publish the findings of the notebook.

Written by:

- Lukas Gehrig, Data Science Student B.Sc. FHNW
- Riccard Nef, Data Science Student B.Sc. FHNW
- Roman Studer, Data Science Student B.Sc. FHNW