Computational statistics - Lab 5

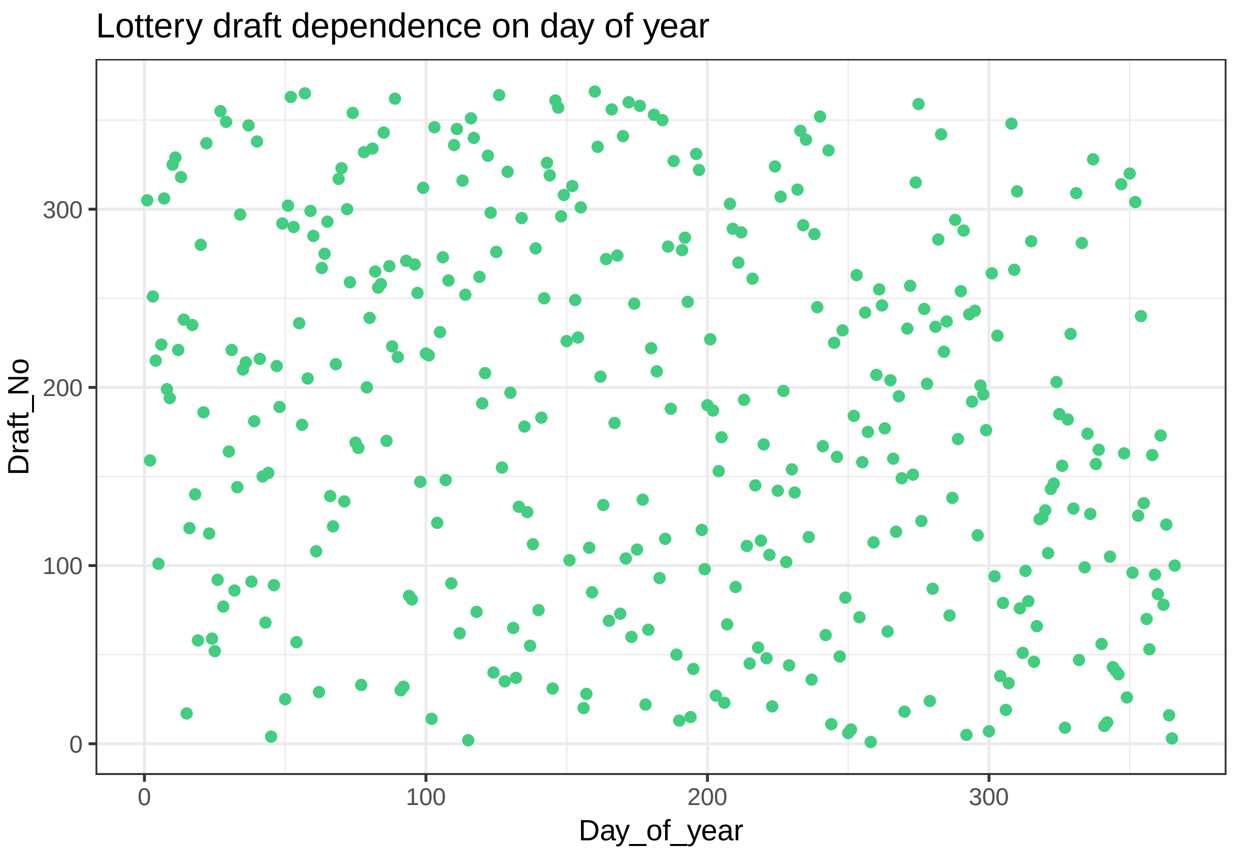
Alexander Karlsson (aleka769)
Lennart Schilling (lensc874)
2019-02-26

Contents

Question 1: Hyphothesis testing

1.1 Scatterplot, X versus Y

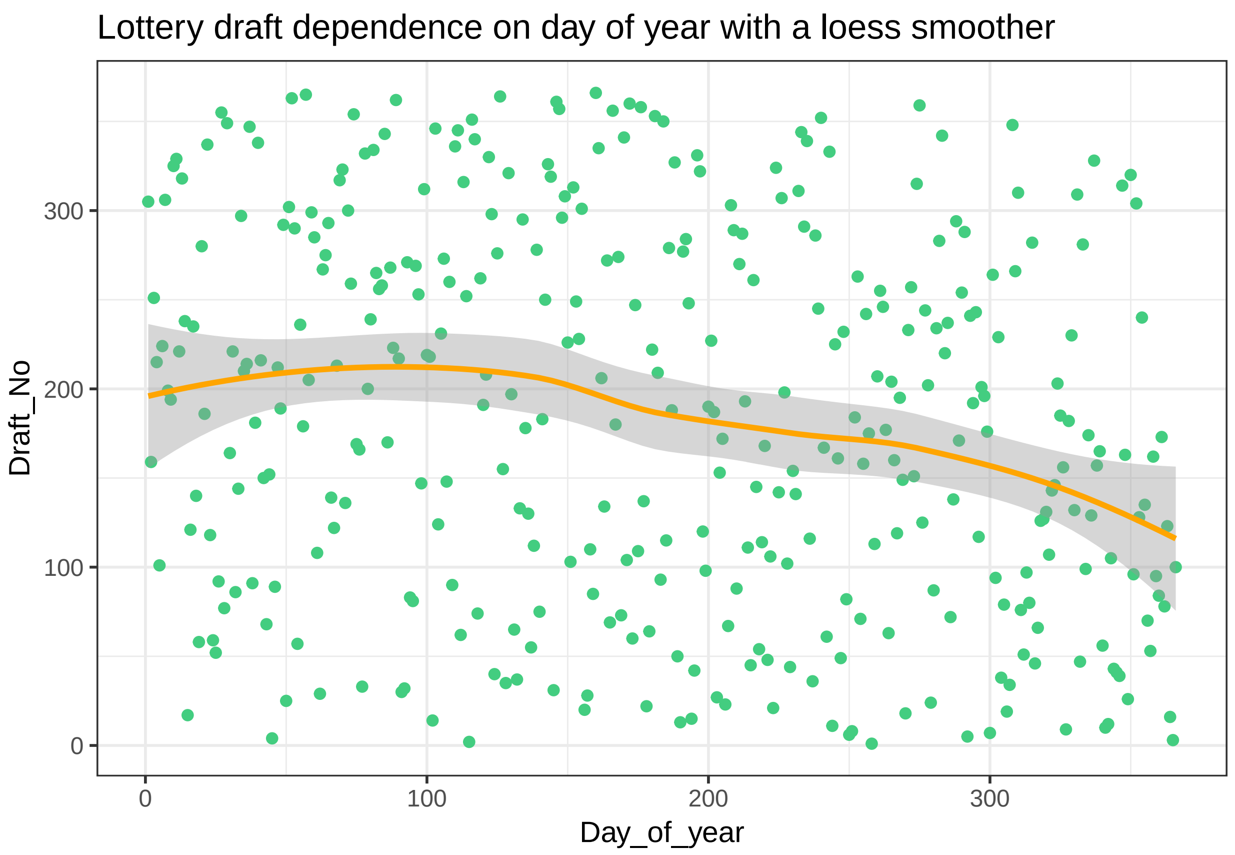
```r
library(dplyr)
library(readxl)
library(ggplot2)
library(gridExtra)
library(boot)
lott = read_excel("lottery.xls")
theme_set(theme_bw())

ggplot(lott, aes(Day_of_year, Draft_No)) +
  geom_point(color = "seagreen3") +
  ggtitle("Lottery draft dependence on day of year")
```



The lottery looks random, as both high and low drafts of the lottery exists in all regions of x.

1.2 Estimate of Yhat with loess curve

```r
ggplot(lott, aes(Day_of_year, Draft_No)) +
  geom_point(color = "seagreen3") +
  geom_smooth(method = "loess", color = "orange") +
  ggtitle("Lottery draft dependence on day of year with a loess smoother")
```



The lottery would pass as random with an ‘eye-test’. However, there seems to be a negative trend in the data, indicating that lower draft numbers occurs more often in later periods of the year.

As we will work with the loess model continuously throughout this lab, it is beneficial to fit a model once and save the predictions (loess curve) once and re-use those results.

```r
fullmod = loess(formula = Draft_No ~ Day_of_year,
                data    = lott)
moddata = data.frame("x"    = lott$Day_of_year,
                     "y"    = lott$Draft_No,
                     "yhat" = fullmod$fitted)
```

1.3 Non-parametric bootstrap with test statistic

The function that is used for all bootstrap samples is defined below. With the model input, one can compute $argmax_X Y(X)$ as well as $argmin_X Y(X)$ as the X and Y variables used as input is stored in the convenient data.frame.

This function is convenient since it can be used outside of the bootstrap iterations as well, but does not have to be defined for the bootstrap to work. Returns the T-statistic from data input with $\hat{y}$.

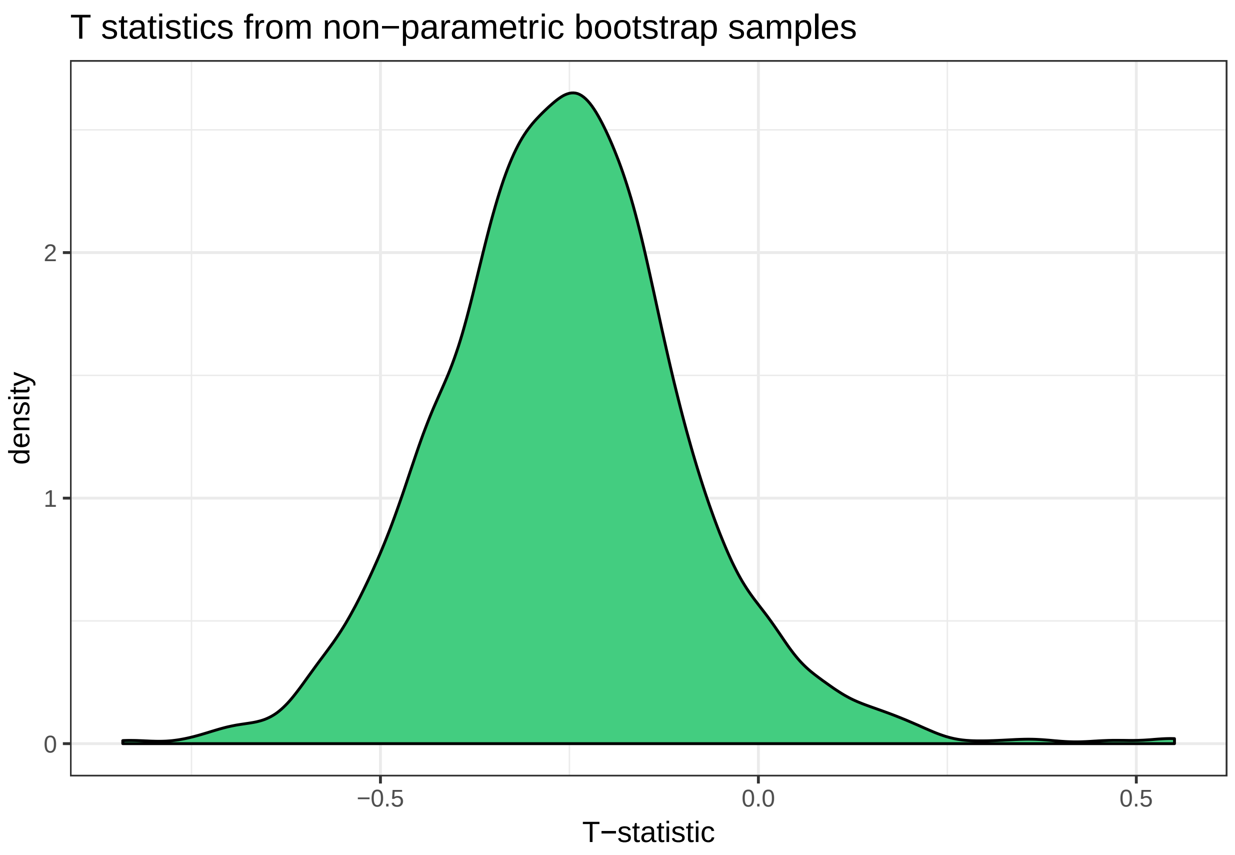
```r
T_stat_fun = function(d){
  Xb     = which.max(d$y)[1]
  Xa     = which.min(d$y)[1]

  T_stat = ( d[Xb, "yhat"] - d[Xa, "yhat"] ) /
    ( d$x[Xb] - d$x[Xa] )
  return(T_stat)
}
```

The function that `boot` requires as input is defined below. This function `f` calls on the `T_stat_fun` that is defined above for actual test statistic computations. This function needs to be defined in order for `boot` to work and has some kind of built-in algorithm for sampling the `ind` argumnt from `data`.

```r
f = function(data,ind){
  mdata   = as.data.frame(data[ind,])
  mod     = loess(formula = y ~ x,
                  data    = mdata)

  mdata$yhat = mod$fitted
  T_stat     = T_stat_fun(mdata)
  return(T_stat)
}
```

The loess-model is recalculated for each sample, because we want to see whether or not this T-stastistic will deviate under iterative sampling. The model-data.frame is created and used as input for calculating T-statistics. With these functions defined, we can now do the actual bootstrapping.

```r
set.seed(123456)
b = boot(data      = moddata,
         statistic = f,
         R         = 2000,
         sim       = "ordinary",
         stype     = "i")

ggplot() + labs(title    = "T statistics from non-parametric bootstrap samples",
                x        = "T-statistic") +
  geom_density(aes(x = b$t), fill = "seagreen3")
```



The distribution is mostly negative, and with the following hyphotheses we eventually define our p-value.

$$\begin{aligned} H_0 &: \quad T_{boot} \geq 0 \\ H_a &: \quad T_{boot} < 0 \end{aligned}$$

Since this test does not include any known distribution, we can not check for critical values in a table or similar. The p-value is instead calculated by checking the percentage of T-statistics from our boot sample that falls into the null hyphothesis.

$$p\text{-}value = P(Reject\ H_0|H_0\ true) = P(T_{boot} > 0)$$

```r
# P-value:
mean(b$t > 0)
```

```
## [1] 0.0595
```

Another interpretation of the p-value is; *how likely is it that we have a positive trend in the data (given that the null hyphothesis is true)?*. The probability of roughly 6 % means we are quite certain that there is no positive trend in the data. We would be able to reject the null hypothesis with $\sim$ 94 percent confidence.

1.4 Permutation test

Implement a function depending on data and B that tests the hypothesis:

$$\begin{aligned} H_0 &: \quad T_{data} = T_{perm} \quad (Lottery\ is\ random) \\ H_1 &: \quad T_{data} \neq T_{perm} \quad (Lottery\ is\ non\text{-}random) \end{aligned}$$

With this test we want to test the null hyphothesis;

$$p\text{-}value = P(Reject\ H_0|H_0\ true) = P(|T_{perm}| > |T_{data}|),$$

with B = 2000 iterations. Since this is a two-sided test, we use absolute values for the T_{perm}-distribution and the T_{data}-value.

```r
perm_fun = function(data, B){
  # Compute n once, not for every iteration!
  n = nrow(data)

  # Full model t statistics:
  tdata = T_stat_fun(data)

  # Reshuffle loop:
  tperm = sapply(X = 1:B, FUN = function(b, d, d_opt, n){
    # New shuffled X --> argmin/max_x Y(X) will change!
    # Also, shuffling happens from d every time, not d_opt,
    # which is used for calculating t!!!
    d_opt$x  = sample(x = d$x, size = n, replace = F)

    # Add yhat column to d_opt to calculate T-statistics:
    d_opt$yhat = loess(formula = y ~ x,
                       data    = d_opt)$fitted

    # T-statistics based on reshuffled data:
    T_stat_fun(d_opt)
  }, d = data, d_opt = data, n = n) %>% t()

  # P-value:
  # mean(tperm %>% abs() > tdata %>% abs())
  sum(tperm %>% abs() > tdata %>% abs())/B
  # mean(tperm %>% abs() > tdata %>% abs(), na.rm = TRUE)
}

set.seed(123456)
perm_fun(data = moddata, B = 2000) # P-value
```

```
## [1] 0.0955
```

According to the test statistics we have around 9 % of the data that is at least as extreme as our observed data from the full model. The corresponding p-value means we can be fairly confident to reject the null hyphothesis.

1.5 Power test

Initially, a generator function is created, mostly for readable code.

```r
generator = function(x, alpha){
  n    = length(x)
  data.frame(a = alpha,
             b = rnorm(n, 183, 10),
             x = x) %>%
    apply(., MARGIN = 1, FUN = function(d){
      max(0, min(d[1]*d[3] + d[2], 366))
    })
}
```

The `powerfun`-function calls our `generator` function and calculates the p-value as in assignment 1.4. There is in other words the same comparison as in 1.4, except this time we generate new data for each iteration (over alpha). Since we dont know the distribution under H_a, the crude estimate of power is simply:

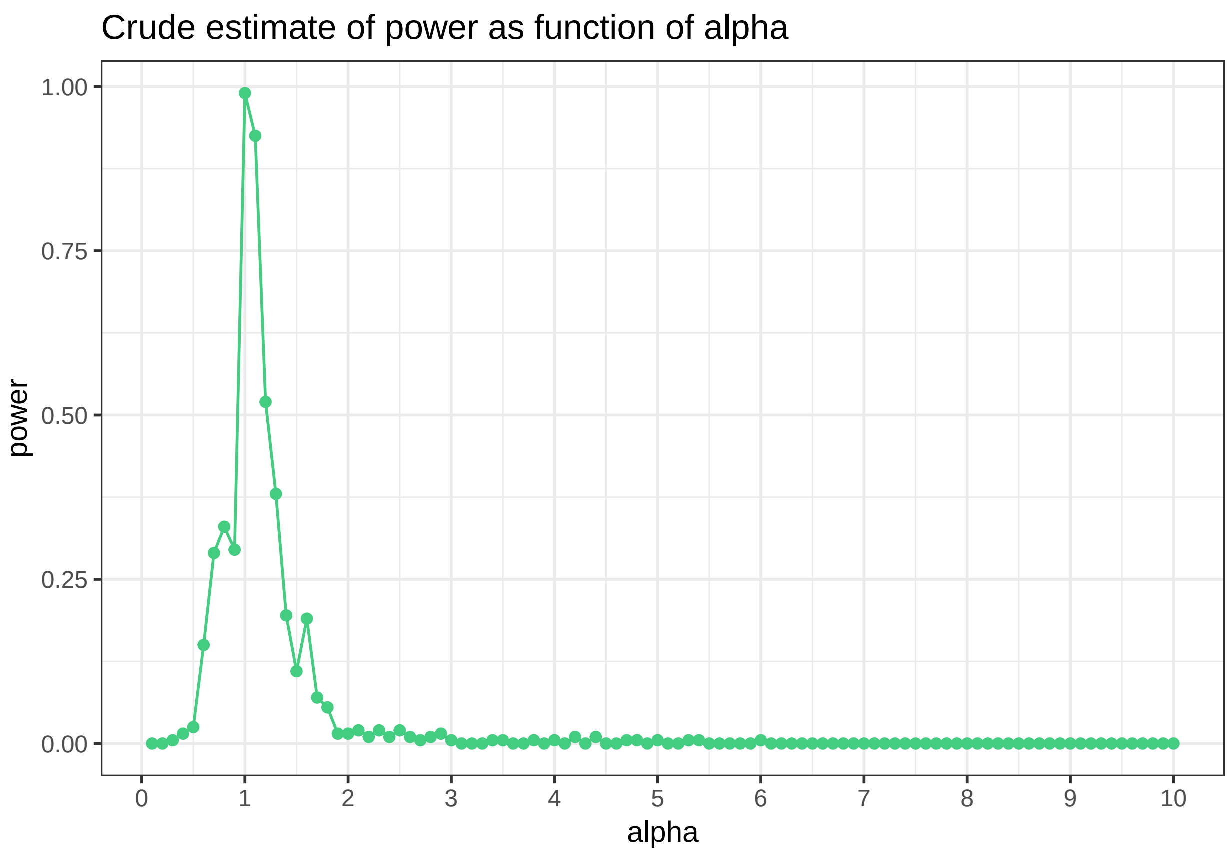
$$Power = P(Reject\ H_0|H_1\ \ True) \approx 1 - p\text{-}value$$

.

```
df = data.frame("Generated_y"   = numeric(),
                "Fitted_y"      = numeric(),
                "Original_x"    = numeric(),
                "current_alpha" = numeric())

powerfun = function(alpha,data,storeindices){
  pw = NULL
  for (i in alpha){
    td = data
    td$y = generator(x = data$x, alpha = i)
    pw = c(pw,perm_fun(data = td, B = 200))

    # Store selected values of x, y, yhat and alpha for plot...
    if(i %in% storeindices){
      df <<- rbind(df,data.frame("Generated_y"   = td$y,
                                 "Fitted_y"      = loess(y ~ x, data = td)$fitted,
                                 "Original_x"    = td$x,
                                 "current_alpha" = i))
    }
  }
  return(pw)
}
```

Below, we call our functions for all alphas.

```
alpha = seq(.1, 10, by = .1)
store = c(.1, .5, 1, 2, 4, 6)
set.seed(123456)
pw = powerfun(alpha, moddata, store)

data.frame("alpha" = alpha, "power" = pw) %>%
  ggplot(., aes(alpha, power)) +
  geom_line(color = "seagreen3") +
  geom_point(color = "seagreen3") +
  scale_x_continuous(breaks = 0:10) +
  ggtitle("Crude estimate of power as function of alpha")
```



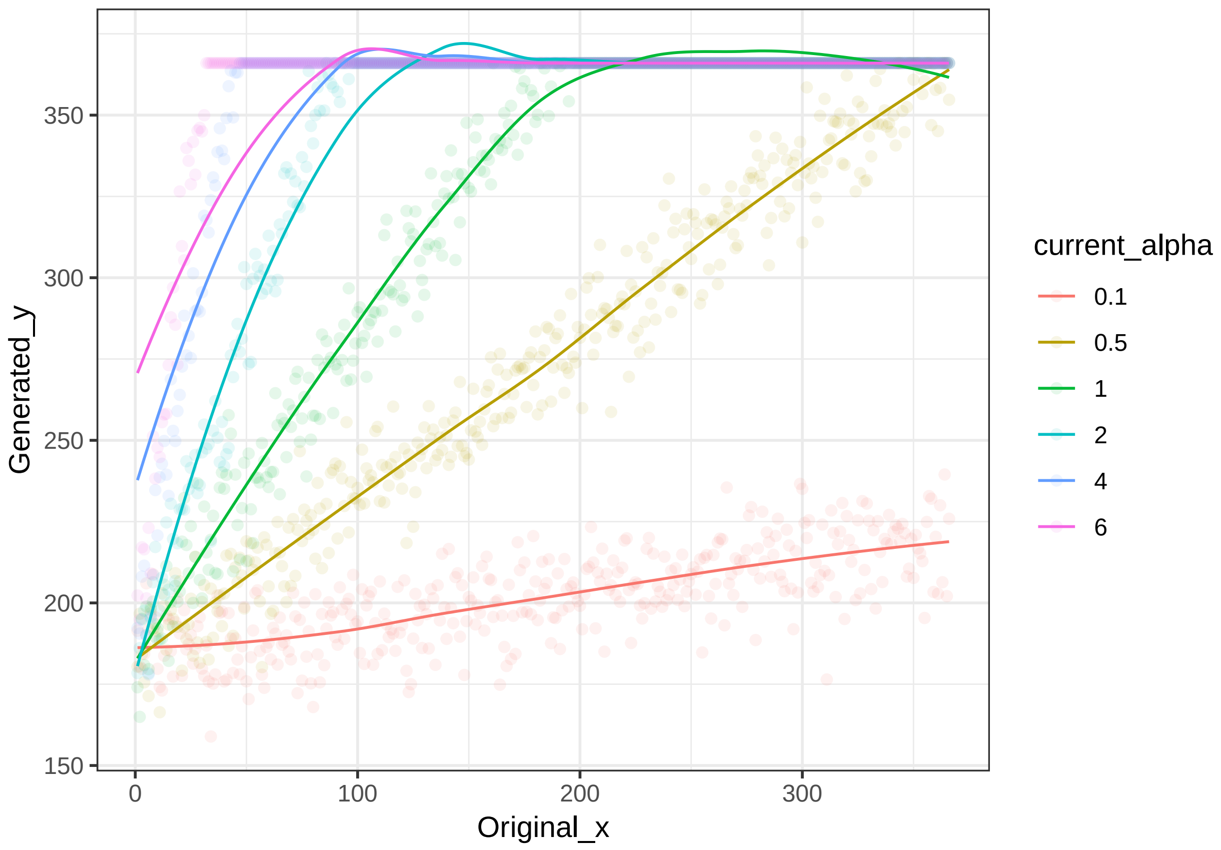
We conclude that the power is almost 1 around $\alpha \approx 1$.

The absolute values of these t-statistics for different alphas are compared to $|T_{data}| \approx 0.267$ to compute the p-values. The above plot shows that:

- With alpha-values in the interval [0.1, 0.5] we **can be certain** that the t-statistics will be more extreme than T(data).
- With alpha-values in the interval [.6, 1.9] we **cannot be certain** that the t-statistics will be more extreme than T(data).
- With alpha-values larger than 1.9 we **can be certain** that the t-statistics will be more extreme than T(data).

The nature of the y-transformation as a function of x is plotted below:

```r
df %>% mutate(., "current_alpha" = factor(current_alpha)) %>%
  ggplot(., aes(Original_x, color = current_alpha)) +
  geom_point(aes(y = Generated_y), alpha = .1) +
  geom_line(aes(y = Fitted_y))
```

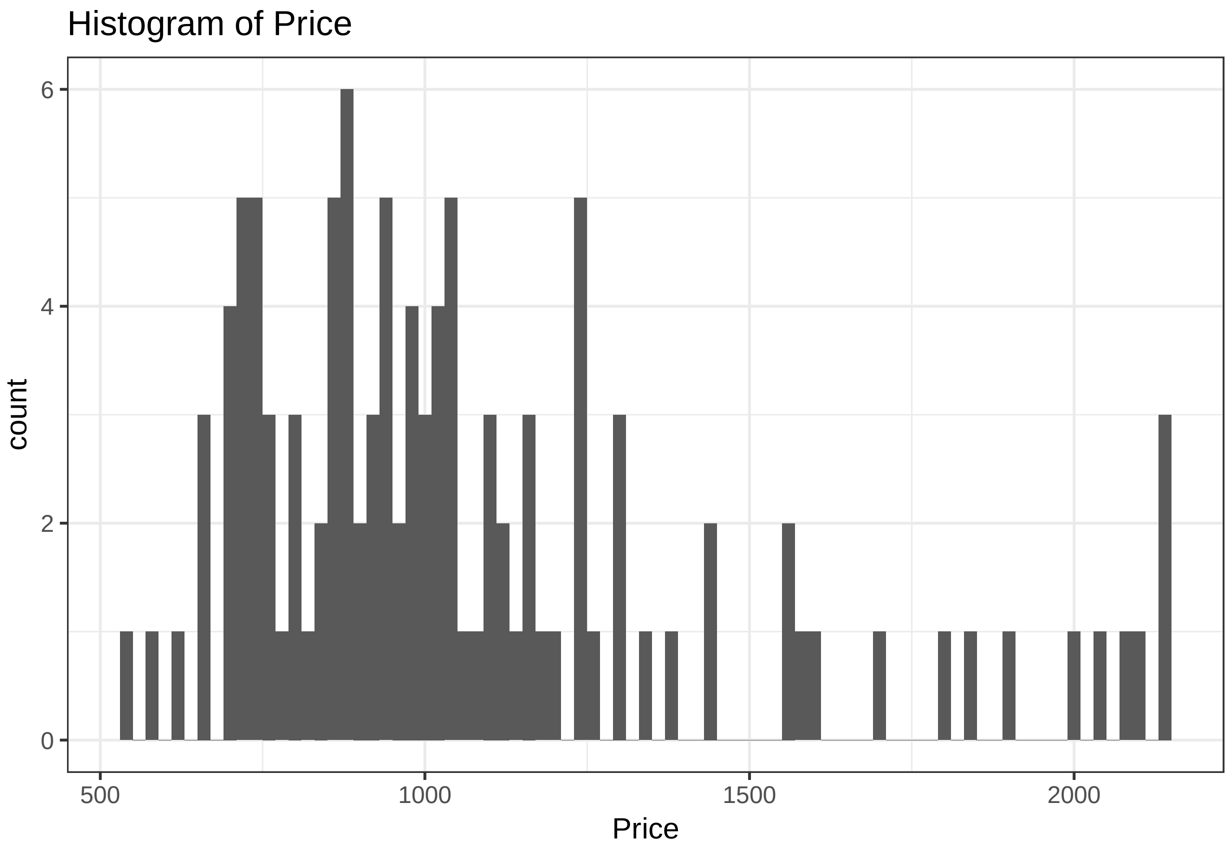


From this plot, we see the non-randomness in this transformation.

Question 2: Bootstrap, jackknife and confidence intervals

2.1: Plotting histogram of Price

```r
# Reading data.
data = read.csv("prices1.csv", sep = ";")
# Plotting histogram of Price.
ggplot() +
  geom_histogram(data = data,
                 aes(x = Price),
                 binwidth = 20) +
  theme_bw() +
  ggtitle("Histogram of Price")
```



The resulting right-skewed histogram reminds for example the gamma distribution. The mean price is calculated:

```r
t_obs = mean(data$Price)
t_obs
```

```
## [1] 1080.473
```

2.2: Estimating mean price distribution using bootstrap

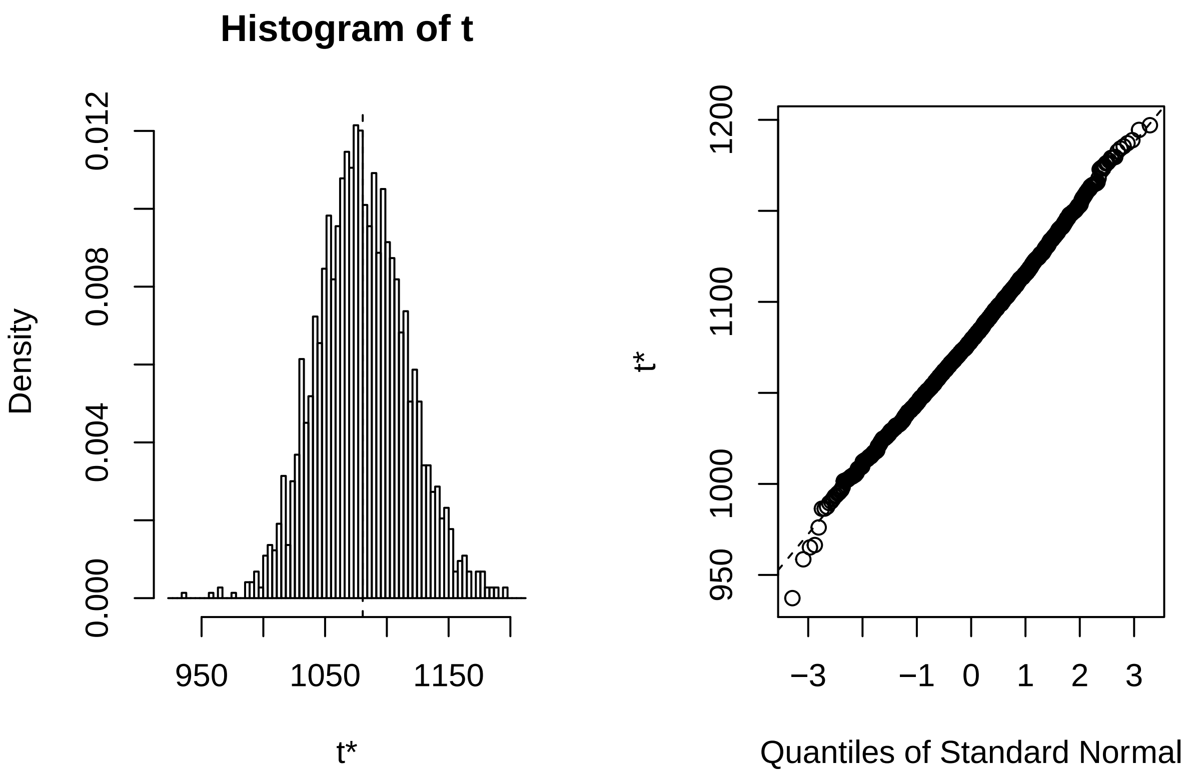
Since the calculated mean price only refers to one sample, the distribution of the mean prices can be generated using bootstrap by sampling from the data with replacement and calculating the mean for each iteration.

The number of bootstrap iterations B is set to 2000.

```
# Defining function for bootstrap to calculate mean in each step for subsetted data.
boot_function = function(data, index) {
  data = as.data.frame(data[index, ])
  t = mean(data$Price)
  return(t)
}
# Performing bootstrap. Storing test statistics in boot_result.
set.seed(12345)
boot_result = boot(data = data,
                   statistic = boot_function,
                   R = 2000)
```

The estimated distribution is plotted.

```
plot(boot_result)
```



As a result, the distribution of the mean price of the house generated using bootstrap is approximately the normal distribution which makes sense based on the *central limit theorem.*

To determine the bootstrap bias-correction, the following formula will be used:

$$T_{biasCorrected} = 2T_{observed} - \frac{1}{B}\sum_{i=1}^{B} T_i^*$$

where

$$T_i^* = nT(D) - (n-1)T(D_i^*)$$

```
# Calculating bias-corrected mean estimator.
t_bias_corrected = 2*t_obs - mean(nrow(data)*t_obs-(nrow(data)-1)*boot_result$t)
t_bias_corrected
```

```
## [1] 1027.05
```

The variance of the mean price is calculated as follows:

$$Var = \frac{1}{B-1}\sum_{i=1}^{B}\left(T(D_i^*) - \overline{T(D^*)}\right)^2$$

```
# Calculating variance of mean price.
t_bootstrap_var = sum((boot_result$t - mean(boot_result$t))^2) / (boot_result$R-1)
t_bootstrap_var
```

```
## [1] 1284.358
```

The 95% confidence intervals for the mean price using bootstrap percentile, bootstrap BCa and first-order normal approximation are generated.

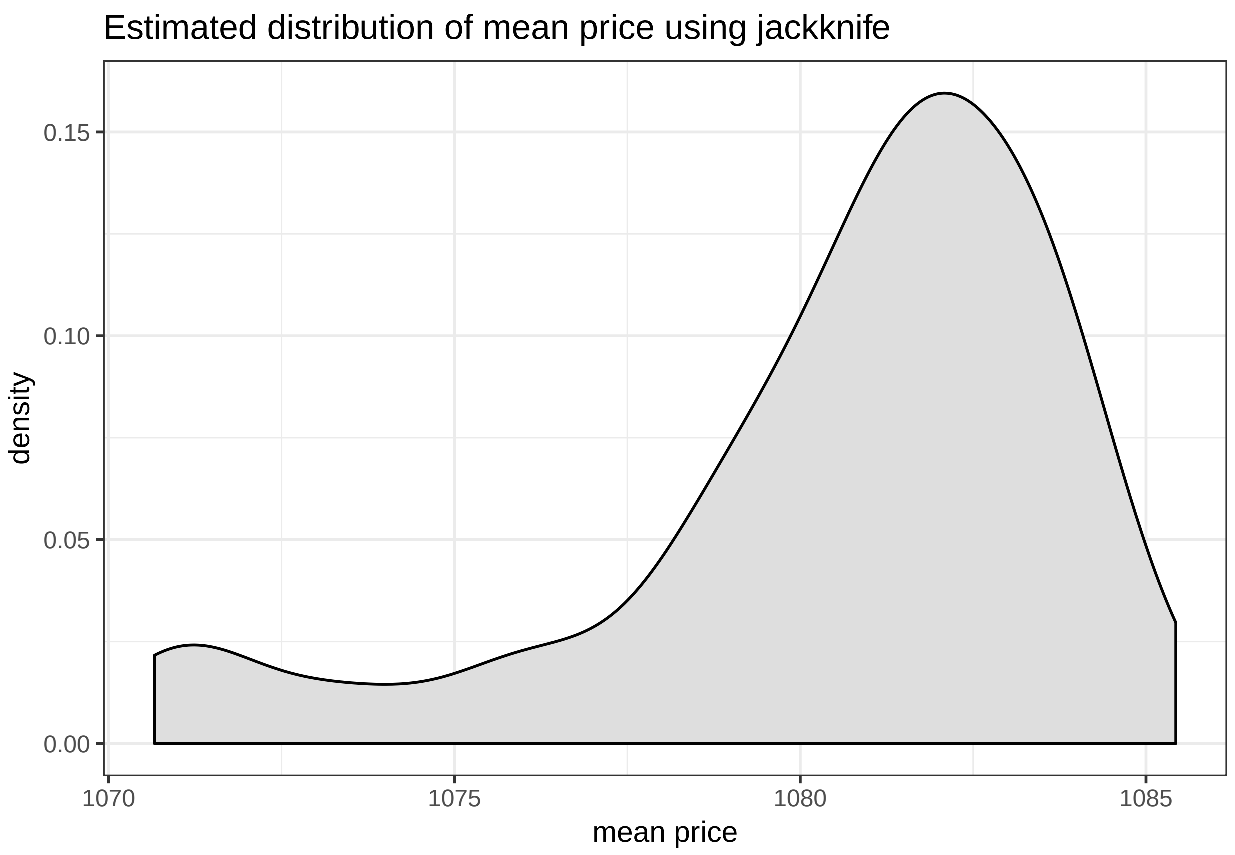
```
# Generating 95% confidence intervals.
boot_ci = boot.ci(boot.out = boot_result,
                  type = c("norm", "perc", "bca"))
boot_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_result, type = c("norm", "perc", "bca"))
##
## Intervals :
## Level      Normal             Percentile            BCa
## 95%   (1011, 1151 )   (1013, 1152 )   (1018, 1162 )
## Calculations and Intervals on Original Scale
```

2.3: Estimating jackknife variance of mean price

In this case, instead of bootrap, jackknife is performed. Within each iteration $i = 1 : B$ where $B <= nrow(data)$, observation i will be removed from the data to calculate the sample mean.

```
# Calculating sample distribution using jackknife with B = nrow(data)
t_jackknife = numeric()
for (i in 1:nrow(data)) {
  data2 = data[-i,]
  t_jackknife[i] = mean(data2$Price)
}
# Plotting estimated distribution of mean price using jackknife.
ggplot() +
  aes(t_jackknife) +
  geom_density(fill = "gray87") +
  theme_bw() +
  ggtitle("Estimated distribution of mean price using jackknife") +
  xlab("mean price")
```



The variance estimated by jackknife will be calculated as follows:

$$Var = \frac{\sum_{i=1}^{B}\left((T_i^*) - J(T)\right)^2}{B(B-1)},$$

where

$$J(T) = \frac{1}{B}\sum_{i=1}^{B} T_i^*$$

and

$$T_i^* = nT(D) - (n-1)T(D_i^*)$$

```
# Calculating jackknife estimation of variance.
t_i = length(t_jackknife)*mean(data$Price) - ((length(t_jackknife)-1)*t_jackknife)


t_jackknife_var =
  sum((t_i-mean(t_i))^2) /
  (length(t_jackknife)*length(t_jackknife)-1)
t_jackknife_var
```

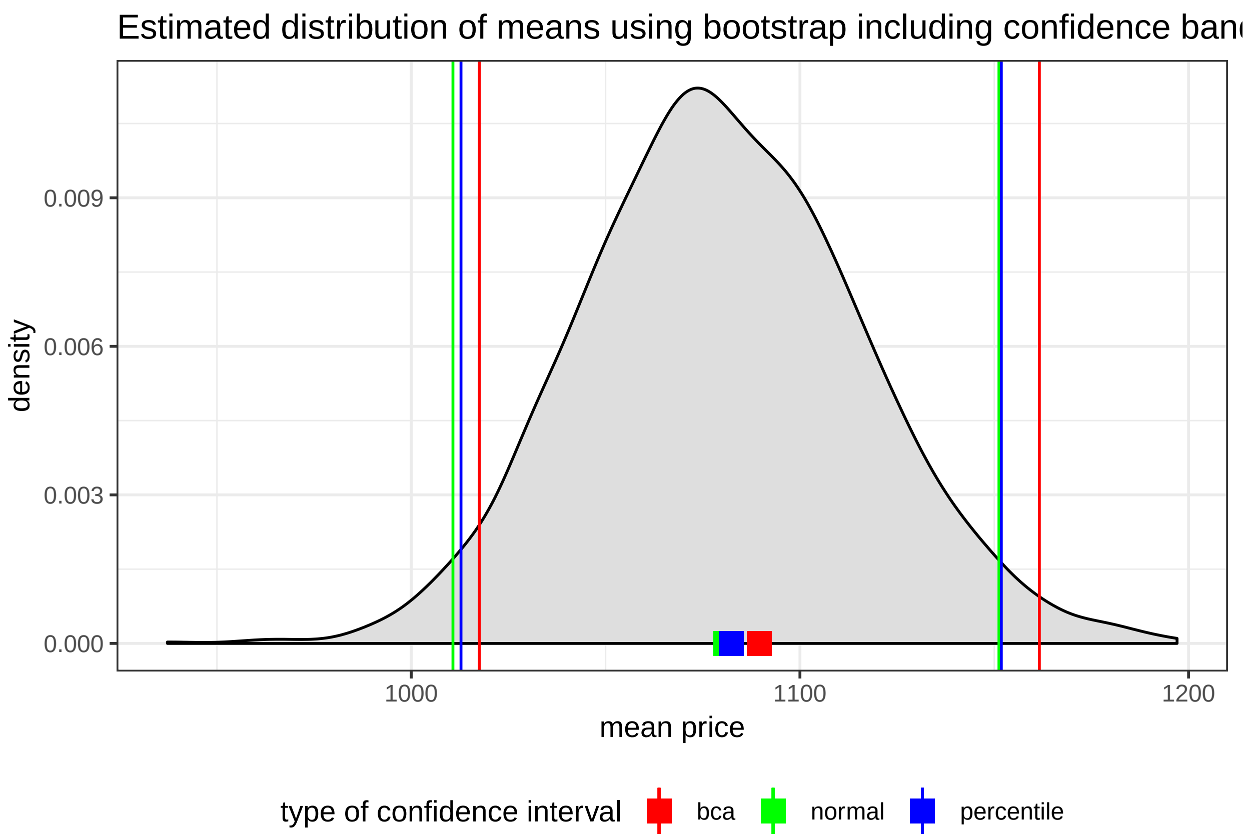
```
## [1] 1309.011
```

As expected the estimated variance obtained by jackknife is larger than the estimated variance using bootstrap.

2.4: Comparing generated bootstrap confidence intervals

The confidence intervals are obtained in 2.2. We insert all of them in the density plot to compare them visually.

```r
# Showing confidence intervals within estimated distribution of means using bootstrap
ggplot() +
  geom_density(aes(boot_result$t),
               fill = "gray87") +
  geom_vline(aes(xintercept=boot_ci$normal[2], color = "normal")) +
  geom_vline(aes(xintercept=boot_ci$normal[3], color = "normal")) +
  geom_point(aes(x = (boot_ci$normal[2]+boot_ci$normal[3])/2,
                 y=0, colour="normal"), size = 4, shape = 15) +
  geom_vline(aes(xintercept=boot_ci$percent[4], color = "percentile")) +
  geom_vline(aes(xintercept=boot_ci$percent[5], color = "percentile")) +
  geom_point(aes(x = (boot_ci$percent[4]+boot_ci$percent[5])/2,
                 y=0, colour="percentile"), size = 4, shape = 15) +
  geom_vline(aes(xintercept=boot_ci$bca[4], color = "bca")) +
  geom_vline(aes(xintercept=boot_ci$bca[5], color = "bca")) +
  geom_point(aes(x = (boot_ci$bca[4]+boot_ci$bca[5])/2,
                 y=0, colour="bca"), size = 4, shape = 15) +
  theme_bw() +
  scale_color_manual(name = "type of confidence interval",
                     values = c(percentile = "blue",
                                bca = "red",
                                normal = "green")) +
  theme(legend.position = "bottom") +
  ggtitle("Estimated distribution of means using bootstrap including confidence bands") +
  xlab("mean price")
```



```
# Calculating lengths of confidence intervals
```

As it can be seen in the plot, all three different types of confidence intervals are characterized by the approximately same length. Every confidence interval is just shifted to the right or to the left, respecively. The comparison of the means confirm that all three confidence intervals are similar.

Appendix

All code is shown in the document.