Lab 5

Stefano Toffol (steto820) // Mim Kemal Tekin (mimte666)

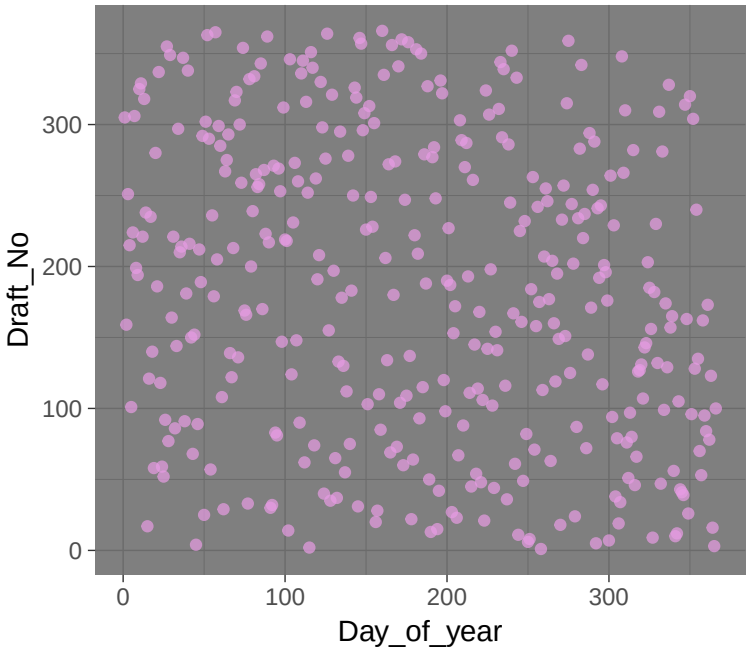
27 February, 2019

Chapter 1

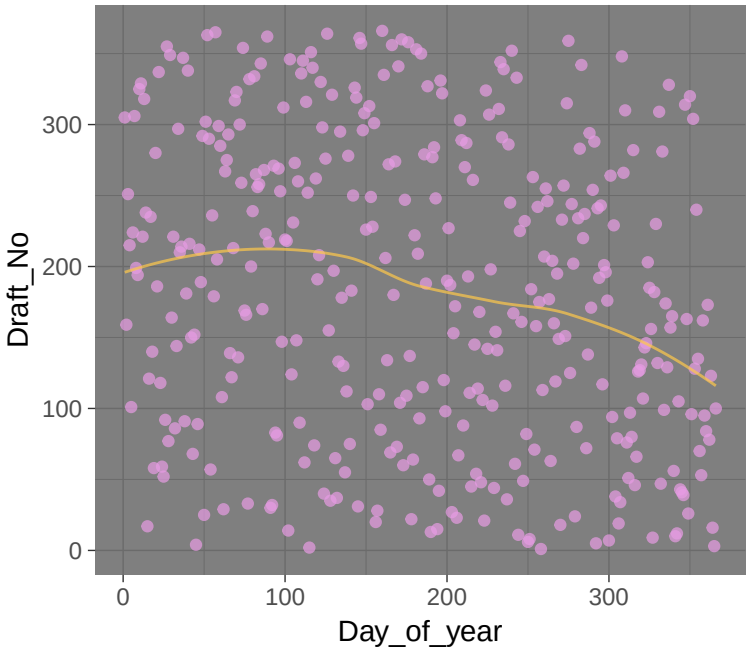
Lab 5

Question 1: Hypothesis Testing

Task 1.1



Task 1.2



Task 1.3

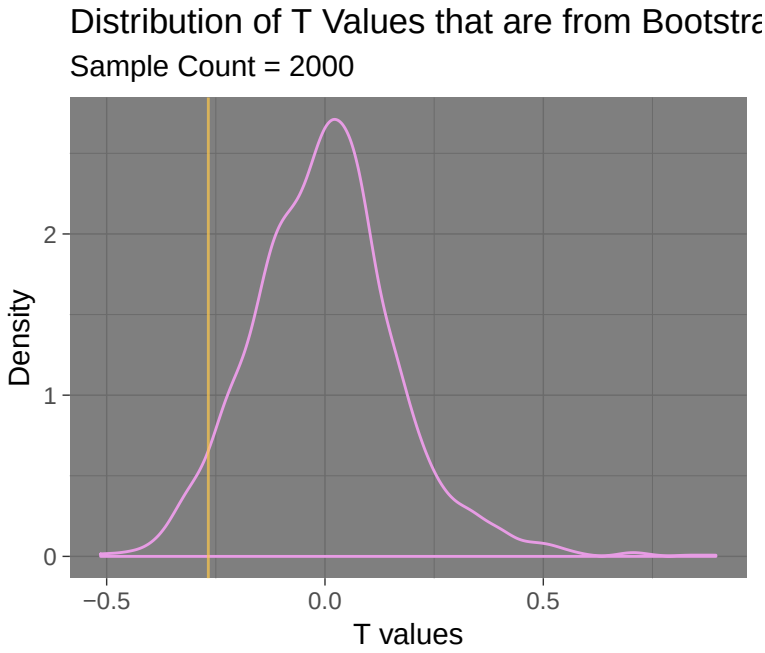
In this task we will check if lottery is random by calculating the test statistics which is defined as following:

$$T = \frac{\hat{Y}(X_b) - \hat{Y}(X_a)}{X_b - X_a}, \quad \text{where} \quad X_b = \text{argmax}_X Y(X), \; X_a = \text{argmin}_X Y(X)$$

We know that $Y(\text{argmax}_X Y(X)) = \max\, Y(X)$

We calculate test statitstics as $T = -0.2671794$. This value is not greater than zero, so we can say the lottery is random.

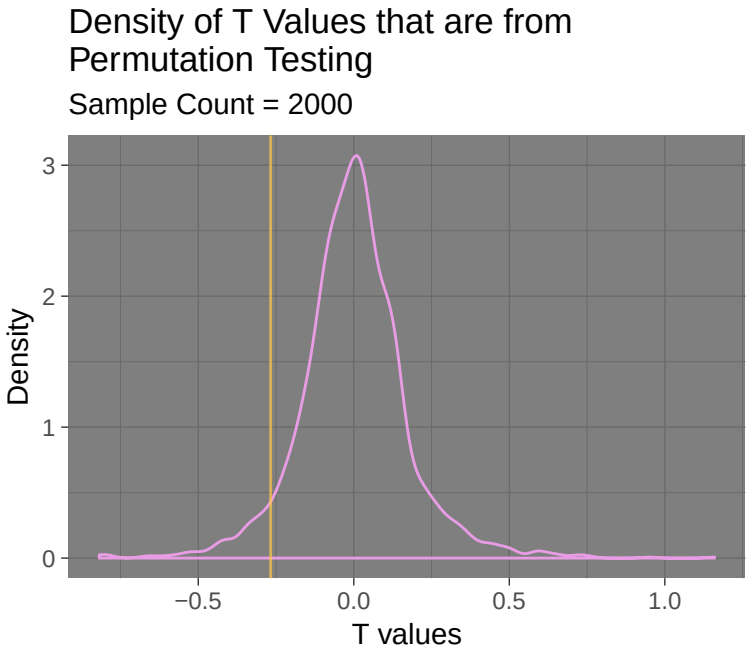
Now we will estimate the distribution of T by using non-parametric bootsrap model with 2000 samples. We will create the function which takes the data and the sample count, and returns the samples that is created with replacement from the original data.



After calculating all Test values we can see density of T values as the plot above, but it is hard to identify the distribution of this density. We are testing the hypothesis below:
H_0 : Lottery is random
H_1 : Lottery is non-random
The condition of being non-random requires that T statistics value should be significantly greater than zero. This means that we tent to reject our H_0 when the observed values of T statistics are on the right tail of the distribution, so we should calculate our p-value by calculating the right side area of T statistic of the original data. We can calculate this simply by the count of T values which are greater that the T value of original data and dividing this value by the sample count or the count of T values. With this method we can find p-value of the test as $p = 0.9545$. P-value that we found should be less then 0.05 (5%) in order to be inside the rejection area of H_0. Obviously, it is not inside this area and we can say that we do not reject H_0 hypothesis. This means T values greater than T values of the original data occur more than 5% of the test samples. In the light of this findings, we can say it is more likely that this data is random.

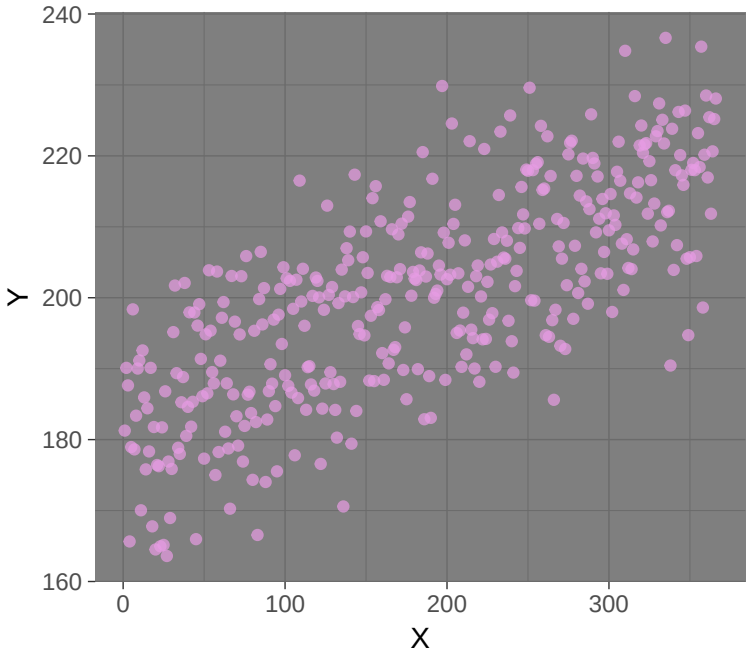
Task 1.4

In this task we will apply the permutation test in order to test the hypothesis below:
H_0 : Lottery is random
H_1 : Lottery is non-random



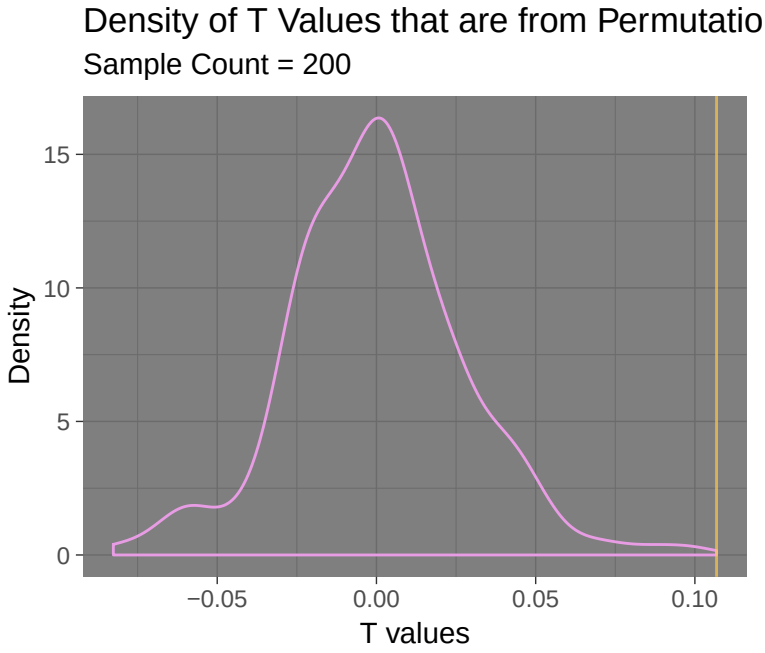
In this task we create permutation test for our data. This method creates different permutations by picking a sample from Y values and creates another data frame by using the same X values as the original data. This process is repated for the sample count times and for each permutation sample we calculate T statistics. After obtaining this all T statistics we calculate and return the p-value of the sample that created. The function that we created finds the p-value of only one permutation test as $p = 0.9475$. This p-value is greater than 0.05 as like before, and we can say that we do not reject H_0 because of the same reasons in task 1.3.

Task 1.5

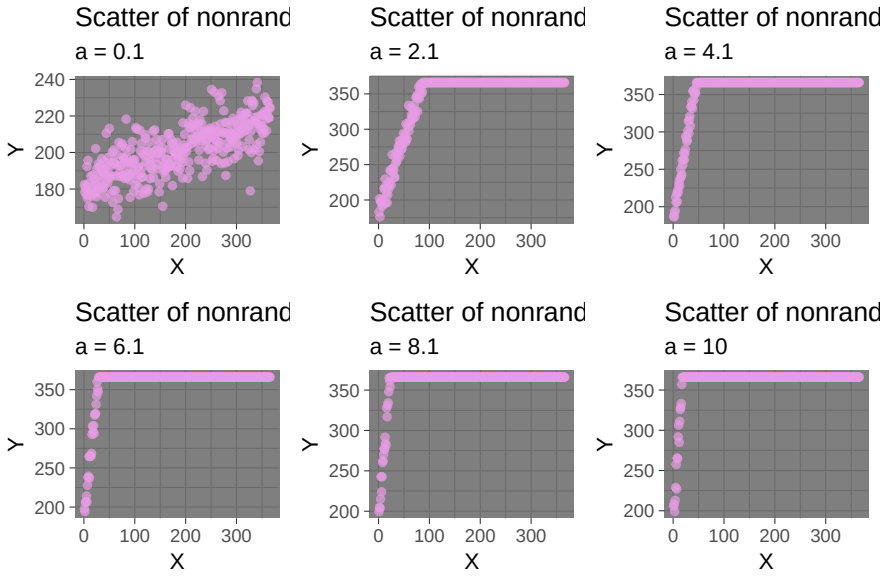


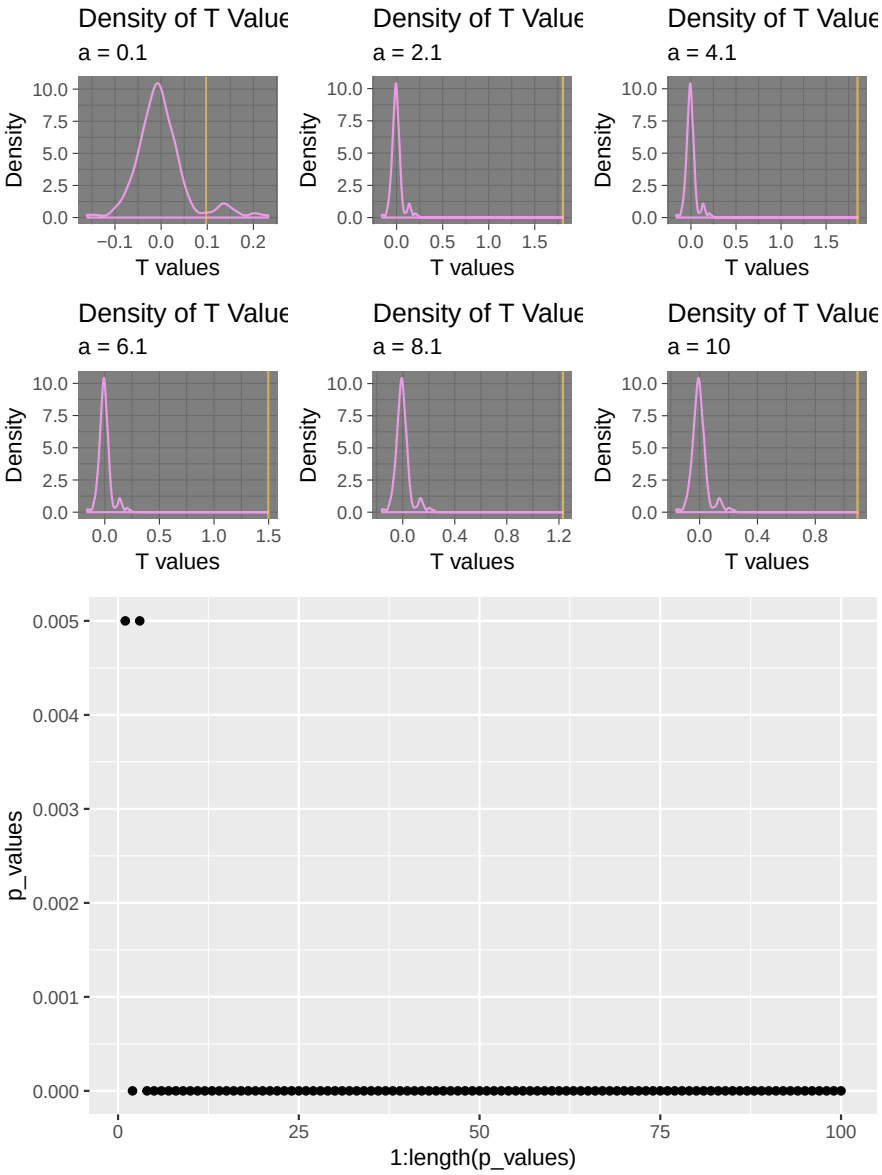
In this task we create a function that generates non-random sample for the Y values. We can see the plot of a sample that was created with this function and we can see that this sample is not random obviously. It has

a increasing trend except some outliers but most of the points have really high density along diagonal of x and y axises. After getting this sample we calculate a p-value for this sample by using permutation testing function that we created one task before with 200 sample count. Result of test we get the p-value as $p = 0$. This result confirms us we have totally non-random sample. The density plot of statistics of this sample is below and the t statistics of the original sample is also showed with a vertical intercept on X axis. As we can see this value has all values in the distribution.



In the scatterplots below show how, with the increase of α the generated Y variable gets closer and closer to being a constant (366). The density plots are instead referring to the T statistics distribution and show how the estimated value of the statistics falls more and more to the right tail of the distribution, increasing every time the significance of the test. In other words, the more the data gets less random, the more strongly the test tends to reject the null hypothesis of randomic data. In the last plot is shown how all the p-values of the statistic are below 0.05, meaning that the test will result into a rejection of the null hypothesis every single time. This is exactly the bahaviour we would expect, since the data was actually not random. This crude estimate of the power of the test shows us the actual reliability of the test itself.




Density of T Value
a = 0.1
Density of T Value
a = 2.1
Density of T Value
a = 4.1
Density of T Value
a = 6.1
Density of T Value
a = 8.1
Density of T Value
a = 10
Density
T values
p_values
1:length(p_values)

Question 2

The data we are about to analyse contains the prices of houses in Albuquerque, New Mexico, 1993. The variables included are: `Price`, our response variable; `SqFt`, the surface of the house in square feet; `FEATS`, the number of different features the houses have (such as dishwasher, refrigerator, ...); `Taxes`, the amount (in dollars) of taxes the owner has to pay for the building.

First of all we study the distribution of the response variable. In Figure 1.1 is shown the density histogram of `Price`. The solid red line is the estimated density distribution of the data. As we can observe, the curve generally catches the trend of the variable, a part from some local minima/maxima and with the exception of the heavy right tail.
The asymmetric shape suggests that the well-known *gamma distribution* may consist of a valid parametric solution to fit the density of our data. The random variable can be parametrised in terms of a shape parameter α and an inverse scale parameter β, resulting in $Gamma(\alpha, \beta)$. The mean and the variance of the distribution are respectively equal to α/β and α/β^2.
Moreover also the log-normal distribution can approximate really well the trend of the data and should therefore be considered. The random variable depends on the parameters μ and σ, respectively the mean and the standard deviation of the normal variable that is implicit in a log-normal distribution.

```
# Estimating the parameters of the two distributions distribution

# Gamma distribution
# 1) ML estimates (method "mle") for the gamma distribution
gamma_ML <- fitdistrplus::fitdist(data$Price, distr = "gamma", "mle")
alpha_ML <- gamma_ML$estimate[1]
beta_ML <- gamma_ML$estimate[2]

# 2) Maximum goodness-of-fit estimates (method "mge") for the gamma distribution
gamma_goodness <- fitdistrplus::fitdist(data$Price, distr = "gamma", "mge")
alpha_goodness <- gamma_goodness$estimate[1]
beta_goodness <- gamma_goodness$estimate[2]

# Log-normal distribution
# 3) ML estimates (method "mle") for the log-normal distribution
lognorm_ML <- fitdistrplus::fitdist(data$Price, distr = "lnorm", "mle")
mu_ML <- lognorm_ML$estimate[1]
sigma_ML <- lognorm_ML$estimate[2]

# 4) Maximum goodness-of-fit estimates (method "mge") for the log-normal distribution
lognorm_goodness <- fitdistrplus::fitdist(data$Price, distr = "lnorm", "mge")
mu_goodness <- lognorm_goodness$estimate[1]
sigma_goodness <- lognorm_goodness$estimate[2]
```

In the chunk above we used the package of R `fitdistrplus` and its function `fitdist(·)`, which allows us to estimates the parameters of a certain random variable using the data available according to a certain algorithm.
For the gamma distribution (panel A, on the right) the ML estimations require an iterative optimization algorithm to be found: despite the fact that the inverse scale parameter has the close-form solution $\hat{\beta}_{ML} = \hat{\mu}/\alpha$, the shape parameter α will require a numerical solution. Using `fitdist(·)`, as seen above, we are able to get the results: $\hat{\alpha}_{ML} = 9.66828$, $\hat{\beta}_{ML} = 0.00895$. The value of the estimated mean of the variable is practically equal to the sample mean of the data (equal to 1080.4727). The resulting line in Figure 1.1 A is the dashed black one.

Since the shape of the estimated density according to the ML estimators does not resemble that closely the empirical one, we also used the method `mge` from the function `fitdist(·)` to get the maximum goodness-of-fit estimates. The parameters estimates are then equal to: $\hat{\alpha}_{mge} = 12.63398$, $\hat{\beta}_{mge} = 0.01237$. In this case the shape of the resulting curve (panel A, solid black line) is much more similar to the estimated density. However the mean of the estimated variable is equal to 1021.05165, which differs from the sample mean of 59.42108 thousands of dollars.
On the other hand the log-normal distribution follows even closer the trend in the data, for both the ML and MGE estimates. Their behaviour is anyway similar to the one of the gamma distribution: the ML estimates underestimates the density mass below the average, even though it's estimate is equal to the sample one ($\mu_{ML} = 6.93258$, $\sigma_{ML} = 0.31394$); the MGE result in a shape closer to the estimated density, however they underestimate the actual mean of the data. In fact, according to the last method, $\hat{\mu}_{mge} = 6.89837$, which means a difference of 47.84248 from the sample average.

It seems impossible to decide which random variable to choose between the two tested. Further tests and analysis should be run to take a decision.

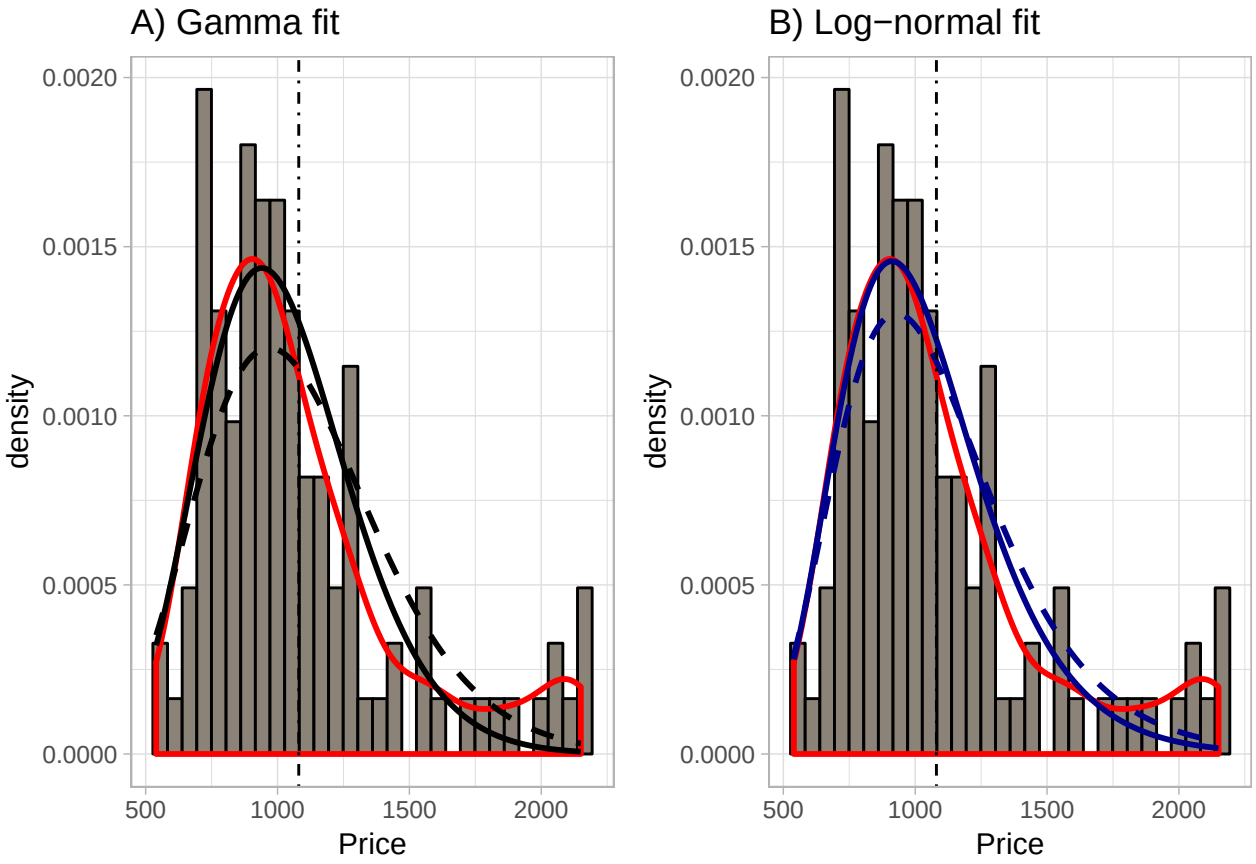


Figure 1.1: Density histogram of the response variable. The line in red is the estimated kernel density of the data.
A) The black lines rapresent the density of a gamma variable; the dashed one is drawn from the ML estimates of the parameters, the solid one from the MGE estimates of the parameters.
B) The blue lines rapresent the density of a lognormal variable; the dashed one is drawn from the ML estimates of the parameters, the solid one from the MGE estimates of the parameters.

We are now going to estimate the distribution of the mean of the variable by generating multiple samples of our data using bootstrap. We chosen to perform a non-parametric one since, even though we previously found comparable distributions to the observed one, we had no means to choose between the two tested distributions and the various possible estimations of their parameters. In the process we took into consideration the bias-correction and computed the *confidence interval* (CI) using three different techniques: bootstrap percentile, bootstrap BCa, and first-order normal approximation. The code to achieve it is the following:

```r
library(boot)

# Function to compute the bootstrap mean
boot_mean <- function(data, ind) {

  return( mean(data[ind]) )

}

# Set seed for reproducibility purposes
set.seed(12345)
# Bootstrap size
B <- 10000
# Bootstrap object
boot_obj <- boot(data$Price, boot_mean, R = B)

# Bootstrap mean
boot_mean <- mean(boot_obj$t)
# Take the mean of the response
average_price <- mean(data$Price)
# Bias-corrected estimator
bias_corrected_estimator <- 2*average_price - boot_mean
# Bootstrap variance
boot_variance <- var(boot_obj$t[,1])

# Confidence interval percentile
ci_percentile <- boot.ci(boot_obj, conf = 0.95, type = "perc")
# Confidence interval BCa
ci_BCa <- boot.ci(boot_obj, conf = 0.95, type = "bca")
# Confidence interval first-order normal approximation
ci_firstNorm <- boot.ci(boot_obj, conf = 0.95, type = "norm")
```

The output of our bootstrap estimates are plotted in Figure 1.2. As we can observe the distribution of the mean closely resemble the normal one. The result is what we were expecting, since it is analytically proven that the sample mean of any distribution with finite variance follows asymptotically the Gaussian density (delta method theorem). The observations on the tails of the distribution in fact do not show any systematic shifting from the normality. The distribution appears symmetric and with a clear bell shape.

The mean of all bootstrap iterations is equal to 1080.6951. Considering the sample mean is 1080.4727 the two values are extremely close, differing of only 0.2224. We can infer therefore infer that the original sample mean we got was practically unbiased due to the very little difference found: in fact the bootstrap bias-correction is equal to 1080.2504. Finally the variance of the bootstrap means is equal to 1289.9399, which may seem a consistently high value. However if we related it to the scale of the original variable computing the *coefficient of variation* ($CV_X = \sigma_X/\mu_X$), we get 0.0332, which means that the resulting bootstrap distribution shows a very low variability.

The results of the CI computed above are instead summarized in Table 1.1. As a consequence of the symmetry of the density the three different interval appear to be very similar: the range and the position of their centre is comparable. The difference between the actual mean of the sample and the centre of the interval is at maximum -7.8, a tiny number if related to the scale of the response. Measuring their asymmetry results in all coefficients close to 1 ("*Shape*" column), confirming our observations. In other words intervals of first-order accuracy (*percentile* and *normal* method, which do not adjust for skewness in the bootstrap distribution) and second-order accuracy (*BCa*, which takes into account asymmetry of the distribution) are comparable in both range and positions of their extremes, as we would expect from an actually normal bootstrap distribution. The intervals are relatively narrow if we consider the nature of our response variable.

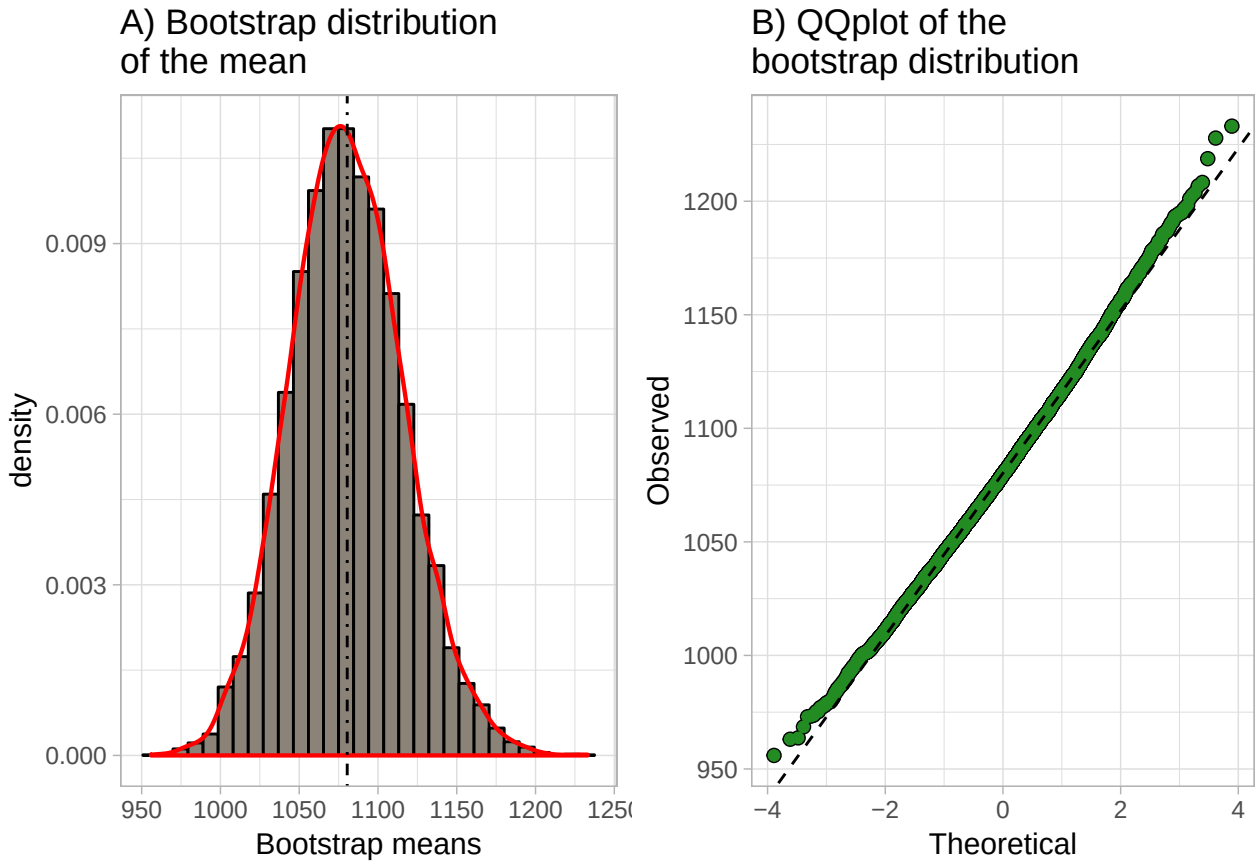


Figure 1.2: Histrogram and QQplot of the bootstrap distribution of the mean.

Table 1.1: Comparative table of the various type of bootstrap CI.

CI type	Lower extreme	Upper extreme	Range	Center	Difference	Shape
Percentile	1012.328	1154.309	141.9807	1083.319	-2.8457960	1.0767456
BCa	1016.248	1160.228	143.9796	1088.238	-7.7655352	1.2340905
First-order normal	1009.857	1150.644	140.7871	1080.250	0.2223655	0.9874438

We finally estimate the variance of the mean using the *Jackknife method* to check if our conclusions are correct. The code used to implement the technique is the following:

```
jackknife <- function(data, statistic) {

  n <- length(data)
  # Statistic applied on the starting dataset
  original_statistic <- statistic(data)
  jackknife_replicates <- rep(NA, n)

  for(i in 1:n) {
    # Value of the statistic for the sample without the i-th observation
    jackknife_replicates[i] <- statistic(data[-i])
  }

  # Formulas for the jackknife from the book (same result as Krzysztof's slides)
  jackknife_mean <- mean(jackknife_replicates)
  jackknife_variance <- (n-1)/n * sum((jackknife_replicates-jackknife_mean)^2)

  return( jackknife_variance )

}

jackknife_var_mean <- jackknife(data$Price, mean)
```

The jackknife estimate of the variance of the sample mean is equal to 1320.911, while the one got from the standard bootstrap was instead 1289.9399. The difference between the two estimates is pretty small, only equal to 30.9711. Considering the tendency to overestimate of the jackknife method and once more the scale of the variable, we consider the difference reasonable and therefore attribute this difference to just random noise rather then a systematic bias of our sample.

Appendix

```
knitr::opts_chunk$set(echo = F, message = F, error = F, warning = F,
                      fig.align='center', out.width="70%")

library(ggplot2)
data_military = read.csv2("../datasets/lottery.csv")

ggplot(data_military, aes(x=Day_of_year, y = Draft_No)) +
  geom_point(color = "#E79CE4", alpha = 0.7) +
  theme_dark()

draft_est_loess = loess(formula = Draft_No~Day_of_year, data = data_military)

y_hat_loess = predict(draft_est_loess, data_military$Day_of_year)

ggplot(data_military, aes(x=Day_of_year, y = Draft_No)) +
  geom_point(color = "#E79CE4", alpha = 0.7) +
  geom_line(aes(y = y_hat_loess), color = "#FFC84B", alpha = 0.7) +
  theme_dark()

#### the function that calculates test statistics
## inputs:
##  data <data.frame> :
##    columns:
##      X:argument,
##      Y: real values
test_statistics = function(data){
  loess_predictor = loess(formula = Y~X, data = data)
  Y_hat = loess_predictor$fitted
  x_b = which.max(data$Y)
  x_a = which.min(data$Y)
  y_hat_b = Y_hat[x_b]
  y_hat_a = Y_hat[x_a]
  return((y_hat_b - y_hat_a) / (data$X[x_b] - data$X[x_a]))
}

df = data.frame(X = data_military$Day_of_year,
                Y = data_military$Draft_No)


T_org_data = test_statistics(df)

#### the function that calculates test statitstics by using non-parametric
#### bootstrap
## inputs:
##  data <data.frame> :
##    columns:
##      X:argument,
##      Y: real values,
##  sample_count <numeric> :  count of the samples that will be created.
bootstrap_np_test_stats = function(data, sample_count){
  n = dim(data)[1]
```

```
  T_values = sapply(1:sample_count, function(x) {
    indexes = sample(1:n, n, replace = TRUE)
    return(test_statistics(data[indexes,]))
  })
  return(T_values)
}
# set seed for report
set.seed(12345)
# get the samples of T_balues
bootstrap_step = 2000
T_values = bootstrap_np_test_stats(df, bootstrap_step)
centered_T_values = scale(T_values, scale=F)

# plot the T_values
ggplot() +
  geom_density(mapping = aes(x = centered_T_values),
               color = "#E79CE4", alpha=0.7) +
  geom_vline(xintercept = T_org_data, color= "#FFC84B", alpha = 0.7) +
  labs(title = "Distribution of T Values that are from Bootstrap",
       subtitle = "Sample Count = 2000",
       x = "T values", y = "Density") +
  theme_dark()

#### the function that calculates p value empirically
## inputs:
##  t_values <list:numeric>: all t values
##  t0: t value of the original data
calculate_p_value = function(t_values, t0){
  return(sum(t_values > t0) / length(t_values))
}

p_value = calculate_p_value(centered_T_values, T_org_data)
#### the function that calculates test statistics by using permutation test
#### and returns the T-values of this
## inputs:
##  data <data.frame> :
##    columns:
##      X:argument,
##      Y: real values,
##  sample_count <numeric> :  count of the samples that will be created.
permutation_test = function(data, sample_count=2000){
  n = dim(data)[1]
  T_values = sapply(1:sample_count, function(x){
    Y_sample = sample(data$Y, n, replace = T)
    n_df = data.frame(X = data$X, Y = Y_sample)
    return(test_statistics(n_df))
  })
  return(T_values)
}

#### the function that calculates test statistics by using permutation test
#### and returns the p-values of this tests
## inputs:
```

```
##  data <data.frame> :
##    columns:
##      X:argument,
##      Y: real values,
##  sample_count <numeric> :  count of the samples that will be created.
p_value_w_permutation = function(data, sample_count = 2000){
  T_values = permutation_test(data, sample_count)
  centered_T_values = scale(T_values, scale = F)
  T_0 = test_statistics(data)
  result = list(t0 = T_0,
                centered_t_values = centered_T_values,
                p_value = calculate_p_value(centered_T_values, T_0))
  return(result)
}

result = p_value_w_permutation(df, 2000)
centered_t_values = result$centered_t_values
t_0 = result$t0
p_value = result$p_value


ggplot() +
  geom_density(mapping = aes(x = centered_t_values),
               color = "#E79CE4", alpha=0.7) +
  geom_vline(xintercept = t_0, color= "#FFC84B", alpha = 0.7) +
  labs(title = "Density of T Values that are from \nPermutation Testing",
       subtitle = "Sample Count = 2000",
       x = "T values", y = "Density") +
  theme_dark()
#### the function that generates non-random Y values with same X values of
#### dataframe, and returns a new non-random dataframe
#### this non-random generator uses normal(mean=188, sd=10)
## inputs:
##  data <data.frame> :
##    columns:
##      X:argument,
##      Y: real values,
##  alpha <float> :  coefficient of x value in the original data
generate_nonran_df = function(data, alpha){
  nonrandom_Y = sapply(data$X, function(x, alpha){
    a = alpha*x + rnorm(1,183,10)
    return(max(0, min(a, 366)))
  }, alpha)
  return(data.frame(X = df$X, Y = nonrandom_Y))
}


# generate the new dataframe
nonran_df = generate_nonran_df(df, 0.1)

result = p_value_w_permutation(nonran_df, 200)
centered_t_values = result$centered_t_values
t_0 = result$t0
```

```
p_value = result$p_value

# p_value = p_value_w_permutation(nonran_df, 200)



ggplot(nonran_df, aes(x = X, y = Y)) +
  geom_point(color = "#E79CE4", alpha = 0.7) +
  theme_dark()



ggplot() +
  geom_density(mapping = aes(x = centered_t_values),
               color = "#E79CE4", alpha=0.7) +
  geom_vline(xintercept = t_0, color= "#FFC84B", alpha = 0.7) +
  labs(title = "Density of T Values that are from Permutation Testing",
       subtitle = "Sample Count = 200",
       x = "T values", y = "Density") +
  theme_dark()


alpha = 1:100/10
scatter_plots = list()
density_plots = list()
p_values = c()
for(i in 1:length(alpha)){
  a = alpha[i]
  nonran_df = generate_nonran_df(df, a)
  result = p_value_w_permutation(nonran_df, 200)
  centered_t_values = result$centered_t_values
  t_0 = result$t0
  p_value = result$p_value
  p_values[i] = p_value
  if(i %% 20 == 1 || i == 100){
    p_scatter = ggplot(nonran_df, aes(x = X, y = Y)) +
                  geom_point(color = "#E79CE4", alpha = 0.7) +
                  labs(title = "Scatter of nonrandom dataset a=",
                       subtitle = paste("a =",a)) +
                  theme_dark()
    p_density = ggplot() +
                  geom_density(mapping = aes(x = centered_t_values),
                               color = "#E79CE4", alpha=0.7) +
                  geom_vline(xintercept = t_0, color= "#FFC84B", alpha = 0.7) +
                  labs(title = "Density of T Values, Permutation Testing",
                       subtitle = paste("a =",a),
                       x = "T values", y = "Density") +
                  theme_dark()
    scatter_plots[[paste("a_", a, sep="")]] = p_scatter
    density_plots[[paste("a_", a, sep="")]] = p_density
  }
}
```

```r
library(gridExtra)

grid.arrange(grobs = scatter_plots, ncol=3)

grid.arrange(grobs = density_plots, ncol=3)

ggplot() +
  geom_point(aes(x=1:length(p_values), y = p_values))



# ------------------------------------------------------------------------------
# A2
# ------------------------------------------------------------------------------


# Read the data
data <- readxl::read_xls("../datasets/prices1.xls")

# Take the mean of the response
average_price <- mean(data$Price)


# Estimating the parameters of the two distributions distribution

# Gamma distribution
# 1) ML estimates (method "mle") for the gamma distribution
gamma_ML <- fitdistrplus::fitdist(data$Price, distr = "gamma", "mle")
alpha_ML <- gamma_ML$estimate[1]
beta_ML <- gamma_ML$estimate[2]

# 2) Maximum goodness-of-fit estimates (method "mge") for the gamma distribution
gamma_goodness <- fitdistrplus::fitdist(data$Price, distr = "gamma", "mge")
alpha_goodness <- gamma_goodness$estimate[1]
beta_goodness <- gamma_goodness$estimate[2]

# Log-normal distribution
# 3) ML estimates (method "mle") for the log-normal distribution
lognorm_ML <- fitdistrplus::fitdist(data$Price, distr = "lnorm", "mle")
mu_ML <- lognorm_ML$estimate[1]
sigma_ML <- lognorm_ML$estimate[2]

# 4) Maximum goodness-of-fit estimates (method "mge") for the log-normal distribution
lognorm_goodness <- fitdistrplus::fitdist(data$Price, distr = "lnorm", "mge")
mu_goodness <- lognorm_goodness$estimate[1]
sigma_goodness <- lognorm_goodness$estimate[2]


library(ggplot2)

p_gamma <- ggplot(data, aes(x = Price)) +
  geom_histogram(aes(y = ..density..), fill = "antiquewhite4", col = "black") +
  geom_density(col = "red", size = 1) +
```

```
  stat_function(size = 1, lty = 2, col = "black", fun = dgamma, n = 500,
                args = list(shape = alpha_ML, rate = beta_ML)) +
  stat_function(size = 1, lty = 1, col = "black", fun = dgamma, n = 500,
                args = list(shape = alpha_goodness, rate = beta_goodness)) +
  geom_vline(aes(xintercept = average_price), lty = 4) +
  ggtitle("A) Gamma fit") +
  theme_light()

p_lognorm <- ggplot(data, aes(x = Price)) +
  geom_histogram(aes(y = ..density..), fill = "antiquewhite4", col = "black") +
  geom_density(col = "red", size = 1) +
  stat_function(size = 1, lty = 2, col = "darkblue", fun = dlnorm, n = 500,
                args = list(meanlog = mu_ML, sdlog = sigma_ML)) +
  stat_function(size = 1, lty = 1, col = "darkblue", fun = dlnorm, n = 500,
                args = list(meanlog = mu_goodness, sdlog = sigma_goodness)) +
  geom_vline(aes(xintercept = average_price), lty = 4) +
  ggtitle("B) Log-normal fit") +
  theme_light()

gridExtra::grid.arrange(p_gamma, p_lognorm, ncol = 2)


library(boot)

# Function to compute the bootstrap mean
boot_mean <- function(data, ind) {

  return( mean(data[ind]) )

}

# Set seed for reproducibility purposes
set.seed(12345)
# Bootstrap size
B <- 10000
# Bootstrap object
boot_obj <- boot(data$Price, boot_mean, R = B)

# Bootstrap mean
boot_mean <- mean(boot_obj$t)
# Take the mean of the response
average_price <- mean(data$Price)
# Bias-corrected estimator
bias_corrected_estimator <- 2*average_price - boot_mean
# Bootstrap variance
boot_variance <- var(boot_obj$t[,1])

# Confidence interval percentile
ci_percentile <- boot.ci(boot_obj, conf = 0.95, type = "perc")
# Confidence interval BCa
ci_BCa <- boot.ci(boot_obj, conf = 0.95, type = "bca")
# Confidence interval first-order normal approximation
ci_firstNorm <- boot.ci(boot_obj, conf = 0.95, type = "norm")
```

```r
# Custom ggplot function for the QQplot

ggQQ <- function(vec, col = "forestgreen") {

  require(ggplot2)

  y <- quantile(vec[!is.na(vec)], c(0.25, 0.75))
  x <- qnorm(c(0.25, 0.75))
  slope <- diff(y)/diff(x)
  int <- y[1L] - slope * x[1L]

  d <- data.frame(resids = vec)

  p <- ggplot(d, aes(sample = resids)) +
    stat_qq(color="black", shape=1, size=I(2)) +
    stat_qq(color = col) +
    geom_abline(slope = slope, intercept = int, lty = 2) +
    labs(x = "Theoretical", y = "Observed") +
    theme_light()

  return(p)

}

boot_hist <- ggplot(as.data.frame(boot_obj$t), aes(x = V1)) +
  geom_histogram(aes(y = ..density..), fill = "antiquewhite4", col = "black") +
  geom_density(col = "red", size = 0.75) +
  geom_vline(aes(xintercept = boot_obj$t0), lty = 4) +
  labs(x = "Bootstrap means") +
  ggtitle("A) Bootstrap distribution\nof the mean") +
  theme_light()

boot_qqplot <- ggQQ(boot_obj$t)
boot_qqplot <- boot_qqplot +
  ggtitle("B) QQplot of the\nbootstrap distribution")

gridExtra::grid.arrange(boot_hist, boot_qqplot, ncol = 2)


library(kableExtra)

df_ci <- data.frame(
  a = c("Percentile", "BCa", "First-order normal"),
  b = c(ci_percentile$per[4], ci_BCa$bca[4], ci_firstNorm$norm[2]),
  c = c(ci_percentile$t0, ci_BCa$t0, ci_firstNorm$t0),
  d = c(ci_percentile$per[5], ci_BCa$bca[5], ci_firstNorm$norm[3]))
df_ci$range <- df_ci$d - df_ci$b
df_ci$center <- (df_ci$b + df_ci$d)/2
df_ci$diff <- df_ci$c - df_ci$center
df_ci <- subset(df_ci, select = -c(c))
df_ci$shape <- (df_ci$d - boot_mean) / (boot_mean - df_ci$b)
```

```r
kable(df_ci,
      col.names = c("CI type", "Lower extreme", "Upper extreme", "Range",
                    "Center", "Difference", "Shape"),
      "latex", booktabs = T, align = "c",
      caption = "Comparative table of the various type of bootstrap CI.") %>%
  column_spec(c(1,3), border_right = T) %>%
  kable_styling(latex_options = "hold_position")


jackknife <- function(data, statistic) {

  n <- length(data)
  # Statistic applied on the starting dataset
  original_statistic <- statistic(data)
  jackknife_replicates <- rep(NA, n)

  for(i in 1:n) {
    # Value of the statistic for the sample without the i-th observation
    jackknife_replicates[i] <- statistic(data[-i])
  }

  # Formulas for the jackknife from the book (same result as Krzysztof's slides)
  jackknife_mean <- mean(jackknife_replicates)
  jackknife_variance <- (n-1)/n * sum((jackknife_replicates-jackknife_mean)^2)

  return( jackknife_variance )

}

jackknife_var_mean <- jackknife(data$Price, mean)
```