Lab 5

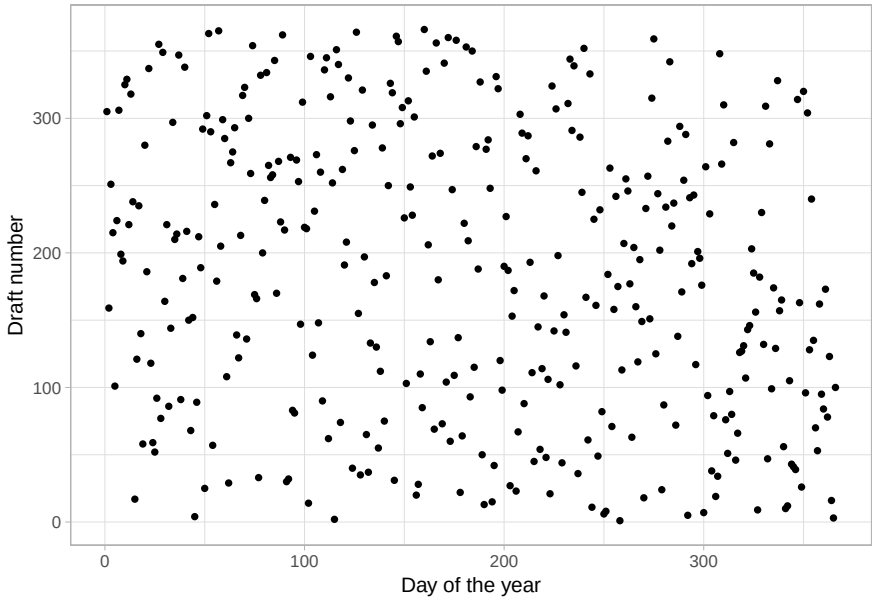
Stefano Toffol (steto820), Mim Kemal Tekin (mimte666)

02 April, 2019

Question 1

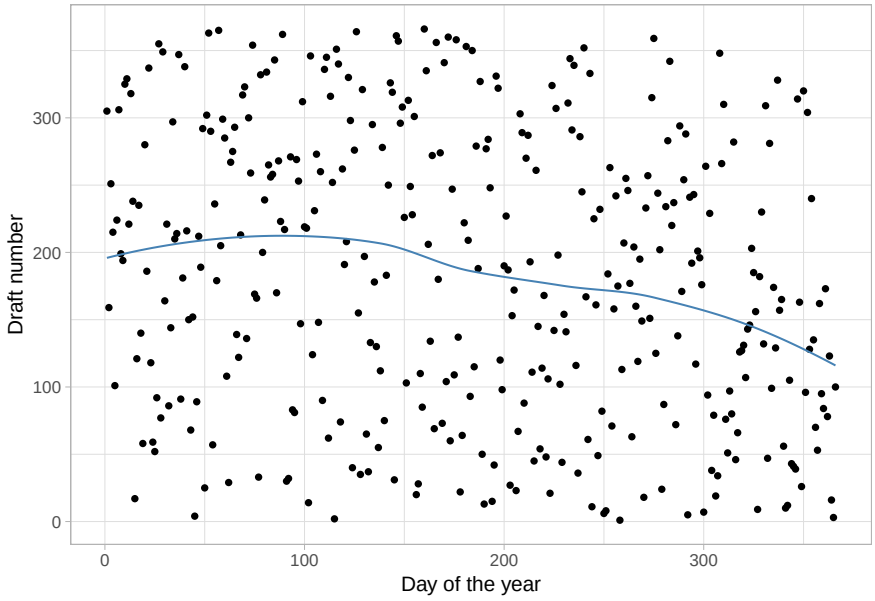
Make a scatterplot of Y versus X and conclude whether the lottery looks random.

Usign exclusively the scatterplot the data points appear distributed on the plane without a proper structure, filling all the space available. One may conclude that the points are actually random, but further analysis should be carried to exclude the actual existance of an underlying periodic process with random noise.



Compute an estimate $\hat{Y}$ of the expected response as a function of X by using a loess smoother (use `loess()`), put the curve $\hat{Y}$ versus X in the previous graph and state again whether the lottery looks random.

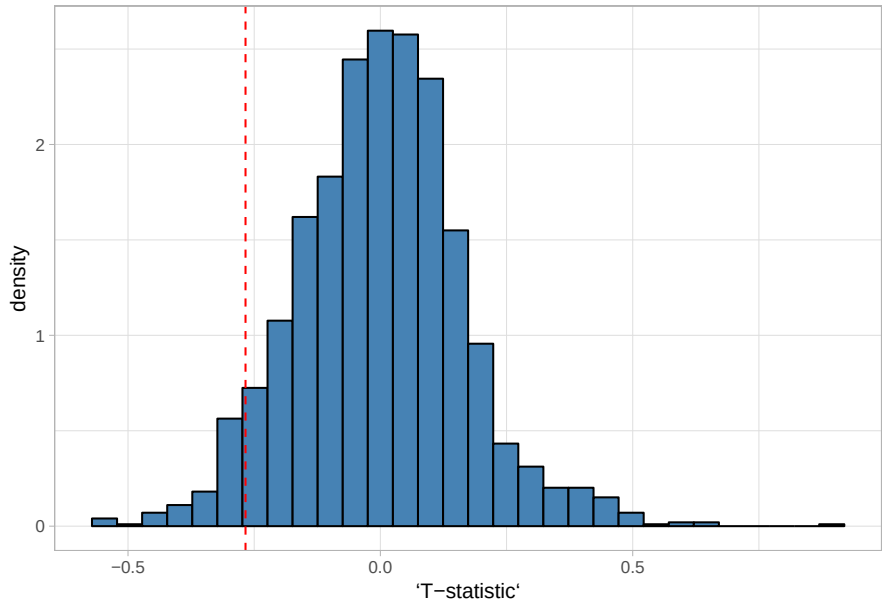
Adding the estimates of the `loess()` function we observe a slightly decreasing general trend in the data. The evidence against the hypothesis of randomness is weak but worth to be studied in deep.



To check whether the lottery is random, it is reasonable to use test statistics
$T = \frac{\hat{Y}(X_b) - \hat{Y}(X_a)}{X_b - X_a}$, where $X_b = argmax_X Y(X)$, $X_a = argmin_X Y(X)$
If this value is significantly greater than zero, then there should be a trend in the data and the lottery is not random. Estimate the distribution of T by using a non–parametric bootstrap with B = 2000 and comment whether the lottery is random or not. What is the p–value of the test?

```
my_statistic <- function(data, indexes_boot) {

  data_boot <- data[indexes_boot,]

  y <- data_boot$Draft_No
  x <- data_boot$Day_of_year

  ind_min <- which.min(y)
  ind_max <- which.max(y)

  model_loess <- loess(Draft_No ~ Day_of_year, data = data_boot)
  y_hat <- model_loess$fitted

  return( (y_hat[ind_max] - y_hat[ind_min]) / (x[ind_max] - x[ind_min]) )

}

B <- 2000
library(boot)
# Set seed for reproducibility purposes
set.seed(12345)
boot_obj <- boot(data, my_statistic, R = B)
df_boot <- as.data.frame(boot_obj$t)
colnames(df_boot) <- c("T-statistic")
centered_boot <- df_boot - mean(df_boot$`T-statistic`)

pval <- mean(abs(centered_boot) > abs(boot_obj$t0))
```

The non-parametric bootstrap estimates of the statistics appear to follow a slightly skewed distribution (heavier right tail). The point estimated of the T-statistic is -0.267179 (red line in the histogram). The observed value of the statistic falls into a relatively unlikely region of the distribution, even if we consider a two-sided test. Its *p-value* is in fact 0.101 that, using a rigid threshold of significance $\alpha = 0.05$ will lead us to not reject the null hypothesis, however its proximity to the rejection region makes it difficult to draw conclusions already.

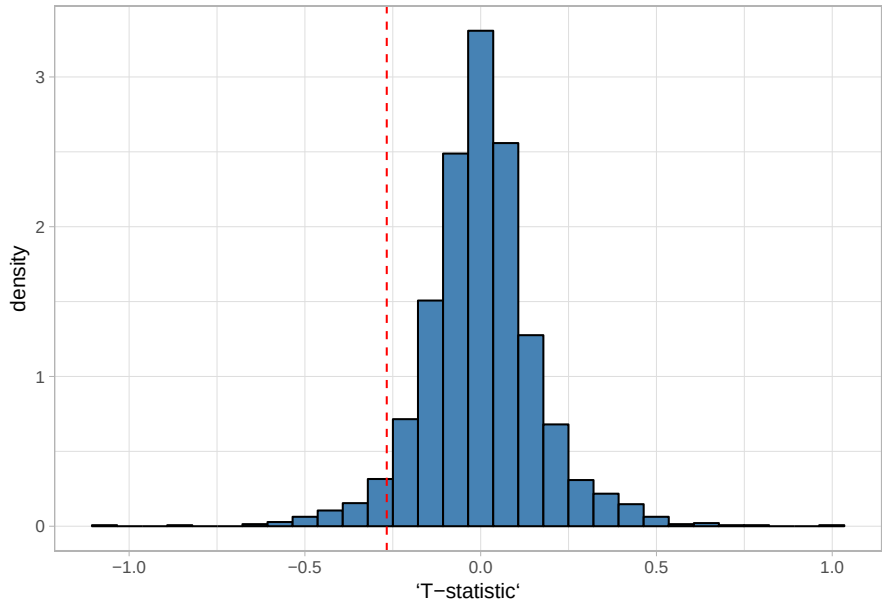


Implement a function depending on data and B that tests the hypothesis
$\begin{cases} H_0 : \text{Lottery is random} \\ H_1 : \text{Lottery is non-random} \end{cases}$
by using a permutation test with statistics T. The function is to return the p–value of this test. Test this function on our data with B = 2000.

```
permutation_test <- function(x, y) {

  n <- length(x)

  permutation_ind <- sample(1:n)

  y_permut <- y[permutation_ind]

  ind_min <- which.min(y_permut)
  ind_max <- which.max(y_permut)

  data_permut <- data.frame(x = x, y_permut = y_permut)
  data_permut$y_permut <- y_permut

  model_loess_permut <- loess(y_permut ~ x, data = data_permut)
  y_hat <- model_loess_permut$fitted

  return( (y_hat[ind_max] - y_hat[ind_min]) / (x[ind_max] - x[ind_min]) )

}

set.seed(123456)

B <- 2000
boot_permut <- 1:B
boot_permut <- sapply(boot_permut, function(nothing) permutation_test(x, y))
```

```
centered_permut <- boot_permut - mean(boot_permut)
permuted_for_pval <- abs(centered_permut)
permuted_stat <- abs(boot_obj$t0)

pval_permut <- mean(permuted_for_pval > permuted_stat)
```

Changing type of test does return a similar but still helpful result. In fact the permutation test performed using the code above returns an almost equal *p-value*, this time of 0.0935. Even though it did not change much, the *p-value* suggests stronger evidence for not rejecting the null hypothesis, having received similar findings with the standard non-parametric one. In other words, draft numbers and date of birth seem exchangeable. Moreover in this case the distribution of the statistic seems more symmetric than the previous one.

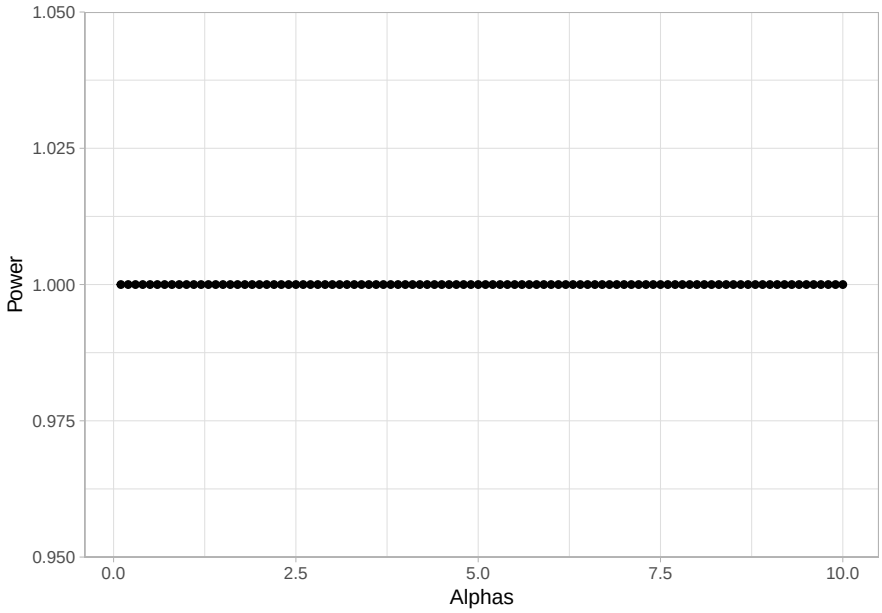


Make a crude estimate of the power of the test constructed in Step 4: (a) Generate (an obviously non-random) dataset with $n = 366$ observations by using same X as in the original data set and $Y(x) = \max(0, \min(\alpha x + \beta, 366)),$ where $\alpha = 0.1$ and $\beta \sim N(183, \sigma = 10)$ (b) Plug these data into the permutation test with B = 200 and note whether it was rejected. (c) Repeat Steps 5a-5b for $\alpha = 0.2, 0.3, ..., 10$. What can you say about the quality of your test statistics considering the value of the power?

```
alphas <- seq(0.1, 10, by = 0.1)
B <- 200

pvalues_power <-
  sapply(alphas, function(a) {

      fake_y <- pmax(0, pmin( a*x + rnorm(n, 183, 10), 366))
      data_fake <- data.frame(Draft_No = fake_y, Day_of_year = x)
      t_0 <- my_statistic(data_fake, 1:n)

      bootstrap_iter <- 1:B
      bootstrap_iter <- sapply(bootstrap_iter,
                               function(nothing) permutation_test(x, fake_y))
```

```
      centered_iter <- bootstrap_iter - mean(bootstrap_iter)
      mean(abs(centered_iter) > abs(t_0))
    }
  )

power <- mean(pvalues_power<0.05)
```

This rought estimate of the power function shows how the statistic is able to correctly detect actual non-random data. Even though the non-randomness of the phenomena was quite evident even without the use of a test, the statistic reassure us on its capabilities, spotting every single case of fake data. Further experiments and different kind of data should be needed, but we can state that the test does indeed seem trustable and therefore we feel safer in not rejecting the null hypothesis. The results are even more remarkable if we consider the usual low power of this family of tests.



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 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 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.

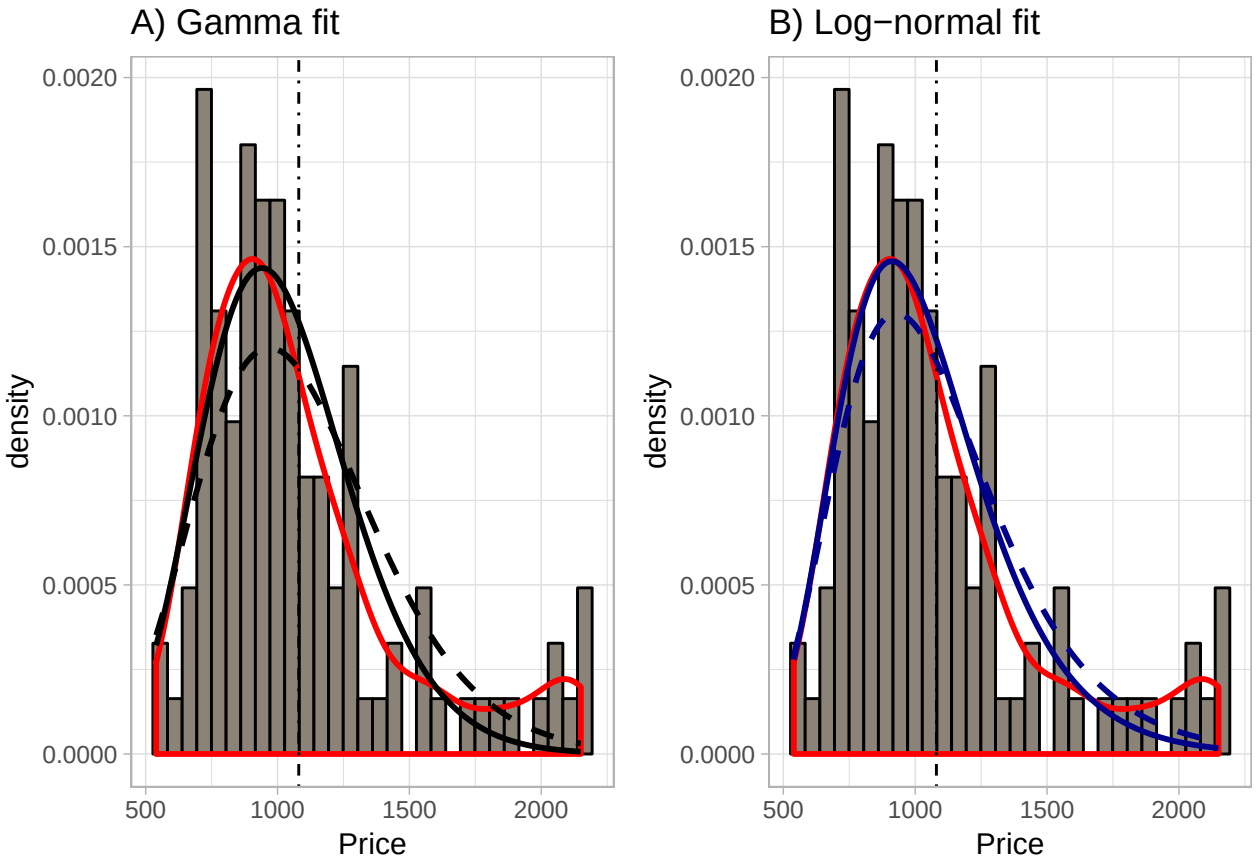


Figure 1: Density histogram of the response variable. The line in red is the estimated kernel density of the data.
A) The black lines rapresent the density of a gamma variable; the dashed one is drawn from the ML estimates of the parameters, the solid one from the MGE estimates of the parameters.
B) 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:

```
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 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. 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.

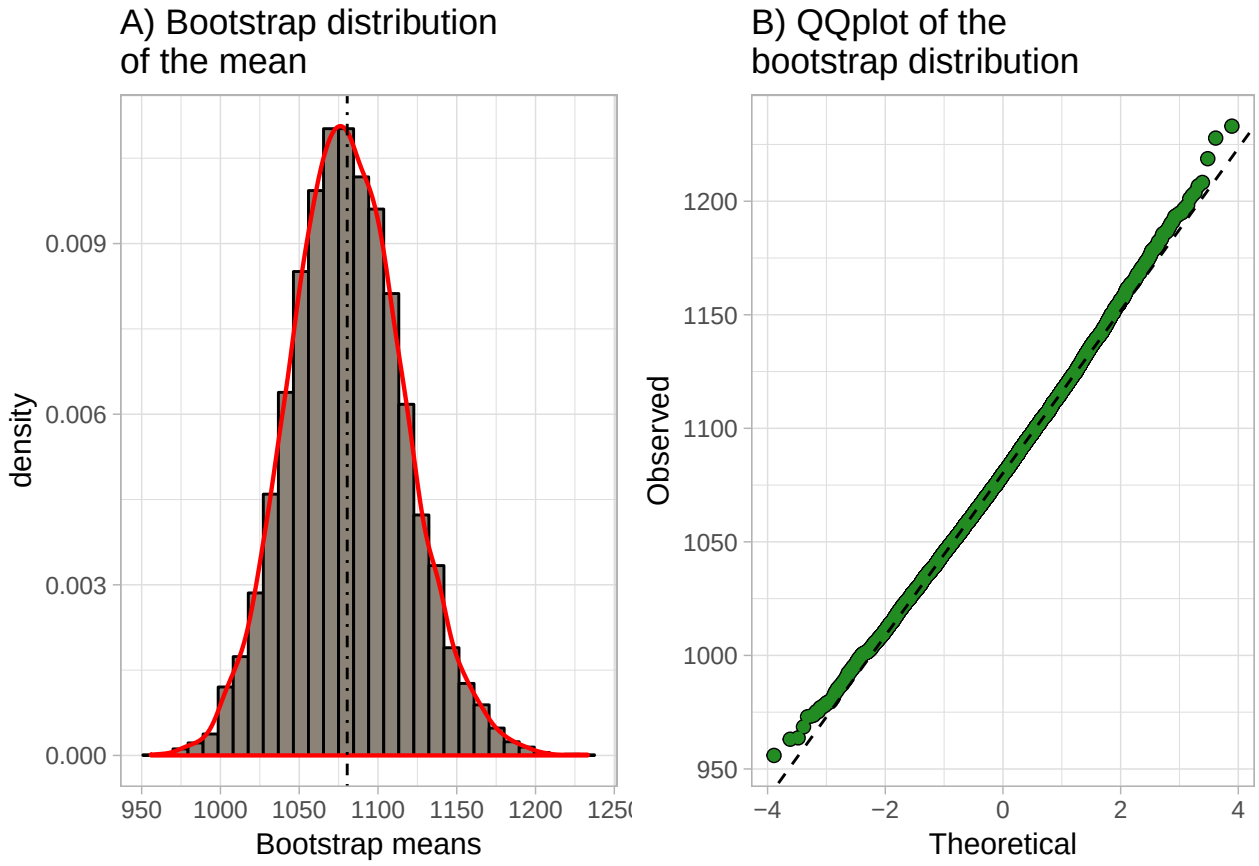


Figure 2: Histrogram and QQplot of the bootstrap distribution of the mean.

Table 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%")


data <- readxl::read_xls("lottery.xls")
y <- data$Draft_No
x <- data$Day_of_year
n <- length(x)


library(ggplot2)
p1 <- ggplot() +
  geom_point(aes(x = Day_of_year, y = Draft_No), data, shape = 16) +
  labs(x = "Day of the year", y = "Draft number") +
  theme_light()
p1


library(nlme)
model_loess <- loess(Draft_No ~ Day_of_year, data = data)
y_hat <- model_loess$fitted
x_loess <- model_loess$x[,1]
df_loess <- data.frame(x_loess = x_loess, y_hat = y_hat)

p1 <- p1 + geom_line(aes(x = x_loess, y = y_hat), df_loess, col = "steelblue")
p1


my_statistic <- function(data, indexes_boot) {

  data_boot <- data[indexes_boot,]

  y <- data_boot$Draft_No
  x <- data_boot$Day_of_year

  ind_min <- which.min(y)
  ind_max <- which.max(y)

  model_loess <- loess(Draft_No ~ Day_of_year, data = data_boot)
  y_hat <- model_loess$fitted

  return( (y_hat[ind_max] - y_hat[ind_min]) / (x[ind_max] - x[ind_min]) )

}

B <- 2000
library(boot)
# Set seed for reproducibility purposes
set.seed(12345)
boot_obj <- boot(data, my_statistic, R = B)
df_boot <- as.data.frame(boot_obj$t)
```

```r
colnames(df_boot) <- c("T-statistic")
centered_boot <- df_boot - mean(df_boot$`T-statistic`)

pval <- mean(abs(centered_boot) > abs(boot_obj$t0))


ggplot(centered_boot, aes(x = `T-statistic`, y = ..density..)) +
  geom_histogram(col = "black", fill = "steelblue") +
  geom_vline(aes(xintercept = boot_obj$t0), col = "red", lty = 2) +
  theme_light()


permutation_test <- function(x, y) {

  n <- length(x)

  permutation_ind <- sample(1:n)

  y_permut <- y[permutation_ind]

  ind_min <- which.min(y_permut)
  ind_max <- which.max(y_permut)

  data_permut <- data.frame(x = x, y_permut = y_permut)
  data_permut$y_permut <- y_permut

  model_loess_permut <- loess(y_permut ~ x, data = data_permut)
  y_hat <- model_loess_permut$fitted

  return( (y_hat[ind_max] - y_hat[ind_min]) / (x[ind_max] - x[ind_min]) )

}

set.seed(123456)

B <- 2000
boot_permut <- 1:B
boot_permut <- sapply(boot_permut, function(nothing) permutation_test(x, y))

centered_permut <- boot_permut - mean(boot_permut)
permuted_for_pval <- abs(centered_permut)
permuted_stat <- abs(boot_obj$t0)

pval_permut <- mean(permuted_for_pval > permuted_stat)


centered_permut <- as.data.frame(centered_permut)
colnames(centered_permut) <- c("T-statistic")

ggplot(centered_permut, aes(x = `T-statistic`, y = ..density..)) +
  geom_histogram(col = "black", fill = "steelblue") +
  geom_vline(aes(xintercept = boot_obj$t0), col = "red", lty = 2) +
  theme_light()
```

```r
alphas <- seq(0.1, 10, by = 0.1)
B <- 200

pvalues_power <-
  sapply(alphas, function(a) {

      fake_y <- pmax(0, pmin( a*x + rnorm(n, 183, 10), 366))
      data_fake <- data.frame(Draft_No = fake_y, Day_of_year = x)
      t_0 <- my_statistic(data_fake, 1:n)

      bootstrap_iter <- 1:B
      bootstrap_iter <- sapply(bootstrap_iter,
                               function(nothing) permutation_test(x, fake_y))

      centered_iter <- bootstrap_iter - mean(bootstrap_iter)
      mean(abs(centered_iter) > abs(t_0))
    }
  )

power <- mean(pvalues_power<0.05)


df_plot <- data.frame(Alphas = alphas, Power = power)

ggplot(df_plot, aes(x = Alphas, y = Power)) +
  geom_point() +
  geom_line() +
  theme_light()


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


# Read the data
data <- readxl::read_xls("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")
```

```r
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)
```

```r
# 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")


# 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()
```

```r
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)

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)
```