Computational Statistics - Lab 05

Annalena Erhard (anner218) and Maximilian Pfundstein (maxpf364)

2019-02-15

Contents

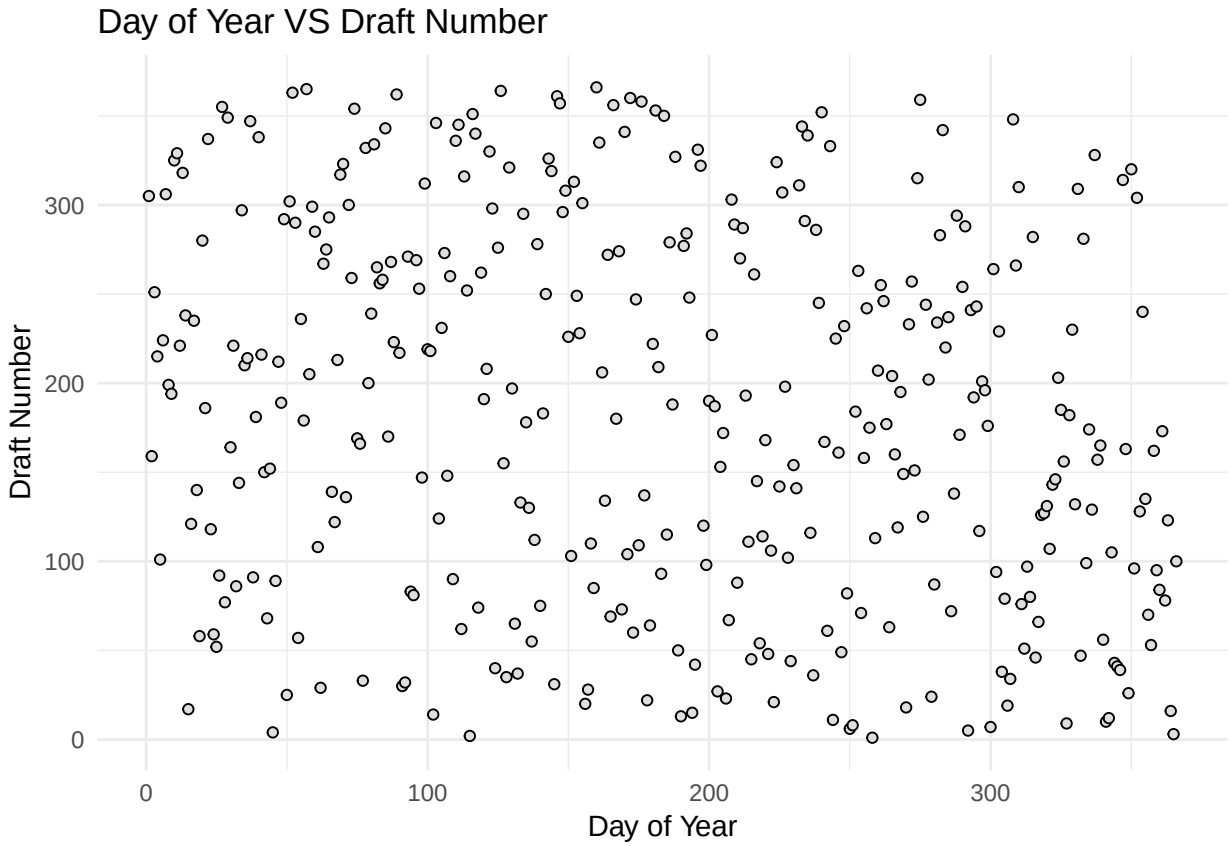
1 Question 1: Hypothesis testing

In 1970, the US Congress instituted a random selection process for the military draft. All 366 possible birth dates were placed in plastic capsules in a rotating drum and were selected one by one. The first date drawn from the drum received draft number one, the second date drawn received draft number two, etc. Then, eligible men were drafted in the order given by the draft number of their birth date. In a truly random lottery there should be no relationship between the date and the draft number. Your task is to investigate whether or not the draft numbers were randomly selected. The draft numbers ($Y = DraftNo$) sorted by day of year ($X =$ Day of year) are given in the file `lottery.xls`.

Day	Month	Mo.Number	Day_of_year	Draft_No
1	Jan	1	1	305
2	Jan	1	2	159
3	Jan	1	3	251
4	Jan	1	4	215
5	Jan	1	5	101
6	Jan	1	6	224

1.1 Scatterplot of Day of Year vs Draft Number

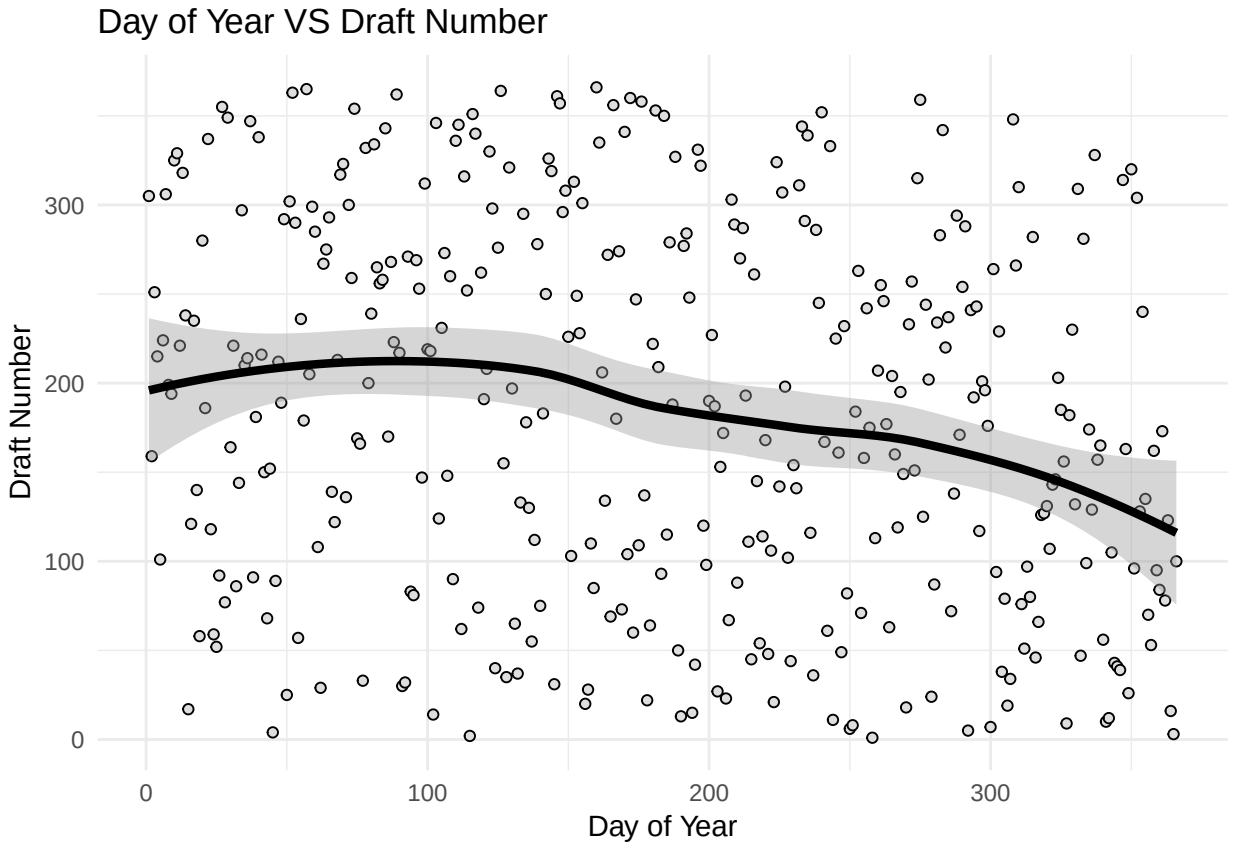
Make a scatterplot of Y versus X and conclude whether the lottery looks random.



Answer: In the plot, it the lottery appears to be random, because of the relatively even distribution of the points over the area.

1.2 Loess Smoother

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.



Answer: Including the insights from this smoothing we see, that the lottery actually doesn not look random. There are two reasons for this:

- The confidence interval of the curve is relatively small, especially compared to the spread of the data points.
- The data points follow the pattern of the curve slightly, especially above the line.

1.3 Randomness of Lottery using a Test Statistics

To check whether the lottery is random, it is reasonable to use test statistics

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

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?

Answer:

```
data = data.frame(X = lottery$Day_of_year, Y = lottery$Draft_No)

test_statistics = function(X, Y, Y_hat) {

  b_index = which.max(Y)
  a_index = which.min(Y)
```

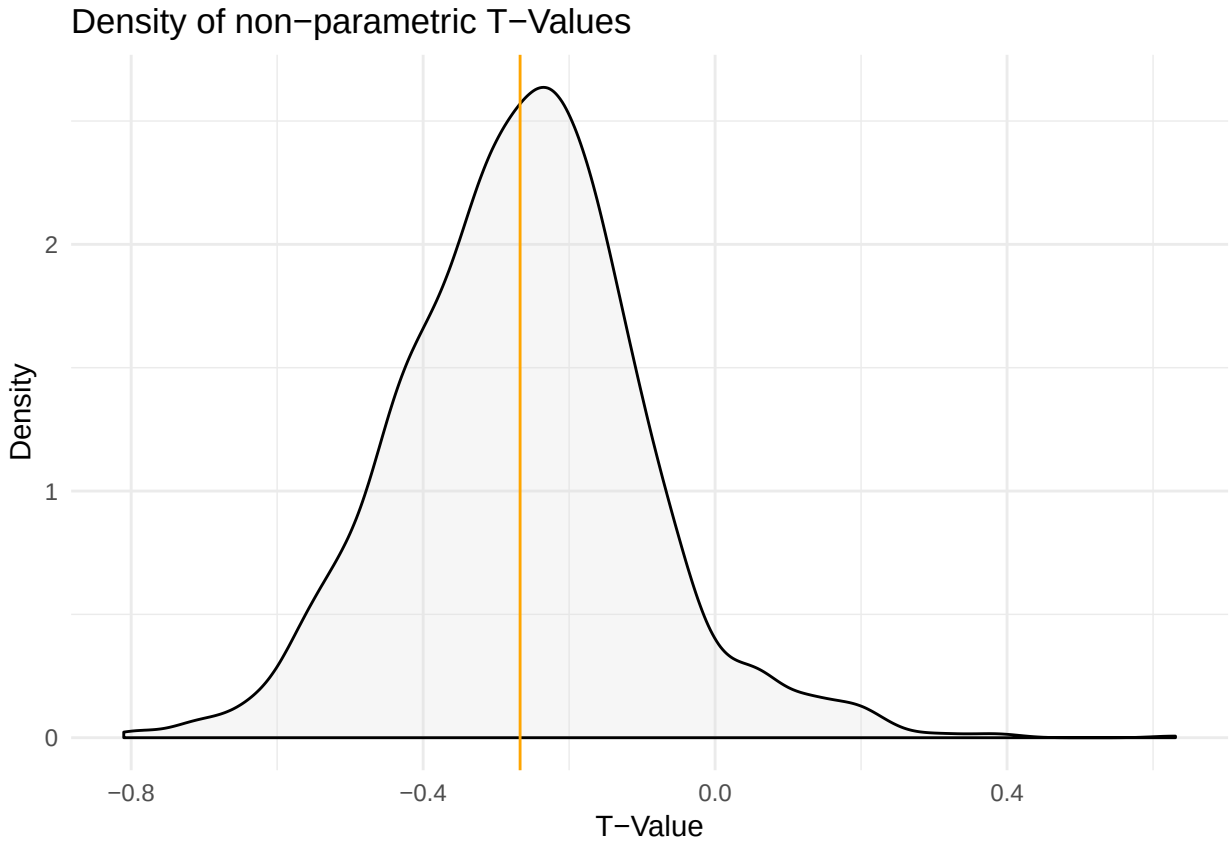
```
  return((Y_hat[b_index] - Y_hat[a_index]) / (X[b_index] - X[a_index]))
}

f = function(data, ind) {
  data1 = data[ind,]
  model = loess(Draft_No ~ Day_of_year, data1)

  T_value =
    test_statistics(data1$Day_of_year, data1$Draft_No, Y_hat = model$fitted)

  return(T_value)
}

# T(D) for te original data
data$Y_hat = loess(Draft_No ~ Day_of_year, lottery)$fitted
T_value_original = test_statistics(data$X, data$Y, data$Y_hat)

# T for the bootstrapped samples
nonparam_bootstrap =
  boot(lottery, statistic = f, R = 2000, parallel = "multicore")
p_value_original = mean(nonparam_bootstrap$t > 0)
```

```
print(T_value_original)
```

```
## [1] -0.2671794
```

```
print(p_value_original)
```

```
## [1] 0.0495
```



1.4 Hypothesis Testing

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

H_0: Lottery is random versus H_1: Lottery is non-random

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 $B = 2000$.

```
test_hypothesis = function (data_input, statistics, B = 2000) {

  t_values = c()

  for (i in 1:B) {
    ind = sample(1:nrow(data_input))
    data_input$X = data_input$X[ind]

    model = loess(Y ~ X, data_input)

    t_values[i] = statistics(data_input$X,
                             data_input$Y, Y_hat = model$fitted)
  }

  return(mean(t_values > 0))
}

p_value_permutated = test_hypothesis(data, test_statistics, 2000)
```

```
print(p_value_permutated)
```

```
## [1] 0.5035
```

1.5 Crude Estimate of the Power

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 $\beta \sim N(183, sd = 10)$.
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?

```
simulate_data = function(X, hypothesis = test_hypothesis,
                         statistics = test_statistics, alpha = 0.1,
                         beta_mean = 183, beta_sd = 10, b = 200, limit = 366) {

  artificial = function(X, alpha) {
    beta = rnorm(n = nrow(lottery), mean = beta_mean, sd = beta_sd)
    return(max(0, min(alpha * X + beta, limit)))
  }

  X_dataframe = X
  Y_dateframe = sapply(X, artificial, alpha)

  data_artificial = data.frame(X_dataframe, Y_dateframe)
  colnames(data_artificial) = c("X", "Y")
  return(test_hypothesis(data_artificial, test_statistics, b))
}

alphas = seq(from = 0.1, to = 10.0, by = 0.1)

no_cores = detectCores()
cl = makeCluster(no_cores)

clusterExport(cl, list("simulate_data", "lottery", "test_hypothesis",
                       "test_statistics"))

simulated_p_values =
  parSapply(cl, alphas, FUN = function(alpha) {
    simulate_data(alpha = alpha, X = lottery$Day_of_year)
  })

stopCluster(cl)

kable(head(simulated_p_values))
```

x
0.560
0.545
0.500
0.520

x
0.540
0.515

2 Question 2: Bootstrap, jackknife and confidence intervals

The data you are going to continue analyzing is the database of home prices in Albuquerque, 1993. The variables present are `Price`; `SqFt`: the area of a house; `FEATS`: number of features such as dishwasher, refrigerator and so on; `Taxes`: annual taxes paid for the house. Explore the file `prices1.xls`.

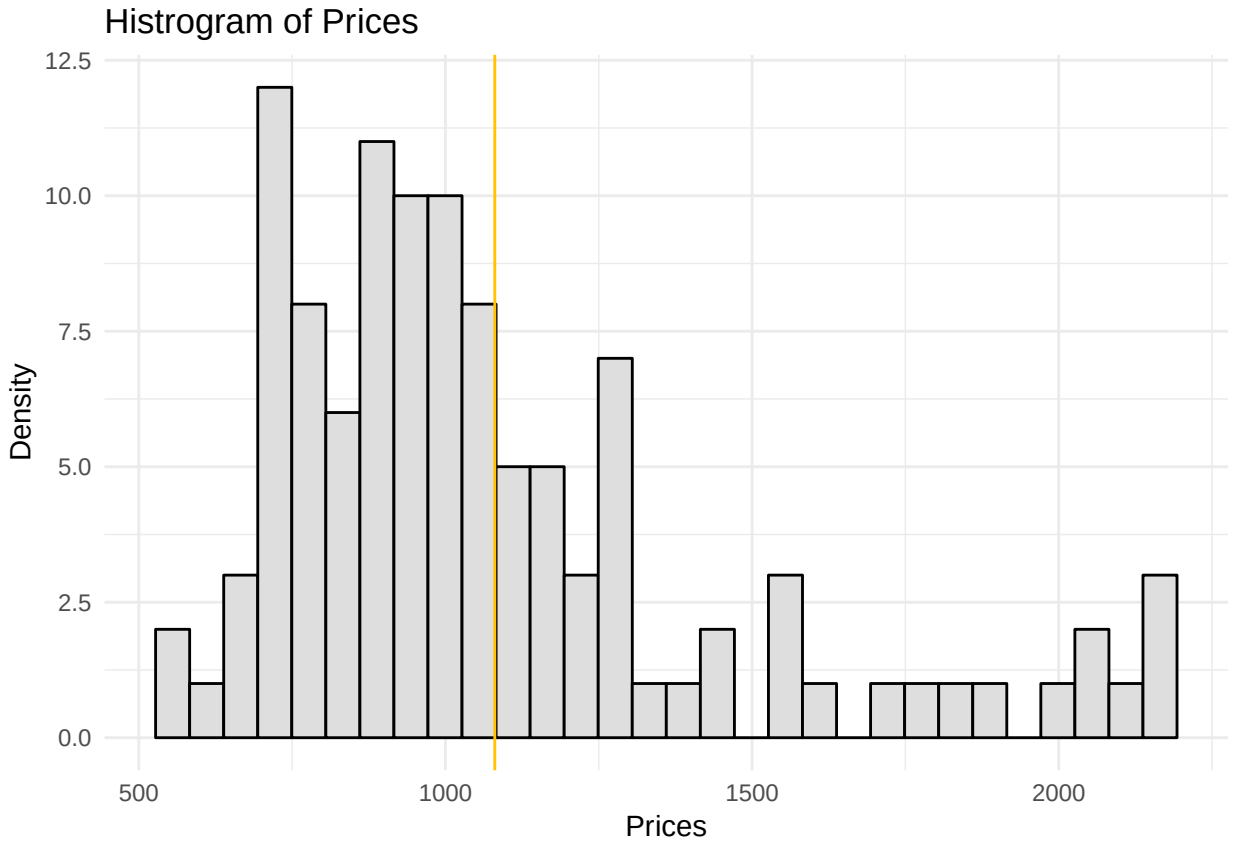
Price	SqFt	FEATS	Taxes
2050	2650	7	1639
2080	2600	4	1088
2150	2664	5	1193
2150	2921	6	1635
1999	2580	4	1732
1900	2580	4	1534

2.1 Histrogram of the Price

Task: Plot the histogram of Price. Does it remind any conventional distribution? Compute the mean price.

Answer: It looks like a Chi-Squared distribution with $k = 4$. The mean of the price is given by:

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

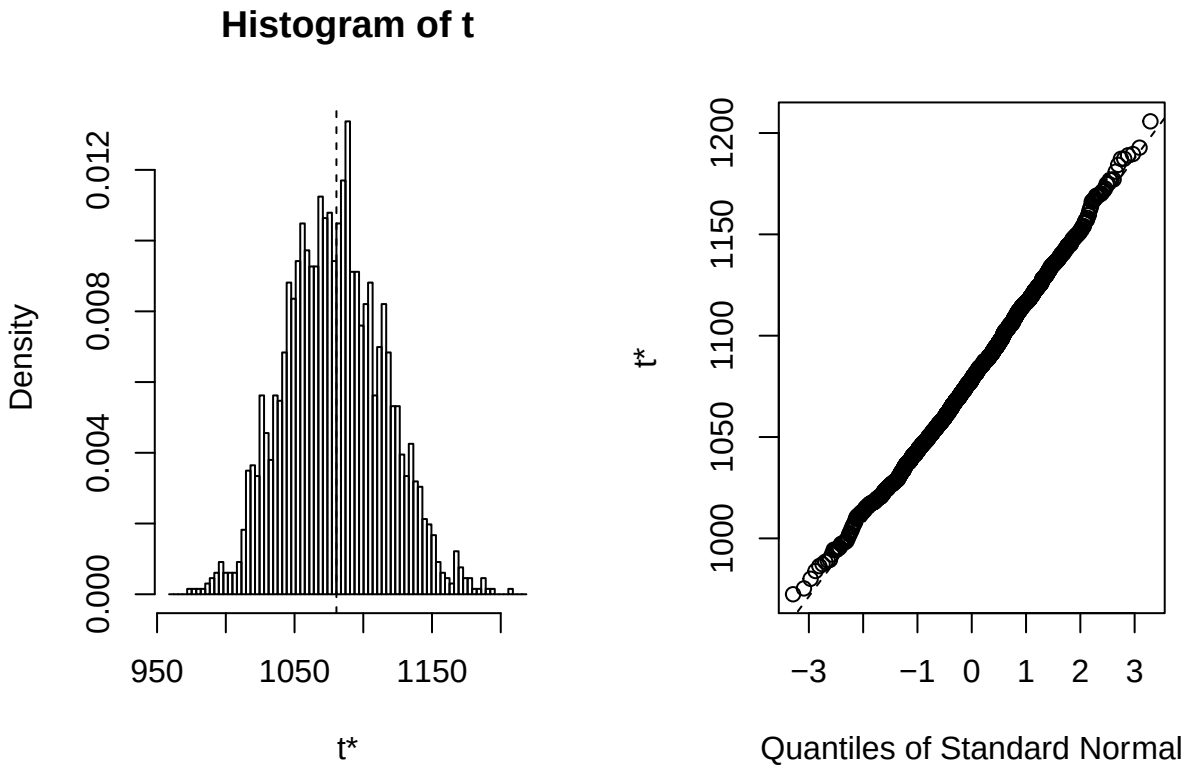


2.2 Distribution Mean and Variance Estimate and Confidence Intervals

Estimate the distribution of the mean price of the house using bootstrap. Determine the bootstrap bias-correction and the variance of the mean price. Compute a 95% confidence interval for the mean price using bootstrap percentile, bootstrap BCa, and first-order normal approximation.

(**Hint**: use `boot()`, `boot.ci()`, `plot.boot()`, `print.bootci()`)

```
f_prices = function(data, ind) {
  data1 = data[ind,]

  return(mean(data1$Price))
}

house_bootstrap = boot(prices, statistic = f_prices, R = 2000,
                       parallel = "multicore")

plot(house_bootstrap)
```



The estimate of the mean price is:

```
## [1] 1079.409
```

The bootstrap bias-correction is given by the following values. The first one is the bias-correction and the second one is the bias corrected mean.

```
## [1] 1.063914
```

```
## [1] 1081.537
```

The variance of the mean price is given by:

```
## [1] 1298.067
```

Now we will create the 95% confidence intervals.

```
confidence_interval = boot.ci(house_bootstrap)
print(confidence_interval)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = house_bootstrap)
##
## Intervals :
## Level      Normal              Basic
## 95%   (1011, 1152 )   (1011, 1146 )
##
## Level     Percentile            BCa
## 95%   (1015, 1150 )   (1018, 1158 )
## Calculations and Intervals on Original Scale
```

As this output does not include the BCa, we will print the intervals manually. Note that the last to values in each row represent the confidence interval.

```
confidence_interval$percent
```

```
##      conf
## [1,] 0.95 50.03 1950.97 1014.816 1150.195
```

```
confidence_interval$bca
```

```
##      conf
## [1,] 0.95 72.05 1968.22 1017.919 1158.451
```

```
confidence_interval$normal
```

```
##      conf
## [1,] 0.95 1010.922 1152.152
```

2.3 Variance Estimate using Jackknife

Estimate the variance of the mean price using the jackknife and compare it with the bootstrap estimate.

```
f_prices_jackknife = function(ind, data) {
  data1 = data[-ind,]
  return(mean(data1$Price))
}

# First create the statistics using jackknife
n = length(prices$Price)

indices = seq(from = 1, to = n, by = 1)
jackknife_statistics = sapply(indices, f_prices_jackknife, prices)

# For the variance estimate we will first calculate Ti_star
Ti_star = sapply(jackknife_statistics, FUN = function(tdi, n, prices_mean) {
  return(n * prices_mean - ((n - 1) * tdi))
}, n = n, prices_mean = prices_mean)

# Now we calculate J_T
J_T = mean(Ti_star)

# And now we can calculate the Variance
variance_jackknife = 1 / (n * (n - 1)) * sum((Ti_star - J_T)^2)

print(variance_jackknife)
```

```
## [1] 1320.911
```

We see that the variance using jackknife is slightly higher compared to using bootstrap.

2.4 Confidence Intervals with Respect to Length and Location

Task: Compare the confidence intervals obtained with respect to their length and the location of the estimated mean in these intervals.

Answer: The following plots show the confidence interval and the estimated mean using bootstrap. The mean includes the bias correction.

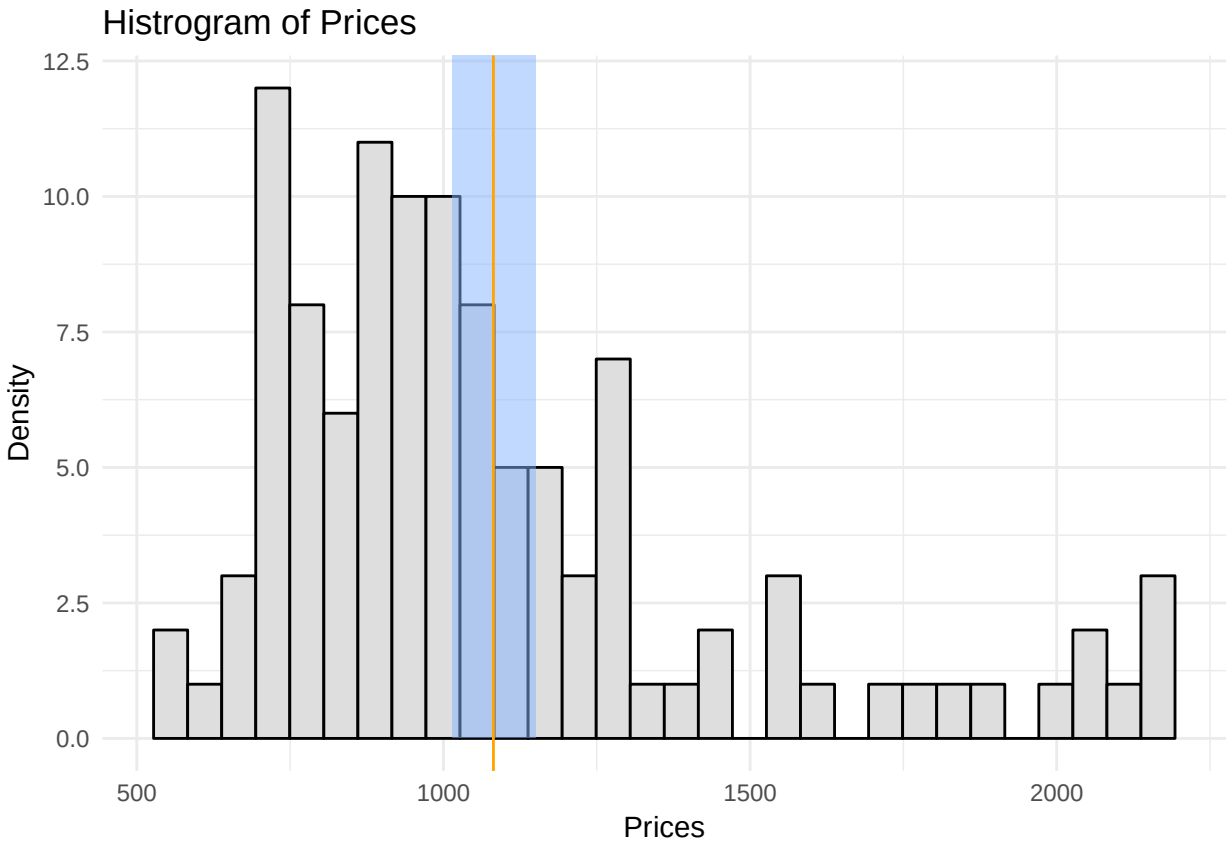
2.4.1 Boostrap Percentile

The length of the CI is:

```
## [1] 135.3792
```

The mean is located the following "percent" of the CI range.

```
## [1] 49.28411
```



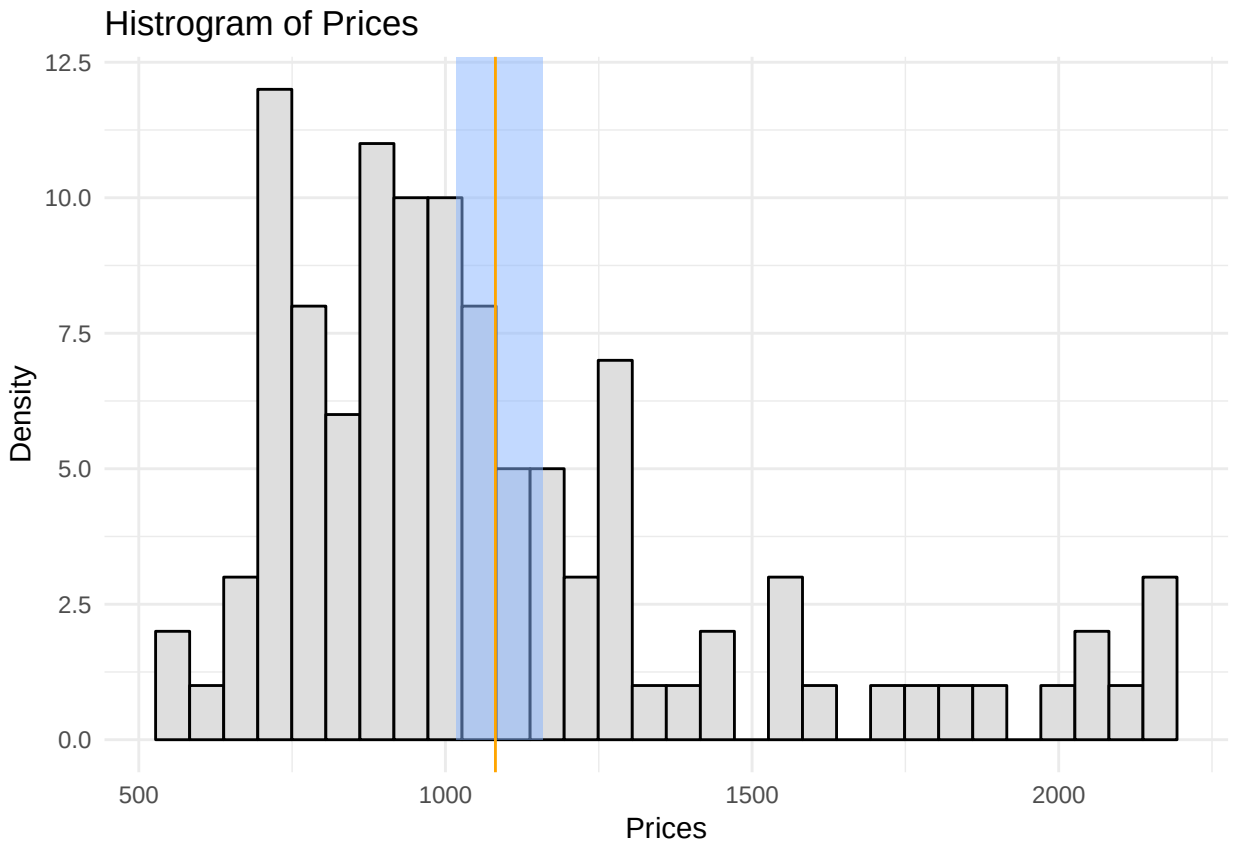
2.4.2 Bootstrap BCa

The length of the CI is:

```
## [1] 140.5324
```

The mean is located the following "percent" of the CI range.

```
## [1] 45.26901
```



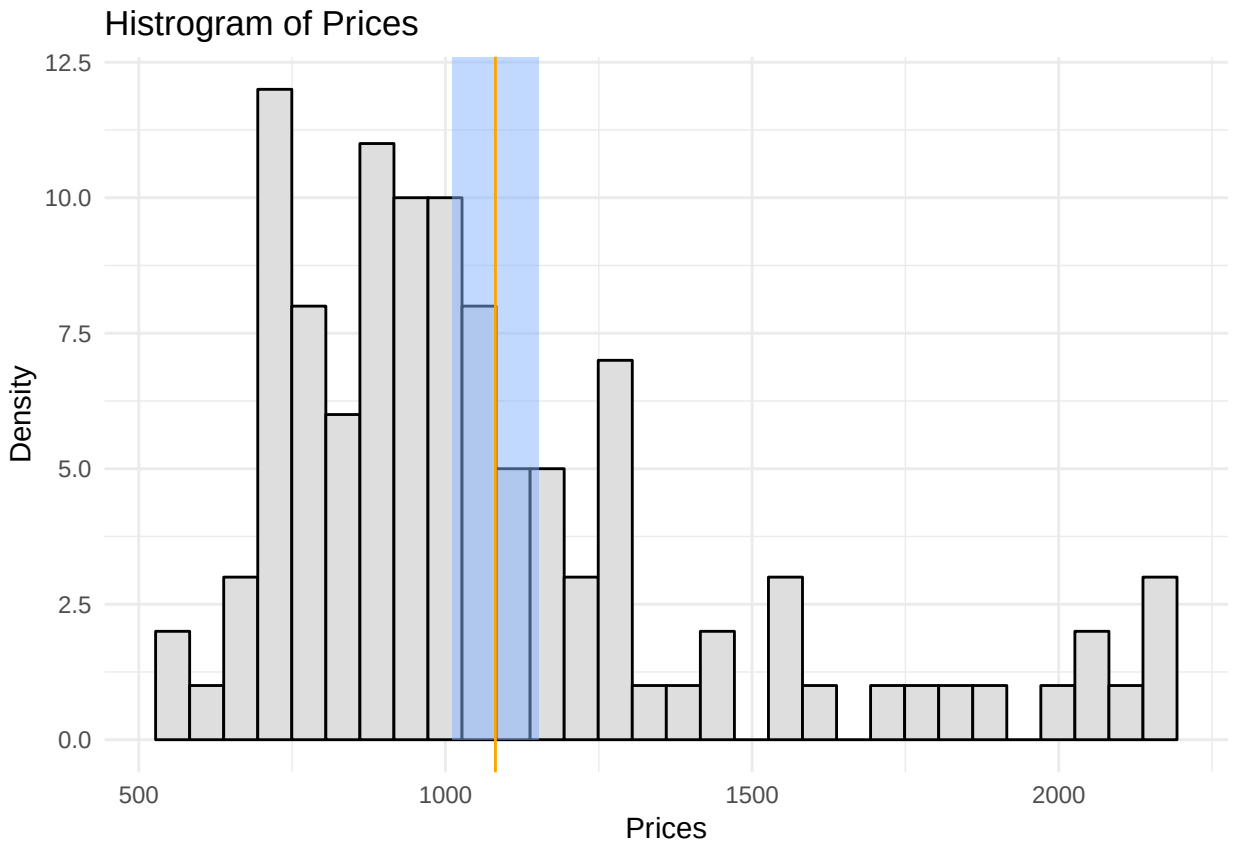
2.4.3 First-Order Normal Approximation

The length of the CI is:

```
## [1] 141.2299
```

The mean is located the following "percent" of the CI range.

```
## [1] 50
```



3 Source Code

```
knitr::opts_chunk$set(echo = TRUE, cache = FALSE, include = TRUE, eval = TRUE)
library(knitr)
library(readxl)
library(ggplot2)
library(gridExtra)
library(boot)
library(parallel)

set.seed(12345)


lottery = read_xls("lottery.xls")
kable(head(lottery))


ggplot(lottery)+
  geom_point(aes(x = Day_of_year, y = Draft_No), color = "black", fill = "#dedede", shape = 21) +
  labs(title = "Day of Year VS Draft Number",
       y = "Draft Number",  x = "Day of Year", color = "Legend") +
  theme_minimal()


ggplot(lottery)+
```

```
  geom_point(aes(x = Day_of_year, y = Draft_No), color = "black", fill = "#dedede", shape = 21) +
  geom_smooth(mapping = aes(x = Day_of_year, y = Draft_No),
              method = "loess", size = 1.5, color = "#000000") +
  labs(title = "Day of Year VS Draft Number",
       y = "Draft Number",  x = "Day of Year", color = "Legend") +
  theme_minimal()


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

test_statistics = function(X, Y, Y_hat) {

  b_index = which.max(Y)
  a_index = which.min(Y)

  return((Y_hat[b_index] - Y_hat[a_index]) / (X[b_index] - X[a_index]))
}

f = function(data, ind) {
  data1 = data[ind,]
  model = loess(Draft_No ~ Day_of_year, data1)

  T_value =
    test_statistics(data1$Day_of_year, data1$Draft_No, Y_hat = model$fitted)

  return(T_value)
}

# T(D) for te original data
data$Y_hat = loess(Draft_No ~ Day_of_year, lottery)$fitted
T_value_original = test_statistics(data$X, data$Y, data$Y_hat)

# T for the bootstrapped samples
nonparam_bootstrap =
  boot(lottery, statistic = f, R = 2000, parallel = "multicore")
p_value_original = mean(nonparam_bootstrap$t > 0)


print(T_value_original)
print(p_value_original)


df = data.frame(nonparam_bootstrap$t)

ggplot(df) +
  geom_density(aes(x = nonparam_bootstrap.t), color = "black",
               fill = "#dedede", alpha = 0.25) +
  geom_vline(aes(xintercept = T_value_original), color = "orange") +
  labs(title = "Density of non-parametric T-Values",
       y = "Density",  x = "T-Value", color = "Legend") +
  theme_minimal()
```

```r
test_hypothesis = function (data_input, statistics, B = 2000) {

  t_values = c()

  for (i in 1:B) {
    ind = sample(1:nrow(data_input))
    data_input$X = data_input$X[ind]

    model = loess(Y ~ X, data_input)

    t_values[i] = statistics(data_input$X,
                             data_input$Y, Y_hat = model$fitted)
  }

  return(mean(t_values > 0))
}

p_value_permutated = test_hypothesis(data, test_statistics, 2000)

print(p_value_permutated)

simulate_data = function(X, hypothesis = test_hypothesis,
                         statistics = test_statistics, alpha = 0.1,
                         beta_mean = 183, beta_sd = 10, b = 200, limit = 366) {

  artificial = function(X, alpha) {
    beta = rnorm(n = nrow(lottery), mean = beta_mean, sd = beta_sd)
    return(max(0, min(alpha * X + beta, limit)))
  }

  X_dataframe = X
  Y_dateframe = sapply(X, artificial, alpha)

  data_artificial = data.frame(X_dataframe, Y_dateframe)
  colnames(data_artificial) = c("X", "Y")
  return(test_hypothesis(data_artificial, test_statistics, b))
}

alphas = seq(from = 0.1, to = 10.0, by = 0.1)

no_cores = detectCores()
cl = makeCluster(no_cores)

clusterExport(cl, list("simulate_data", "lottery", "test_hypothesis",
                       "test_statistics"))

simulated_p_values =
  parSapply(cl, alphas, FUN = function(alpha) {
    simulate_data(alpha = alpha, X = lottery$Day_of_year)
  })

stopCluster(cl)
```

```r
kable(head(simulated_p_values))


prices = read_xls("prices1.xls")
kable(head(prices))


prices_mean = mean(prices$Price)
print(prices_mean)


ggplot(prices) +
  geom_histogram(aes(x = prices$Price),
                 color = "black", fill = "#dedede") +
  geom_vline(aes(xintercept = prices_mean), color = "#FFC300") +
  labs(title = "Histrogram of Prices",
       y = "Density",
  x = "Prices", color = "Legend") +
  theme_minimal()


f_prices = function(data, ind) {
  data1 = data[ind,]

  return(mean(data1$Price))
}

house_bootstrap = boot(prices, statistic = f_prices, R = 2000,
                       parallel = "multicore")

plot(house_bootstrap)


bootstrap_mean_price = mean(house_bootstrap$t)
print(bootstrap_mean_price)


print(prices_mean - bootstrap_mean_price)
print(2 * prices_mean - bootstrap_mean_price)


bootstrap_variance_price = as.numeric(var(house_bootstrap$t))
# bootstrap_mean_price = 1 / (B-1) * sum((house_bootstrap$t -
#                                         mean(house_bootstrap$t))^2)

print(bootstrap_variance_price)


confidence_interval = boot.ci(house_bootstrap)
print(confidence_interval)


confidence_interval$percent
```

```r
confidence_interval$bca
confidence_interval$normal


f_prices_jackknife = function(ind, data) {
  data1 = data[-ind,]
  return(mean(data1$Price))
}

# First create the statistics using jackknife
n = length(prices$Price)

indices = seq(from = 1, to = n, by = 1)
jackknife_statistics = sapply(indices, f_prices_jackknife, prices)

# For the variance estimate we will first calculate Ti_star
Ti_star = sapply(jackknife_statistics, FUN = function(tdi, n, prices_mean) {
  return(n * prices_mean - ((n - 1) * tdi))
}, n = n, prices_mean = prices_mean)

# Now we calculate J_T
J_T = mean(Ti_star)

# And now we can calculate the Variance
variance_jackknife = 1 / (n * (n - 1)) * sum((Ti_star - J_T)^2)

print(variance_jackknife)


boostrap_estimated_mean_corrected = (2 * prices_mean - bootstrap_mean_price)


print(confidence_interval$percent[5] - confidence_interval$percent[4])


(boostrap_estimated_mean_corrected - confidence_interval$percent[4]) / (confidence_interval$percent[5]


ggplot(prices) +
  geom_histogram(aes(x = prices$Price),
                 color = "#000000", fill = "#dedede") +
  annotate("rect", xmin=confidence_interval$percent[4], xmax=confidence_interval$percent[5], ymin=0, yma
  geom_vline(aes(xintercept = boostrap_estimated_mean_corrected), color = "orange") +
  labs(title = "Histrogram of Prices",
       y = "Density",
  x = "Prices", color = "Legend") +
  theme_minimal()


print(confidence_interval$bca[5] - confidence_interval$bca[4])


(boostrap_estimated_mean_corrected - confidence_interval$bca[4]) / (confidence_interval$bca[5] - confid
```

```r
ggplot(prices) +
  geom_histogram(aes(x = prices$Price),
                 color = "#000000", fill = "#dedede") +
  annotate("rect", xmin=confidence_interval$bca[4], xmax=confidence_interval$bca[5], ymin=0, ymax=Inf, a
  geom_vline(aes(xintercept = boostrap_estimated_mean_corrected), color = "orange") +
  labs(title = "Histrogram of Prices",
       y = "Density",
  x = "Prices", color = "Legend") +
  theme_minimal()


print(confidence_interval$normal[3] - confidence_interval$normal[2])


(boostrap_estimated_mean_corrected - confidence_interval$normal[2]) / (confidence_interval$normal[3] - c


ggplot(prices) +
  geom_histogram(aes(x = prices$Price),
                 color = "#000000", fill = "#dedede") +
  annotate("rect", xmin=confidence_interval$normal[2], xmax=confidence_interval$normal[3], ymin=0, ymax=
  geom_vline(aes(xintercept = boostrap_estimated_mean_corrected), color = "orange") +
  labs(title = "Histrogram of Prices",
       y = "Density",
  x = "Prices", color = "Legend") +
  theme_minimal()
```