Computational Statistics Lab5 Report by Group6

Jaskirat S Marar (jasma356) Filip Berndtsson (filbe354)
Dinuke Jayaweera (dinja628) Raja Uzair Saeed (rajsa233)

12/3/2021

Statement of Contribution

This lab work was divided among group members as follows:

1. Assignment1: Dinuke Jayaweera, Filip Berndtsson
2. Assignment2: Jaskirat Marar, Uzair Saeed

Assignment 1

Compute an estimate $\hat{Y}$ of the expected response as a function of X by using a loess smoother(use loess()). Estimate the distribution of T by using nonparametric bootstrap with B= 2000 (construct histogram).

First create a model with loess(y~x,data)

Then use the fitted values from the model as the predicted new $\hat{Y}$.

Using the argmin and argmax expressions we get X_a and X_b. We assemble the parameters for the test statistic T and calculate it. This will be put into the vector which will contain all 2000 T values.

Use test statistic:

$$T = \frac{\hat{Y}(X_b) - \hat{Y}(X_a)}{X_b - X_a} \; where;$$
$$X_b = arg\ max_X \hat{Y}(X)$$
$$X_a = arg\ min_X \hat{Y}(X)$$

```
library("boot")
lottery <- read.csv("lottery.csv", sep=";")

T_func <- function(data,idx){
  lottery1 <- data[idx,]
   #using loess to get a model
  loess_Y_model <- loess(Draft_No ~ Day_of_year, lottery1) #(Y ~ X, data)
  #using the model to get the new Y value
  Y_estimate <- loess_Y_model$fitted

  #Implementing the argmin and argmax expressions
  Xa <- lottery1$Day_of_year[which.min(Y_estimate)]
  Xb <- lottery1$Day_of_year[which.max(Y_estimate)]

  Ya <- Y_estimate[which.min(Y_estimate)]
  Yb <- Y_estimate[which.max(Y_estimate)]
  #the expression for the given test statistic
  T <- (Yb-Ya)/(Xb-Xa)
  return(T)
}

b <- boot(lottery, statistic=T_func, R=2000)
```

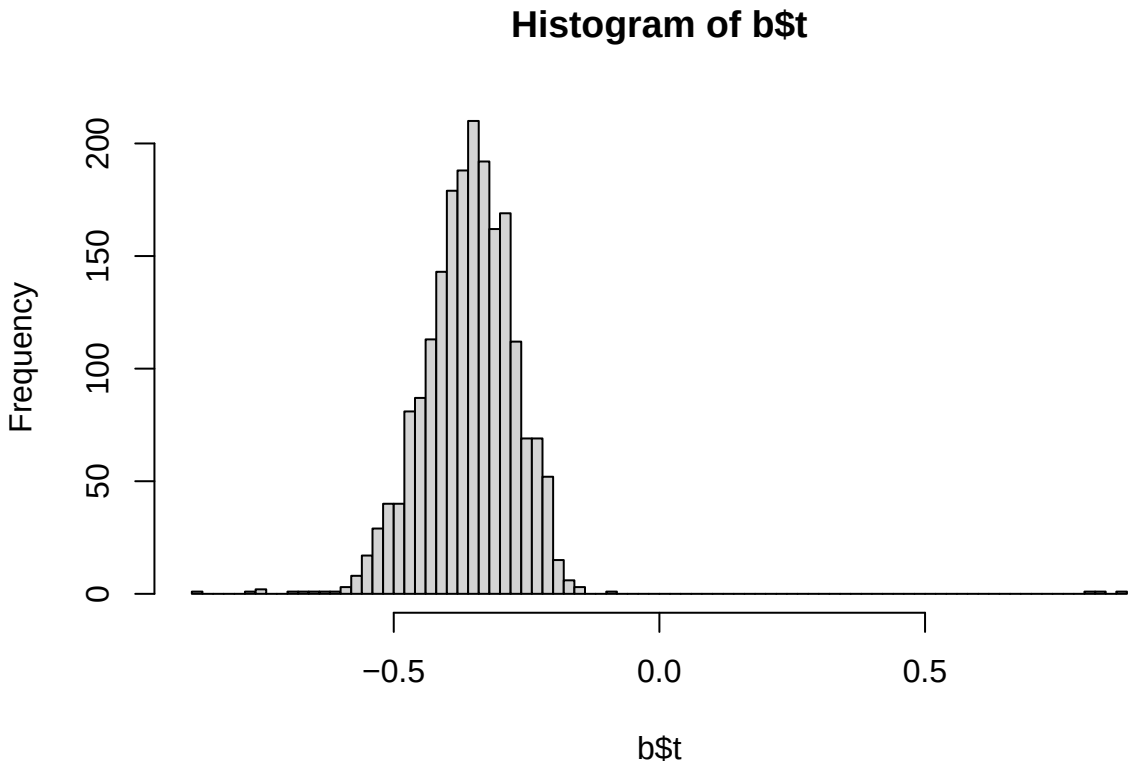
Using the obtained distribution (histogram)conclude whether the lottery is random or not.

From the vector of 2000 iterations of the test statistic produced by the boot function we can plot the test statistic values as a histogram to get a clear image.

```
hist(b$t,breaks=100, title = "Boot")
```

```
## Warning in plot.window(xlim, ylim, "", ...): "title" is not a graphical
## parameter
```

```
## Warning in title(main = main, sub = sub, xlab = xlab, ylab = ylab, ...): "title"
## is not a graphical parameter
## Warning in axis(1, ...): "title" is not a graphical parameter
## Warning in axis(2, ...): "title" is not a graphical parameter
```



The histogram shows us how test statistic is distributed, from this we can see that there is a region of higher concentration between -1 and 0 and not directly on 0. We can clearly see from the histogram that the bootstrapped T statistic is not very close to 0 at all and thus it is not random.

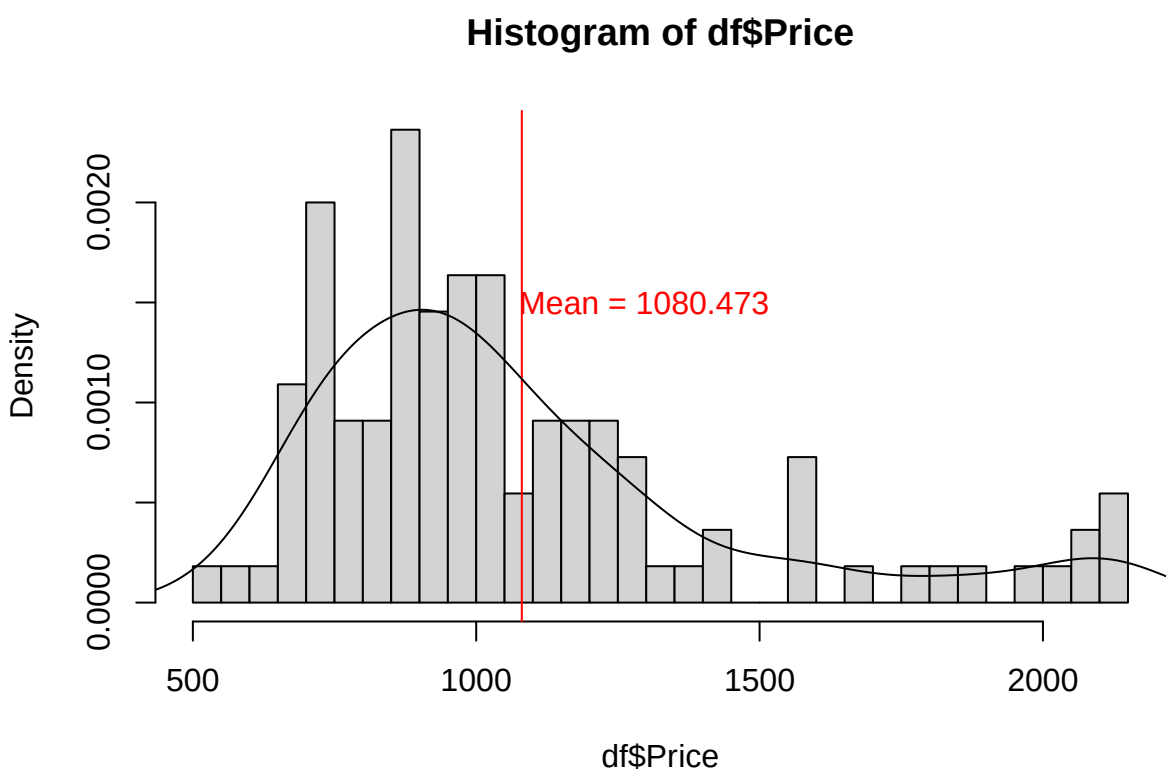
Assignment 2

We first read the data and plot a histogram to see what conventional distribution is resembles

```
setwd("D:/Documents/LiU Final/Autumn 21/732A90 - Computational Stats/Labs/Lab5")
df <- read.csv2("prices1.csv")
head(df, 5)
```

```
##   Price SqFt FEATS Taxes
## 1  2050 2650     7  1639
## 2  2080 2600     4  1088
## 3  2150 2664     5  1193
## 4  2150 2921     6  1635
## 5  1999 2580     4  1732
```

```
#2.1
#plotting histogram
hist(df$Price, probability = TRUE, breaks = 40)
obs_mean <- round(mean(df$Price), digits = 3)
abline(v = obs_mean, col = "red")
text(x = obs_mean * 1.2, y = 0.0015, paste("Mean =", obs_mean), col = "red", cex = 1)
lines(density(df$Price))
```




```
cat("\n Mean of Prices: ", obs_mean)
```

```
##
##  Mean of Prices:  1080.473
```

From the density plot, the data seems to resemble a gamma distribution

Bootstrap

We will now estimate the mean prices using non-parametric bootstrap. For this we will make use of the "boot" package and use the inbuilt functions boot() & boot.ci()

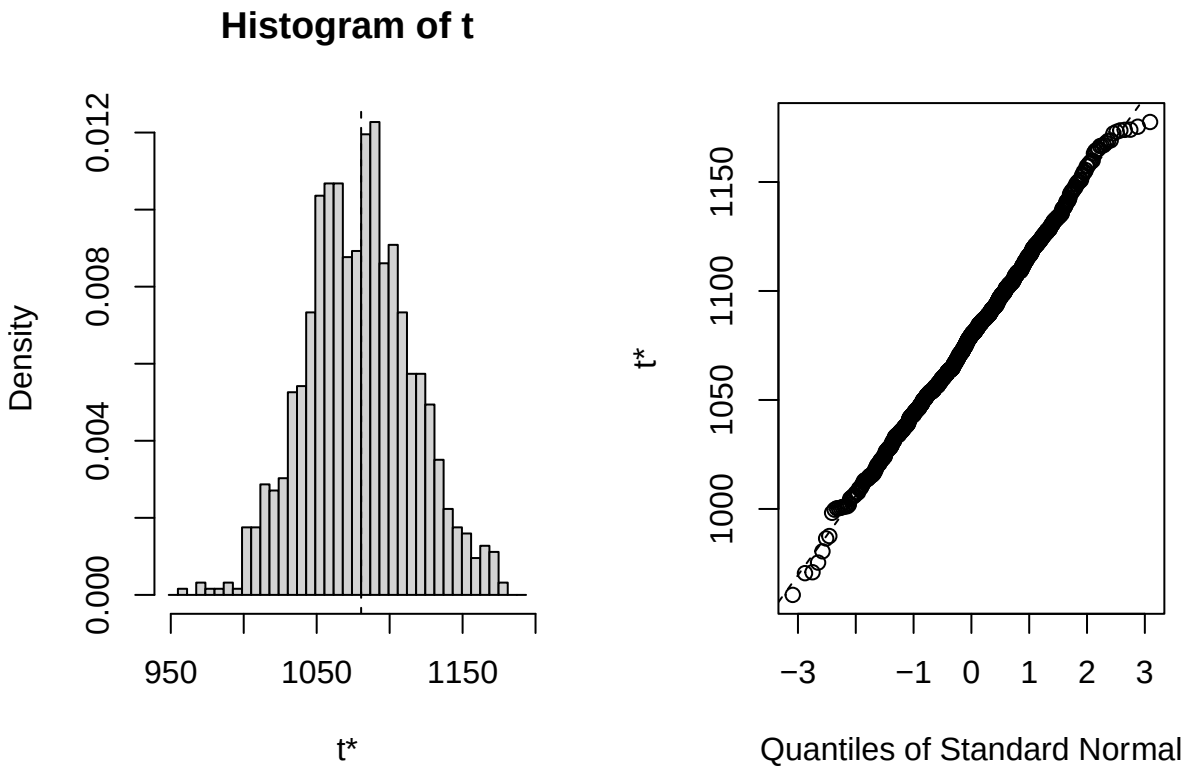
First we will write a function that takes the data and the indices of chosen elements as its input. This function will calculate the sample means for the chosen elements

```
#2.2
#bootstrap function
fn<- function(data, indices)
{
  res <- mean(data[indices])
  return(res)
}
```

We now use this function as an input to the boot() along with the number of repetitions or re-sampling iterations.

```
set.seed(1234)
boot_res <- boot(df$Price, fn, 999) #resampling with 999 replication

plot(boot_res, breaks = 40) #plotting bootstrap distribution
```



The bootstrap realizations seem to normally distributed.

Even though the boot() gives us the estimate of bias, The bootstrap estimate of bias can also be calculated analytically (from lecture slides) as follows:

$$T_1 := 2 * T(D) - \frac{1}{B}\sum_{i=1}^{B} T_i^*$$

```
bias_corrected <- 2*mean(df$Price) - 1/length(boot_res$t)*sum(boot_res$t)

cat("\n Bias corrected estimate (analytical): ", bias_corrected)
```

```
##
##  Bias corrected estimate (analytical):  1081.752
```

The variance of the mean price is calculated using the following expression (from lecture slides):

$$\hat{Var[T(.)]} = \frac{1}{B-1}\sum_{i=1}^{B}\left(T(D_i^*) - T(\bar{D^*})\right)^2$$

```
boot_var <- 1/(length(boot_res$t) - 1) * sum((boot_res$t - mean(boot_res$t))^2)

cat("\n Variance = ", boot_var)
```

```
##
##  Variance =  1334.235
```

Now we will be constructing the 95% CI for the mean price using the percentile, BCa and first-order normal approximation. This will be achieved using the boot.ci()

```
intervals <- boot.ci(boot_res, conf =0.95, type = c("perc", "bca", "norm"), index = 1)
intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_res, conf = 0.95, type = c("perc", "bca",
##     "norm"), index = 1)
##
## Intervals :
## Level      Normal             Percentile            BCa
## 95%   (1010, 1153 )   (1008, 1155 )   (1013, 1160 )
## Calculations and Intervals on Original Scale
```

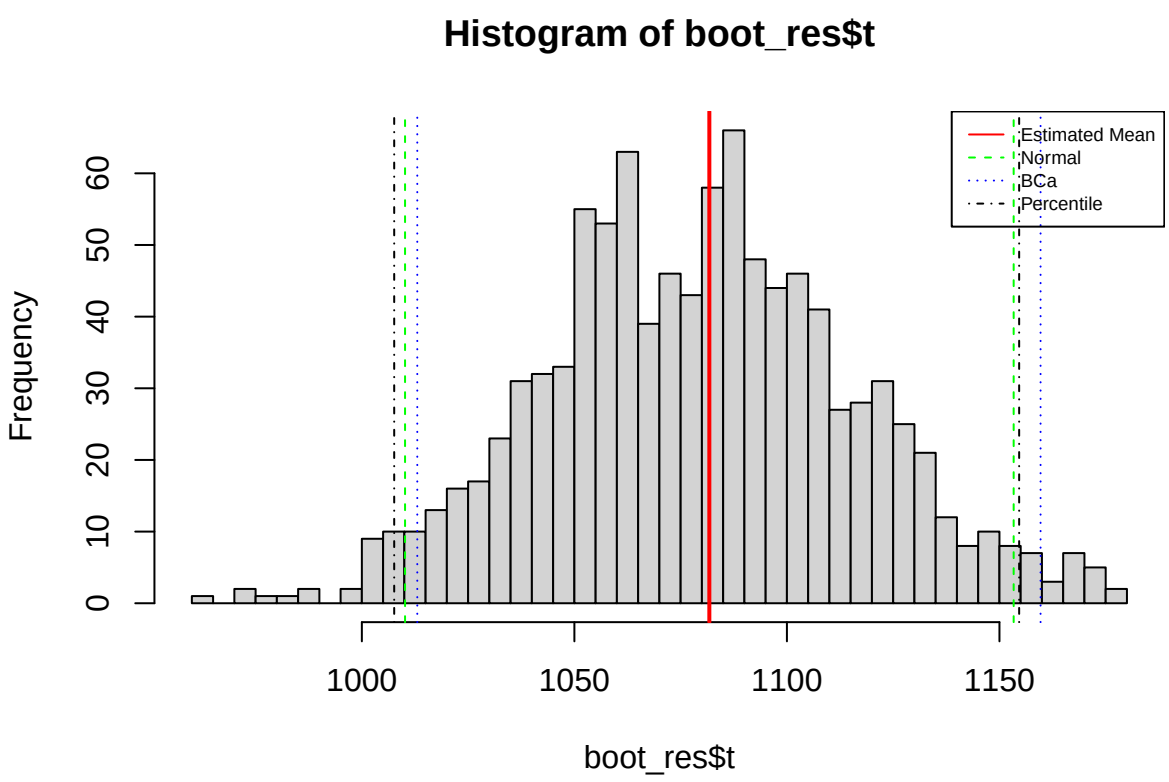
We will plot these together on a plot to get visualize the result of the CI construction

```
#2.3

hist(boot_res$t, breaks = 40)
abline(v = bias_corrected, col = "red",lwd = 2 )
abline(v = intervals$normal[2], col = "green",lwd = 1, lty=2)
abline(v = intervals$normal[3], col = "green",lwd = 1, lty=2 )
abline(v = intervals$bca[4], col = "blue",lwd = 1, lty=3)

abline(v = intervals$bca[5], col = "blue",lwd = 1, lty=3 )
abline(v = intervals$percent[4], col = "black",lwd = 1, lty=4)
abline(v = intervals$percent[5], col = "black",lwd = 1, lty=4)

legend( "topright", legend=c("Estimated Mean", "Normal", "BCa", "Percentile"),
        col=c("red", "green", "blue", "black"), lty = c(1, 2, 3, 4), cex=0.6)
```



The estimated mean lies within the confidence intervals for each of the methods

Jackknife

We specify a jackknife sample vector of n-1 sample points. We will also specify a vector of n values which will function as the vector for the pseudo values. We design the algorithm with the following logic:

1. For j<i, the $j^{\{th\}}$ element of jackknife sample = $j^{\{th\}}$ element of original data
2. for j=i, the value must be excluded in the sampling
3. for j>i, the $j\text{-}1^{\{th\}}$ element of the sample = $j^{\{th\}}$ element of original data

The code for the above is as follows:

```
#2.4

n = length(df$Price)
jack_vec <- numeric(n - 1)
jack_pseudo <- numeric(n)

for (i in 1:n) {
  for (j in 1:n) {
    if (j < i) {
      jack_vec[j] <- df$Price[j]
    } else if (j > i) {
      jack_vec[j-1] <- df$Price[j]
    }
  }
  jack_pseudo[i] <- n * mean(df$Price) - (n - 1) * mean(df$Price[-i])
```

```
}
```

In the sampling, we are removing the i^{th} value which means that by rearranging mean of data with X_i removed, each X_i can be written as:

$$X_i = n\bar{X} - (n-1)\bar{X}_i$$

we can replace the sample means with the estimators for pseudo values of X_i

$$Pseudo(X_i) = n\hat{\theta} - (n-1)\hat{\theta}_i$$

Each Pseudo value is an estimate of θ, hence the expected value of pseudo values will be

$$= \frac{1}{n}\sum_{i=1}^{n}\left(n\hat{\theta} - (n-1)\hat{\theta}_i\right) = n\hat{\theta} - (n-1)\hat{\theta}_{(.)} = Bias\ corrected\ Jackknife\ estimate$$

Which suggests

$$mean(Pseudo\ Values) = \hat{\theta}_{jack}\ ;$$

$$\hat{var}(\hat{\theta}_{jack}) = \frac{s^2_{jack}}{n}$$

Thus, The mean using Jackknife is calculated as follows:

```
mean(jack_pseudo)
```

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

The variance is calculated as follows:

```
var(jack_pseudo)/n
```

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

The variance and mean are both very close to what we saw in the bootstrap method.