# Lab 5 - Computational Statistics (732A90)

*Julius Kittler (julki092), Vinay Bengaluru Ashwath Narayan Murthy (vinbe289)*

*February 27, 2019*

## Contents



# 1 Assignment 1: Hypothesis testing

For this assignment, we are given a data set with 366 observations, one for each day of the year.

*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. (E.g. if the first draw issued the date 12.03., then all eligible men born on this date would get drafted first).*

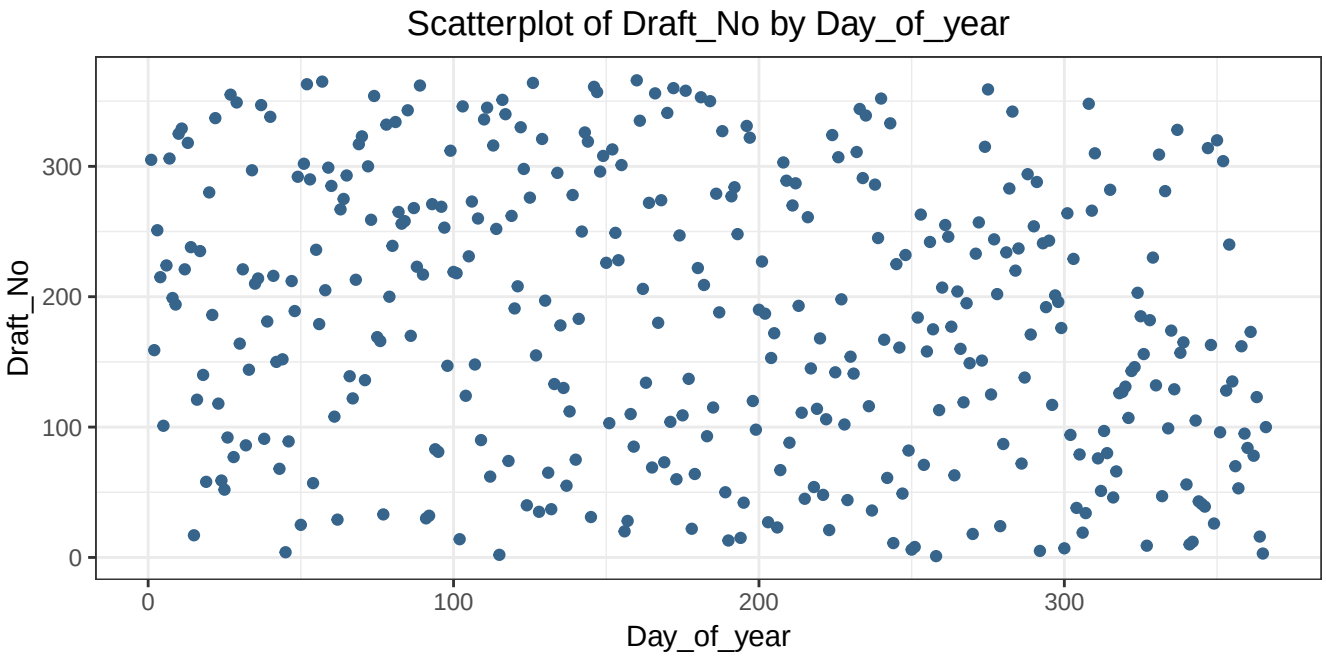
## 1.1 Task 1: Scatterplot of Y by X

First, we make a scatterplot of Y (`Draft_No`) by X (`Day_of_year`) and conclude whether the lottery looks random.

**Randomness of lottery based on plot**

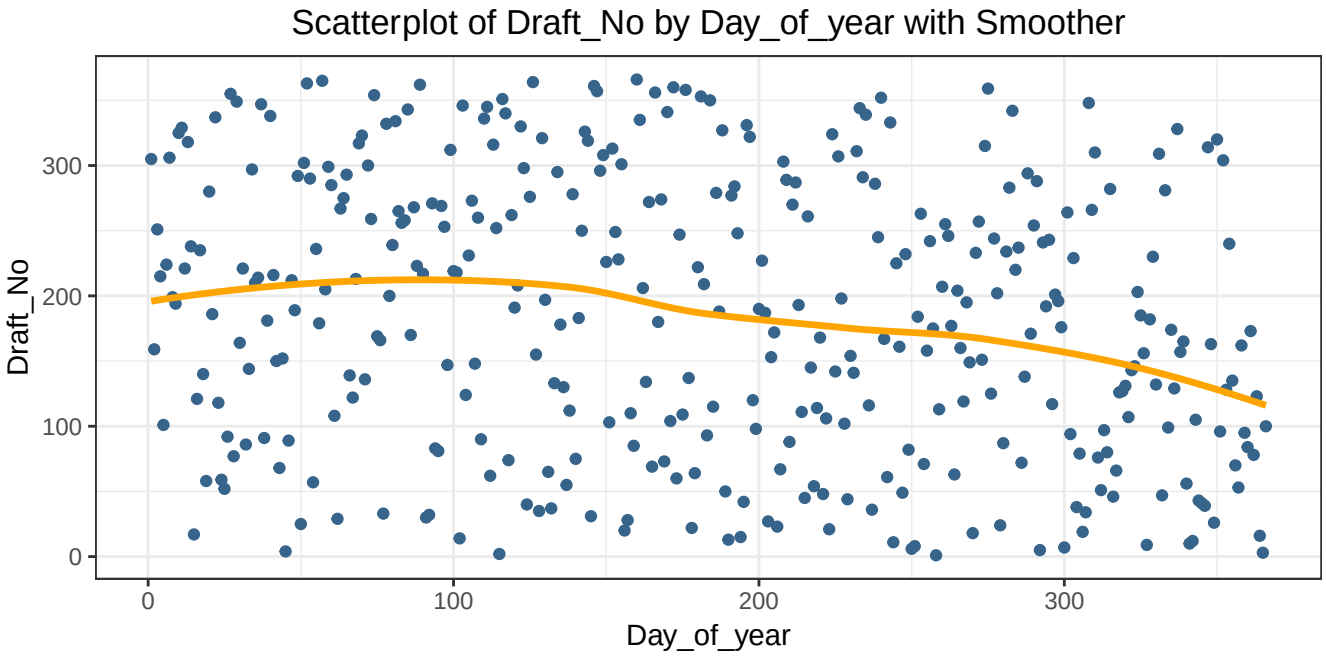
Based on the scatterplot below, the lottery not look random. The points are distributed quite equally across the plot and it does not seem like there is any trend. However, if we look closely, it does seem like the density of the data points in the bottom right is a bit higher then e.g. in the

top right. This is something we might want to investigate because for a truly random lottery we would expect the density in the top left and top right to be rather equal.




## 1.2 Task 2: Smoother for estimating Y_hat (with loess())

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



**Randomness of lottery based on plot**

We can see that $\hat{Y}$, the result of the smoother is decreasing with increasing day index (roughly from the middle of the year onwards). This reflects the observation from the previous task that the density of the data points is slightly higher in the bottom right than in the top right. The smoother supports the suspicion that the lottery was not entirely random.

## 1.3 Task 3: Test of randomness - with non-parametric Bootstrap

Now, we use the following test statistic, where $X_b = \text{argmax}_X\, Y(X), X_a = \text{argmin}_X\, Y(X)$

$$T = \frac{\hat{Y}(X_b) - \hat{Y}(X_a)}{X_b - X_a}$$

If this value is significantly greater than zero, then there should be a trend in the data and the lottery is not random. We estimate the distribution of T by using a non–parametric bootstrap with B = 2000 and comment whether the lottery is random or not. The T statistic for the overall data and a density plot can be found below.

**Implementation of Test with Bootstrap**

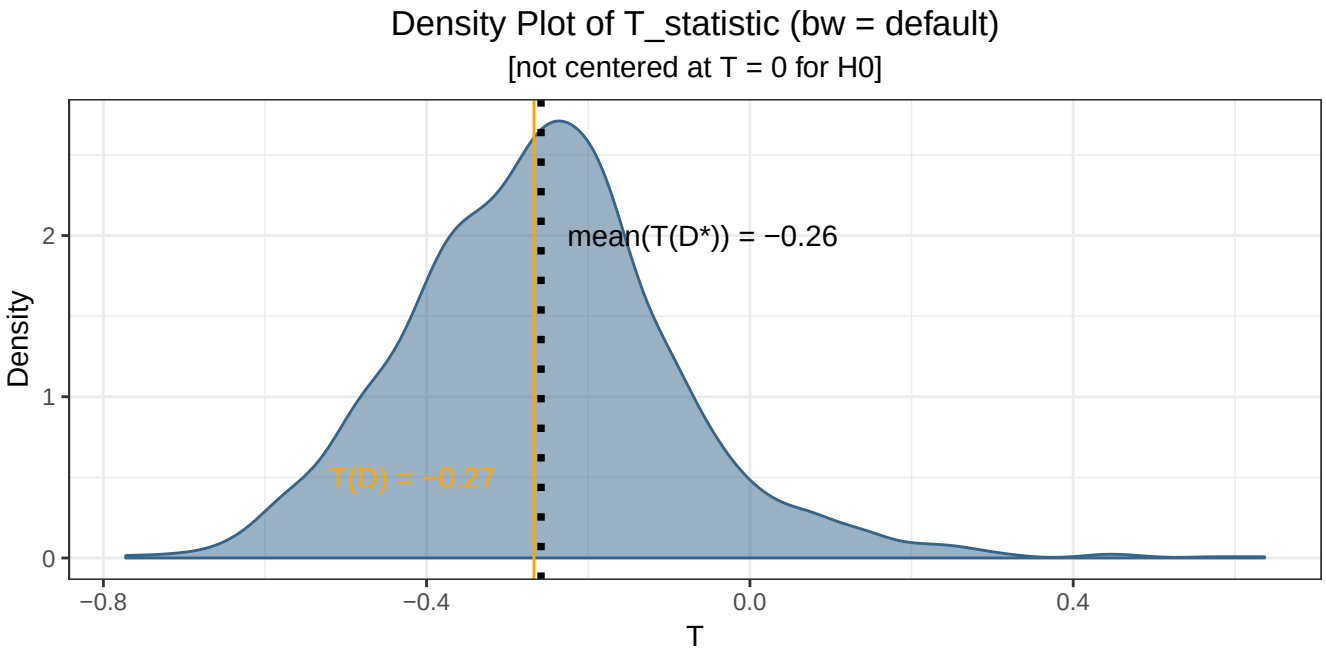
```
# Function to compute T_statistic -------------------------------------------
T_statistic = function(X, Y, Y_hat){

  idx_max = which.max(Y)
  idx_min = which.min(Y)

  num = Y_hat[idx_max] - Y_hat[idx_min]
  den = X[idx_max] - X[idx_min]

  T_statistic = num/den

}

bootstrap_test = function(X, Y, B = 2000){

  df = data.frame(X = X, Y = Y)
  N = nrow(df)
  T_statistics = numeric(B)

  for (b in 1:B){

    # Sample bootstrap data
    idx = sample(1:N, N, replace = TRUE)
    df_sub = df[idx, ]

    # Compute Y_hat
```

```
    df_sub$Y_hat = loess(Y ~ X, df_sub)$fitted

    # Compute T(D_b)
    T_statistics[b] = T_statistic(X = df_sub[, 1, drop = TRUE],
                                  Y = df_sub[, 2, drop = TRUE],
                                  Y_hat = df_sub[, 3, drop = TRUE])

  }

  # Compute T(D)
  df$Y_hat = loess(Y ~ X, df)$fitted
  T_stat = T_statistic(X = df$X, Y = df$Y, Y_hat = df$Y_hat)

  # Compute p-value
  T_statistics_centered = as.numeric(T_statistics - as.numeric(mean(T_statistics)))

  if (T_stat < 0){
    p_val = mean(T_statistics_centered < T_stat)
  } else {
    p_val = mean(T_statistics_centered > T_stat)
  }

  # Prepare output
  out = list(T_statistics = T_statistics, # bootstrapped, uncentered
             T_statistics_centered = T_statistics_centered, # bootstrapped
             T_stat = T_stat, # original
             p_val = p_val)
  return(out)

}
```

### Distribution (not centered at T = 0 for H0)

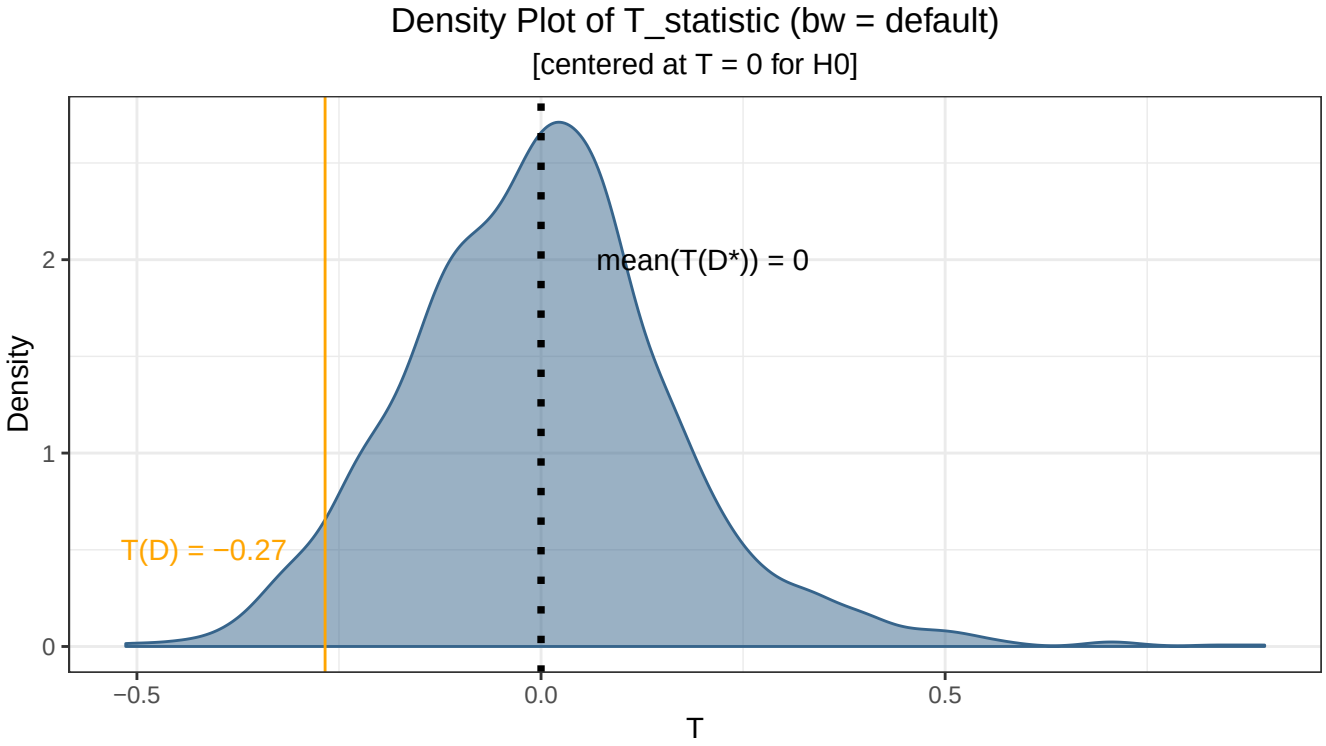
The mean (and the median) are below zero. The estimated true distribution is not symmetrical. It is slightly skewed to the right. Note that we assume the 2000 `T` values computed with non-parametric bootstrap to represent the true distribution of the population.

However, to compute the p-value and conduct the hypothesis test, we need the distribution under the null hypothesis. Given this distribution, we can compute the proportion of bootstrapped `T` values that are more extreme than the `T` value for the complete, original data.




**Distribution (centered at T = 0 for H0)**

Therefore, we center the distribution of the 2000 bootstrapped `T` values at `T = 0` since we have $H_0 : T = 0$. We do this by subtracting the mean of the 2000 bootstrapped `T` values. The result can be seen in the plot below. The p-value is computed as the percentage of points under this distribution that are smaller than the `T` of the complete original data (indicated by the orange line).




**Hypothesis**

The T value for the complete original data is printed below (and displayed as orange line in the

density plots). It is slightly below zero. According to the instructions, a value significantly greater than zero would indicate that the lottery is not random. In this case, we actually need to consider values significantly smaller than zero.

In our case, since we noticed a denser region in the bottom right than in the top right, we in fact only want to test if the T value for the original data is too small to belong to a distribution with T = 0. Therefore, we have a one-sided alternative hypothesis. We use the common $\alpha = 0.05$ to make our decision regarding rejecting or accepting hypotheses.

$$H_0 : T = 0 \,(\text{random})$$
$$H_a : T < 0 \,(\text{non-random})$$

**P-value of the test and decision**

Note that the p-value represents the probability of observing an estimate more extreme (larger or smaller) than the estimate that was obtained, given that the null hypothesis is true. In our case, we estimate the p-value by computing the percentage of bootstrapped `T` values (`T = 0` since we have $H_0 : T = 0$) that are smaller than our test statistic, the `T` value for all original data.

In this case, the probability of getting a `T` value smaller than the value obtained for the original data is 4.9%. Since we would require a p-value smaller than 5% (given $\alpha = 0.05$), we **successfully to reject the null hypothesis that the lottery is random**.

```
## T_statistic (original data):  -0.2671794
```

```
## p = P(T < T(D)):  0.0455
```

**Reference**

- https://stats.stackexchange.com/questions/92542/how-to-perform-a-bootstrap-test-to-compare-the-mean

## 1.4 Task 4: Test of randomness - with Permutation

Here, we implement a function depending on `data` and B that tests the same hypotheses from the previous task (again, with $\alpha = 0.05$). For this, permutation tests with statistics T are used. The function returns the p-value of the test. We apply the function on our data with B = 2000.

**Implementation of Permutation Test**

The implementation is similar to the previous task. However, instead of sampling a new data set in every iteration, we just shuffle the variable `Draft_No` in the original data set.

```
# Conduct permutation ------------------------------------------------------

permutation = function(X, Y, B = 2000){

  N = length(X)
  T_statistics = numeric(B)
  df = data.frame(X = X, Y = Y)
```

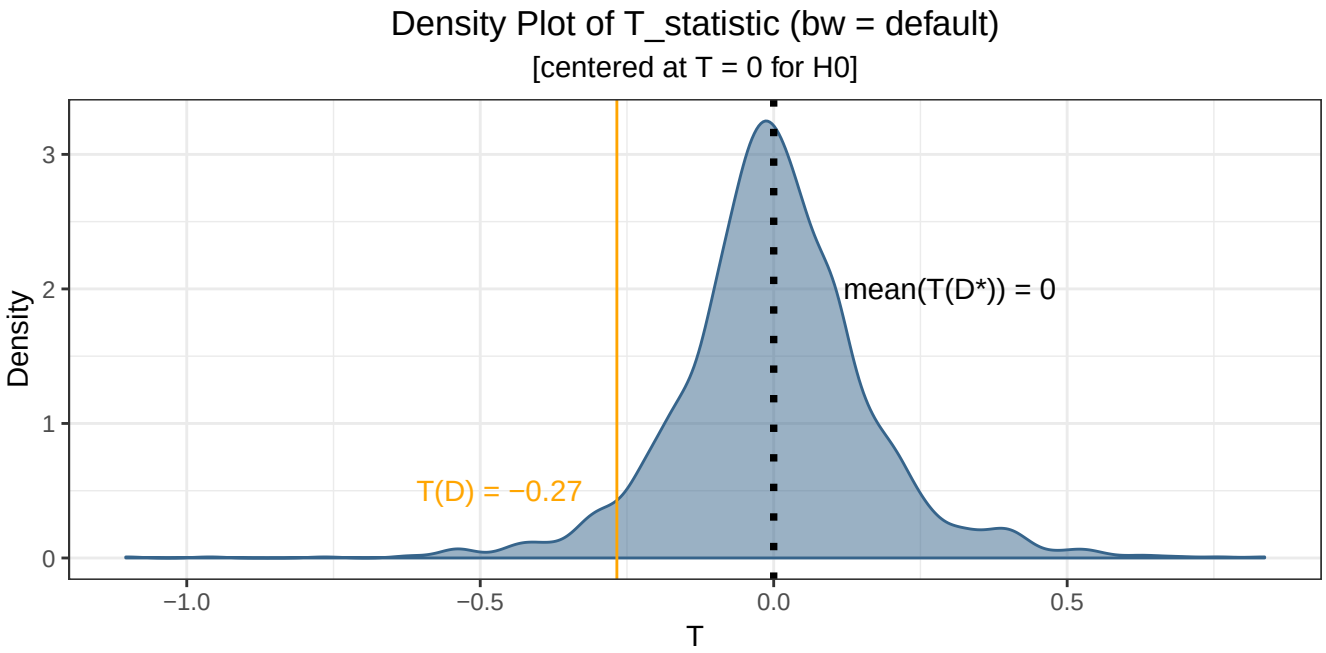
```r
  for (b in 1:B){

    # Shuffle (only) Y
    idx = sample(1:N, N)
    df_sub = data.frame(X = df$X, Y = df$Y[idx])

    # Compute Y_hat
    df_sub$Y_hat = loess(Y ~ X, df_sub)$fitted

    # Compute T(D_b)
    T_statistics[b] = T_statistic(X = df_sub[, 1, drop = TRUE],
                                  Y = df_sub[, 2, drop = TRUE],
                                  Y_hat = df_sub[, 3, drop = TRUE])

  }

  # Compute T(D)
  df$Y_hat = loess(Y ~ X, df)$fitted
  T_stat = T_statistic(X = df$X, Y = df$Y, Y_hat = df$Y_hat)

  # Compute p-value
  T_statistics_centered = as.numeric(T_statistics - as.numeric(mean(T_statistics)))

  if (T_stat < 0){
    p_val = mean(T_statistics_centered < T_stat)
  } else {
    p_val = mean(T_statistics_centered > T_stat)
  }

  # Prepare output
  out = list(T_statistics = T_statistics, # bootstrapped, uncentered
             T_statistics_centered = T_statistics_centered, # bootstrapped
             T_stat = T_stat, # original
             p_val = p_val)
  return(out)

}
```

### Distribution

Even though the distribution seems already centered at `T = 0`, it is not exactly. Therefore, the centering procedure is conducted again $H_0 : T = 0$.

```
## mean(T(D*)) before centering:  0.0006373915
```

```
## mean(T(D*)) after centering:  0.000000000000000001681098
```




**P-value of the test and decision**

In this case, the probability of getting a `T` value smaller than the value obtained for the original data is 4.8%. Since we require a p-value smaller than 5% (given $\alpha = 0.05$) for rejecting $H_0$, we again **successfully reject the null hypothesis that the lottery is random**.

```
## T_statistic (original data):  -0.2671794
```

```
## p = P(T < T(D)):  0.0475
```

## 1.5 Task 5: Estimate of power of statistical test

Here, we make an estimate of the power (= 1 - Type II error) of the permutation test (see task 4).

a) We 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) We plug this data into the permutation test with B = 200 and note whether it was rejected.
c) Repeat Steps 5a - 5b for $\alpha = 0.2, 0.3, ..., 10$.

**Process for estimating the power**

Note that whenever we fail to reject the $H_0$, we are making a type II error (failing to reject the $H_0$ even though the $H_a$ is correct). Since in our case, the data definitely comes from $H_a$, all hypotheses should get rejected. The power is estimated by taking the percentage of all rejections (i.e. the correct decisions).

The general process for estimating the power is:

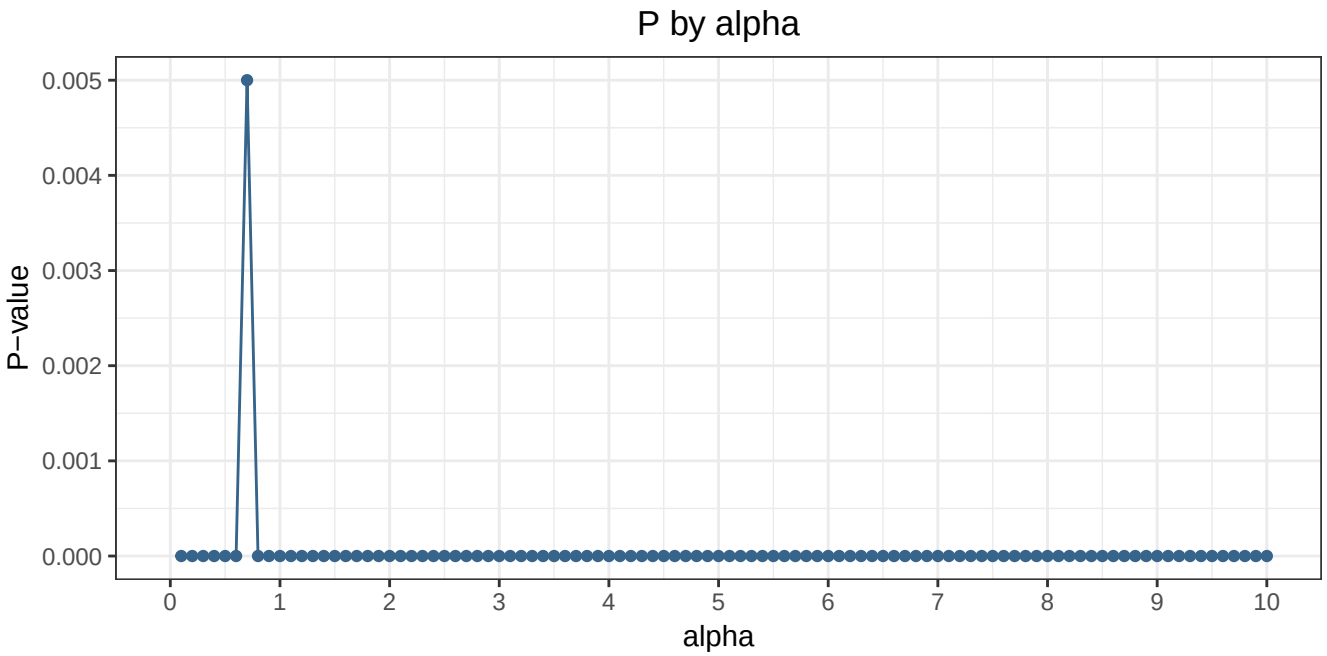
- Generate data samples that fulfill $H_a$
- Compute percentage of correct rejections (here: of all rejections)

The plot below shows that the H0 gets rejected for all $\alpha$ values since the p-values are (almost all) 0, which is smaller than $\alpha = 0.05$ for a one-sided and $\alpha = 0.025$ for a two-sided test.

```
# cat("Start timer...\n")
# ptm = proc.time()

# Initialize objects and parameters
beta = rnorm(1, mean = 183, sd = 10)
alphas = seq(0.1, 10, 0.1)
p = numeric(length(alphas))

for (i in 1:length(alphas)){

  # Create new data set
  Y = pmax(0, pmin(alphas[i] * df$Day_of_year + beta, 366))
  df_nonrandom = data.frame(X = df$Day_of_year, Y = Y)

  # Conduct permutation test and save p-value
  res = permutation(X = df_nonrandom$X, Y = df_nonrandom$Y, B = 200)
  T_statistics = res$T_statistics - mean(res$T_statistics)
  p[i] = res$p_val
  # p[i] = ifelse(res$p_val < 0.5, res$p_val, 1 - res$p_val)

}

cat("Estimate of the power: ", mean(p < 0.05), "\n")
```

```
## Estimate of the power:  1
```

```
# cat("Stop timer...\n")
# proc.time() - ptm
```




**Quality of your test statistics considering the value of the power**

The percentage of correctly rejected values, printed out above, is 100%. This means that the estimated power of the test is 1. This in turn means that in 100% of the cases we correctly reject the $H_0$ when the $H_a$ is indeed correct.

Considering that we have rejected the $H_0$ in task 4, we can however **not** say that this decision was correct with a probability of 100%. Instead, there is a 5% chance ($\alpha = 0.05$) that this decision was false ($\alpha$ is the probability of wrongly rejecting the $H_0$). The only thing we can say is that given that the $H_a$ is correct, the probability of the test rejecting the $H_0$ is 100%.

Note that $\alpha$ and $\beta$ (i.e. type I and type II error) are negatively related: decreasing $\alpha$ corresponds to increasing $\beta$ and increasing $\alpha$ corresponds to decreasing $\beta$. In our case, we have $\alpha = 0.05$, $\beta = 0$ (and power = 1 (= 1 - $\beta$)).

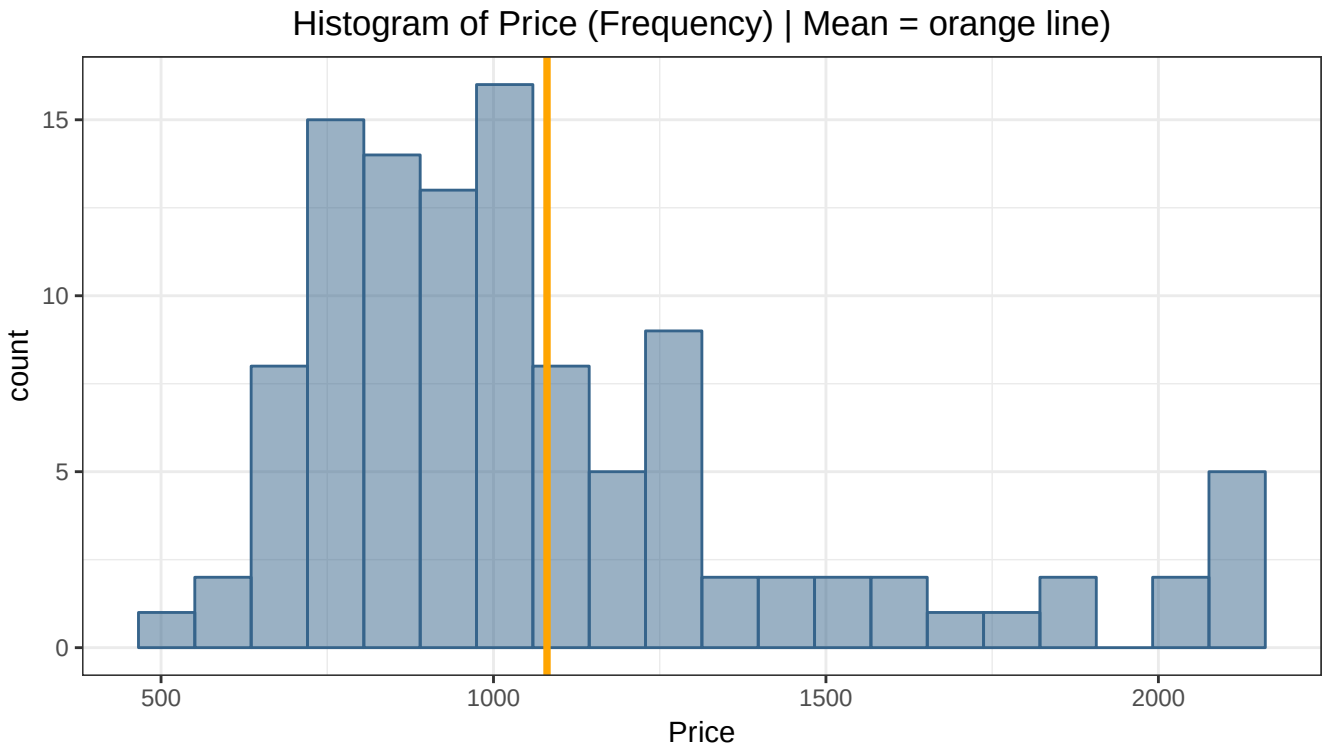
# 2 Assignment 2: Bootstrap, jackknife and CIs

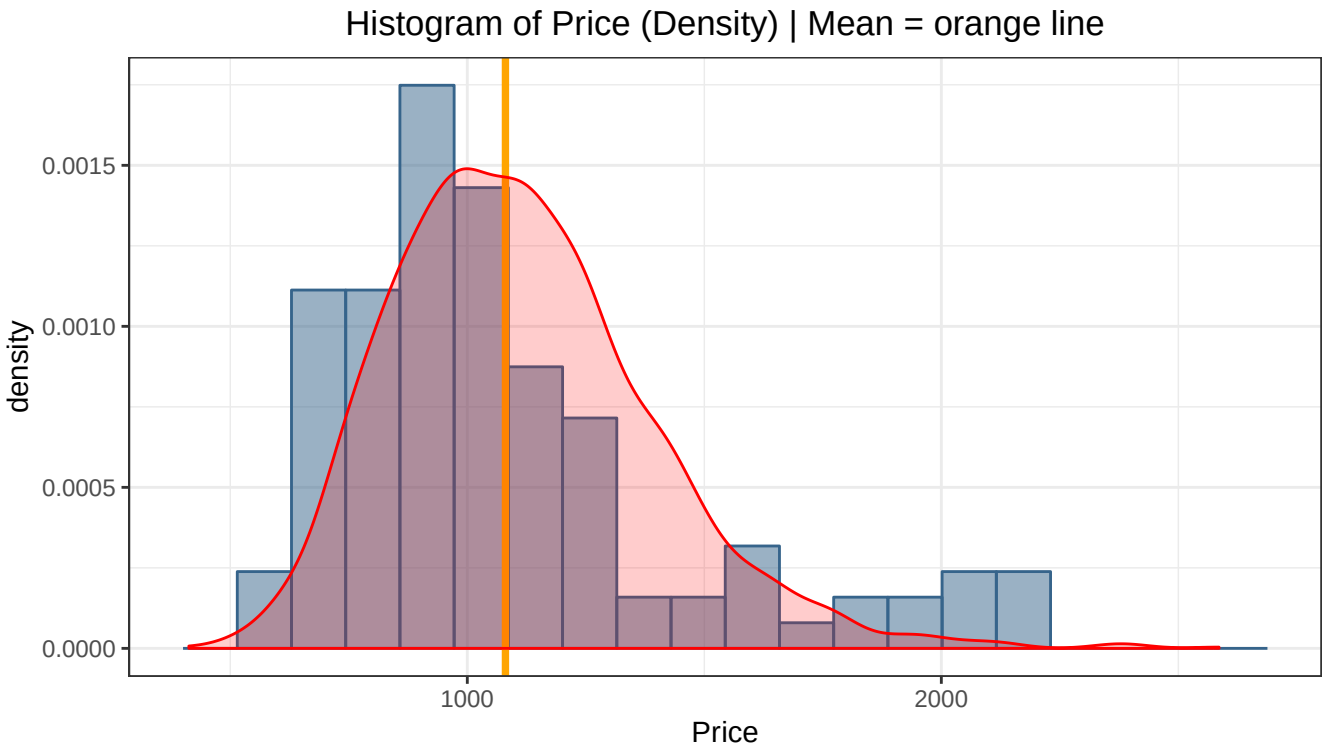
For this assignment, we are given a data set with 110 observations, where each observation represents an apartment or house in the US in 1993.

## 2.1 Task 1: Histogram and Mean of RV

First, we import the data and create a histogram of the variable `Price`. The mean price is printed out below and also displayed as a vertical orange line in the histogram.

```
## Mean price:  1080.473
```




### Distribution

The first histogram displays the frequency and the second histogram displays the density of the `Price`. A density plot of a log-normal distribution with meanlog `log(mean(df$Price)) = 6.985154` and `sdlog = 1/4` was added to the second histogram. It seems to fit the histogram quite well.

In conclusion, it seems like the histogram of `Price` resembles a log-normal distribution. Log-normal distributions are also used for modelling data such as revenues and stock prices (http://people.stern.nyu.edu/adamodar/New_Home_Page/StatFile/statdistns.htm).

## 2.2 Task 2: Estimation of Distribution (of Mean) with Bootstrap

Here, we estimate the distribution of the mean price of the house using non-parametric bootstrap. We determine the bootstrap bias–correction and the variance of the mean price. Then, we compute a 95% confidence interval for the mean price using bootstrap percentile, bootstrap BCa, and first–order normal approximation.

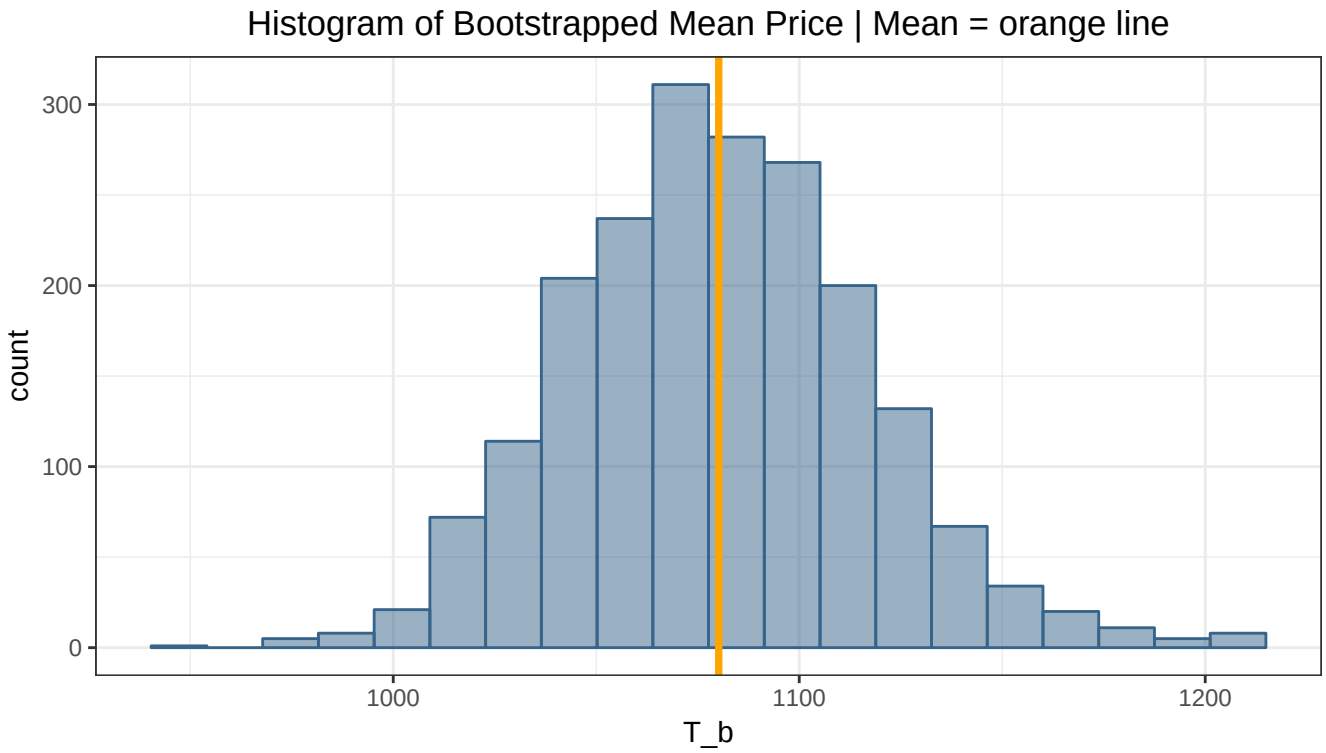
### Distribution of mean price

Below, we estimate the distribution of the mean price of the house using non-parametric bootstrap. Interestingly, the distribution of the mean price resembles a *Poisson distribution* and looks quite different than the distribution of the `Price` variable from the given data set (see previous task).

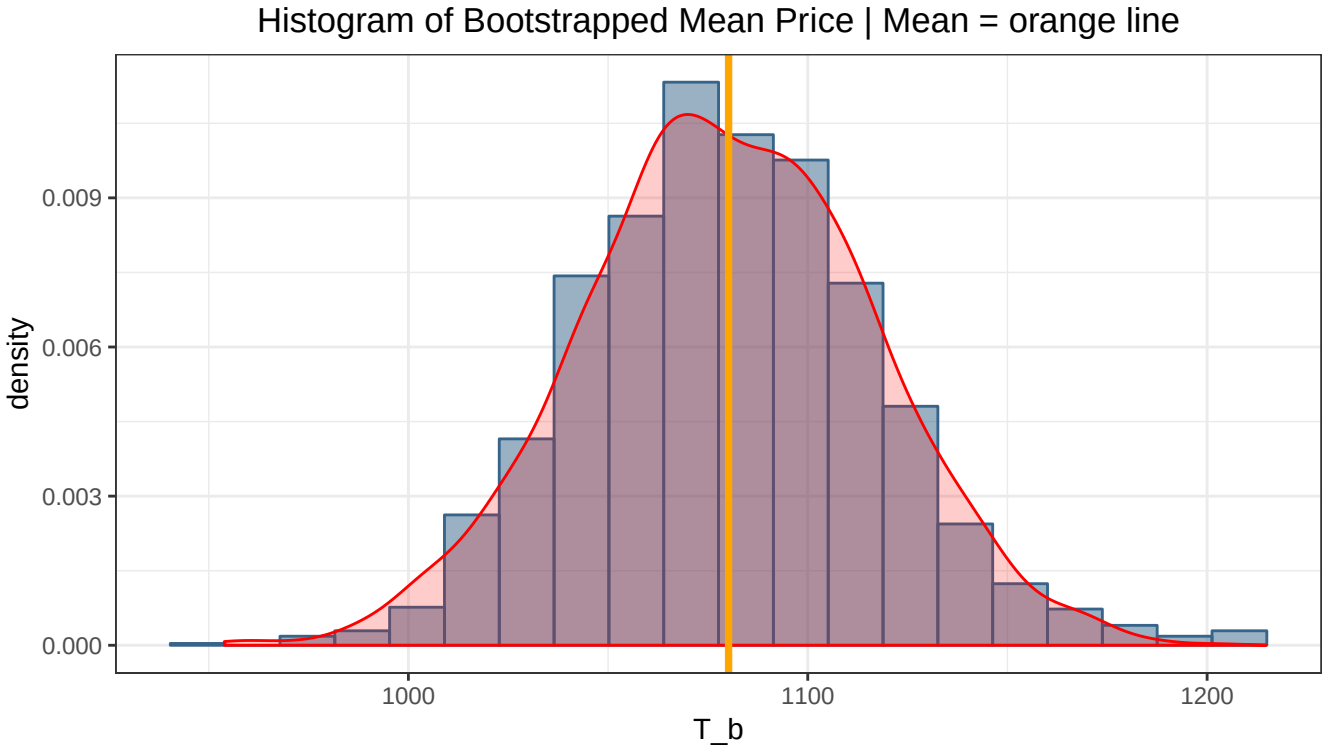
The average bootstrapped mean price is also printed out below. It is noticeably larger than the mean of the `Price` variable from the given data set, which was at ca 1080.

```
stat1 = function(data, n){

  data1 = data[n]
  res = mean(data1)

}

res = boot::boot(data = df$Price, statistic = stat1, R = 2000)

# res$t is a [B x 1] matrix, every every element is T(D_b)
# hist(res$t, 50)
```

```
## Average bootstrapped mean price:  1080.173
```




Again, a density plot was added, this time using a normal distribution with parameters `mean = mean(res$t) = 1080.57` and `sd = sd(res$t) = 36.85`. The normal distribution seems to resemble the histogram of the bootstrapped mean prices quite well.




### Variance of mean price

The variance is computed as sample variance of all the bootstrapped mean values.

$$\hat{V}(T) = \frac{1}{B-1}\sum_{b=1}^{B}(T^{*b} - \bar{T}^{*})^2, \text{where } T^{*b} = T(D_b)$$

```
B = length(res$t)
T_bar = mean(res$t)
variance = 1/(B - 1) * sum((res$t - T_bar)^2)
```

```
## Estimate of variance of mean price (own implementation):  1,347.808
```

```
## Estimate of variance of mean price (var(res$t) in R):  1,347.808
```

### Bootstrap bias-correction

The bias corrected estimator is computed as follows:

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

```
T_biasCorrected = 2 * mean(df$Price) - mean(res$t)
```

```
## Bias corrected estimator:  1080.772
```

### 95% CI

The three 95% confidence intervals are printed below (see `Percentile`, `BCa`, and `Normal` for first–order normal approximation).

```
res = boot::boot.ci(res)
res
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot::boot.ci(boot.out = res)
##
## Intervals :
## Level      Normal              Basic
## 95%   (1009, 1153 )   (1003, 1149 )
##
## Level     Percentile            BCa
## 95%   (1012, 1157 )   (1018, 1166 )
## Calculations and Intervals on Original Scale
```

## 2.3 Task 3: Estimation of Variance (of Mean) with jackknife

Here, we estimate the variance of the mean price using the jackknife and then compare it with the bootstrap variance estimate. Note: The question did not ask for a computation of another estimate of the mean price itself with the Jackknife method. Instead it only asked for a variance estimate with the jackknife method, which is shown below.

$$\hat{V} = \frac{1}{n \cdot (n-1)} \sum_{i=1}^{n} (T_i^* - J(T))^2$$

$$\text{where } T_i^* = n \cdot T(D) - (n-1) \cdot T(D_i^*), J(T) = \frac{1}{n} \sum_{i=1}^{n} T_i^*, n = B$$

Note that $D_i^* = X[-i]$ and that $T(D_i^*) = mean(D_i^*)$.

```
# Initialize variables ------------------------------------------------------

D = df$Price
N = length(D)
Ti = numeric(N)

# Compute Ti* ---------------------------------------------------------------

term1 = N * mean(D)

for (i in 1:N){
  term2 = (N - 1) * mean(D[-i])
  Ti[i] = term1 - term2
}

# Estimate mean -------------------------------------------------------------

mean_jackknife = mean(Ti)

# Estimate variance ---------------------------------------------------------

J = mean(Ti)
variance_jackknife = 1 / (N * (N - 1)) * sum((Ti - J)^2)
```

```
## Jackknife estimation of mean price:  1,080.473
```

```
## Jackknife variance estimation of mean price:  1,320.911
```

### Comparison with bootstrap estimate

Interestingly, the jackknife variance estimate is smaller than the bootstrap variance estimate (1,347.808). This was somewhat unexpected since the jackknife variance estimate has a tendency to give overestimates of the variance (see lecture slides).

## 2.4 Task 4: Comparison of Confidence Intervals

Here, we compare the confidence intervals obtained with respect to their length and the location of the estimated mean in these intervals (see task 2).

*As can be seen below, the bootstrap mean estimate from task 2 is slightly above the center of all three confidence intervals.* Note that we expect the mean to be near the center of the confidence intervals. This is because the function used to compute the confidence intervals `boot::boot.ci()` received the bootstrap result with the estimate of said mean as input and had the task to construct a CI around this mean.

*It seems like the center of the CI is lower than the boostrap estimate of the mean price because there are noticeably more observations with a smaller price than the mean than with a larger price than the mean (due to the right-skewed distribution, see plots from task 1).*

Table 1: CIs for bootstrapped mean price 1080.173

| name | mean_in_CI | min | max | CI_center | CI_length |
|---|---|---|---|---|---|
| Normal | TRUE | 1008.817 | 1152.728 | 1152.728 | 143.9103 |
| Percent | TRUE | 1012.211 | 1157.458 | 1157.458 | 145.2465 |
| BCa | TRUE | 1017.540 | 1166.287 | 1166.287 | 148.7468 |

# 3 Appendix

```
# Set up general options

knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE,
                      cache = TRUE, cache.path = "cache/", fig.path = "cache/")

# options(digits=22)
options(scipen=999)

library(dplyr)
library(ggplot2)
library(magrittr)

set.seed(12345)


# ------------------------------------------------------------------------------
# Assignment 1, Task 1
# ------------------------------------------------------------------------------


# Import data
```

```r
df = readxl::read_xls("lottery.xls")

# Plot Draft_No by Day_of_year
ggplot(df, aes(x = Day_of_year, y = Draft_No)) +
  geom_point(color = "steelblue4") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5)) +
  labs(title = "Scatterplot of Draft_No by Day_of_year")


# ------------------------------------------------------------------------------
# Assignment 1, Task 2
# ------------------------------------------------------------------------------


res = loess(Draft_No ~ Day_of_year, df)
df$Y_hat = res$fitted

# Plot Draft_No by Day_of_year
ggplot(df, aes(x = Day_of_year, y = Draft_No)) +
  geom_point(color = "steelblue4") +
  geom_line(aes(y = Y_hat), color = "orange", size = 1.2) +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5)) +
  labs(title = "Scatterplot of Draft_No by Day_of_year with Smoother")


# ------------------------------------------------------------------------------
# Assignment 1, Task 3
# ------------------------------------------------------------------------------


# Function to compute T_statistic ------------------------------------------
T_statistic = function(X, Y, Y_hat){

  idx_max = which.max(Y)
  idx_min = which.min(Y)

  num = Y_hat[idx_max] - Y_hat[idx_min]
  den = X[idx_max] - X[idx_min]

  T_statistic = num/den

}

bootstrap_test = function(X, Y, B = 2000){

  df = data.frame(X = X, Y = Y)
```

```r
  N = nrow(df)
  T_statistics = numeric(B)

  for (b in 1:B){

    # Sample bootstrap data
    idx = sample(1:N, N, replace = TRUE)
    df_sub = df[idx, ]

    # Compute Y_hat
    df_sub$Y_hat = loess(Y ~ X, df_sub)$fitted

    # Compute T(D_b)
    T_statistics[b] = T_statistic(X = df_sub[, 1, drop = TRUE],
                                  Y = df_sub[, 2, drop = TRUE],
                                  Y_hat = df_sub[, 3, drop = TRUE])

  }

  # Compute T(D)
  df$Y_hat = loess(Y ~ X, df)$fitted
  T_stat = T_statistic(X = df$X, Y = df$Y, Y_hat = df$Y_hat)

  # Compute p-value
  T_statistics_centered = as.numeric(T_statistics - as.numeric(mean(T_statistics)))

  if (T_stat < 0){
    p_val = mean(T_statistics_centered < T_stat)
  } else {
    p_val = mean(T_statistics_centered > T_stat)
  }

  # Prepare output
  out = list(T_statistics = T_statistics, # bootstrapped, uncentered
             T_statistics_centered = T_statistics_centered, # bootstrapped
             T_stat = T_stat, # original
             p_val = p_val)
  return(out)

}


res = bootstrap_test(X = df$Day_of_year, Y = df$Draft_No)
T_stat = res$T_stat
T_statistics = res$T_statistics
```

```r
# Density plot (not centered at T = 0 for H0) ---------------------------------

df_plot = data.frame(T_statistic = T_statistics)

ggplot(df_plot, aes(x = T_statistic)) +
  geom_density(fill = "steelblue4", color = "steelblue4", alpha = 0.5) +
  geom_vline(xintercept = T_stat, color = "orange") +
  geom_vline(xintercept = mean(T_statistics), color = "black",
             linetype = "dotted", size = 1.3) +
  annotate("text", label = paste0("T(D) = ", round(T_stat, 2)),
           size = 4, x = T_stat - 0.15, y = 0.5, color = "orange") +
  annotate("text", label = paste0("mean(T(D*)) = ", round(mean(T_statistics), 2)),
           size = 4, x = mean(T_statistics) + 0.2, y = 2, color = "black") +
  labs(title = "Density Plot of T_statistic (bw = default)",
       subtitle = "[not centered at T = 0 for H0]", y = "Density", x = "T") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5),
                     plot.subtitle = element_text(hjust = 0.5))


# Density plot (centered at T = 0 for H0) -------------------------------------

T_statistics = res$T_statistics_centered
df_plot = data.frame(T_statistic = T_statistics)

ggplot(df_plot, aes(x = T_statistic)) +
  geom_density(fill = "steelblue4", color = "steelblue4", alpha = 0.5) +
  geom_vline(xintercept = T_stat, color = "orange") +
  geom_vline(xintercept = mean(T_statistics),
             color = "black", linetype = "dotted", size = 1.3) +
  annotate("text", label = paste0("T(D) = ", round(T_stat, 2)),
           size = 4, x = T_stat - 0.15, y = 0.5, color = "orange") +
  annotate("text", label = paste0("mean(T(D*)) = ", round(mean(T_statistics), 2)),
           size = 4, x = mean(T_statistics) + 0.2, y = 2, color = "black") +
  labs(title = "Density Plot of T_statistic (bw = default)",
       subtitle = "[centered at T = 0 for H0]", y = "Density", x = "T") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5),
                     plot.subtitle = element_text(hjust = 0.5))



# T_statistic (original data) -------------------------------------------------

cat("T_statistic (original data): ", T_stat, "\n")

# P-val -----------------------------------------------------------------------
```

```r
cat("p = P(T < T(D)): ", res$p_val, "\n") # P-value here
# cat("p = P(T < T(D)): ", mean(T_statistics < T_stat), "\n") # same


# ------------------------------------------------------------------------------
# Assignment 1, Task 4
# ------------------------------------------------------------------------------


# Conduct permutation ----------------------------------------------------------

permutation = function(X, Y, B = 2000){

  N = length(X)
  T_statistics = numeric(B)
  df = data.frame(X = X, Y = Y)

  for (b in 1:B){

    # Shuffle (only) Y
    idx = sample(1:N, N)
    df_sub = data.frame(X = df$X, Y = df$Y[idx])

    # Compute Y_hat
    df_sub$Y_hat = loess(Y ~ X, df_sub)$fitted

    # Compute T(D_b)
    T_statistics[b] = T_statistic(X = df_sub[, 1, drop = TRUE],
                                  Y = df_sub[, 2, drop = TRUE],
                                  Y_hat = df_sub[, 3, drop = TRUE])

  }

  # Compute T(D)
  df$Y_hat = loess(Y ~ X, df)$fitted
  T_stat = T_statistic(X = df$X, Y = df$Y, Y_hat = df$Y_hat)

  # Compute p-value
  T_statistics_centered = as.numeric(T_statistics - as.numeric(mean(T_statistics)))

  if (T_stat < 0){
    p_val = mean(T_statistics_centered < T_stat)
  } else {
    p_val = mean(T_statistics_centered > T_stat)
  }
```

```r
  # Prepare output
  out = list(T_statistics = T_statistics, # bootstrapped, uncentered
             T_statistics_centered = T_statistics_centered, # bootstrapped
             T_stat = T_stat, # original
             p_val = p_val)
  return(out)

}



res = permutation(X = df$Day_of_year, Y = df$Draft_No)
T_stat = res$T_stat

cat("mean(T(D*)) before centering: ", mean(res$T_statistics),"\n")

T_statistics = res$T_statistics_centered
cat("mean(T(D*)) after centering: ", mean(T_statistics),"\n")

# Density plots ----------------------------------------------------------------

df_plot = data.frame(T_statistic = T_statistics)

ggplot(df_plot, aes(x = T_statistic)) +
  geom_density(fill = "steelblue4", color = "steelblue4", alpha = 0.5) +
  geom_vline(xintercept = T_stat, color = "orange") +
  geom_vline(xintercept = mean(T_statistics),
             color = "black", linetype = "dotted", size = 1.3) +
  annotate("text", label = paste0("T(D) = ", round(T_stat, 2)),
           size = 4, x = T_stat - 0.2, y = 0.5, color = "orange") +
  annotate("text", label = paste0("mean(T(D*)) = ", round(mean(T_statistics), 2)),
           size = 4, x = mean(T_statistics) + 0.3, y = 2, color = "black") +
  labs(title = "Density Plot of T_statistic (bw = default)", y = "Density",
       subtitle = "[centered at T = 0 for H0]", x = "T") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5),
                     plot.subtitle = element_text(hjust = 0.5))


# T_statistic (original data) --------------------------------------------------

cat("T_statistic (original data): ", T_stat, "\n")

# P-val ------------------------------------------------------------------------

cat("p = P(T < T(D)): ", res$p_val, "\n") # same
# cat("p = P(T < T(D)): ", mean(T_statistics < T_stat), "\n") # same
```

```r
# Example from lecture ------------------------------------------------------

mouse = data.frame(Group = c("y", "z", "z", "y", "y", "z", "y", "y", "y", "y",
                             "z", "z", "y", "z", "y", "z"),
                   Value = c(10, 16, 23, 27, 31, 38, 40, 46, 50, 52, 94, 99,
                             104, 141, 146, 197))

B=1000
stat=numeric(B)
n=dim(mouse)[1]

for(b in 1:B){

  Gb = sample(mouse$Group, n)
  stat[b] = mean(mouse$Value[Gb=='z']) - mean( (mouse$Value[Gb=='y']))

}

hist(stat,50)
stat0 = mean(mouse$Value[mouse$Group=='z']) - mean(mouse$Value[mouse$Group=='y'])
print(c(stat0, mean(stat>stat0))) # mean(stat>stat0) is the p value


# ---------------------------------------------------------------------------
# Assignment 1, Task 5
# ---------------------------------------------------------------------------


# cat("Start timer...\n")
# ptm = proc.time()

# Initialize objects and parameters
beta = rnorm(1, mean = 183, sd = 10)
alphas = seq(0.1, 10, 0.1)
p = numeric(length(alphas))

for (i in 1:length(alphas)){

  # Create new data set
  Y = pmax(0, pmin(alphas[i] * df$Day_of_year + beta, 366))
  df_nonrandom = data.frame(X = df$Day_of_year, Y = Y)

  # Conduct permutation test and save p-value
  res = permutation(X = df_nonrandom$X, Y = df_nonrandom$Y, B = 200)
  T_statistics = res$T_statistics - mean(res$T_statistics)
```

```r
  p[i] = res$p_val
  # p[i] = ifelse(res$p_val < 0.5, res$p_val, 1 - res$p_val)

}

cat("Estimate of the power: ", mean(p < 0.05), "\n")

# cat("Stop timer...\n")
# proc.time() - ptm


df_plot = data.frame(P = p, alpha = alphas)

ggplot(df_plot, aes(x = alpha, y = P)) +
  geom_point(color = "steelblue4") + geom_line(color = "steelblue4") +
  labs(title = "P by alpha", y = "P-value") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5)) +
  scale_x_continuous(limits = c(0, 10), breaks = seq(0, 10, 1))


# ------------------------------------------------------------------------
# Assignment 2, Task 1
# ------------------------------------------------------------------------

df = readxl::read_xls("prices1.xls") %>% as.data.frame()
cat("Mean price: ", mean(df$Price), "\n")

# Simple Histogram ------------------------------------------------------
ggplot(df, aes(x = Price)) +
  geom_histogram(fill = "steelblue4", color = "steelblue4",
                 alpha = 0.5, bins = 20) +
  geom_vline(xintercept =  mean(df$Price), color = "orange", size = 1.3) +
  labs(title = "Histogram of Price (Frequency) | Mean = orange line)") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


# Histogram with lognormal density --------------------------------------
df_data = data.frame(x = rlnorm(2000, meanlog = log(mean(df$Price)),
                                sdlog = 1/4))

ggplot(df, aes(x = Price)) +
  geom_histogram(aes(y = ..density..), fill = "steelblue4",
                 color = "steelblue4", alpha = 0.5, bins = 20) +
  geom_vline(xintercept =  mean(df$Price), color = "orange", size = 1.3) +
  geom_density(data = df_data, aes(x = x), color = "red",
               fill = "red", alpha = 0.2) +
```

```r
  labs(title = "Histogram of Price (Density) | Mean = orange line") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))



# ------------------------------------------------------------------------------
# Assignment 2, Task 2
# ------------------------------------------------------------------------------


stat1 = function(data, n){

  data1 = data[n]
  res = mean(data1)

}

res = boot::boot(data = df$Price, statistic = stat1, R = 2000)

# res$t is a [B x 1] matrix, every every element is T(D_b)
# hist(res$t, 50)


bootstrap_mean_price = mean(res$t)
cat("Average bootstrapped mean price: ", bootstrap_mean_price,"\n")
df_plot = data.frame(T_b = res$t)

ggplot(df_plot, aes(x = T_b)) +
  geom_histogram(fill = "steelblue4", color = "steelblue4",
                 alpha = 0.5, bins = 20) +
  geom_vline(xintercept =  mean(res$t), color = "orange", size = 1.3) +
  labs(title = "Histogram of Bootstrapped Mean Price | Mean = orange line") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))


df_data = data.frame(x = rnorm(2000, mean = mean(res$t), sd = sd(res$t)))

ggplot(df_plot, aes(x = T_b)) +
  geom_histogram(aes(y = ..density..), fill = "steelblue4", color = "steelblue4",
                 alpha = 0.5, bins = 20) +
  geom_density(data = df_data, aes(x = x), color = "red",
               fill = "red", alpha = 0.2) +
  geom_vline(xintercept = mean(res$t), color = "orange", size = 1.3) +
  labs(title = "Histogram of Bootstrapped Mean Price | Mean = orange line") +
  theme_bw() + theme(plot.title = element_text(hjust = 0.5))
```

```r
B = length(res$t)
T_bar = mean(res$t)
variance = 1/(B - 1) * sum((res$t - T_bar)^2)


cat("Estimate of variance of mean price (own implementation): ",
    format(variance, nsmall=0, big.mark=","), "\n")

cat("Estimate of variance of mean price (var(res$t) in R): ",
    format(var(res$t), nsmall=0, big.mark=","), "\n")


T_biasCorrected = 2 * mean(df$Price) - mean(res$t)


cat("Bias corrected estimator: ", T_biasCorrected,"\n")


res = boot::boot.ci(res)
res


# ------------------------------------------------------------------------------
# Assignment 2, Task 3
# ------------------------------------------------------------------------------


# Initialize variables ---------------------------------------------------------

D = df$Price
N = length(D)
Ti = numeric(N)

# Compute Ti* ------------------------------------------------------------------

term1 = N * mean(D)

for (i in 1:N){
  term2 = (N - 1) * mean(D[-i])
  Ti[i] = term1 - term2
}

# Estimate mean ----------------------------------------------------------------

mean_jackknife = mean(Ti)
```

```r
# Estimate variance ------------------------------------------------------------

J = mean(Ti)
variance_jackknife = 1 / (N * (N - 1)) * sum((Ti - J)^2)


cat("Jackknife estimation of mean price: ",
    format(mean_jackknife, nsmall=0, big.mark=","), "\n")

cat("Jackknife variance estimation of mean price: ",
    format(variance_jackknife, nsmall=0, big.mark=","), "\n")


# ------------------------------------------------------------------------------
# Assignment 2, Task 4
# ------------------------------------------------------------------------------

name = c("Normal", "Percent", "BCa")

mean_in_CI = rep(TRUE, 3)

min = c(res$normal[length(res$normal) - 1],
        res$percent[length(res$percent) - 1],
        res$bca[length(res$bca) - 1])

max = c(res$normal[length(res$normal)],
        res$percent[length(res$percent)],
        res$bca[length(res$bca)])

CI_center = c(mean(max[1], min[1]),
              mean(max[2], min[2]),
              mean(max[3], min[3]))

CI_length = c(max[1] - min[1],
              max[2] - min[2],
              max[3] - min[3])

df = data.frame(name = name,
                mean_in_CI = mean_in_CI,
                min = min,
                max = max,
                CI_center = CI_center,
                CI_length = CI_length)

knitr::kable(df, caption = paste0("CIs for bootstrapped mean price ",
                                  round(bootstrap_mean_price, 4)))
```