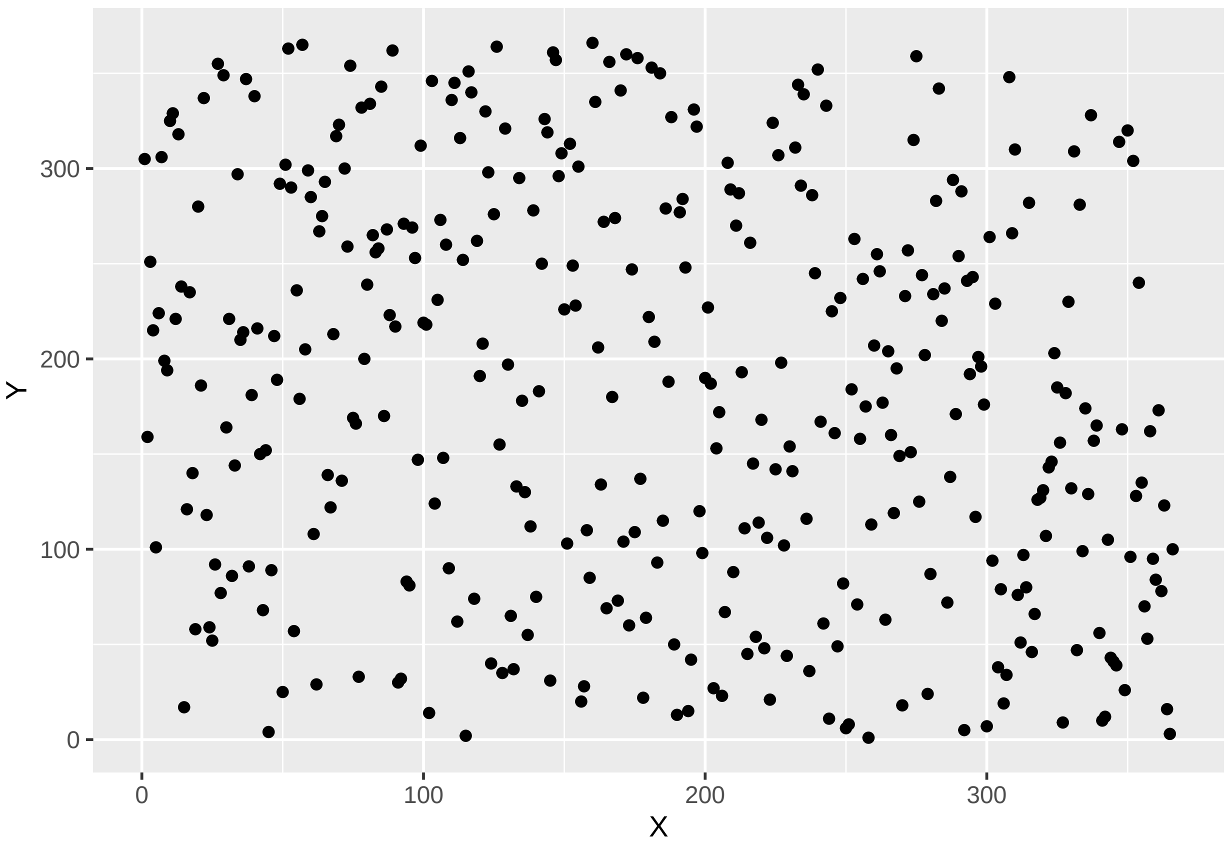
Computational Statistics - Lab 5

Nicolas Taba & Tim Yuki Washio

07/12/2020

Question 1: Hypothesis testing

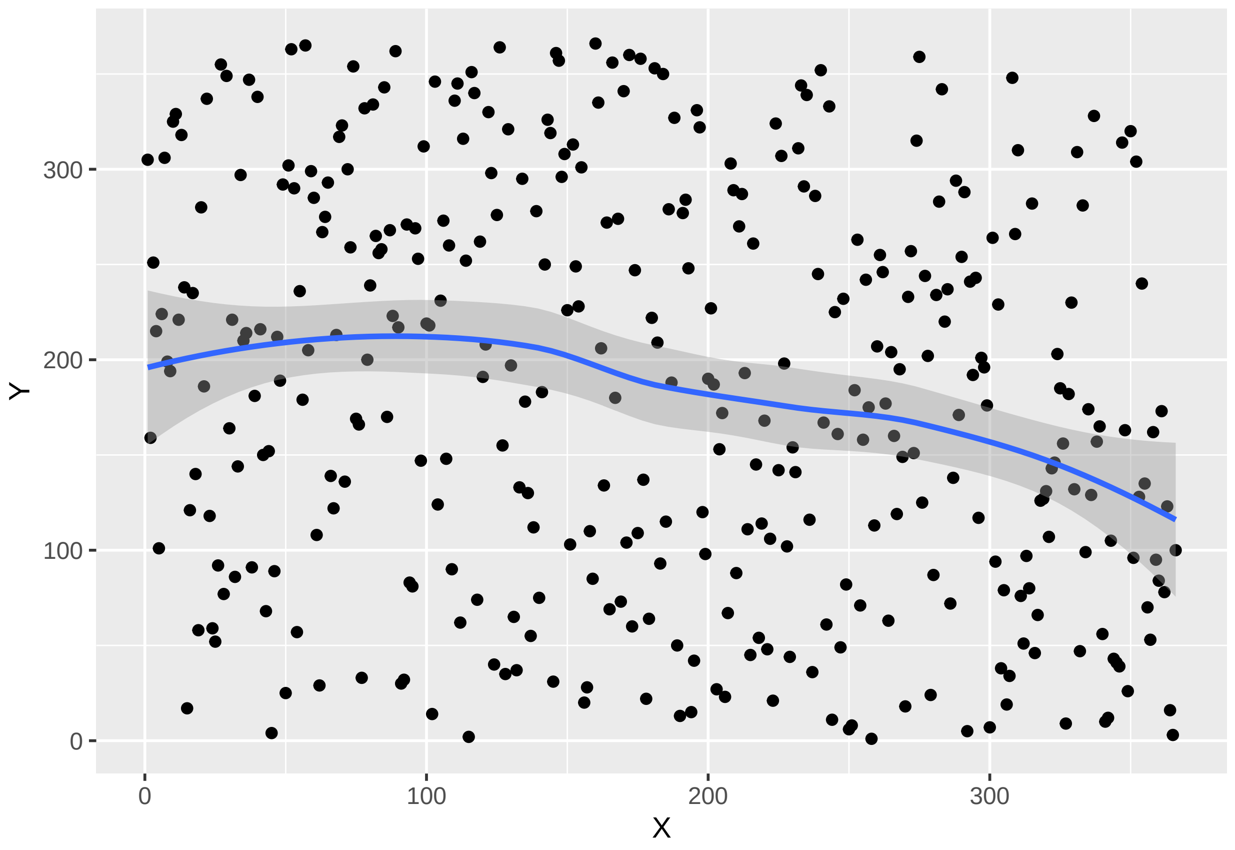
1 Investigate the data



The data covers the whole area and looks to be randomly distributed with no obvious relationship between X and Y.

2. Compute the estimate of Y using a loess smoother.

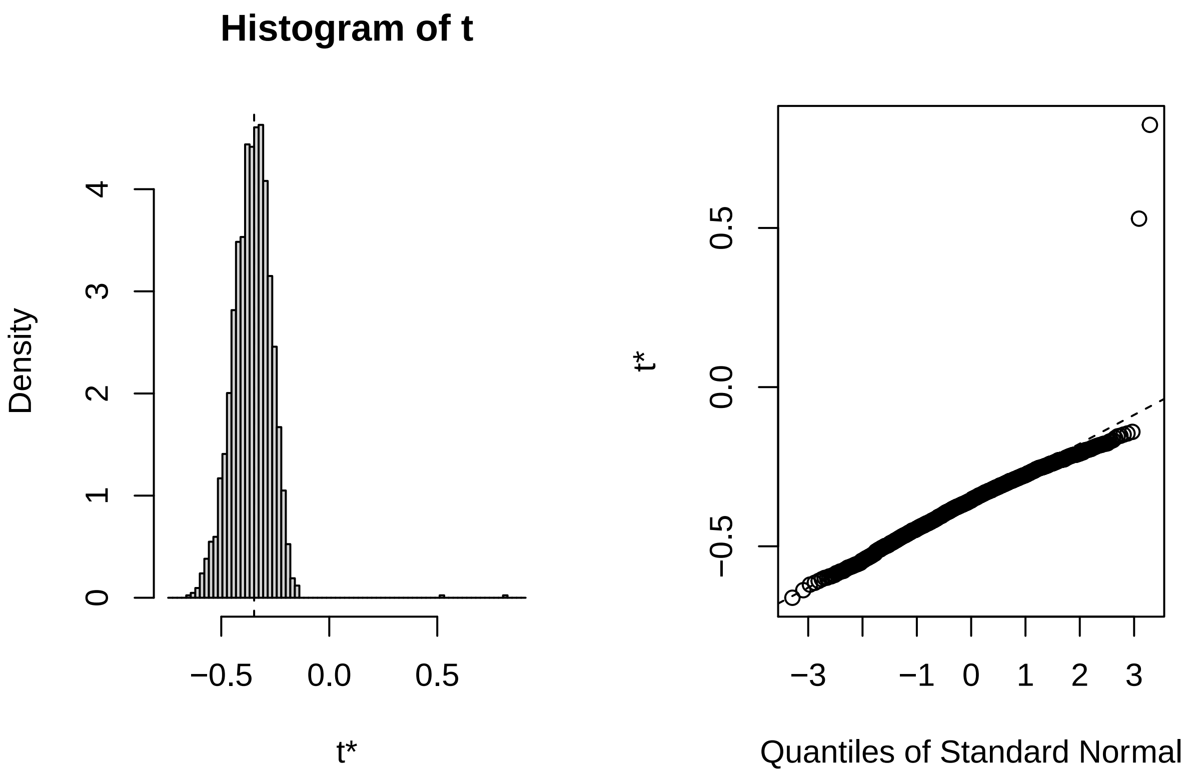
```
## `geom_smooth()` using formula 'y ~ x'
```



There seems to be a downward trend to the estimate of Y in the data using the linear smoother loess as X increases. This could indicate that the data is not truly random.

3. Use non-parametric bootstrap to estimate a test-statistic and compute the p-value.

```
test_statistic = function(data, ind){
  data1=data[ind,] # shuffle
  loess_model = loess(Y~X, data=data1)
  min_estimate = min(loess_model$fitted) # f(X_min)
  max_estimate = max(loess_model$fitted) # f(X_max)
  min_estimate_index = which.min(loess_model$fitted) # X_min_index
  max_estimate_index = which.max(loess_model$fitted) # X_max_index

  t = (max_estimate - min_estimate) / (data1$X[max_estimate_index] - data1$X[min_estimate_index])
  return(t)
}

set.seed(12345)
bootstrap = boot(as.data.frame(data), test_statistic, 2000)
plot(bootstrap)
```



```r
t_vals = bootstrap$t
t_avg = mean(t_vals)

p_value = sum(t_vals > bootstrap$t0) / length(t_vals)

cat("We calculate a p-value of", p_value)
```

```
## We calculate a p-value of 0.472
```

By observing the bootstrap plot we estimate our distribution to be normal. Also, given our test statistic we observe $T < 0$. We come to the conclusion that the lottery is random.

4. Implement a function to do hypothesis testing.

Our null-hypothesis is H_0: Lottery is random. We perform permutation test using the statistic of the previous step and compute the p-value.

```r
B = 2000
n = dim(data)[1]

testing_lottery <- function(data, B) {
  stat = numeric(B)
  n = dim(data)[1]
  for(b in 1:B){
    Y_sample = sample(data$Y, n, replace=TRUE) # Create permutation of sample
    loess_model = loess(Y_sample~data$X)
    min_estimate = min(loess_model$fitted) # f(X_min)
    max_estimate = max(loess_model$fitted) # f(X_max)
    min_estimate_index = which.min(loess_model$fitted) # X_min_index
    max_estimate_index = which.max(loess_model$fitted) # X_max_index
    t = (max_estimate - min_estimate) / (data$X[max_estimate_index] - data$X[min_estimate_index])
    stat[b] = t
  }
  t_0 = test_statistic(data)
  p_value = sum(stat > t_0) / length(stat)
  return(p_value)
}

set.seed(12345)
res = testing_lottery(data, B)

cat("We calculate a p-value of ", res, ". Therefore, we cannot reject H0.", sep="")
```

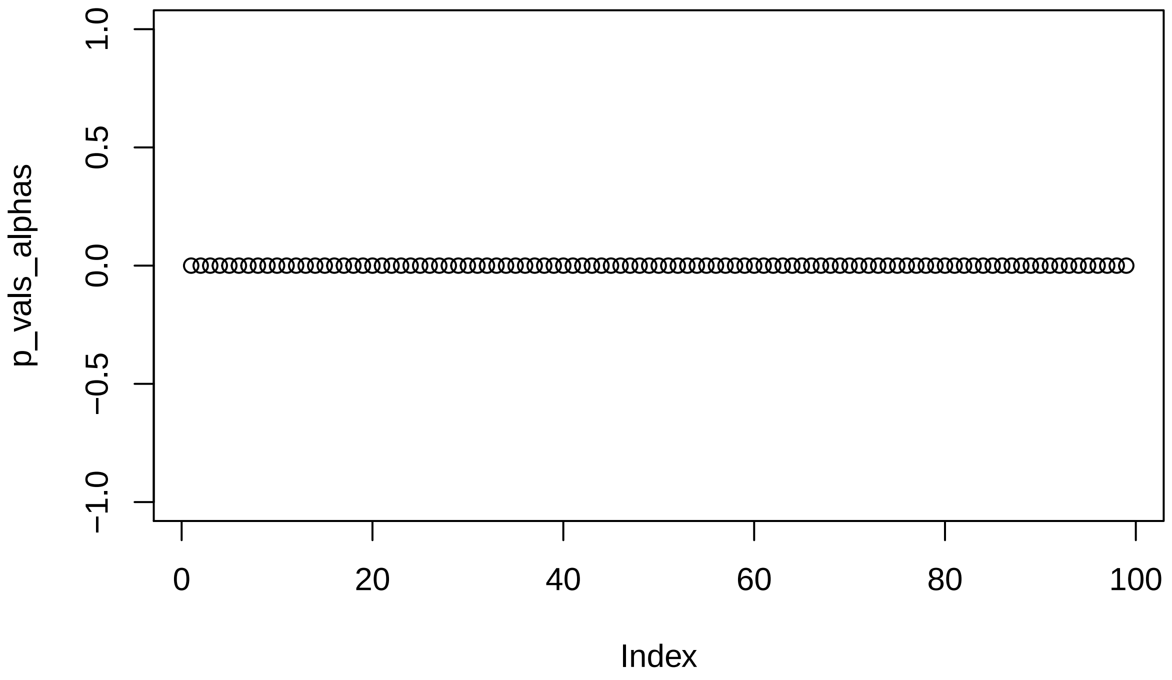
```
## We calculate a p-value of 0.925. Therefore, we cannot reject H0.
```

5. A crude estimate of the power of the test.

```
# a)
alpha = 0.1
Y_x <- function(x, alpha) {
  return (max(0, min(alpha * x + rnorm(1, 183, 10), 366)))
}

set.seed(12345)
Y = sapply(data$X, Y_x, alpha)
non_random_data = data.frame(X=data$X, Y=Y)

# b)
set.seed(12345)
res = testing_lottery(non_random_data, 200)
cat("We calculate a p-value of ", res, ". Therefore, we reject H0.", sep="")
```

```
## We calculate a p-value of 0.005. Therefore, we reject H0.
```

```
# c)
alphas = seq(0.2,10,0.1)
p_vals_alphas = c()
set.seed(12345)
for (alpha in alphas) {
  Y = sapply(data$X, Y_x, alpha)
  non_random_data = data.frame(X=data$X, Y=Y)
  res = testing_lottery(non_random_data, 200)
  p_vals_alphas = c(p_vals_alphas, res)
}

plot(p_vals_alphas)
```

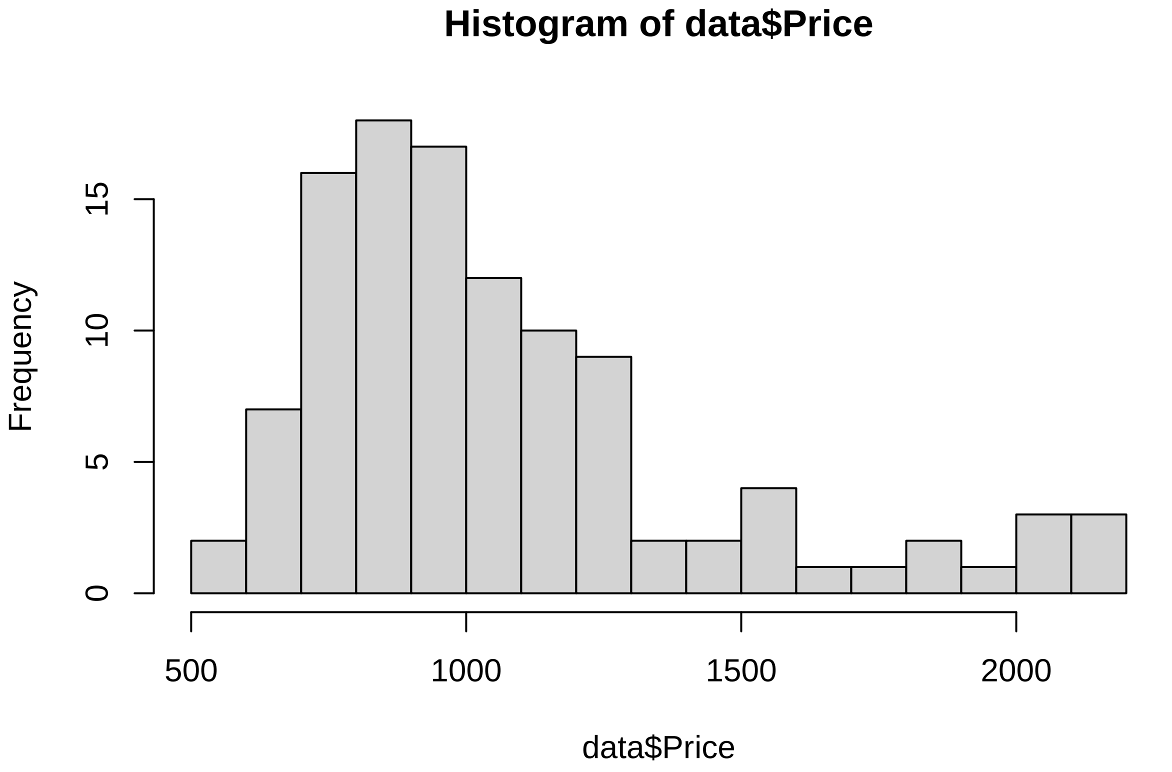


We have generated data that is not random and are rejecting the hypothesis that the data is random for different values of alpha. This is the desired behavior of our test statistic. This rough approach to the power of this statistic shows that we can have some measure of confidence in our result of the previous question.

Question 2: Bootstrap, jacknife and confidence intervals.

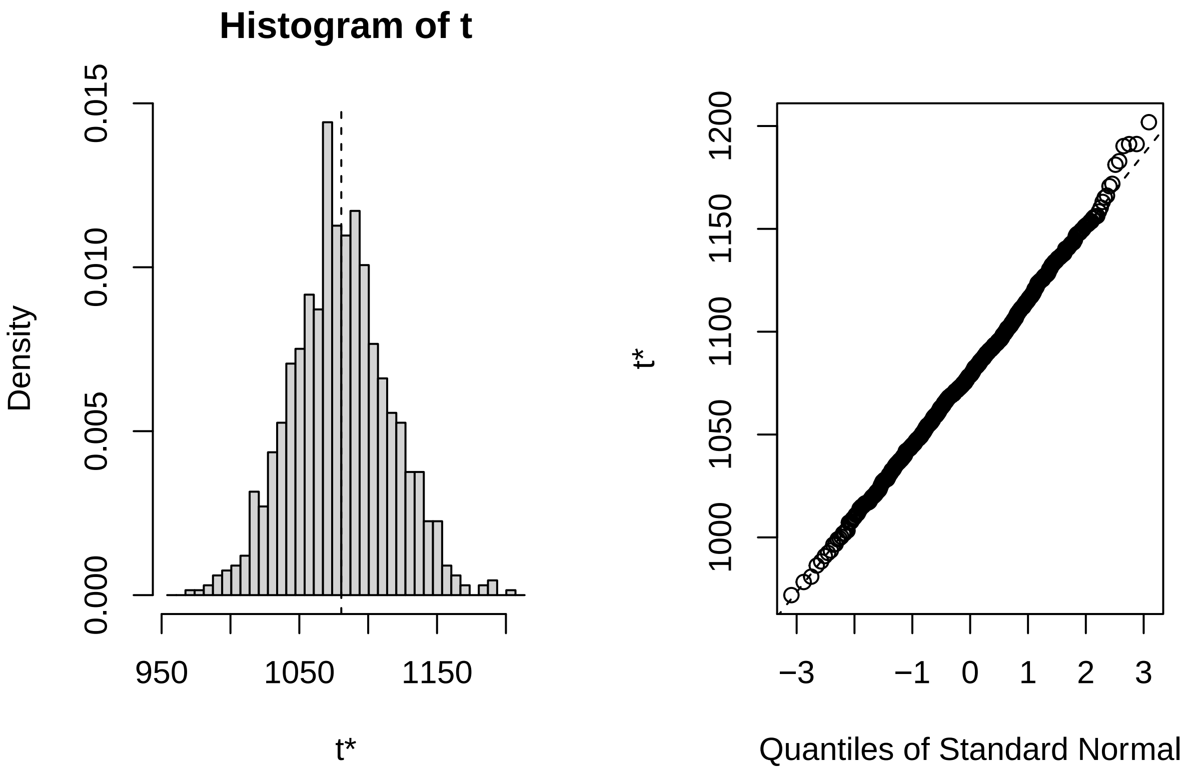
What kind of distribution does Price resemble

```
## The mean price of a home is:  1080.473
```



The plotted distribution reminds us of the Poisson distribution or a Gamma distribution. We will lean for a Gamma distribution as it is the general form of a more general form of distribution that has more flexibility in shape depending on the parameters.

2 Estimating using bootstrap.



```
##
## The estimated bootstrap mean is:  1079.917
```

```
##
## The estimated bootstrap variance is:  1260.909
```

```
##
## The bias-corrected estimated mean is:  1081.029
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bootstrap_obj)
##
## Intervals :
## Level      Normal              Basic
## 95%   (1011, 1151 )   (1011, 1150 )
##
## Level     Percentile            BCa
## 95%   (1011, 1150 )   (1017, 1155 )
## Calculations and Intervals on Original Scale
```

The estimated bootstrap mean, bias corrected estimated mean and sample mean differ by only 2 at most. This means that there is very little bias to correct from our original data calculated mean.

3. Estimating the variance of the mean price using the jackknife and compare it to the previously obtained variance.

```
## The estimated jackknife variance for the estimated mean is:  1320.911
```

The jackknife variance and the bootstrap variance differ by 175. The jackknife variance is larger than that of the bootstrap variance because the jackknife method has a tendency to overestimate. The variances are high overall, but this may be due to the wide range of values of *Price*. The range of price and the presence of some outliers with high values may explain the difference in these values.

4 Comparing confidence intervals.

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = bootstrap_obj)
##
## Intervals :
## Level      Normal              Basic
## 95%   (1011, 1151 )   (1011, 1150 )
##
## Level     Percentile            BCa
## 95%   (1011, 1150 )   (1017, 1155 )
## Calculations and Intervals on Original Scale
```

```
##
## The bias-corrected estimated mean is:  1081.029
```

```
## The length of the
```

The bootstrap distribution is symmetric and has a shape of a normal distribution, this yields confidence intervals that are of comparable size and with mean located in similar locations for each method that computes the confidence intervals. The center of these intervals differ from the estimated mean by 5 units, which is small compared the the range of values of *Price*.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(boot)
library(gridExtra)

data = read.csv("lottery.csv", sep=";")
Y = data$Draft_No
X = data$Day_of_year
data = data.frame(X=data$Day_of_year, Y=data$Draft_No)

plot1 = ggplot(data = data, aes(x = X, y = Y))+
  geom_point()
plot1

loess_model = loess(Y~X, data=data)

plot2 = ggplot(data = data, aes(x = X, y = Y)) +
  geom_point() +
  geom_smooth(method=loess)
plot2


test_statistic = function(data, ind){
  data1=data[ind,] # shuffle
  loess_model = loess(Y~X, data=data1)
  min_estimate = min(loess_model$fitted) # f(X_min)
  max_estimate = max(loess_model$fitted) # f(X_max)
  min_estimate_index = which.min(loess_model$fitted) # X_min_index
  max_estimate_index = which.max(loess_model$fitted) # X_max_index

  t = (max_estimate - min_estimate) / (data1$X[max_estimate_index] - data1$X[min_estimate_index])
  return(t)
}

set.seed(12345)
bootstrap = boot(as.data.frame(data), test_statistic, 2000)
plot(bootstrap)

t_vals = bootstrap$t
t_avg = mean(t_vals)

p_value = sum(t_vals > bootstrap$t0) / length(t_vals)

cat("We calculate a p-value of", p_value)

B = 2000
n = dim(data)[1]

testing_lottery <- function(data, B) {
  stat = numeric(B)
  n = dim(data)[1]
  for(b in 1:B){
    Y_sample = sample(data$Y, n, replace=TRUE) # Create permutation of sample
    loess_model = loess(Y_sample~data$X)
    min_estimate = min(loess_model$fitted) # f(X_min)
    max_estimate = max(loess_model$fitted) # f(X_max)
    min_estimate_index = which.min(loess_model$fitted) # X_min_index
    max_estimate_index = which.max(loess_model$fitted) # X_max_index

    t = (max_estimate - min_estimate) / (data$X[max_estimate_index] - data$X[min_estimate_index])
    stat[b] = t
  }
  t_0 = test_statistic(data)
  p_value = sum(stat > t_0) / length(stat)
  return(p_value)
}

set.seed(12345)
res = testing_lottery(data, B)

cat("We calculate a p-value of ", res, ". Therefore, we cannot reject H0.", sep="")
# a)
alpha = 0.1
Y_x <- function(x, alpha) {
  return (max(0, min(alpha * x + rnorm(1, 183, 10), 366)))
}

set.seed(12345)
Y = sapply(data$X, Y_x, alpha)
non_random_data = data.frame(X=data$X, Y=Y)

# b)
set.seed(12345)
res = testing_lottery(non_random_data, 200)
cat("We calculate a p-value of ", res, ". Therefore, we reject H0.", sep="")

# c)
alphas = seq(0.2,10,0.1)
p_vals_alphas = c()
set.seed(12345)
for (alpha in alphas) {
  Y = sapply(data$X, Y_x, alpha)
  non_random_data = data.frame(X=data$X, Y=Y)
  res = testing_lottery(non_random_data, 200)
  p_vals_alphas = c(p_vals_alphas, res)
}

plot(p_vals_alphas)

library(boot)
data = read.csv2("prices1.csv")

#mean price
average_price = mean(data$Price)
```

```r
cat("The mean price of a home is: ", average_price)

# plot the histogram of price
hist(data$Price, breaks = 12)
stat1 = function(data,vn){
  data = as.data.frame(data[vn,])
  res= mean(data$Price)
  return(res)
}

bootstrap_obj = boot(data, stat1, R= 1000)

plot(bootstrap_obj)

# plot.boot(bootstrap_obj) #doesn't work?
# print.bootci(bootstrap_obj) #doesn't work?

#Bootstrap mean
boot_mean = mean(bootstrap_obj$t)
cat("\nThe estimated bootstrap mean is: ", boot_mean)
#Bootstrap variance
boot_var = var(bootstrap_obj$t[,1])
cat("\nThe estimated bootstrap variance is: ", boot_var)
# Bias corrected estimator (slide 20 of lecture)
bias_correct_estimate = 2*average_price - boot_mean
cat("\nThe bias-corrected estimated mean is: ", bias_correct_estimate,"\n")
#intervals
intervals <- boot.ci(bootstrap_obj)
intervals


# read documentation for type in ?boot.ci
# Confidence interval percentile
ci_percent = boot.ci(bootstrap_obj, conf = 0.95, type = "perc")
# confidence interval BCa
ci_bca = boot.ci(bootstrap_obj, conf = 0.95, type="bca")
#confidence interval first order normal approx
ci_norm = boot.ci(bootstrap_obj, conf = 0.95, type = "norm")

#Slide 15 and 18
n = nrow(data)
#initialize T_star
T_star = c()
for(i in 1:n){
  T_star[i] = n*mean(data$Price) - (n-1)*mean(data$Price[-i])
}

# hist(T_star)
jackknife_var = (1/(n*(n-1))) * sum((T_star - ((1/n) * sum(T_star)))^2)

cat("The estimated jackknife variance for the estimated mean is: ", jackknife_var)

intervals
```

```r
cat("\nThe bias-corrected estimated mean is: ", bias_correct_estimate)
cat("The length of the ")
```