Computational Statistics Lab 5

Simge Cinar & Ronald Yamashita

2023-12-05

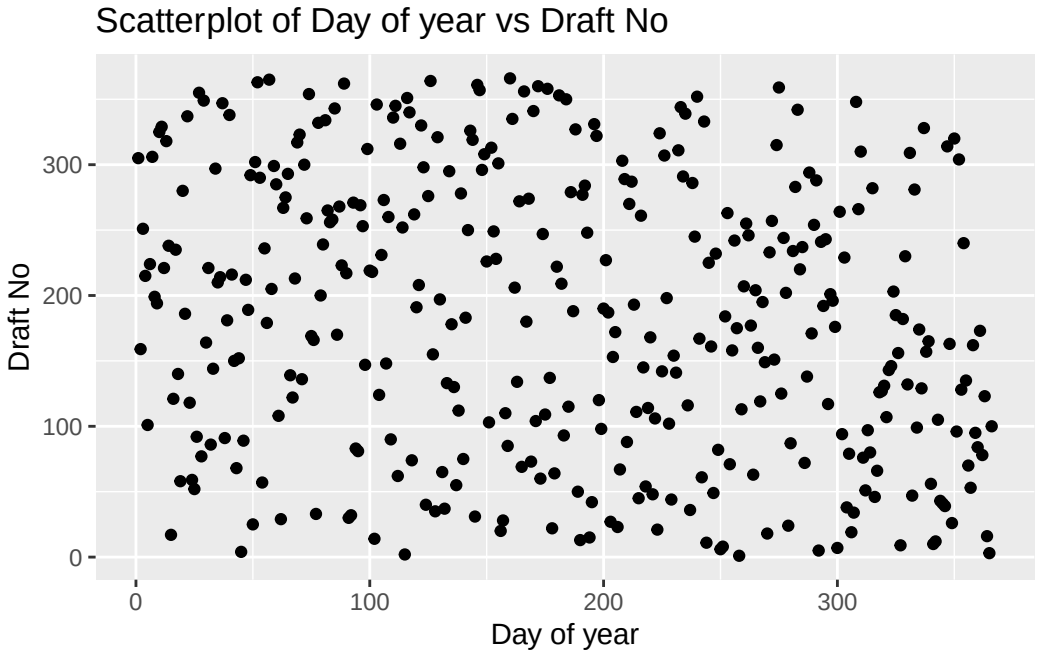
Question 1: Hypothesis testing

Your task is to investigate whether there can be doubts concerning the randomness of the selection of the draft numbers. X=Day of year, Y=Draft No

Part 1)

Question: Create a scatterplot of Y versus X, are any patterns visible?

Answer: The scatterplot can be seen below, there is no pattern visible

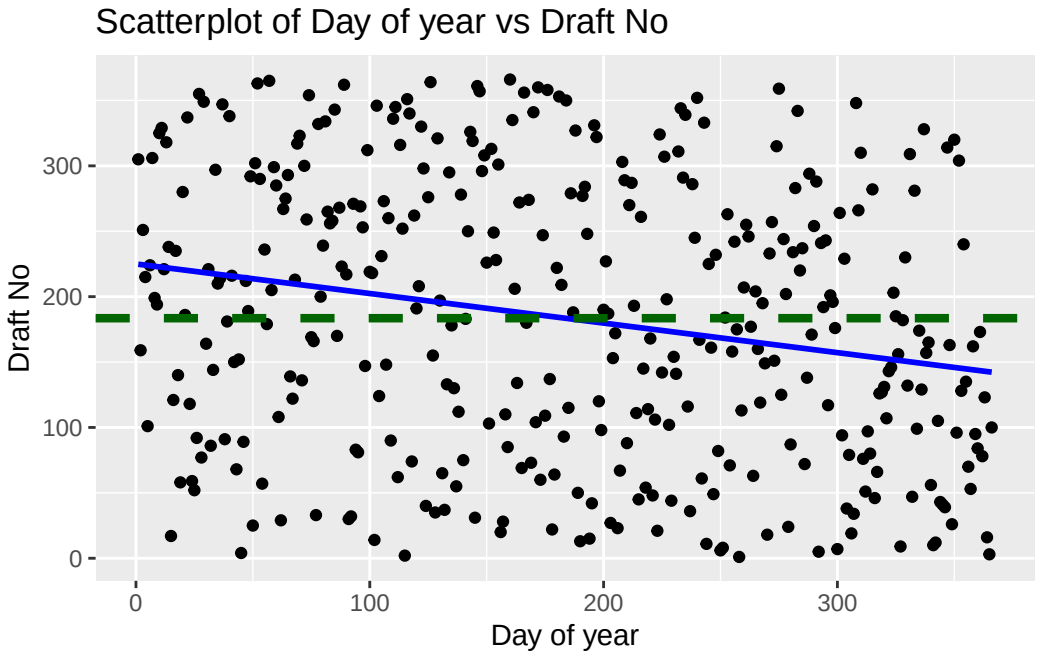


Part 2)

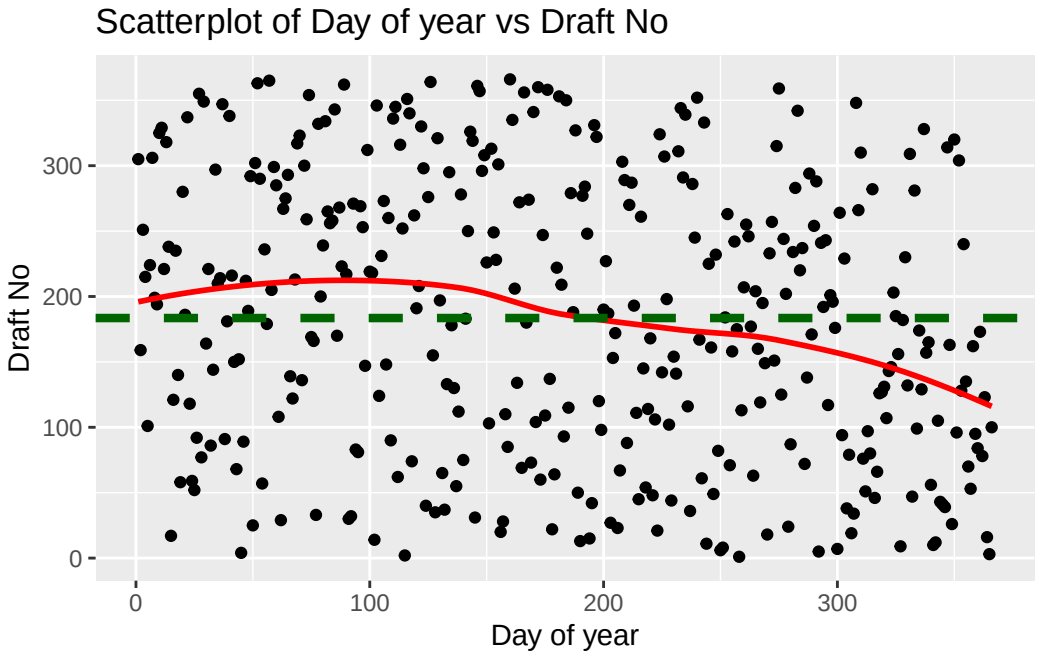
Question: Fit a curve to the data. First fit an ordinary linear model and then fit and then one using loess(). Do these curves suggest that the lottery is random? Explore how the resulting estimated curves are encoded and whether it is possible to identify which parameters are responsible for non–randomness.

Answer: The dashed green line shows the mean value of the y variable in the graphs below. By looking at the graphs, it can be said that lottery might be random since the both lines close to mean and almost flat. The linear regression is encoded in the form $y = \beta_0 + \beta_1 x$. The loess function is not formed from one equation, it fits a curve by making local approximations around each data point. In linear regression, coefficients of the model; and in loess function, the shape and trends of the loess curve can be examined to assess non-randomness.

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




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



Part 3)

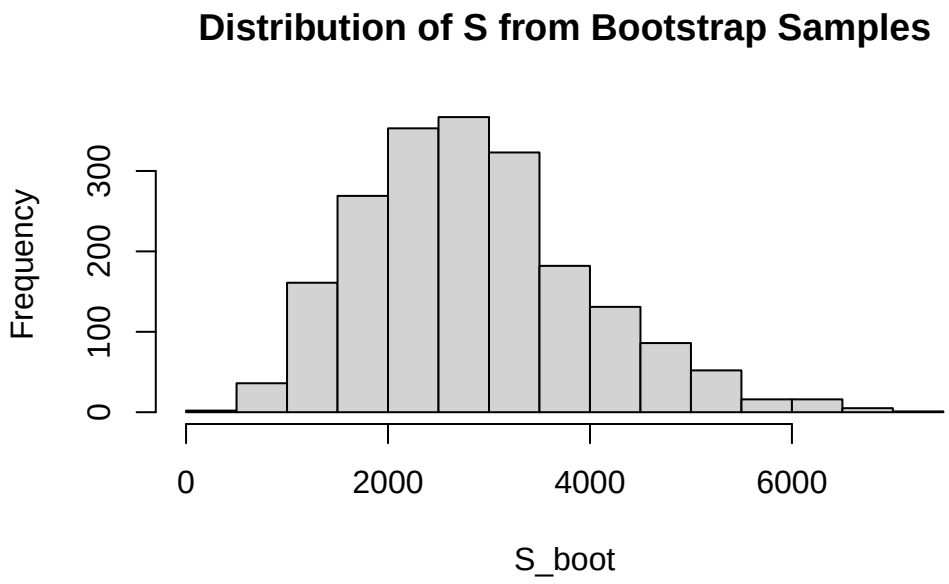
Question: In order to check if the lottery is random, one can use various statistics. One such possibility is based on the expected responses. The fitted loess smoother provides an estimate $\hat{Y}$ as a function of X. It the lottery was random, we would expect $\hat{Y}$ to be a flat line, equalling the empirical mean of the observed responses, Y . The statistic we will consider will be

$$S = \sum_{i=1}^{n} |\hat{Y}_i - \overline{Y}|$$

If S is not close to zero, then this indicates some trend in the data, and throws suspicion on the randomness of the lottery. Estimate S's distribution through a non–parametric bootstrap, taking B = 2000 bootstrap samples. Decide if the lottery looks random, what is the p–value of the observed value of S.

Answer:

```r
set.seed(12345)
calculate_S <- function(y_hat, y_mean) {
  S_boot <- sum(abs(y_hat - y_mean))
  return(S_boot)
}

loess_model <- loess(y ~ x, data = df1)
y_hat <- predict(loess_model, newdata = df1$x)
y_mean <- mean(df1$y)

B <- 2000
S_obs <- calculate_S(y_hat, y_mean)

# Find the distribution of S
S_boot <- vector(length = B)
for (b in 1:B){
  y_resampled <- sample(df1$y, replace = FALSE)
  df_boot <- data.frame(x = df1$x, y = y_resampled)
  loess_model <- loess(y ~ x, data = df_boot)
  y_hat_resampled <- predict(loess_model, newdata = df_boot$x)
  y_mean_resampled <- mean(y_resampled)
  S_boot[b] <- calculate_S(y_hat_resampled, y_mean_resampled)
}
```




```r
p_value <- mean(S_boot  >= S_obs) # area of the right side
cat("S from the data:",S_obs, "\n",
    "p-value:", p_value)
```

```
## S from the data: 8238.649
##  p-value: 0
```

S value calculated from the data is 8238.649. In the histogram above, it can be observed that S values generally lie between 2000 and 4000 if the data is random. Observed value is really high and it is extreme to observe S = 8238.649 when the lottery is random. p-value is 0 which means in the bootstrap we never observed the S value 8238.649.

Part 4)

Question: We will now want to investigate the power of our considered test. First based on the test statistic S, implement a function that tests the hypothesis

$$H_0 : \text{Lottery is random}$$

$$H_1 : \text{Lottery is non–random}$$

The function should return the value of S and its p–value, based on 2000 bootstrap samples.

Answer: We used the bootstrap that is calculated previously for S. The function is as follows:

```
randomness_test <- function(df1, S_boot, B = 2000) {
  loess_model <- loess(y ~ x, data = df1)
  y_hat <- predict(loess_model, newdata = df1$x)
  y_mean <- mean(df1$y)
  S_obs <- calculate_S(y_hat, y_mean)
  p_value <- mean(S_boot >= S_obs)
  return(list(S_observed = S_obs, p_value = p_value))
}

randomness_test(df1, S_boot, B = 2000)
```

```
## $S_observed
## [1] 8238.649
##
## $p_value
## [1] 0
```

Part 5)

Question: Now we will try to make a rough estimate of the power of the test constructed in Step 4 by generating more and more biased samples:

(a) & (b) Create a dataset of the same dimensions as the original data. Choose k, out of the 366, dates and assign them the end numbers of the lottery (i.e., they are not legible for the draw). The remaining 366 − k dates should have random numbers assigned (from the set {1,.., 366 − k}). The k dates should be chosen in two ways:
1) k consecutive dates,
2) as blocks (randomly scattered) of $\lfloor k/3 \rfloor$ consecutive dates (this is of course for k >= 3, and if k is not divisible by 3, then some blocks can be of length $\lfloor k/3 \rfloor + 1$.
For each of the Plug the two new not–completely–random datasets from item 5a into the bootstrap test with B = 2000 and note whether it was rejected.

```
# k consecutive dates (i)
set.seed(12345)
k <- 11
df_new1 <- data.frame(x = df1$x, y = df1$x)
start_date <- sample(1:(366 - k + 1), 1)
```

```r
end_date <- start_date + k - 1
exclude_idx <- start_date:end_date
include_idx <- setdiff(1:366, exclude_idx)
df_new1[include_idx, "y"] <- sample(include_idx)
randomness_test(df_new1, S_boot, B = 2000)
```

```
## $S_observed
## [1] 3350.004
##
## $p_value
## [1] 0.2875
```

k = 11 consecutive draft number gets the value date directly and other 366-11 = 355 draft numbers are assigned randomly in the code above. Also starting point of the consecutive point is selected randomly.

Observed S value is 3350.004 and p-value is 0.2875. p-value is high which means we don't reject the null hypothesis, the lottery is random. It makes sense because 355 samples out of 366 are assigned randomly.

```r
# as 3 blocks (ii)
set.seed(12345)
df_new2 <- data.frame(x = df1$x, y = df1$x)
k <- 11
block_length <- 3
a <- rep(k %/% 3, block_length)
remainder <- k %% 3

idx <- 1
while (remainder != 0){
  a[idx] <- a[idx] + 1
  remainder <- remainder-1
  idx <- idx + 1
}

exclude <- c()
for (i in 1:3){
  k <- a[i]
  start_date <- sample(1:(366 - k + 1), 1)
  end_date <- start_date + k - 1
  exclude <- c(exclude, start_date:end_date)
}

include_idx <- setdiff(1:366, exclude)
df_new2[include_idx, "y"] <- sample(include_idx)
randomness_test(df_new2, S_boot, B = 2000)
```

```
## $S_observed
## [1] 4553.815
##
## $p_value
## [1] 0.0855
```

k = 11 consecutive draft number gets the value date directly and other 366-11 = 355 draft numbers are assigned randomly. We have 3 blocks which means each block has length 4,4,3 respectively and consecutive draft numbers in those blocks get the date number directly. Observed S value is 4553.815 and p-value is 0.0855, we don't reject the null hypothesis. Lottery is random.

(c) Repeat Steps 5a–5b for k = 1,.. , until you have observed a couple of rejections.

Answer:

```
biased_fnc1 <- function(k){
  k <- 30
  df_new1 <- data.frame(x = df1$x, y = df1$x)
  start_date <- sample(1:(366 - k + 1), 1)
  end_date <- start_date + k - 1
  exclude_idx <- start_date:end_date
  include_idx <- setdiff(1:366, exclude_idx)
  df_new1[include_idx, "y"] <- sample(include_idx)
  biased_p1 <- randomness_test(df_new1, S_boot, B = 2000)$p_value
  return(biased_p1)
}

set.seed(12345)
p_val_1 <- vector()
for (k in 1:100){
  p_val_1[k] <- biased_fnc1(k)
  if (sum(p_val_1 < 0.05) >= 10){
    cat("We have 10 rejection\n")
    cat("k:", k, "\n")
    break
  }
}
```

```
## We have 10 rejection
## k: 31
```

We tried k values 1..100 and when k = 31, we get 10 rejection of the null hypothesis and break the loop. There are 366-k randomly assigned draft number hence increasing k decreases the number of draft number assigned randomly. As k increases, we are tend to reject the null hypothesis more.

```
biased_fnc2 <- function(k){
  df_new2 <- data.frame(x = df1$x, y = df1$x)
  k <- 17
  block_length <- 3
  a <- rep(k %/% 3, block_length)
  remainder <- k %% 3

  idx <- 1
  while (remainder != 0){
    a[idx] <- a[idx] + 1
    remainder <- remainder-1
    idx <- idx + 1
  }

  exclude <- c()
  for (i in 1:3){
    k <- a[i]
    start_date <- sample(1:(366 - k + 1), 1)
    end_date <- start_date + k - 1
    exclude <- c(exclude, start_date:end_date)
  }
```

```
  #cat("excluded indices:", exclude, "\n")
  include_idx <- setdiff(1:366, exclude)
  df_new2[include_idx, "y"] <- sample(include_idx)
  biased_p2 <- randomness_test(df_new2, S_boot, B = 2000)$p_value
  return(biased_p2)
}

set.seed(12345)
p_val_2 <- vector()
i <- 0
for (k in 1:100){
  i <- i+1
  p_val_2[i] <- biased_fnc2(k)
  if (sum(p_val_2 < 0.05) >= 10){
    cat("We have 10 rejection\n")
    cat("k:", k, "\n")
    break
  }
}
```

```
## We have 10 rejection
## k: 92
```

We tried k values 3..100 and when k = 92, we get 10 rejection of the null hypothesis and break the loop. Since we have 3 block scattered randomly, the data tend to be more random and takes more iteration compared to the previous part to get 10 rejection.

Question 2: Bootstrap, jackknife and confidence intervals

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.

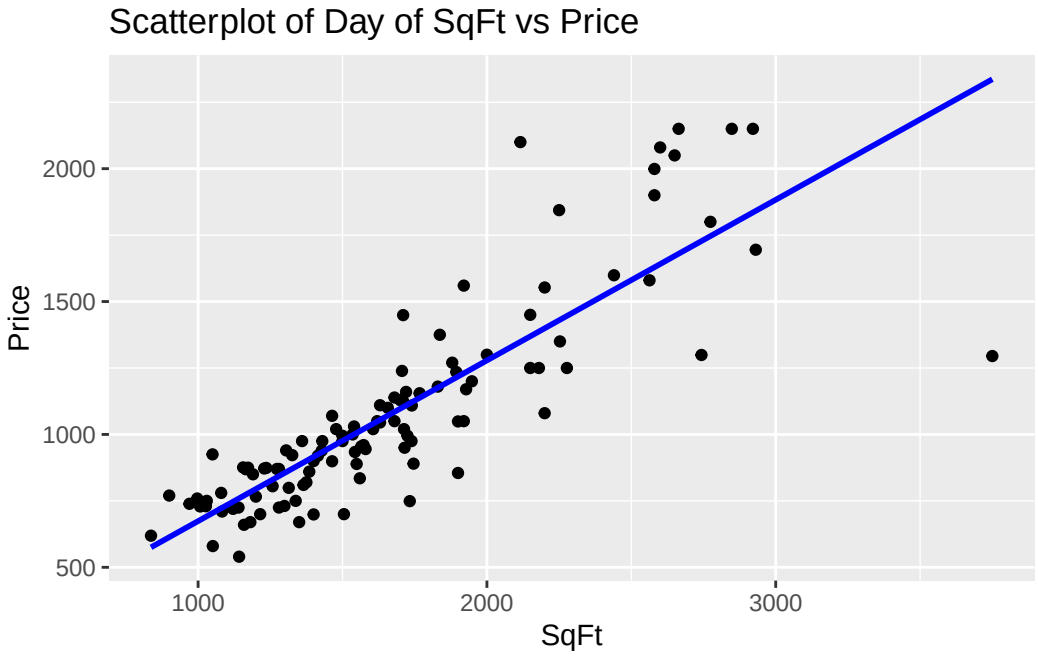
```
## Number of rows: 110
```

part 1)

Question: Create a scatter plot of SqFt versus Price. Fit a linear model to it—does a straight line seem like a good fit?

Answer: The scatterplot can be seen below. It can be said that it's a good fit when SqFt is less than 2000.

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



part 2)

Question: While the data do seem to follow a linear trend, a new sort of pattern seems to appear around 2000ft^2. Consider a new linear model

$$Price = b + a_1 SqFt + a_2(SqFt - c)1_{SqFt>c}$$

where c is the area value where the model changes. You can determine c using an optimizer, e.g., optim(), with the residual sum of squares (RSS) as the value to be minimized. For each value of c, the objective function should estimate b, a1, and a2; then calculate (and return) the resulting RSS.

Answer: The optimal parameters are as follows:

```
calculate_RSS <- function(par, data){
  a1 <- par[1]
  a2 <- par[2]
  b <- par[3]
  c <- par[4]
  y <- data$Price
  x <- data$SqFt
  y_hat <- ifelse(x > c, b + a1 * x + a2 * (x - c), b + a1 * x)
  rss <- sum((y_hat-y)^2)
  return(rss)
}

optimize_c <- function(data){
  # choose initial values from the linear regression model
  init_par <- c(0.6046,0.6046, 69.3066, 2000)
  optimal_par <- optim(init_par, calculate_RSS, data = data)$par
  return(optimal_par)
}

optimal_par <- optimize_c(df2)
optimal_c <- optimal_par[4]
cat("Optimal b:", optimal_par[3], "\n")
```

```
## Optimal b: -63.23802
```

```
cat("Optimal a1:", optimal_par[1], "\n")
```

```
## Optimal a1: 0.6908508
```

```
cat("Optimal a2:", optimal_par[2], "\n")
```

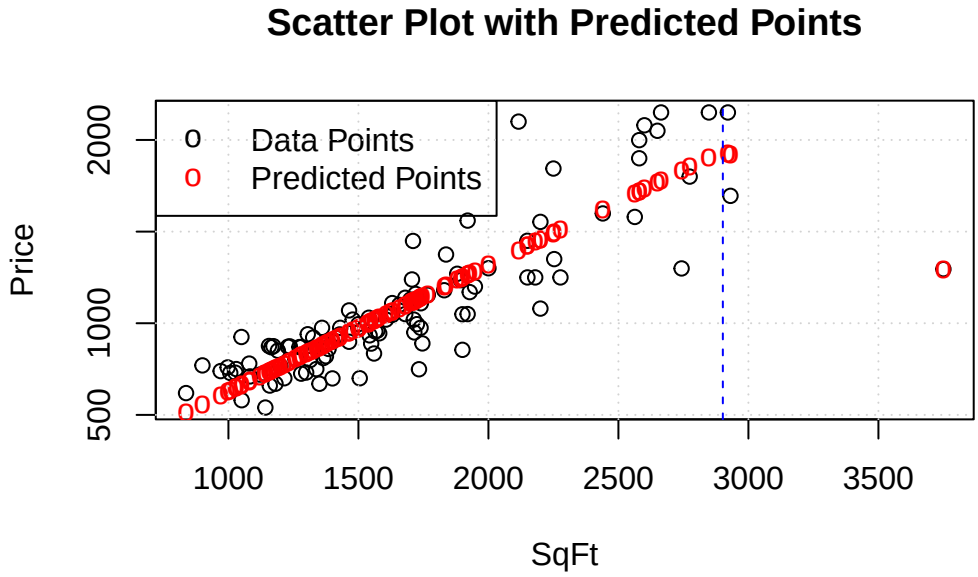
```
## Optimal a2: -1.455803
```

```
cat("Optimal c:", optimal_c, "\n")
```

```
## Optimal c: 2901.664
```

```
cat("RSS with optimal parameters:", calculate_RSS(optimal_par, df2))
```

```
## RSS with optimal parameters: 3300648
```

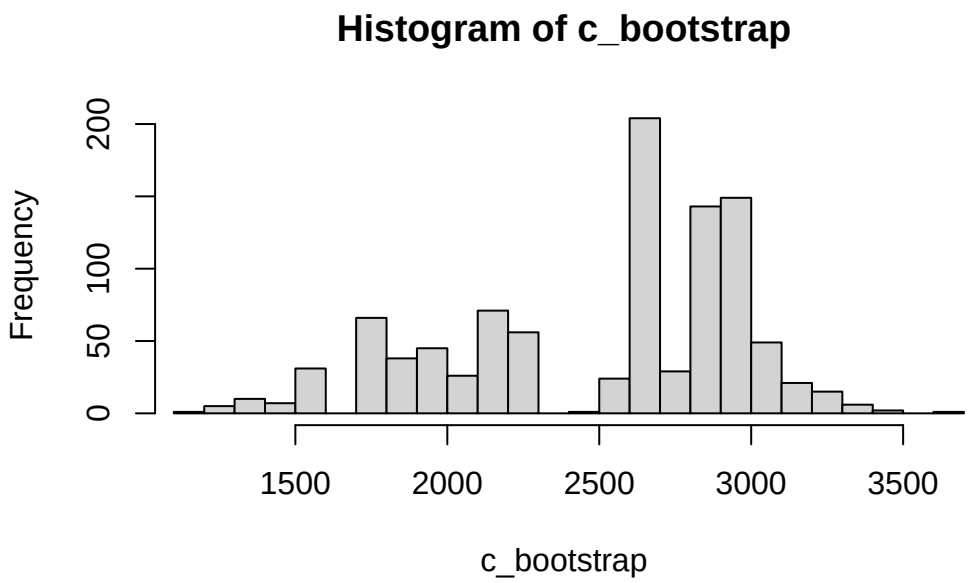


Part 3)

Question: Using the bootstrap estimate the distribution of c. Determine the bootstrap bias–correction and the variance of c. Compute a 95% confidence interval for c using bootstrap percentile, bootstrap BCa, and first–order normal approximation

Answer: The histogram of c can be seen below

```
estimate_c <- function(data, indices) {
  sampled_data <- data[indices, ]
  optimal_c <- optimize_c(sampled_data)[4]
  return(optimal_c)
}

set.seed(12345)
boot_results <- boot(data = df2, statistic = estimate_c, R = 1000)
c_bootstrap <- boot_results$t
hist(c_bootstrap, xlim = c(min(c_bootstrap), max(c_bootstrap)), breaks = 20)
```




```
cat("bootstrap bias-correction:", 2*optimal_c - mean(c_bootstrap), "\n")
```

```
## bootstrap bias-correction: 3292.881
```

```
cat("variance:", var(c_bootstrap))
```

```
## variance: 237227.2
```

```
ci_interval <- boot.ci(boot_results,  type = c("norm", "bca", "perc"))
ci_interval
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_results, type = c("norm", "bca", "perc"))
##
## Intervals :
## Level      Normal             Percentile            BCa
## 95%   (2338, 4248 )   (1505, 3197 )   (2277, 3630 )
## Calculations and Intervals on Original Scale
## Warning : BCa Intervals used Extreme Quantiles
## Some BCa intervals may be unstable
```

part 4)

Question: Estimate the variance of c using the jackknife and compare it with the bootstrap estimate
Answer:

```
c_jacknife <- numeric(nrow(df2))
for (i in 1:nrow(df2)) {
  # Leave one observation out
  temp_data <- df2[-i, ]
  c_jacknife[i] <- optimize_c(temp_data)[4]
}

n <- nrow(df2)
T_i <- (n*c) - ((n-1) * (c_jacknife))
```

```r
jackknife_variance <- 1/(n*(n-1)) * sum((T_i - mean(c_jacknife))^2)
cat("Jacknife variance:", jackknife_variance)
```

```
## Jacknife variance: 2555057
```

Jacknife variance is more than bootstrap variance as expected. Bootstrap tends to underestimate the variance due to its overlap in the resampled dataset, Jackknife tends to provide a larger variance estimate due to its emphasis on leaving out one observation at a time.

part 5)

Question: Summarize the results of your investigation by comparing all of the confidence intervals with respect to their length and the location of c inside them.

Answer: The length of the confidence intervals are as follows:

```r
normal_ci_length <- 4248 - 2338
bca_ci_length <- 3630 - 2277
perc_ci_length <- 3197 - 1505
cat("Normal CI length:", normal_ci_length, "\n")
```

```
## Normal CI length: 1910
```

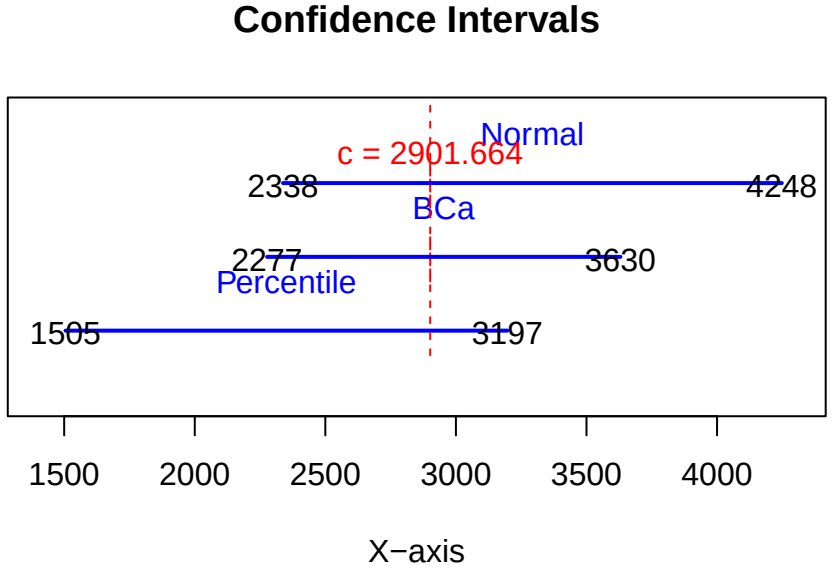
```r
cat("BCa CI length:", bca_ci_length, "\n")
```

```
## BCa CI length: 1353
```

```r
cat("Percentile CI length:", perc_ci_length, "\n")
```

```
## Percentile CI length: 1692
```

The confidence interval of normal is more compared to the other. Each confidence interval contains the optimal c value.



In normal confidence interval, c value is closer to the the lower bound and in the percentile confidence interval c values is closer to the upper bound.

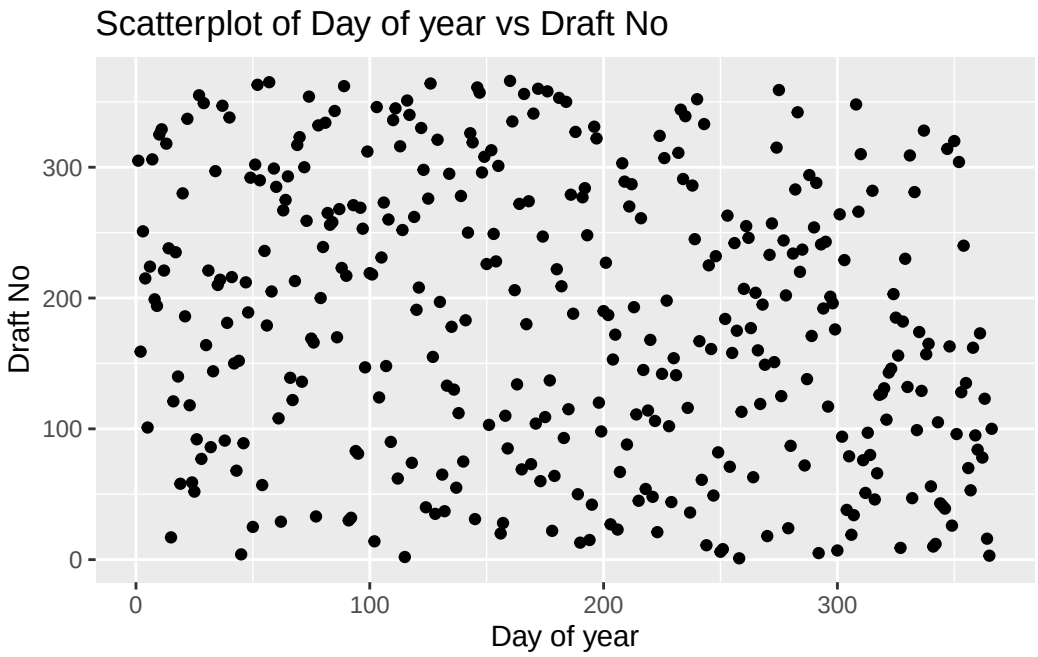
Appendix

Question 1

```
# Data manipulation
df1_init <- read.csv("lottery.csv")
split_data <- strsplit(as.character(df1_init$Day.Month.Mo.Number.Day_of_year.Draft_No), ";")
split_matrix <- do.call(rbind, split_data)
df1_init2 <- as.data.frame(split_matrix)
colnames(df1_init2) <- c("Day", "Month", "Mo.Number", "Day_of_year", "Draft_No")
df1_init2[, c(1,3,4,5)] <- lapply(df1_init2[, c(1,3,4,5)], as.numeric)

df1 <- df1_init2[, c("Day_of_year", "Draft_No")]
colnames(df1) <- c("x", "y")
```

Part 1)

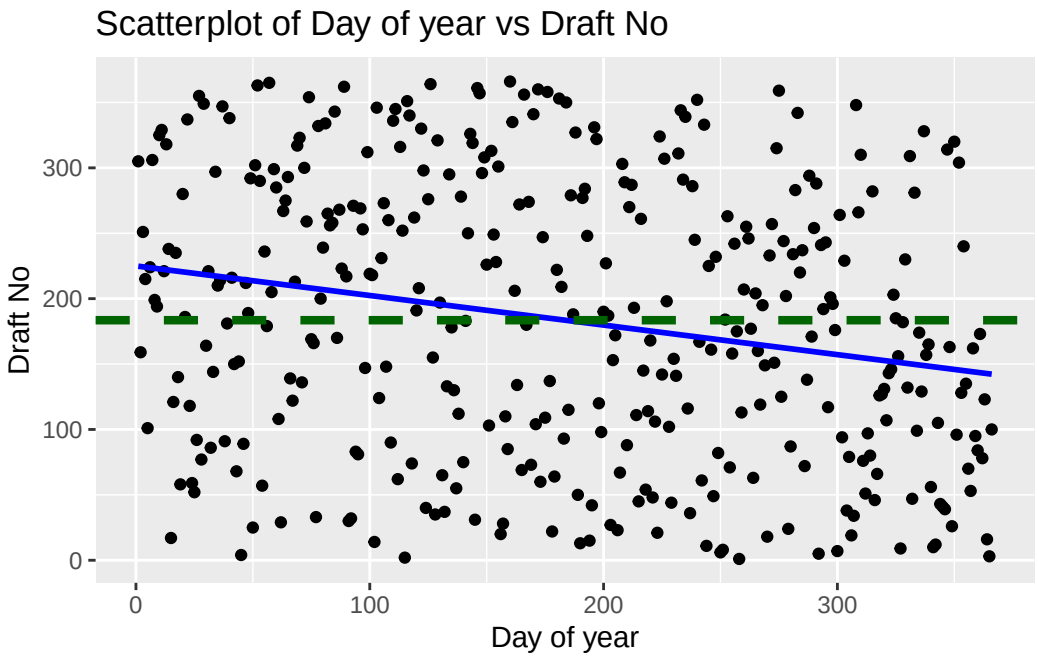
```
ggplot(df1, aes(x = x, y = y)) +
  geom_point() +
  labs(x = "Day of year", y = "Draft No") +
  ggtitle("Scatterplot of Day of year vs Draft No")
```



Part 2)

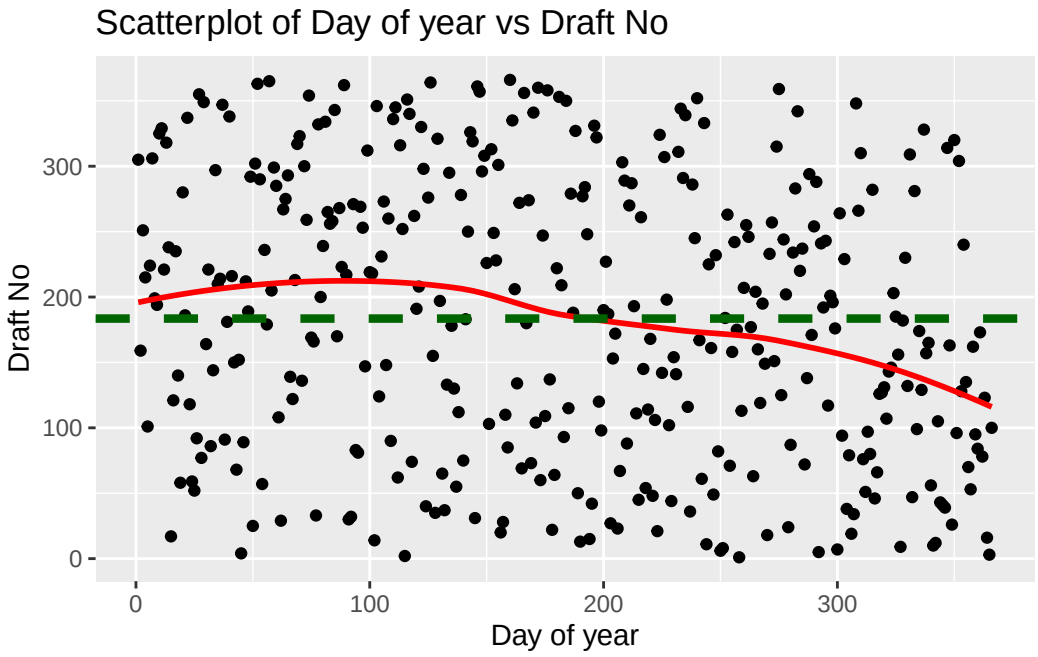
```
# Create a scatterplot with an ordinary linear model
ggplot(df1, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "blue") +
  labs(x = "Day of year", y = "Draft No") +
  ggtitle("Scatterplot of Day of year vs Draft No") +
  geom_hline(yintercept = mean(df1$y), color = "darkgreen",
             linetype = "dashed", linewidth = 1.5)
```

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




```
# Create a scatterplot with an ordinary with loess()
ggplot(df1, aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "loess", se = FALSE, color = "red") +
  labs(x = "Day of year", y = "Draft No") +
  ggtitle("Scatterplot of Day of year vs Draft No") +
  geom_hline(yintercept = mean(df1$y), color = "darkgreen",
             linetype = "dashed", linewidth = 1.5)
```

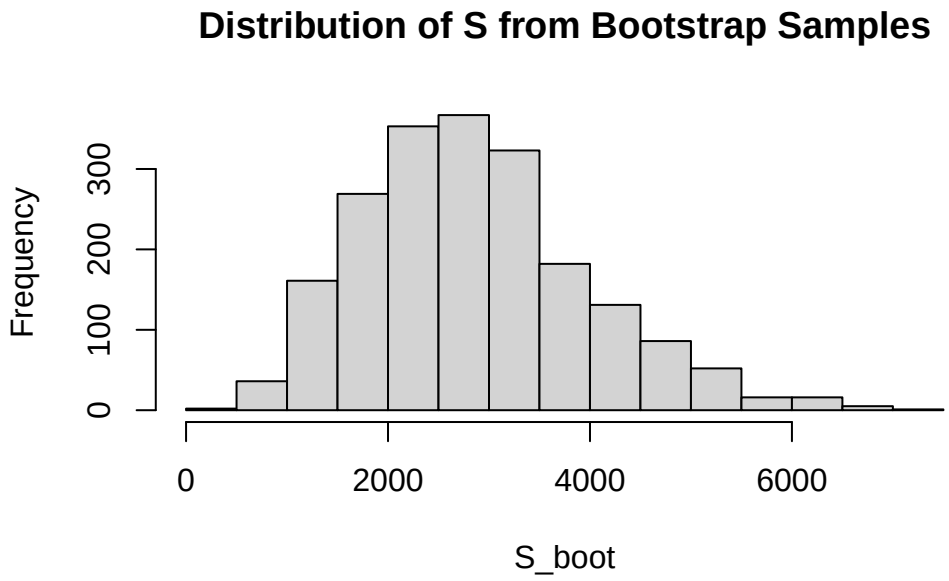
```
## `geom_smooth()` using formula = 'y ~ x'
```



Part 3)

```
set.seed(12345)
calculate_S <- function(y_hat, y_mean) {
  S_boot <- sum(abs(y_hat - y_mean))
  return(S_boot)
}

loess_model <- loess(y ~ x, data = df1)
y_hat <- predict(loess_model, newdata = df1$x)
y_mean <- mean(df1$y)

B <- 2000
S_obs <- calculate_S(y_hat, y_mean)

# Find the distribution of S
S_boot <- vector(length = B)
for (b in 1:B){
  y_resampled <- sample(df1$y, replace = FALSE)
  df_boot <- data.frame(x = df1$x, y = y_resampled)
  loess_model <- loess(y ~ x, data = df_boot)
  y_hat_resampled <- predict(loess_model, newdata = df_boot$x)
  y_mean_resampled <- mean(y_resampled)
  S_boot[b] <- calculate_S(y_hat_resampled, y_mean_resampled)
}
```

```
hist(S_boot, breaks = 10, main = "Distribution of S from Bootstrap Samples")
abline(v = S_obs, col = "red", lwd = 2)
```




```
p_value <- mean(S_boot  >= S_obs) # area of the right side
cat("S from the data:",S_obs, "\n",
    "p-value:", p_value)
```

```
## S from the data: 8238.649
##  p-value: 0
```

Part 4)

```
randomness_test <- function(df1, S_boot, B = 2000) {
  loess_model <- loess(y ~ x, data = df1)
  y_hat <- predict(loess_model, newdata = df1$x)
  y_mean <- mean(df1$y)
  S_obs <- calculate_S(y_hat, y_mean)
  p_value <- mean(S_boot >= S_obs)
  return(list(S_observed = S_obs, p_value = p_value))
}

randomness_test(df1, S_boot, B = 2000)
```

```
## $S_observed
## [1] 8238.649
##
## $p_value
## [1] 0
```

Part 5)

```
# k consecutive dates (i)
set.seed(12345)
k <- 11
df_new1 <- data.frame(x = df1$x, y = df1$x)
start_date <- sample(1:(366 - k + 1), 1)
end_date <- start_date + k - 1
exclude_idx <- start_date:end_date
include_idx <- setdiff(1:366, exclude_idx)
df_new1[include_idx, "y"] <- sample(include_idx)
randomness_test(df_new1, S_boot, B = 2000)
```

```
## $S_observed
## [1] 3350.004
##
## $p_value
## [1] 0.2875
```

```
# as 3 blocks (ii)
set.seed(12345)
df_new2 <- data.frame(x = df1$x, y = df1$x)
k <- 11
block_length <- 3
a <- rep(k %/% 3, block_length)
remainder <- k %% 3

idx <- 1
while (remainder != 0){
  a[idx] <- a[idx] + 1
  remainder <- remainder-1
  idx <- idx + 1
}

exclude <- c()
for (i in 1:3){
```

```
  k <- a[i]
  start_date <- sample(1:(366 - k + 1), 1)
  end_date <- start_date + k - 1
  exclude <- c(exclude, start_date:end_date)
}

include_idx <- setdiff(1:366, exclude)
df_new2[include_idx, "y"] <- sample(include_idx)
randomness_test(df_new2, S_boot, B = 2000)
```

```
## $S_observed
## [1] 4553.815
##
## $p_value
## [1] 0.0855
```

```
biased_fnc1 <- function(k){
  k <- 30
  df_new1 <- data.frame(x = df1$x, y = df1$x)
  start_date <- sample(1:(366 - k + 1), 1)
  end_date <- start_date + k - 1
  exclude_idx <- start_date:end_date
  include_idx <- setdiff(1:366, exclude_idx)
  df_new1[include_idx, "y"] <- sample(include_idx)
  biased_p1 <- randomness_test(df_new1, S_boot, B = 2000)$p_value
  return(biased_p1)
}

set.seed(12345)
p_val_1 <- vector()
for (k in 1:100){
  p_val_1[k] <- biased_fnc1(k)
  if (sum(p_val_1 < 0.05) >= 10){
    cat("We have 10 rejection\n")
    cat("k:", k, "\n")
    break
  }
}
```

```
## We have 10 rejection
## k: 31
```

```
biased_fnc2 <- function(k){
  df_new2 <- data.frame(x = df1$x, y = df1$x)
  k <- 17
  block_length <- 3
  a <- rep(k %/% 3, block_length)
  remainder <- k %% 3

  idx <- 1
  while (remainder != 0){
    a[idx] <- a[idx] + 1
    remainder <- remainder-1
    idx <- idx + 1
  }
```

```
  exclude <- c()
  for (i in 1:3){
    k <- a[i]
    start_date <- sample(1:(366 - k + 1), 1)
    end_date <- start_date + k - 1
    exclude <- c(exclude, start_date:end_date)
  }

  #cat("excluded indices:", exclude, "\n")
  include_idx <- setdiff(1:366, exclude)
  df_new2[include_idx, "y"] <- sample(include_idx)
  biased_p2 <- randomness_test(df_new2, S_boot, B = 2000)$p_value
  return(biased_p2)
}

set.seed(12345)
p_val_2 <- vector()
i <- 0
for (k in 1:100){
  i <- i+1
  p_val_2[i] <- biased_fnc2(k)
  if (sum(p_val_2 < 0.05) >= 10){
    cat("We have 10 rejection\n")
    cat("k:", k, "\n")
    break
  }
}
```

```
## We have 10 rejection
## k: 92
```

Question 2

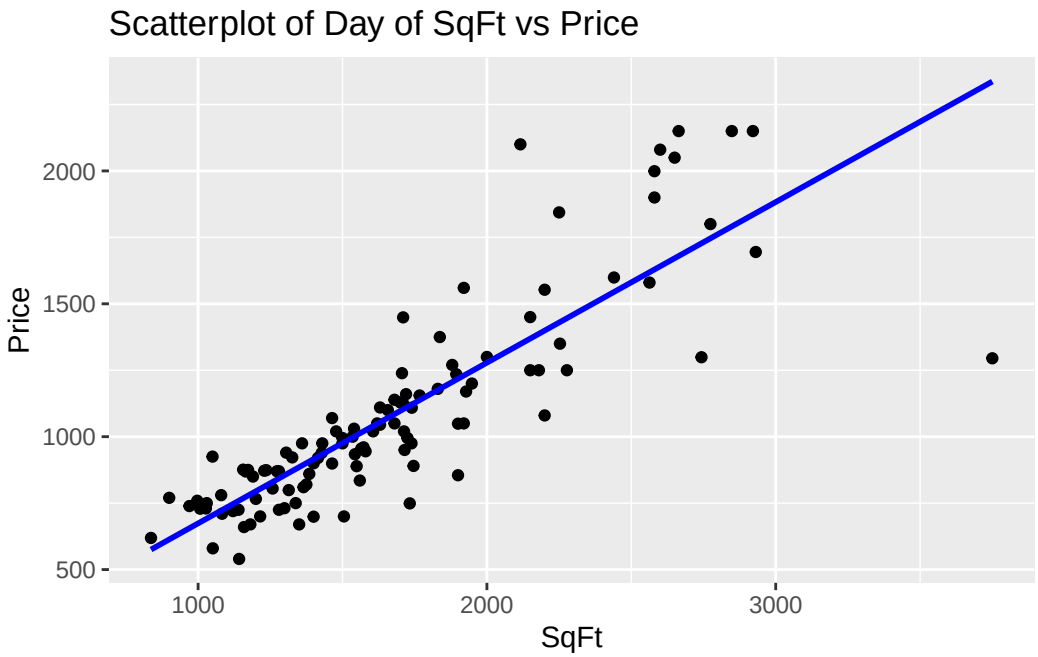
```
df2_init <- read.csv("prices1.csv")
split_data <- strsplit(as.character(df2_init$Price.SqFt.FEATS.Taxes), ";")
split_matrix <- do.call(rbind, split_data)
df2_init2 <- as.data.frame(split_matrix)
colnames(df2_init2) <- c("Price", "SqFt", "FEATS", "Taxes")

df2_init2[, c(1,2,3,4)] <- lapply(df2_init2[, c(1,2,3,4)], as.numeric)
df2 <- df2_init2[, c("SqFt", "Price")]
cat("Number of rows:", nrow(df2))
```

```
## Number of rows: 110
```

part 1)

```
ggplot(df2, aes(x = SqFt, y = Price)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "blue") +
  labs(x = "SqFt", y = "Price") +
  ggtitle("Scatterplot of Day of SqFt vs Price")
```

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



part 2)

```
calculate_RSS <- function(par, data){
  a1 <- par[1]
  a2 <- par[2]
  b <- par[3]
  c <- par[4]
  y <- data$Price
  x <- data$SqFt
  y_hat <- ifelse(x > c, b + a1 * x + a2 * (x - c), b + a1 * x)
  rss <- sum((y_hat-y)^2)
  return(rss)
}

optimize_c <- function(data){
  # choose initial values from the linear regression model
  init_par <- c(0.6046,0.6046, 69.3066, 2000)
  optimal_par <- optim(init_par, calculate_RSS, data = data)$par
  return(optimal_par)
}

optimal_par <- optimize_c(df2)
optimal_c <- optimal_par[4]
cat("Optimal b:", optimal_par[3], "\n")
```

```
## Optimal b: -63.23802
```

```
cat("Optimal a1:", optimal_par[1], "\n")
```

```
## Optimal a1: 0.6908508
```

```r
cat("Optimal a2:", optimal_par[2], "\n")
```

```
## Optimal a2: -1.455803
```

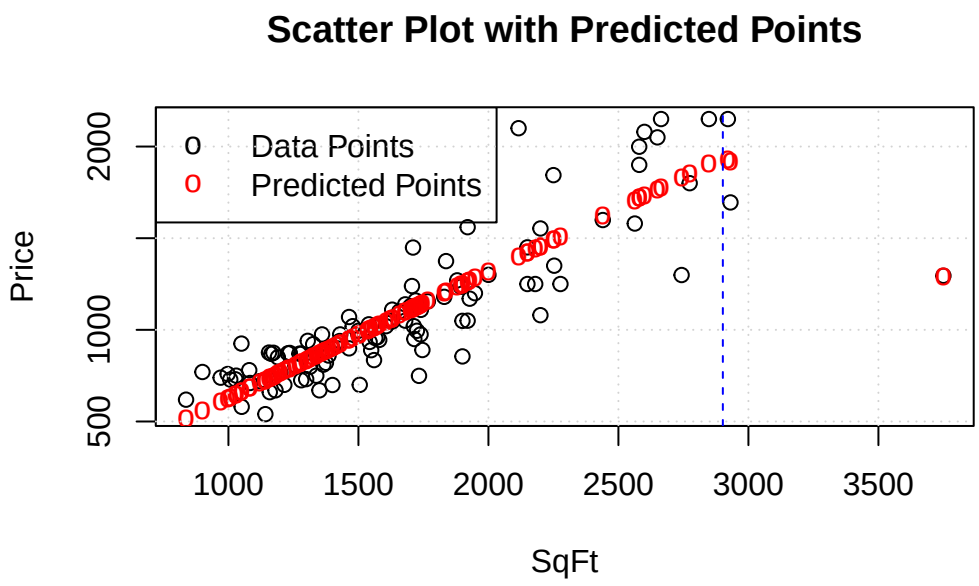
```r
cat("Optimal c:", optimal_c, "\n")
```

```
## Optimal c: 2901.664
```

```r
cat("RSS with optimal parameters:", calculate_RSS(optimal_par, df2))
```

```
## RSS with optimal parameters: 3300648
```

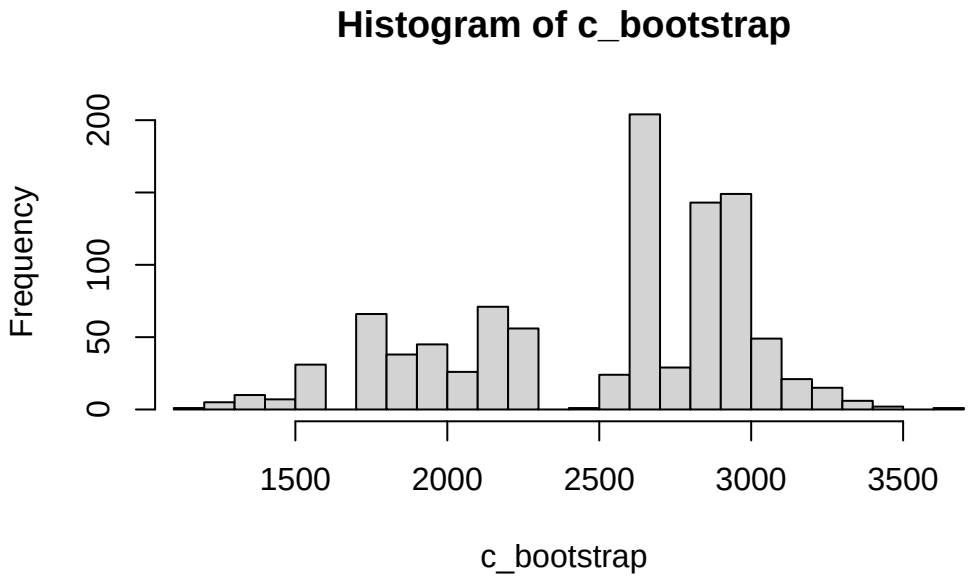
```r
# Plot with predictions
a1 <- optimal_par[1]
a2 <- optimal_par[2]
b <- optimal_par[3]
c <- optimal_par[4]

y <- df2$Price
x <- df2$SqFt

pred <- ifelse(x > c, b + a1 * x + a2 * (x - c), b + a1 * x)

plot(x, y, xlab = "SqFt", ylab = "Price", main = "Scatter Plot with Predicted Points")
points(x, pred, col = "red", pch = "o", lwd = 1.5)
legend("topleft", legend = c("Data Points", "Predicted Points"), col = c("black", "red"), pch = "o", lty
abline(v = c, col = "blue", lty = 2) # breaking point of the linear regression
grid()
```



Part 3)

```r
estimate_c <- function(data, indices) {
  sampled_data <- data[indices, ]
  optimal_c <- optimize_c(sampled_data)[4]
  return(optimal_c)
}
```

```r
set.seed(12345)
boot_results <- boot(data = df2, statistic = estimate_c, R = 1000)
c_bootstrap <- boot_results$t
hist(c_bootstrap, xlim = c(min(c_bootstrap), max(c_bootstrap)), breaks = 20)
```




```r
cat("bootstrap bias-correction:", 2*optimal_c - mean(c_bootstrap), "\n")
```

```
## bootstrap bias-correction: 3292.881
```

```r
cat("variance:", var(c_bootstrap))
```

```
## variance: 237227.2
```

```r
ci_interval <- boot.ci(boot_results,  type = c("norm", "bca", "perc"))
ci_interval
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_results, type = c("norm", "bca", "perc"))
##
## Intervals :
## Level      Normal             Percentile            BCa
## 95%   (2338, 4248 )   (1505, 3197 )   (2277, 3630 )
## Calculations and Intervals on Original Scale
## Warning : BCa Intervals used Extreme Quantiles
## Some BCa intervals may be unstable
```

part 4)

```r
c_jacknife <- numeric(nrow(df2))
for (i in 1:nrow(df2)) {
  # Leave one observation out
  temp_data <- df2[-i, ]
  c_jacknife[i] <- optimize_c(temp_data)[4]
}
```

```r
n <- nrow(df2)
T_i <- (n*c) - ((n-1) * (c_jacknife))
jackknife_variance <- 1/(n*(n-1)) * sum((T_i - mean(c_jacknife))^2)
cat("Jacknife variance:", jackknife_variance)
```

```
## Jacknife variance: 2555057
```

part 5)

```r
normal_ci_length <- 4248 - 2338
bca_ci_length <- 3630 - 2277
perc_ci_length <- 3197 - 1505
cat("Normal CI length:", normal_ci_length, "\n")
```

```
## Normal CI length: 1910
```

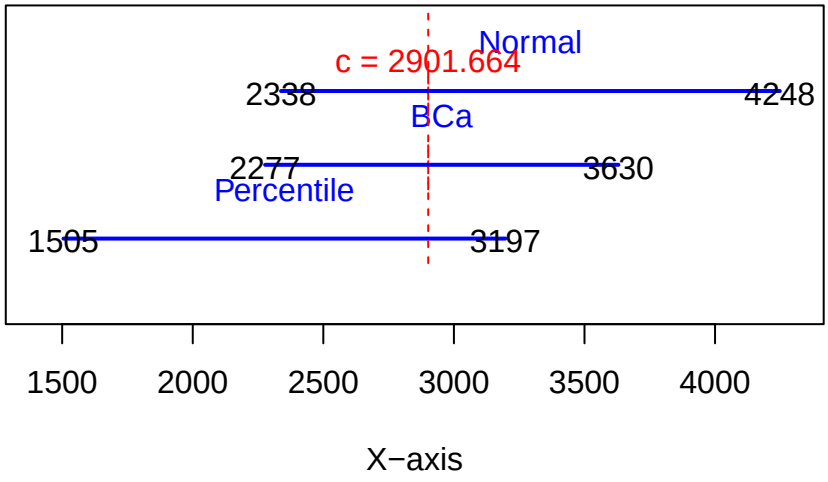
```r
cat("BCa CI length:", bca_ci_length, "\n")
```

```
## BCa CI length: 1353
```

```r
cat("Percentile CI length:", perc_ci_length, "\n")
```

```
## Percentile CI length: 1692
```

```r
# Create a number line plot
plot(1, type = "n", xlim = c(1400, 4300), ylim = c(0, 1.2), xlab = "X-axis", ylab = "", main = "Confide

# Define the confidence intervals and their labels
intervals <- list(c(1505, 3197), c(2277, 3630), c(2338, 4248))
c_values <- c(2901.664, 2901.664, 2901.664)
labels <- c("Percentile", "BCa", "Normal")

# Draw confidence intervals on the number line
for (i in 1:length(intervals)) {
  segments(intervals[[i]][1], 0.3 * i, intervals[[i]][2], 0.3 * i,
           col = "blue", lwd = 2)
  text(intervals[[i]][1], 0.3 * i + 0.1, intervals[[i]][1], pos = 1)
  text(intervals[[i]][2], 0.3 * i + 0.1, intervals[[i]][2], pos = 1)
  text((intervals[[i]][1] + intervals[[i]][2]) / 2, 0.3 * i + 0.2,
       labels[i], col = "blue")

  # Adding vertical dashed lines at c
  segments(c_values[i], 0.3 * i - 0.1, c_values[i], 0.3 * i + 0.4,
           col = "red", lty = "dashed")
}

text(c_values[1], 0.9, "c = 2901.664", pos = 3, col = "red")
```


Confidence Intervals
Normal
c = 2901.664
2338
4248
BCa
2277
3630
Percentile
1505
3197
1500
2000
2500
3000
3500
4000
X−axis