Homework 06

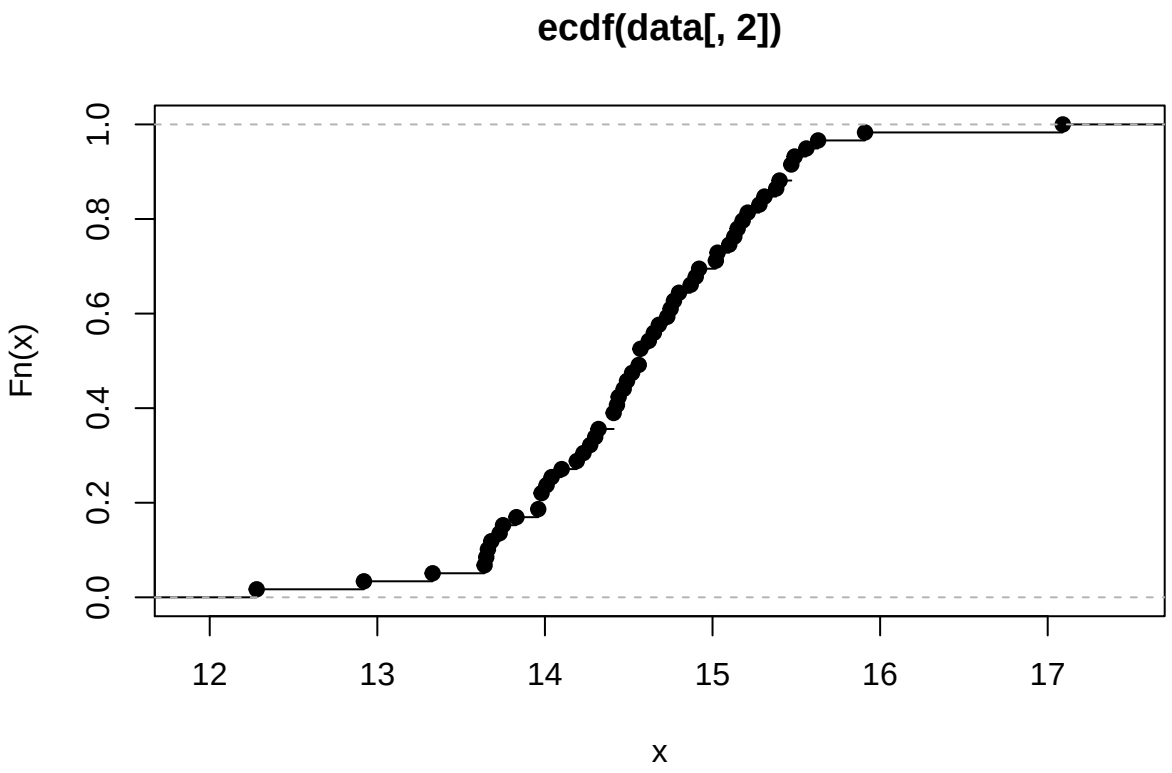
Jonathan Stuart

4/3/2018

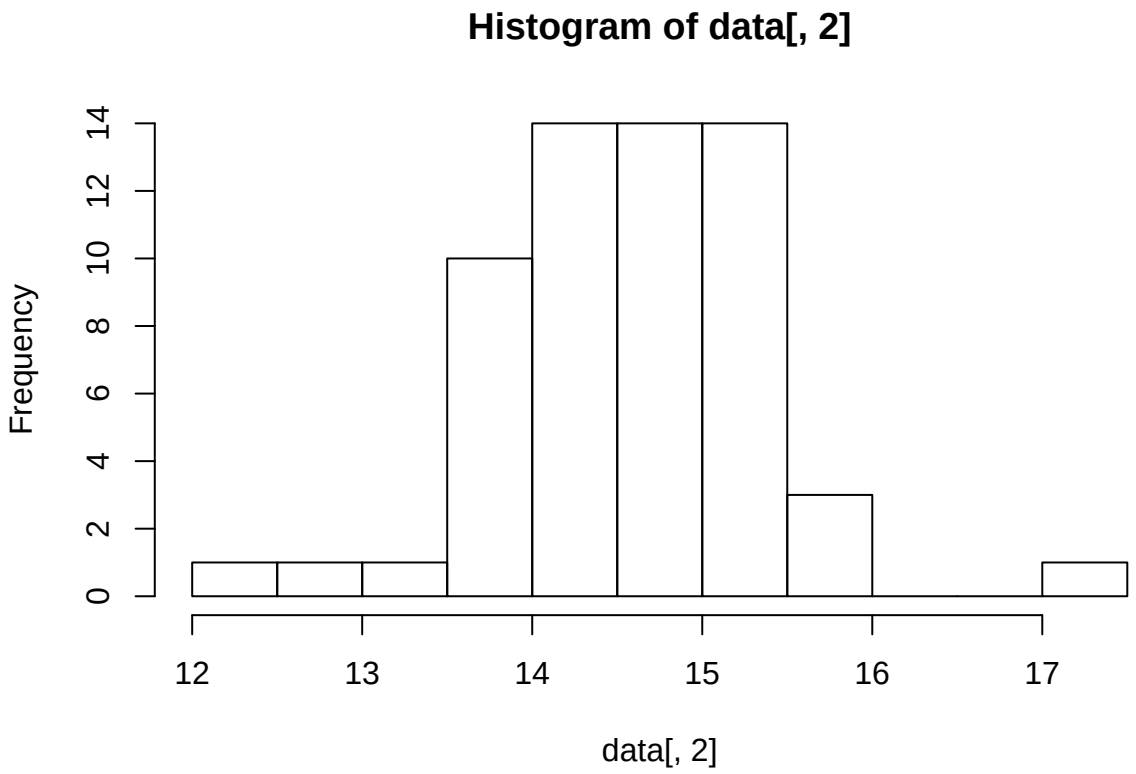
Problem 6

(a)

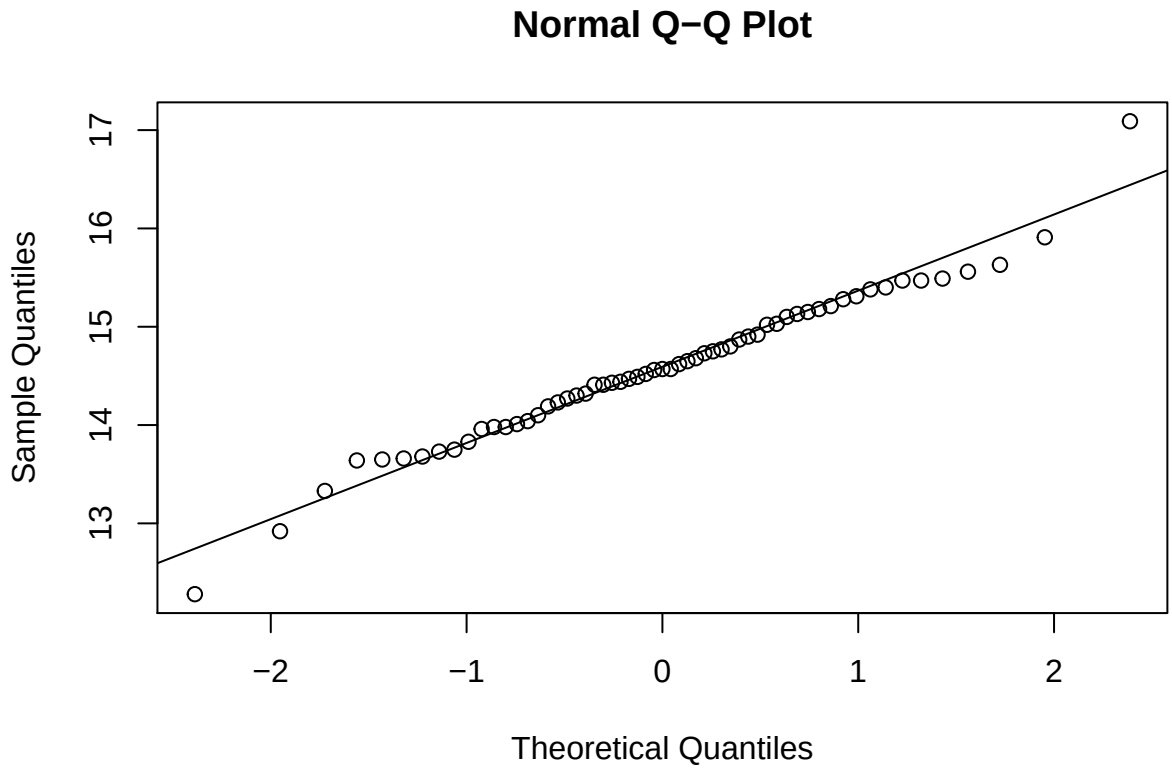
```
#plot the ecdf of hydrocarbon data
plot(ecdf(data[ , 2]))
```




```
#histogram of hydrocarbon data
hist(data[ , 2], breaks = 15)
```




```r
#normal probability plot of hydrocarbon data
qqnorm(data[ , 2])
qqline(data[ , 2])
```



```r
#quantiles of hydrocarbon data
quantiles <- quantile(data[ , 2], probs = seq(0, 1, 0.05))
quantiles[c(3, 6, 11, 16, 19)]
```

```
##    10%    25%    50%    75%    90%
## 13.676 14.070 14.570 15.115 15.470
```

Does the distribution appear Gaussian?
Yes, the distribution does appear Gaussian. We make this determination based on the fact that the QQ plot gives the data in almost straight line, along with visual confirmation from the histograms.

(b) *Will diluting the beeswax with microcrystalline wax be detectable?*
To answer this, let's consider weighted averages.

```r
#average percentage of hydrocarbons
avg_hydrocarbons <- mean(data[ , 2])

(0.99 * avg_hydrocarbons) + (0.01 * 0.85)
```

```
## [1] 14.4427
```

```r
(0.97 * avg_hydrocarbons) + (0.03 * 0.85)
```

```
## [1] 14.1681
```

```r
(0.95 * avg_hydrocarbons) + (0.05 * 0.85)
```

```
## [1] 13.8935
```

Assuming a normal distribution, we know that the standard deviation of the hydrocarbon data is

```r
sd(data[ , 2])
```

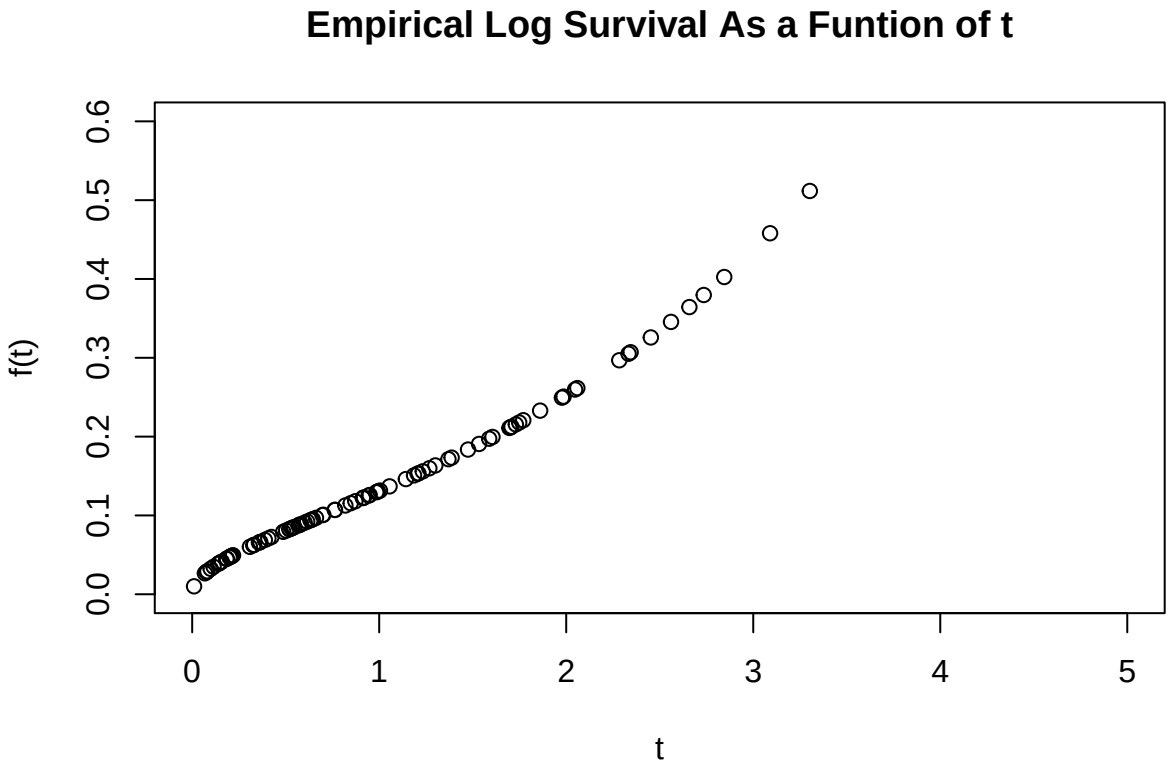
```
## [1] 0.7764197
```

This means that 1% and 3% dilutions would have negligible effects on the mean of the hydrcarbon data (with the 3% dilution having more of an impact than the 1% dilution), while a 5% dilution would change the mean by nearly a standard deviaton. A 5% dilution of the beeswax by the microcrystalline wax would therefore be quite noticeable.

Problem 8

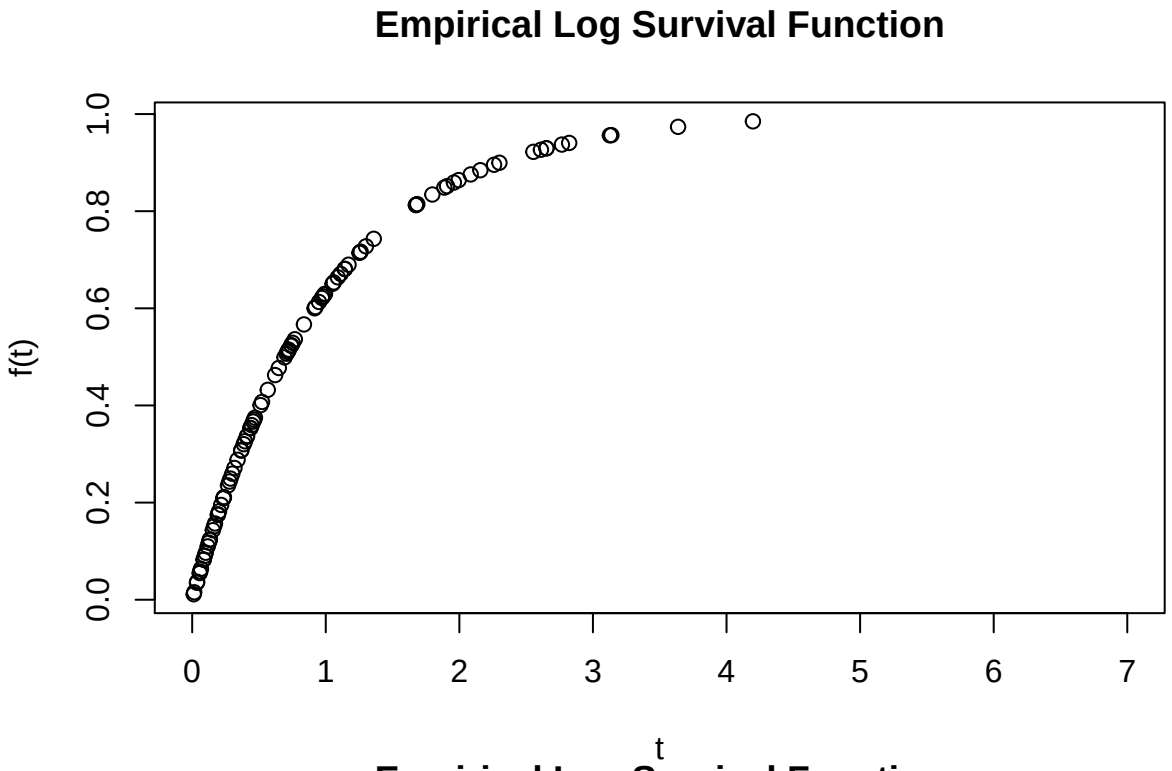
(a) We know that the survival function will be of the form $S(t) = 1 - F(t)$. We also know that the variability of the log survival funtion will be of the form $\frac{1}{n}\frac{F(t)}{1-F(t}$, and that the standard deviation of the empirical log survival function will be of the form $\sqrt{\frac{1}{n}\frac{F(t)}{1-F(t)}}$. For the exponential survival funtion, $F(t) = 1 - e^{(-t)}$.

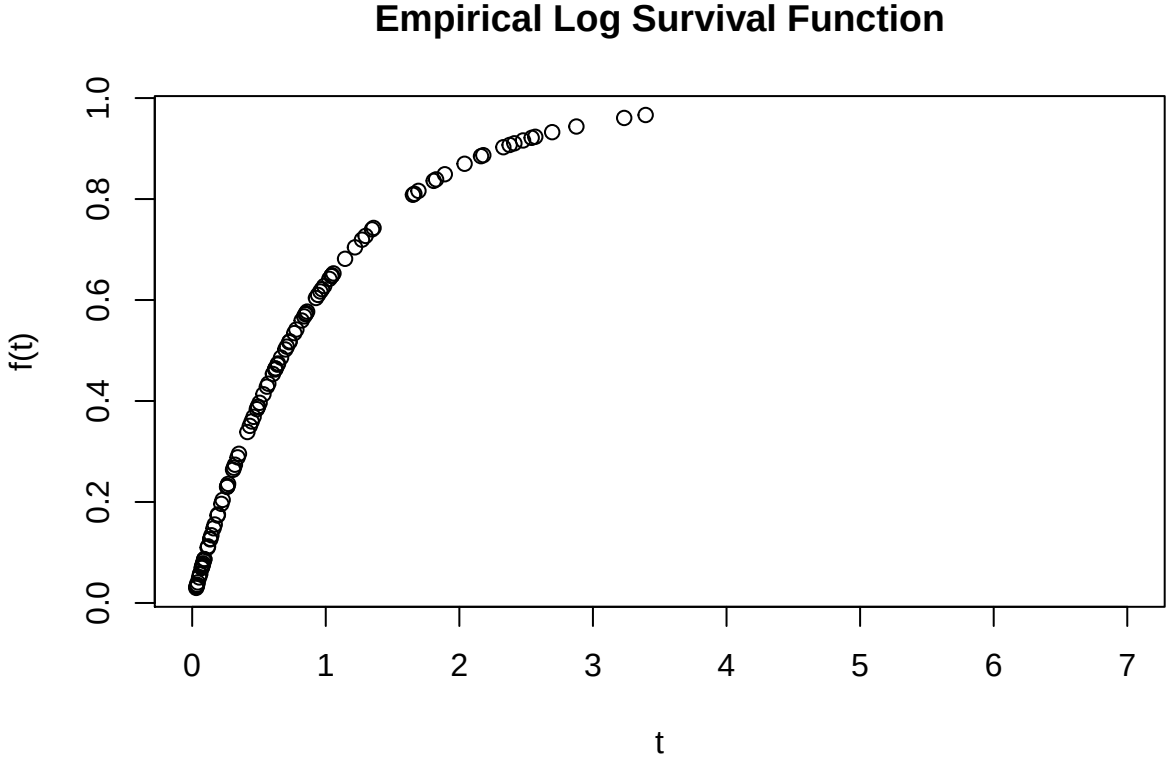
```r
#generating an exponential sample of size 100
set.seed(3)
exp_sample <- rexp(n = 100, rate = 1)

#empirical log survival function
sd_exp_sample <- sqrt((1/100) * ((1 - exp(-exp_sample)) / (exp(-exp_sample))))

plot(exp_sample, sd_exp_sample, xlim = c(0, 5), ylim = c(0, 0.6), xlab = "t", ylab = "f(t)", main = "Em
```

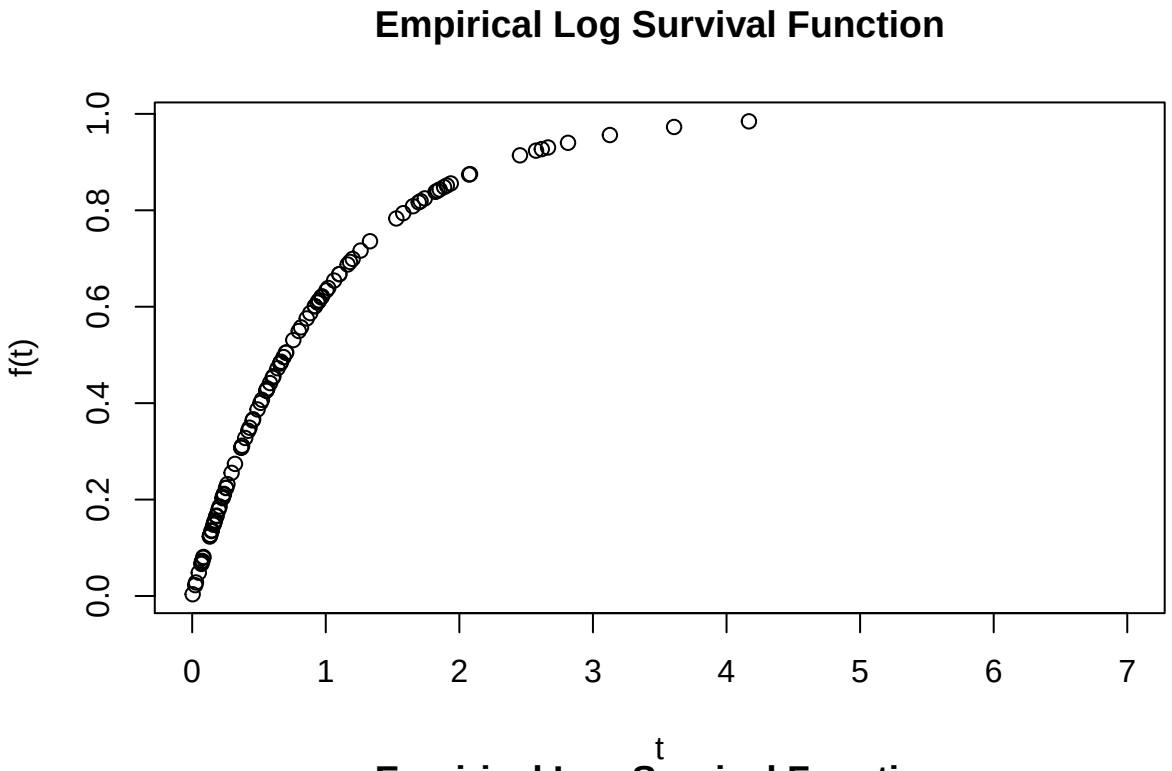


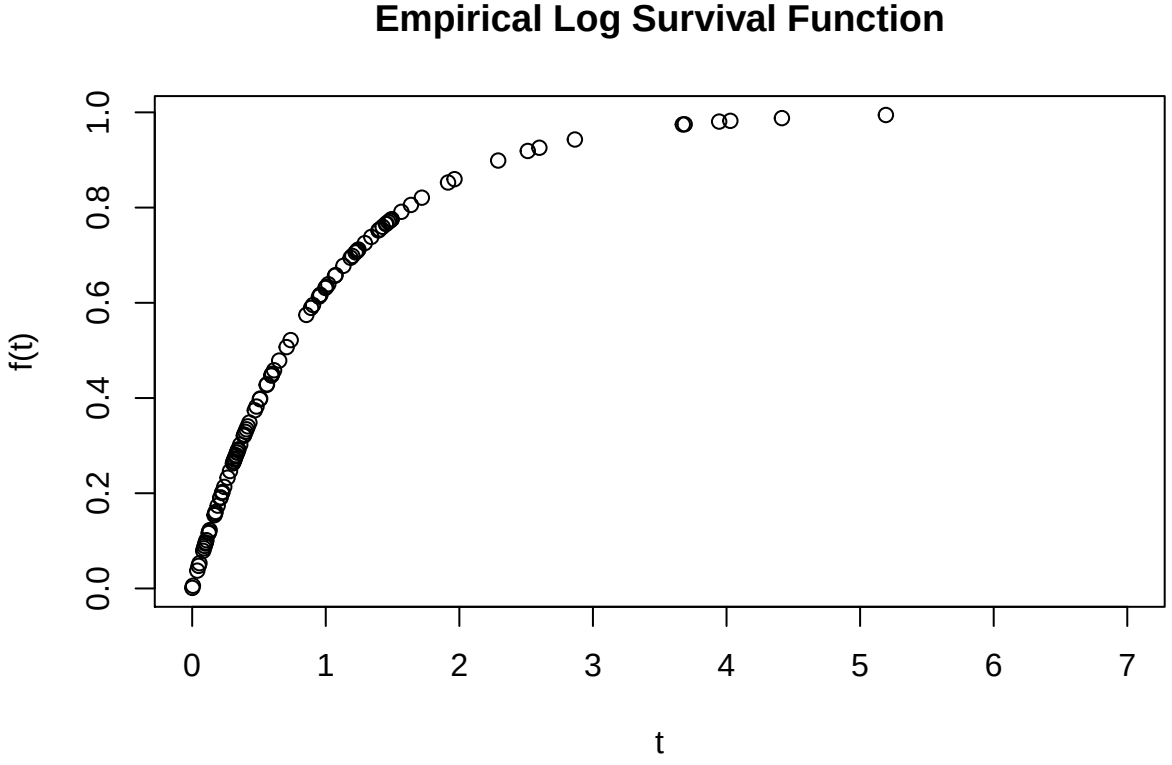
(b) *Generate several such samples of size 100 on a computer and for each sample plot the empirical log survival function. Relate the plots to your answer to (a).*

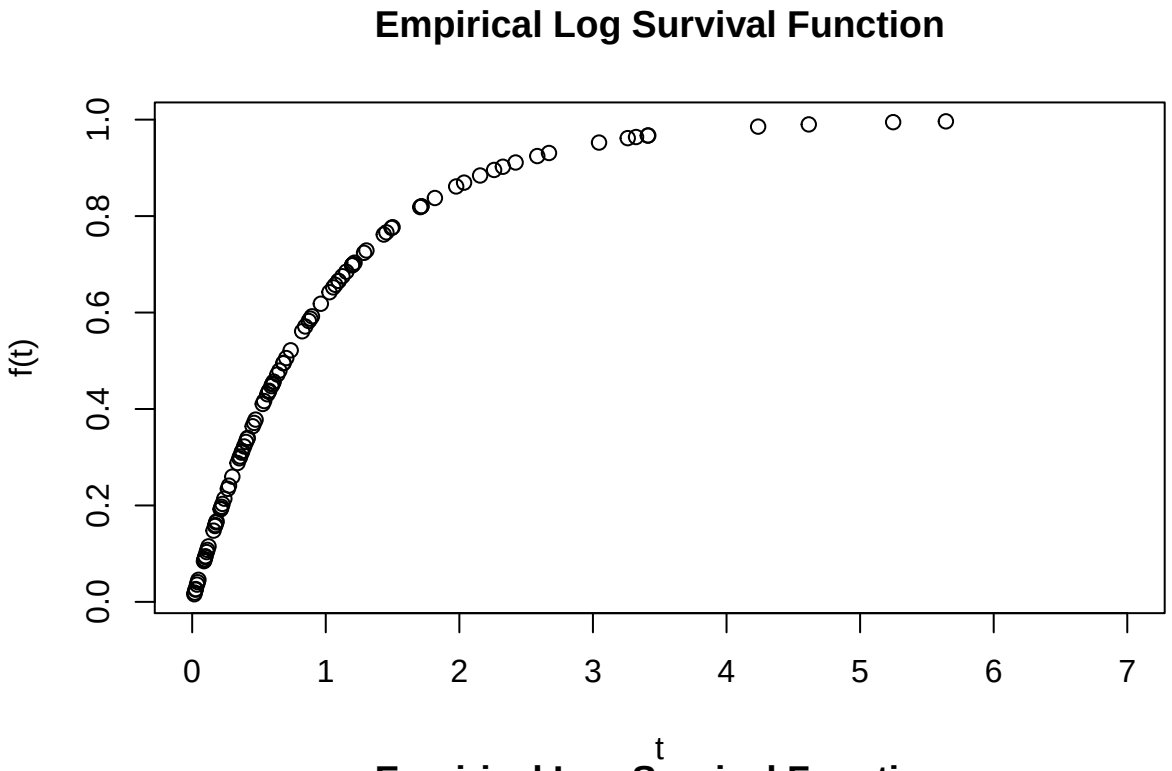
```r
exp_samples <- matrix(nrow = 100, ncol = 10)
#sd_exp_samples <- matrix(nrow = 100, ncol = 10)

#generating samples of size 100 from an exponential distribution
for (i in 1:10) {
  exp_samples[ , i] <- rexp(n = 100, rate = 1)
}

#plotting empirical log survival functions
for (i in 1:10) {
  plot(exp_samples[ , i], (1 - exp(-exp_samples[ , i])), xlim = c(0, 7), xlab = "t", ylab = "f(t)", mai
}
```

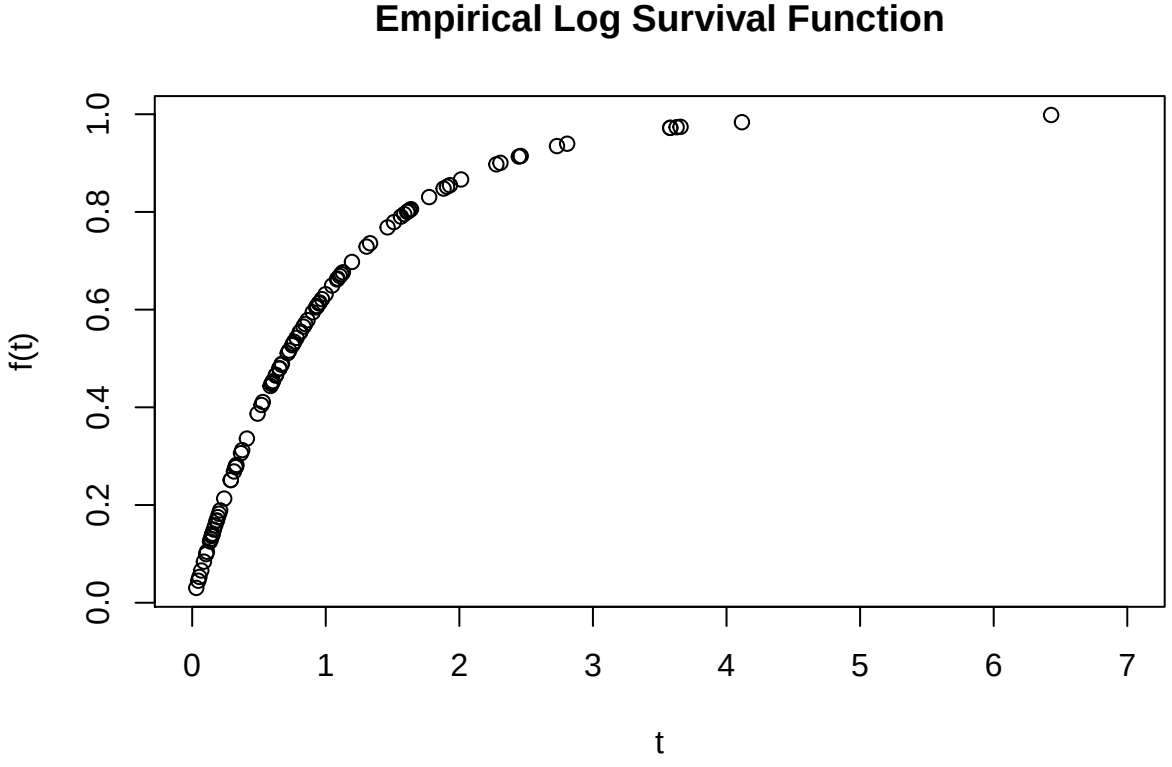

Empirical Log Survival Function
f(t)
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
t

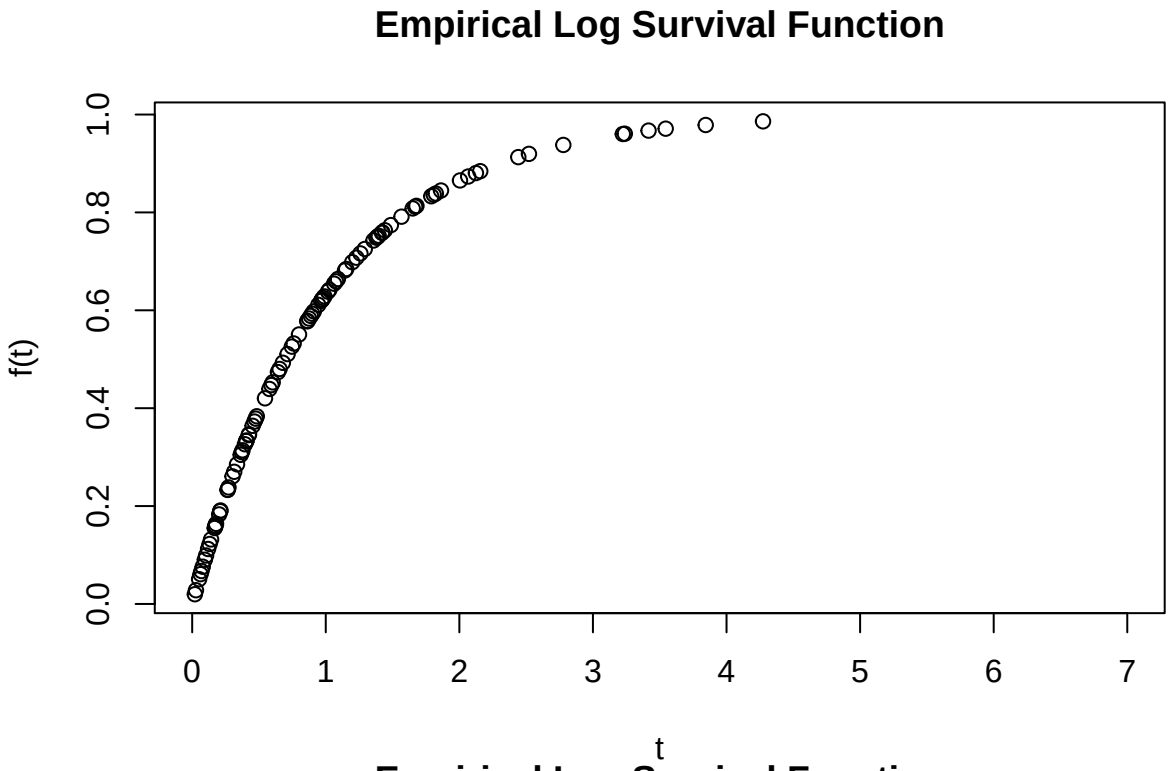

Empirical Log Survival Function
f(t)
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
t

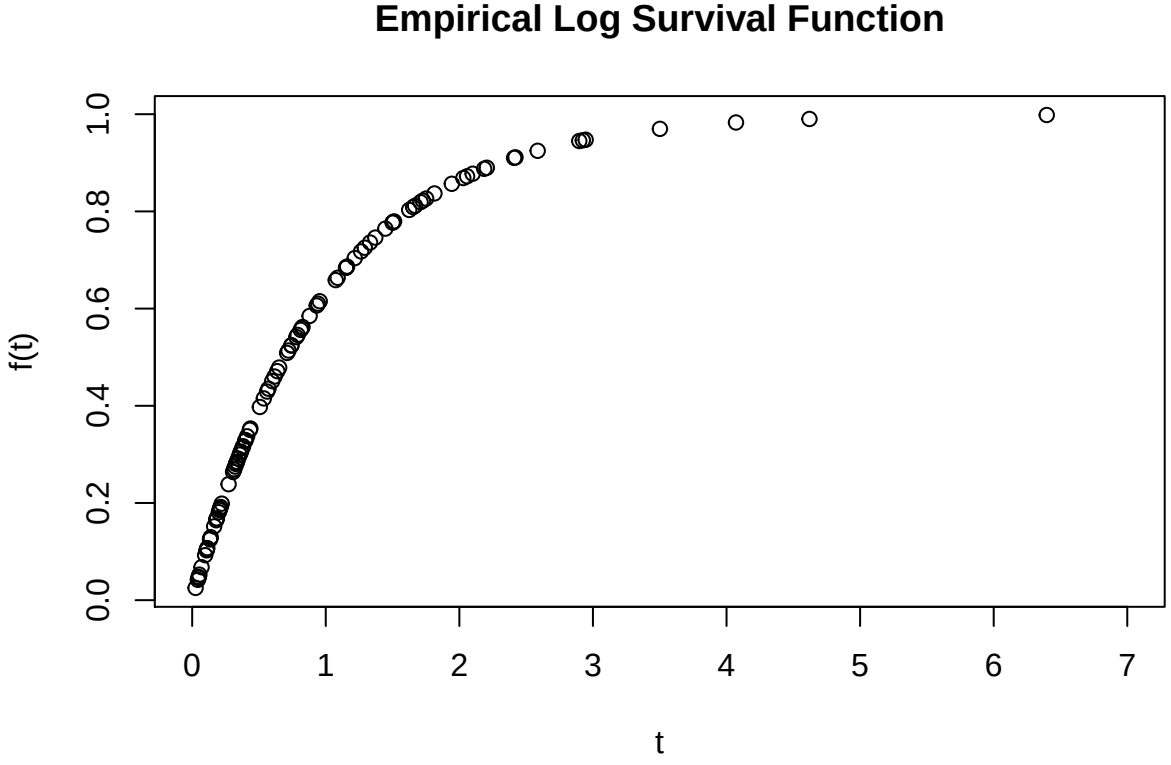

Empirical Log Survival Function
f(t)
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
t

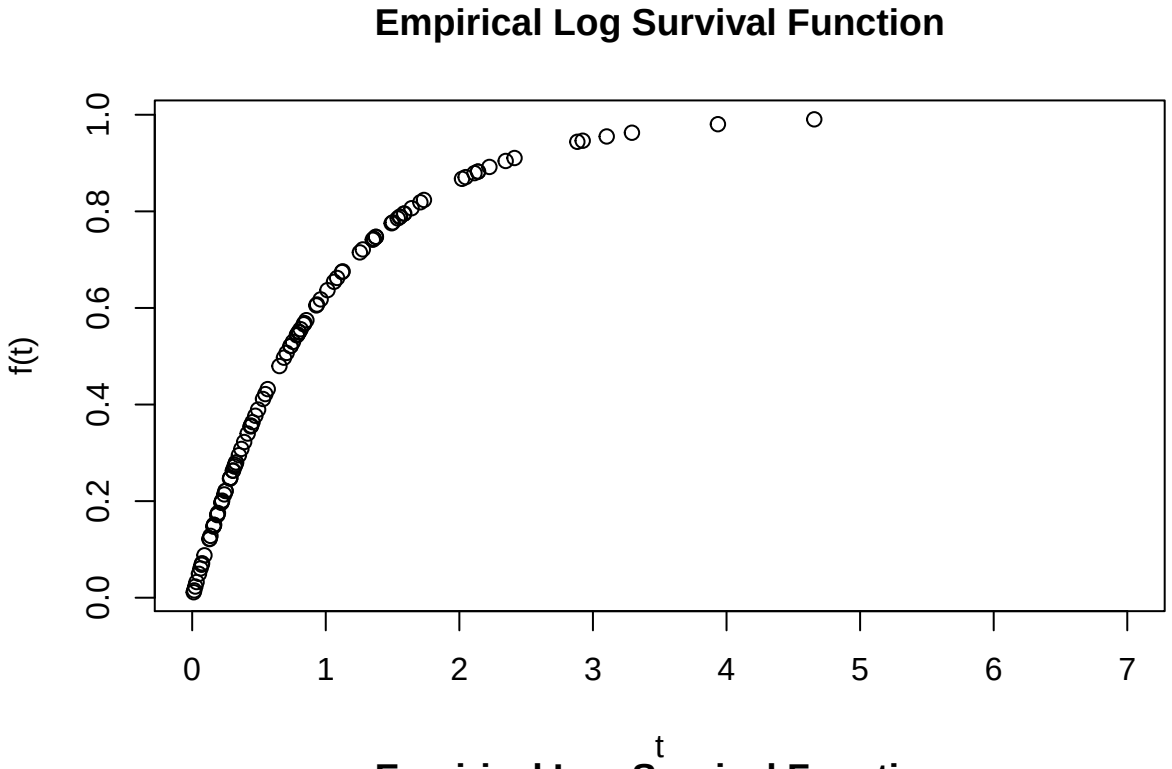

Empirical Log Survival Function
f(t)
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
t

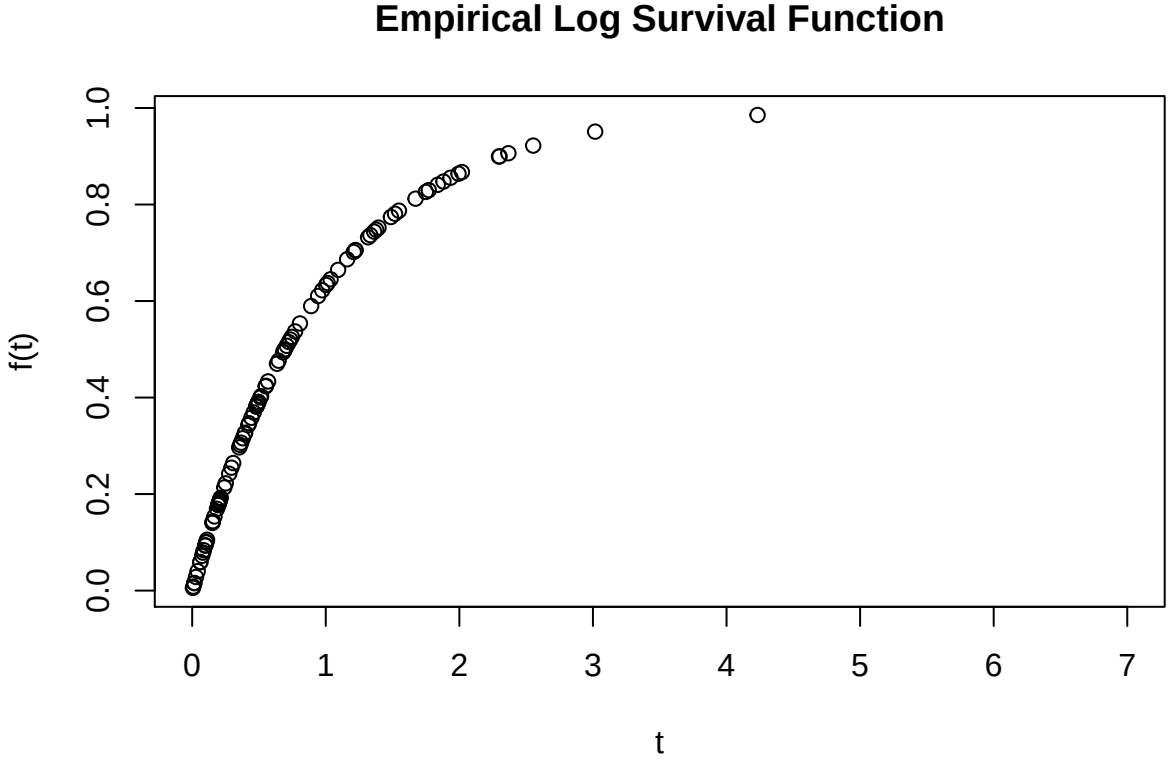

Empirical Log Survival Function
f(t)
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
t


Empirical Log Survival Function
f(t)
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
t


Empirical Log Survival Function
f(t)
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
t


Empirical Log Survival Function
f(t)
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
t





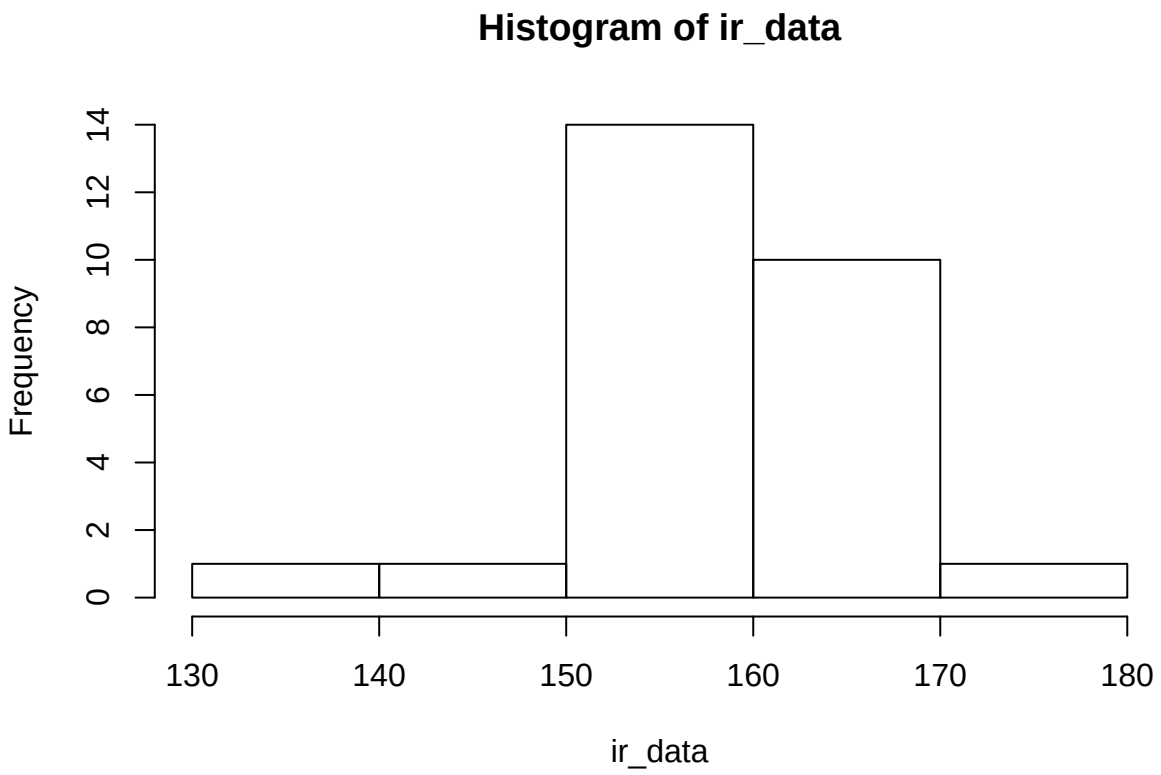
Here we have plots of the value of SDs on the empirical log survival function and of the survival function as t increase, t being values generated from the exponential distribution. We can see that as t increases, so do the values for the empirical log survival function as well as the survival function. We also notice that around $t = 2$, the function values begin showing more separation between then for nearly all the plots.

Problem 26

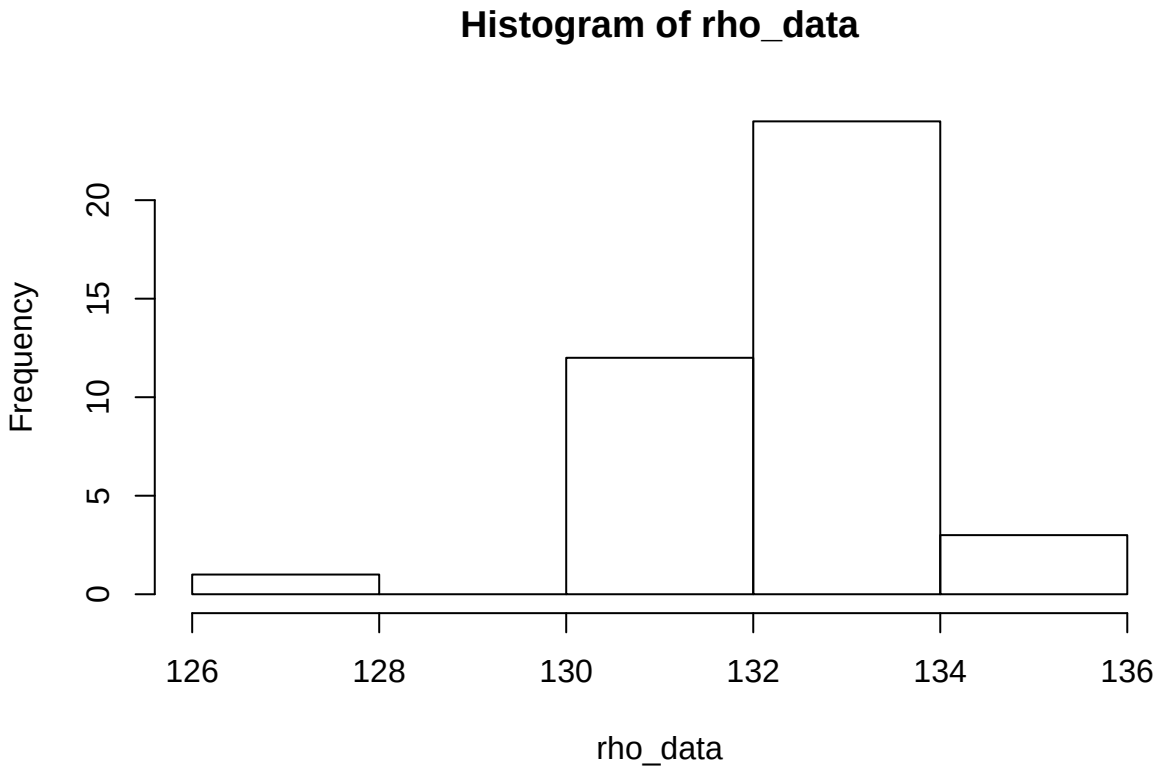
```
ir_data <- read.csv("iridium.csv", header = FALSE)
rho_data <- read.csv("rhodium.csv", header = FALSE)
```

(a)

```
#histograms
ir_data <- as.numeric(unlist(ir_data))
rho_data <- as.numeric(unlist(rho_data))


hist(ir_data, breaks = 5)
```




```
hist(rho_data, breaks = 5)
```



(b)

```
#stem and leaf plots
stem(ir_data)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   13 | 7
##   14 |
##   14 | 5
##   15 | 2
##   15 | 999
##   16 | 00000000000000001113
##   16 |
##   17 | 4
```

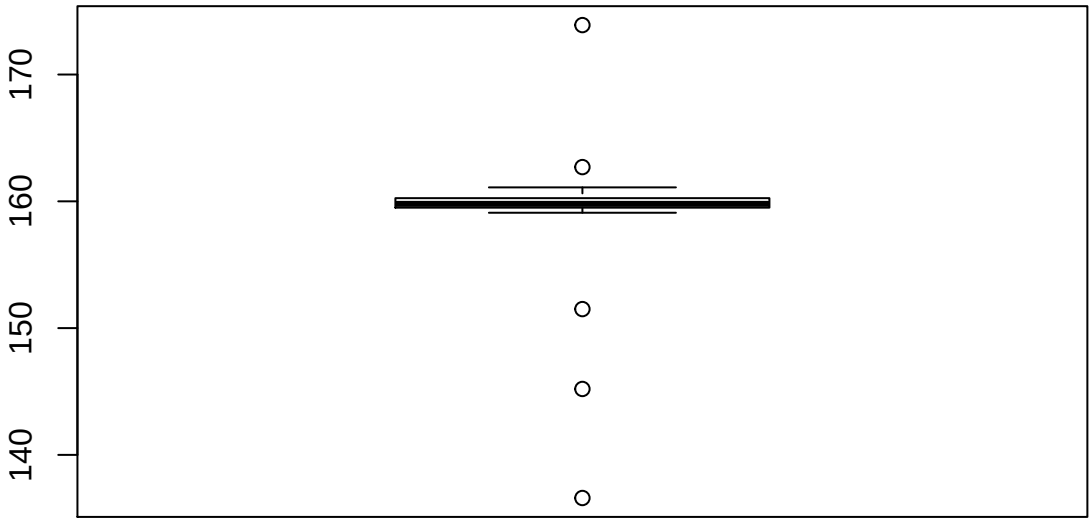
```
stem(rho_data)
```

```
##
##   The decimal point is at the |
##
##   126 | 4
##   127 |
##   128 |
##   129 |
##   130 |
##   131 | 111112234569
##   132 | 123456677899
##   133 | 000333455558
##   134 | 12
```

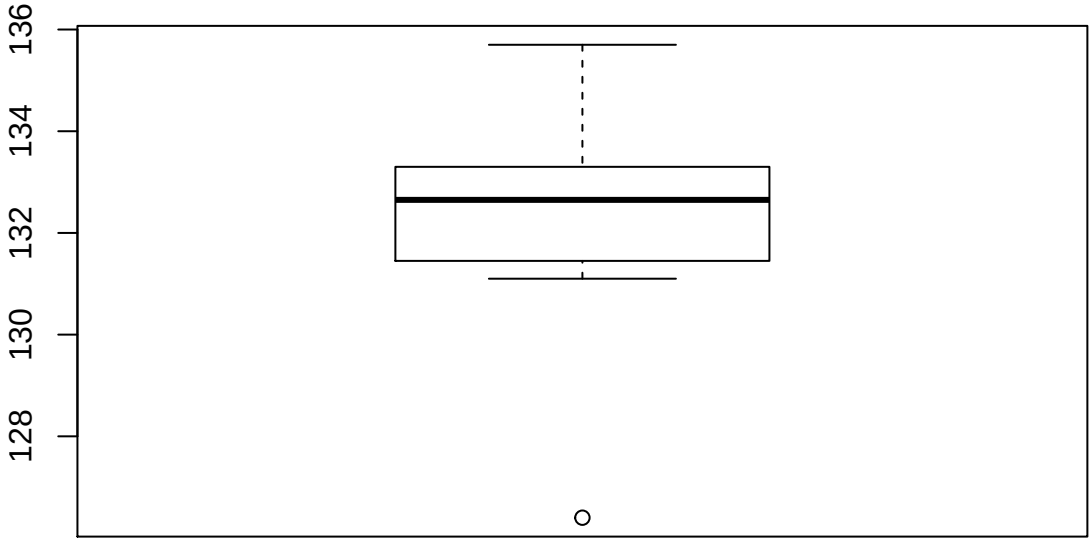
```
##   135 | 7
```

(c)

```
#boxplots
boxplot(ir_data)
```



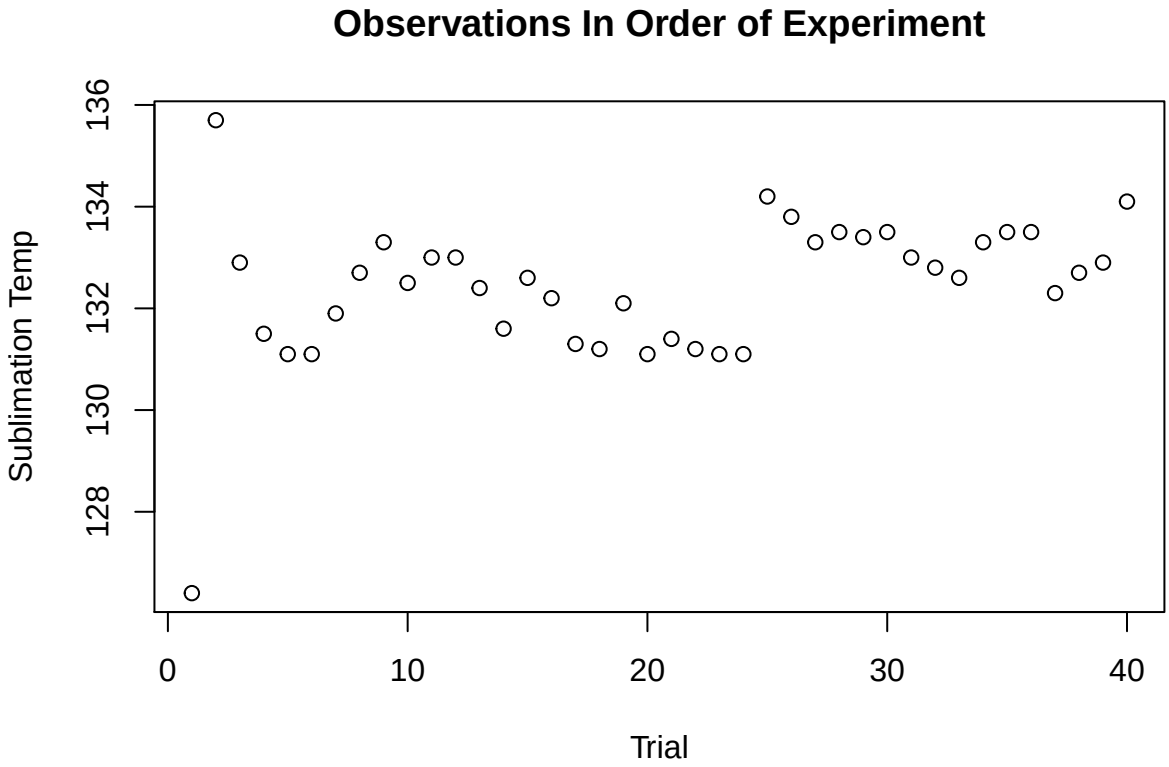

```
boxplot(rho_data)
```



(d)

```
#plots of data in the order of the experiment
plot(ir_data, xlab = "Trial", ylab = "Sublimation Temp", main = "Observations In Order of Experiment")
```


```
plot(rho_data, xlab = "Trial", ylab = "Sublimation Temp", main = "Observations In Order of Experiment")
```



(e) Since outliers can have significant impact on calculations, summaries and conclusions, this is an important question to ask. Looking at the iridium plot, we definitely see the presence of outlying values, indicating possible measurement error. The same is true for the rhodium plot, with at least 2 outlying values. Since the sublimation temperature is the object of investigation, each trial is in fact

independent.

(f) *Iridium*

```
#mean
mean(ir_data)
```

```
## [1] 158.8148
```

```
#median
median(ir_data)
```

```
## [1] 159.8
```

```
#10% trimmed mean
mean(ir_data, trim = 0.1)
```

```
## [1] 159.5478
```

```
#20% trimmed mean
mean(ir_data, trim = 0.2)
```

```
## [1] 159.8412
```

Rhodium

```
mean(rho_data)
```

```
## [1] 132.42
```

```
#median
median(rho_data)
```

```
## [1] 132.65
```

```
#10% trimmed mean
mean(rho_data, trim = 0.1)
```

```
## [1] 132.4781
```

```
#20% trimmed mean
mean(rho_data, trim = 0.2)
```

```
## [1] 132.5292
```

In comparing the iridium and rhodium means, medians and trimmed means, we see that as the percentage applied to the trimmed mean increases, the closer the value of the means comes to the median.

(g)

```
#standard error iridium data
se_iridium <- sd(ir_data) / sqrt(27)

#standard error rhodium data
se_rhodium <- sd(rho_data) / sqrt(40)

#upper bound of CI for iridium
mean(ir_data) + (1.645 * se_iridium)
```

```
## [1] 160.7854
```

```
#lower bound of CI for iridium
mean(ir_data) - (1.645 * se_iridium)
```

```
## [1] 156.8442
```

```
#upper bound of CI for rhodium
mean(rho_data) + (1.645 * se_rhodium)
```

```
## [1] 132.794
```

```
#lower bound of CI for rhodium
mean(rho_data) - (1.645 * se_rhodium)
```

```
## [1] 132.046
```

So the 90% confidence interval for iridium would be (156.84, 160.79), and the 90% confidence interval for rhodium would be (132.05, 132.79).

(h) We know that this interval will be of the for (X_k, X_{n-k+1}). We also know that for $k = 9$, $p = (P(Y < k)) = 0.0610$, and that $2p = .1221$. This gets us very close to a 90% confidence interval. So from this all we know that $k = 10$ and that $n - k + 1$ is $27 - 10 + 1 = 18$. Sorting the iridum data gives

```
sort(ir_data)
```

```
##  [1] 136.6 145.2 151.5 159.1 159.2 159.3 159.5 159.5 159.5 159.5 159.6
## [12] 159.6 159.7 159.8 160.0 160.0 160.1 160.1 160.2 160.2 160.3 160.4
## [23] 160.6 160.8 161.1 162.7 173.9
```

From this we determine that a 90% confidence interval for iridium is (159.5, 160.1). Carrying out the same process for rhodium yields a confidence interval of (132.2, 133) where $k = 17$ and $n - k + 1$ is $40 - 15 + 1 = 26$.

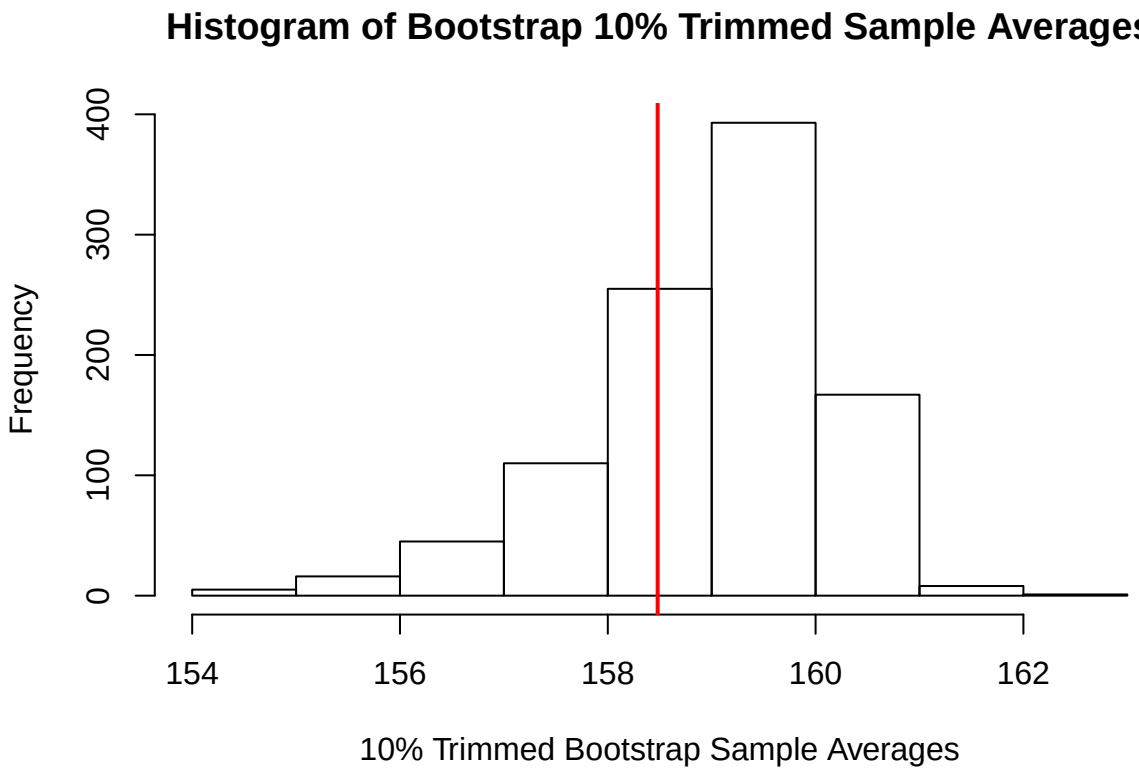
(i) *Iridium*

```
#constructing the bootstrap population
vals = sort(unique(ir_data))
counts = table(ir_data)
# makes the bootstrap pop as rounded version of sample, not quite right
boot_pop <- rep(vals, 100)
length(boot_pop)
```

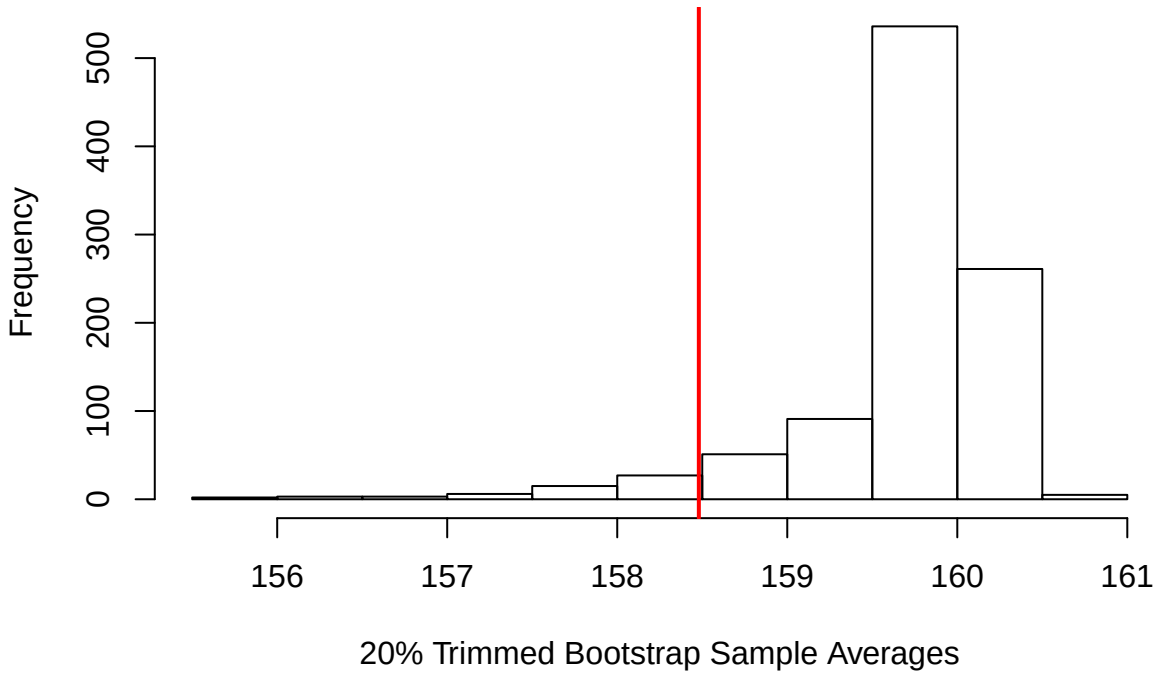
```
## [1] 2000
```

```
#sampling from the bootstrap population
boot_pop_sample <- replicate(1000, sample(boot_pop, length(ir_data), FALSE))

# boot_pop_sample is now a matrix of 27 rows (the number of temps per sample),
# and 1000 columns (the total number of samples).

#calculating the sample averages of the bootstrap samples
boot_x_bars_10 <- 0
for (i in 1:1000) {
  boot_x_bars_10[i] <- mean(boot_pop_sample[ , i], trim = 0.1)
}

boot_x_bars_20 <- 0
for (i in 1:1000) {
  boot_x_bars_20[i] <- mean(boot_pop_sample[ , i], trim = 0.2)
}

#histogram of 10% trimmed sample averages with vertical line at bootstrap pop. average
hist(boot_x_bars_10, main = "Histogram of Bootstrap 10% Trimmed Sample Averages",
     xlab = "10% Trimmed Bootstrap Sample Averages")
```

```r
boot_pop_mean <- mean(boot_pop)
abline(v=boot_pop_mean,col="red", lwd = 2)
```




```r
#histogram of 20% trimmed sample averages with vertical line at bootstrap pop. average
hist(boot_x_bars_20, main = "Histogram of Bootstrap 20% Trimmed Sample Averages",
     xlab = "20% Trimmed Bootstrap Sample Averages")
boot_pop_mean <- mean(boot_pop)
abline(v=boot_pop_mean,col="red", lwd = 2)
```



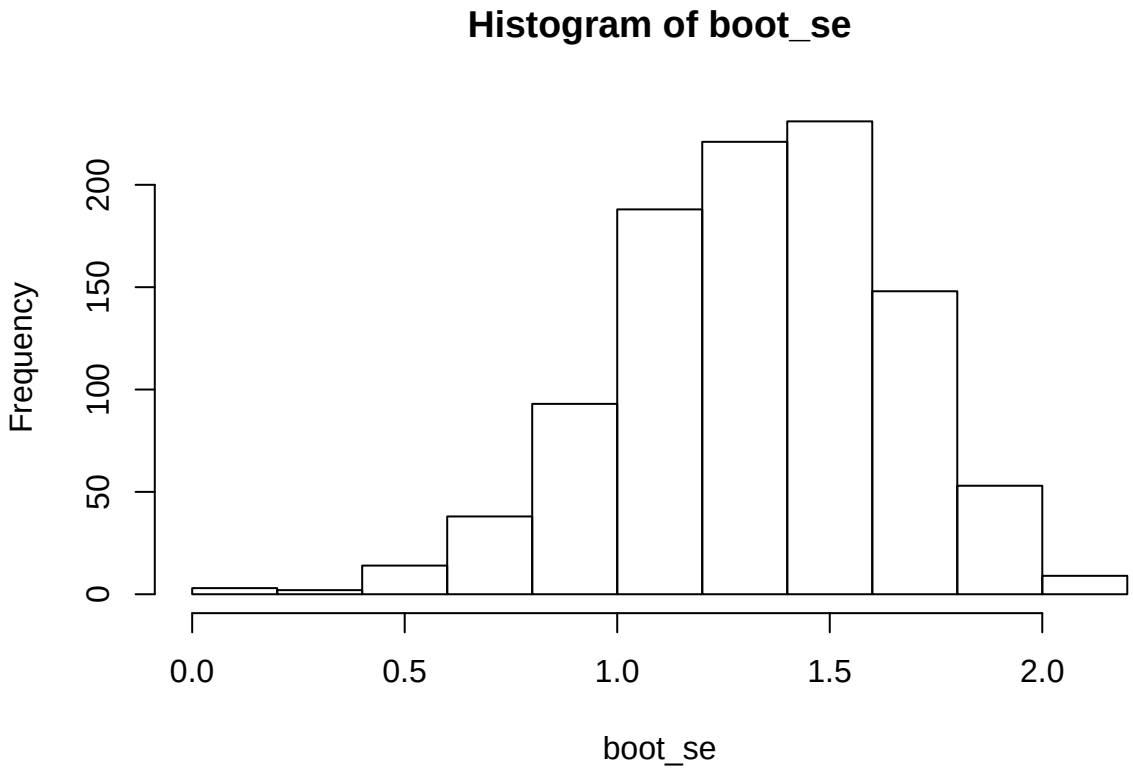

```
#standard errors
sd(boot_x_bars_10) / sqrt(1000)
```

```
## [1] 0.03675395
```

```
sd(boot_x_bars_20) / sqrt(1000)
```

```
## [1] 0.0192098
```

```
#calculating standard errors
boot_se <- 0
for (i in 1:1000) {
  boot_se[i] <- (sd(boot_pop_sample[ , i]) / sqrt(27))
}

hist(boot_se)
```



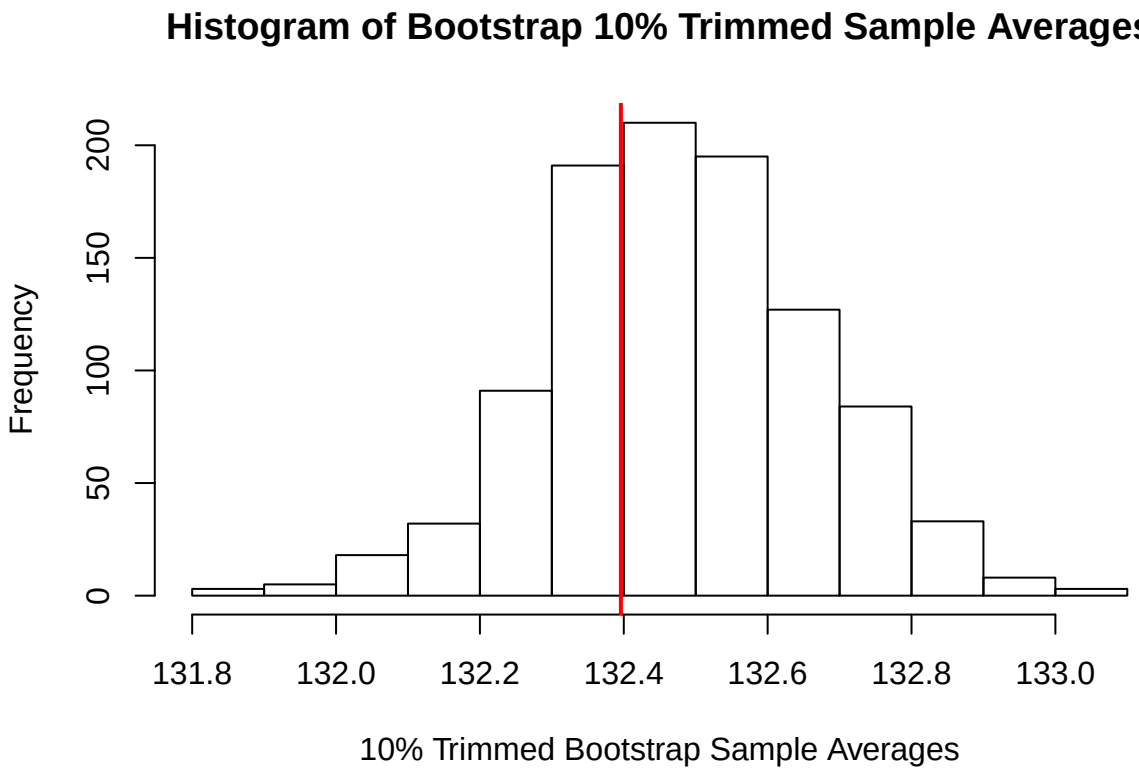
Rhodium

```
#constructing the bootstrap population
vals = sort(unique(rho_data))
counts = table(rho_data)
# makes the bootstrap pop as rounded version of sample, not quite right
boot_pop <- rep(vals, 100)
length(boot_pop)
```

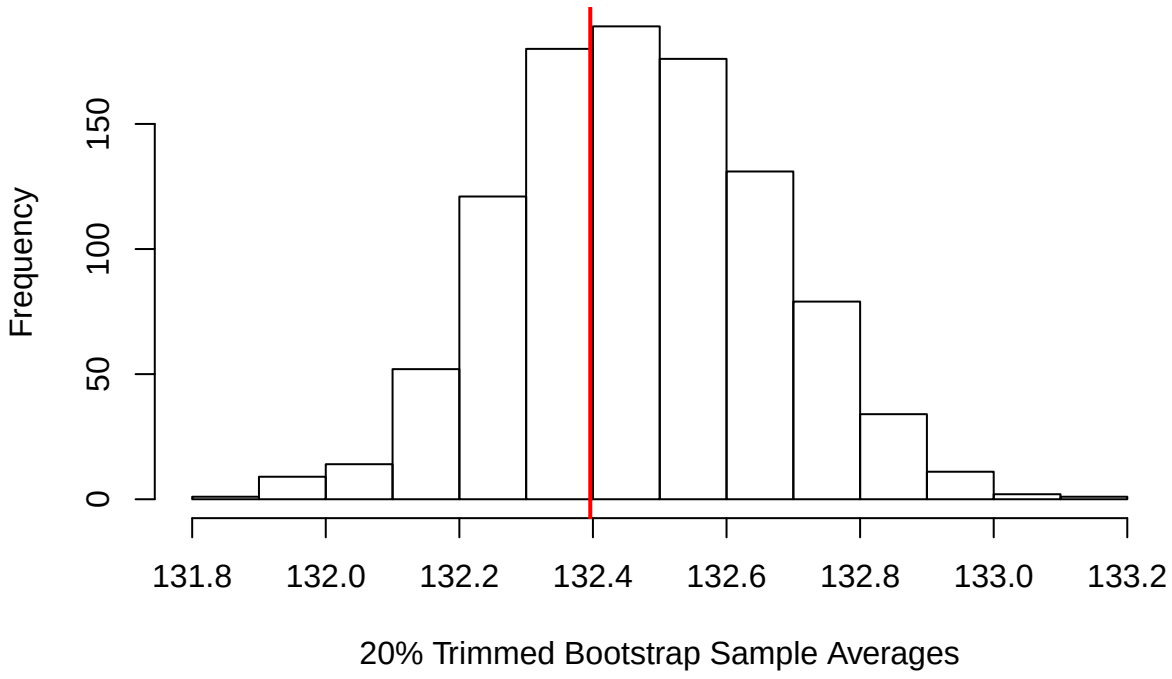
```
## [1] 2500
```

```
#sampling from the bootstrap population
boot_pop_sample <- replicate(1000, sample(boot_pop, length(rho_data), FALSE))

# boot_pop_sample is now a matrix of 40 rows (the number of temps per sample),
# and 1000 columns (the total number of samples).

#calculating the sample averages of the bootstrap samples
boot_x_bars_10 <- 0
for (i in 1:1000) {
  boot_x_bars_10[i] <- mean(boot_pop_sample[ , i], trim = 0.1)
}

boot_x_bars_20 <- 0
for (i in 1:1000) {
  boot_x_bars_20[i] <- mean(boot_pop_sample[ , i], trim = 0.2)
}

#histogram of 10% trimmed sample averages with vertical line at bootstrap pop. average
hist(boot_x_bars_10, main = "Histogram of Bootstrap 10% Trimmed Sample Averages",
     xlab = "10% Trimmed Bootstrap Sample Averages")
```

```r
boot_pop_mean <- mean(boot_pop)
abline(v=boot_pop_mean,col="red", lwd = 2)
```




```r
#histogram of 20% trimmed sample averages with vertical line at bootstrap pop. average
hist(boot_x_bars_20, main = "Histogram of Bootstrap 20% Trimmed Sample Averages",
     xlab = "20% Trimmed Bootstrap Sample Averages")
boot_pop_mean <- mean(boot_pop)
abline(v=boot_pop_mean,col="red", lwd = 2)
```



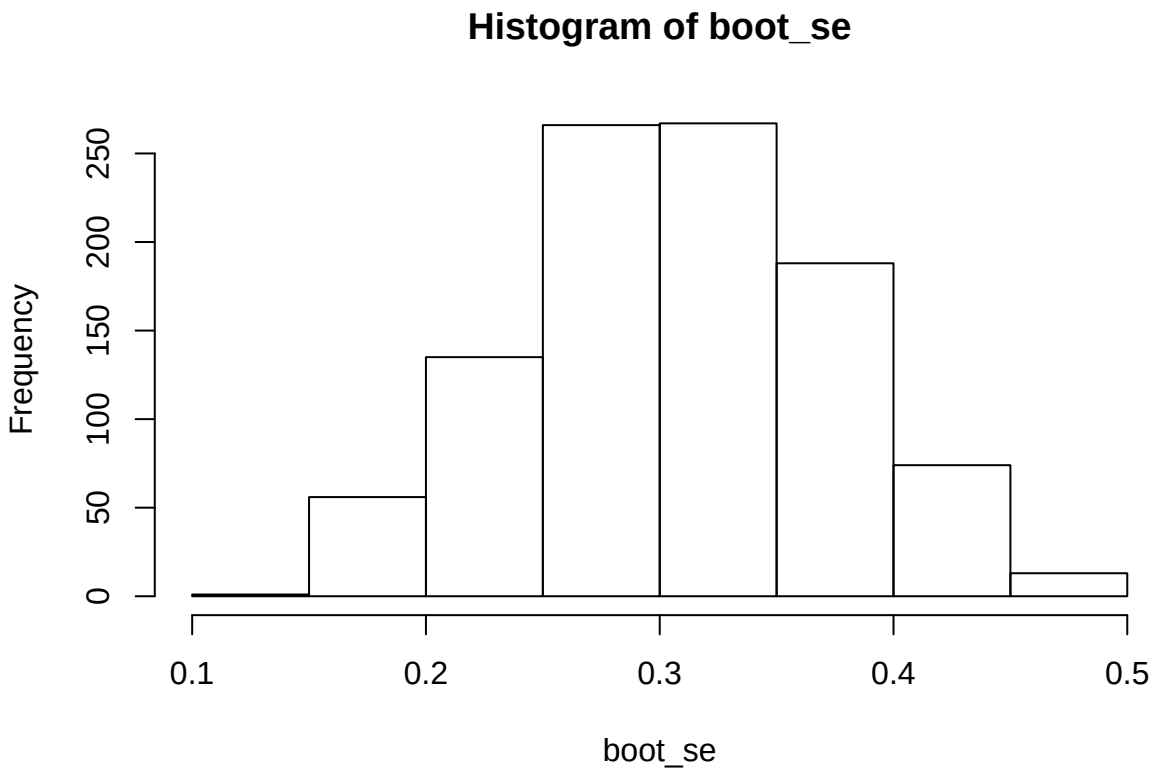

```
#standard errors
sd(boot_x_bars_10) / sqrt(1000)
```

```
## [1] 0.005909574
```

```
sd(boot_x_bars_20) / sqrt(1000)
```

```
## [1] 0.006221628
```

```
#calculating standard errors
boot_se <- 0
for (i in 1:1000) {
  boot_se[i] <- (sd(boot_pop_sample[ , i]) / sqrt(27))
}

hist(boot_se)
```



From the plots above, we see that the rhodium data appears to be more normally distributed than the iridium data, while the standard errors of the sampling distributions are very close to each other in the rhodium data, but less so in the iridium data.

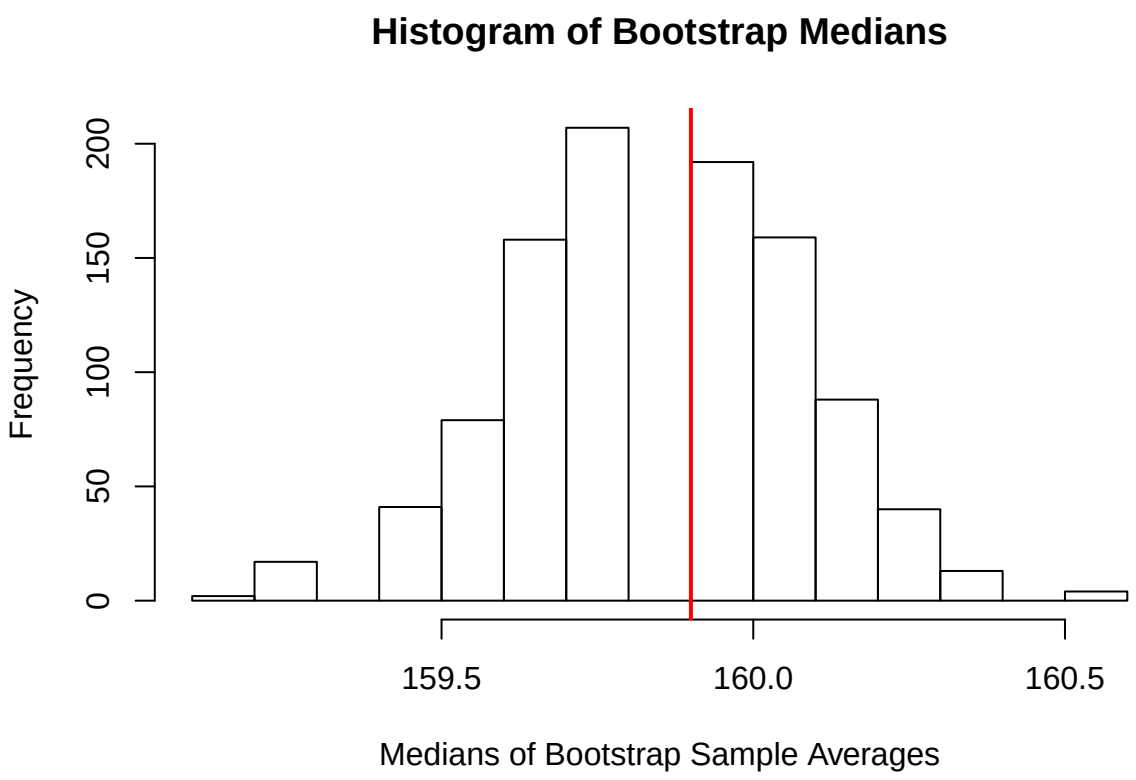
(j) *Iridium*

```
#constructing the bootstrap population
vals = sort(unique(ir_data))
counts = table(ir_data)
# makes the bootstrap pop as rounded version of sample, not quite right
boot_pop <- rep(vals, 100)
length(boot_pop)
```

```
## [1] 2000
```

```
#sampling from the bootstrap population
boot_pop_sample <- replicate(1000, sample(boot_pop, length(ir_data), FALSE))

# boot_pop_sample is now a matrix of 27 rows (the number of temps per sample),
# and 1000 columns (the total number of samples).

#calculating the sample averages of the bootstrap samples
boot_medians <- 0
for (i in 1:1000) {
  boot_medians[i] <- median(boot_pop_sample[ , i])
}


#histogram of 10% trimmed sample averages with vertical line at bootstrap pop. average
hist(boot_medians, main = "Histogram of Bootstrap Medians",
     xlab = "Medians of Bootstrap Sample Averages")
```

```
boot_pop_median <- median(boot_pop)
abline(v=boot_pop_median,col="red", lwd = 2)
```




```
#standard error
sd(boot_medians) / sqrt(1000)
```

```
## [1] 0.007561429
```

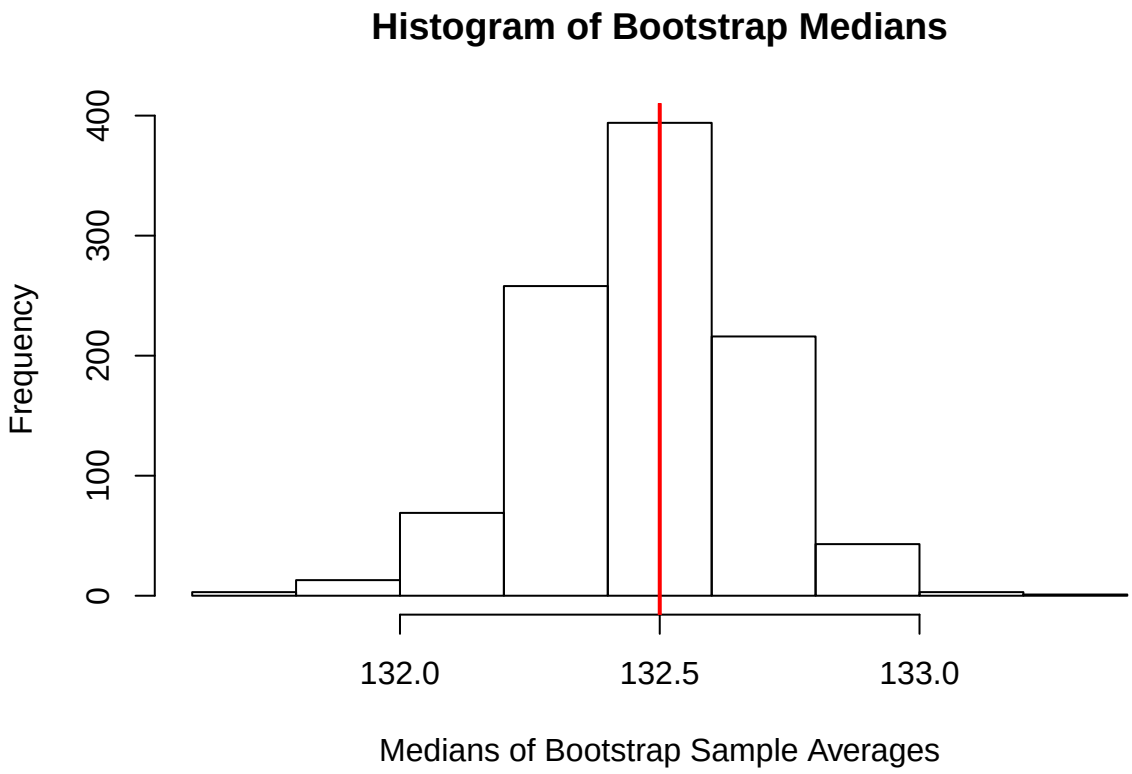
Rhodium

```
#constructing the bootstrap population
vals = sort(unique(rho_data))
counts = table(rho_data)
# makes the bootstrap pop as rounded version of sample, not quite right
boot_pop <- rep(vals, 100)
length(boot_pop)
```

```
## [1] 2500
```

```
#sampling from the bootstrap population
boot_pop_sample <- replicate(1000, sample(boot_pop, length(rho_data), FALSE))

# boot_pop_sample is now a matrix of 40 rows (the number of temps per sample),
# and 1000 columns (the total number of samples).

#calculating the sample averages of the bootstrap samples
boot_medians <- 0
for (i in 1:1000) {
  boot_medians[i] <- median(boot_pop_sample[ , i])
}
```

```r
#histogram of 10% trimmed sample averages with vertical line at bootstrap pop. average
hist(boot_medians, main = "Histogram of Bootstrap Medians",
     xlab = "Medians of Bootstrap Sample Averages")
boot_pop_median <- median(boot_pop)
abline(v=boot_pop_median,col="red", lwd = 2)
```




```r
#standard error
sd(boot_medians) / sqrt(1000)
```

```
## [1] 0.006359654
```

Notably, the histograms of the bootstrap medians for both sets of data appear to be normally distributed and centered or nearly centered on the median value of the original data.

(k)

```r
#standard error iridium data
se_iridium <- sd(ir_data) / sqrt(27)

#standard error rhodium data
se_rhodium <- sd(rho_data) / sqrt(40)

##10% trimmed
#upper bound of CI for iridium
mean(ir_data, trim = 0.1) + (1.645 * se_iridium)
```

```
## [1] 161.5184
```

```r
#lower bound of CI for iridium
mean(ir_data, trim = 0.1) - (1.645 * se_iridium)
```

```
## [1] 157.5773
```

```r
#upper bound of CI for rhodium
mean(rho_data, trim = 0.1) + (1.645 * se_rhodium)
```

```
## [1] 132.8521
```

```r
#lower bound of CI for rhodium
mean(rho_data, trim = 0.1) - (1.645 * se_rhodium)
```

```
## [1] 132.1042
```

```r
##20% trimmed
mean(ir_data, trim = 0.2) + (1.645 * se_iridium)
```

```
## [1] 161.8118
```

```r
#lower bound of CI for iridium
mean(ir_data, trim = 0.2) - (1.645 * se_iridium)
```

```
## [1] 157.8706
```

```r
#upper bound of CI for rhodium
mean(rho_data, trim = 0.2) + (1.645 * se_rhodium)
```

```
## [1] 132.9031
```

```r
#lower bound of CI for rhodium
mean(rho_data, trim = 0.2) - (1.645 * se_rhodium)
```

```
## [1] 132.1552
```

The 90% confidence interval for iridium would be:
10% trimmed mean: (157.58, 161.52)
20% trimmed mean: (157.87, 161.81)

The 90% confidence interval for rhodium would be:
10% trimmed mean: (132.10, 132.85)
20% trimmed mean: (132.26, 132.90)

Given that the 90% confidence intervals for iridium and rhodium were (156.84, 160.79) and (132.05, 132.79), respectively, we have that the trimmed means effected the iridium data more than the rhodium data when constructing confidence intervals. This makes sense given that the iridium data had more extreme outliers.