EEB 201 Bootcamp Assignments

Shawn Schwartz

9/19/2019

Noa Pinter-Wollman (Day 1)

Iris Exercise

1. Take a look at the 'iris' data set

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
tail(iris)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

2a) Assign data for the setosa species to one variable named 'setosa' and assign the subset of the data for the virginica species to another variable named 'virginica'.

```
iris_ds <- iris
setosa <- iris_ds %>%
  filter(Species == "setosa")

head(setosa)
```

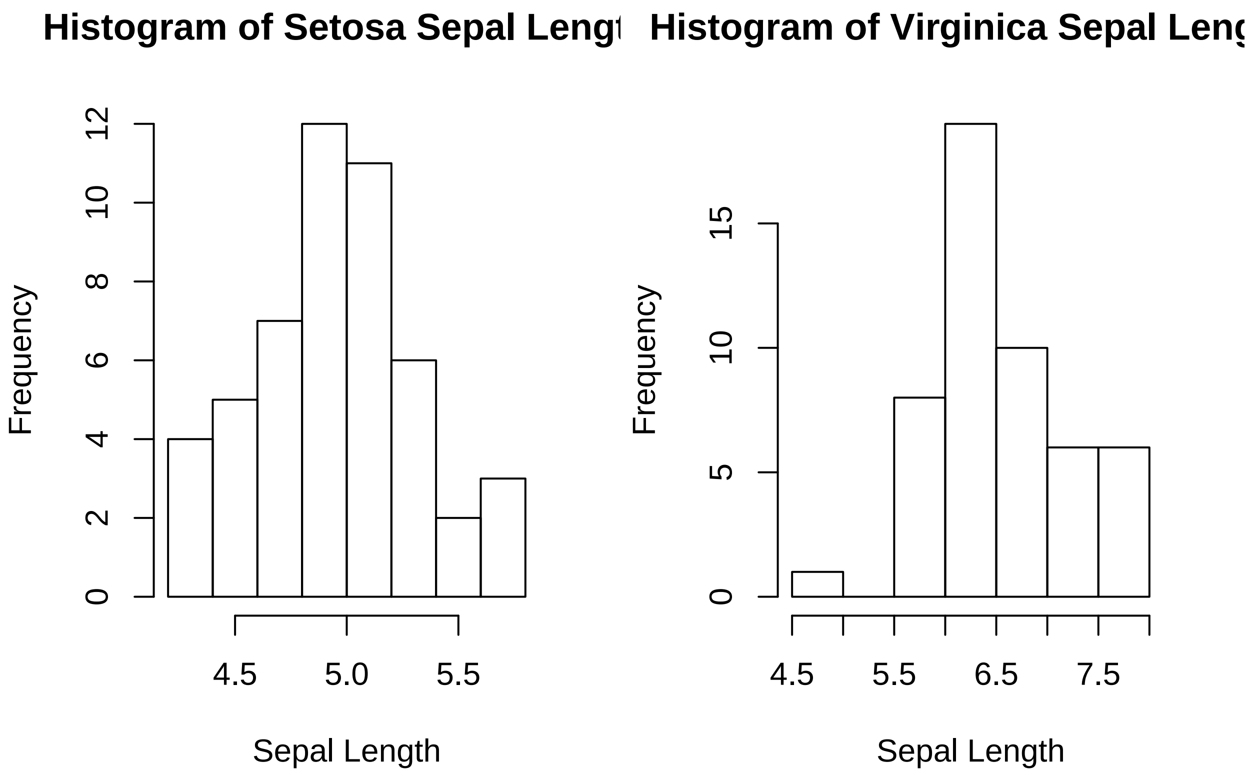
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
virginica <- iris_ds %>%
  filter(Species == "virginica")

head(virginica)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          6.3         3.3          6.0         2.5 virginica
## 2          5.8          2.7          5.1          1.9 virginica
## 3          7.1          3.0          5.9          2.1 virginica
## 4          6.3          2.9          5.6          1.8 virginica
## 5          6.5          3.0          5.8          2.2 virginica
## 6          7.6          3.0          6.6          2.1 virginica
```

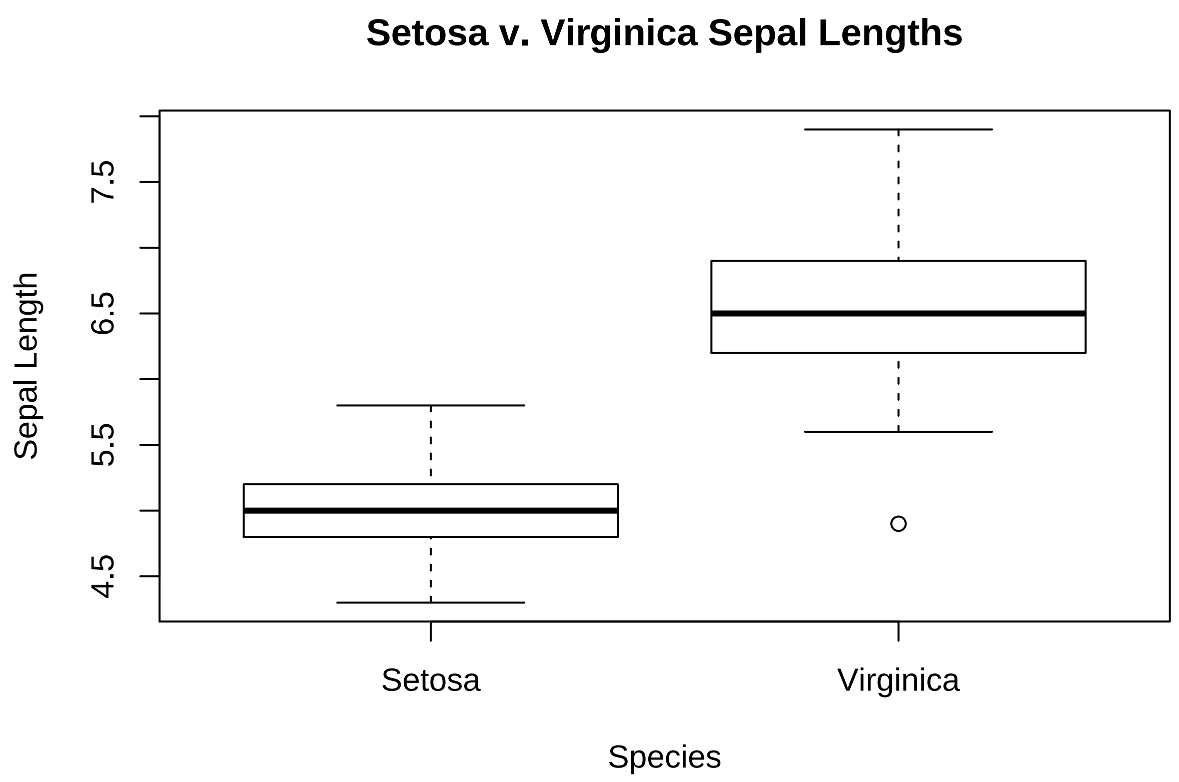
2b) Plot the sepal lengths of the two species as two histograms.

```
par(mfrow=c(1,2))
hist(setosa$Sepal.Length, main = "Histogram of Setosa Sepal Length", xlab = "Sepal Length")
hist(virginica$Sepal.Length, main = "Histogram of Virginica Sepal Length", xlab = "Sepal Length")
```



2c) Now plot the sepal lengths of the two species so that you can compare them.

```
par(mfrow=c(1,1))
boxplot(setosa$Sepal.Length, virginica$Sepal.Length, names = c("Setosa", "Virginica"), main = "Setosa v
```



2d) Is there a statistically significant difference between the sepal lengths of the two species?

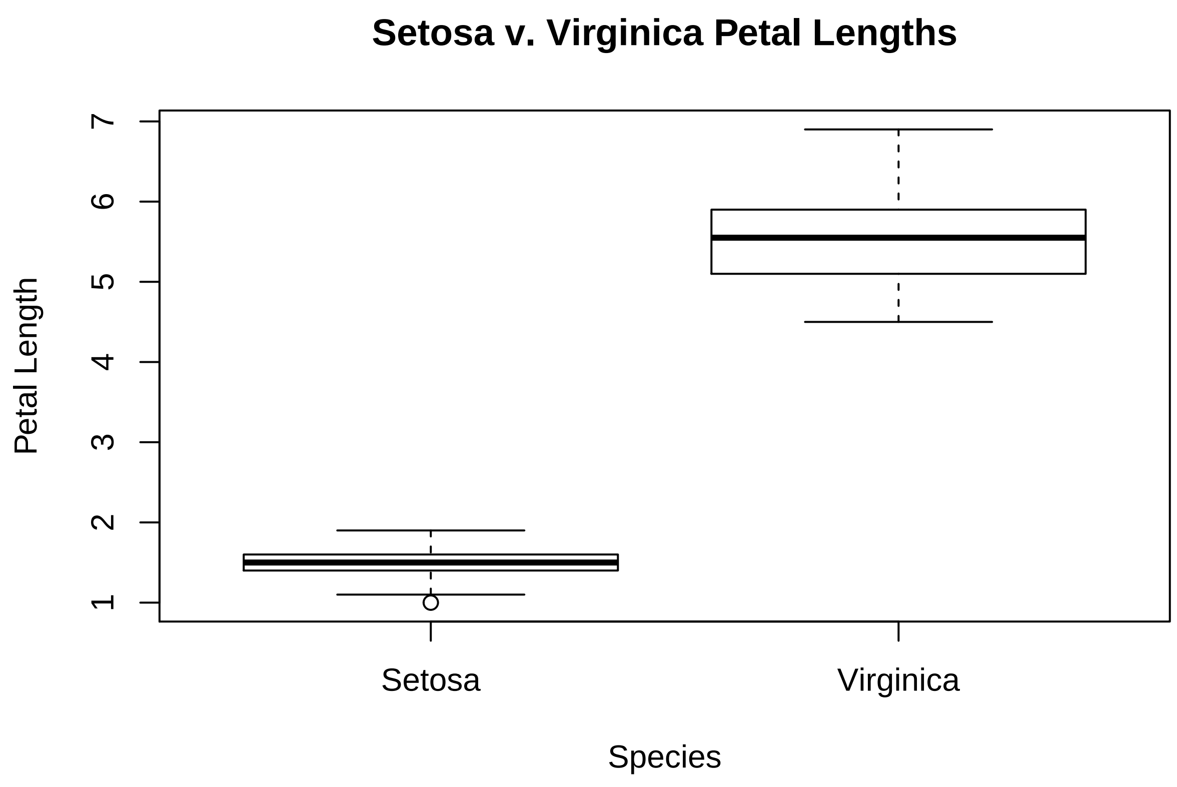
```
sepalLengthttest <- t.test(setosa$Sepal.Length, virginica$Sepal.Length)
sepalLengthttest
```

```
##
##  Welch Two Sample t-test
##
## data:  setosa$Sepal.Length and virginica$Sepal.Length
## t = -15.386, df = 76.516, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.78676 -1.37724
## sample estimates:
## mean of x mean of y
##     5.006     6.588
```

Yes, there is a statistically significant difference between the sepal lengths of the two species.

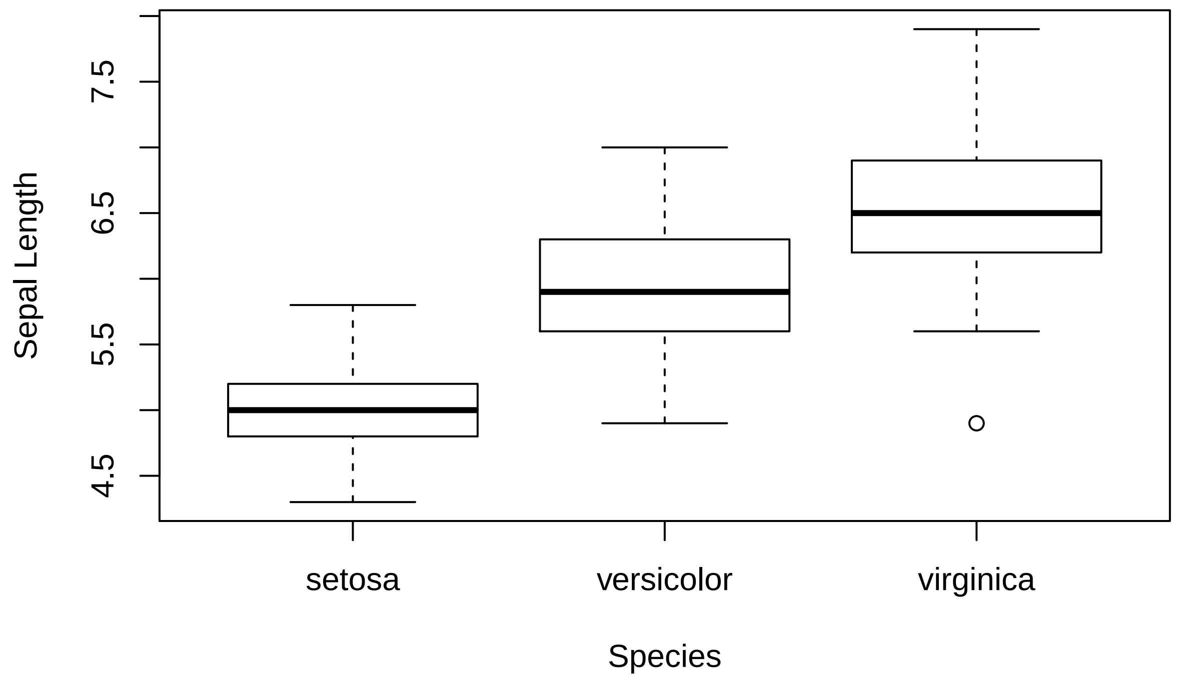
3) Plot and compare two other measures on your own.

```
boxplot(setosa$Petal.Length, virginica$Petal.Length, names = c("Setosa", "Virginica"), main = "Setosa v
```



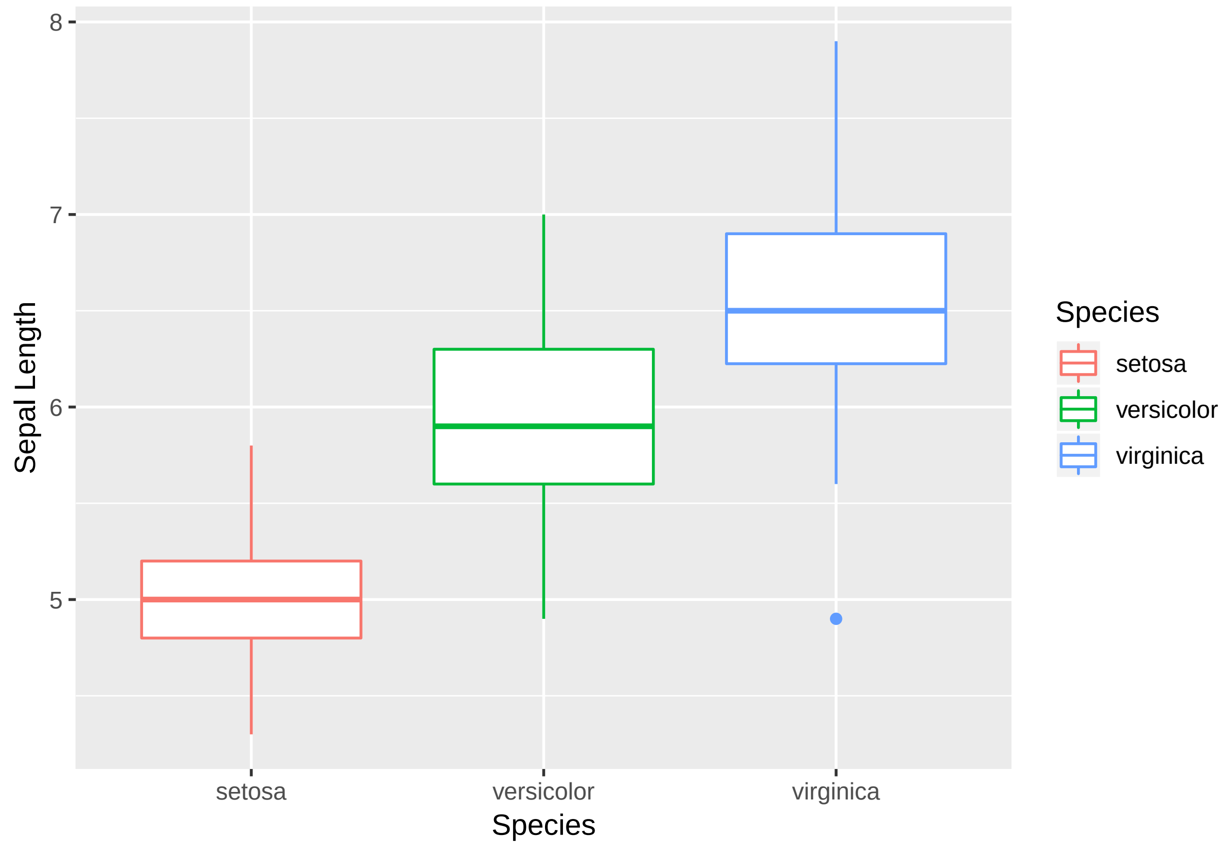
4) Compare the means of more than two groups. 4a) Plot the sepal lengths of all three species.

```r
plot(iris_ds$Species, iris_ds$Sepal.Length, xlab = "Species", ylab="Sepal Length")
```



4b) Add a legend with species names to your plot.

```r
iris_sepals <- ggplot(iris_ds, aes(x=Species, y=Sepal.Length, col = Species)) +
  geom_boxplot() +
  xlab("Species") +
  ylab("Sepal Length")

iris_sepals
```



4c) Is there a statisically significant difference in sepal length among the three species? (Hint: use an ANOVA)

```
aov.sepallength.out <- aov(iris_ds$Sepal.Length ~ iris_ds$Species)
summary(aov.sepallength.out)
```

```
##                  Df Sum Sq Mean Sq F value Pr(>F)    
## iris_ds$Species   2  63.21  31.606   119.3 <2e-16 ***
## Residuals       147  38.96   0.265                   
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Yes, there is a statisically significant difference in sepal length among the three species.

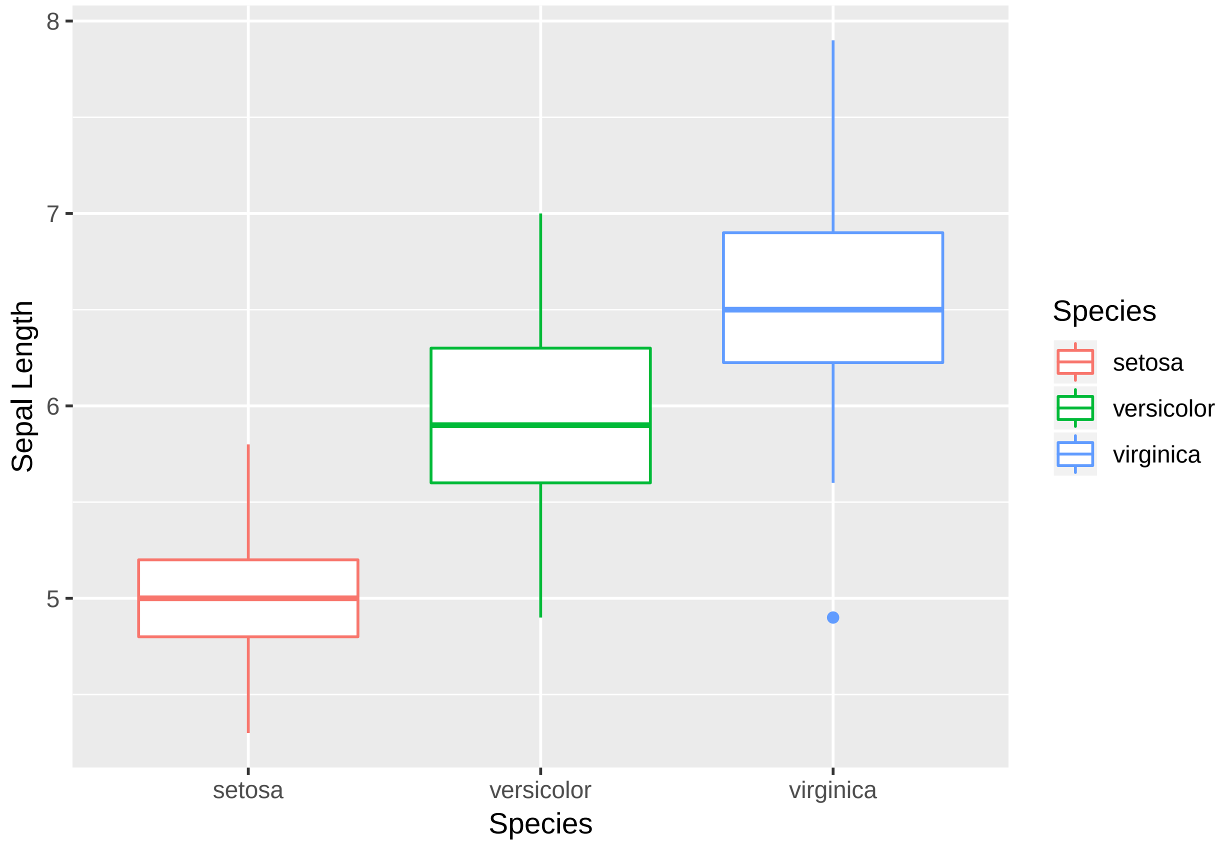
4d) Which species are significantly different (Hint: run a post-hoc test)

```
aov.sepallength.out.tukey <- TukeyHSD(aov.sepallength.out)
aov.sepallength.out.tukey
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = iris_ds$Sepal.Length ~ iris_ds$Species)
##
## $`iris_ds$Species`
##                       diff       lwr       upr p adj
## versicolor-setosa    0.930 0.6862273 1.1737727     0
## virginica-setosa     1.582 1.3382273 1.8257727     0
## virginica-versicolor 0.652 0.4082273 0.8957727     0
```

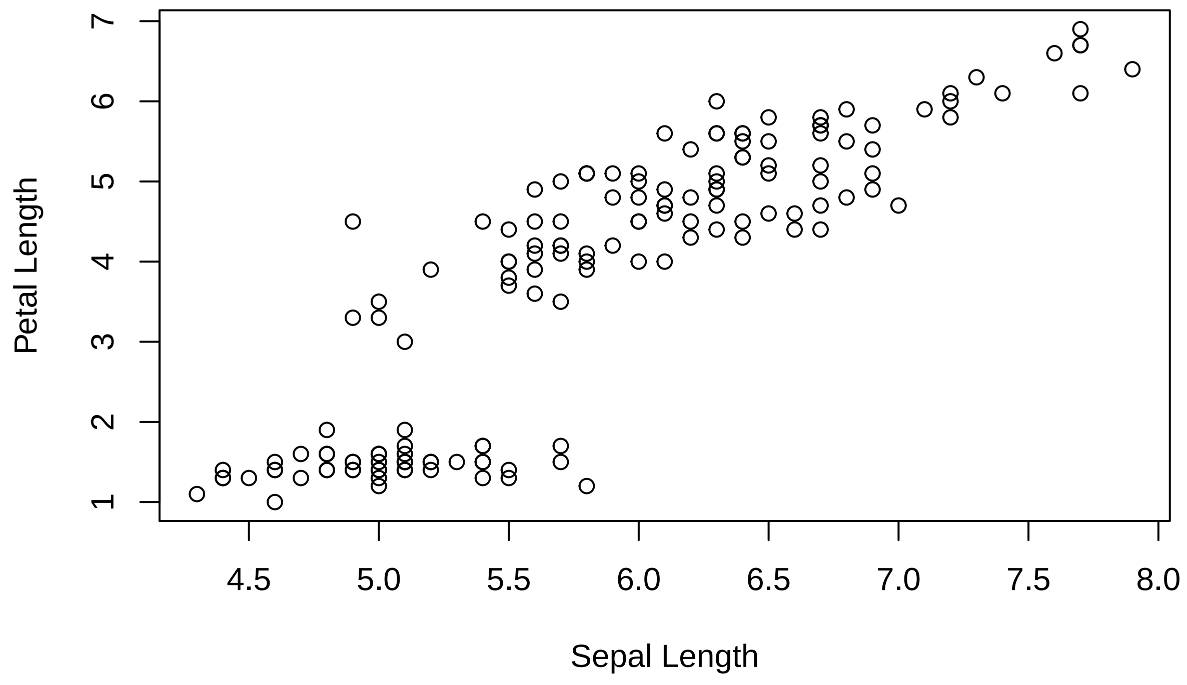
4e) Add the information from the post-hoc test to your figure

```r
iris_sepals_posthoc <- ggplot(iris_ds, aes(x=Species, y=Sepal.Length, col = Species)) +
  geom_boxplot() +
  xlab("Species") +
  ylab("Sepal Length")

iris_sepals_posthoc
```



5) Are sepal and petal length correlated? 5a) Plot sepal length against petal length.

```r
plot(iris_ds$Sepal.Length, iris_ds$Petal.Length, xlab = "Sepal Length", ylab = "Petal Length")
```



5b) Are these two measures correlated? (Hint: use a Pearson's correlation)

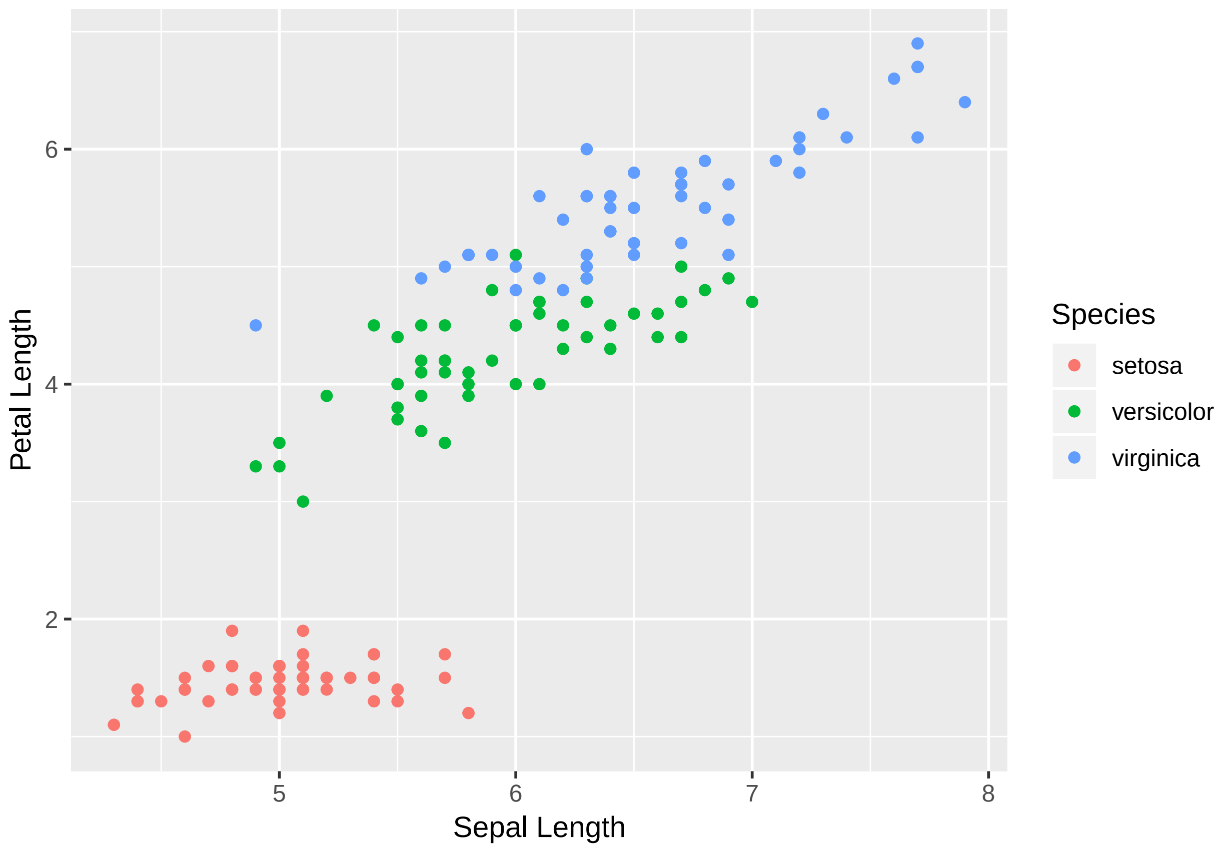
```
iris_ds_sep_pet_cor <- cor.test(iris_ds$Petal.Length, iris_ds$Sepal.Length)
iris_ds_sep_pet_cor
```

```
##
##  Pearson's product-moment correlation
##
## data:  iris_ds$Petal.Length and iris_ds$Sepal.Length
## t = 21.646, df = 148, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8270363 0.9055080
## sample estimates:
##       cor
## 0.8717538
```

Yes, these two measures are correlated (Pearson's R = .87, $p < 2.2e-16$).

5c) Color code your plot by species, include a legend on the plot.

```
speces.length.plot <- ggplot(iris_ds, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
  geom_point() +
  xlab("Sepal Length") +
  ylab("Petal Length")

speces.length.plot
```



5d) Do species vary in the relationship between sepal and petal length?

```
sepal_vs_petal_lm <- lm(iris_ds$Sepal.Length ~ iris_ds$Petal.Length + iris_ds$Species - 1)
summary(sepal_vs_petal_lm)
```

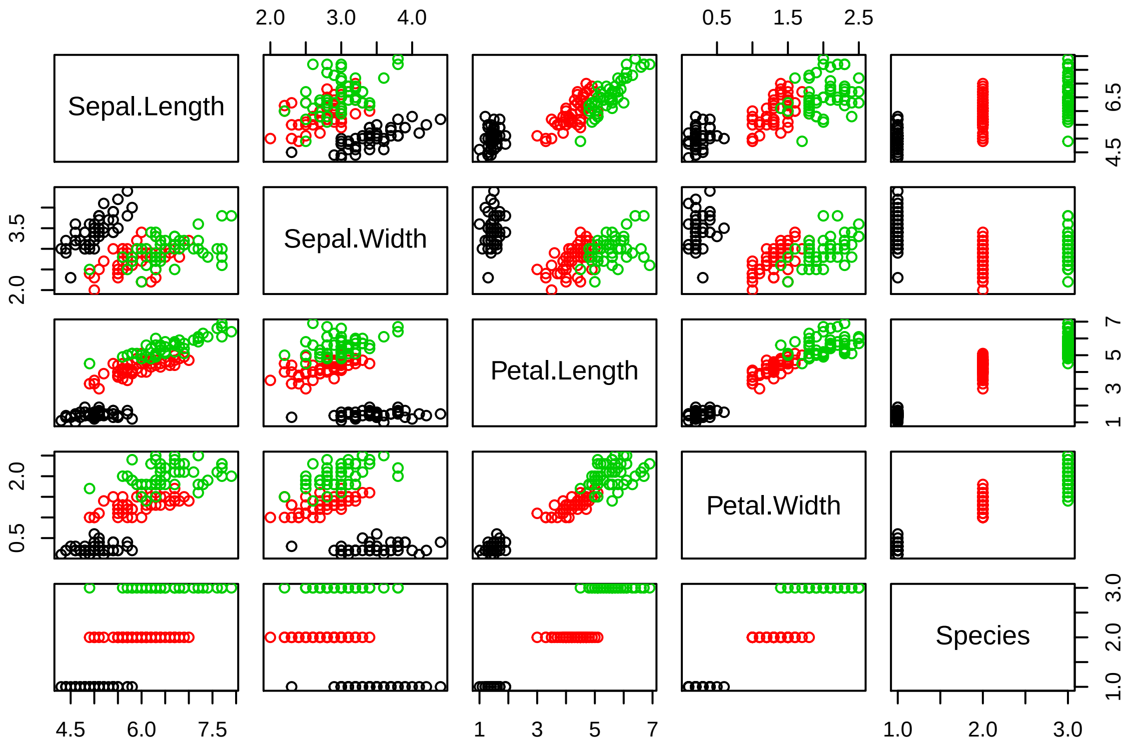
```
##
## Call:
## lm(formula = iris_ds$Sepal.Length ~ iris_ds$Petal.Length + iris_ds$Species -
##     1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.75310 -0.23142 -0.00081  0.23085  1.03100
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## iris_ds$Petal.Length        0.90456    0.06479  13.962  < 2e-16 ***
## iris_ds$Speciessetosa       3.68353    0.10610  34.719  < 2e-16 ***
## iris_ds$Speciesversicolor  2.08255    0.28010   7.435 8.17e-12 ***
## iris_ds$Speciesvirginica   1.56586    0.36285   4.315 2.92e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.338 on 146 degrees of freedom
## Multiple R-squared:  0.9968, Adjusted R-squared:  0.9967
## F-statistic: 1.139e+04 on 4 and 146 DF,  p-value: < 2.2e-16
```

```r
sepal_vs_petal_lm_int <- lm(iris_ds$Sepal.Length ~ iris_ds$Petal.Length * iris_ds$Species - 1)
summary(sepal_vs_petal_lm_int)
```

```
##
## Call:
## lm(formula = iris_ds$Sepal.Length ~ iris_ds$Petal.Length * iris_ds$Species -
##     1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.73479 -0.22785 -0.03132  0.24375  0.93608
##
## Coefficients:
##                                              Estimate Std. Error t value
## iris_ds$Petal.Length                           0.5423     0.2768   1.959
## iris_ds$Speciessetosa                          4.2132     0.4074  10.341
## iris_ds$Speciesversicolor                      2.4075     0.4383   5.493
## iris_ds$Speciesvirginica                       1.0597     0.4859   2.181
## iris_ds$Petal.Length:iris_ds$Speciesversicolor   0.2860     0.2951   0.969
## iris_ds$Petal.Length:iris_ds$Speciesvirginica    0.4534     0.2901   1.563
##                                              Pr(>|t|)
## iris_ds$Petal.Length                           0.0520 .
## iris_ds$Speciessetosa                          < 2e-16 ***
## iris_ds$Speciesversicolor                     1.74e-07 ***
## iris_ds$Speciesvirginica                       0.0308 *
## iris_ds$Petal.Length:iris_ds$Speciesversicolor   0.3340
## iris_ds$Petal.Length:iris_ds$Speciesvirginica    0.1203
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3365 on 144 degrees of freedom
## Multiple R-squared:  0.9969, Adjusted R-squared:  0.9967
## F-statistic:  7667 on 6 and 144 DF,  p-value: < 2.2e-16
```

6) Examine (plot and stats) the relationship between petal length and sepal width and how this relationship differs among the three species.

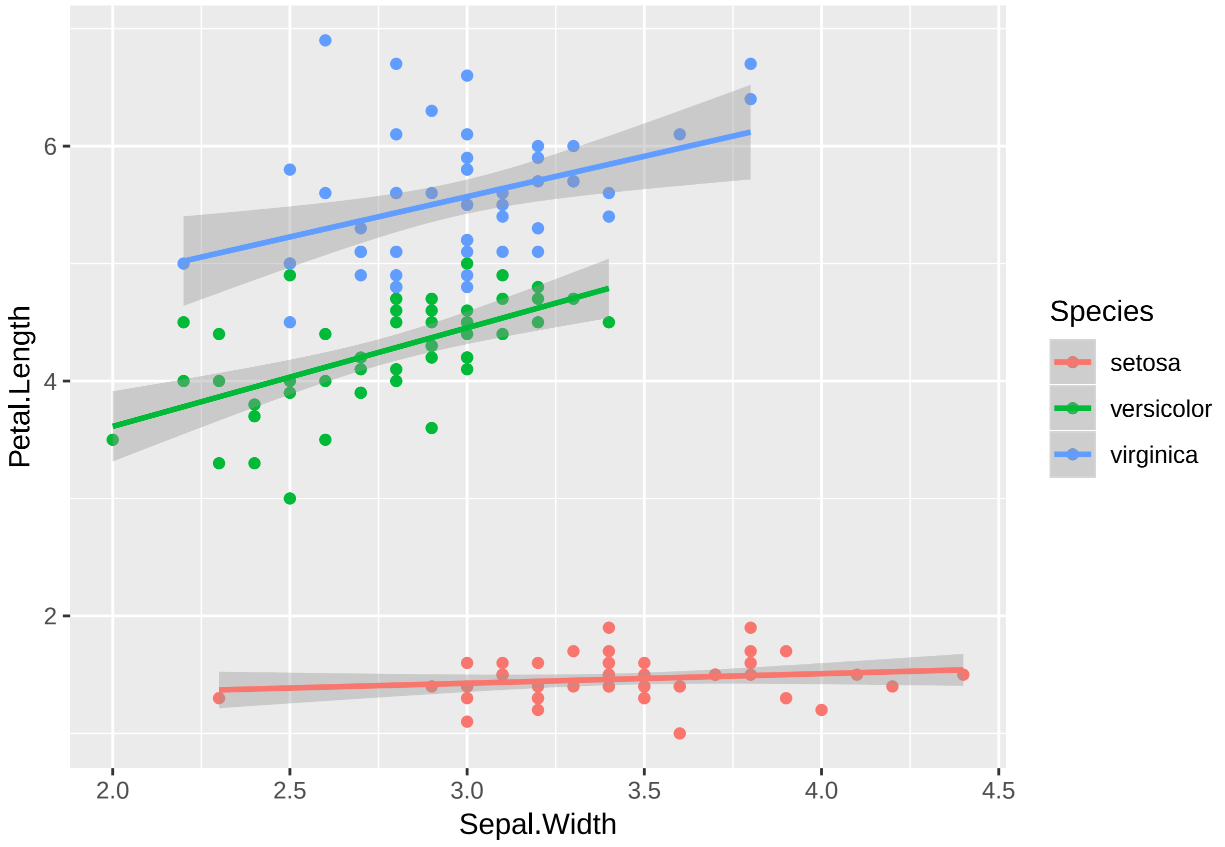
```r
plot(iris_ds, col = iris_ds$Species)
```



```r
sepalwidth_vs_petallength_lm_with_int <- lm(iris_ds$Petal.Length ~ iris_ds$Sepal.Width * iris_ds$Specie
summary(sepalwidth_vs_petallength_lm_with_int)
```

```
##
## Call:
## lm(formula = iris_ds$Petal.Length ~ iris_ds$Sepal.Width * iris_ds$Species -
##     1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.03337 -0.22012 -0.03026  0.17149  1.60468
##
## Coefficients:
##                                              Estimate Std. Error t value
## iris_ds$Sepal.Width                           0.08141    0.14520   0.561
## iris_ds$Speciessetosa                         1.18292    0.50072   2.362
## iris_ds$Speciesversicolor                     1.93492    0.48891   3.958
## iris_ds$Speciesvirginica                      3.51090    0.51049   6.877
## iris_ds$Sepal.Width:iris_ds$Speciesversicolor  0.75797    0.22770   3.329
## iris_ds$Sepal.Width:iris_ds$Speciesvirginica   0.60490    0.22408   2.699
##                                              Pr(>|t|)
## iris_ds$Sepal.Width                          0.575889
## iris_ds$Speciessetosa                        0.019494 *
## iris_ds$Speciesversicolor                    0.000118 ***
## iris_ds$Speciesvirginica                     1.72e-10 ***
## iris_ds$Sepal.Width:iris_ds$Speciesversicolor 0.001108 **
## iris_ds$Sepal.Width:iris_ds$Speciesvirginica  0.007776 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3853 on 144 degrees of freedom
## Multiple R-squared:  0.9917, Adjusted R-squared:  0.9914
## F-statistic:  2876 on 6 and 144 DF,  p-value: < 2.2e-16
```

```
linear_sepal_petal_plot <- ggplot(iris_ds, aes(x = Sepal.Width, y = Petal.Length, color = Species)) +
  geom_point() +
  geom_smooth(method = 'lm')
linear_sepal_petal_plot
```



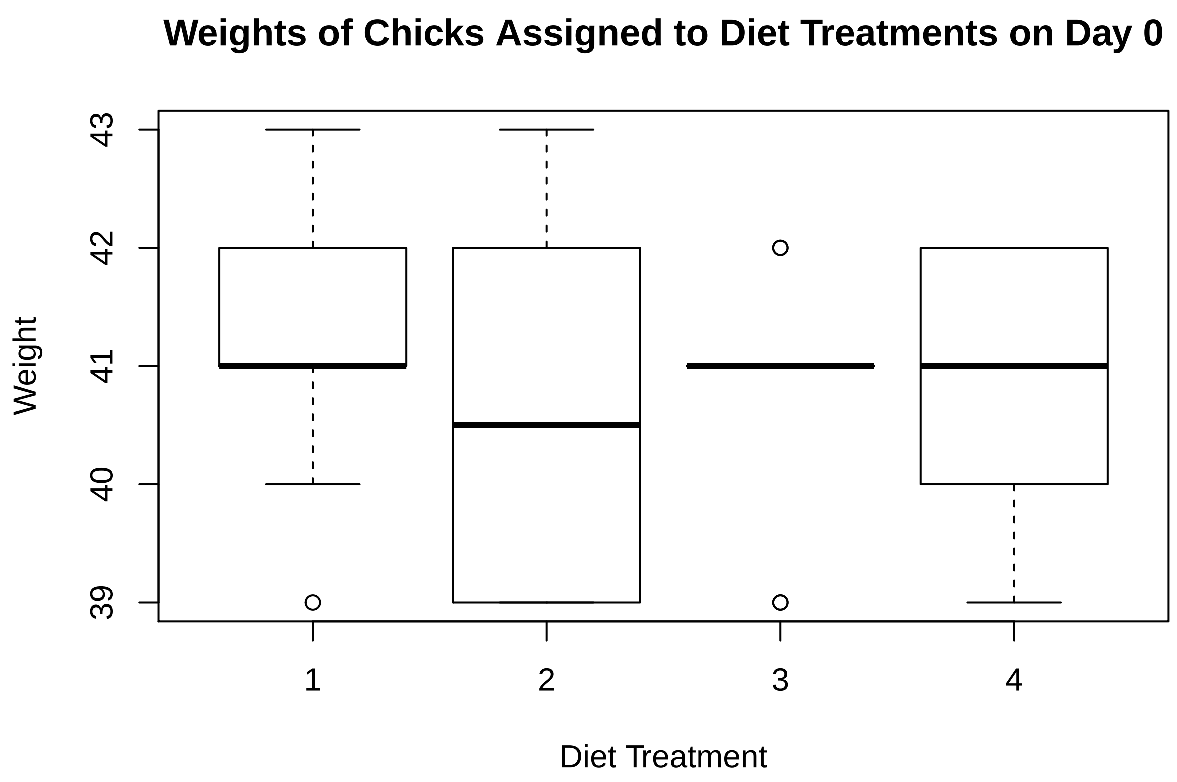
ChickWeight Exercise

1. Plot and examine if there is a statistically significant difference between the weights of chicks assigned to different diet treatments on day 0.

```
day_zero_chicks <- ChickWeight %>% filter(Time == 0)
head(day_zero_chicks)
```

```
##   weight Time Chick Diet
## 1     42    0     1    1
## 2     40    0     2    1
## 3     43    0     3    1
## 4     42    0     4    1
## 5     41    0     5    1
## 6     41    0     6    1
```

```
boxplot(data = day_zero_chicks, weight~Diet, main = "Weights of Chicks Assigned to Diet Treatments on Da
```



```r
anova_out <- oneway.test(day_zero_chicks$weight ~ day_zero_chicks$Diet)
anova_out
```

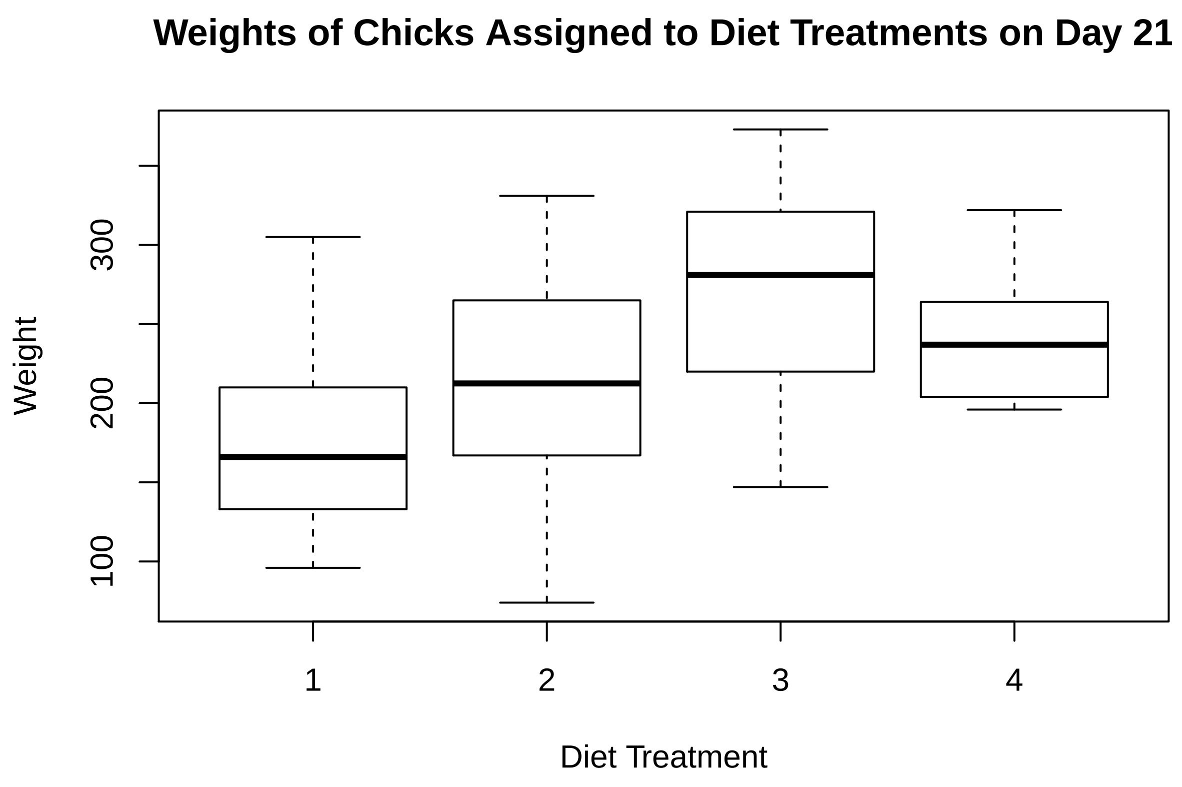
```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  day_zero_chicks$weight and day_zero_chicks$Diet
## F = 1.0842, num df = 3.0, denom df = 20.7, p-value = 0.3777
```

2. Plot and examine if there is a statistically significant difference between the weights of chicks assigned to different diet treatments at the end of the study (on day 21).

```r
day_twentyone_chicks <- ChickWeight %>% filter(Time == 21)
head(day_twentyone_chicks)
```

```
##   weight Time Chick Diet
## 1    205   21     1    1
## 2    215   21     2    1
## 3    202   21     3    1
## 4    157   21     4    1
## 5    223   21     5    1
## 6    157   21     6    1
```

```r
boxplot(data = day_twentyone_chicks, weight~Diet, main = "Weights of Chicks Assigned to Diet Treatments
```



```
anova_out <- oneway.test(day_twentyone_chicks$weight ~ day_twentyone_chicks$Diet)
anova_out
```

```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  day_twentyone_chicks$weight and day_twentyone_chicks$Diet
## F = 4.6618, num df = 3.000, denom df = 20.594, p-value = 0.01219
```

3. Is there an effect of diet on chick growth? (HINT: you will need to use a repeated measures ANOVA).

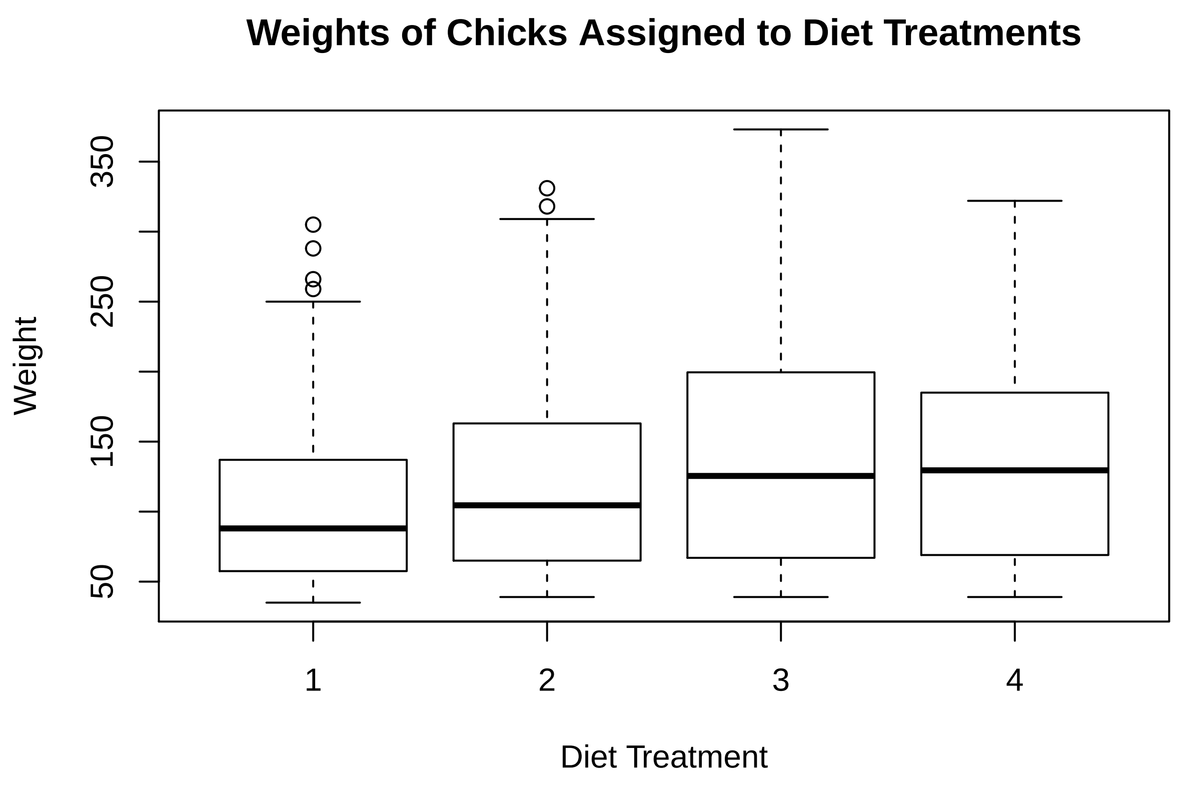
```
summary(aov(as.numeric(ChickWeight$Diet) ~ ChickWeight$weight + Error(rownames(ChickWeight)/ChickWeight$
```

```
## Warning in aov(as.numeric(ChickWeight$Diet) ~ ChickWeight$weight +
## Error(rownames(ChickWeight)/ChickWeight$weight)): Error() model is singular
```

```
##
## Error: rownames(ChickWeight)
##                     Df Sum Sq Mean Sq F value   Pr(>F)
## ChickWeight$weight   1   33.8   33.81    26.1 4.42e-07 ***
## Residuals          576  746.2    1.30
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4. Plot the effects of diet on chick growth.

```
boxplot(data = ChickWeight, weight~Diet, main = "Weights of Chicks Assigned to Diet Treatments", xlab =
```



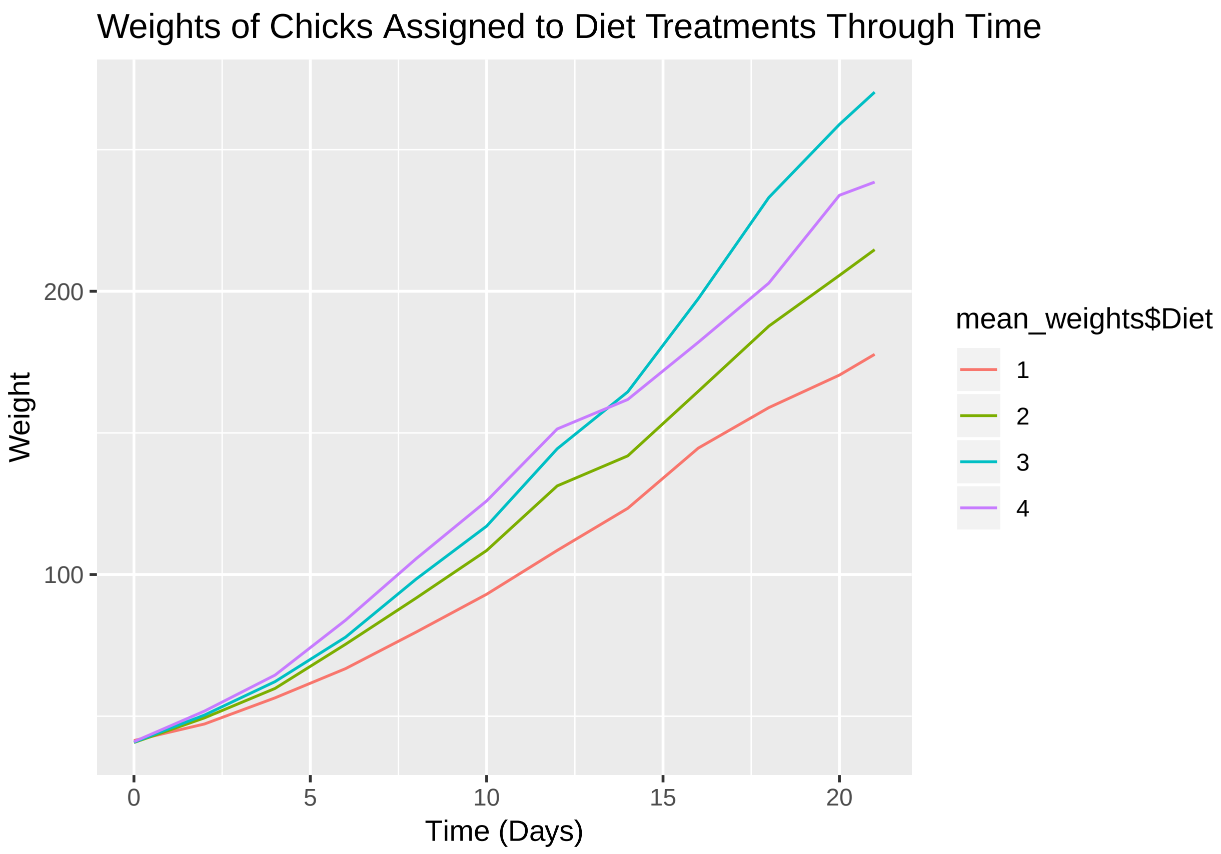
5. Using a 'for' loop, plot the growth of each chick in a different color, all on the same plot.

```
#average across diet type (4 lines)
mean_weights <- ChickWeight %>%
  group_by(Time, Diet) %>%
  summarise(MeanWeight = mean(weight))

mean_weights
```

```
## # A tibble: 48 x 3
## # Groups:   Time [12]
##      Time Diet  MeanWeight
##     <dbl> <fct>      <dbl>
##  1     0 1           41.4
##  2     0 2           40.7
##  3     0 3           40.8
##  4     0 4           41
##  5     2 1           47.2
##  6     2 2           49.4
##  7     2 3           50.4
##  8     2 4           51.8
##  9     4 1           56.5
## 10     4 2           59.8
## # ... with 38 more rows
```

```
p <- ggplot() +
  geom_line(data = mean_weights, aes(x = mean_weights$Time, y = mean_weights$MeanWeight, col = mean_wei
  xlab("Time (Days)") +
  ylab("Weight") +
  ggtitle("Weights of Chicks Assigned to Diet Treatments Through Time")
```

```
p
```



Kirk Lohmueller (Day 1)

Plotting and Simulation Exercises

Imagine that you wish to conduct a study of height of people living in Los Angeles. One strategy is to take a sample of 100 people and compute the average height. Another study design would involve taking a sample of 1000 people and computing the average height. You wish to test the extent to which the different sample sizes affect your estimates of the average height.

To test this, conduct a simple simulation study in R. Assume that the heights of people from LA are normally distributed with a mean of 69 inches and a standard deviation of 10 inches.

In order to conduct the simulation study, you should do the following:

1. Write a function (called "get_heights") in R to draw a sample of individuals (either 100 or 1000) from the population. Hint: You will want to use "rnorm" within your function. Store the random heights that you've generated in a variable called "heights".
2. Within your function, compute the average height from your "heights" vector.
3. Make your function return the average height.

```
get_heights <- function(n) {
  heights_people_LA <- rnorm(n, mean = 69, sd = 10)
  avg_heights <- mean(heights_people_LA)
  return(avg_heights)
}

heights <- get_heights(1000)
heights
```

```
## [1] 69.44785
```

4. Use a “for” loop to call your “get_heights” function 1000 times, with taking a sample of size 100 from the population. Save the mean height from each replicate in a vector called “mean_heights_100”.

```
mean_heights_100 <- vector()
for(i in 1:1000) {
  mean_heights_100[i] <- get_heights(100)
}
```

5. Use a “for” loop to call your “get_heights” function 1000 times, with taking a sample of size 1000 from the population. Save the mean height from each replicate in a vector called “mean_heights_1000”.

```
mean_heights_1000 <- vector()
for(i in 1:1000) {
  mean_heights_1000[i] <- get_heights(1000)
}
```

6. Plot a histogram of the distribution of the average heights for your sample size of 100 and 1000 individuals. The two sets of data should be plotted on the same axes. Add a legend. Label the axes. Plot the data from the 100 samples in red and the data from the 1000 samples in blue. Your plot should look something like the one shown on the next page.

```
mean_heights_100_df <- data.frame(mean_heights_100, rep(100))
mean_heights_1000_df <- data.frame(mean_heights_1000, rep(1000))

col_names <- c("Height", "SampleSize")
colnames(mean_heights_100_df) <- col_names
colnames(mean_heights_1000_df) <- col_names

mean_heights <- rbind(mean_heights_100_df, mean_heights_1000_df)

head(mean_heights)
```

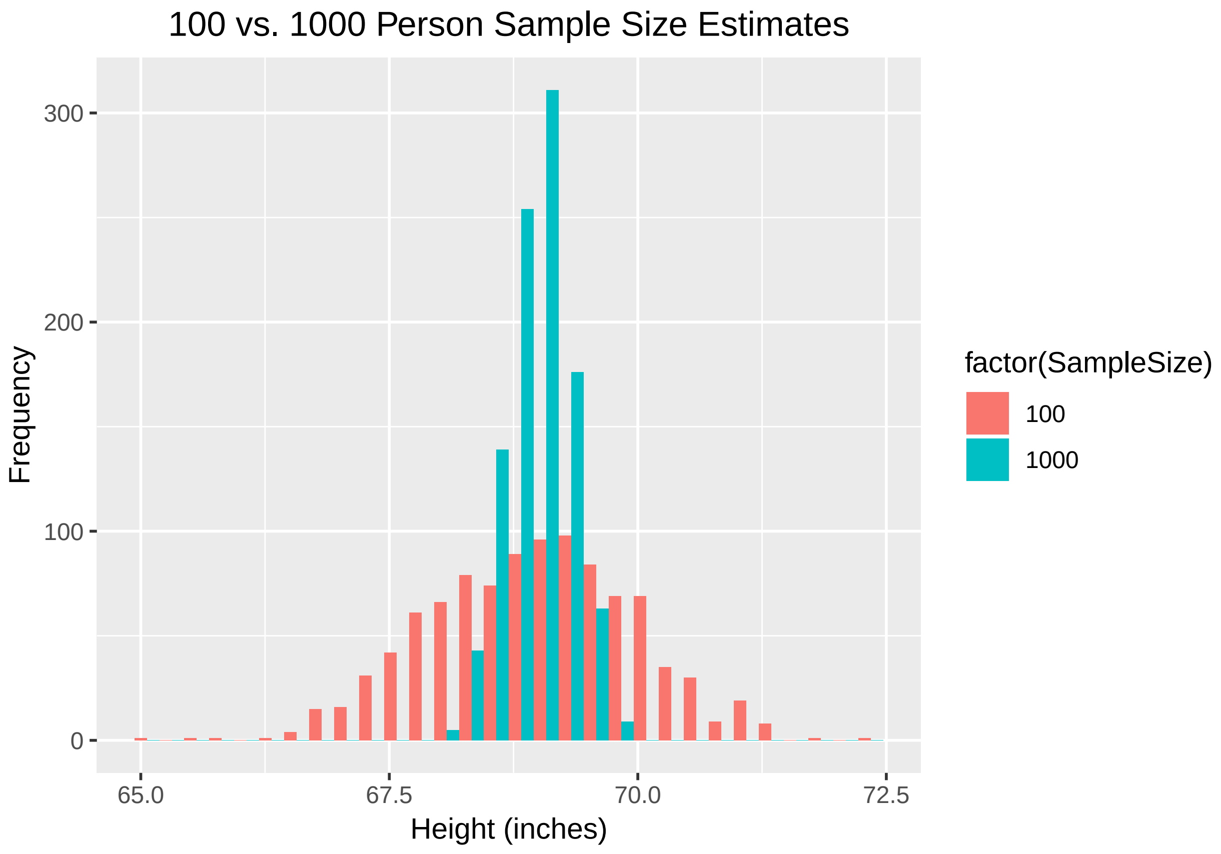
```
##     Height SampleSize
## 1 68.44552        100
## 2 67.81883        100
## 3 68.16173        100
## 4 68.36676        100
## 5 70.50538        100
## 6 68.67280        100
```

```
tail(mean_heights)
```

```
##        Height SampleSize
## 1995 69.09615       1000
## 1996 68.77235       1000
## 1997 69.24444       1000
## 1998 69.18098       1000
## 1999 68.81749       1000
## 2000 68.98784       1000
```

```
freq_hist_heights_plot <- ggplot(mean_heights, aes(x = Height, fill = factor(SampleSize))) +
  geom_histogram(alpha = 1, position = "dodge") +
  labs(title = "100 vs. 1000 Person Sample Size Estimates", x = "Height (inches)", y = "Frequency") +
  theme(plot.title = element_text(hjust = 0.5))

freq_hist_heights_plot
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Michael Alfaro (Day 2)

Control and Flow Exercises

1) Write a for loop statements so that it runs from 1:9 and prints the following output to your screen: • • • • • • • • • •

```
for(ii in 1:9)
{
  cat("•\n")
}
```

```
## •
## •
## •
## •
## •
## •
## •
## •
## •
```

2) Modify your for loop so that it prints 10 asterisks, with each asterisk separated by exactly one ampersand sign, with no spaces or new line characters.

```r
for(ii in 1:10)
{
  if (ii != 10)
  {
    cat("*&")
  }
  else
  {
    cat("*")
  }

}
```

```
## *&*&*&*&*&*&*&*&*&*
```

Exercise 3) by hand, figure out the initial values of these variables and values at the the start and end of each iteration of the loop

```r
dogs <- 10
for (i in 1:5){
  dogs <- dogs + 1;
}

###
meatloaf <- 0
for (i in 5:9){
  meatloaf <- meatloaf - i + 1
  cat(meatloaf)
}
```

```
## -4-9-15-22-30
```

```r
###
bubbles <- 12
for (i in -1:-4){
    bubbles <- i
}
```

dogs: (1st iteration start = 10; 1st iteration end = 11, 2nd iteration start = 11, 2nd iteration end = 12, 3rd iteration start = 12, 3rd iteration end = 13, 4th iteration start = 13, 4th iteration end = 14, 5th iteration start = 14, 5th iteration end = 15)

meatloaf: (1st iteration start = 0; 1st iteration end = -4, 2nd iteration start = -4, 2nd iteration end = -9, 3rd iteration start = -9, 3rd iteration end = -15, 4th iteration start = -15, 4th iteration end = -22, 5th iteration start = -22, 5th iteration end = -30)

bubbles: (1st iteration start = 12; 1st iteration end = -1, 2nd iteration start = -1, 2nd iteration end = -2, 3rd iteration start = -2, 3rd iteration end = -3, 4th iteration start = -3, 4th iteration end = -4)

5) More fun with loops. Here are the bank accounts from seven randomly selected UCLA grad students

```
bankAccounts <- c(10, 9.2, 5.6, 3.7, 8.8, 0.5)

# Now look at the error message the following lines of code produce.
# Can you think of a way to modify this?

###Modification (START)###
compounded <- rep(length(bankAccounts))
compounded
```

```
## [1] 6
```

```
###Modification (END)###

interestRate <- 0.0125
for (i in 1:length(bankAccounts)) {
  compounded[i] <- interestRate*bankAccounts[i] + bankAccounts[i]
}

compounded
```

```
## [1] 10.12500  9.31500  5.67000  3.74625  8.91000  0.50625
```

```
# HINT: variables must be initialized before you can perform operations on them
# HINT 2: look at the rep() function and see if you can use that to initialize a variable that will hel
```

6) Go back to the compounded interest example. Suppose we now want to compound the interest annually, but across a period of 5 years. The for loop we discussed earlier only compounds for a single year. Try this:

```
bankAccounts <- c(10, 9.2, 5.6) #define bank accounts here
interestRate <- 0.0525;
house <- c(4.8, 3.8, 5.7) #deduct
food<- c(3.5, 4.3, 5.0) #deduct
fun <- c(7.8, 2.1, 10.5) #deduct
#and incomes (through TAships) of
income <- c(21, 21, 21) #add this

for (i in 1:length(bankAccounts)) {
  #step 1 modify bankAccounts so that amounts reflect income and expenses
  bankAccounts[i] <- bankAccounts[i] - house[i] - food[i] - fun[i] + income[i]

  for (j in 1:5) {
    bankAccounts[i] <- interestRate*bankAccounts[i] + bankAccounts[i]
  }
}

bankAccounts
```

```
## [1] 19.244064 25.830958  6.974359
```

7) Three students have estimated annual expenditures for food, housing, and fun of: (in thousands of dollars)

```
house <- c(4.8, 3.8, 5.7)
food<- c(3.5, 4.3, 5.0)
fun <- c(7.8, 2.1, 10.5)
```

```r
#and incomes (through TAships) of

income <- c(21, 21, 21)

new_bankAccounts <- c(10, 9.2, 5.6)

years_to_calc <- c(2015, 2016, 2017, 2018, 2019, 2020)

trust_fund_amt <- 5000
```

Modify the 5-year interest-compounding code from #5 and #6 so that it runs from 2015-2020 and so that in odd numbered years students 1 and 3 get trust fund disbursements of $5000. (hint the modulus function %% will be helpful)

```r
for (i in 1:length(new_bankAccounts)) {
  new_bankAccounts[i] <- new_bankAccounts[i] - house[i] - food[i] - fun[i] + income[i]

  for (j in 1:length(years_to_calc)) {
    if (years_to_calc[j] %% 2 != 0)
    {
      new_bankAccounts[1] <- new_bankAccounts[1] + trust_fund_amt
      new_bankAccounts[3] <- new_bankAccounts[3] + trust_fund_amt
    }
    new_bankAccounts[i] <- interestRate*new_bankAccounts[i] + new_bankAccounts[i]
  }
}

new_bankAccounts
```

```
## [1] 48491.42608    27.18708 59259.13762
```

8) use a while loop to sum all numbers from 1:17. You will need to use a counter variable (like index seen in class).

```r
counter <- 1
counter_max <- 17
csum <- 0
while (counter < counter_max + 1)
{
  csum <- csum + counter
  counter <- counter + 1
}
csum
```

```
## [1] 153
```

9) write a function takes a number, and prints 'small' if number less than or equal to -1; 'medium' if between -1 and + 1'big' if greater than or equal to + 1

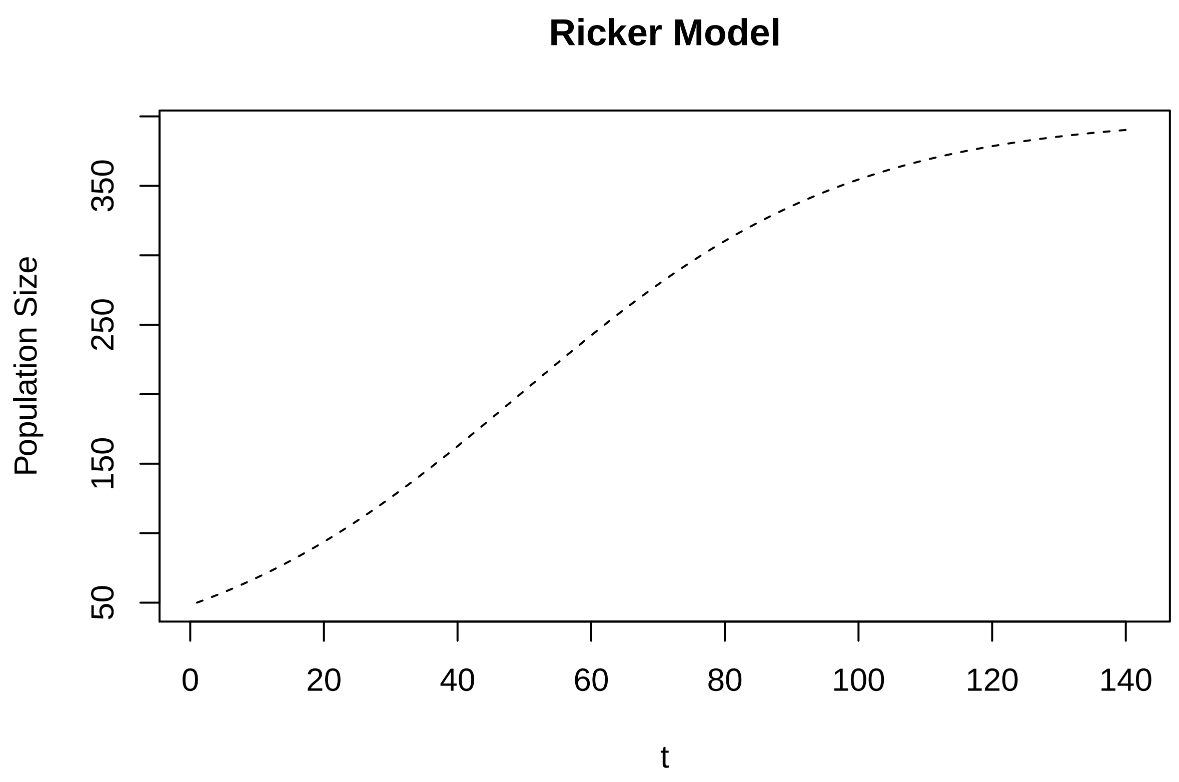
```r
smlfun <- function(x)
{
  if (x <= -1)
  {
    print('small')
  }
  else if (x < 1 && x > -1)
  {
    print('medium')
  }
  else
  {
    print('large')
  }
}
```

Jamie Lloyd-Smith (Day 2)

Bootcamp Modeling Exercises

(a) Write a function that runs the Ricker model, plots the result, and returns the time series as an output. At minimum, your function should take all parameter values and initial conditions as input arguments. (Hint: this should involve minimal modification of a modeling function we wrote in class.)

```
RickerModel <- function(n_t=50, r=0.04, K=400, ttMax=140, PLOT_FLAG = 1)
{
  NN <- rep(NA, ttMax + 1)
  NN[1] <- n_t

  for (tt in 1:ttMax) {
    NN[tt + 1] <- NN[tt]*exp(r*(1-(NN[tt]/K)))
  }

  if(PLOT_FLAG == 1) {
    plot(1:(ttMax+1), NN, lty = 2, type = 'l', xlab = 't', ylab = 'Population Size', main = 'Ricker Mode
  }


  return(NN)
}

RickerModel(50, 0.04, 400, 140)
```

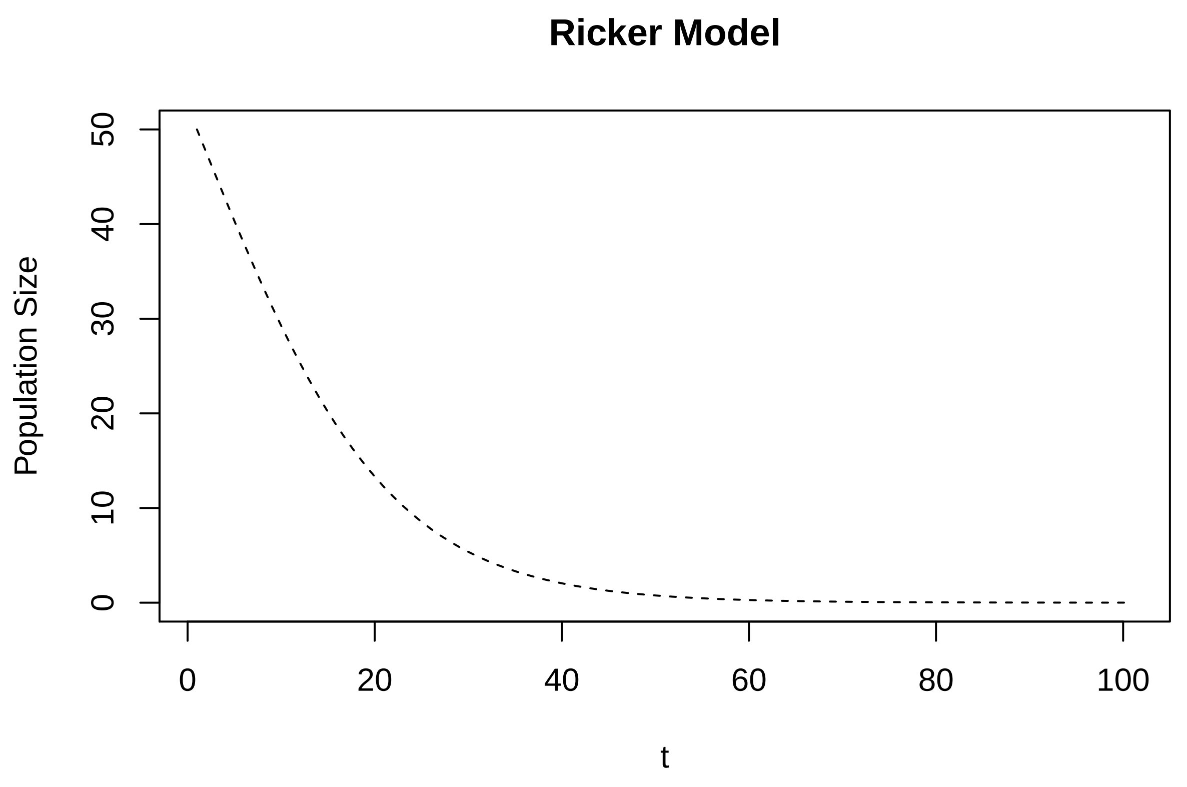


```
##   [1]  50.00000  51.78099  53.61586  55.50557  57.45102  59.45309  61.51262
##   [8]  63.63038  65.80712  68.04351  70.34017  72.69764  75.11642  77.59691
##  [15]  80.13942  82.74420  85.41139  88.14105  90.93312  93.78744  96.70375
##  [22]  99.68167 102.72070 105.82022 108.97948 112.19760 115.47359 118.80631
##  [29] 122.19448 125.63670 129.13144 132.67700 136.27160 139.91327 143.59997
##  [36] 147.32948 151.09949 154.90756 158.75113 162.62756 166.53407 170.46781
##  [43] 174.42585 178.40516 182.40265 186.41518 190.43955 194.47252 198.51082
##  [50] 202.55117 206.59026 210.62481 214.65153 218.66716 222.66849 226.65232
##  [57] 230.61554 234.55507 238.46795 242.35125 246.20216 250.01797 253.79605
##  [64] 257.53390 261.22915 264.87952 268.48287 272.03720 275.54063 278.99143
##  [71] 282.38797 285.72880 289.01258 292.23813 295.40437 298.51038 301.55538
##  [78] 304.53869 307.45978 310.31822 313.11373 315.84611 318.51529 321.12131
##  [85] 323.66429 326.14446 328.56213 330.91771 333.21168 335.44460 337.61708
##  [92] 339.72981 341.78355 343.77910 345.71730 347.59905 349.42528 351.19697
##  [99] 352.91510 354.58072 356.19486 357.75860 359.27302 360.73921 362.15829
## [106] 363.53135 364.85952 366.14391 367.38563 368.58579 369.74550 370.86584
## [113] 371.94790 372.99276 374.00147 374.97508 375.91463 376.82113 377.69557
## [120] 378.53894 379.35219 380.13628 380.89212 381.62062 382.32266 382.99911
## [127] 383.65079 384.27854 384.88316 385.46542 386.02609 386.56589 387.08556
## [134] 387.58578 388.06724 388.53059 388.97646 389.40549 389.81826 390.21537
## [141] 390.59737
```

(b) Explore the dynamics of the model. Try to find combinations of parameter values that yield the following patterns:

- Population decreases to n = 0.

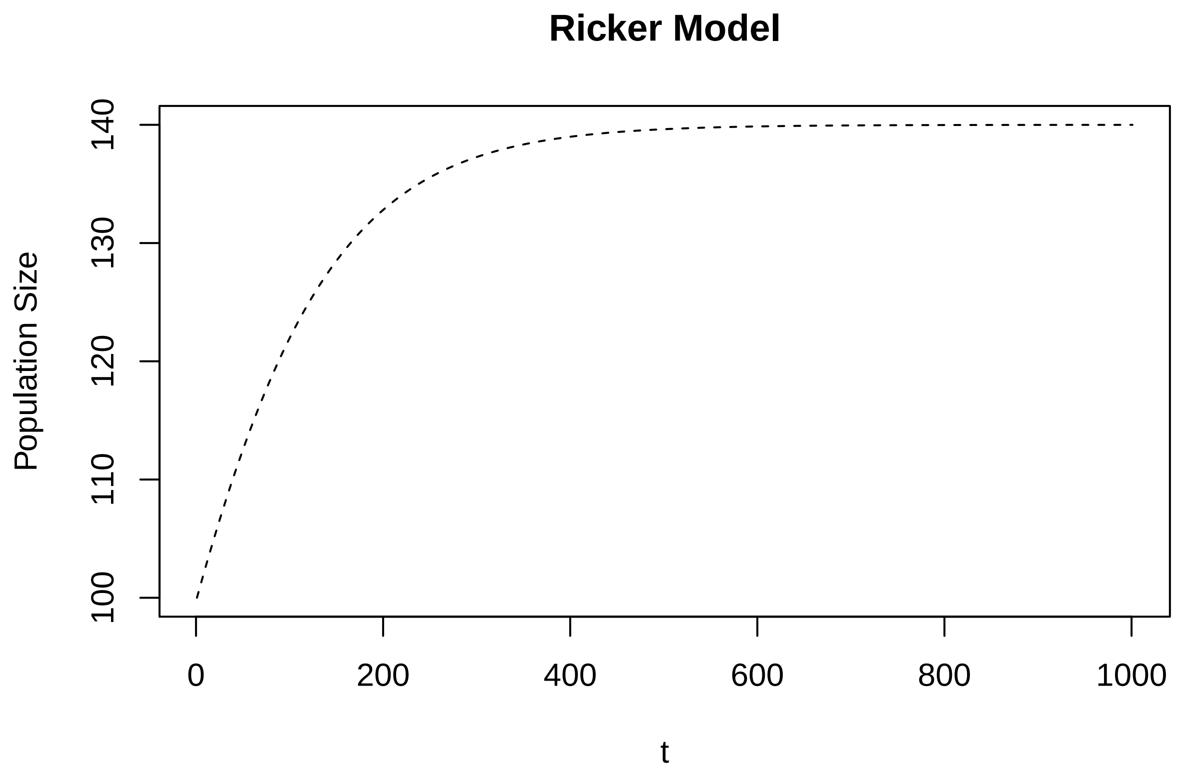
```
RickerModel(50, -0.1, 100, 100)
```



```
##   [1] 50.000000000 47.561471225 45.131681702 42.722091781 40.343821089
##   [6] 38.007445366 35.722813439 33.498890874 31.343635149 29.263905314
##  [11] 27.265407151 25.352673006 23.529073849 21.796859840 20.157224804
##  [16] 18.610389500 17.155698449 15.791725305 14.516382123 13.327028557
##  [21] 12.220577656 11.193595693 10.242394163  9.363112727  8.551792449
##  [26]  7.804439174  7.117077202  6.485793782  5.906775057  5.376334283
##  [31]  4.890933163  4.447197174  4.041925739  3.672098042  3.334875243
##  [36]  3.027599755  2.747792179  2.493146425  2.261523461  2.050944066
##  [41]  1.859580922  1.685750276  1.527903421  1.384618134  1.254590224
##  [46]  1.136625283  1.029630726  0.932608156  0.844646112  0.764913216
##  [51]  0.692651715  0.627171450  0.567844220  0.514098543  0.465414805
##  [56]  0.421320774  0.381387454  0.345225279  0.312480607  0.282832512
##  [61]  0.255989832  0.231688481  0.209688984  0.189774228  0.171747413
##  [66]  0.155430178  0.140660902  0.127293151  0.115194269  0.104244092
##  [71]  0.094333789  0.085364794  0.077247854  0.069902148  0.063254501
##  [76]  0.057238660  0.051794646  0.046868161  0.042410053  0.038375831
##  [81]  0.034725220  0.031421770  0.028432486  0.025727509  0.023279812
##  [86]  0.021064935  0.019060743  0.017247202  0.015606183  0.014121279
##  [91]  0.012777642  0.011561836  0.010461703  0.009466239  0.008565489
##  [96]  0.007750441  0.007012943  0.006345618  0.005741789  0.005195415
## [101]  0.004701031
```

- Population approaches stable equilibrium at n = K, without oscillations.

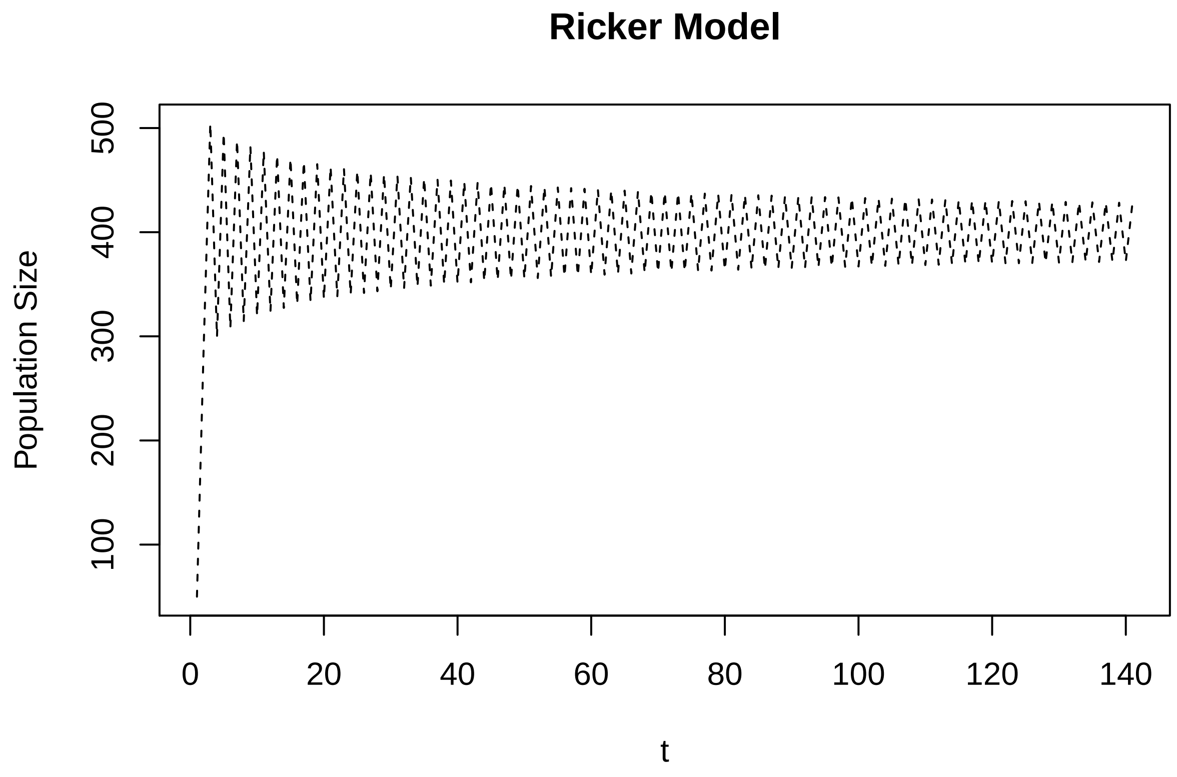
```
RickerModel(100, 0.01, 140, 1000)
```



```
##    [1] 100.0000 100.2861 100.5710 100.8547 101.1370 101.4182 101.6981
##    [8] 101.9767 102.2540 102.5301 102.8049 103.0784 103.3506 103.6215
##   [15] 103.8911 104.1594 104.4264 104.6921 104.9564 105.2195 105.4812
##   [22] 105.7416 106.0007 106.2584 106.5148 106.7699 107.0236 107.2760
##   [29] 107.5270 107.7767 108.0251 108.2721 108.5177 108.7620 109.0050
##   [36] 109.2466 109.4868 109.7257 109.9632 110.1994 110.4342 110.6677
##   [43] 110.8998 111.1306 111.3600 111.5880 111.8147 112.0400 112.2640
##   [50] 112.4867 112.7079 112.9279 113.1464 113.3637 113.5796 113.7941
##   [57] 114.0073 114.2192 114.4297 114.6389 114.8468 115.0533 115.2585
##   [64] 115.4624 115.6649 115.8661 116.0660 116.2646 116.4619 116.6579
##   [71] 116.8526 117.0459 117.2380 117.4287 117.6182 117.8064 117.9933
##   [78] 118.1789 118.3633 118.5463 118.7281 118.9087 119.0879 119.2660
##   [85] 119.4427 119.6182 119.7925 119.9655 120.1373 120.3079 120.4773
##   [92] 120.6454 120.8123 120.9780 121.1425 121.3057 121.4678 121.6287
##   [99] 121.7884 121.9470 122.1043 122.2605 122.4155 122.5694 122.7221
##  [106] 122.8736 123.0240 123.1733 123.3214 123.4684 123.6143 123.7591
##  [113] 123.9027 124.0453 124.1867 124.3271 124.4663 124.6045 124.7416
##  [120] 124.8776 125.0126 125.1465 125.2793 125.4111 125.5419 125.6716
##  [127] 125.8003 125.9280 126.0546 126.1802 126.3048 126.4285 126.5511
##  [134] 126.6727 126.7933 126.9130 127.0317 127.1494 127.2662 127.3820
##  [141] 127.4969 127.6108 127.7238 127.8358 127.9469 128.0571 128.1664
##  [148] 128.2748 128.3823 128.4888 128.5945 128.6993 128.8033 128.9063
##  [155] 129.0085 129.1098 129.2103 129.3099 129.4087 129.5066 129.6037
##  [162] 129.7000 129.7955 129.8901 129.9840 130.0770 130.1692 130.2606
##  [169] 130.3513 130.4412 130.5303 130.6186 130.7061 130.7929 130.8790
##  [176] 130.9643 131.0488 131.1326 131.2157 131.2981 131.3797 131.4606
##  [183] 131.5408 131.6203 131.6992 131.7773 131.8547 131.9314 132.0075
##  [190] 132.0829 132.1576 132.2316 132.3050 132.3778 132.4499 132.5213
##  [197] 132.5921 132.6623 132.7318 132.8008 132.8691 132.9368 133.0039
##  [204] 133.0703 133.1362 133.2015 133.2662 133.3303 133.3939 133.4568
## [211] 133.5192 133.5810 133.6423 133.7030 133.7631 133.8227 133.8818
## [218] 133.9403 133.9983 134.0558 134.1127 134.1691 134.2250 134.2804
## [225] 134.3353 134.3896 134.4435 134.4969 134.5497 134.6021 134.6540
## [232] 134.7055 134.7564 134.8069 134.8569 134.9065 134.9555 135.0042
## [239] 135.0524 135.1001 135.1474 135.1943 135.2407 135.2866 135.3322
## [246] 135.3773 135.4220 135.4663 135.5102 135.5537 135.5967 135.6394
## [253] 135.6816 135.7235 135.7650 135.8060 135.8467 135.8870 135.9270
## [260] 135.9665 136.0057 136.0445 136.0829 136.1210 136.1587 136.1961
## [267] 136.2331 136.2698 136.3061 136.3421 136.3777 136.4130 136.4479
## [274] 136.4826 136.5169 136.5508 136.5845 136.6178 136.6508 136.6835
## [281] 136.7159 136.7480 136.7797 136.8112 136.8424 136.8732 136.9038
## [288] 136.9341 136.9641 136.9938 137.0232 137.0523 137.0812 137.1098
## [295] 137.1381 137.1661 137.1939 137.2214 137.2486 137.2756 137.3023
## [302] 137.3288 137.3550 137.3809 137.4066 137.4321 137.4573 137.4823
## [309] 137.5070 137.5315 137.5557 137.5798 137.6035 137.6271 137.6504
## [316] 137.6735 137.6964 137.7191 137.7415 137.7637 137.7857 137.8075
## [323] 137.8291 137.8505 137.8717 137.8926 137.9134 137.9339 137.9543
## [330] 137.9744 137.9944 138.0142 138.0338 138.0531 138.0723 138.0914
## [337] 138.1102 138.1288 138.1473 138.1656 138.1837 138.2016 138.2194
## [344] 138.2369 138.2544 138.2716 138.2887 138.3056 138.3223 138.3389
## [351] 138.3553 138.3716 138.3877 138.4036 138.4194 138.4350 138.4505
## [358] 138.4658 138.4810 138.4960 138.5109 138.5256 138.5402 138.5547
## [365] 138.5690 138.5831 138.5972 138.6110 138.6248 138.6384 138.6519
## [372] 138.6652 138.6785 138.6916 138.7045 138.7174 138.7301 138.7426
## [379] 138.7551 138.7674 138.7797 138.7918 138.8037 138.8156 138.8273
## [386] 138.8390 138.8505 138.8619 138.8732 138.8844 138.8954 138.9064
## [393] 138.9172 138.9280 138.9386 138.9492 138.9596 138.9699 138.9801
## [400] 138.9903 139.0003 139.0102 139.0200 139.0298 139.0394 139.0489
## [407] 139.0584 139.0677 139.0770 139.0862 139.0953 139.1042 139.1131
## [414] 139.1220 139.1307 139.1393 139.1479 139.1563 139.1647 139.1730
## [421] 139.1813 139.1894 139.1975 139.2054 139.2133 139.2212 139.2289
## [428] 139.2366 139.2442 139.2517 139.2591 139.2665 139.2738 139.2810
## [435] 139.2882 139.2953 139.3023 139.3092 139.3161 139.3229 139.3296
## [442] 139.3363 139.3429 139.3494 139.3559 139.3623 139.3687 139.3750
## [449] 139.3812 139.3873 139.3934 139.3995 139.4055 139.4114 139.4172
## [456] 139.4231 139.4288 139.4345 139.4401 139.4457 139.4512 139.4567
## [463] 139.4621 139.4675 139.4728 139.4780 139.4832 139.4884 139.4935
## [470] 139.4985 139.5035 139.5085 139.5133 139.5182 139.5230 139.5278
## [477] 139.5325 139.5371 139.5417 139.5463 139.5508 139.5553 139.5597
## [484] 139.5641 139.5685 139.5728 139.5770 139.5812 139.5854 139.5896
## [491] 139.5936 139.5977 139.6017 139.6057 139.6096 139.6135 139.6174
## [498] 139.6212 139.6250 139.6287 139.6324 139.6361 139.6397 139.6433
## [505] 139.6468 139.6504 139.6539 139.6573 139.6607 139.6641 139.6675
## [512] 139.6708 139.6741 139.6773 139.6805 139.6837 139.6869 139.6900
## [519] 139.6931 139.6962 139.6992 139.7022 139.7052 139.7081 139.7110
## [526] 139.7139 139.7168 139.7196 139.7224 139.7252 139.7279 139.7306
## [533] 139.7333 139.7360 139.7386 139.7412 139.7438 139.7464 139.7489
## [540] 139.7514 139.7539 139.7563 139.7588 139.7612 139.7636 139.7659
## [547] 139.7683 139.7706 139.7729 139.7751 139.7774 139.7796 139.7818
## [554] 139.7840 139.7861 139.7883 139.7904 139.7925 139.7945 139.7966
## [561] 139.7986 139.8006 139.8026 139.8046 139.8065 139.8085 139.8104
## [568] 139.8123 139.8142 139.8160 139.8179 139.8197 139.8215 139.8233
## [575] 139.8250 139.8268 139.8285 139.8302 139.8319 139.8336 139.8353
## [582] 139.8369 139.8385 139.8401 139.8417 139.8433 139.8449 139.8464
## [589] 139.8480 139.8495 139.8510 139.8525 139.8539 139.8554 139.8569
## [596] 139.8583 139.8597 139.8611 139.8625 139.8639 139.8652 139.8666
## [603] 139.8679 139.8692 139.8705 139.8718 139.8731 139.8744 139.8756
## [610] 139.8769 139.8781 139.8793 139.8805 139.8817 139.8829 139.8841
## [617] 139.8852 139.8864 139.8875 139.8886 139.8897 139.8908 139.8919
## [624] 139.8930 139.8941 139.8951 139.8962 139.8972 139.8983 139.8993
## [631] 139.9003 139.9013 139.9023 139.9032 139.9042 139.9052 139.9061
## [638] 139.9070 139.9080 139.9089 139.9098 139.9107 139.9116 139.9125
## [645] 139.9134 139.9142 139.9151 139.9159 139.9168 139.9176 139.9184
## [652] 139.9192 139.9200 139.9208 139.9216 139.9224 139.9232 139.9240
## [659] 139.9247 139.9255 139.9262 139.9270 139.9277 139.9284 139.9291
## [666] 139.9298 139.9305 139.9312 139.9319 139.9326 139.9333 139.9339
## [673] 139.9346 139.9353 139.9359 139.9365 139.9372 139.9378 139.9384
## [680] 139.9390 139.9397 139.9403 139.9409 139.9414 139.9420 139.9426
## [687] 139.9432 139.9437 139.9443 139.9449 139.9454 139.9460 139.9465
## [694] 139.9470 139.9476 139.9481 139.9486 139.9491 139.9496 139.9501
## [701] 139.9506 139.9511 139.9516 139.9521 139.9526 139.9531 139.9535
## [708] 139.9540 139.9544 139.9549 139.9554 139.9558 139.9562 139.9567
## [715] 139.9571 139.9575 139.9580 139.9584 139.9588 139.9592 139.9596
## [722] 139.9600 139.9604 139.9608 139.9612 139.9616 139.9620 139.9624
## [729] 139.9627 139.9631 139.9635 139.9638 139.9642 139.9646 139.9649
## [736] 139.9653 139.9656 139.9660 139.9663 139.9666 139.9670 139.9673
## [743] 139.9676 139.9680 139.9683 139.9686 139.9689 139.9692 139.9695
## [750] 139.9698 139.9701 139.9704 139.9707 139.9710 139.9713 139.9716
## [757] 139.9719 139.9722 139.9724 139.9727 139.9730 139.9733 139.9735
## [764] 139.9738 139.9741 139.9743 139.9746 139.9748 139.9751 139.9753
## [771] 139.9756 139.9758 139.9761 139.9763 139.9765 139.9768 139.9770
## [778] 139.9772 139.9775 139.9777 139.9779 139.9781 139.9783 139.9786
## [785] 139.9788 139.9790 139.9792 139.9794 139.9796 139.9798 139.9800
## [792] 139.9802 139.9804 139.9806 139.9808 139.9810 139.9812 139.9814
## [799] 139.9816 139.9817 139.9819 139.9821 139.9823 139.9825 139.9826
## [806] 139.9828 139.9830 139.9832 139.9833 139.9835 139.9837 139.9838
## [813] 139.9840 139.9841 139.9843 139.9845 139.9846 139.9848 139.9849
## [820] 139.9851 139.9852 139.9854 139.9855 139.9857 139.9858 139.9859
## [827] 139.9861 139.9862 139.9864 139.9865 139.9866 139.9868 139.9869
## [834] 139.9870 139.9872 139.9873 139.9874 139.9875 139.9877 139.9878
## [841] 139.9879 139.9880 139.9881 139.9883 139.9884 139.9885 139.9886
## [848] 139.9887 139.9888 139.9890 139.9891 139.9892 139.9893 139.9894
## [855] 139.9895 139.9896 139.9897 139.9898 139.9899 139.9900 139.9901
## [862] 139.9902 139.9903 139.9904 139.9905 139.9906 139.9907 139.9908
## [869] 139.9909 139.9910 139.9911 139.9911 139.9912 139.9913 139.9914
## [876] 139.9915 139.9916 139.9917 139.9917 139.9918 139.9919 139.9920
## [883] 139.9921 139.9922 139.9922 139.9923 139.9924 139.9925 139.9925
## [890] 139.9926 139.9927 139.9928 139.9928 139.9929 139.9930 139.9930
## [897] 139.9931 139.9932 139.9932 139.9933 139.9934 139.9935 139.9935
## [904] 139.9936 139.9936 139.9937 139.9938 139.9938 139.9939 139.9940
## [911] 139.9940 139.9941 139.9941 139.9942 139.9943 139.9943 139.9944
## [918] 139.9944 139.9945 139.9945 139.9946 139.9946 139.9947 139.9947
## [925] 139.9948 139.9949 139.9949 139.9950 139.9950 139.9951 139.9951
## [932] 139.9952 139.9952 139.9953 139.9953 139.9953 139.9954 139.9954
## [939] 139.9955 139.9955 139.9956 139.9956 139.9957 139.9957 139.9957
## [946] 139.9958 139.9958 139.9959 139.9959 139.9960 139.9960 139.9960
## [953] 139.9961 139.9961 139.9962 139.9962 139.9962 139.9963 139.9963
## [960] 139.9963 139.9964 139.9964 139.9965 139.9965 139.9965 139.9966
##  [967] 139.9966 139.9966 139.9967 139.9967 139.9967 139.9968 139.9968
##  [974] 139.9968 139.9969 139.9969 139.9969 139.9969 139.9970 139.9970
##  [981] 139.9970 139.9971 139.9971 139.9971 139.9972 139.9972 139.9972
##  [988] 139.9972 139.9973 139.9973 139.9973 139.9973 139.9974 139.9974
##  [995] 139.9974 139.9975 139.9975 139.9975 139.9975 139.9976 139.9976
```

- Decaying oscillations around n = K.

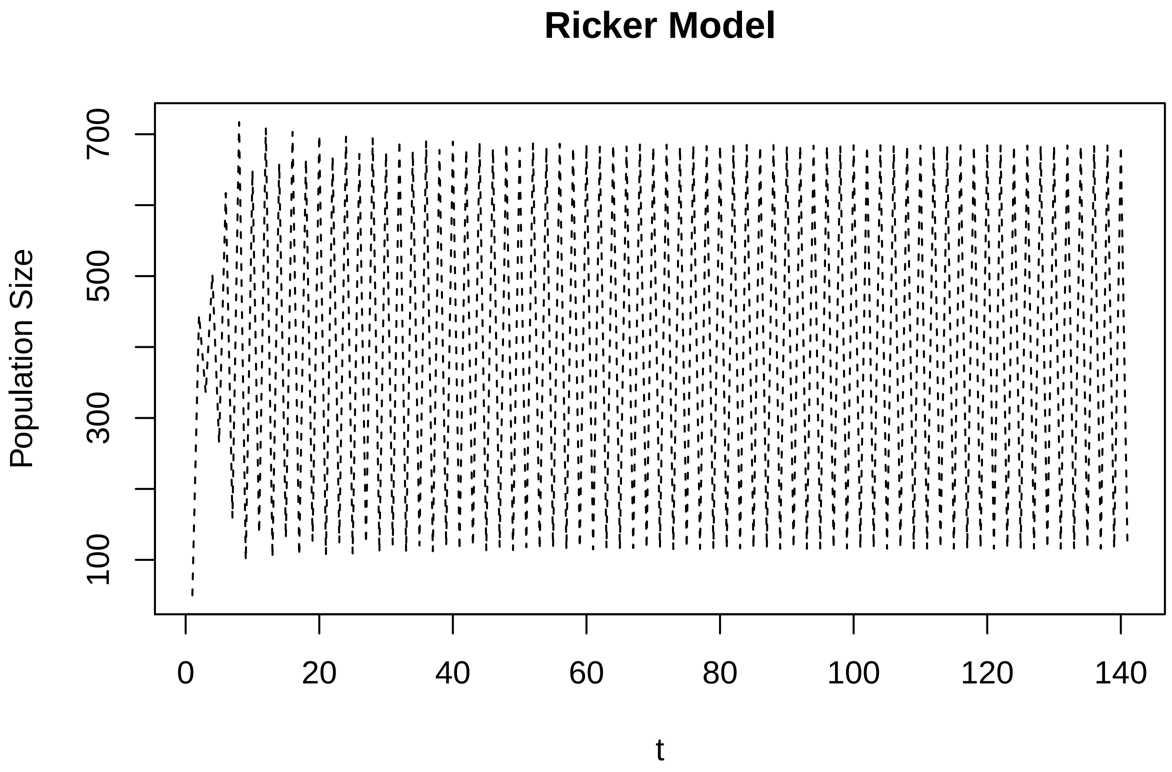
```r
RickerModel(50, 2.0, 400, 140)
```



```
##   [1]  50.0000 287.7301 504.4014 299.2758 495.2123 307.6384 488.2039
##   [8] 314.1007 482.6106 319.3083 478.0023 323.6310 474.1145 327.2997
##  [15] 470.7738 330.4676 467.8607 333.2414 465.2897 335.6979 462.9977
##  [22] 337.8945 460.9370 339.8745 459.0706 341.6720 457.3693 343.3137
##  [29] 455.8100 344.8212 454.3737 346.2120 453.0449 347.5007 451.8106
##  [36] 348.6993 450.6601 349.8179 449.5842 350.8652 448.5751 351.8485
##  [43] 447.6261 352.7741 446.7314 353.6475 445.8859 354.4735 445.0854
##  [50] 355.2562 444.3259 355.9993 443.6040 356.7062 442.9166 357.3796
##  [57] 442.2612 358.0221 441.6352 358.6361 441.0365 359.2236 440.4633
##  [64] 359.7865 439.9136 360.3264 439.3860 360.8449 438.8790 361.3434
##  [71] 438.3912 361.8232 437.9215 362.2853 437.4688 362.7309 437.0321
##  [78] 363.1610 436.6104 363.5763 436.2030 363.9778 435.8090 364.3661
##  [85] 435.4276 364.7421 435.0583 365.1063 434.7004 365.4593 434.3533
##  [92] 365.8018 434.0165 366.1342 433.6895 366.4570 433.3718 366.7707
##  [99] 433.0630 367.0757 432.7627 367.3724 432.4705 367.6611 432.1860
## [106] 367.9423 431.9089 368.2162 431.6389 368.4831 431.3757 368.7434
## [113] 431.1189 368.9973 430.8685 369.2451 430.6240 369.4869 430.3852
## [120] 369.7232 430.1520 369.9540 429.9241 370.1795 429.7014 370.4001
## [127] 429.4835 370.6157 429.2704 370.8267 429.0619 371.0332 428.8578
## [134] 371.2354 428.6580 371.4333 428.4623 371.6272 428.2706 371.8172
## [141] 428.0827
```

- Persistent, regular oscillations.

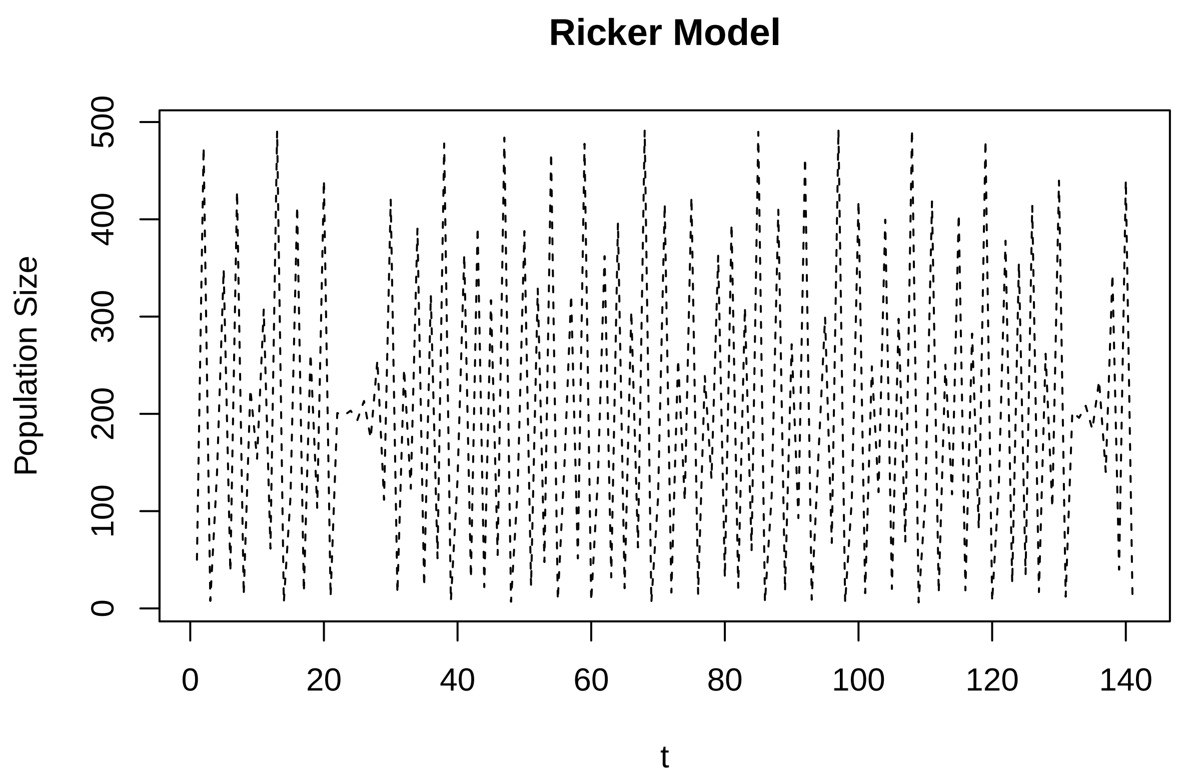
```
RickerModel(50, 2.5, 400, 140)
```



```
##   [1]  50.00000 445.64515 335.03695 502.83390 264.42246 617.02385 158.93711
##   [8] 717.05436  98.84588 649.22227 136.74747 708.72264 102.91956 658.98480
##  [15] 130.58776 703.36220 105.62110 664.95961 126.94180 699.48368 107.61601
##  [22] 669.12398 124.45503 696.52279 109.16201 672.20983 122.64072 694.19618
##  [29] 110.39098 674.57631 121.26556 692.33719 111.38199 676.42949 120.19842
##  [36] 690.83690 112.18768 677.90027 119.35754 689.61874 112.84574 679.07792
##  [43] 118.68808 688.62605 113.38460 680.02654 118.15129 687.81531 113.82642
##  [50] 680.79384 117.71872 687.15236 114.18887 681.41627 117.36887 686.60988
##  [57] 114.48623 681.92222 117.08518 686.16583 114.73016 682.33410 116.85470
##  [64] 685.80231 114.93020 682.66976 116.66717 685.50471 115.09420 682.94353
##  [71] 116.51443 685.26111 115.22861 683.16694 116.38992 685.06171 115.33872
##  [78] 683.34934 116.28835 684.89852 115.42892 683.49830 116.20545 684.76498
##  [85] 115.50277 683.62000 116.13778 684.65572 115.56324 683.71944 116.08250
##  [92] 684.56632 115.61272 683.80070 116.03735 684.49319 115.65323 683.86711
##  [99] 116.00046 684.43336 115.68637 683.92140 115.97031 684.38442 115.71348
## [106] 683.96578 115.94568 684.34440 115.73566 684.00206 115.92554 684.31166
## [113] 115.75381 684.03172 115.90907 684.28488 115.76865 684.05597 115.89562
## [120] 684.26299 115.78079 684.07580 115.88462 684.24508 115.79072 684.09201
## [127] 115.87562 684.23043 115.79884 684.10526 115.86827 684.21846 115.80548
## [134] 684.11610 115.86225 684.20866 115.81091 684.12496 115.85734 684.20065
## [141] 115.81536
```

- Crazy, random-looking fluctuations (chaos).

```
RickerModel(50, 3.0, 200, 140)
```

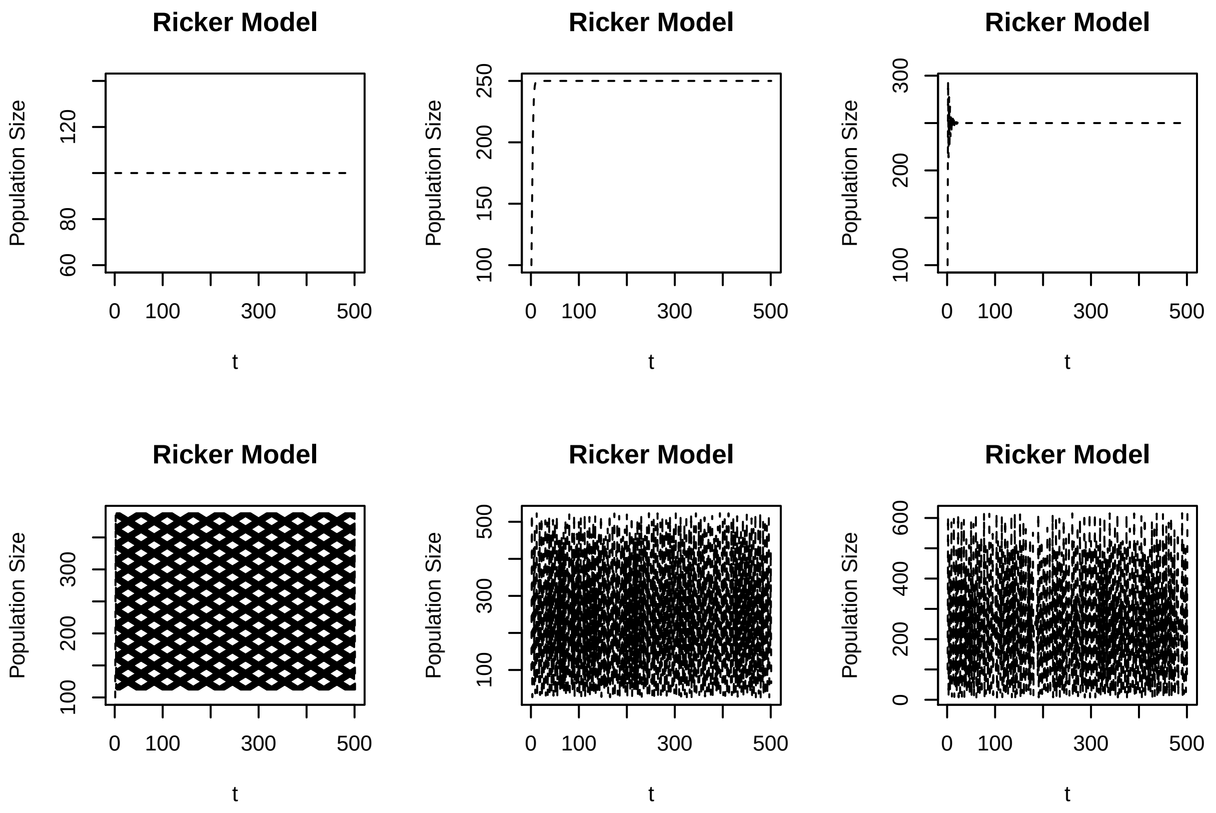


```
##   [1]  50.000000 474.386792   7.738603 138.399176 348.679949  37.485538
##   [7] 429.092340  13.808537 225.463658 153.884776 307.333192  61.432228
##  [13] 491.003455   6.242616 114.178167 413.677874  16.775500 261.984930
##  [19] 103.390495 440.387758  11.963247 200.816294 198.372410 203.275046
##  [25] 193.530337 213.252995 174.807315 255.079328 111.651752 420.148572
##  [31]  15.461913 246.275568 123.016389 390.360925  22.458247 322.074919
##  [37]  51.607159 477.972413   7.388811 132.838574 363.782459  31.181319
##  [43] 392.328452  21.915028 316.855926  54.905212 483.973475   6.837540
##  [49] 123.948342 387.858178  23.167886 328.733962  47.667001 468.358647
##  [55]   8.363313 148.176620 322.387952  51.415329 477.567936   7.427486
##  [61] 133.456436 362.102950  31.829207 396.607138  20.776845 305.572317
##  [67]  62.715067 491.703413   6.186222 113.242461 416.086961  16.274344
##  [73] 256.076110 110.424613 423.250354  14.867960 238.934428 133.242761
##  [79] 362.683777  31.603715 395.131628  21.162792 309.451896  59.920832
##  [85] 489.905122   6.332119 115.659801 409.835605  17.605688 271.547401
##  [91]  92.844467 463.246213   8.931332 156.897939 299.503601  67.327746
##  [97] 492.579680   6.116323 112.080369 419.058556  15.676023 248.885259
## [103] 119.547418 399.615097  20.010865 297.707778  68.751305 492.367871
## [109]   6.133147 112.360321 418.344829  15.817764 250.602281 117.311516
## [115] 405.515942  18.586231 282.485604  81.969197 481.451537   7.064149
## [121] 127.621684 377.943621  26.195581 355.191475  34.632235 413.766317
## [127]  16.756841 261.766788 103.642986 439.794414  12.053935 202.063528
## [133] 195.904877 208.315980 183.886066 234.164854 140.267675 343.620359
## [139]  39.854370 440.282475  11.979291
```

Which parameter is the key driver of these patterns? −> The “r” parameter is the key driver of these patterns.

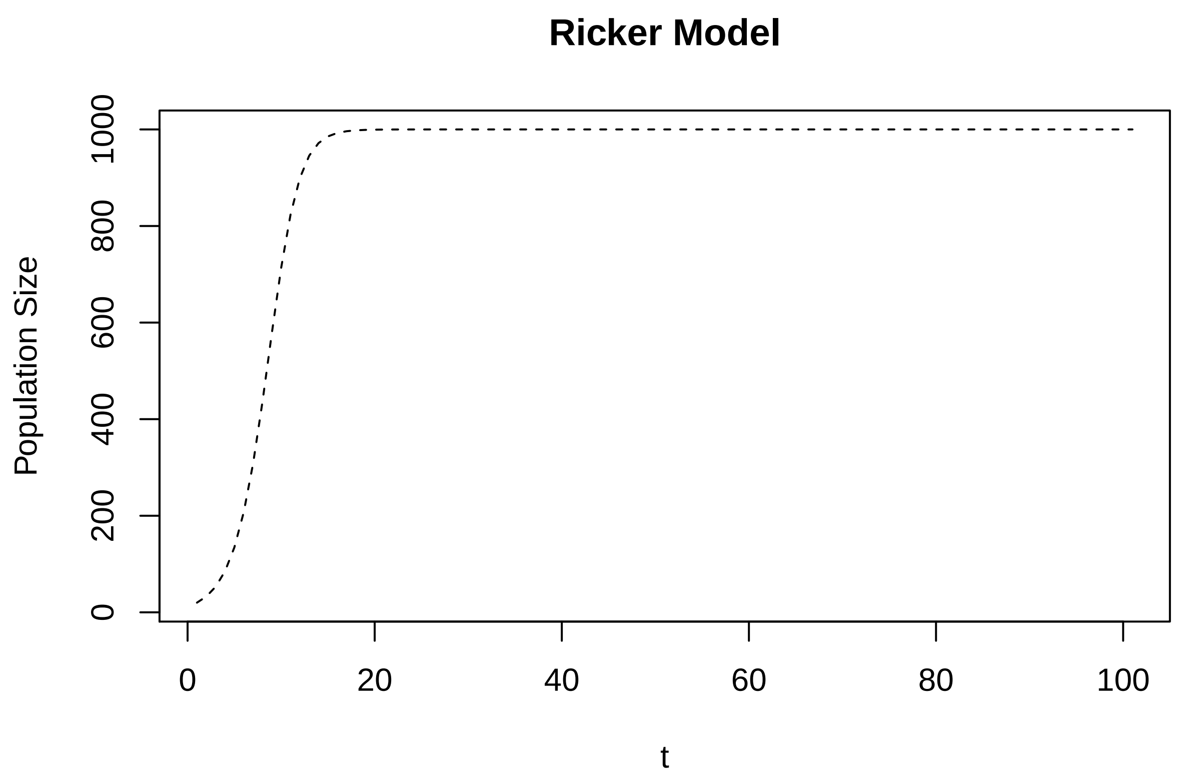
(c) Choose six interesting values of this parameter. Write a script that makes an array of six plots showing the model dynamics for each of these values.

```r
K_param <- 250
r_set <- c(0.0, 0.5, 1.8, 2.25, 2.75, 3.0)
init_pop <- 100
time_to_run <- 500

par(mfrow=c(2,3))

for(ii in r_set)
{
  RickerModel(init_pop, ii, K_param, time_to_run)
}
```



(d) Imagine n0 = 20 and K = 1000 for a certain population of deer that is growing according to the Ricker model. You are a wildlife manager, and are concerned about how long it will take for the population to reach half of its carrying capacity. That is, you want to know tK/2, the first year that nt greater-than or equal to K/2. Suppose your output time series is called nVec. Write an R command that will determine the index of the first element of nVec that is greater-than or equal to K/2.

```r
K_param <- 1000
r_static <- 0.5
init_pop <- 20
time_to_run <- 100

nVec <- RickerModel(init_pop, r_static, K_param, time_to_run)
```



```r
RickerModelHalfCarrying_Index <- function(time_series, K_capacity, PRINT_FLAG = 1) {
  K_half <- K_capacity / 2
  if(PRINT_FLAG == 1) {
    cat("Half of Carrying Capacity is: ", K_half, "\n")
  }

  for(ii in 1:length(time_series)) {
    if(time_series[ii] >= K_half) {
      return(ii)
    }
  }
}

tk2_index <- RickerModelHalfCarrying_Index(nVec, K_param)
```

```
## Half of Carrying Capacity is:  500
```

```r
tk2_index
```

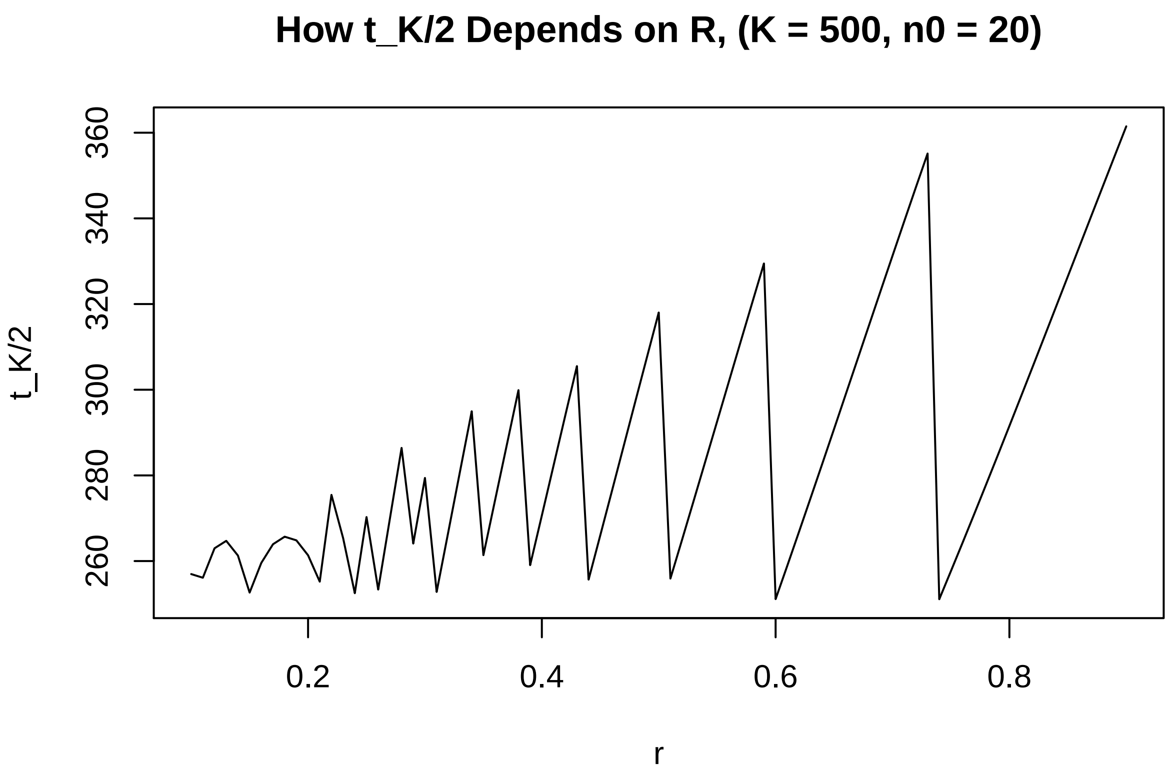
```
## [1] 9
```

(e) Write a script that runs the necessary simulations and collects the necessary data to plot how tK/2 depends on r, for the range of r from 0.1 to 0.9.

```r
r_sim_seq <- seq(0.1, 0.9, by = 0.01)
r_sim_seq
```

```
##  [1] 0.10 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23
## [15] 0.24 0.25 0.26 0.27 0.28 0.29 0.30 0.31 0.32 0.33 0.34 0.35 0.36 0.37
## [29] 0.38 0.39 0.40 0.41 0.42 0.43 0.44 0.45 0.46 0.47 0.48 0.49 0.50 0.51
## [43] 0.52 0.53 0.54 0.55 0.56 0.57 0.58 0.59 0.60 0.61 0.62 0.63 0.64 0.65
## [57] 0.66 0.67 0.68 0.69 0.70 0.71 0.72 0.73 0.74 0.75 0.76 0.77 0.78 0.79
## [71] 0.80 0.81 0.82 0.83 0.84 0.85 0.86 0.87 0.88 0.89 0.90
```

```
K_param <- 500
init_pop <- 20
time_to_run <- 100

nVec_set <- list()
for(nnn in 1:length(r_sim_seq)) {
  nVec_set[[nnn]] <- RickerModel(init_pop, r_sim_seq[nnn], K_param, time_to_run, 0)
}

tk2_data_set_indices <- matrix(NA, nrow = length(r_sim_seq), ncol = 1)

for(ii in 1:length(r_sim_seq)) {
  tk2_data_set_indices[ii] <- RickerModelHalfCarrying_Index(nVec_set[[ii]], K_param, 0)
}

tk2_values <- matrix(NA, nrow = length(tk2_data_set_indices), ncol = 1)
for(ii in 1:length(tk2_data_set_indices)) {
  tk2_values[ii] <- nVec_set[[ii]][tk2_data_set_indices[ii]]
}

plot(r_sim_seq, tk2_values, type='l', xlab = "r", ylab = "t_K/2", main = "How t_K/2 Depends on R, (K = 5
```

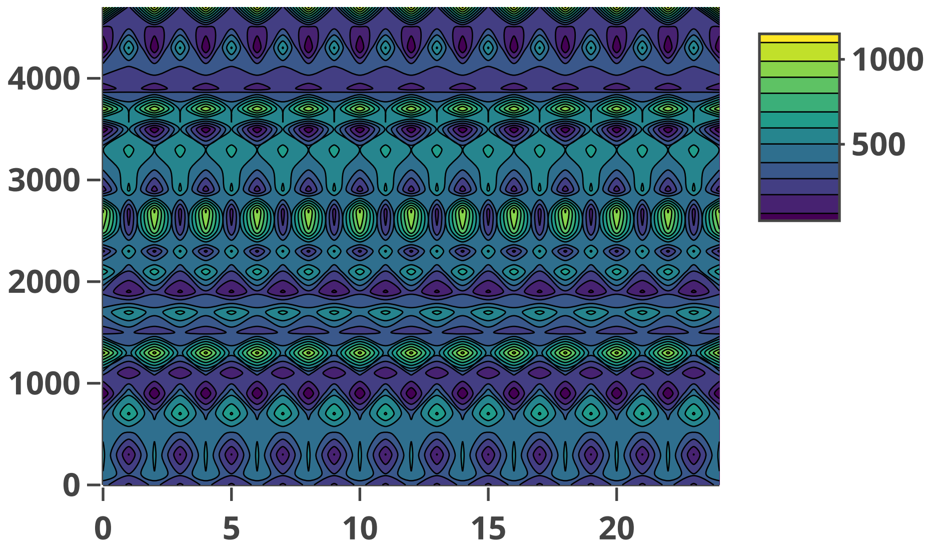


(f) Write pseudo-code to do a joint sensitivity analysis for two parameters. That is, choose a vector of values you'd like to consider for both r and K, and choose a simple output from your model (e.g. population size at t = 10). Run the model for all possible combinations of these values and collect the results in a matrix with appropriate dimensions. Plot the results in some way. (Hint: the straight-forward way to do this uses a 'nested loop'.) (g) (BONUS, OPTIONAL, PRETTY STRAIGHTFORWARD) Convert your pseudo-code into an R script to do a two-dimensional sensitivity analysis, and create a visually appealing plot to summarize the results. (You will need to google for plotting commands, e.g. contour or surface plots).

```r
r_sens <- c(runif(50, min = 0, max = 3))
K_sens <- c(floor(runif(50, min = 0, max = 700)))
init_pop <- 100
time_to_run <- 400

nVec_sens_set <- list()
for(rr in 1:length(r_sens)) {
  nVec_sens_set[[rr]] <- RickerModel(init_pop, r_sens[rr], K_sens[rr], time_to_run, 0)
}

time_index_to_select_from <- 10

n_at_time_t <- matrix(NA, nrow = length(nVec_sens_set), ncol = 1)
for(ii in 1:length(n_at_time_t)) {
  n_at_time_t[ii] <- nVec_sens_set[[ii]][time_index_to_select_from]
}

## For 3D Surface Plot
library(plotly)
```

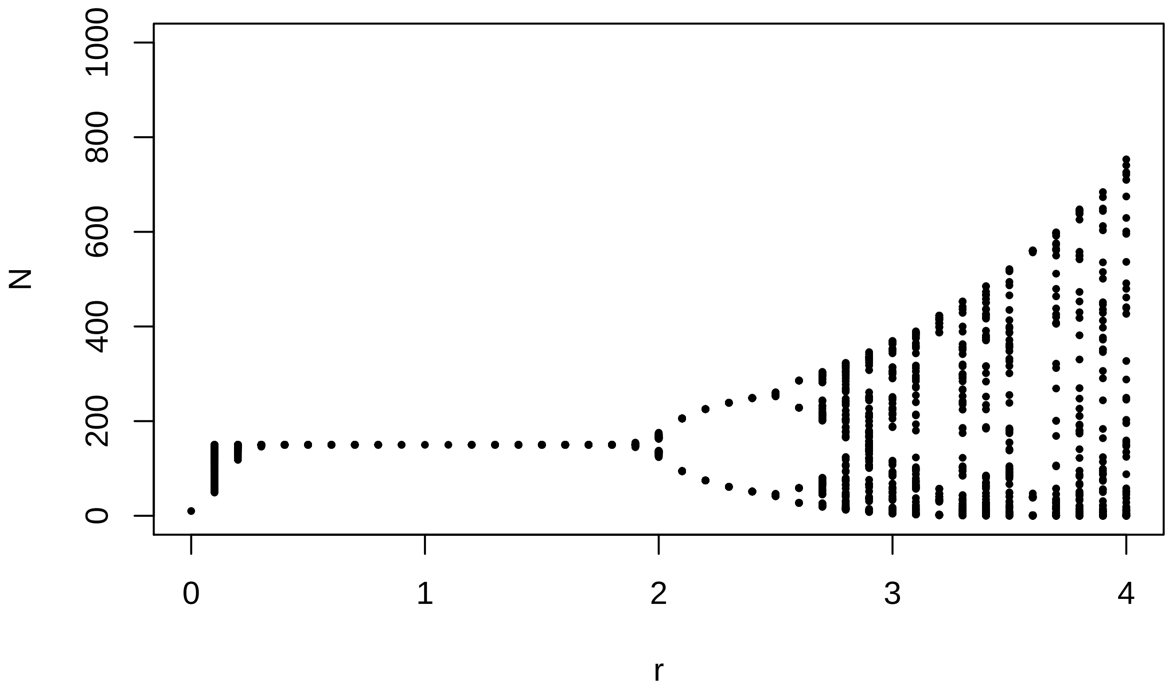
```
##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout
```

```r
x_coords = c(0, 1, 2, 3)
y_coords = c(0, 100, 300, 500, 700)
z_vals <- matrix(n_at_time_t, nrow = length(n_at_time_t)/2, ncol = length(n_at_time_t)/2)
sf_plot_sensitivity <- plot_ly(x = x_coords, y = y_coords, z = z_vals, type = "contour")
sf_plot_sensitivity
```



(h) (BONUS, OPTIONAL, PRETTY HARD) Write a script to make a bifurcation plot for this model, showing how its long-term dynamics change as the parameter r is varied. You will need to collect a set of values reflecting the long-term dynamics of N for each value of r, where r falls between 0 and 4. Plot these N-values as points on the y-axis, versus the corresponding value of r on the x-axis. Hint: you may need to look up the R command matplot.

```
r_bifur <- seq(0, 4, by = 0.1)
K_bifur <- 150
init_pop <- 10
time_to_run <- 100

nVec_bifur_set <- list()
plot(-1, -1, xlim = c(0, max(r_bifur)), ylim = c(0, 1000), xlab = "r", ylab = "N")
for(rr in 1:length(r_bifur)) {
  nVec_set <- vector()
  nVec_set[1] <- init_pop
  for(j in 1:time_to_run) {
    nVec_set[j + 1] <-  nVec_set[j]*exp(r_bifur[rr]*(1-(nVec_set[j]/K_bifur)))
  }
  values_to_plot <- unique(nVec_set[20:time_to_run])
  points(rep(r_bifur[rr], length(values_to_plot)), values_to_plot, cex = 0.4, pch = 19)
}
```


N
0
200
400
600
800
1000
0
1
2
3
4
r