STAT 302: Homework 6

Jaiden Atterbury

Due on 05-21-23

1. In this task, we will use a `for loop` to create a dataset and do some analysis.

- a) Create a data set with 4 variables `X, Y, Z, W`. Use a `for loop` for the variable `X` that takes the values `2,4,6,8,10,12,14`. Moreover, `Y=X+3, Z = (X-10)^2, W = Z-Y`. Print out the data set.

Below we will initialize an empty data frame and use a for loop to create the X, Y, Z, and W variables as well as print out the resulting data frame.

```
# Initialize the empty data set:
data <- data.frame(X=rep(NA, 7), Y=rep(NA, 7), Z=rep(NA, 7), W=rep(NA, 7))

# Initialize the index:
index <- 0

# Generate the data:
for (val in seq(2, 14, 2)) {
  # Increment the index:
  index <- index + 1

  # Assign the X value:
  data$X[index] <- val

  # Assign the Y value:
  data$Y[index] <- val + 3

  # Assign the Z value:
  data$Z[index] <- (val - 10)^2

  # Assign the W value:
  data$W[index] <- data$Z[index] - data$Y[index]
}

data
```

```
##    X  Y  Z   W
## 1  2  5 64  59
## 2  4  7 36  29
## 3  6  9 16   7
## 4  8 11  4  -7
## 5 10 13  0 -13
## 6 12 15  4 -11
## 7 14 17 16  -1
```

- b) Add a new variable `G` such that if `Z < 10`, `G=1`; if `Z` ≥ 10, `G = 2`. Print out the data set.

Below we will initialize an empty data frame and use a for loop to add a new variable G to the data set such that if $Z < 10,\ G = 1$; or else if $Z \geq 10,\ G = 2$

```
# Initialize the empty data set:
data <- data.frame(X=rep(NA, 7), Y=rep(NA, 7), Z=rep(NA, 7), W=rep(NA, 7),
                   G=rep(NA, 7))

# Initialize the index:
index <- 0

# Generate the data:
for (val in seq(2, 14, 2)) {
  # Increment the index:
  index <- index + 1

  # Assign the X value:
  data$X[index] <- val

  # Assign the Y value:
  data$Y[index] <- val + 3

  # Assign the Y value:
  data$Z[index] <- (val - 10)^2

  # Assign the W value:
  data$W[index] <- data$Z[index] - data$Y[index]

  # Assign the G value:
  if (data$Z[index] < 10) {
      data$G[index] <- 1
  } else {
      data$G[index] <- 2
  }
}

data
```

```
##    X  Y  Z   W G
## 1  2  5 64  59 2
## 2  4  7 36  29 2
## 3  6  9 16   7 2
## 4  8 11  4  -7 1
## 5 10 13  0 -13 1
## 6 12 15  4 -11 1
## 7 14 17 16  -1 2
```

- c) Print out the dataset for those observations with `W < 0` or `Z` ≤ 6.

Below we will conditionally print out the dataset for those observations where $W < 0$ or $Y \leq 6$.

```r
# Conditionally print out the data:
data_condition <- data %>%
  filter(W < 0 | Y <= 6)

# Display the data:
data_condition
```

```
##    X  Y  Z   W G
## 1  2  5 64  59 2
## 2  8 11  4  -7 1
## 3 10 13  0 -13 1
## 4 12 15  4 -11 1
## 5 14 17 16  -1 2
```

- d) Add another new variable `L`, which equals to the sum of the smallest and the second smallest values among the variable `X, Y, Z, W, G` for each observation. Print out the data set.

Below we will initialize an empty data frame and use a for loop to add another new variable `L`, which equals to the sum of the smallest and the second smallest values among the variable `X, Y, Z, W, G` for each observation, as well as print out the data set.

```r
# Initialize the empty data set
data <- data.frame(X=rep(NA, 7), Y=rep(NA, 7), Z=rep(NA, 7), W=rep(NA, 7),
                   G=rep(NA, 7), L=rep(NA, 7))

# Initialize the index:
index <- 0

# Generate the data:
for (val in seq(2, 14, 2)) {
  # Increment the index:
  index <- index + 1

  # Assign the X value:
  data$X[index] <- val

  # Assign the Y value:
  data$Y[index] <- val + 3

  # Assign the Y value:
  data$Z[index] <- (val - 10)^2

  # Assign the W value:
  data$W[index] <- data$Z[index] - data$Y[index]

  # Assign the G value:
  if (data$Z[index] < 10) {
      data$G[index] <- 1
  } else {
      data$G[index] <- 2
  }

  # Sort the data for the L variable:
```

```
  sorted = unlist(sort(data[index, 1:5]), use.names=FALSE)

  # Assign the L value:
  data$L[index] <- sorted[1] + sorted[2]
}

data
```

```
##    X  Y  Z   W G   L
## 1  2  5 64  59 2   4
## 2  4  7 36  29 2   6
## 3  6  9 16   7 2   8
## 4  8 11  4  -7 1  -6
## 5 10 13  0 -13 1 -13
## 6 12 15  4 -11 1 -10
## 7 14 17 16  -1 2   1
```

2. Write a function `funx` that takes a vector `x`, consisting of n positive components, and output the the difference between the mean of `x` (look at the documentation of `mean`) and the n^{th} root of the product of `x`, i.e, $(x_1 \cdot x_2 \cdot \cdots \cdot x_n)^{1/n}$, also called the geometric mean of the $x_1, \ldots, x_n$.

(a) Show your function through the use of an `echo=T` chunk.

```
funx <- function(x) {
  # Find the mean of x:
  xbar <- mean(x)

  # Find the geometric mean of x:
  n <- length(x)
  geom <- (prod(x))^(1/n)

  # Compute and return the difference
  diff <- xbar - geom

  return(diff)
}
```

(b) Execute the function for `x <- 1:6` and `x <- c(8,1,3,2,1,6)` and show your result.

Below we will test our function by executing the function for two different types of vector input.

Test 1:

```
# Set up test vector:
test_1 <- 1:6

# Test the function
funx(x=test_1)
```

```
## [1] 0.5062048
```

As can be seen from above, the difference between the mean and the geometric mean of the above vector is 0.5062048.

Test 2:

```
# Set up test vector:
test_2 <- c(8, 1, 3, 2, 1, 6)

# Test the function:
funx(x=test_2)
```

```
## [1] 0.9302034
```

As can be seen from above, the difference between the mean and the geometric mean of the above vector is 0.9302034.

(c) For what kind of vector `x` would you get 0 as a result?

The type of vectors `x` that would give us zero as a result are those types of vectors who have the same geometric mean as they do mean, in other words, vectors in which the mean equals the geometric mean. In particular, this occurs when all of the numbers in the vector are the same. For example take the vector `c(2, 2, 2)`. Then the function would be computing $\frac{2+2+2}{3} - (2 \cdot 2 \cdot 2)^{1/n} = 2 - 2 = 0$. In a more general sense, say we have a vector filled with the component y, that appears n times. Then the function would be computing $\frac{y+y+\cdots+y}{n} - (y \cdot y \cdot y \ldots y)^{1/n} = \frac{y \cdot n}{n} = (y^n)^{1/n} = y - y = 0$. Below we will show this property holds for our function.

```
# Set up test vector:
test_3 <- c(2, 2, 2)

# Test the function:
funx(x=test_3)
```

```
## [1] 0
```

3. The hard-threshold function is defined as

$$f_\lambda(x) = \begin{cases} x, & |x| \geq \lambda \\ 0, & |x| < \lambda \end{cases}$$

Write an R function that takes two parameters, a numeric vector input x and a threshold λ. Your function should return the value of $f_\lambda(x)$ and work for vector input x of any length. Also, set $\lambda = 4$, and demonstrate your function on the vector `c(-5, -3, 0, 3, 5)`.

Below we will write and display an R function that takes two parameters, a numeric vector input x and a threshold λ. This function will return the value of $f_\lambda(x)$ and work for vector input x of any length.

```
hard_threshold <- function(x, lambda) {
  # Create the output template:
  output <- vector(length = length(x))

  # Setup the index:
  index = 0
```

```
  # Loop through each vector element and apply the hard threshold function:
  for (value in x) {
    index = index + 1
    if (abs(value) >= lambda) {
      output[index] = value
    }
    else {
      output[index] = 0
    }
  }
  # Return the output vector:
  return(output)
}
```

Below we will test our above function for a certain combination of λ and vector x.

```
# Set up test vector:
test_4 <- c(-5, -3, 0, 3, 5)

# Test the function:
hard_threshold(x=test_4, lambda=4)
```

```
## [1] -5  0  0  0  5
```

As can be seen from the above test, we obtain the vector `c(-5, 0, 0, 0, 5)`, which is what we'd expect from a working hard-threshold function.

4. The soft-thereshold function is defined as

$$f_\lambda(x) = \begin{cases} sign(x)(|x| - \lambda), & |x| \geq \lambda \\ 0, & |x| < \lambda \end{cases}$$

Write an R function that takes two parameters, a numeric vector input x and a threshold λ. Here $sign(x)$ should return 1 if x is positive or 0 and should return -1 if x is negative. Your function should return the value of $g_\lambda(x)$ and work for vector input x of any length. Also, set $\lambda = 4$, and demonstrate your function on the vector `c(-5, -3, 0, 3, 5)`.

Below we will write and display an R function that takes two parameters, a numeric vector input x and a threshold λ. This function will return the value of $g_\lambda(x)$ and work for vector input x of any length.

```
soft_threshold <- function(x, lambda) {
  # Create the output template:
  output <- vector(length = length(x))

  # Setup the index:
  index = 0

  # Loop through each vector element and apply the soft threshold function:
  for (value in x) {
    index = index + 1
    if (abs(value) >= lambda) {
      if (value >= 0) {
        output[index] = abs(value) - lambda
```

```
      }
      else {
        output[index] = -(abs(value) - lambda)
      }
    }
    else {
      output[index] = 0
    }
  }
  # Return the output vector:
  return(output)
}
```

Below we will test our above function for a certain combination of λ and vector x.

```
# Set up test vector:
test_5 <- c(-5, -3, 0, 3, 5)

# Test the function:
soft_threshold(x=test_5, lambda=4)
```

```
## [1] -1  0  0  0  1
```

As can be seen from the above test, we obtain the vector `c(-1, 0, 0, 0, 1)`, which is what we'd expect from a working soft-threshold function.

5. Using the Handout 1.csv data set of Canvas. Run a **linear regression model using principal component analysis (PCA)**. Make a screeplot of your results with the function `screeplot()`. What is an appropriate number of principal components to analyze? Why?

Before we can compute a PCA of the commitment data set, we will need to remove any categorical variables.

```
# Select all of the quantitative data from the data set:
handout_1_new <- handout_1 %>%
  select(COMMIT, AGE, SALARY, CLASSSIZE, RESOURCES, AUTONOMY, CLIMATE, SUPPORT)
```

Since our dependent variable is `COMMIT` in this case, we will only find the PCA of the remaining quantitative variables. Next we will need to make sure all of these variables are on the same scale by standardizing them using the scale() function.

```
# Standardize the variables
standardized_commit <- as.data.frame(scale(handout_1_new[2:ncol(handout_1_new)]))
```

We will now perform a PCA on this standardized data set.

```
# Perform a PCA on the data:
commit_pca <- prcomp(standardized_commit)

# Get a summary of the PCA:
summary(commit_pca)
```

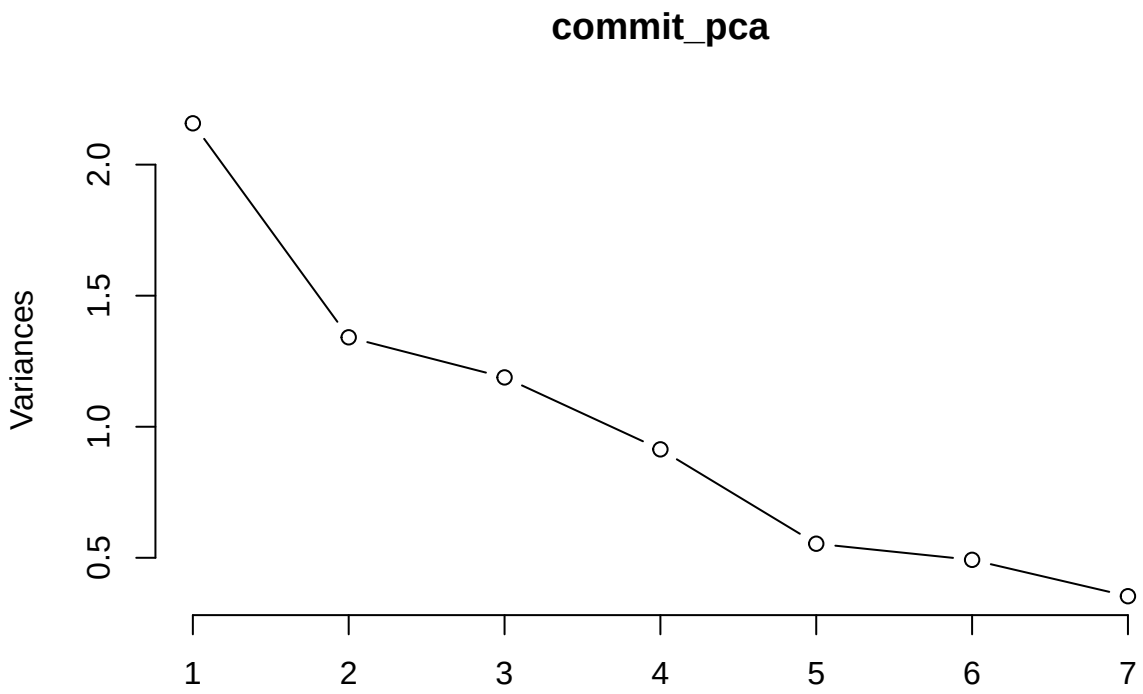
```
## Importance of components:
##                            PC1    PC2    PC3    PC4    PC5    PC6     PC7
## Standard deviation     1.4689 1.1581 1.0901 0.9559 0.7441 0.7015 0.59447
## Proportion of Variance 0.3082 0.1916 0.1698 0.1305 0.0791 0.0703 0.05048
## Cumulative Proportion  0.3082 0.4998 0.6696 0.8001 0.8792 0.9495 1.00000
```

Based on the above summary of the PCA, the first principal component explains around 31 of the variability in the data, the second principal component explains around 19 of the variability in the data, the third principal component explains around 17 of the variability in the data, the fourth principal component explains around 13 of the variability in the data, the fifth principal component explains around 8 of the variability in the data, the sixth principal component explains around 7 of the variability in the data, the seventh principal component explains around 5 of the variability in the data. It appears as if none of the singular components on their own capture the majority of the variability in the data.

In order to find the PCA equation we must use a scree plot and Kaiser's criterion to decide how many principal components to keep.

First we will create a scree plot.

```
# Plot a scree plot:
screeplot(commit_pca, type="lines")
```



As can be seen from the above scree plot, the slope changes significantly for the first three principal components, however, after that, the change in the slope of the line between the final 4 principal components does not change that much. Thus we will retain only the first three principal components.

Next we will use Kaiser's Criterion.

```r
# Calculate the squared standard deviations to use Kaiser's criterion:
(commit_pca$sdev)^2
```

```
## [1] 2.1577039 1.3412025 1.1882256 0.9136996 0.5536752 0.4920985 0.3533946
```

As computed above, the only principal components that have variances above 1 are principal components 1, 2, and 3. Thus under Kaiser's criterion we will only retain the first three principal components. This validates our conclusions from the above scree plot.

We will now run a linear model of `COMMIT` on the rest of the numeric variables in order to be able to create a PCA regression. But first we will check the normality of the dependent variable to assess the validity of the model. All tests will be run at the 5% level.

```r
# Test the normality of the dependent variable:
shapiro.test(handout_1_new$COMMIT)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  handout_1_new$COMMIT
## W = 0.98438, p-value = 0.08778
```

As can be seen from the above Shapiro-Wilk normality test, since the p-value is 0.08778 which is greater than 0.05, we fail to reject the null hypothesis at the 5% level. Thus we will conclude that the dependent variable is approximately normally distributed.

```r
# Create linear regression model:
model_1 <- lm(COMMIT ~ ., data=handout_1_new)

# Get the model summary:
summary(model_1)
```

```
##
## Call:
## lm(formula = COMMIT ~ ., data = handout_1_new)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.9378  -8.5427  -0.1178   8.0696  24.4704
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -18.06055   22.13236  -0.816 0.415853
## AGE           1.02408    0.41351   2.477 0.014442 *
## SALARY        0.41939    0.41824   1.003 0.317696
## CLASSSIZE    -1.10823    0.28821  -3.845 0.000181 ***
## RESOURCES     1.25011    0.66955   1.867 0.063952 .
## AUTONOMY      2.16240    0.54082   3.998 0.000102 ***
## CLIMATE       1.25741    0.41936   2.998 0.003205 **
## SUPPORT      -0.05835    0.59704  -0.098 0.922281
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 11.77 on 142 degrees of freedom
## Multiple R-squared:  0.4133, Adjusted R-squared:  0.3844
## F-statistic: 14.29 on 7 and 142 DF,  p-value: 5.653e-14
```

Checking for Autocorrelation:

```
dwtest(model_1)
```

```
##
##  Durbin-Watson test
##
## data:  model_1
## DW = 2.2821, p-value = 0.9553
## alternative hypothesis: true autocorrelation is greater than 0
```

As can be seen by the above Durbin-Watson test, the p-value is 0.9553, since the p-value is greater than 0.05, we fail to reject the null hypothesis and we see that there is no significant evidence that the residuals are auto-correlated.

Checking for Multicollinearity:

```
vif(model_1)
```

```
##       AGE    SALARY CLASSSIZE RESOURCES  AUTONOMY   CLIMATE   SUPPORT
##  1.099803  1.675339  1.710476  1.292500  1.360117  1.385461  1.289327
```

As can be seen by the above vif tests, the multicollinearity for each variable is below 5, hence we can assume that there is little to no multicollinearity between the variables, and hence we don't violate the assumption.

Checking for Linearity:

```
# Run a Rainbow test to check the linearity of the variables:
raintest(model_1)
```

```
##
##  Rainbow test
##
## data:  model_1
## Rain = 1.0065, df1 = 75, df2 = 67, p-value = 0.4909
```

As can be seen from the above Rainbow test, since the p-value is 0.4909 which is greater than 0.05, we fail to reject the null hypothesis and we assume that the relationship between the independent and dependent variables is linear.

Checking for Normality of the Residuals:

```
# Calculate the residuals:
resid_1 <- residuals(model_1)

# Run a Shapiro test to check the normality of the residuals:
shapiro.test(resid_1)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid_1
## W = 0.98869, p-value = 0.2667
```

As can be seen from the above Shapiro-Wilk test, since the p-value is 0.03035 which is greater than 0.05, we fail to reject the null hypothesis and we assume that the residuals are normally distributed.

Checking for equal variance:

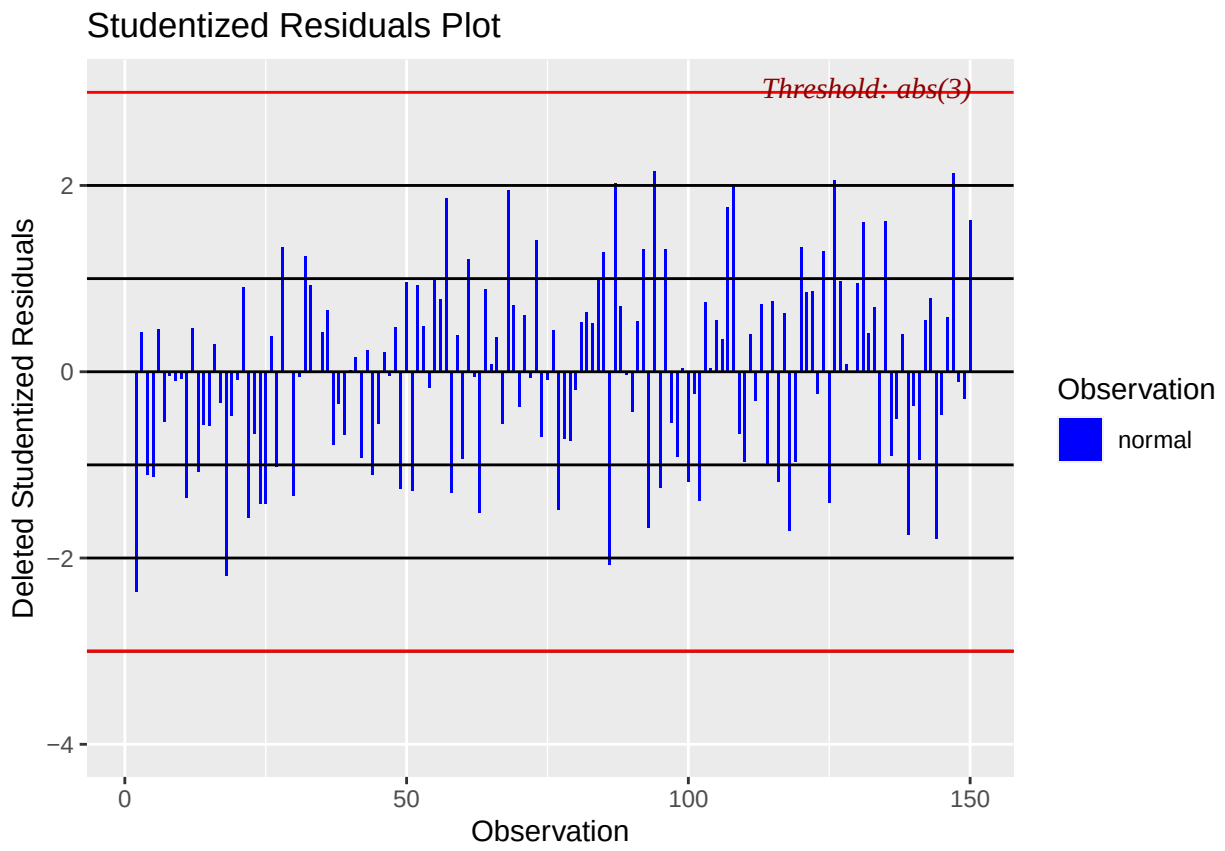
```
# Run the Breusch Pagan Test for Heteroskedasticity to test for equal variance:
ols_test_breusch_pagan(model_1)
```

```
##
##  Breusch Pagan Test for Heteroskedasticity
##  -----------------------------------------
##  Ho: the variance is constant
##  Ha: the variance is not constant
##
##                Data
##  ----------------------------------
##  Response : COMMIT
##  Variables: fitted values of COMMIT
##
##          Test Summary
##  -------------------------------
##  DF            =    1
##  Chi2          =    0.0001251535
##  Prob > Chi2   =    0.9910741
```

As can be seen from the above Breusch Pagan Test for Heteroskedasticity, since the p-value is 0.9910741 which is greater than 0.05, we fail to reject the null hypothesis, thus we have no significant evidence that the equal/constant variance assumption is violated.

Checking for no outliers:

```
# Create a Studentized Residual Plot to check for outliers:
ols_plot_resid_stud(model_1)
```



As can be seen from the above studentized residual plot, there are no extreme outliers present in the data.

Due to the fact that we violated none of the residual and model assumptions we could use this model for prediction, however, we will instead create an even better model using PCA.

For the completeness, the regression equation is $\widehat{\text{COMMIT}} = -18.061 + 1.024 \cdot \text{AGE} + 0.419 \cdot \text{SALARY} - 1.108 \cdot \text{CLASSSIZE} + 1.25 \cdot \text{RESOURCES} + 2.162 \cdot \text{AUTONOMY} + 1.257 \cdot \text{CLIMATE} - 0.058 \cdot \text{SUPPORT}$. However it is important to notes that only `AGE, CLASSSIZE, AUTONOMY, and CLIMATE` are significant predictors at the 5% level. If we were to extend the level of significance to 10% then `RESOURCES` would also be significant. The intercept, `SUPPORT`, and `SALARY` were all insignificant predictors.

Again, for completeness, we will calculate the corresponding 95% confidence intervals of the above estimates.

```
# Calculate the 95% confidence intervals of the above estimates:
confint(model_1)
```

```
##                   2.5 %     97.5 %
## (Intercept) -61.81203907 25.6909399
## AGE           0.20663482  1.8415168
## SALARY       -0.40740253  1.2461742
## CLASSSIZE    -1.67796581 -0.5385002
## RESOURCES    -0.07346877  2.5736970
## AUTONOMY      1.09330940  3.2314994
## CLIMATE       0.42841091  2.0864114
## SUPPORT      -1.23858204  1.1218805
```

All of the above intervals validate our significance summary from above.

Before we combine the multiple linear regression with the PCAs, lets take a look at all of our principal components, even though we are only using the first 3:

```
# Show the PCAs:
commit_pca
```

```
## Standard deviations (1, .., p=7):
## [1] 1.4689125 1.1581030 1.0900576 0.9558764 0.7440936 0.7014973 0.5944700
##
## Rotation (n x k) = (7 x 7):
##                    PC1         PC2        PC3          PC4        PC5
## AGE        -0.05510014  0.01546130 -0.7628124  0.449340530 -0.4542840
## SALARY     -0.54576108 -0.17550770 -0.1213570  0.044142806  0.4254346
## CLASSSIZE   0.48357651  0.39984827 -0.2168594 -0.084270970  0.2545694
## RESOURCES  -0.36829745 -0.08268648 -0.4730917 -0.541021013  0.2052094
## AUTONOMY   -0.08995096  0.76739547 -0.1406112 -0.001498058  0.2857285
## CLIMATE    -0.40249053  0.41071529  0.2291405 -0.339704406 -0.6287422
## SUPPORT    -0.39941178  0.21132784  0.2454975  0.617201620  0.1695804
##                    PC6         PC7
## AGE         0.068736884  0.04285460
## SALARY      0.458185701  0.51355339
## CLASSSIZE -0.275481840  0.64144345
## RESOURCES -0.508209563 -0.20152558
## AUTONOMY   0.354513738 -0.41942640
## CLIMATE   -0.009792302  0.32557217
## SUPPORT   -0.570436693  0.02117536
```

Now we will combine the multiple linear regression with the PCAs in order to create our PCA linear regression model.

```
# Combine the model and principal components:
combine <- cbind(handout_1_new, data.frame(commit_pca$x))

# Get a glimpse of the combined data:
head(combine)
```

```
##   COMMIT AGE SALARY CLASSSIZE RESOURCES AUTONOMY CLIMATE SUPPORT       PC1
## 1     48  30 42.277        26         5       12      12      12 0.9208188
## 2     20  30 40.147        21         5       11       9      11 1.4650172
## 3     40  26 41.826        28         7        9      10      12 1.2917874
## 4     20  27 37.000        22         7        5      10       8 2.5330493
## 5     26  24 39.032        25         7       10      11       9 1.9777338
## 6     48  30 40.785        23         5       10       9      13 1.1770219
##           PC2         PC3        PC4         PC5         PC6         PC7
## 1  1.37728799 -0.56849266  0.6832808 -0.11398965  0.4572366  0.61020695
## 2  0.10606881 -0.55430744  0.7887869 -0.24149762  0.5964013 -0.66008750
## 3  0.04956359  0.05411139 -0.5046159  0.98423584 -0.9754445  0.87405648
## 4 -2.14578511 -0.02917915 -1.6200780 -1.15822929 -0.7483287 -0.05789363
## 5  0.10221034  0.55573730 -1.9916102  0.41057217 -0.1717221 -0.19769819
## 6  0.11272160 -0.34400965  1.4335570  0.01335444 -0.2241809 -0.03228015
```

We will now run a linear model on the first three principal components in order to come up with a new regression model.

```r
# Create the PCA regression model:
model_2 <- lm(COMMIT ~ PC1 + PC2 + PC3, data=combine)

# Get the model summary:
summary(model_2)
```

```
##
## Call:
## lm(formula = COMMIT ~ PC1 + PC2 + PC3, data = combine)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -27.8802  -9.1871  -0.6789   8.3349  26.8099
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  50.0200     0.9877  50.642  < 2e-16 ***
## PC1          -5.6511     0.6747  -8.376 4.14e-14 ***
## PC2           2.5364     0.8557   2.964  0.00355 **
## PC3          -1.8576     0.9092  -2.043  0.04283 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.1 on 146 degrees of freedom
## Multiple R-squared:  0.3628, Adjusted R-squared:  0.3497
## F-statistic: 27.71 on 3 and 146 DF,  p-value: 3.052e-14
```

As can be seen above, all of the principal components and intercepts are significant at the 5% level. Below we will construct the PCA regression equation:

Our unsimplified regression equations is $\widehat{\text{COMMIT}} = 50.02 - 5.6511 \cdot \text{PC1} + 2.536 \cdot \text{PC2} - 1.858 \cdot \text{PC3}$.

Which turns into $\widehat{\text{COMMIT}} = 50.02 - 5.6511(-0.055 \cdot \text{AGE} - 0.546 \cdot \text{SALARY} + 0.486 \cdot \text{CLASSSIZE} - 0.368 \cdot \text{RESOURCES} - 0.09 \cdot \text{AUTONOMY} - 0.402 \cdot \text{CLIMATE} - 0.399 \cdot \text{SUPPORT}) + 2.536 \cdot (0.015 \cdot \text{AGE} - 0.176 \cdot \text{SALARY} + 0.4 \cdot \text{CLASSSIZE} - 0.083 \cdot \text{RESOURCES} - 0.767 \cdot \text{AUTONOMY} + 0.411 \cdot \text{CLIMATE} + 0.211 \cdot \text{SUPPORT}) - 1.858 \cdot (-0.763 \cdot \text{AGE} - 0.121 \cdot \text{SALARY} - 0.217 \cdot \text{CLASSSIZE} - 0.473 \cdot \text{RESOURCES} - 0.141 \cdot \text{AUTONOMY} + 0.229 \cdot \text{CLIMATE} + 0.246 \cdot \text{SUPPORT})$

Which simplifies to our final PCA regression equation of $\widehat{\text{COMMIT}} = -18.061 + 4.542 \cdot \text{AGE} + 2.842 \cdot \text{SALARY} - 1.309 \cdot \text{CLASSSIZE} + 2.733 \cdot \text{RESOURCES} - 1.178 \cdot \text{AUTONOMY} + 2.872 \cdot \text{CLIMATE} - 2.317 \cdot \text{SUPPORT}$

Since there are a lot of regression coefficients we will only interpret the coefficents in general for those that are positively and negativelly associated.

As computed above the independent variables that were positively associated in the PCA model were `AGE`, `SALARY`, `RESOURCES`, `CLIMATE`, and `SUPPORT`. Thus, holding all other variables constant, for a 1 unit increase in the previous variables the mean value of `COMMIT` will increase by 4.542, 2.842, 2.733, 2.872, 2.317 points respectively over the sampled range of `COMMIT` scores.

As computed above the independent variables that were negatively associated in the PCA model were `CLASSSIZE` and `AUTONOMY`. Thus, holding all other variables constant, for a 1 unite increase in the previous variables the mean value of `COMMIT` will decrease by 1.309 and 1.178 points respectively over the sampled range of `COMMIT` scores.

Lastly, as computed above, the PCA regression y-intercept was 50.02. This means that the estimated mean `COMMIT` score is equal to 50.02 when all other variables are zero.

6. Using the same data set from Homework 2. Run a principal component analysis (PCA) and explain an appropriate number of principal components to analyze.

Before we can compute a PCA of the board games data set, we will need to remove any categorical variables

```
# Select only the numeric variables from the board games dataset and remove all
# of the missing values:
board_games_new <- board_games %>%
  select(maxplayers, maxplaytime, minage, minplayers, minplaytime, playingtime,
         average_rating, users_rated, average_complexity) %>%
  drop_na()
```

Next we will need to make sure all of these variables are on the same scale by standardizing them using the scale() function.

```
# Standardize the variables
standardized_games <- as.data.frame(scale(board_games_new))
```

We will now perform a PCA on this standardized data set.

```
# Perform a PCA on the data:
games_pca <- prcomp(standardized_games)

# Get a summary of the PCA:
summary(games_pca)
```

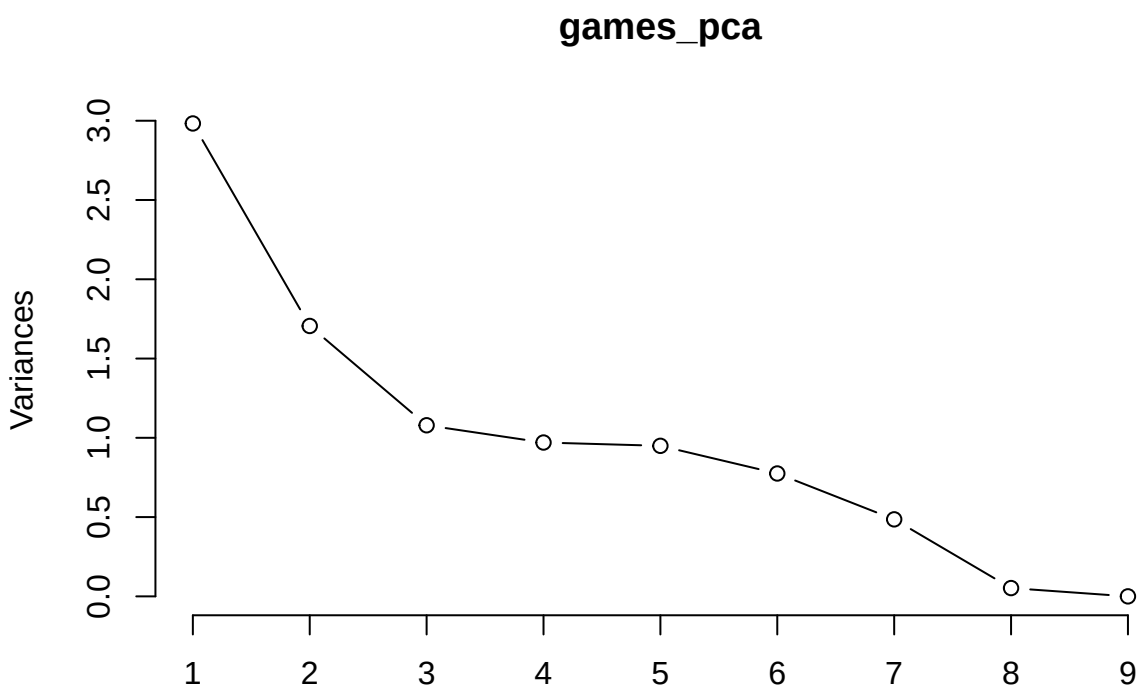
```
## Importance of components:
##                           PC1    PC2    PC3    PC4    PC5    PC6     PC7    PC8
## Standard deviation     1.7271 1.3059 1.0388 0.9851 0.9744 0.8803 0.69693 0.2284
## Proportion of Variance 0.3314 0.1895 0.1199 0.1078 0.1055 0.0861 0.05397 0.0058
## Cumulative Proportion  0.3314 0.5209 0.6408 0.7486 0.8541 0.9402 0.99420 1.0000
##                            PC9
## Standard deviation     1.147e-13
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

Based on the above summary of the PCA, the first principal component explains around 33 of the variability in the data, the second principal component explains around 19 of the variability in the data, the third principal component explains around 12 of the variability in the data, the fourth principal component explains around 11 of the variability in the data, the fifth principal component explains around 11 of the variability in the data, the sixth principal component explains around 9 of the variability in the data, the seventh principal component explains around 5 of the variability in the data, the sixth principal component explains around 9 of the variability in the data, the seventh principal component explains around 5 of the variability in the data, the eigth principal component explains around 0.1 of the variability in the data, the ninth principal component explains around 0 of the variability in the data. It appears as if none of the singular components on their own capture the majority of the variability in the data.

In order to find the PCA equation we must use a scree plot and Kaiser's criterion to decide how many principal components to keep.

First we will create a scree plot.

```r
# Plot a scree plot:
screeplot(games_pca, type="lines")
```



As can be seen from the above scree plot, the slope changes significantly for the first three principal components, however, after that, the change between the slope of the line between the final 6 principal components does not change that much. Thus we will retain only the first three principal components.

Next we will use Kaiser's Criterion.

```r
# Calculate the squared standard deviations to use Kaiser's criterion:
(games_pca$sdev)^2
```

```
## [1] 2.982762e+00 1.705417e+00 1.079187e+00 9.703517e-01 9.494656e-01
## [6] 7.749367e-01 4.857075e-01 5.217287e-02 1.315493e-26
```

As computed above, the only principal components that have variances above 1 are principal components 1, 2, and 3. Thus under Kaiser's criterion we will only retain the first three principal components. This validates our conclusions from the above scree plot.