Assignment 1 Running Data

Daniel Jackson

January 29th, 2024

Load libraries that will be used:

```r
library(readxl)
library(lubridate)
library(tidyverse)
library(dplyr)
library(caret)
library(stats)
library(corrplot)
library(ggplot2)
library(glmnet)
library(tree)
library(ipred)
library(randomForest)
library(fpc)
library(cluster)
```

Read in and clean data:

```r
# Read in data
setwd("/Users/doojerthekid/Documents/Merrimack Grad School Documents/DSE6620")
run_df = read_excel("Week_2/Run.xlsx")
dim(run_df)
```

```
## [1] 387  15
```

```r
head(run_df)
```

```
## # A tibble: 6 x 15
##   Date                Distance Elevation Minutes Heartrate Cadence  Temp Terrain
##   <dttm>                 <dbl>     <dbl>   <dbl>     <dbl>   <dbl> <dbl>   <dbl>
## 1 2023-11-20 00:00:00      2         141    23         148     158    68       0
## 2 2023-11-19 00:00:00      2.8      1824   108         136      NA    51       1
## 3 2023-11-17 00:00:00      6         856    79.5       151     142    63       1
## 4 2023-11-15 00:00:00      6.2       105    59.5       176     165    35       0
## 5 2023-11-13 00:00:00      3.7       118    35.5       174     167    33       0
## 6 2023-11-11 00:00:00      4          82    38         174     169    45       0
## # i 7 more variables: PE <dbl>, Sneaker <chr>, Area <chr>, Pace <dbl>,
## #   Speed <dbl>, Elev_Per_Mile <dbl>, Notes <chr>
```

```r
# 387 rows
# 15 columns
# Rows after row 361 are all NA
# Subset data frame
run_df = run_df[1:361,]

# Change date column to date structure
str(run_df$Date)
```

```
##  POSIXct[1:361], format: "2023-11-20" "2023-11-19" "2023-11-17" "2023-11-15" "2023-11-13" ...
```

```r
run_df$Date = as.Date(run_df$Date,format = '%Y-%m-%d')

# Create a year column
run_df = run_df %>%
  mutate(Year = year(Date))

# Convert Terrain column to a factor variable
run_df$Terrain = as.factor(run_df$Terrain)
head(run_df)
```

```
## # A tibble: 6 x 16
##   Date       Distance Elevation Minutes Heartrate Cadence  Temp Terrain    PE
##   <date>        <dbl>     <dbl>   <dbl>     <dbl>   <dbl> <dbl> <fct>   <dbl>
## 1 2023-11-20      2         141    23         148     158    68 0           2
## 2 2023-11-19      2.8      1824   108         136      NA    51 1           4
## 3 2023-11-17      6         856    79.5       151     142    63 1           3
## 4 2023-11-15      6.2       105    59.5       176     165    35 0           4
## 5 2023-11-13      3.7       118    35.5       174     167    33 0           3
## 6 2023-11-11      4          82    38         174     169    45 0           4
## # i 7 more variables: Sneaker <chr>, Area <chr>, Pace <dbl>, Speed <dbl>,
## #   Elev_Per_Mile <dbl>, Notes <chr>, Year <dbl>
```

```r
# View race data
race_df = run_df %>%
  filter(!Area %in% "Treadmill") %>%
  select(Date:Terrain, Pace:Elev_Per_Mile, PE:Year, Notes)
dim(race_df)
```

```
## [1] 300  16
```

```r
# 300 observations

# Check for missing values
colSums(is.na(race_df))
```

```
##          Date      Distance     Elevation       Minutes     Heartrate
##             0             0             0             1            34
##       Cadence          Temp       Terrain          Pace         Speed
##            38            35             0             1             1
## Elev_Per_Mile            PE       Sneaker          Area         Notes
```

```
##              0            182               0              0            281
##            Year
##              0
```

```r
# Check for NA by year
result_df = race_df %>%
  group_by(Year) %>%
  summarize_all(list(~ sum(is.na(.))))
head(result_df)
```

```
## # A tibble: 3 x 16
##    Year  Date Distance Elevation Minutes Heartrate Cadence  Temp Terrain  Pace
##   <dbl> <int>    <int>     <int>   <int>     <int>   <int> <int>   <int> <int>
## 1  2021     0        0         0       0        22      22    22       0     0
## 2  2022     0        0         0       0        12      13    12       0     0
## 3  2023     0        0         0       1         0       3     1       0     1
## # i 6 more variables: Speed <int>, Elev_Per_Mile <int>, PE <int>,
## #   Sneaker <int>, Area <int>, Notes <int>
```

```r
# Remove PE, Notes, Sneaker, Area
race_df = race_df %>%
  select(-c(Notes, PE, Sneaker, Area)) %>%
  na.omit()
```

Question 1

What is the relationship between Heartrate, Cadence and elevation?

```r
# Correlation between numeric variables
cor_df = race_df %>%
  select_if(is.numeric)
cor(cor_df)
```

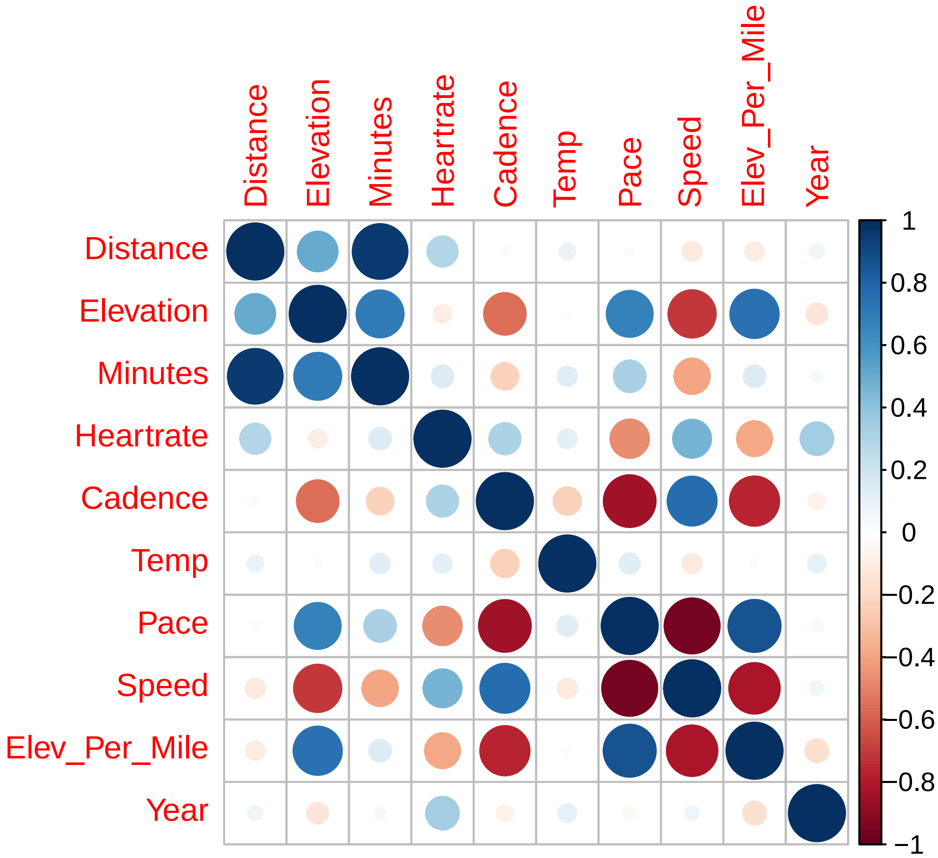
```
##                  Distance   Elevation     Minutes  Heartrate     Cadence
## Distance       1.00000000  0.50303453  0.94826296  0.2926372  0.02156308
## Elevation      0.50303453  1.00000000  0.69535383 -0.1011141 -0.55010785
## Minutes        0.94826296  0.69535383  1.00000000  0.1547401 -0.22509509
## Heartrate      0.29263717 -0.10111407  0.15474011  1.0000000  0.31129360
## Cadence        0.02156308 -0.55010785 -0.22509509  0.3112936  1.00000000
## Temp           0.08022194 -0.01682557  0.12202646  0.1064639 -0.24284101
## Pace           0.01990837  0.66837284  0.31562573 -0.4738925 -0.84988365
## Speed         -0.11935781 -0.70832431 -0.40382856  0.4572067  0.76383548
## Elev_Per_Mile -0.11021304  0.73875751  0.14851138 -0.3933806 -0.77015871
## Year           0.05951672 -0.13506148  0.04334407  0.3383927 -0.07567415
##                      Temp        Pace       Speed Elev_Per_Mile        Year
## Distance       0.08022194  0.01990837 -0.11935781   -0.11021304  0.05951672
## Elevation     -0.01682557  0.66837284 -0.70832431    0.73875751 -0.13506148
## Minutes        0.12202646  0.31562573 -0.40382856    0.14851138  0.04334407
## Heartrate      0.10646389 -0.47389246  0.45720668   -0.39338062  0.33839269
## Cadence       -0.24284101 -0.84988365  0.76383548   -0.77015871 -0.07567415
## Temp           1.00000000  0.13165820 -0.11871008   -0.01513362  0.10340006
```

```
## Pace           0.13165820  1.00000000 -0.95856396    0.85543936 -0.04286603
## Speed         -0.11871008 -0.95856396  1.00000000   -0.81121639  0.05541841
## Elev_Per_Mile -0.01513362  0.85543936 -0.81121639    1.00000000 -0.17319238
## Year           0.10340006 -0.04286603  0.05541841   -0.17319238  1.00000000
```

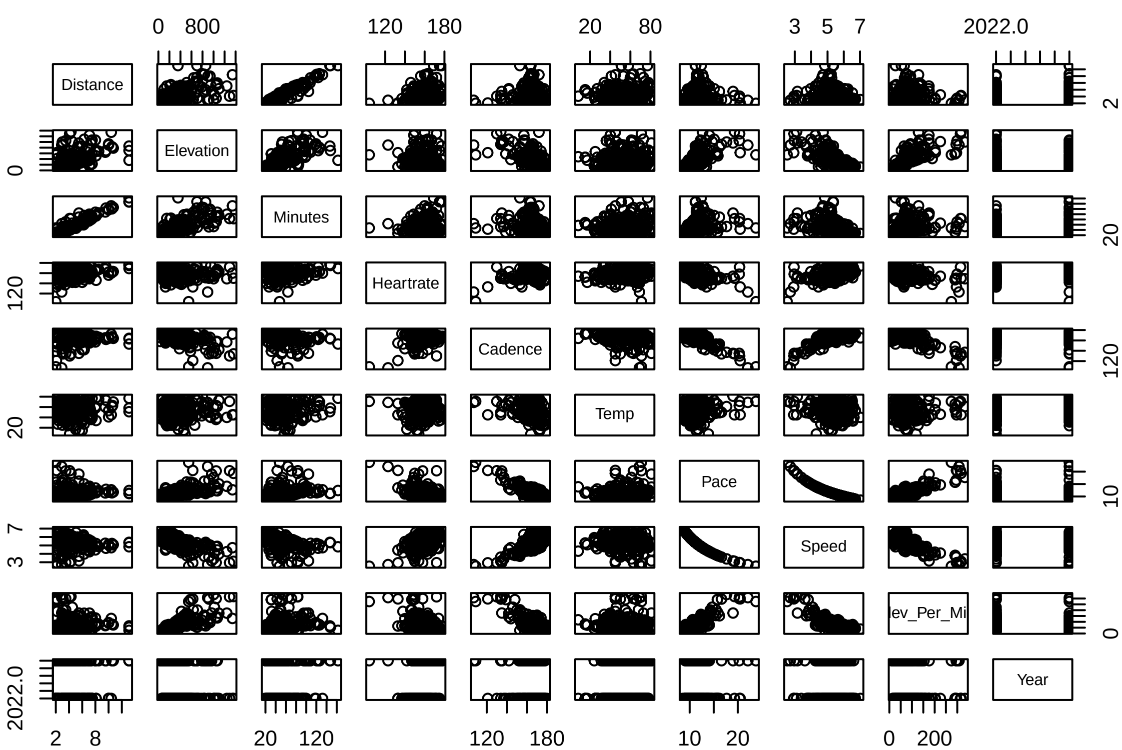
```
c = round(cor(cor_df), digits = 2)
cor(cor_df)
```

```
##                  Distance   Elevation     Minutes  Heartrate     Cadence
## Distance       1.00000000  0.50303453  0.94826296  0.2926372  0.02156308
## Elevation      0.50303453  1.00000000  0.69535383 -0.1011141 -0.55010785
## Minutes        0.94826296  0.69535383  1.00000000  0.1547401 -0.22509509
## Heartrate      0.29263717 -0.10111407  0.15474011  1.0000000  0.31129360
## Cadence        0.02156308 -0.55010785 -0.22509509  0.3112936  1.00000000
## Temp           0.08022194 -0.01682557  0.12202646  0.1064639 -0.24284101
## Pace           0.01990837  0.66837284  0.31562573 -0.4738925 -0.84988365
## Speed         -0.11935781 -0.70832431 -0.40382856  0.4572067  0.76383548
## Elev_Per_Mile -0.11021304  0.73875751  0.14851138 -0.3933806 -0.77015871
## Year           0.05951672 -0.13506148  0.04334407  0.3383927 -0.07567415
##                      Temp        Pace       Speed Elev_Per_Mile        Year
## Distance       0.08022194  0.01990837 -0.11935781   -0.11021304  0.05951672
## Elevation     -0.01682557  0.66837284 -0.70832431    0.73875751 -0.13506148
## Minutes        0.12202646  0.31562573 -0.40382856    0.14851138  0.04334407
## Heartrate      0.10646389 -0.47389246  0.45720668   -0.39338062  0.33839269
## Cadence       -0.24284101 -0.84988365  0.76383548   -0.77015871 -0.07567415
## Temp           1.00000000  0.13165820 -0.11871008   -0.01513362  0.10340006
## Pace           0.13165820  1.00000000 -0.95856396    0.85543936 -0.04286603
## Speed         -0.11871008 -0.95856396  1.00000000   -0.81121639  0.05541841
## Elev_Per_Mile -0.01513362  0.85543936 -0.81121639    1.00000000 -0.17319238
## Year           0.10340006 -0.04286603  0.05541841   -0.17319238  1.00000000
```

```
# Correlation plot
corrplot(c)
```




```
pairs(cor_df)
```

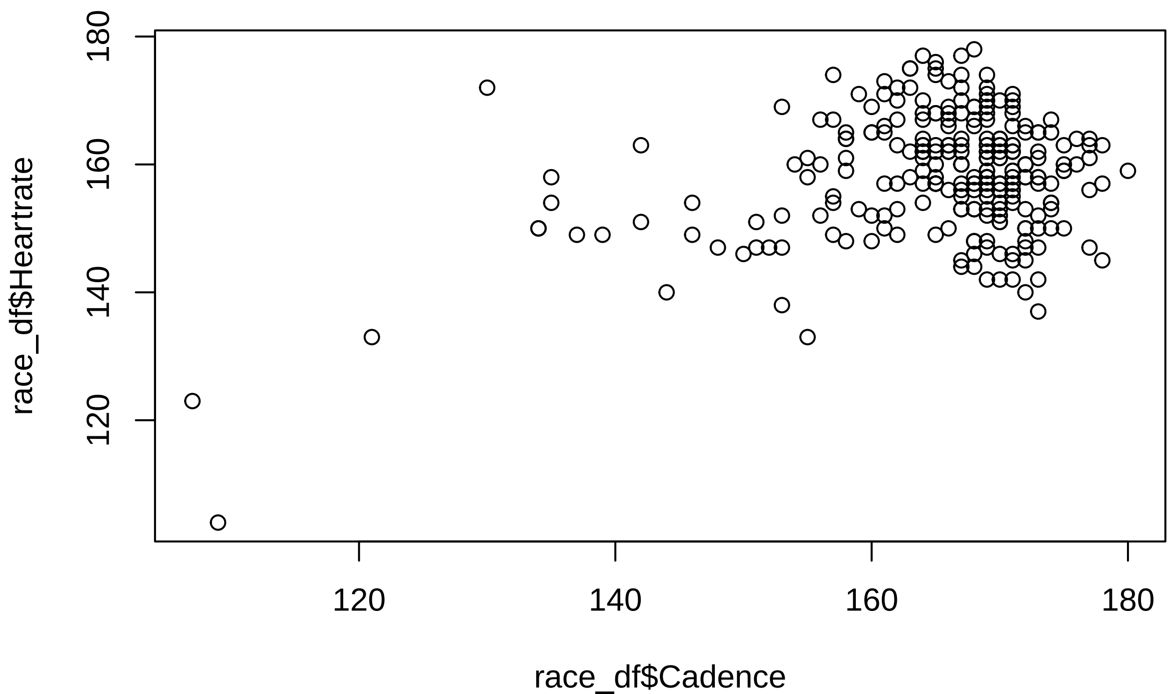


Looking at correlation plot, Heartrate and cadence have 0.31 correlation. This moderate positive correlation means has

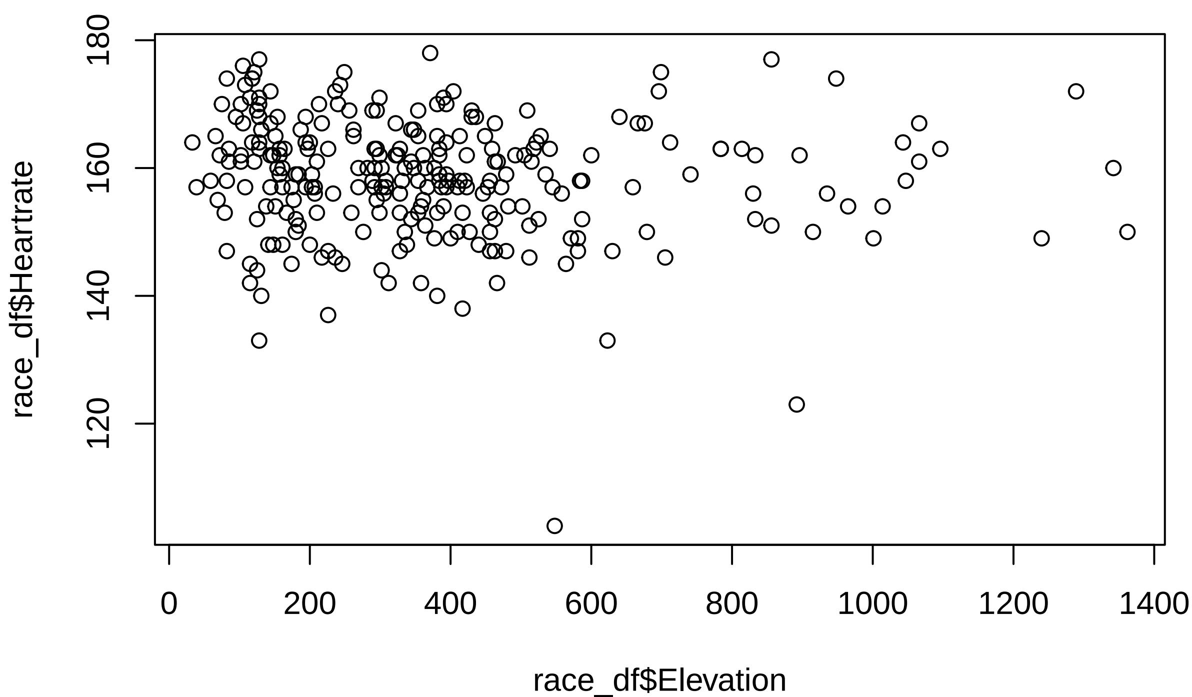
heart rate increases, cadence also increases.
Heartrate and elevation have -0.10 correlation. This negative relationship means that as elevation increases, heartrate slightly decreases.
Cadence and elevation have -0.55 correlation. As elevation increases, cadence decreases.

Let's look at some plots.

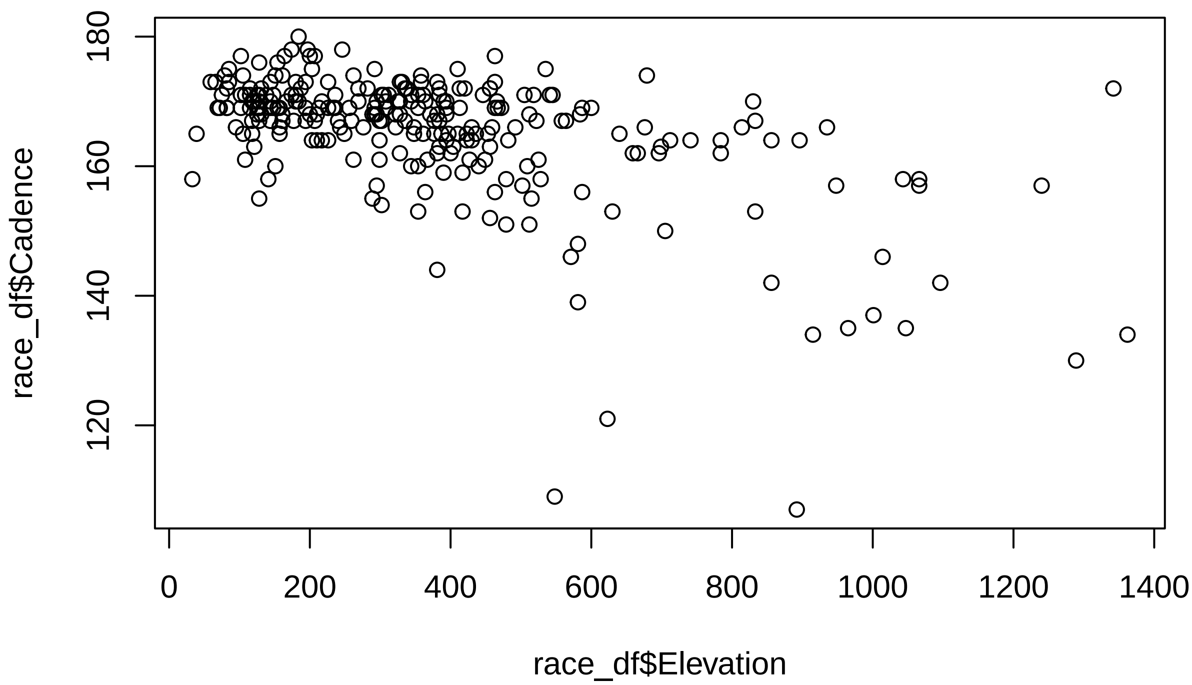
```
plot(race_df$Cadence, race_df$Heartrate)
```




```
plot(race_df$Elevation, race_df$Heartrate)
```

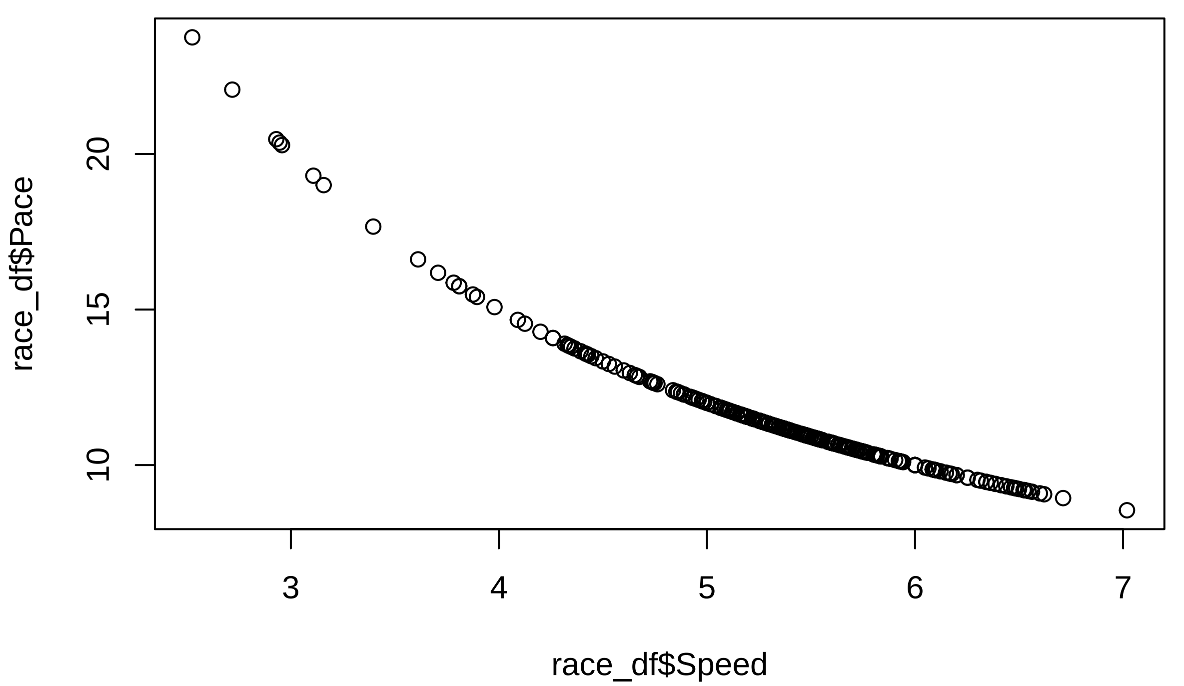


```r
plot(race_df$Elevation, race_df$Cadence)
```



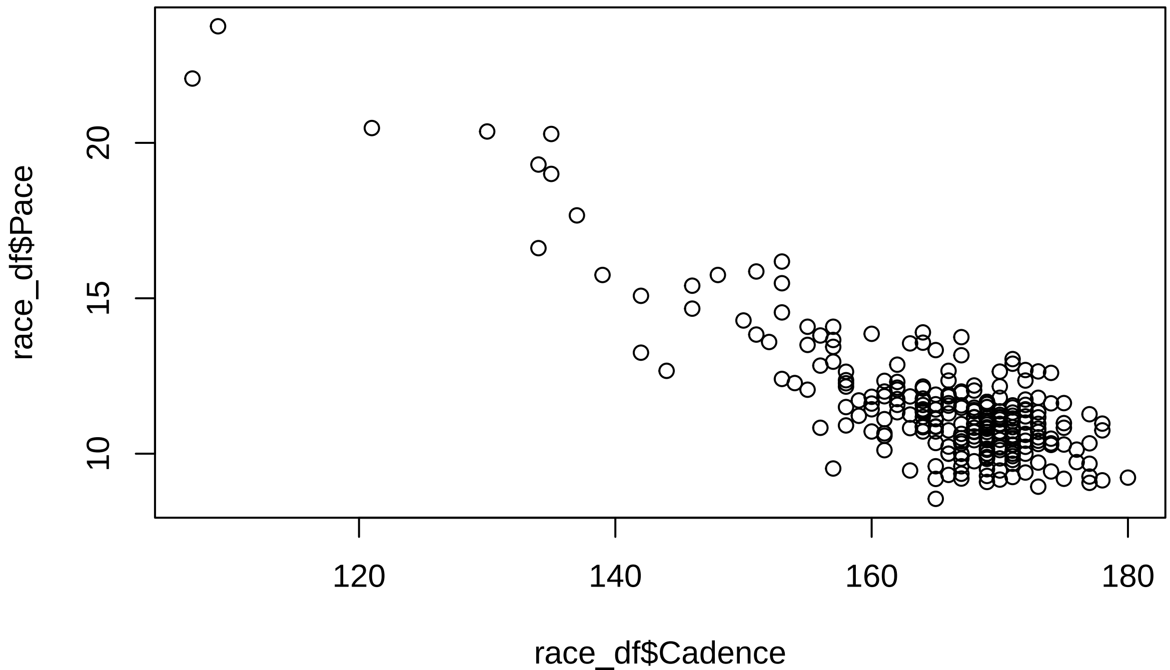
Pace and speed seem to have highest absolute correlation value.Pace and speed have a correlation value of -0.95. That means they are almost perfectly negatively correlated. As speed increases, pace decreases.

```r
plot(race_df$Speed, race_df$Pace)
```



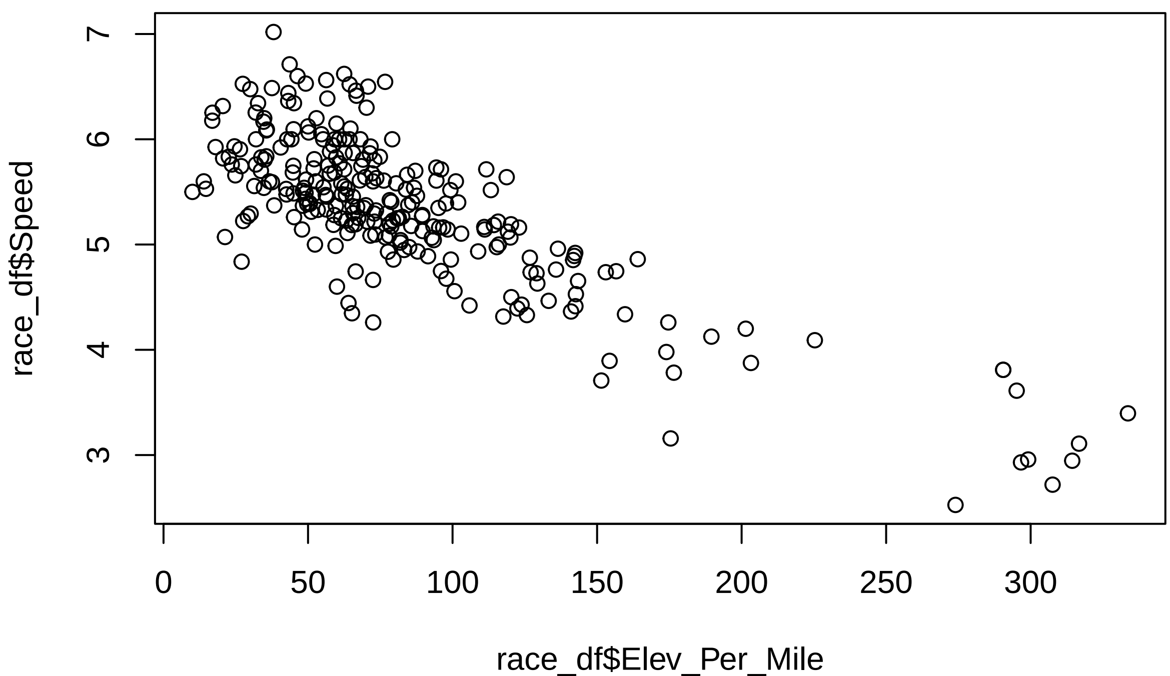
Pace and cadence also have a high absolute correlation value. The value is -0.85. This Means that as cadence increases, pace decreases.

```r
plot(race_df$Cadence, race_df$Pace)
```



Speed and elevation per mile have high absolute correlation value. The value is -0.81. As elevation per mile increases, speed decreases.

```r
plot(race_df$Elev_Per_Mile, race_df$Speed)
```



Question 2

Improve the accuracy of the current model on pace.
Linear regression:

```
run_lm = lm(Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate + Cadence
            + Temp, race_df)
summary(run_lm)
```

```
##
## Call:
## lm(formula = Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate +
##     Cadence + Temp, data = race_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1587 -0.4468 -0.0470  0.5235  3.5193
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 35.062993   1.840668  19.049  < 2e-16 ***
## Distance                     0.123707   0.028804   4.295 2.49e-05 ***
## Heartrate                   -0.048834   0.006481  -7.535 8.52e-13 ***
## Cadence                     -0.119060   0.007719 -15.424  < 2e-16 ***
## Temp                         0.003416   0.004065   0.841    0.401
## log(Elev_Per_Mile):Terrain0  0.693347   0.129355   5.360 1.86e-07 ***
## log(Elev_Per_Mile):Terrain1  0.853993   0.118337   7.217 6.09e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8673 on 255 degrees of freedom
## Multiple R-squared:  0.8372, Adjusted R-squared:  0.8334
## F-statistic: 218.6 on 6 and 255 DF,  p-value: < 2.2e-16
```

Using a null hypothesis that none of the predictors are significant of pace, we fail to reject that hypothesis for any predictor with a p-value less than 0.05. Let's remove the predictors with p-values greater than 0.05. Temp is the only predictor that does not have statistical significance in linear model.

```
run_lm = lm(Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate + Cadence, race_df)
summary(run_lm)
```

```
##
## Call:
## lm(formula = Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate +
##     Cadence, data = race_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1198 -0.4452 -0.0548  0.5026  3.5260
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)                  35.605325   1.722883  20.666  < 2e-16 ***
## Distance                      0.124473   0.028774   4.326 2.18e-05 ***
## Heartrate                    -0.048129   0.006423  -7.493 1.09e-12 ***
## Cadence                      -0.121483   0.007157 -16.975  < 2e-16 ***
## log(Elev_Per_Mile):Terrain0  0.676706   0.127758   5.297 2.54e-07 ***
## log(Elev_Per_Mile):Terrain1  0.833838   0.115815   7.200 6.70e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8668 on 256 degrees of freedom
## Multiple R-squared:  0.8368, Adjusted R-squared:  0.8336
## F-statistic: 262.5 on 5 and 256 DF,  p-value: < 2.2e-16
```

Let's create training and test data from 262 observations. Train linear model using training data and predict test data.

```r
262 / 2
```

```
## [1] 131
```

```r
# Returns 131
set.seed(1)
train = sample(1:nrow(race_df), 131)
race_train = race_df[train,]
race_test = race_df[-train,]
```

```r
train_race_lm = lm(Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate +
                     Cadence, race_train)
summary(train_race_lm)
```

```
##
## Call:
## lm(formula = Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate +
##     Cadence, data = race_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.93655 -0.46576 -0.01254  0.50250  2.91843
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  33.498217   3.136029  10.682  < 2e-16 ***
## Distance                      0.120789   0.039705   3.042 0.002863 **
## Heartrate                    -0.035355   0.009857  -3.587 0.000478 ***
## Cadence                      -0.122279   0.011587 -10.553  < 2e-16 ***
## log(Elev_Per_Mile):Terrain0  0.736777   0.209581   3.515 0.000612 ***
## log(Elev_Per_Mile):Terrain1  0.944478   0.190760   4.951 2.34e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8398 on 125 degrees of freedom
## Multiple R-squared:  0.8091, Adjusted R-squared:  0.8014
## F-statistic: 105.9 on 5 and 125 DF,  p-value: < 2.2e-16
```

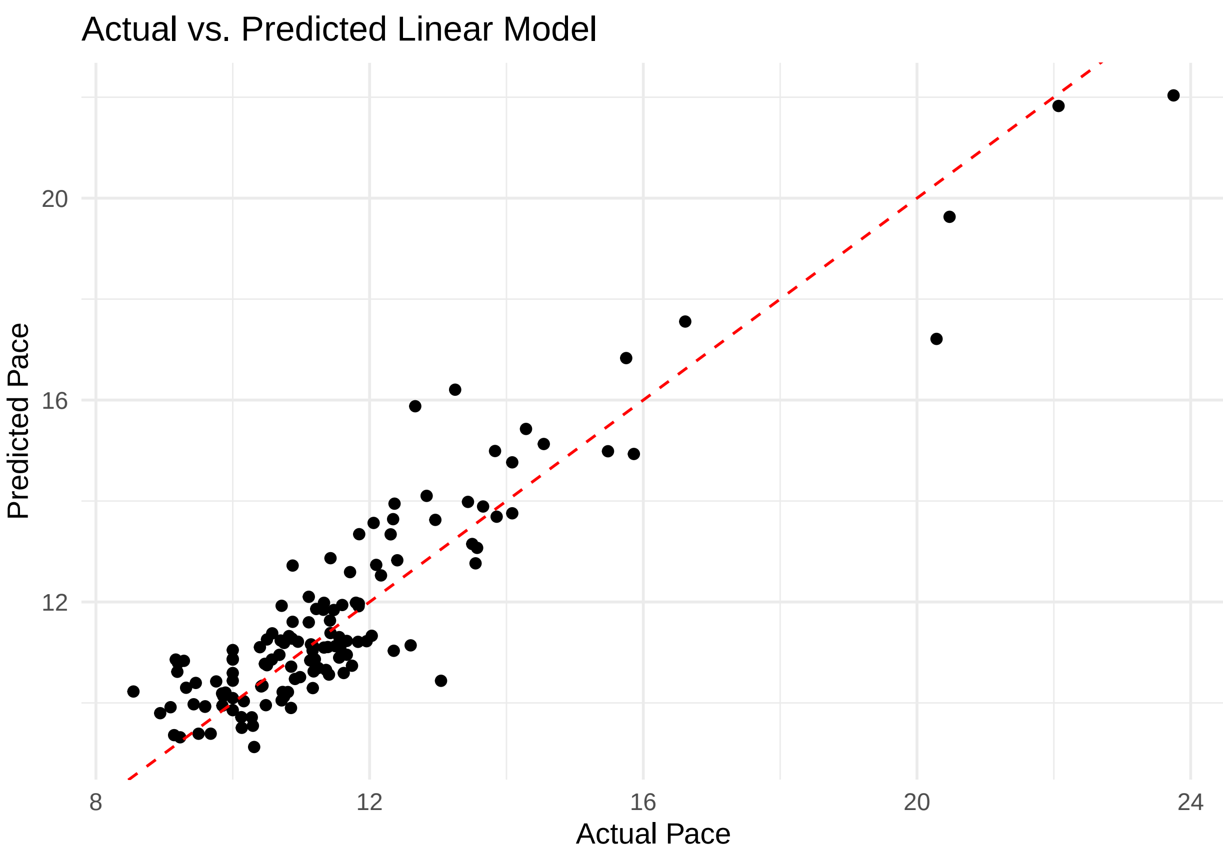
All p-values are less than 0.05.
Let's predict test data using mode.

```
pred_race_lm = predict(train_race_lm, newdata = race_test)
mean((pred_race_lm - race_test$Pace)^2)
```

```
## [1] 0.8537969
```

Test error rate is 0.85. Prediction rate of approx 15%.
Create a scatter plot comparing actual vs. predicted values.

```
par(mfrow = c(1, 1))
plot_df = data.frame(Actual = race_test$Pace, Predicted = pred_race_lm)
ggplot(plot_df, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +  # Adds a line of perfect
  labs(x = "Actual Pace",
       y = "Predicted Pace",
       title = "Actual vs. Predicted Linear Model") +
  theme_minimal()
```



Try some transformations: Try sqrt() of predictors.

```
sqrt_race_lm = lm(Pace ~ sqrt(Distance) + log(Elev_Per_Mile):Terrain +
                    sqrt(Heartrate) + sqrt(Cadence), race_train)
summary(sqrt_race_lm)
```

```
##
## Call:
## lm(formula = Pace ~ sqrt(Distance) + log(Elev_Per_Mile):Terrain +
##     sqrt(Heartrate) + sqrt(Cadence), data = race_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.93909 -0.48923 -0.00595  0.48284  2.74458
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  58.4602     5.4646  10.698  < 2e-16 ***
## sqrt(Distance)                0.6134     0.1850   3.315 0.001198 **
## sqrt(Heartrate)              -0.8974     0.2406  -3.729 0.000290 ***
## sqrt(Cadence)                -3.1204     0.2841 -10.983  < 2e-16 ***
## log(Elev_Per_Mile):Terrain0   0.7093     0.2051   3.458 0.000744 ***
## log(Elev_Per_Mile):Terrain1   0.9157     0.1868   4.902 2.89e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8222 on 125 degrees of freedom
## Multiple R-squared:  0.817,  Adjusted R-squared:  0.8097
## F-statistic: 111.6 on 5 and 125 DF,  p-value: < 2.2e-16
```

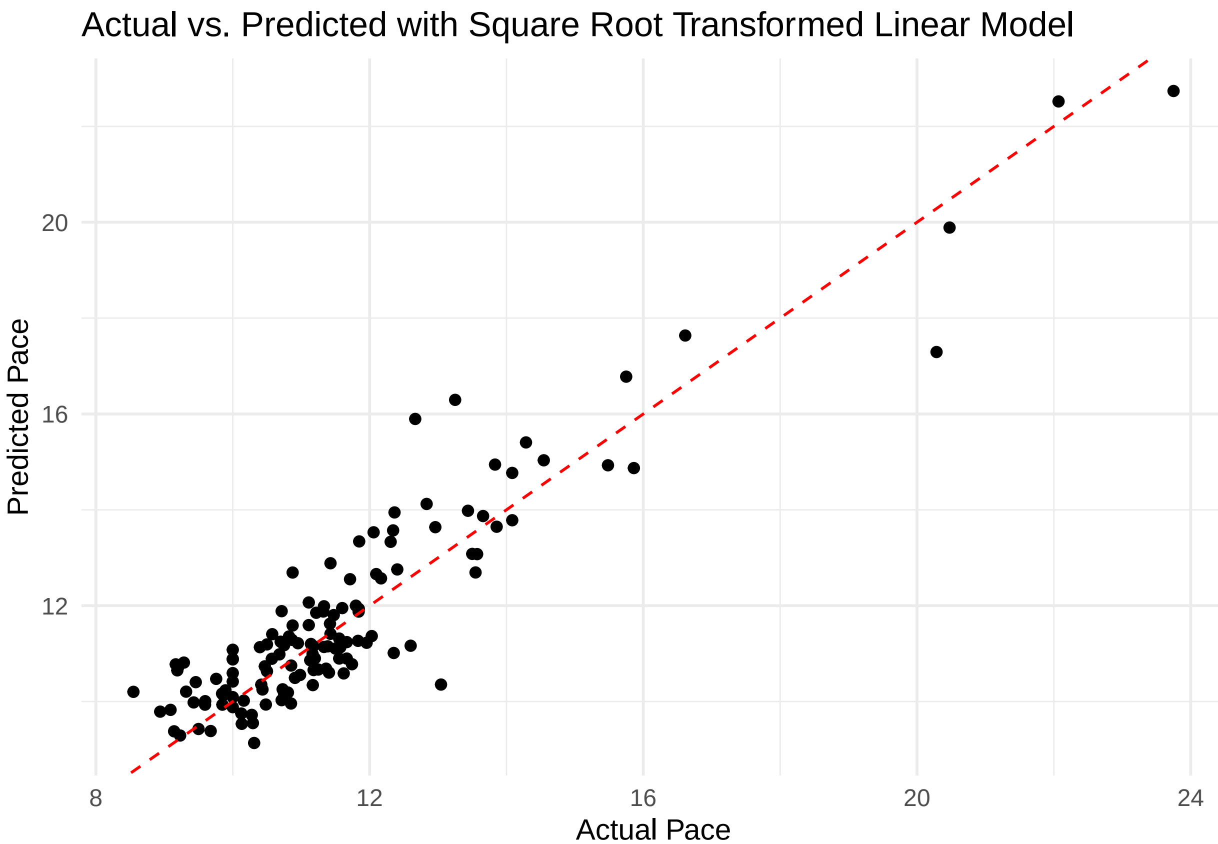
Predictors pass null hypothesis test.
Predict test data.

```r
pred_sqrt_race_lm = predict(sqrt_race_lm, newdata = race_test)
mean((pred_sqrt_race_lm - race_test$Pace)^2)
```

```
## [1] 0.827948
```

Test error rate of 83%. Prediction rate of approx 17%.
Create a scatter plot comparing actual vs. predicted values.

```r
par(mfrow = c(1, 1))
plot_df = data.frame(Actual = race_test$Pace, Predicted = pred_sqrt_race_lm)
ggplot(plot_df, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +  # Adds a line of perfect
  labs(x = "Actual Pace",
       y = "Predicted Pace",
       title = "Actual vs. Predicted with Square Root Transformed Linear Model") +
  theme_minimal()
```



Try squared transformation:

```
sq_race_lm = lm(Pace ~ (Distance)^2 + log(Elev_Per_Mile):Terrain +
                    (Heartrate)^2 + (Cadence)^2, race_train)
summary(sq_race_lm)
```

```
##
## Call:
## lm(formula = Pace ~ (Distance)^2 + log(Elev_Per_Mile):Terrain +
##     (Heartrate)^2 + (Cadence)^2, data = race_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.93655 -0.46576 -0.01254  0.50250  2.91843
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 33.498217   3.136029  10.682  < 2e-16 ***
## Distance                     0.120789   0.039705   3.042 0.002863 **
## Heartrate                   -0.035355   0.009857  -3.587 0.000478 ***
## Cadence                     -0.122279   0.011587 -10.553  < 2e-16 ***
## log(Elev_Per_Mile):Terrain0  0.736777   0.209581   3.515 0.000612 ***
## log(Elev_Per_Mile):Terrain1  0.944478   0.190760   4.951 2.34e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8398 on 125 degrees of freedom
```

```
## Multiple R-squared:  0.8091, Adjusted R-squared:  0.8014
## F-statistic: 105.9 on 5 and 125 DF,  p-value: < 2.2e-16
```

All predictors pass null hypothesis test.

```r
pred_sq_race_lm = predict(sq_race_lm, newdata = race_test)
mean((pred_sq_race_lm - race_test$Pace)^2)
```

```
## [1] 0.8537969
```

Test error rate of approx 85%.

Try log transformation on predictors:

```r
log_race_lm = lm(Pace ~ log(Distance) + log(Elev_Per_Mile):Terrain +
                   log(Heartrate) + log(Cadence), race_train)
summary(log_race_lm)
```

```
##
## Call:
## lm(formula = Pace ~ log(Distance) + log(Elev_Per_Mile):Terrain +
##     log(Heartrate) + log(Cadence), data = race_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.94820 -0.43739 -0.02266  0.45490  2.54634
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 137.3527    12.6765  10.835  < 2e-16 ***
## log(Distance)                 0.7137     0.2029   3.518 0.000608 ***
## log(Heartrate)               -5.6295     1.4685  -3.833 0.000199 ***
## log(Cadence)                -19.8686     1.7402 -11.417  < 2e-16 ***
## log(Elev_Per_Mile):Terrain0   0.6933     0.2006   3.457 0.000748 ***
## log(Elev_Per_Mile):Terrain1   0.8953     0.1829   4.895 2.97e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.806 on 125 degrees of freedom
## Multiple R-squared:  0.8241, Adjusted R-squared:  0.8171
## F-statistic: 117.2 on 5 and 125 DF,  p-value: < 2.2e-16
```

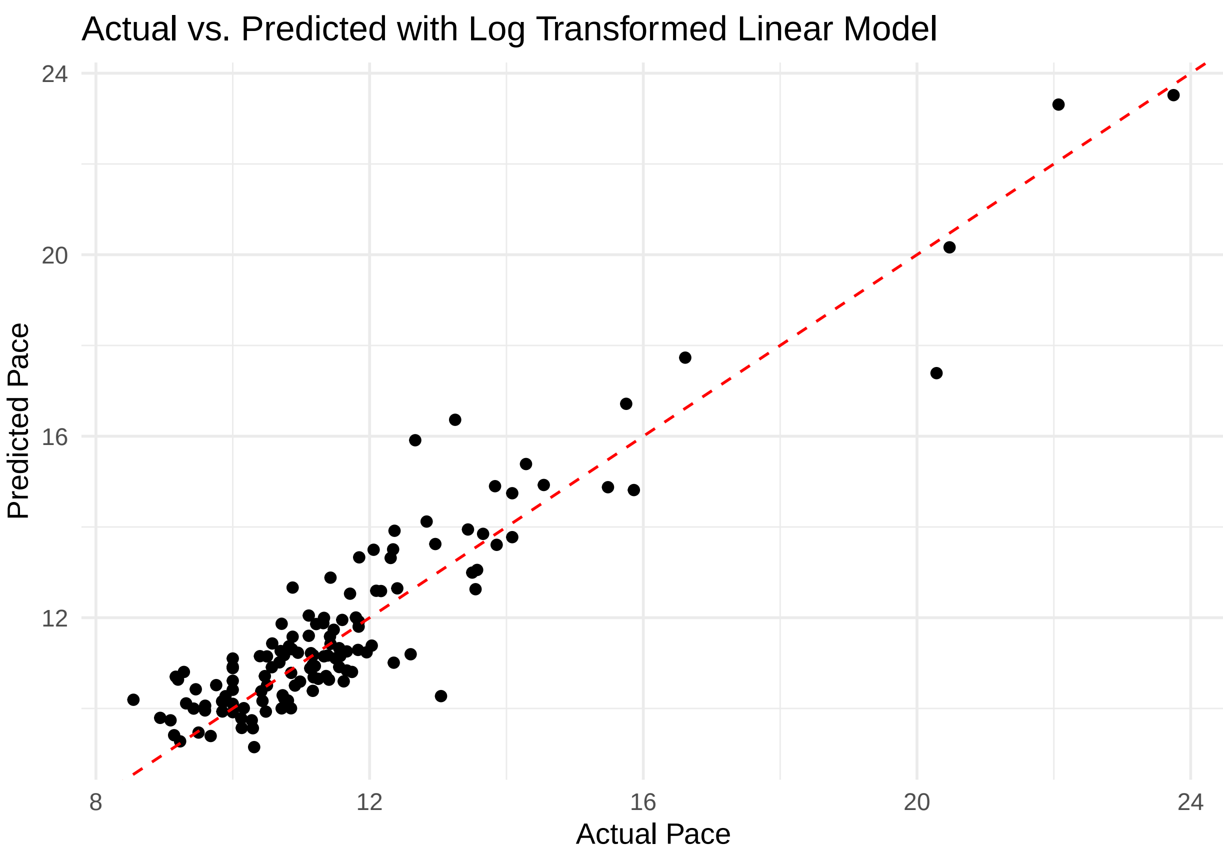
All predictors pass null hypothesis test.

```r
pred_log_race_lm = predict(log_race_lm, newdata = race_test)
mean((pred_log_race_lm - race_test$Pace)^2)
```

```
## [1] 0.8202037
```

Test error rate of 0.82.

```r
par(mfrow = c(1, 1))
plot_df = data.frame(Actual = race_test$Pace, Predicted = pred_log_race_lm)
ggplot(plot_df, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +  # Adds a line of perfect
  labs(x = "Actual Pace",
       y = "Predicted Pace",
       title = "Actual vs. Predicted with Log Transformed Linear Model") +
  theme_minimal()
```



This is the best model so far.

Try Ridge Regression Model:

```r
set.seed(1)
train_matrix = model.matrix(Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate +
                              Cadence, race_train)
test_matrix = model.matrix(Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate +
                             Cadence, race_test)
# Now we need to select lambda using cross-validation
cv_out = cv.glmnet(train_matrix, race_train$Pace, alpha = 0)
best_lam = cv_out$lambda.min
best_lam
```

```
## [1] 0.1552007
```

```r
# Lambda chosen by cross-validation is 0.15
```

Now we fit ridge regression model and make predictions.

```r
race_ridge = glmnet(train_matrix, race_train$Pace, alpha = 0)
pred_race_ridge = predict(race_ridge, s = best_lam, newx = test_matrix)
# Find test error
mean((pred_race_ridge - race_test$Pace)^2)
```

```
## [1] 0.9342368
```

This model produced a test error of 0.93.

Try Lasso Regression Model:

```r
set.seed(1)
train_matrix = model.matrix(Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate +
                              Cadence, race_train)
test_matrix = model.matrix(Pace ~ Distance + log(Elev_Per_Mile):Terrain + Heartrate +
                             Cadence, race_test)
# Now we need to select lambda using cross-validation
cv_out = cv.glmnet(train_matrix, race_train$Pace, alpha = 1)
best_lam = cv_out$lambda.min
best_lam
```

```
## [1] 0.001094911
```

```r
# Lambda chosen by cross-validation is 0.00109
```

Now we fit ridge regression model and make predictions.

```r
race_lasso = glmnet(train_matrix, race_train$Pace, alpha = 1)
pred_race_lasso = predict(race_lasso, s = best_lam, newx = test_matrix)
# Find test error
mean((pred_race_lasso - race_test$Pace)^2)
```

```
## [1] 0.853422
```

Test error rate is 0.85. Not better than log transformation.

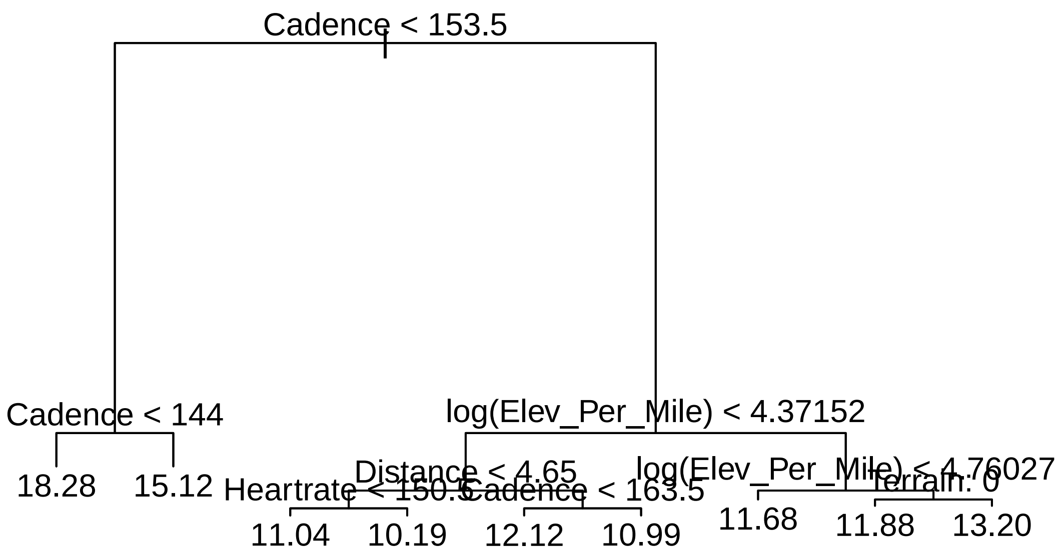
Fit a regression tree:

```r
race_tree = tree(Pace ~ Distance + log(Elev_Per_Mile) + Terrain + Heartrate +
                   Cadence, race_train)
summary(race_tree)
```

```
##
## Regression tree:
## tree(formula = Pace ~ Distance + log(Elev_Per_Mile) + Terrain +
##     Heartrate + Cadence, data = race_train)
## Number of terminal nodes:  9
```

```
## Residual mean deviance:  0.5388 = 65.73 / 122
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.20300 -0.37030  0.01938  0.00000  0.41380  2.08300
```

```r
plot(race_tree)
text(race_tree, pretty = 0)
```



Number of terminal nodes: 9.

```r
yhat = predict(race_tree, newdata = race_test)
mean((yhat - race_test$Pace)^2)
```

```
## [1] 1.421462
```

The test mean squared error rate is 1.42. This model performed poorly on data.

Try bagging model:

```r
set.seed(1)
race_bag = bagging(Pace ~ Distance + Elevation:Terrain + Heartrate +
                     Cadence, data = race_train)
race_bag
```

```
##
## Bagging regression trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = Pace ~ Distance + Elevation:Terrain +
##     Heartrate + Cadence, data = race_train)
```

```r
yhat_bag = predict(race_bag, newdata = race_test)
mean((yhat_bag - race_test$Pace)^2)
```

```
## [1] 1.623184
```

Test mean squared error of 1.62. This model also performed poorly on data.

Try random forest model with 100 trees:

```
set.seed(1)
race_rf = randomForest(Pace ~ Distance + Elevation:Terrain + Heartrate +
                         Cadence, data = race_train, ntree = 100,
                       mtry = 11, importance = TRUE)
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range
```

```
race_rf
```

```
##
## Call:
##  randomForest(formula = Pace ~ Distance + Elevation:Terrain +      Heartrate + Cadence, data = race_
##               Type of random forest: regression
##                     Number of trees: 100
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 0.6340539
##                     % Var explained: 82.01
```

```
yhat_rf = predict(race_rf, newdata = race_test)
mean((yhat_rf - race_test$Pace)^2)
```

```
## [1] 1.093195
```

Test mean squared error of 1.093. I believe that all of the models with test MSE's greater than 1, were overfitting to the training data.
Best model we were able to make was the linear regression model with log transformations. Test MSE of 82%, which is 0.3% less than liner regression model fit in the notes from class.

```
log_race_lm = lm(Pace ~ log(Distance) + log(Elev_Per_Mile):Terrain +
                   log(Heartrate) + log(Cadence), race_train)
summary(log_race_lm)
```

```
##
## Call:
## lm(formula = Pace ~ log(Distance) + log(Elev_Per_Mile):Terrain +
##     log(Heartrate) + log(Cadence), data = race_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.94820 -0.43739 -0.02266  0.45490  2.54634
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   137.3527    12.6765  10.835  < 2e-16 ***
## log(Distance)                   0.7137     0.2029   3.518 0.000608 ***
```

```
## log(Heartrate)               -5.6295     1.4685  -3.833 0.000199 ***
## log(Cadence)                -19.8686     1.7402 -11.417  < 2e-16 ***
## log(Elev_Per_Mile):Terrain0   0.6933     0.2006   3.457 0.000748 ***
## log(Elev_Per_Mile):Terrain1   0.8953     0.1829   4.895 2.97e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.806 on 125 degrees of freedom
## Multiple R-squared:  0.8241, Adjusted R-squared:  0.8171
## F-statistic: 117.2 on 5 and 125 DF,  p-value: < 2.2e-16
```

Question 3:

Make a race prediction of minutes for a 5k race held on November 23, 2023. Assume a distance of 3.1 miles, 95 feet of elevation and a PE (perceived exertion) of 5, temperature of 47 degrees and terrain = 0. How about a 10k race (6.2 miles), 200 feet of elevation with same temp and terrain? Be sure to account for expected heart rate and cadence when solving this problem!
We can make prediction using log transformation model from question 2.
Let's create new data frame with information from question. We need to use expected heart rate and cadence, which will be the averages of each predictor column.

```
expected_hr = mean(race_df$Heartrate)
expected_cadence = mean(race_df$Cadence)
race_projection_data = data.frame(
  Distance = 3.1,
  Elev_Per_Mile = 95/3.1,
  Terrain = 0,
  Heartrate = expected_hr,
  Cadence = expected_cadence
)
```

Now, lets make our prediction of minutes for the 5k race.

```
race_projection_data$Terrain = as.factor(race_projection_data$Terrain)
predictions = predict(log_race_lm, newdata = race_projection_data)
predictions
```

```
##        1
## 10.55482
```

Predicted pace = 10.55 minutes per mile.
To get total minutes for the race, multiply prediction pace by 3.1.

```
total_minutes = 10.55 * 3.1
total_minutes
```

```
## [1] 32.705
```

32.71 total minutes for the 5k race.

Let's do the same thing for the 10k race.

```
race_projection_data = data.frame(
  Distance = 6.1,
  Elev_Per_Mile = 200/6.2,
  Terrain = 0,
  Heartrate = expected_hr,
  Cadence = expected_cadence
)
race_projection_data$Terrain = as.factor(race_projection_data$Terrain)
predictions = predict(log_race_lm, newdata = race_projection_data)
predictions
```

```
##        1
## 11.07344
```

Predicted pace = 11.07 minutes per mile.
To get total minutes for the race, multiply prediction pace by 6.2.

```
total_minutes = 11.07 * 6.2
total_minutes
```

```
## [1] 68.634
```

68.63 total minutes for the 10k race.

Question 4

Write a brief summary of your findings and what additional data you would want to more accurately make a prediction.

I found that some of the non-linear models that I fitted, like the regression tree model, the random forest model and the bagging model, were all over fitting to the training data, as I was getting mean squared error values of over 1 when using the training models to predict the test data. On the other models that I fitted using the training data, I felt like I was not able to move the needle on the test error rates. If anything, most of my models had worse test rate than the original linear model that was fit in the class notes. I was only able to produce a smaller mean squared error value in the linear model where I log transformed Distance, Heartrate, and Cadence along with the interaction term of log(Elev_Per_Mile):Terrain.

I think the only thing I could ask for at this point is more data. The more data would probably have some influence on the models that I fit. If I could have a bigger overall data set, I could then have more data to train the models, which would hopefully minimize any over fitting occurring, which could then produce better prediction rates.

Question 5

With unfiltered data (i.e., don't remove treadmill data), create a clustering model to backfill perceived exertion. Write a short summary explaining how and why it works to a non-technical audience. Be sure to test the performance of your model if you can.

The clustering method is used to impute missing values.
Let's reset our data for:

```r
setwd("/Users/doojerthekid/Documents/Merrimack Grad School Documents/DSE6620")
run_df = read_excel("Week_2/Run.xlsx")
# Subset data frame
run_df = run_df[1:361,]

# Change date column to date structure
str(run_df$Date)
```

```
##  POSIXct[1:361], format: "2023-11-20" "2023-11-19" "2023-11-17" "2023-11-15" "2023-11-13" ...
```

```r
run_df$Date = as.Date(run_df$Date,format = '%Y-%m-%d')

# Create a year column
run_df = run_df %>%
  mutate(Year = year(Date))

# Convert Terrain column to a factor variable
run_df$Terrain = as.factor(run_df$Terrain)
head(run_df)
```

```
## # A tibble: 6 x 16
##   Date       Distance Elevation Minutes Heartrate Cadence  Temp Terrain    PE
##   <date>        <dbl>     <dbl>   <dbl>     <dbl>   <dbl> <dbl> <fct>   <dbl>
## 1 2023-11-20      2         141    23         148     158    68 0           2
## 2 2023-11-19      2.8      1824   108         136      NA    51 1           4
## 3 2023-11-17      6         856    79.5       151     142    63 1           3
## 4 2023-11-15      6.2       105    59.5       176     165    35 0           4
## 5 2023-11-13      3.7       118    35.5       174     167    33 0           3
## 6 2023-11-11      4          82    38         174     169    45 0           4
## # i 7 more variables: Sneaker <chr>, Area <chr>, Pace <dbl>, Speed <dbl>,
## #   Elev_Per_Mile <dbl>, Notes <chr>, Year <dbl>
```

```r
# Keep treadmill data
run_df = run_df %>%
  select(Date:Terrain, Pace:Elev_Per_Mile, PE, Year)
run_df
```

```
## # A tibble: 361 x 13
##    Date       Distance Elevation Minutes Heartrate Cadence  Temp Terrain  Pace
##    <date>        <dbl>     <dbl>   <dbl>     <dbl>   <dbl> <dbl> <fct>   <dbl>
##  1 2023-11-20      2         141    23         148     158    68 0       11.5
##  2 2023-11-19      2.8      1824   108         136      NA    51 1       38.6
##  3 2023-11-17      6         856    79.5       151     142    63 1       13.2
##  4 2023-11-15      6.2       105    59.5       176     165    35 0        9.60
##  5 2023-11-13      3.7       118    35.5       174     167    33 0        9.59
##  6 2023-11-11      4          82    38         174     169    45 0        9.5
##  7 2023-11-09      6.2       371    60.5       178     168    54 0        9.76
##  8 2023-11-07      3.7       100    39.5       145     163    NA <NA>    10.7
##  9 2023-11-06      4         128    40         177     167    42 0       10
## 10 2023-11-05      4.7       236    46.5       172     169    48 0        9.89
## # i 351 more rows
## # i 4 more variables: Speed <dbl>, Elev_Per_Mile <dbl>, PE <dbl>, Year <dbl>
```

```r
# Flip terrain back into numeric column
run_df$Terrain = as.numeric(run_df$Terrain)
```

Before using cluster model to backfill missing PE values, we need to handle NA values in other columns. Let's remove all rows in data frame that have NA besides rows that have NA in PE values, since that is what we are looking to backfill.

```r
colnames(run_df)
```

```
##  [1] "Date"          "Distance"      "Elevation"     "Minutes"
##  [5] "Heartrate"     "Cadence"       "Temp"          "Terrain"
##  [9] "Pace"          "Speed"         "Elev_Per_Mile" "PE"
## [13] "Year"
```

```r
columns_to_check = c("Date", "Distance", "Elevation", "Minutes", "Heartrate", "Cadence",
                     "Temp", "Terrain", "Pace", "Speed", "Elev_Per_Mile", "Year")
run_df = run_df[complete.cases(run_df[, columns_to_check]), ]
```

Now, let us use clustering to backfill PE missing values.

```r
columns = run_df[, c("Distance", "Elevation", "Minutes", "Heartrate",
                      "Cadence", "Temp", "Terrain", "Pace", "Speed", "Elev_Per_Mile")]
# Handle missing values in columns
columns[is.na(columns)] = 0
# Replace NA with 0

# Normalize features
normalized_columns = scale(columns)

# Perform hierarchical clustering:
hclust = hclust(dist(normalized_columns), method = "complete")

# Perform k-means clustering
k = 3  # Set the number of clusters
clusters = cutree(hclust, k)
run_df$cluster = clusters

# Backfill PE column based on cluster means
run_df = run_df %>%
  group_by(cluster) %>%
  mutate(PE = ifelse(is.na(PE), mean(PE, na.rm = TRUE), PE))

# Remove the 'cluster' column if you don't need it anymore
run_df$cluster = NULL

# Since PE is a whole number, let's adjust the new data to be rounded to nearest
# whole number:
run_df$PE = round(run_df$PE)
```

Clustering is a way to group certain events, in this case, similar runs. What we did is group similar runs together based on distance, elevation, minutes,heart rate, cadence, temp, terrain, pace, speed and elevation per mile. The clustering algorithm automatically does this sorting and takes the average of those similar runs to help replace the NA values in our data set. When speaking to a non-technical audience, it is easy to convey how you compare similar observations in the data set to help infer what the missing values could be.