

CMDA-4654

Project 1

Spencer Paragas and Christopher Mascis

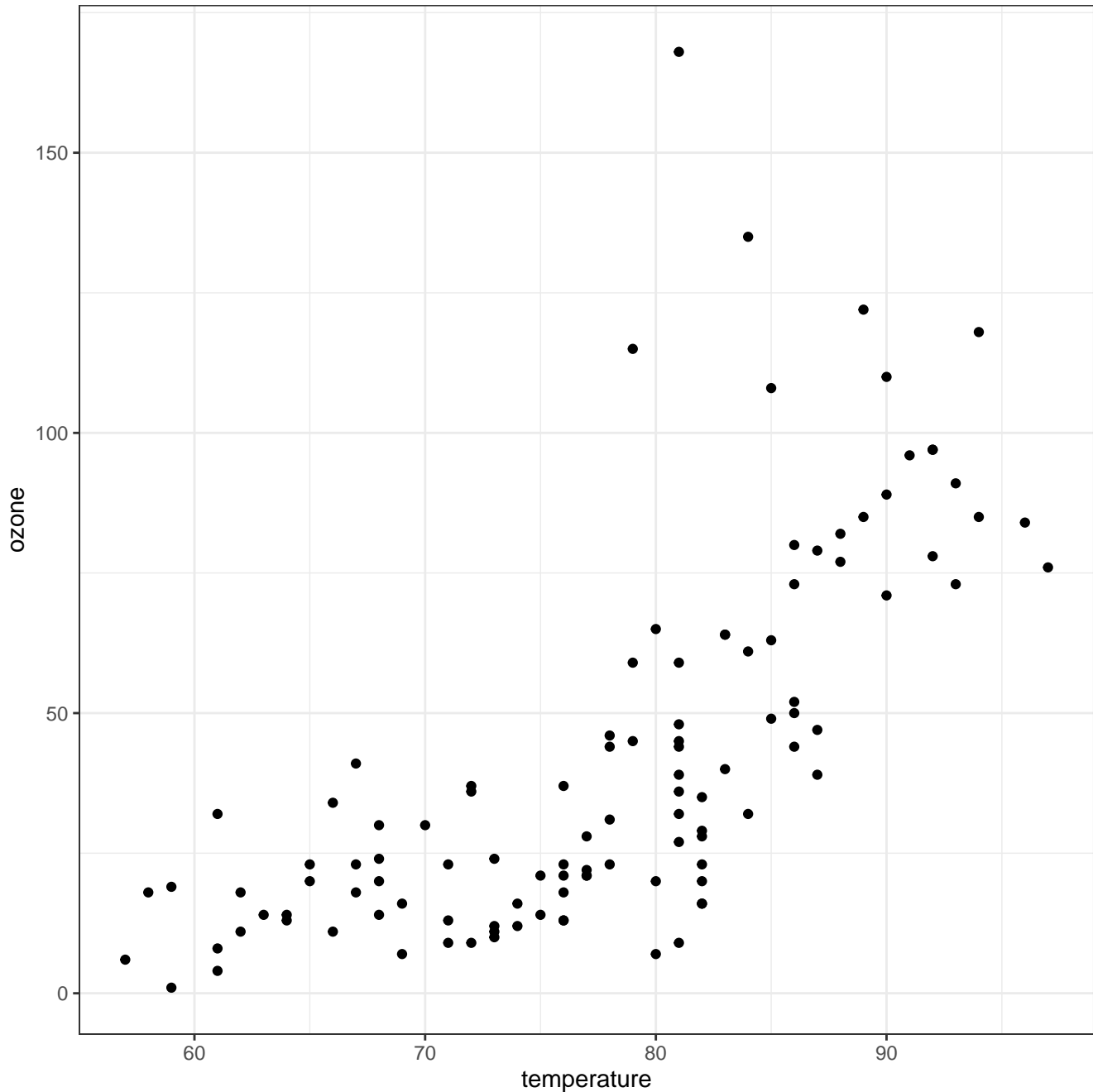
10/31/2021

Problem 1

```
library(dplyr)
library(ggplot2)

#setwd("C:/Users/Spencer/Downloads")
load("ozone.RData")

ggplot(ozone, aes(x = temperature, y = ozone)) + theme_bw() + geom_point()
```



Part a

Fitting polynomials of different degrees

```
model1 <- lm(ozone ~ poly(radiation,1), data = ozone)
model2 <- lm(ozone ~ poly(radiation,2), data = ozone)
model3 <- lm(ozone ~ poly(radiation,3), data = ozone)
model4 <- lm(ozone ~ poly(radiation,4), data = ozone)
```

```

model5 <- lm(ozone ~ poly(radiation,5), data = ozone)
model6 <- lm(ozone ~ poly(radiation,6), data = ozone)

summary(model1)

Call:
lm(formula = ozone ~ poly(radiation, 1), data = ozone)

Residuals:
    Min       1Q   Median       3Q      Max
-48.292 -21.361  -8.864   16.373  119.136

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      42.099      2.974   14.15 < 2e-16 ***
poly(radiation, 1) 121.572     31.335    3.88 0.000179 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 31.33 on 109 degrees of freedom
Multiple R-squared:  0.1213,    Adjusted R-squared:  0.1133
F-statistic: 15.05 on 1 and 109 DF,  p-value: 0.0001793

summary(model2)

Call:
lm(formula = ozone ~ poly(radiation, 2), data = ozone)

Residuals:
    Min       1Q   Median       3Q      Max
-40.155 -22.793  -6.438   18.061  115.117

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      42.099      2.832   14.864 < 2e-16 ***
poly(radiation, 2)1 121.572     29.840    4.074 8.84e-05 ***
poly(radiation, 2)2 -104.178     29.840   -3.491 0.000698 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 29.84 on 108 degrees of freedom
Multiple R-squared:  0.2104,    Adjusted R-squared:  0.1958
F-statistic: 14.39 on 2 and 108 DF,  p-value: 2.875e-06

summary(model3)

Call:
lm(formula = ozone ~ poly(radiation, 3), data = ozone)

Residuals:
    Min       1Q   Median       3Q      Max
-45.05 -19.44  -4.52   15.61  109.95

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      42.10      2.78   15.141 < 2e-16 ***
poly(radiation, 3)1 121.57     29.29    4.150 6.69e-05 ***
poly(radiation, 3)2 -104.18     29.29   -3.556 0.000562 ***
poly(radiation, 3)3  -65.93     29.29   -2.251 0.026450 *

```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 29.29 on 107 degrees of freedom
Multiple R-squared:  0.2461,    Adjusted R-squared:  0.225
F-statistic: 11.65 on 3 and 107 DF,  p-value: 1.154e-06
```

```
summary(model4)
```

```
Call:
lm(formula = ozone ~ poly(radiation, 4), data = ozone)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-44.929 -19.519  -4.647  15.700 110.071
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      42.099      2.793   15.072 < 2e-16 ***
poly(radiation, 4)1 121.572     29.428    4.131 7.23e-05 ***
poly(radiation, 4)2 -104.178     29.428   -3.540 0.000596 ***
poly(radiation, 4)3  -65.932     29.428   -2.240 0.027149 *
poly(radiation, 4)4    4.777     29.428    0.162 0.871356
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 29.43 on 106 degrees of freedom
Multiple R-squared:  0.2463,    Adjusted R-squared:  0.2179
F-statistic: 8.661 on 4 and 106 DF,  p-value: 4.352e-06
```

```
summary(model5)
```

```
Call:
lm(formula = ozone ~ poly(radiation, 5), data = ozone)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-46.076 -16.978  -5.621  16.472 108.924
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      42.099      2.801   15.029 < 2e-16 ***
poly(radiation, 5)1 121.572     29.511    4.119 7.59e-05 ***
poly(radiation, 5)2 -104.178     29.511   -3.530 0.000618 ***
poly(radiation, 5)3  -65.932     29.511   -2.234 0.027595 *
poly(radiation, 5)4    4.777     29.511    0.162 0.871718
poly(radiation, 5)5   18.759     29.511    0.636 0.526389
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 29.51 on 105 degrees of freedom
Multiple R-squared:  0.2492,    Adjusted R-squared:  0.2135
F-statistic: 6.971 on 5 and 105 DF,  p-value: 1.168e-05
```

```
summary(model6)
```

```
Call:
lm(formula = ozone ~ poly(radiation, 6), data = ozone)
```

```
Residuals:
```

| Min | 1Q | Median | 3Q | Max |
|---------|---------|--------|--------|---------|
| -45.079 | -17.204 | -5.404 | 16.483 | 109.921 |

Coefficients:

| | Estimate | Std. Error | t value | Pr(> t) |
|---------------------|----------|------------|---------|--------------|
| (Intercept) | 42.099 | 2.812 | 14.971 | < 2e-16 *** |
| poly(radiation, 6)1 | 121.572 | 29.626 | 4.104 | 8.11e-05 *** |
| poly(radiation, 6)2 | -104.178 | 29.626 | -3.516 | 0.00065 *** |
| poly(radiation, 6)3 | -65.932 | 29.626 | -2.225 | 0.02821 * |
| poly(radiation, 6)4 | 4.777 | 29.626 | 0.161 | 0.87221 |
| poly(radiation, 6)5 | 18.759 | 29.626 | 0.633 | 0.52800 |
| poly(radiation, 6)6 | -12.854 | 29.626 | -0.434 | 0.66527 |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 29.63 on 104 degrees of freedom

Multiple R-squared: 0.2506, Adjusted R-squared: 0.2073

F-statistic: 5.795 on 6 and 104 DF, p-value: 3.079e-05

The polynomial fit that appears to work the best is the one with degree 3.

Part b

Writing the function that carries out LOESS regression

```
# This function has the following inputs.
#
# * x - a numeric input vector
# * y - a numeric response
#
# Note span and degree are shown with their default values.
# * degree should be 1 or 2 only
# * span can be any value in interval (0, 1) non-inclusive.
#
# If show.plot = TRUE then a plot of the final fit is shown

myloess <- function(x, y, span = 0.5, degree = 1, show.plot = TRUE) {

  # Getting range of values
  xrange <- diff(range(x))

  # Checking span meets requirements and setting width
  if (between(span, 0, 1)) {
    width <- span*xrange
  }
  else {
    stop("Span must be between 0 and 1 non-inclusive")
  }

  # Getting total number of points and windows (they'll be the same)
  N_total <- length(x)
  Win_total <- length(x)

  # Allocating space for vector of each window's population
  n_points <- vector(mode = "integer", length = length(x))

  # Allocating space for vector of fitted values
  yhat <- vector(mode = "numeric", length = length(x))

  # Combining our variables in data frame
  mydata <- cbind.data.frame(x, y)
```

```

# Fitting each point
for(x0 in x) {

  # Setting population of window
  sample <- subset(mydata, between(x, x0-width/2, x0+width/2))
  n <- length(sample[,1])

  # Getting weights into diagonal matrix
  weights <- (1-(abs(sample[,1] - x0)/width*2)^3)^3
  W_mat <- diag(weights, n, n)

  # Checking degree and completing our regression accordingly
  if (degree == 1) {

    X_mat <- cbind(rep(1, n), sample[,1])
    betahat <- solve( t(X_mat)%*%W_mat%*%X_mat ) %*% t(X_mat) %*% W_mat %*% sample[,2]

    # Getting fitted value
    yhat[which(x0 == x)] <- betahat[1] + betahat[2]*x0
  }
  else if (degree == 2) {

    X_mat <- cbind(rep(1, n), sample[,1], sample[,1]^2)
    betahat <- solve( t(X_mat)%*%W_mat%*%X_mat ) %*% t(X_mat) %*% W_mat %*% sample[,2]

    # Getting fitted value
    yhat[which(x0 == x)] <- betahat[1] + betahat[2]*x0 + betahat[3]*x0^2
  }
  else {
    stop("Degree must be set to either 1 or 2")
  }

  # Getting population of window
  n_points[which(x0 == x)] <- n
}

# Calculating SSE, MSE, and residual standard error
SSE <- sum((y-yhat)^2)
MSE <- SSE/(N_total-2)
RSE <- sqrt(MSE)

# Creating plot of final fit
loessplot <- ggplot(mydata, aes(x, y)) +
  geom_point(size = 3, alpha = 0.5, color = "grey") +
  geom_line(aes(x, yhat), color = "red", lty = 1) +
  xlab(deparse(substitute(x))) + ylab(deparse(substitute(y))) +
  ggtitle(paste("LOESS with degree =", degree,"and span =", span, sep = " "))

# Checking whether to show plot or not
if (show.plot == T) {
  print(loessplot)
}

# Returning named list
return(invisible(list(span = span, degree = degree, N_total = N_total, Win_total = Win_total,
  n_points = n_points, SSE = SSE, RSE = RSE, loessplot = loessplot)))
}

```

Determining LOESS regression fits on the data

```

# Creating an empty data frame
fit_table <- data.frame()

# Determining fits and putting info into data frame
for(j in 1:2) {
  for (i in seq(0.25, 0.75, by = 0.05)) {
    fit_table <- rbind(fit_table, c(i, j, myloess(ozone$temperature, ozone$ozone,
                                              span = i, degree = j, show.plot = F)$RSE))
  }
}

```

```

# Changing column names
colnames(fit_table) <- c("Span", "Degree", "RSE")

```

```

# Displaying data frame
fit_table

```

| | Span | Degree | RSE |
|----|------|--------|----------|
| 1 | 0.25 | 1 | 21.69723 |
| 2 | 0.30 | 1 | 21.84914 |
| 3 | 0.35 | 1 | 21.87405 |
| 4 | 0.40 | 1 | 21.87005 |
| 5 | 0.45 | 1 | 21.88484 |
| 6 | 0.50 | 1 | 21.92533 |
| 7 | 0.55 | 1 | 21.98316 |
| 8 | 0.60 | 1 | 22.05339 |
| 9 | 0.65 | 1 | 22.12531 |
| 10 | 0.70 | 1 | 22.18786 |
| 11 | 0.75 | 1 | 22.24218 |
| 12 | 0.25 | 2 | 20.70123 |
| 13 | 0.30 | 2 | 21.21788 |
| 14 | 0.35 | 2 | 21.65352 |
| 15 | 0.40 | 2 | 21.83486 |
| 16 | 0.45 | 2 | 21.82088 |
| 17 | 0.50 | 2 | 21.75230 |
| 18 | 0.55 | 2 | 21.71153 |
| 19 | 0.60 | 2 | 21.71786 |
| 20 | 0.65 | 2 | 21.76031 |
| 21 | 0.70 | 2 | 21.81288 |
| 22 | 0.75 | 2 | 21.86516 |

The three “best” fits with degree 1 appear to be span 0.25, 0.30, and 0.40. The three “best” fits with degree 2 appear to be span 0.25, 0.30, and 0.35.

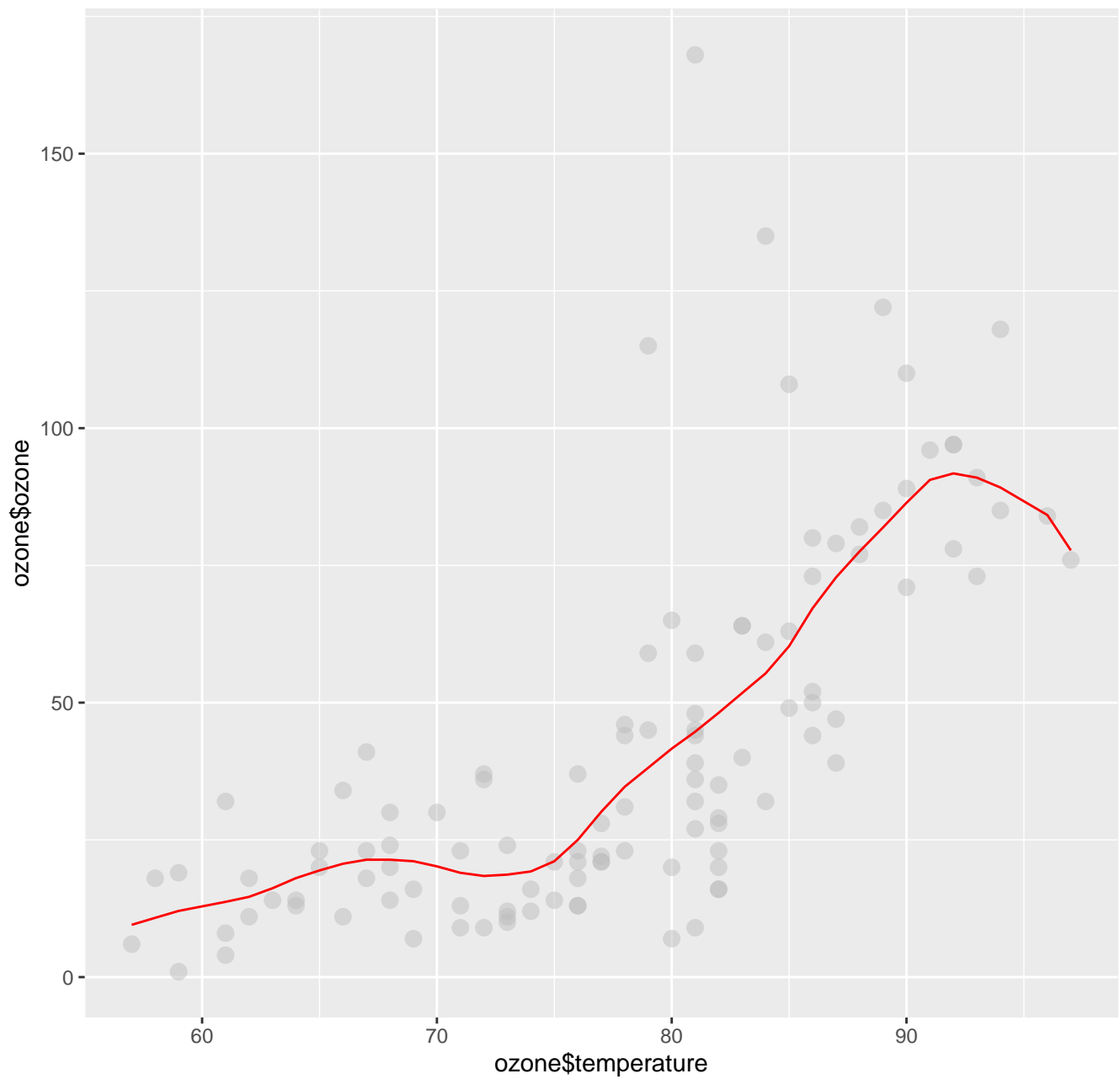
Plotting the best fits found above

```

myloess(ozone$temperature, ozone$ozone,
                                              span = 0.25, degree = 1, show.plot = F)$loessplot

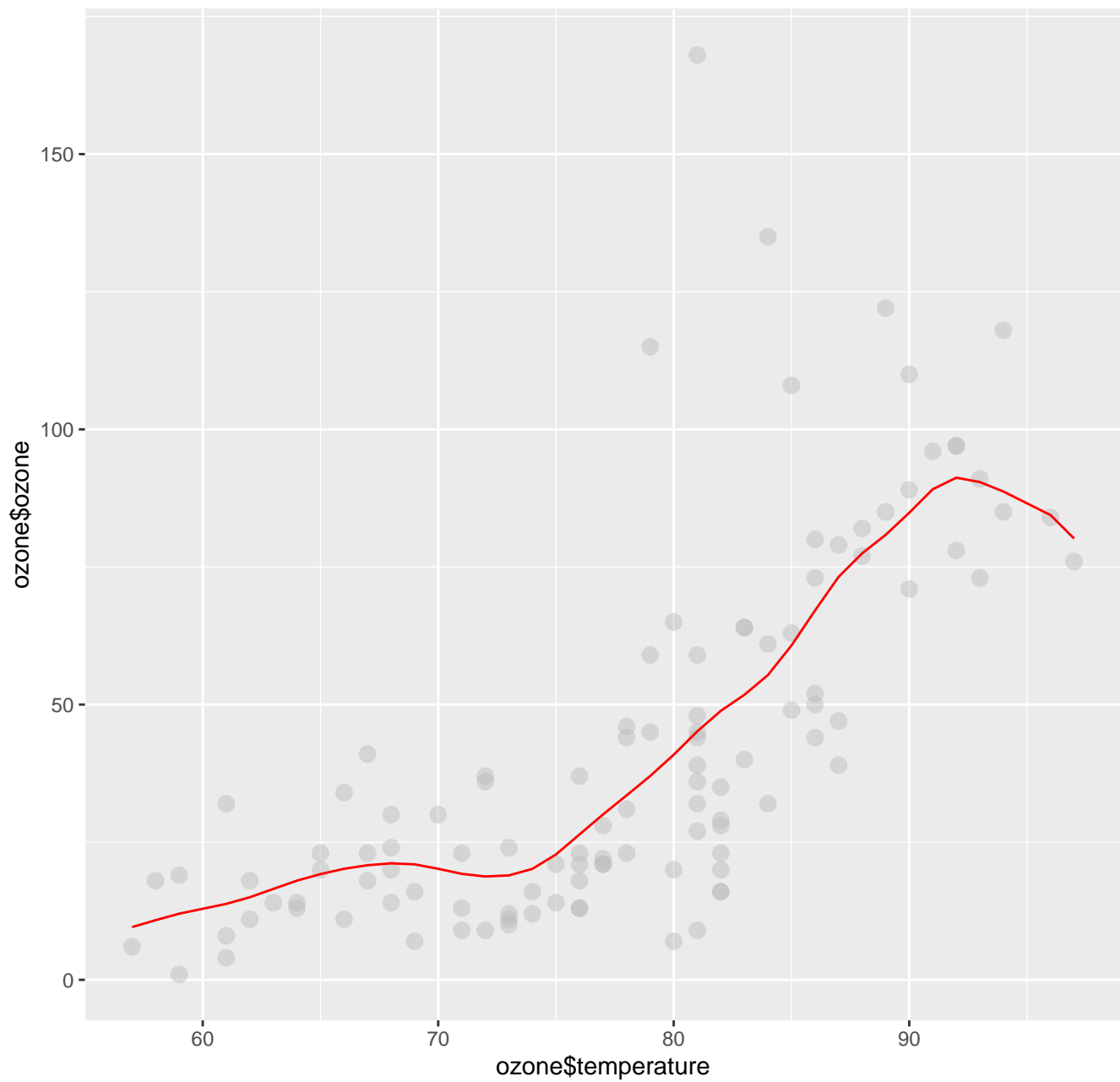
```

LOESS with degree = 1 and span = 0.25



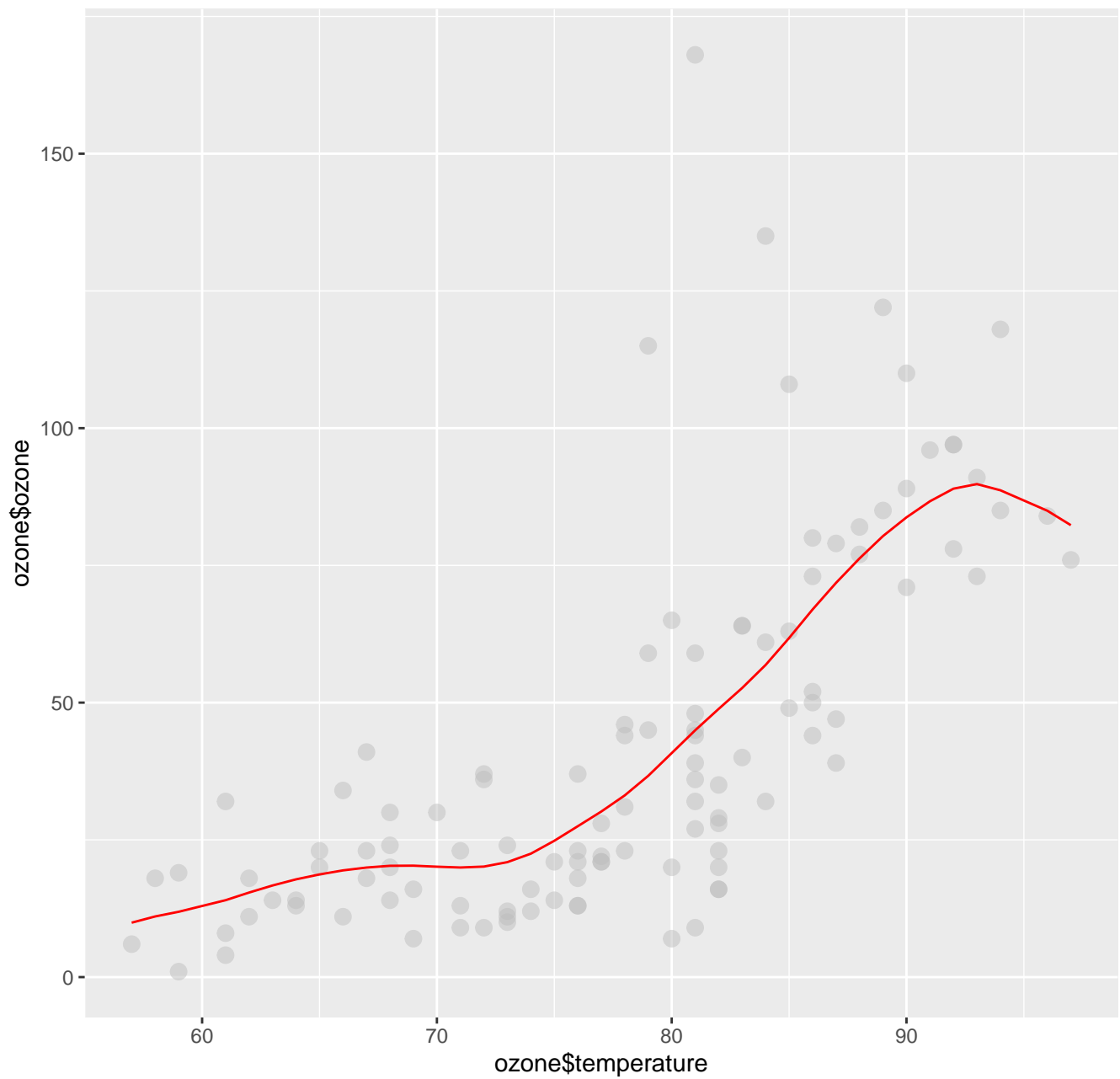
```
myloess(ozone$temperature, ozone$ozone,  
span = 0.30, degree = 1, show.plot = F)$loessplot
```


LOESS with degree = 1 and span = 0.3



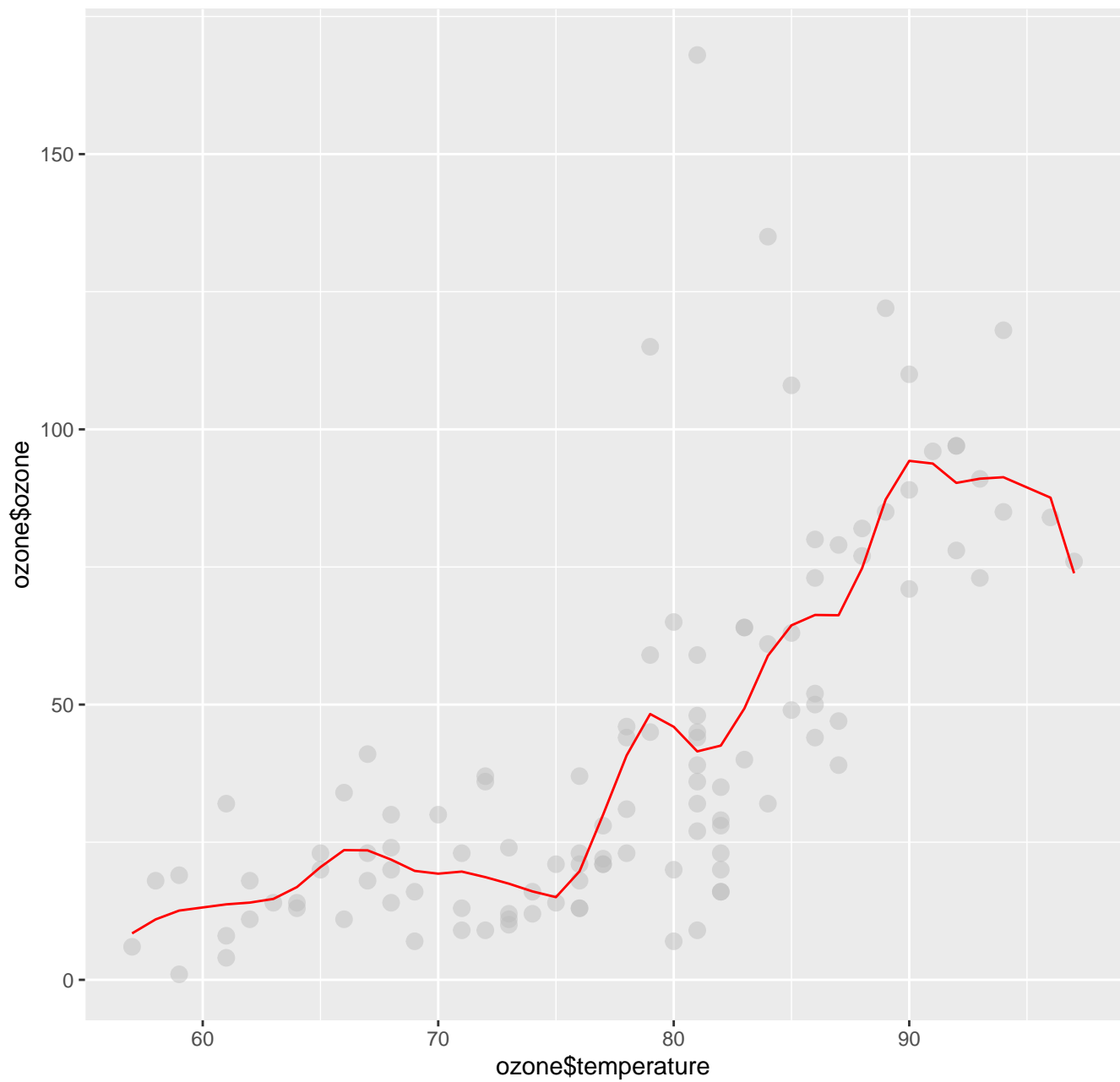
```
myloess(ozone$temperature, ozone$ozone,  
span = 0.40, degree = 1, show.plot = F)$loessplot
```

LOESS with degree = 1 and span = 0.4



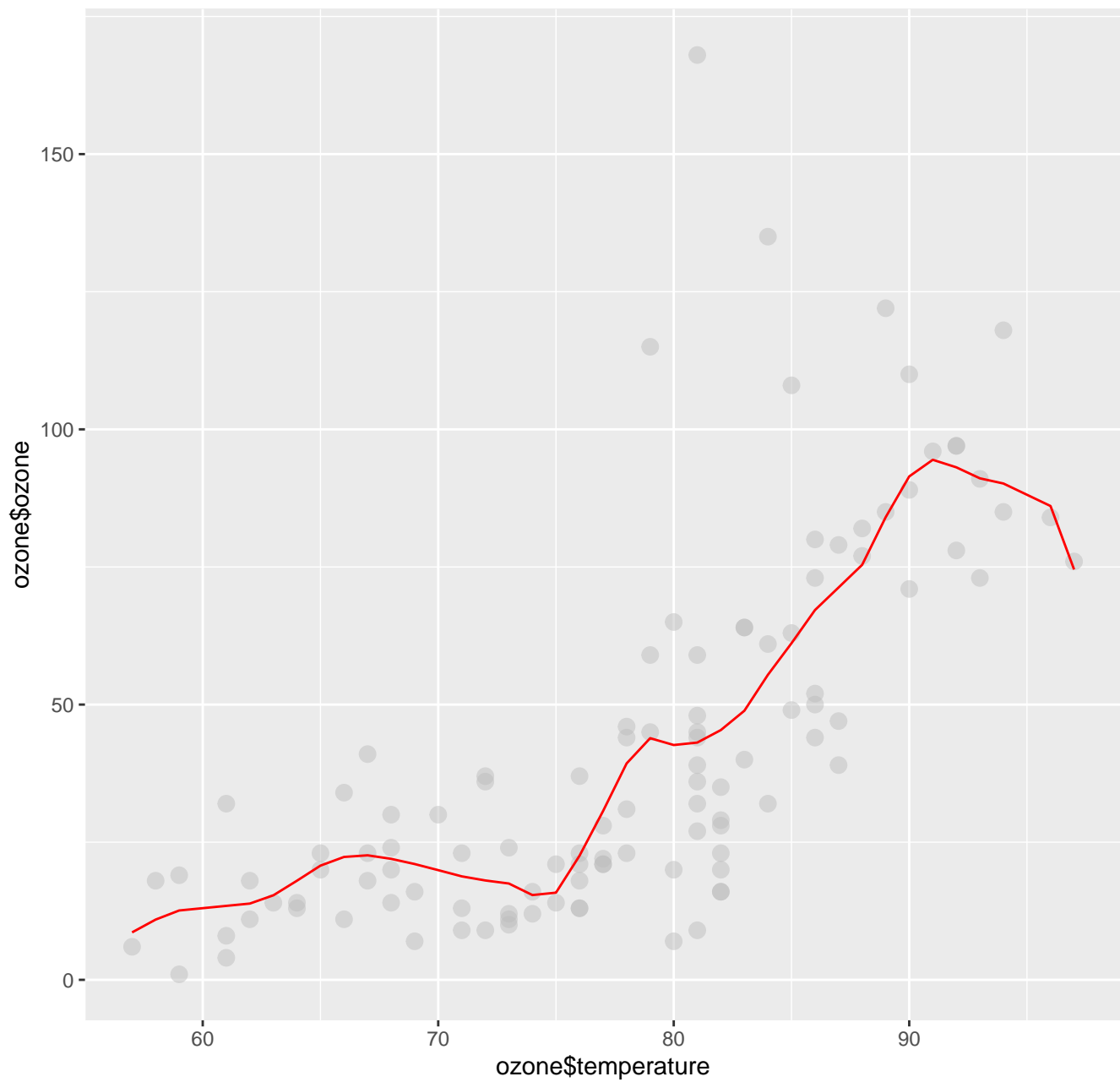
```
myloess(ozone$temperature, ozone$ozone,  
span = 0.25, degree = 2, show.plot = F)$loessplot
```

LOESS with degree = 2 and span = 0.25



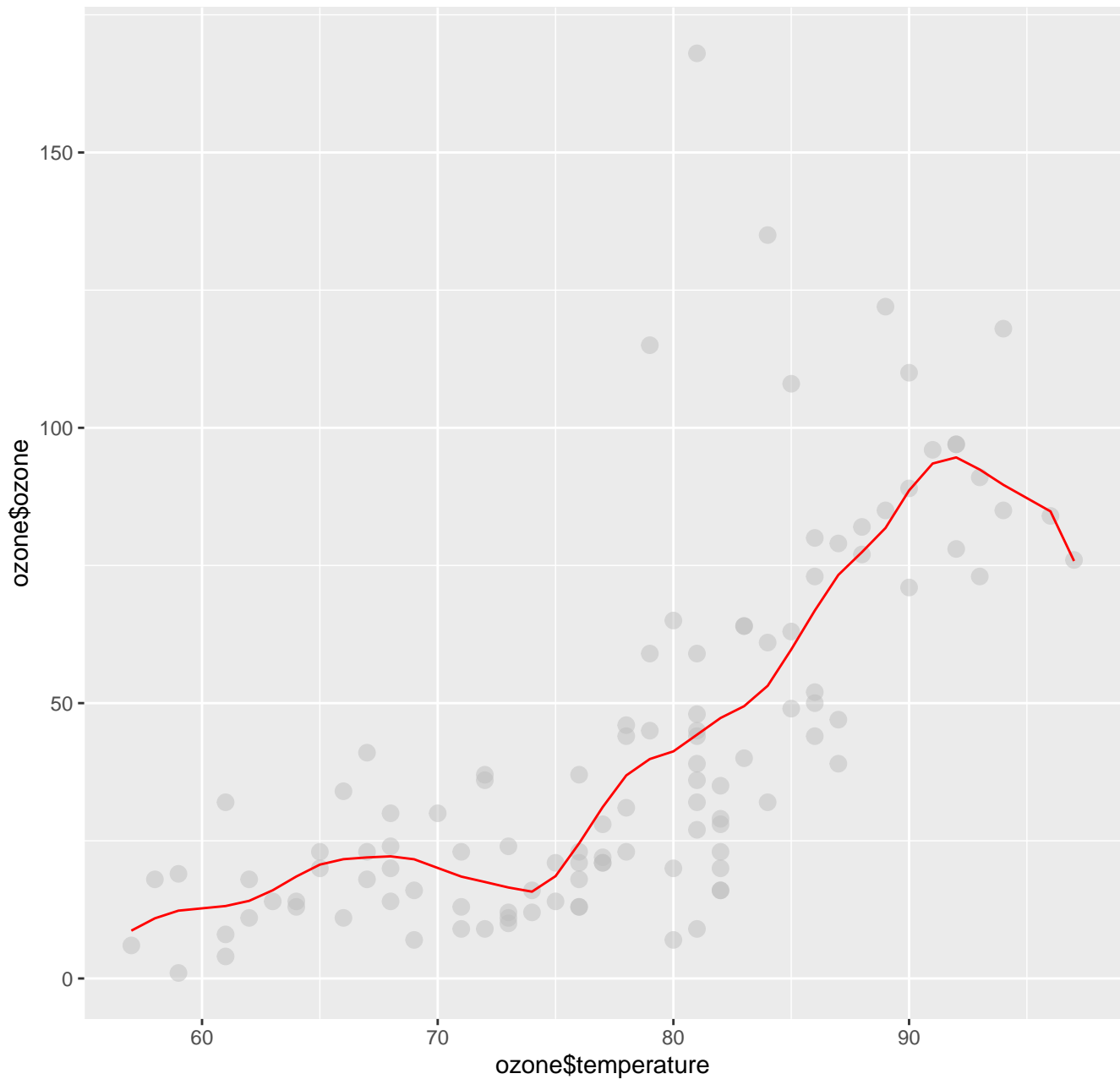
```
myloess(ozone$temperature, ozone$ozone,  
span = 0.30, degree = 2, show.plot = F)$loessplot
```

LOESS with degree = 2 and span = 0.3



```
myloess(ozone$temperature, ozone$ozone,  
span = 0.35, degree = 2, show.plot = F)$loessplot
```

LOESS with degree = 2 and span = 0.35

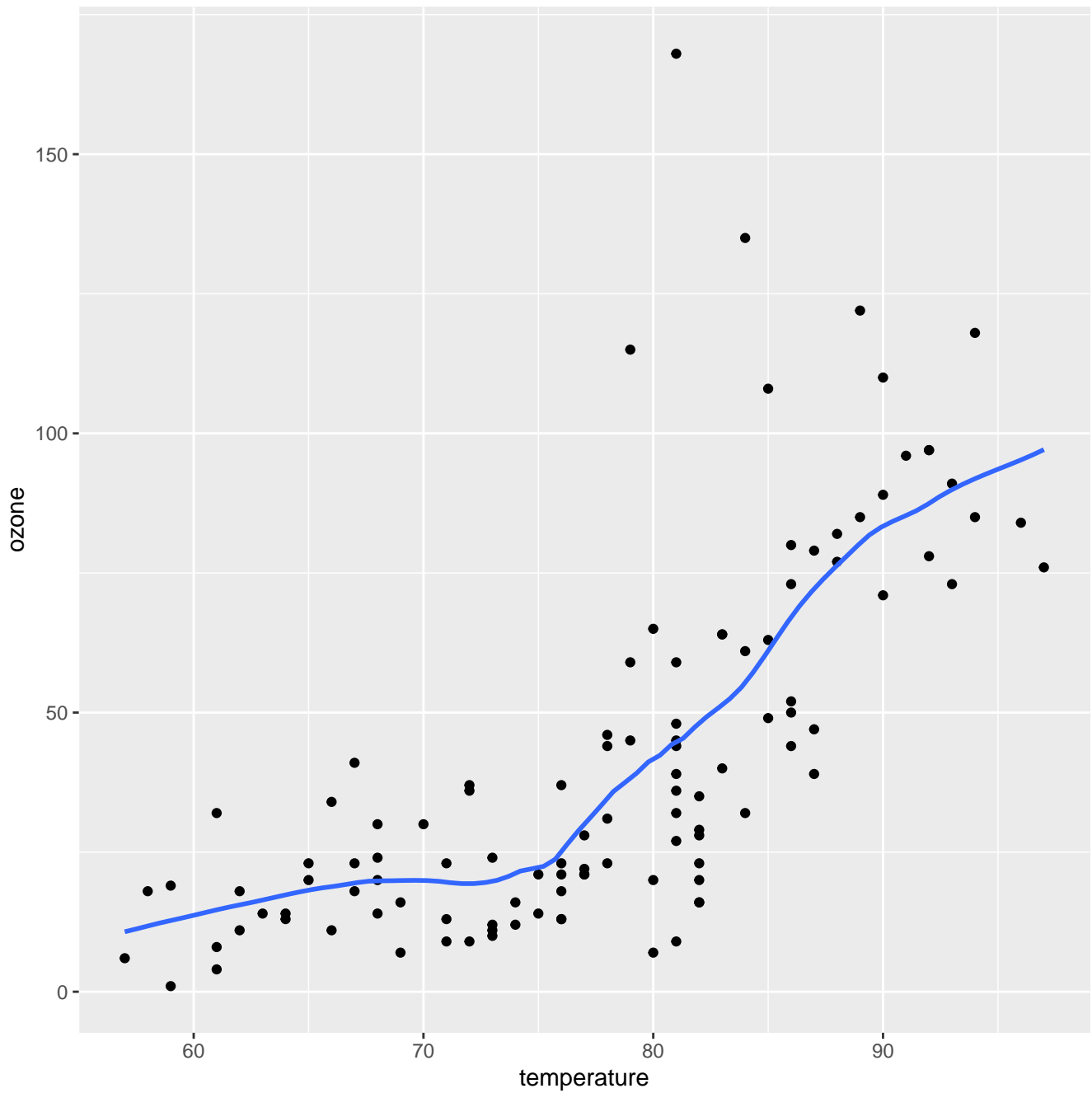


Visually inspecting the best fits compared to the 2nd and 3rd best fits, we do feel we may have over-fit the data, especially for the fits with degree 2. The model with degree 1 and span 0.4 appears to be the “best” fit.

Part c

Comparing results with built-in LOESS function

```
ozone %>% ggplot(aes(temperature, ozone)) +  
  geom_point() +  
  geom_smooth(method = "loess", degree = 1, span = 0.40, se = F, method.args = list(degree=1))
```

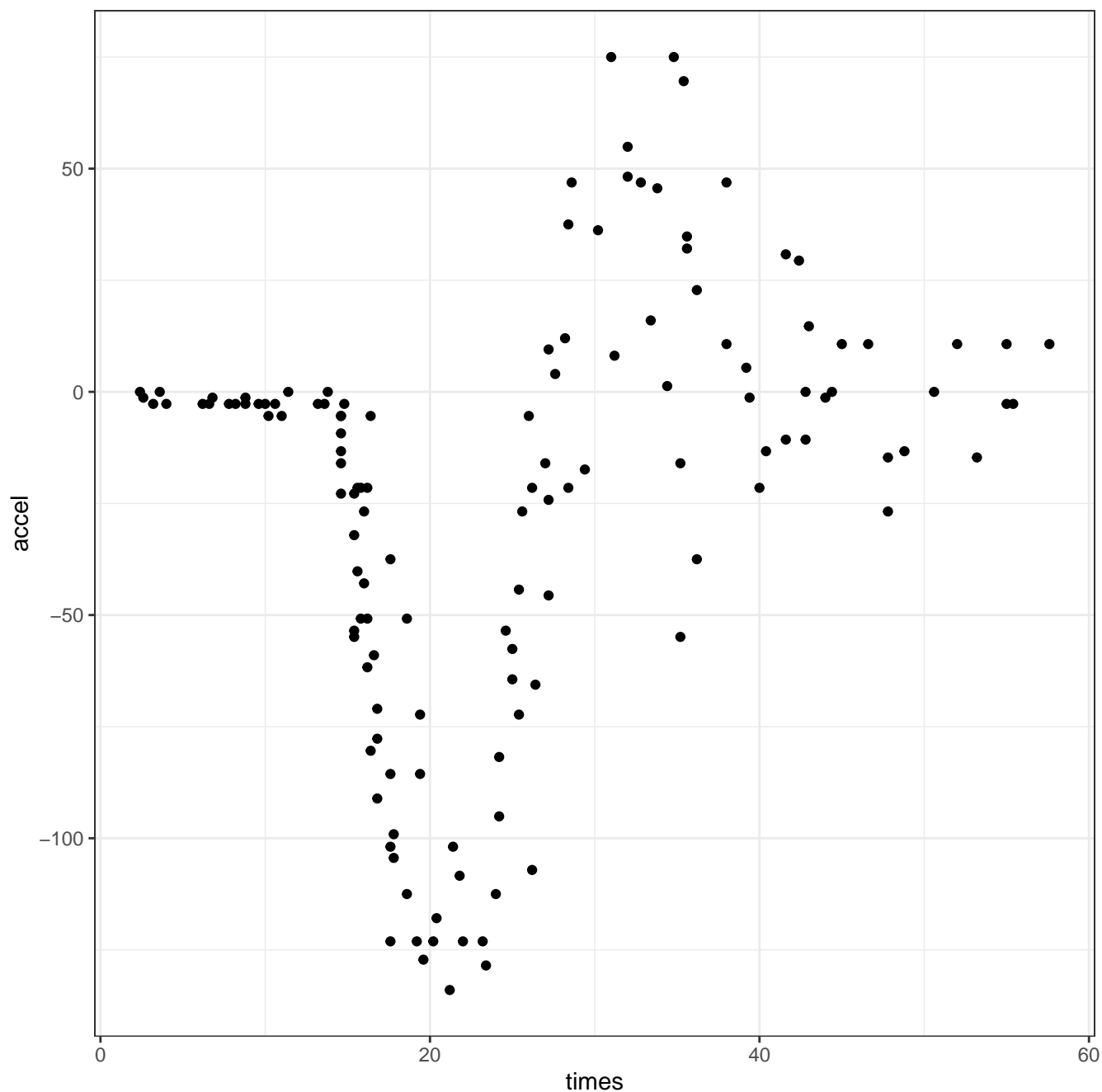


The built-in LOESS function results in a fit that does not over-fit the data, but we feel our model provides the better fit.

Problem 2

```
library(MASS)
data("mcycle")

ggplot(mcycle, aes(x = times, y = accel)) + theme_bw() + geom_point()
```



Part a

Determining LOESS regression fits on the data

```
# Creating an empty data frame
fit_table2 <- data.frame()

# Determining fits and putting info into data frame
for(j in 1:2) {
  for (i in seq(0.25, 0.75, by = 0.05)) {
    fit_table2 <- rbind(fit_table2, c(i, j, myloess(mcycle$times, mcycle$accel,
                                                    span = i, degree = j, show.plot = F)$RSE))
  }
}

# Changing column names
colnames(fit_table2) <- c("Span", "Degree", "RSE")
```

```
# Displaying data frame
fit_table2
```

| | Span | Degree | RSE |
|----|------|--------|----------|
| 1 | 0.25 | 1 | 24.54703 |
| 2 | 0.30 | 1 | 26.63732 |
| 3 | 0.35 | 1 | 28.88638 |
| 4 | 0.40 | 1 | 30.88313 |
| 5 | 0.45 | 1 | 32.57616 |
| 6 | 0.50 | 1 | 34.02896 |
| 7 | 0.55 | 1 | 35.27381 |
| 8 | 0.60 | 1 | 36.36153 |
| 9 | 0.65 | 1 | 37.32657 |
| 10 | 0.70 | 1 | 38.18843 |
| 11 | 0.75 | 1 | 38.96106 |
| 12 | 0.25 | 2 | 21.66747 |
| 13 | 0.30 | 2 | 21.99463 |
| 14 | 0.35 | 2 | 22.42284 |
| 15 | 0.40 | 2 | 23.12502 |
| 16 | 0.45 | 2 | 24.29479 |
| 17 | 0.50 | 2 | 25.87832 |
| 18 | 0.55 | 2 | 27.50811 |
| 19 | 0.60 | 2 | 29.01310 |
| 20 | 0.65 | 2 | 30.39499 |
| 21 | 0.70 | 2 | 31.68249 |
| 22 | 0.75 | 2 | 32.87589 |

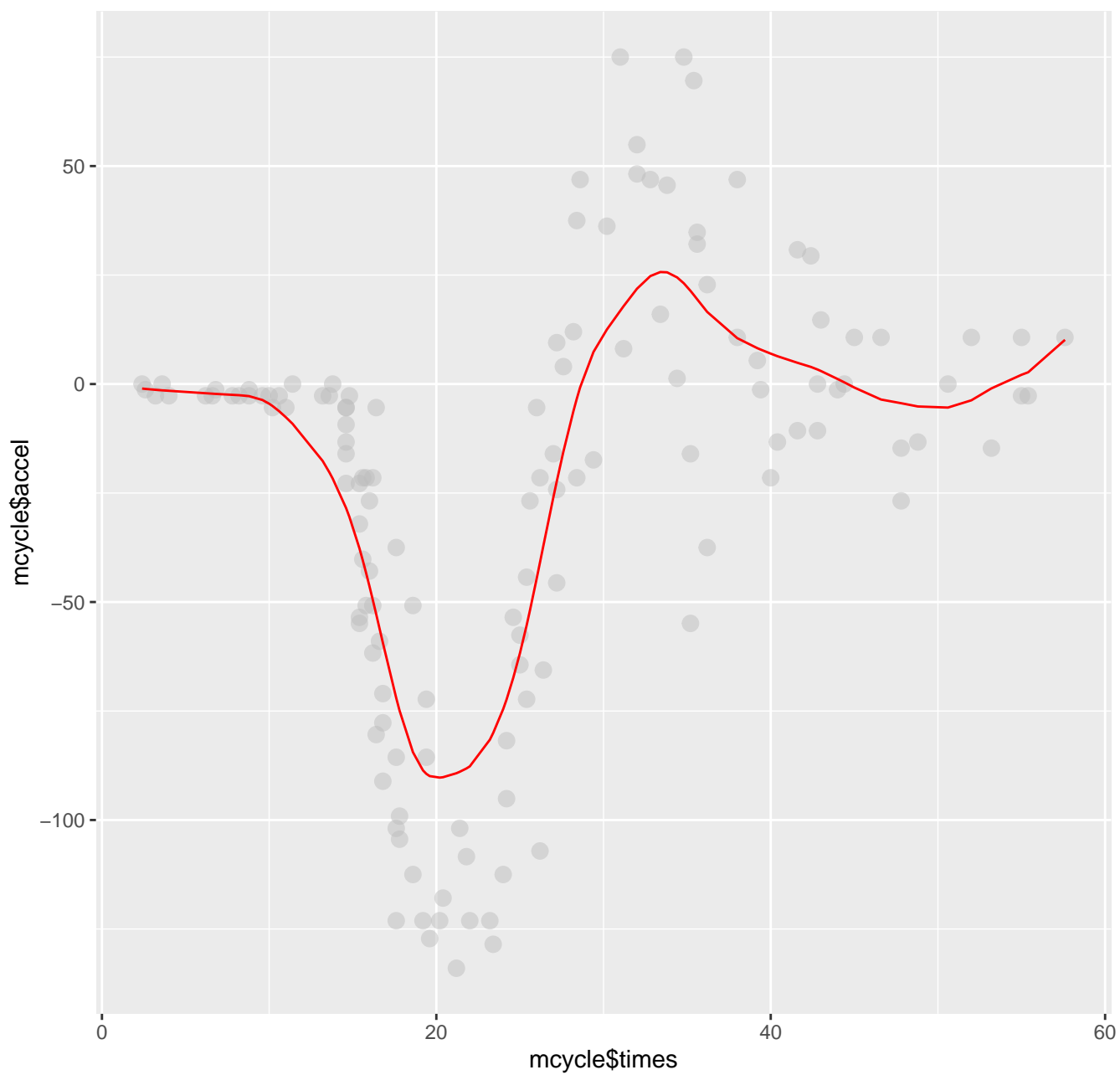
The three “best” fits with degree 1 appear to be span 0.25, 0.30, and 0.35. The three “best” fits with degree 2 appear to be span 0.25, 0.30, and 0.35.

Plotting the best fits found above

```
myloess(mcycle$times, mcycle$accel,
```

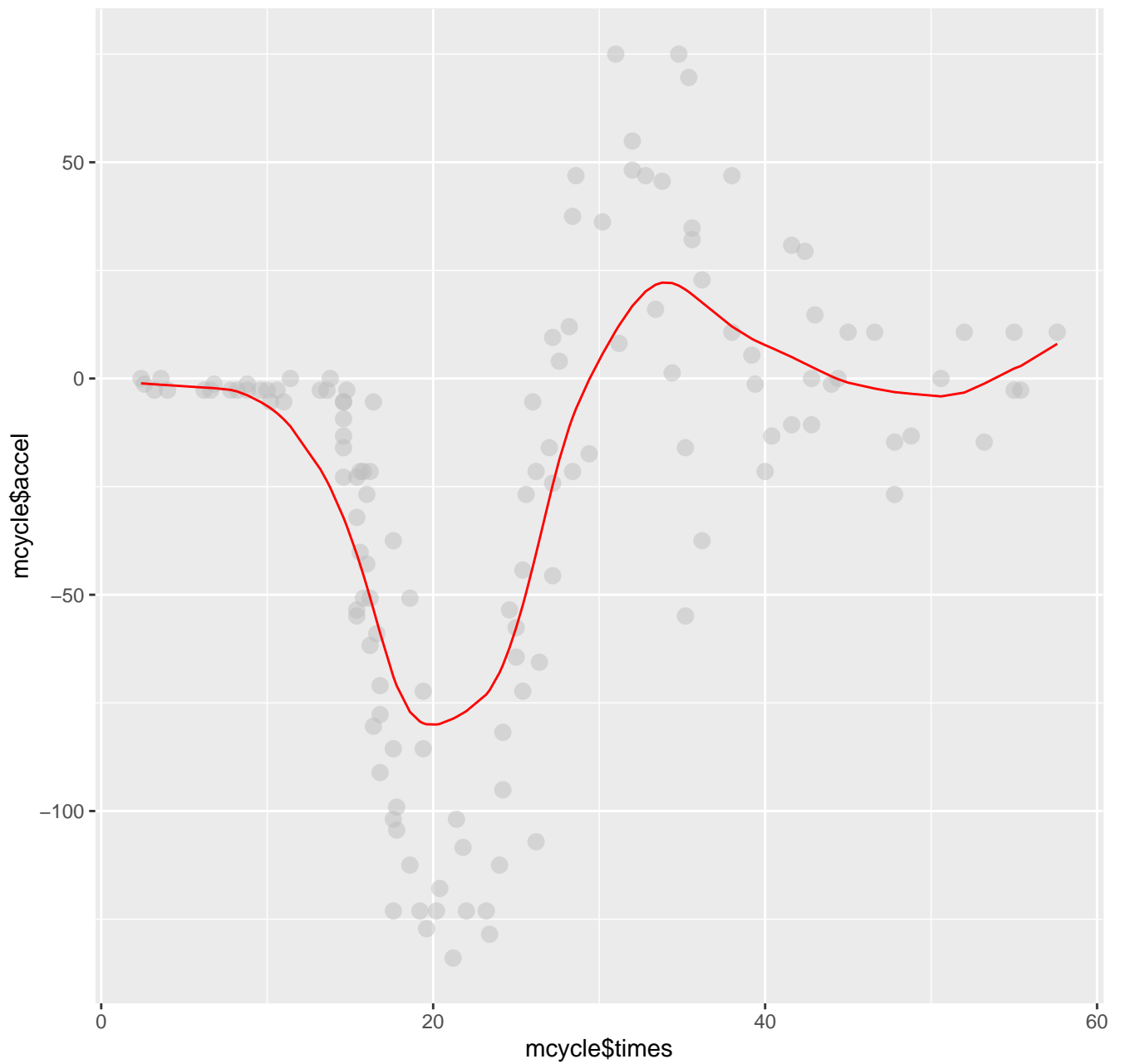
```
span = 0.25, degree = 1, show.plot = F)$loessplot
```


LOESS with degree = 1 and span = 0.25



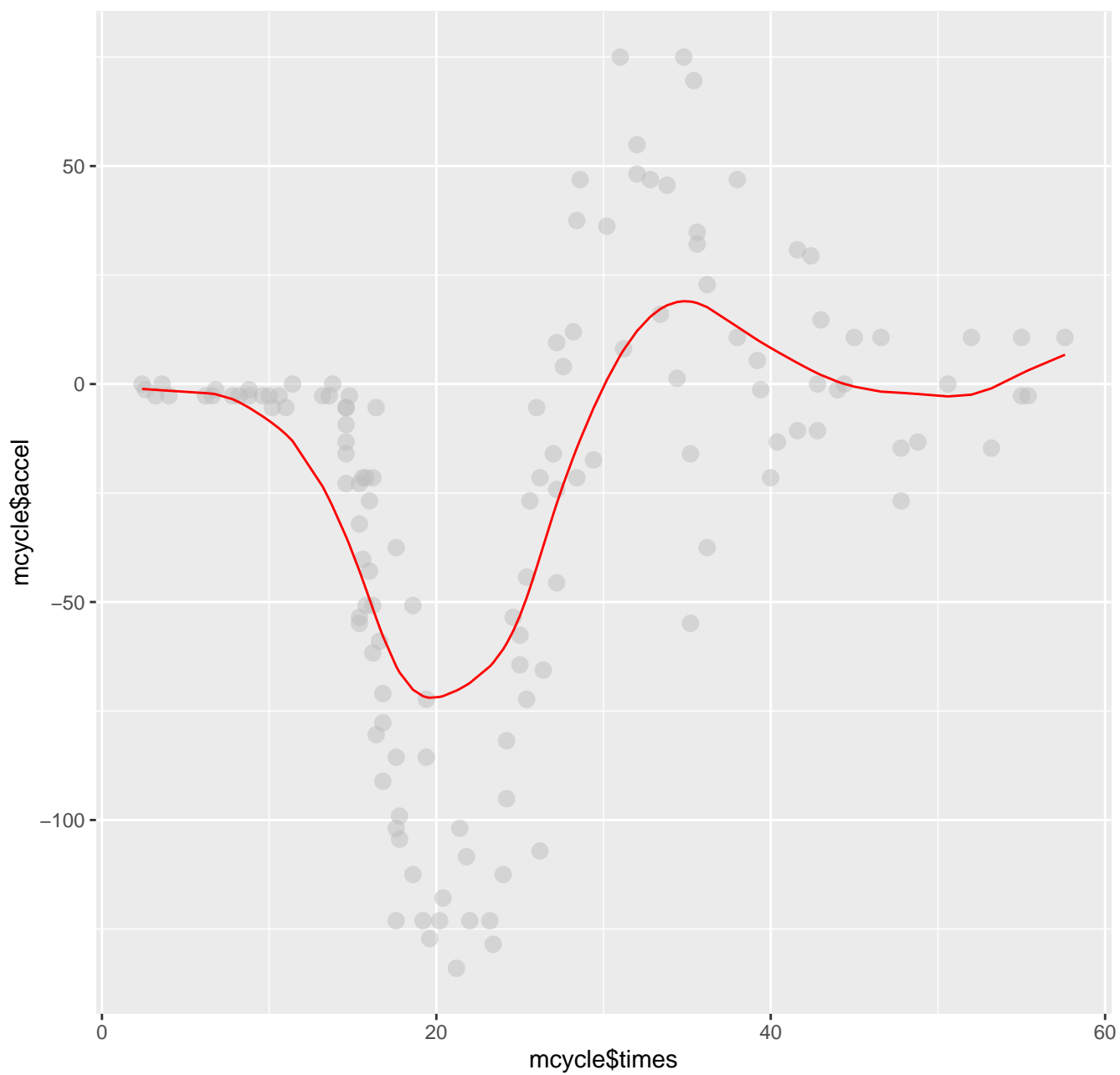
```
myloess(mcycle$times, mcycle$accel,  
span = 0.30, degree = 1, show.plot = F)$loessplot
```

LOESS with degree = 1 and span = 0.3



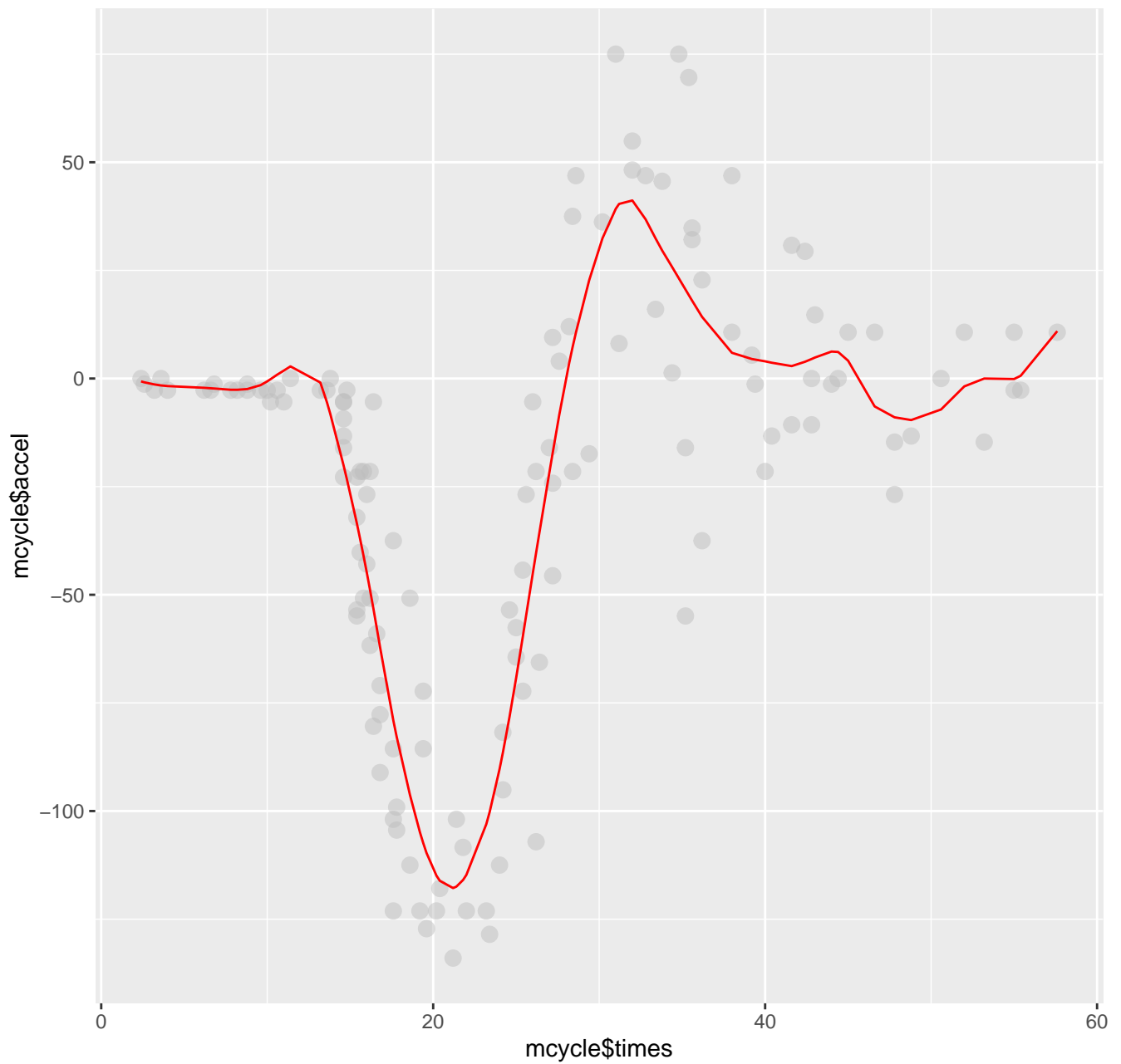
```
myloess(mcycle$times, mcycle$accel,  
span = 0.35, degree = 1, show.plot = F)$loessplot
```

LOESS with degree = 1 and span = 0.35



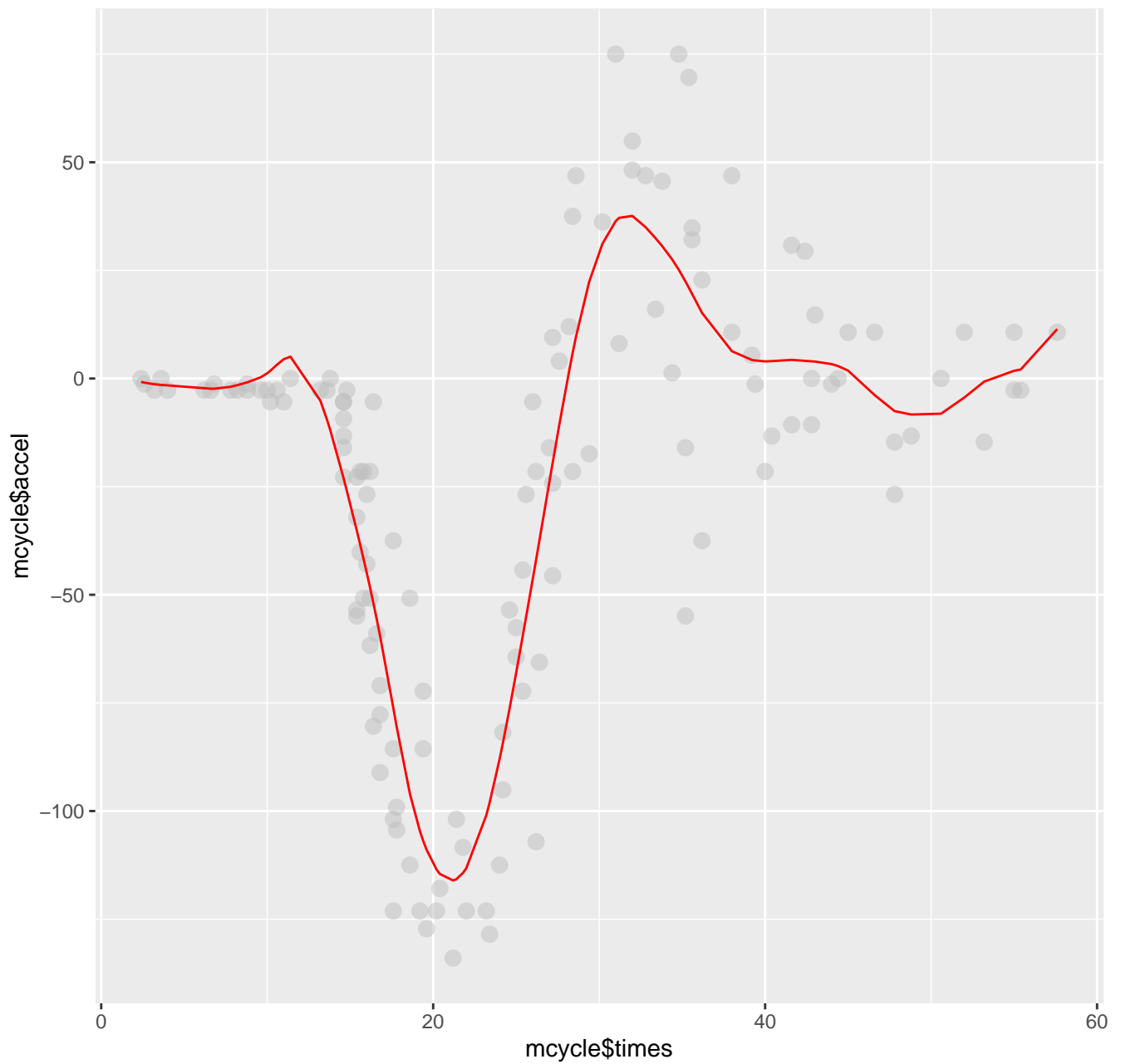
```
myloess(mcycle$times, mcycle$accel,  
span = 0.25, degree = 2, show.plot = F)$loessplot
```

LOESS with degree = 2 and span = 0.25



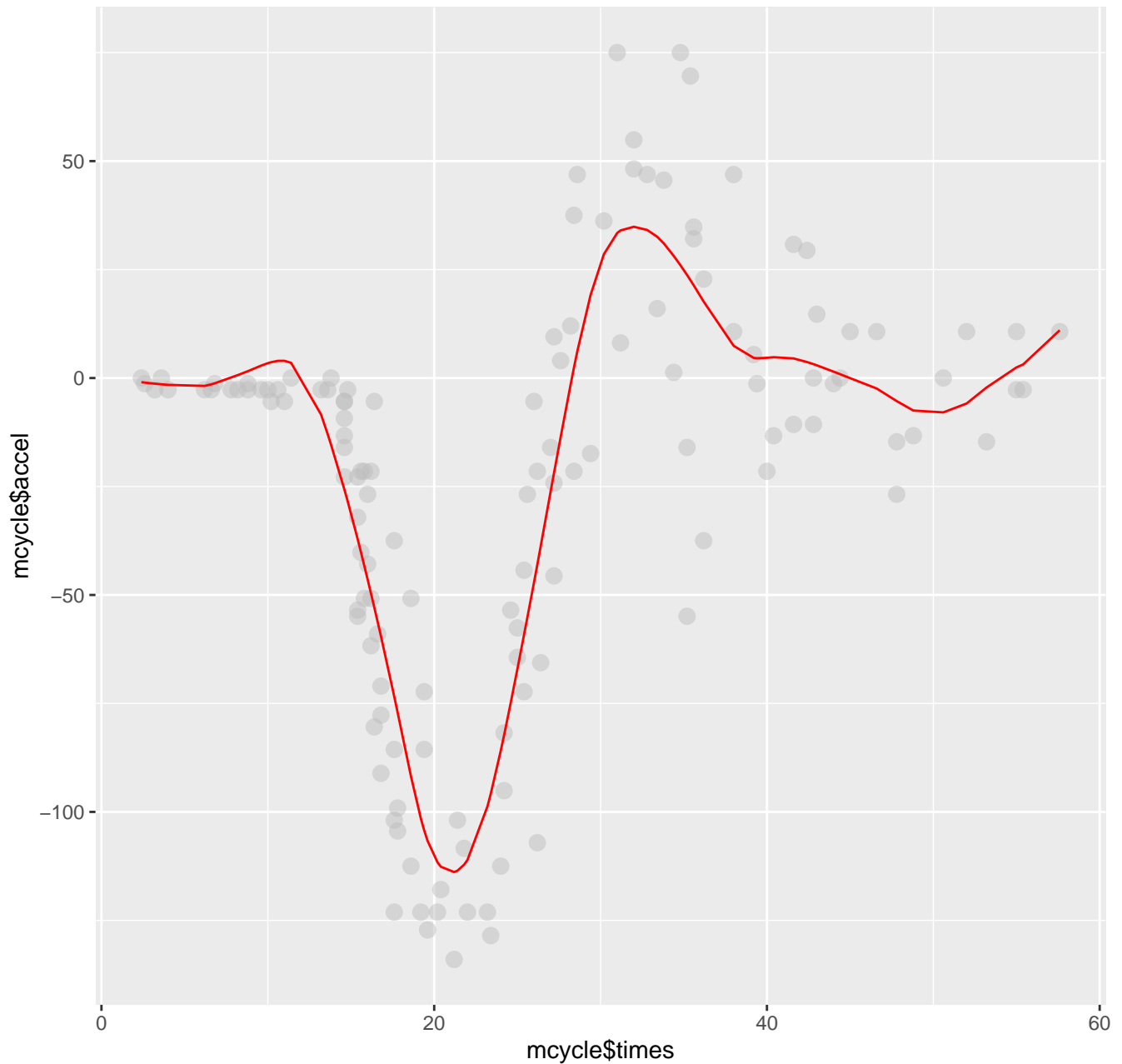
```
myloess(mcycle$times, mcycle$accel,  
span = 0.30, degree = 2, show.plot = F)$loessplot
```

LOESS with degree = 2 and span = 0.3



```
myloess(mcycle$times, mcycle$accel,  
span = 0.35, degree = 2, show.plot = F)$loessplot
```

LOESS with degree = 2 and span = 0.35

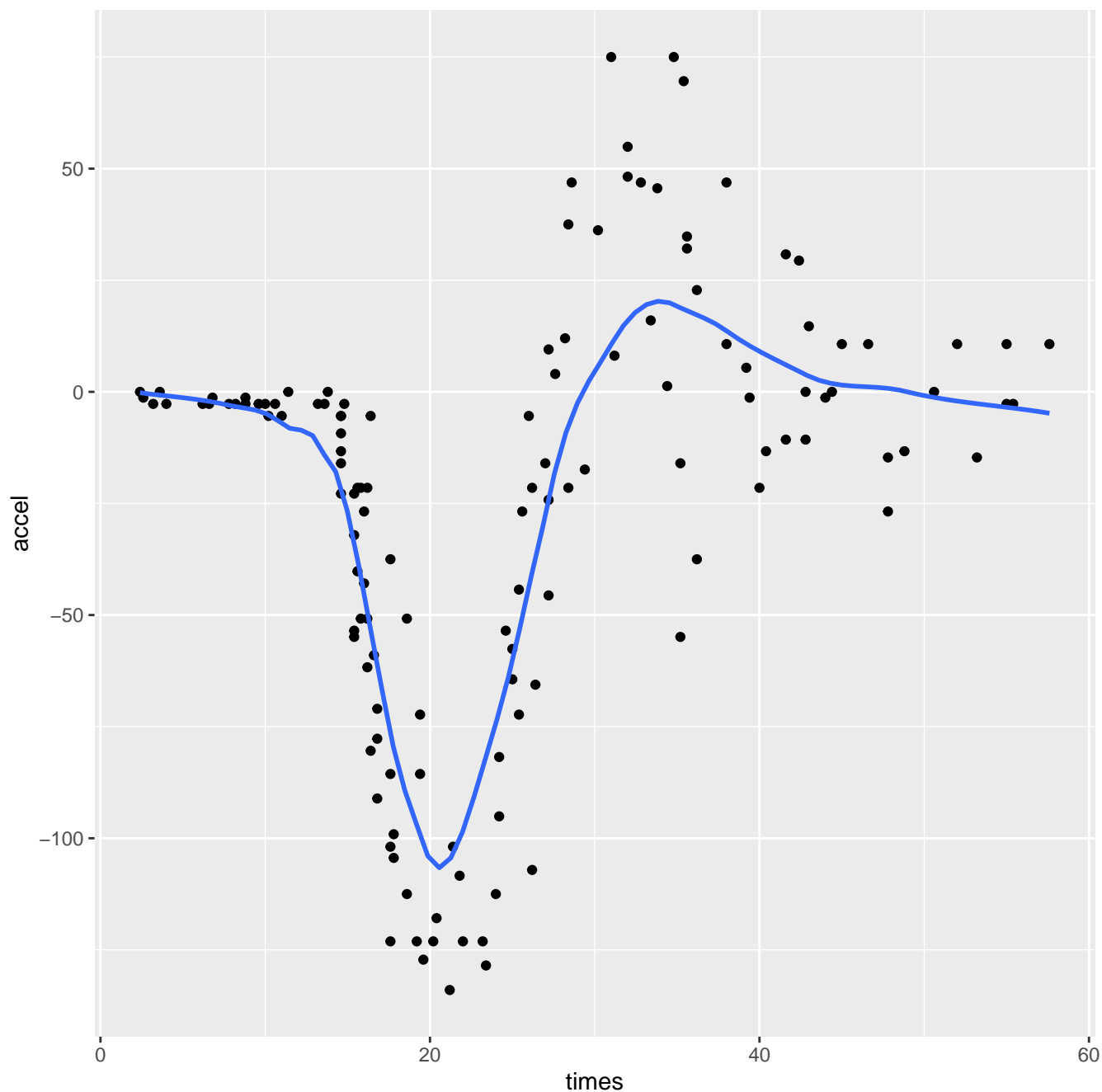


Visually inspecting the models, we believe that the model with degree 2 and span 0.35 provided the “best” fit.

Part b

Comparing results with built-in LOESS function

```
mcycle %>% ggplot(aes(times, accel)) +  
  geom_point() +  
  geom_smooth(method = "loess", degree = 2, span = 0.35, se = F, method.args = list(degree=1))
```



The built-in LOESS function results in a fit that does not over-fit the data, but we feel our model provides the better fit.

Problem 3

```
# Some pre-processing
library(ISLR)
# Remove the name of the car model and change the origin to categorical with actual name
Auto_new <- Auto[, -9]
# Lookup table
newOrigin <- c("USA", "European", "Japanese")
Auto_new$origin <- factor(newOrigin[Auto_new$origin], newOrigin)

# Look at the first 6 observations to see the final version
head(Auto_new)
```

| | mpg | cylinders | displacement | horsepower | weight | acceleration | year | origin |
|---|-----|-----------|--------------|------------|--------|--------------|------|--------|
| 1 | 18 | 8 | 307 | 130 | 3504 | 12.0 | 70 | USA |

| | | | | | | | | |
|---|----|---|-----|-----|------|------|----|-----|
| 2 | 15 | 8 | 350 | 165 | 3693 | 11.5 | 70 | USA |
| 3 | 18 | 8 | 318 | 150 | 3436 | 11.0 | 70 | USA |
| 4 | 16 | 8 | 304 | 150 | 3433 | 12.0 | 70 | USA |
| 5 | 17 | 8 | 302 | 140 | 3449 | 10.5 | 70 | USA |
| 6 | 15 | 8 | 429 | 198 | 4341 | 10.0 | 70 | USA |

Writing the function that carries out kNN classification and regression

```
library(caret)

# This function has the following inputs
#
# * train - matrix or data frame of training set cases
# * test - matrix or data frame of test set cases.
#   (A vector will be interpreted as a row vector for a single case.)
# * y_train - Either a numeric vector, or factor vector for the responses in the training set
# * y_test - Either a numeric vector, or factor vector for the responses in the testing set
# * k - number of neighbors considered, the default value is 3
#
# If weighted = TRUE, then function uses the distance weighted kNN,
# otherwise it does the default kNN method.

mykNN <- function(train, test, y_train, y_test, k = 3, weighted = TRUE) {

  # Allocating space for vector of fitted values
  yhat <- vector(mode = "numeric", length = length(y_test))

  # Fitting each point
  for (row in 1:nrow(test)) {

    # Getting features
    x0 <- test[row,]

    # Getting Euclidean distance and weights
    distmat <- as.matrix(dist(rbind(x0, train), method = "euclidean", diag = T, upper = T))
    dist <- distmat[1,]
    weights <- 1/dist

    # Getting kth nearest neighbor
    a.order <- order(weights, decreasing = T)[-1]
    last <- weights[a.order[k]]

    # Setting weights to 0 except for kNN
    y_train_sub <- y_train[-which(!is.finite(weights))]
    weights <- weights[-which(!is.finite(weights))]
    weights[weights < last] <- 0

    # Handling unweighted case
    if(weighted == F) {
      weights[weights != 0] <- 1
    }

    tot_weight <- sum(weights)

    # Handling classification
    if (is.factor(y_train)) {

      # Allocating space for vector of class weights
      class <- vector(mode = "numeric", length = nlevels(y_test))

      # Summing weights for each class
```



```

    for (i in 1:nlevels(y_test)) {
      class[i] <- sum(weights[which(y_train == levels(y_test)[i])])
    }

    # Assigning factor with highest class weight to yhat
    yhat[row] <- levels(y_test)[which(class == max(class))]
  }

  # Handling regression
  else {

    # Calculating yhat
    yhat[row] <- sum(weights*y_train/tot_weight)
  }
}

# Handling classification
if (is.factor(y_train)) {

  yhat <- factor(yhat)

  # Setting number of correct predictions to 0
  num_correct <- 0

  # Getting number of correct predictions
  for (j in 1:length(yhat)) {

    # Checking if classes match
    if (y_test[j] == yhat[j]) {

      num_correct <- num_correct + 1
    }
  }

  # Calculating accuracy and error rate
  accuracy <- num_correct / length(yhat)
  error_rate <- 1 - accuracy

  # Getting confusion matrix
  confmat <- confusionMatrix(yhat, y_test)

  # Returning named list
  return(invisible(list(yhat = yhat, accuracy = accuracy,
                        error_rate = error_rate, confmat = confmat, k = k)))
}

# Handling regression
else {

  # Calculating residuals, SSE, MSE, and residual standard error
  residuals <- y_test - yhat
  SSE <- sum(residuals^2)
  MSE <- SSE/(nrow(test)-2)
  RSE <- sqrt(MSE)

  # Returning named list
  return(invisible(list(yhat = yhat, residuals = residuals, SSE = SSE, k = k)))
}
}

```

Splitting data into training data and testing data

```
ntrain <- round(nrow(Auto_new)*0.7)
ntest <- nrow(Auto_new) - ntrain
train <- Auto_new[1:ntrain,1:7]
test <- Auto_new[(ntrain+1):nrow(Auto_new),1:7]
y_train <- Auto_new[1:ntrain,8]
y_test <- Auto_new[(ntrain+1):nrow(Auto_new),8]
```

Determining accuracy of kNN function on the data

```
# Creating an empty data frame
```

```
acc_table <- data.frame()
```

```
# Determining fits and putting info into data frame
```

```
for (i in c(1, 3, 5, 10, 20)) {
  acc_table <- rbind(acc_table, c("False", i, mykNN(train, test, y_train, y_test, k = i, weighted = F)$accuracy))
  acc_table <- rbind(acc_table, c("True", i, mykNN(train, test, y_train, y_test, k = i)$accuracy))
}
```

```
# Changing column names
```

```
colnames(acc_table) <- c("Weighted", "k", "Accuracy")
```

```
# Displaying data frame
```

```
acc_table
```

| | Weighted | k | Accuracy |
|----|----------|----|-------------------|
| 1 | False | 1 | 0.559322033898305 |
| 2 | True | 1 | 0.559322033898305 |
| 3 | False | 3 | 0.542372881355932 |
| 4 | True | 3 | 0.576271186440678 |
| 5 | False | 5 | 0.559322033898305 |
| 6 | True | 5 | 0.550847457627119 |
| 7 | False | 10 | 0.593220338983051 |
| 8 | True | 10 | 0.567796610169492 |
| 9 | False | 20 | 0.550847457627119 |
| 10 | True | 20 | 0.584745762711864 |

```
# Displaying confusion matrix of best fit
```

```
best <- mykNN(train, test, y_train, y_test, k = 9, weighted = F)
best$confmat
```

Confusion Matrix and Statistics

| | Reference | | |
|------------|-----------|----------|----------|
| Prediction | USA | European | Japanese |
| USA | 47 | 11 | 9 |
| European | 9 | 5 | 8 |
| Japanese | 5 | 4 | 20 |

Overall Statistics

```
Accuracy : 0.6102
95% CI : (0.5161, 0.6986)
No Information Rate : 0.5169
P-Value [Acc > NIR] : 0.02613
```

```
Kappa : 0.3479
```

```
McNemar's Test P-Value : 0.44429
```

Statistics by Class:

| | Class: USA | Class: European | Class: Japanese |
|----------------------|------------|-----------------|-----------------|
| Sensitivity | 0.7705 | 0.25000 | 0.5405 |
| Specificity | 0.6491 | 0.82653 | 0.8889 |
| Pos Pred Value | 0.7015 | 0.22727 | 0.6897 |
| Neg Pred Value | 0.7255 | 0.84375 | 0.8090 |
| Prevalence | 0.5169 | 0.16949 | 0.3136 |
| Detection Rate | 0.3983 | 0.04237 | 0.1695 |
| Detection Prevalence | 0.5678 | 0.18644 | 0.2458 |
| Balanced Accuracy | 0.7098 | 0.53827 | 0.7147 |

Problem 4

Part a

```
ntrain <- round(nrow(ozone)*0.63)
ntest <- nrow(ozone) - ntrain
train <- ozone[1:ntrain,3]
test <- ozone[(ntrain+1):nrow(ozone),3]
y_train <- ozone[1:ntrain,1]
y_test <- ozone[(ntrain+1):nrow(ozone),1]

'# Creating an empty data frame
acc_table2 <- data.frame()

# Determining fits and putting info into data frame
for (i in c(1, 3, 5, 10, 20)) {
  acc_table2 <- rbind(acc_table2, c("False", i, mykNN(train, test, y_train, y_test, k = i, weighted = F)$SSE))
  acc_table2 <- rbind(acc_table2, c("True", i, mykNN(train, test, y_train, y_test, k = i)$SSE))
}

# Changing column names
colnames(acc_table2) <- c("Weighted", "k", "SSE")

# Displaying data frame
acc_table2'
```

```
[1] "# Creating an empty data frame\nacc_table2 <- data.frame()\n\n# Determining fits and putting info into da
```

Part b

```
'best <- mykNN(train, test, y_train, y_test, k = 1)'

[1] "best <- mykNN(train, test, y_train, y_test, k = 1)"
```