LOESS regression and distance weighted KNN analysis

Theodore Li

10/18/2024

Part 1: Implementing a custom LOESS regression

```
#A helper function to calculate the weights
# Using the Tukey tri-weight equation from:

# https://rafalab.dfci.harvard.edu/dsbook/smoothing.html#local-weighted-regression-loess
#Inputs:
# x_i: the specific neighbor point whose weight we're determining
# x_0: the fixed data point
# window_size: size of the window for loess
#
#return
#weight: numeric weight corresponding to x_i
tri_weight_func <- function(x_i, x_0, window_size){

  h <- floor(window_size/2)
  u = (x_i - x_0)/h

  # Tukey tri-weight formula: W(u) = (1 - u^3)^3 for u in [0, 1]
  if (abs(u) <= 1) {
    weight <- (1 - abs(u)^3)^3
  } else {
    weight <- 0 # Points outside the window get a weight of 0
  }

  return(weight)
}

# Input:
# * x - a numeric input vector
# * y - a numeric response
# * degree should be 1 or 2 only
# * span can be any value in interval (0, 1) non-inclusive.
#
# If show.plot = TRUE then you must show a plot of either the final fit

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

  #total number of input data
  n <- length(x)

  #Placeholder vector to store predictions
  fitted_values <- numeric(n)

  #Create a for loop to iterate through each data point
  for (i in 1:n) {
    #Get the average distance between x[i] and all other points
```

```r
  distance <- abs(x - x[i])

  #Get the window size from the argument span, we take
  # the greatest integer <= span * n
  # This is what breaks our data into bins for us to build a regression
  # curve on.
  window_size <- floor(span * n)

  #We order the distance from closest to furthest and
  #take a total of "window_size" number of neighbor points as a subset of input points
  neighbors <- order(distance)[1:window_size]

  #Find the weights corresponding to the neighbors of x
  weights <- sapply(x[neighbors], tri_weight_func, x_0 = x[i], window_size = window_size)

  # Create the design matrix used for calculations of coefficients (based on degree)
  if (degree == 1) {
    #X_mat is a matrix with 2 columns
    #Create a column of 1s for the intercept
    #Create column of x values for corresponding neighbor points
    X_mat <- cbind(1, x[neighbors])
  } else if (degree == 2) {
    #X_mat is a matrix with 3 columns
    #Column 1: intercept
    #Column 2: x values
    #Column 3: x^2 values
    X_mat <- cbind(1, x[neighbors], x[neighbors]^2)
  }
  #Turn vector of weights into a weight matrix
  W <- diag(weights)

  #Follow the formula on Lecture 12 slide 5 to solve for coefficients (beta)
  beta <- solve(t(X_mat) %*% W %*% X_mat) %*% (t(X_mat) %*% W %*% y[neighbors])

  #Compute the predictions based on corresponding degree argument
   if (degree == 1) {
    fitted_values[i] <- beta[1] + beta[2] * x[i]  # Linear prediction
  } else if (degree == 2) {
    fitted_values[i] <- beta[1] + beta[2] * x[i] + beta[3] * x[i]^2  # polynomial prediction
  }
}

# Calculate the residuals
residuals <- y - fitted_values
#Use residuals for calculating SSE
SSE <- sum(residuals^2)

#Calculate MSE
MSE <- SSE/n

#Create the plot
plot <- ggplot(data = data.frame(x, y), aes(x = x, y = y)) +
geom_point(color = "blue") +                          # Scatter plot
geom_line(aes(y = fitted_values), color = "red", lwd = 1.5) +  # Fitted values line
ggtitle("Custom LOESS") +
xlab("x") +
ylab("y") +
annotate("text", x = max(x), y = max(y), label = paste("Span:", span, "Degree:", degree),
         hjust = 1, vjust = 1, size = 4, color = "black", fontface = "bold")
```

```
  # Display the plot if desired
  if (show.plot) {
    print(plot)
  }


  return(list("span" = span,
              "fitted_values" = fitted_values,
              "degree" = degree,
              "N_total" = n,
              "MSE" = MSE,
              "SSE" = SSE,
              "loessplot" = plot))
}

library(ggplot2)

#Load in the ozone data
load("C:/Users/theod/OneDrive/Documents/CMDA_4654/Exercise 2/ozone.RData")
```

1.)

```
x <- ozone$temperature
y <- ozone$ozone
n <- length(x)

#Iterate through degrees of 1 to 6 and print the MSE to assess performance
for (i in 1:6) {
  model <- lm(y ~ poly(x, degree = i, raw = TRUE))
  MSE <- mean(model$residuals^2)
  cat("The MSE of polynomial model degree ",i," is:", MSE, "\n")
}
```

```
The MSE of polynomial model degree  1  is: 561.8688
The MSE of polynomial model degree  2  is: 501.8821
The MSE of polynomial model degree  3  is: 496.0853
The MSE of polynomial model degree  4  is: 467.9096
The MSE of polynomial model degree  5  is: 466.8294
The MSE of polynomial model degree  6  is: 466.8266
```

The polynomial with degree 6 seems to work the best as it has the lowest MSE: The reason I chose to evaluate the model with MSE is because it takes into account the sample size so we find the average difference between label and prediction of each sample.

2.)

```
#Degree 1 LOESS span: 0.25 to 0.75
#Similarly iterate through spans of .25 to .75 and print MSE for performance
for (i in seq(.25, .75, by = 0.05)) {
  output = myloess(x, y, span = i, degree = 1, show.plot = FALSE)
  MSE <- output$MSE
  cat("The MSE of myloess model degree 1, span ",i," is:", MSE, "\n")
}
```

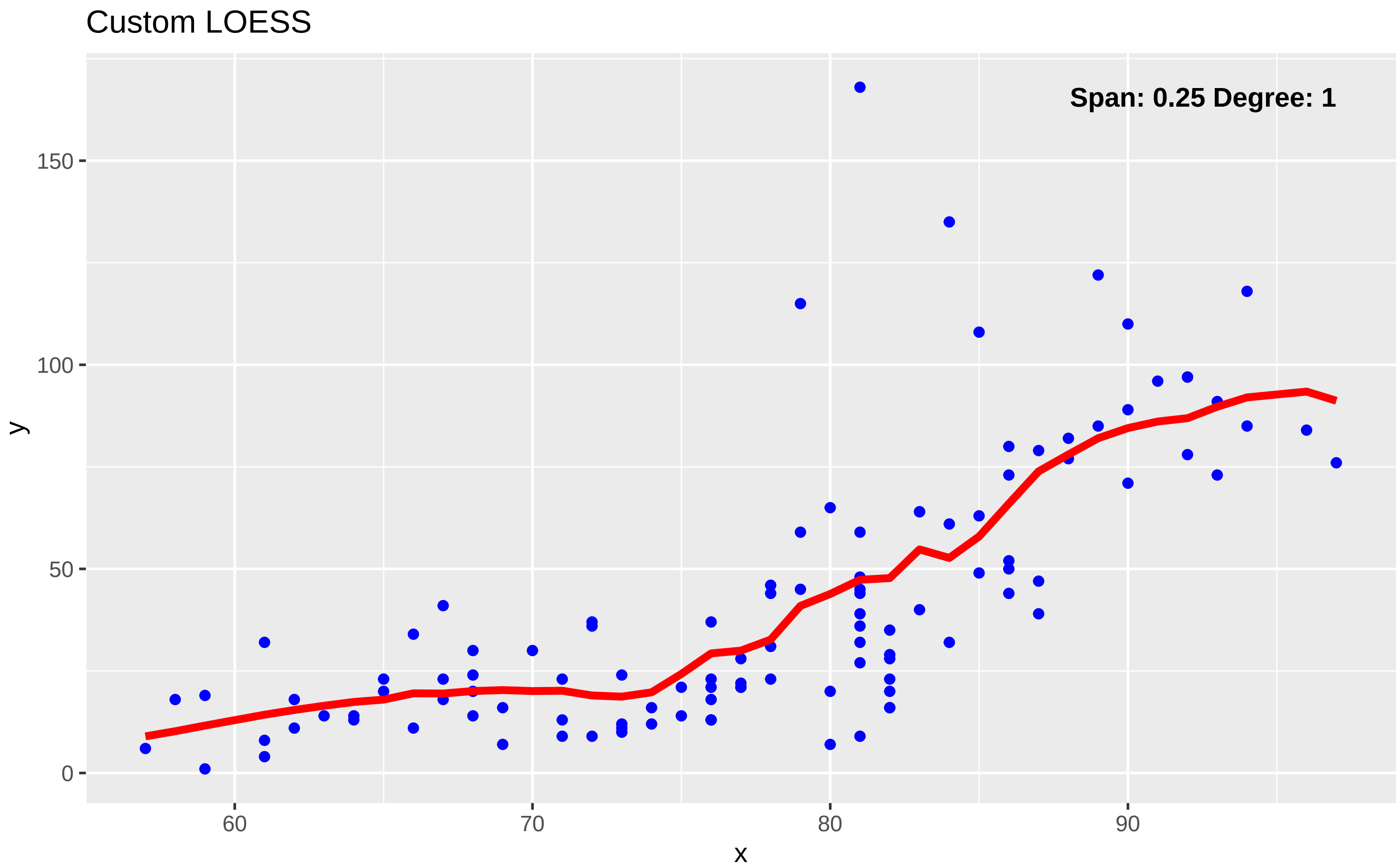
```
The MSE of myloess model degree 1, span  0.25  is: 467.4628
The MSE of myloess model degree 1, span  0.3  is: 483.2206
```

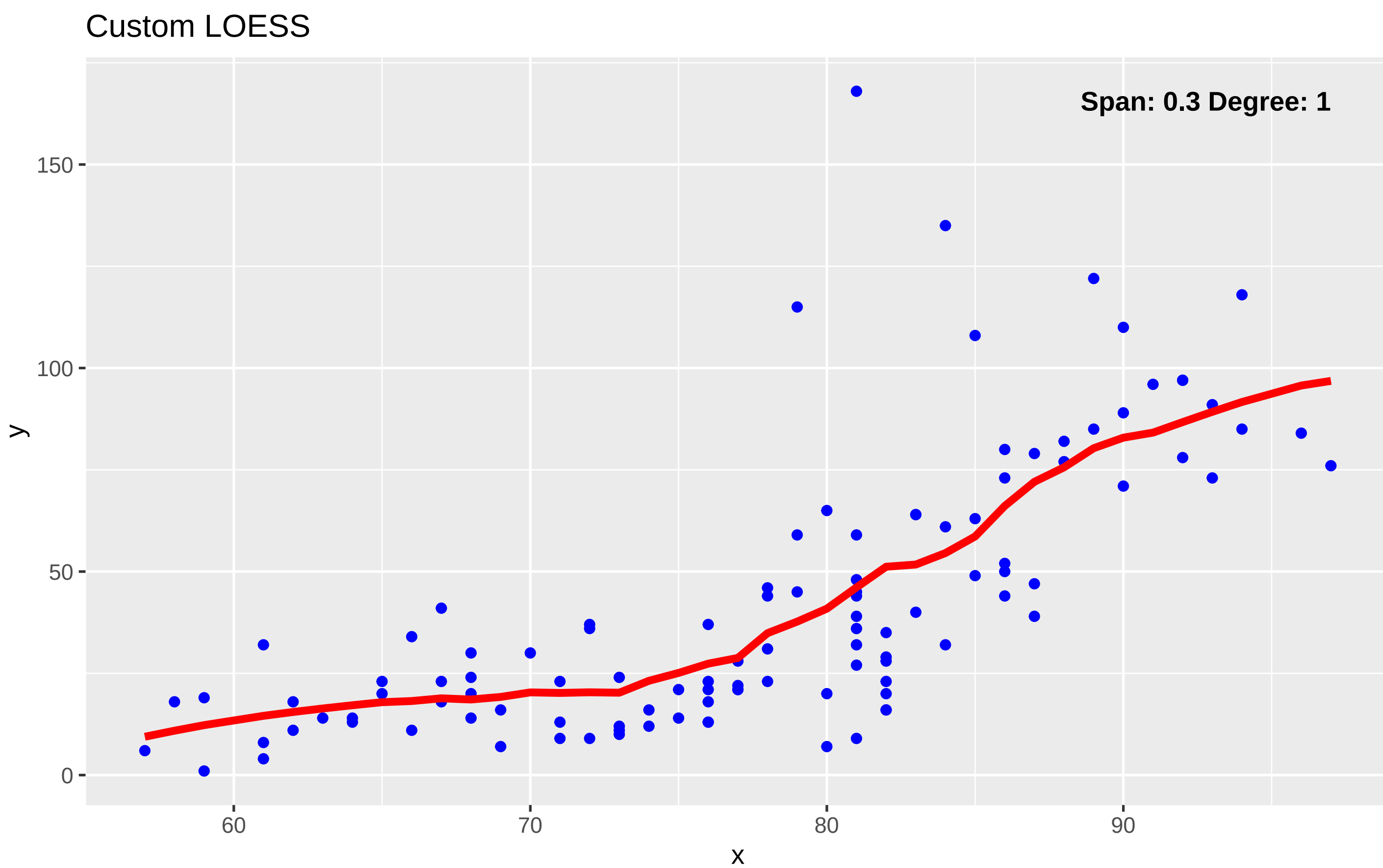
```
The MSE of myloess model degree 1, span  0.35  is: 486.0087
The MSE of myloess model degree 1, span  0.4  is: 489.2471
The MSE of myloess model degree 1, span  0.45  is: 486.5696
The MSE of myloess model degree 1, span  0.5  is: 484.9913
The MSE of myloess model degree 1, span  0.55  is: 490.6261
The MSE of myloess model degree 1, span  0.6  is: 494.9657
The MSE of myloess model degree 1, span  0.65  is: 498.5184
The MSE of myloess model degree 1, span  0.7  is: 497.6645
The MSE of myloess model degree 1, span  0.75  is: 503.1995
```

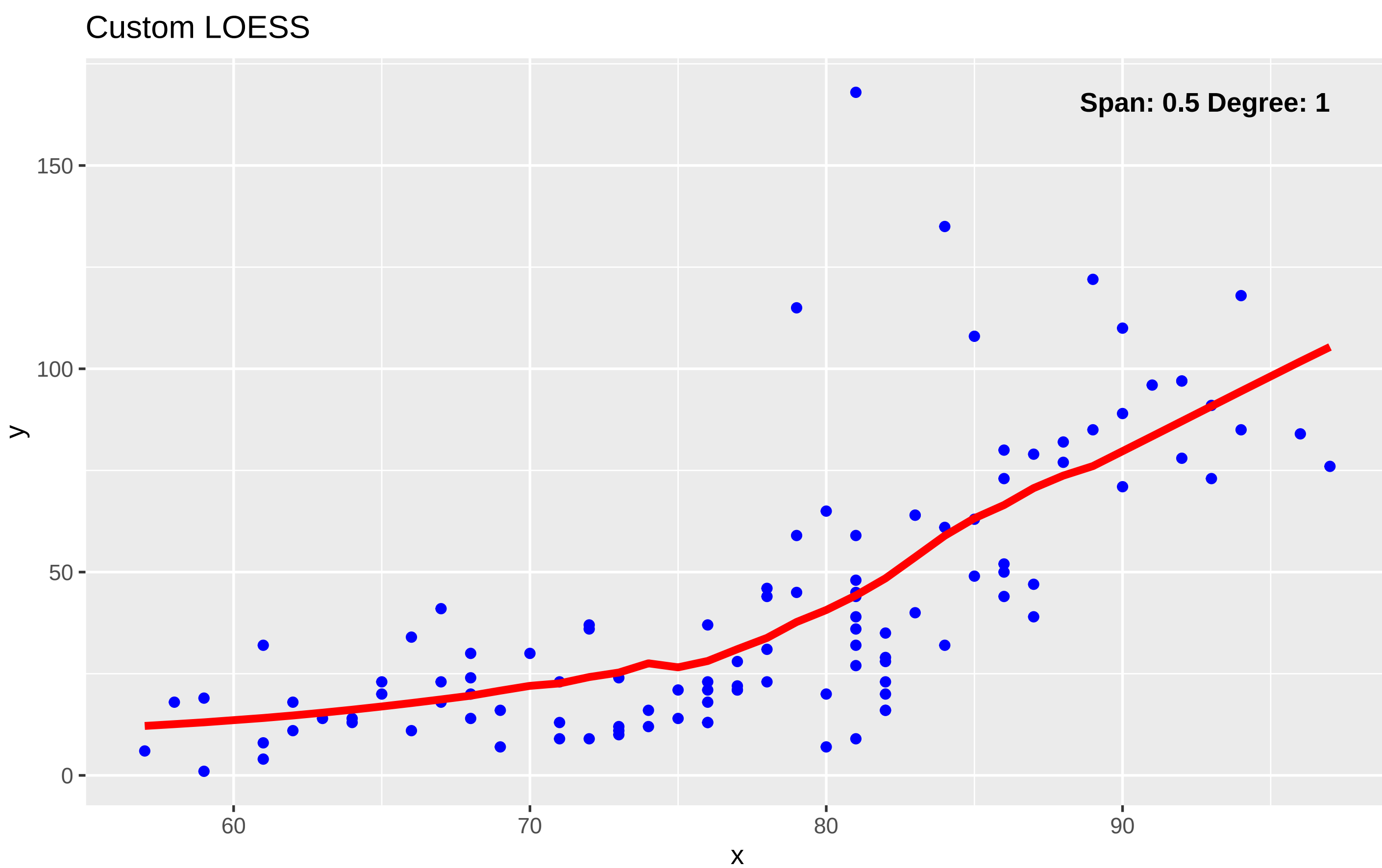
Based on the SSE, the three best degree = 1, fits is: model span 0.25, 0.3, 0.5 as they have the lowest SSE.

Plotting the three models that I deemed the best fit:

```r
best_spans <- c(0.25, 0.3, 0.5)

#Display the plots for the spans we deemed have the best performance
for (span in best_spans) {
  myloess(x, y, span = span, degree = 1, show.plot = TRUE)
}
```








```
#Degree 2 LOESS span: 0.25 to 0.75
#Now iterate through spans .25 to .75 with degree 2
```

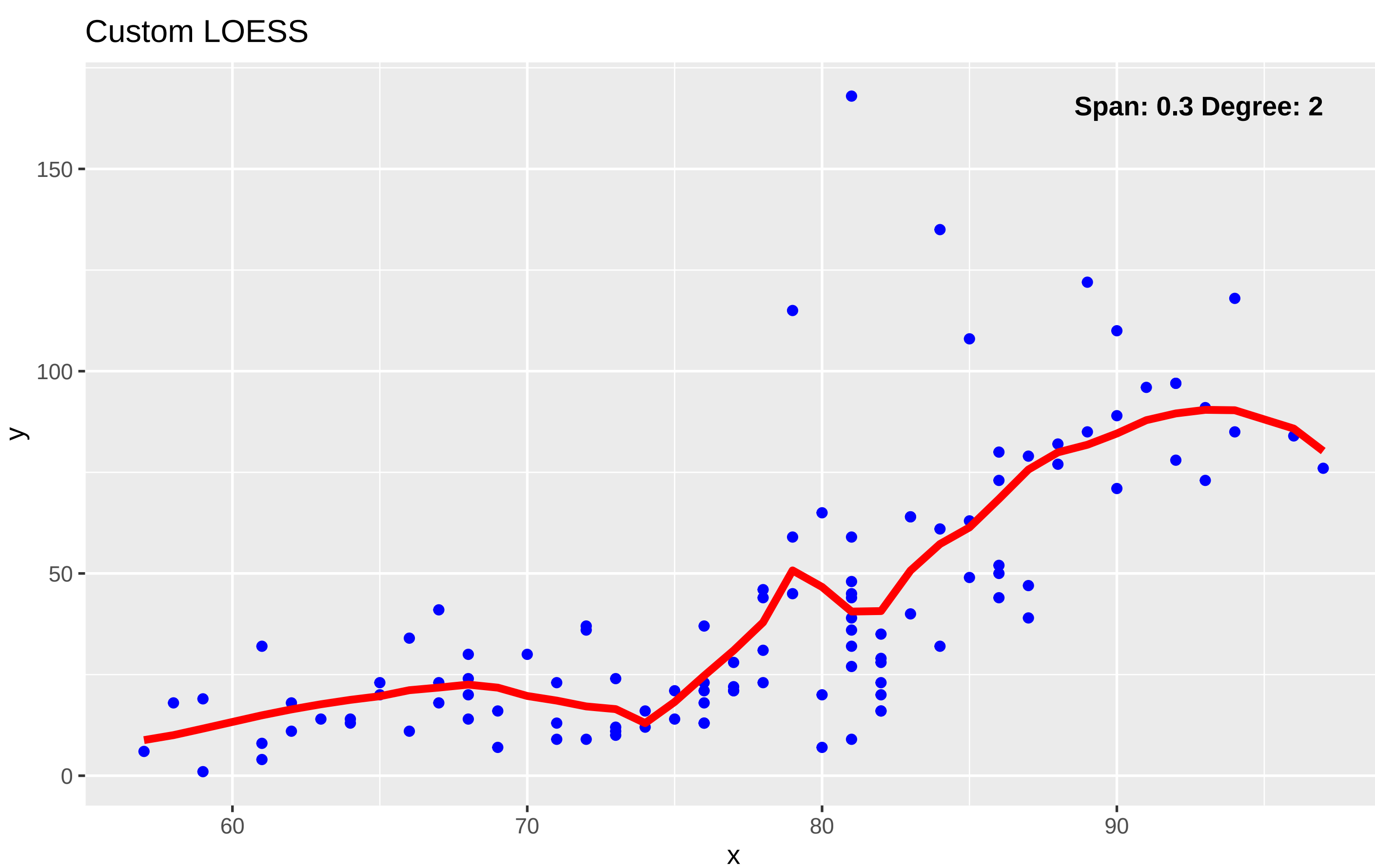
```
for (i in seq(.25, .75, by = 0.05)) {
  output = myloess(x, y, span = i, degree = 2, show.plot = FALSE)
  MSE <- output$MSE
  cat("The MSE of myloess model  degree 2, span ",i," is:", MSE, "\n")
}
```

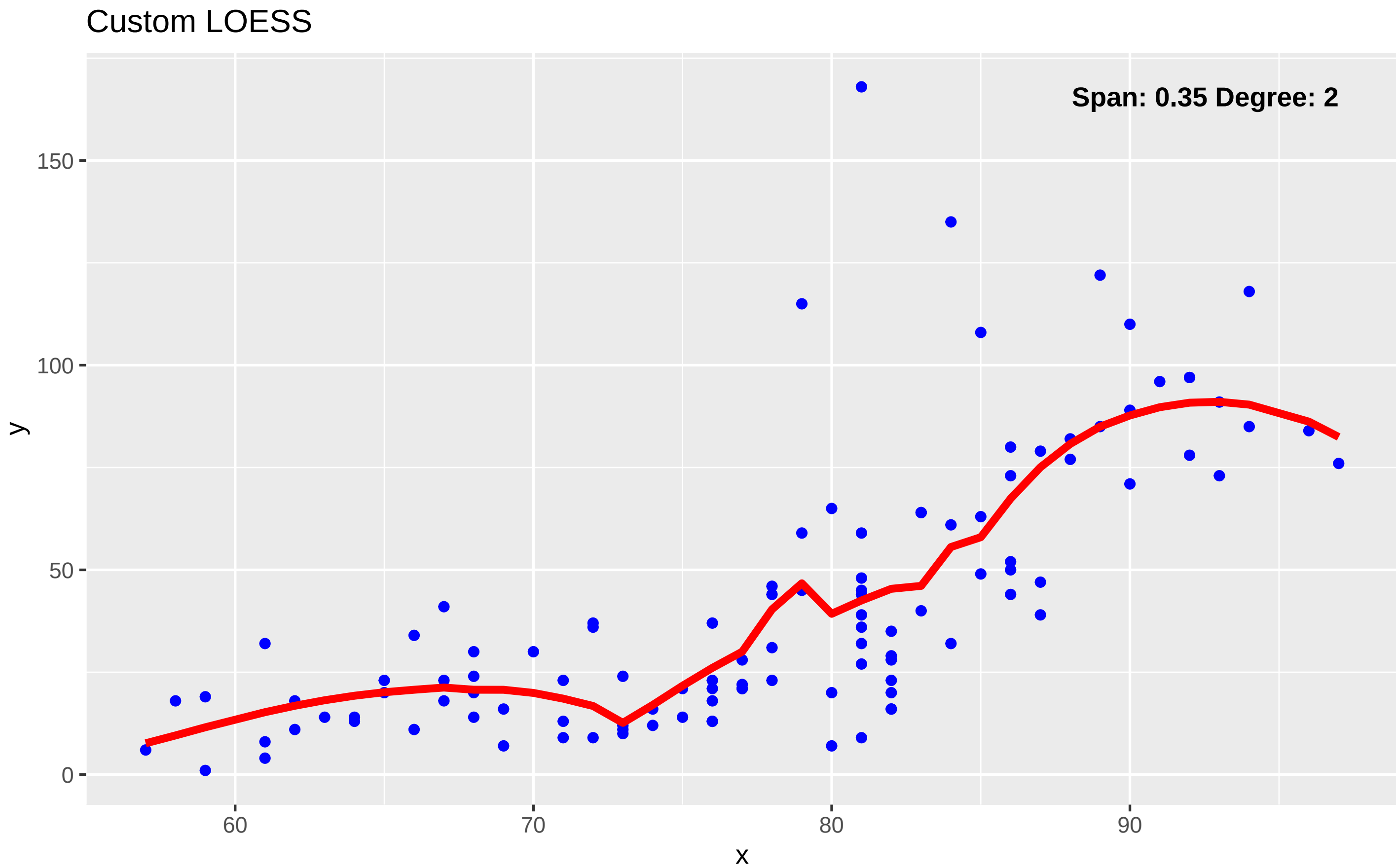
```
The MSE of myloess model  degree 2, span  0.25  is: 452.9573
The MSE of myloess model  degree 2, span  0.3  is: 434.9186
The MSE of myloess model  degree 2, span  0.35  is: 446.6754
The MSE of myloess model  degree 2, span  0.4  is: 474.4457
The MSE of myloess model  degree 2, span  0.45  is: 475.095
The MSE of myloess model  degree 2, span  0.5  is: 478.845
The MSE of myloess model  degree 2, span  0.55  is: 477.7989
The MSE of myloess model  degree 2, span  0.6  is: 476.1987
The MSE of myloess model  degree 2, span  0.65  is: 475.5754
The MSE of myloess model  degree 2, span  0.7  is: 480.9455
The MSE of myloess model  degree 2, span  0.75  is: 487.3245
```

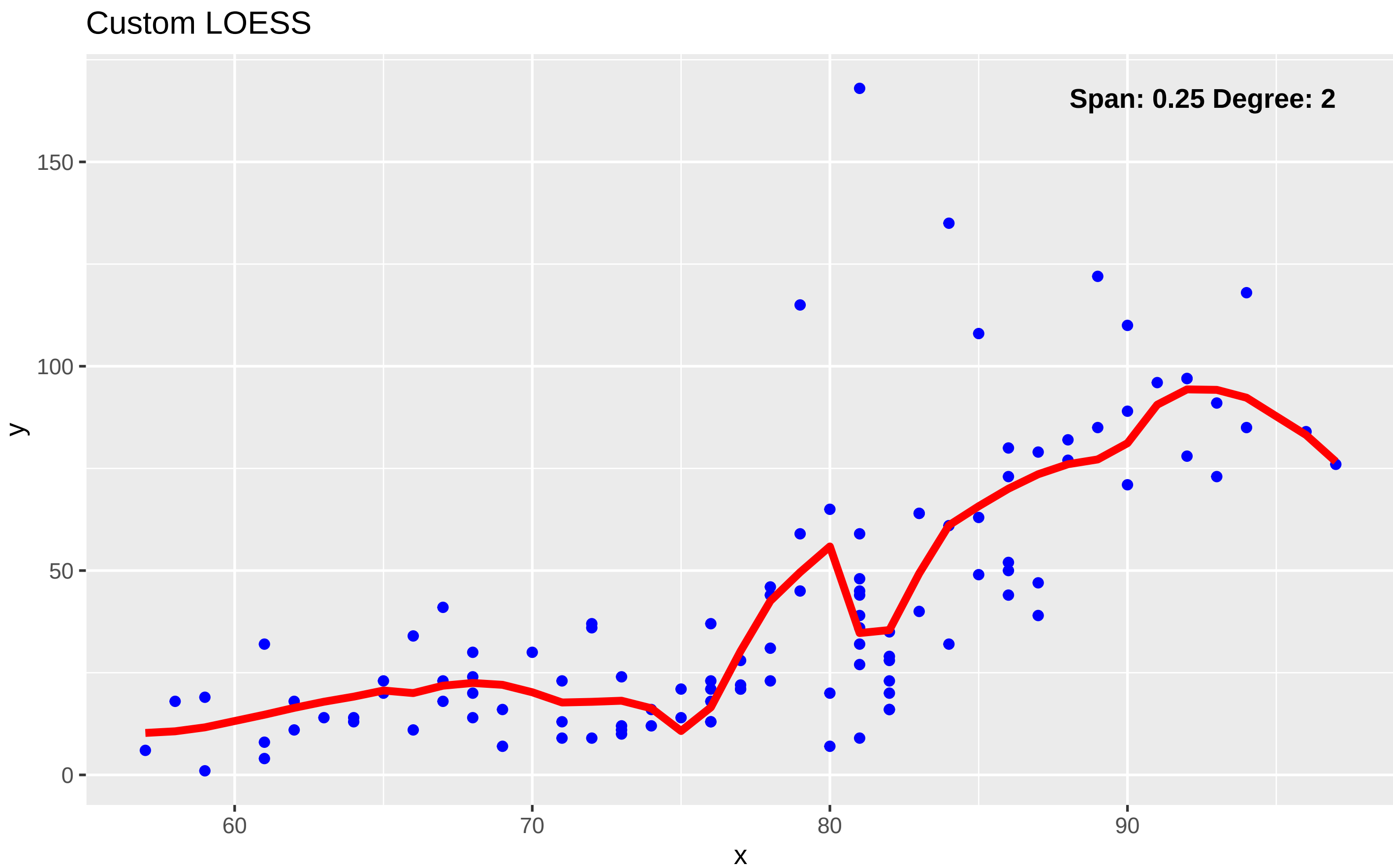
I see that for degree 2 the model with best fits come from span = 0.3, 0.35, 0.25 based on the lowest SSE

Plotting the 3 models with the best fit:

```
best_spans_d2 <- c(0.3, 0.35, 0.25)
for (span in best_spans_d2){
  output = myloess(x, y, span = span, degree = 2, show.plot = TRUE)
}
```







Visually inspecting the plots for Degree = 2 it's not as evident if the plots are overfitting. This is likely due to the model is using a quadratic regression which polynomial which already provides a more curved relationship introducing therefore the span has less of an impact on smoothing the data.

Visually inspecting the plots for Degree = 1 based on a linear regression, I can see that the smaller span = .25, the model does provide a better fit but may be overfitting to the trends in the data. Compared to span = .5 the model has a much more general upward trend that may have better performance when applied on unseen data. The smaller span leads to each prediction being more sensitive to data around it, whereas a larger span smooths out these variations by providing a more general prediction.

3.)

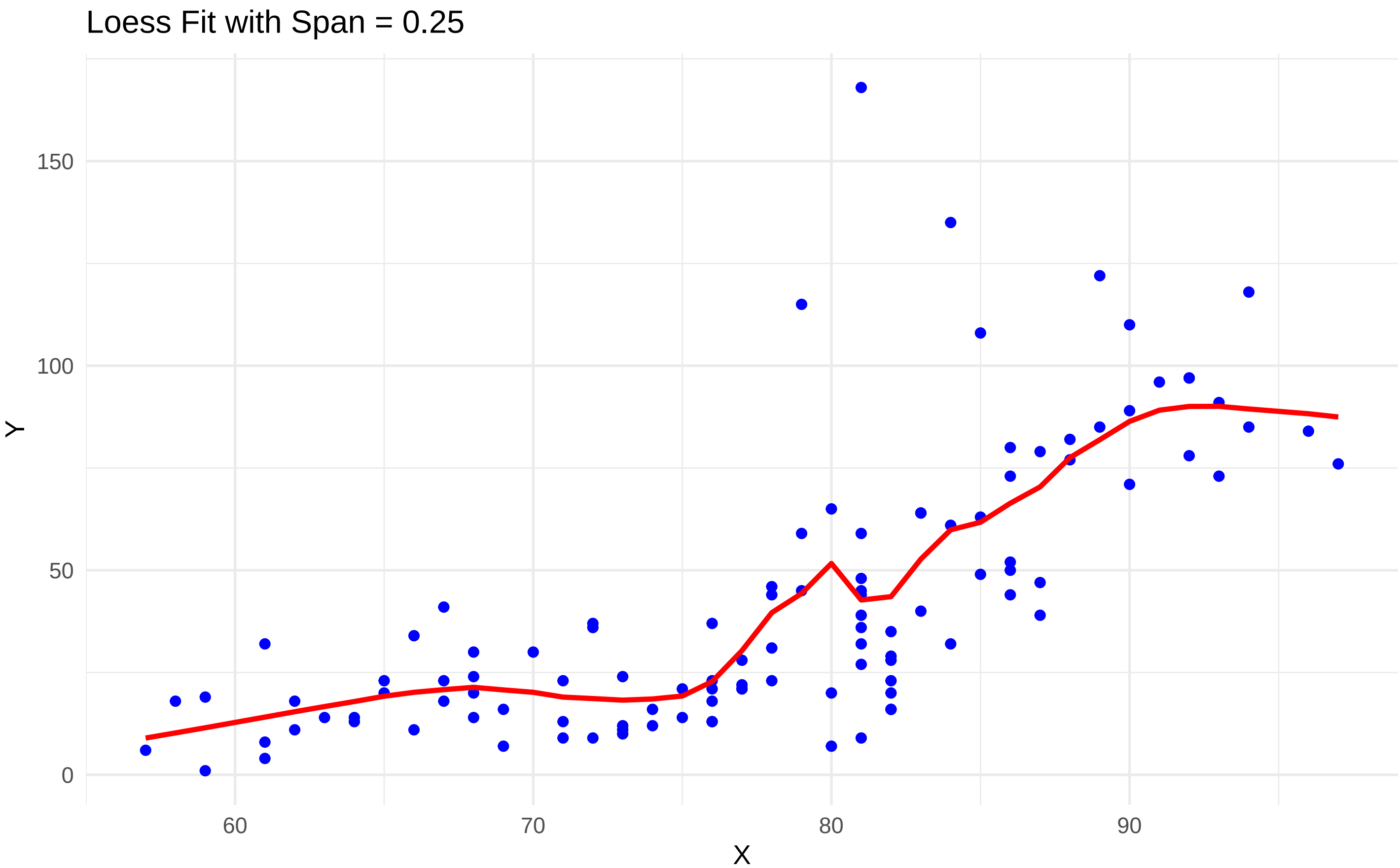
Here I compare the results using the built in loess() function. Analyzing the outputs we got from myloess() degree = 1, I can see that the MSE is different but follows a smiliar upward trend of the MSE. However, in the built in loess() function, I can see each increase in span increases the MSE which isn't always the case in my custom myloess() function.

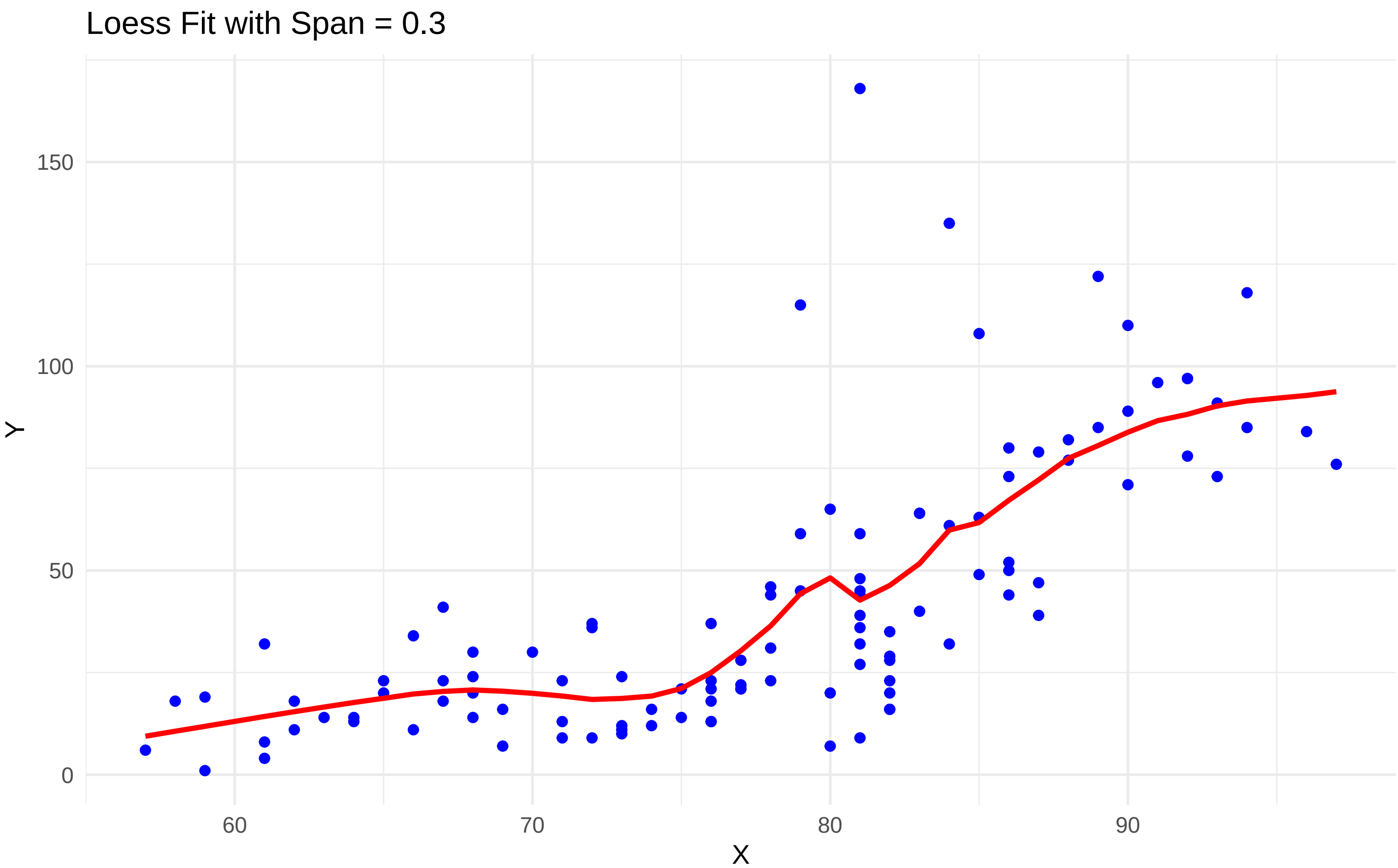
```
for (i in seq(.25, .75, by = 0.05)) {
  output = loess(y ~ x, span = i, degree = 1, show.plot = FALSE)

  MSE <- mean(output$residuals^2)

  #Display the MSE
  cat("The MSE of loess model span ",i," is:", MSE, "\n")
  }
```

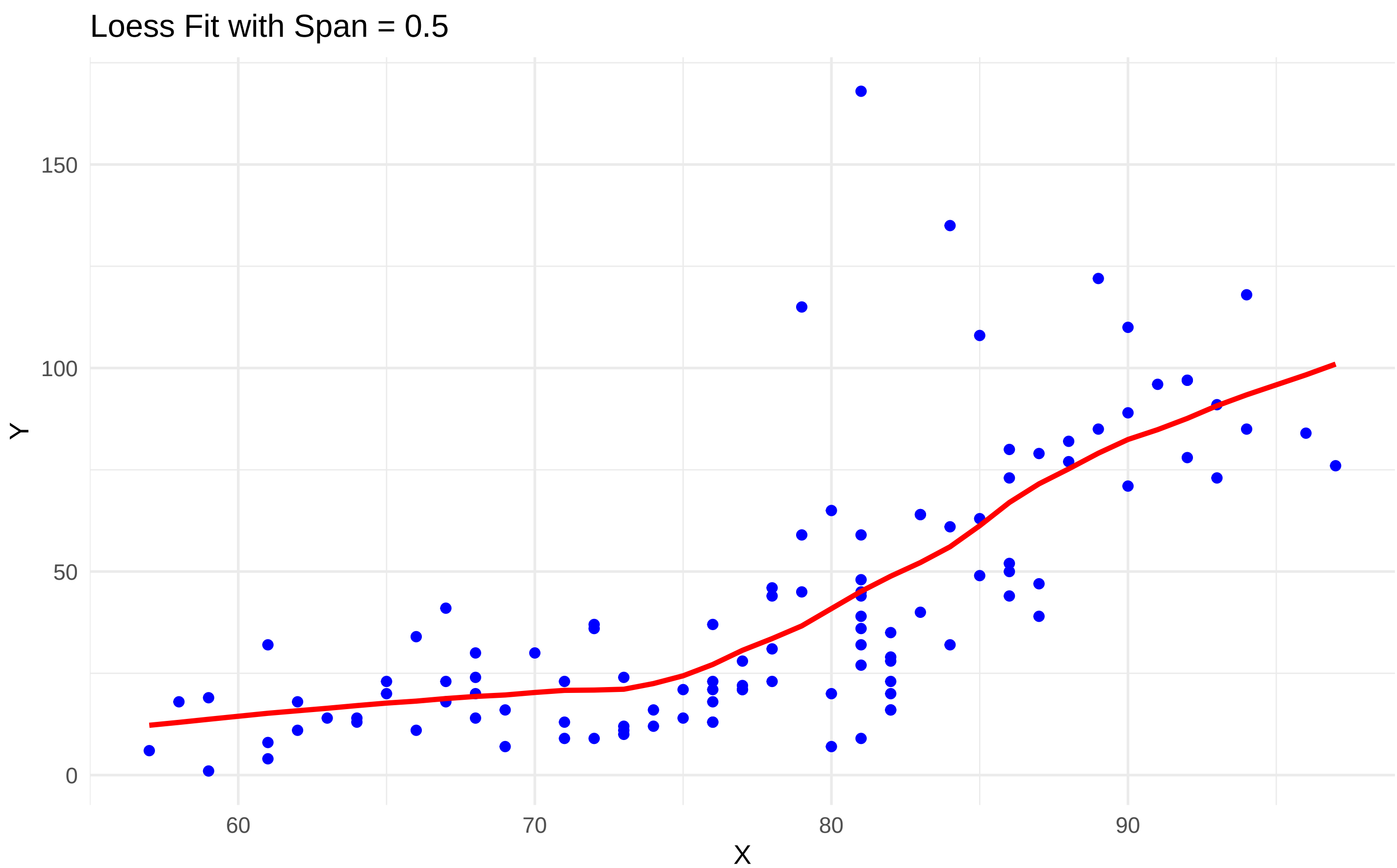
```
The MSE of loess model span  0.25  is: 442.1623
The MSE of loess model span  0.3  is: 452.9852
The MSE of loess model span  0.35  is: 458.938
The MSE of loess model span  0.4  is: 469.5251
The MSE of loess model span  0.45  is: 474.7817
The MSE of loess model span  0.5  is: 479.1056
The MSE of loess model span  0.55  is: 479.8493
The MSE of loess model span  0.6  is: 482.2303
The MSE of loess model span  0.65  is: 483.2988
The MSE of loess model span  0.7  is: 484.3554
The MSE of loess model span  0.75  is: 487.6304
```

While .25, .3, .35 are the span values leading to lowest MSE, I will still plot .25, .3, .5 span values corresponding to the myloess() plot for a better visual comparison.

```
span_values <- c(0.25, 0.3, 0.5) #Designate specific span values

df <- data.frame(x = x, y = y) #Use a dataframe to store x and y values

for (i in span_values) {
  loess_fit <- loess(y ~ x, span = i, degree = 1)  # Fit the loess model
  fitted_values <- predict(loess_fit)  #Make our predictions

  # Create a plot for each span value
  plot <- ggplot(df, aes(x = x, y = y)) +
    geom_point(color = "blue") +  # Original points
    geom_line(aes(y = fitted_values), color = "red", linewidth = 1) +
    labs(title = paste("Loess Fit with Span =", i), x = "X", y = "Y") +
    theme_minimal()

  print(plot)
}
```


Loess Fit with Span = 0.25
Y
X
0
50
100
150
60
70
80
90

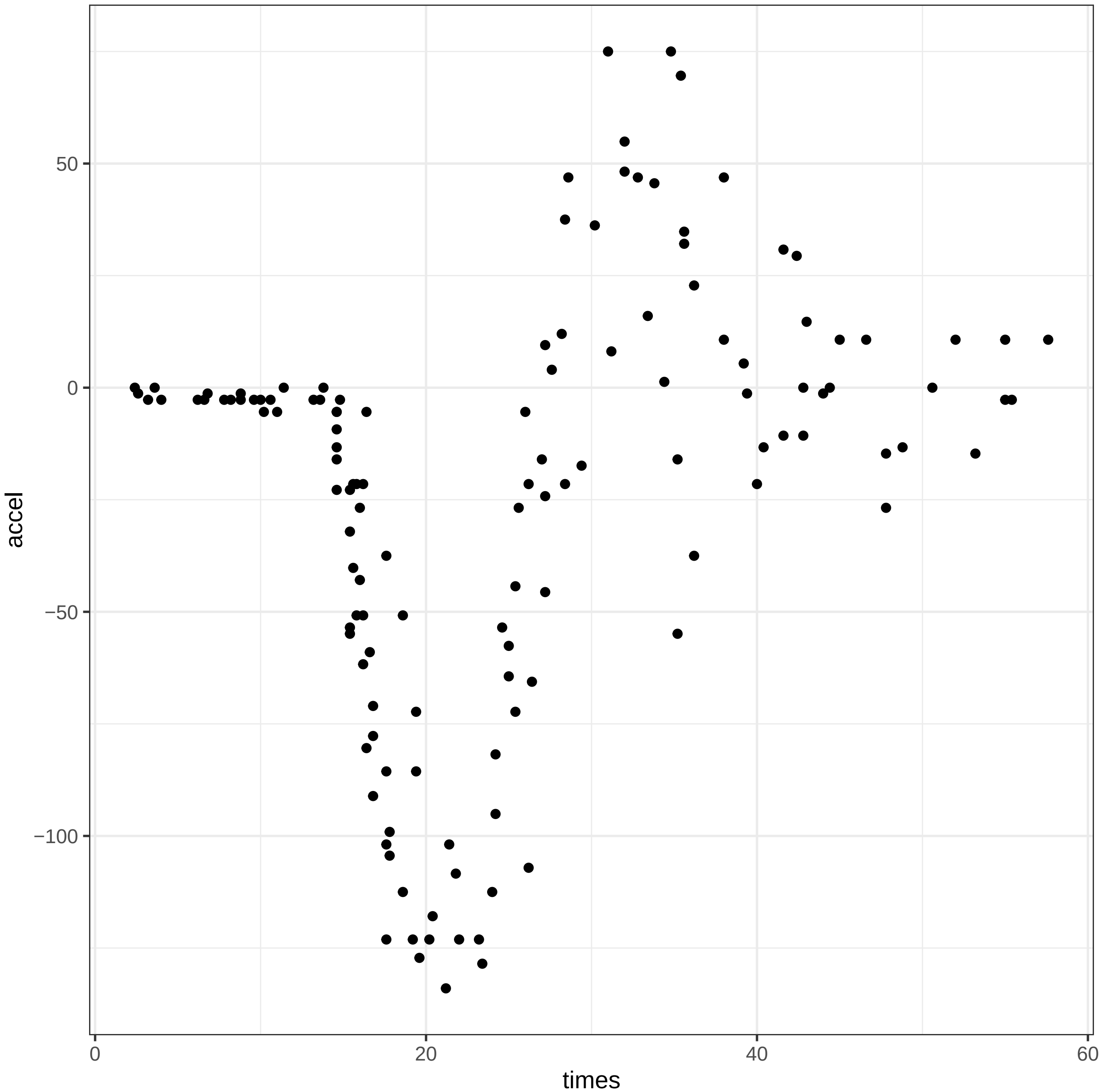

Loess Fit with Span = 0.3
Y
X
0
50
100
150
60
70
80
90



Analyzing the plots the built in loess function, when dealing with a smaller span, has a lot more variation, similar to that in myloess() function of degree 2. This is likely what's causing the lower MSE in the built in loess() function, as it's overfitting to the data. However as the span increases we can quickly see a smoothing of the fitted line as it creates a more generalized fit, which results in a similar plot as myloess() when the span = 0.5

Problem 2

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

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



1.)

```
#Extract the x and the y values
x = mcycle$times
y = mcycle$accel
```

Fit myloess() with the mcycle dataset using degree 1 with spans from .25 - .75

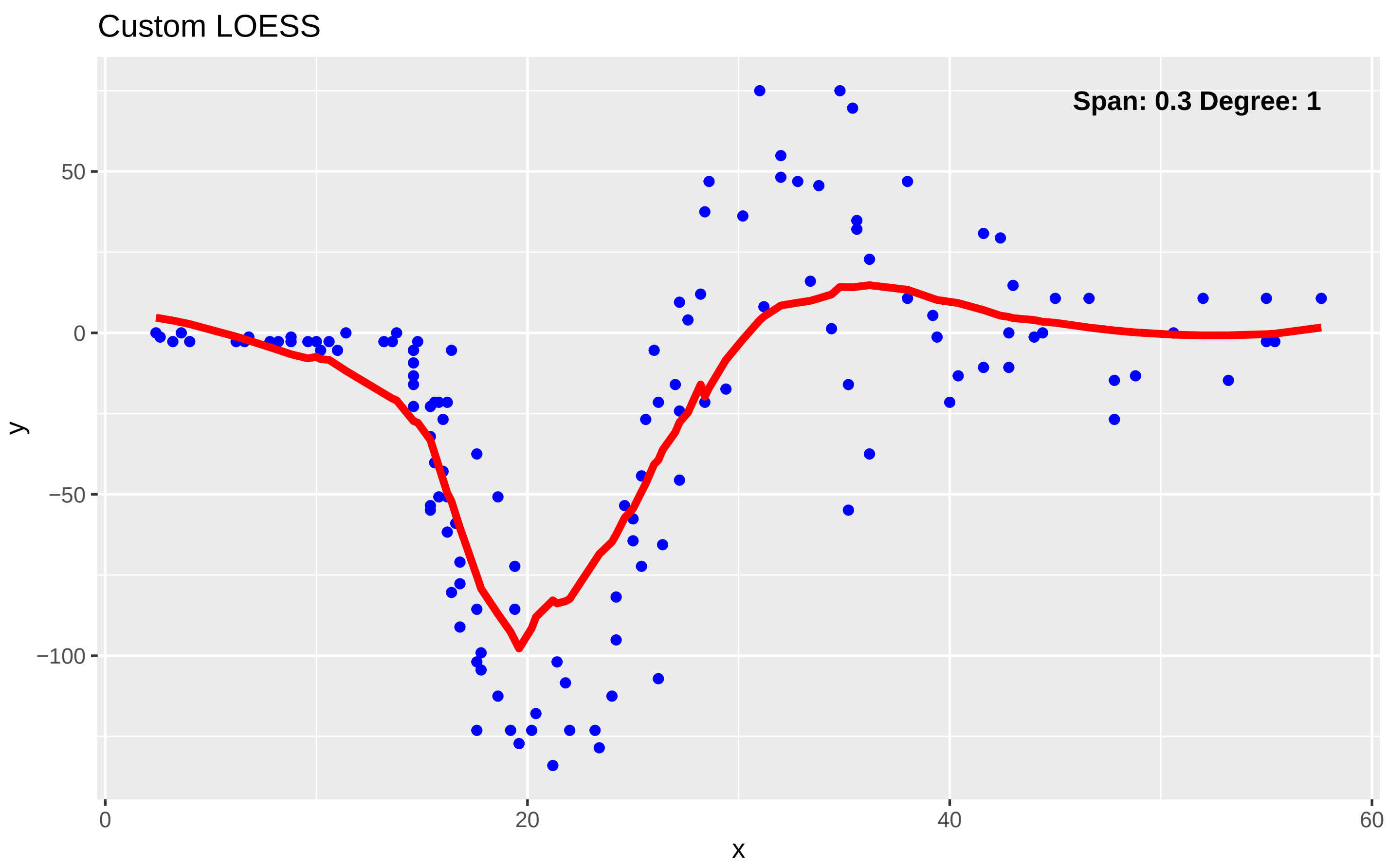
```
#Degree 1 LOESS span: 0.25 to 0.75
for (i in seq(.25, .75, by = 0.05)) {
  output = myloess(x, y, span = i, degree = 1, show.plot = FALSE)
  model_rss <- sum(output$SSE)
  MSE <- model_rss/n
  cat("The MSE of myloess model degree 1, span ",i," is:", MSE, "\n")
}
```

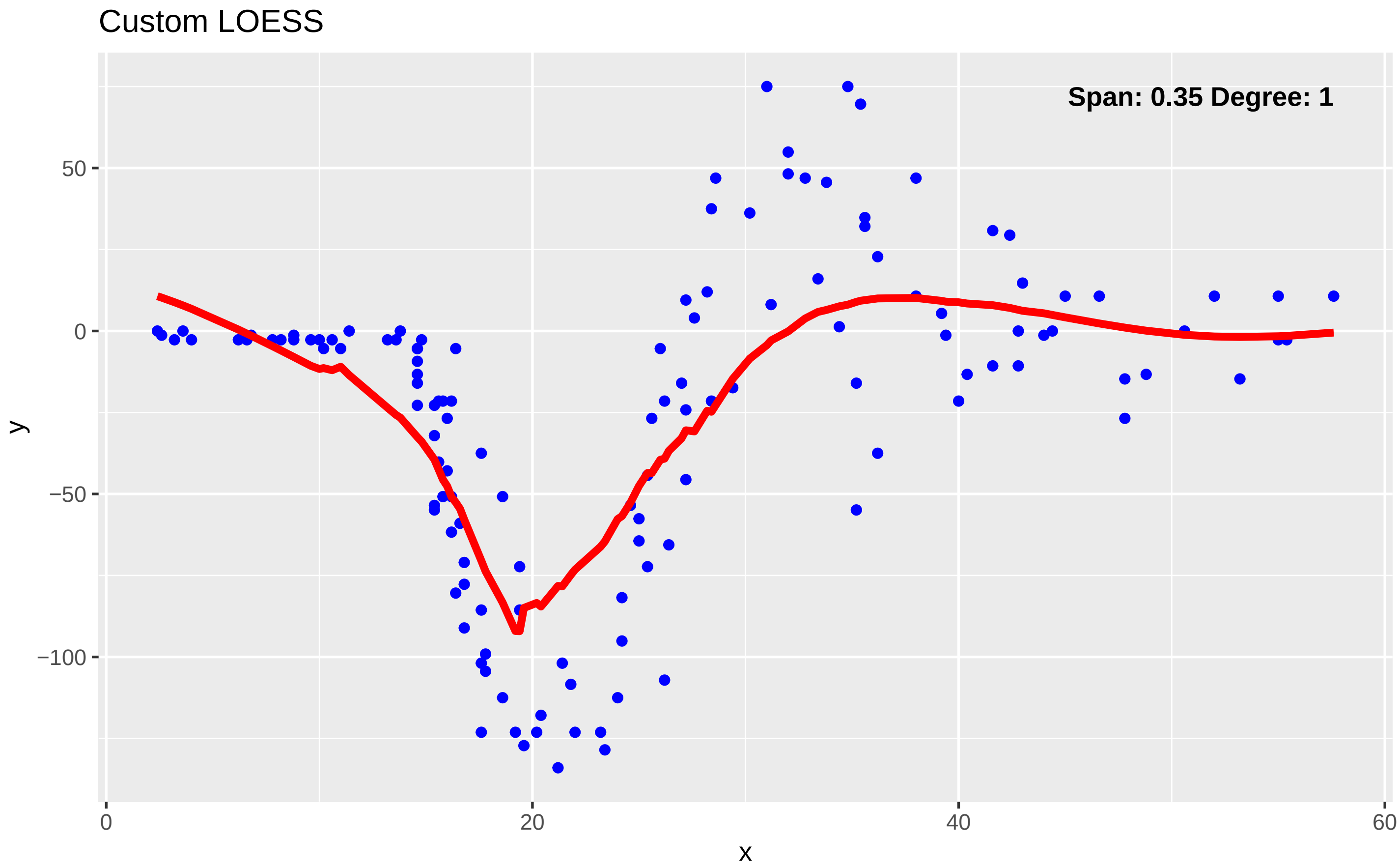
```
The MSE of myloess model degree 1, span  0.25  is: 730.9171
The MSE of myloess model degree 1, span  0.3  is: 836.3168
The MSE of myloess model degree 1, span  0.35  is: 998.0003
The MSE of myloess model degree 1, span  0.4  is: 1189.189
The MSE of myloess model degree 1, span  0.45  is: 1366.923
The MSE of myloess model degree 1, span  0.5  is: 1558.911
The MSE of myloess model degree 1, span  0.55  is: 1702.483
The MSE of myloess model degree 1, span  0.6  is: 1816.304
The MSE of myloess model degree 1, span  0.65  is: 1949.505
The MSE of myloess model degree 1, span  0.7  is: 2097.255
The MSE of myloess model degree 1, span  0.75  is: 2250.611
```

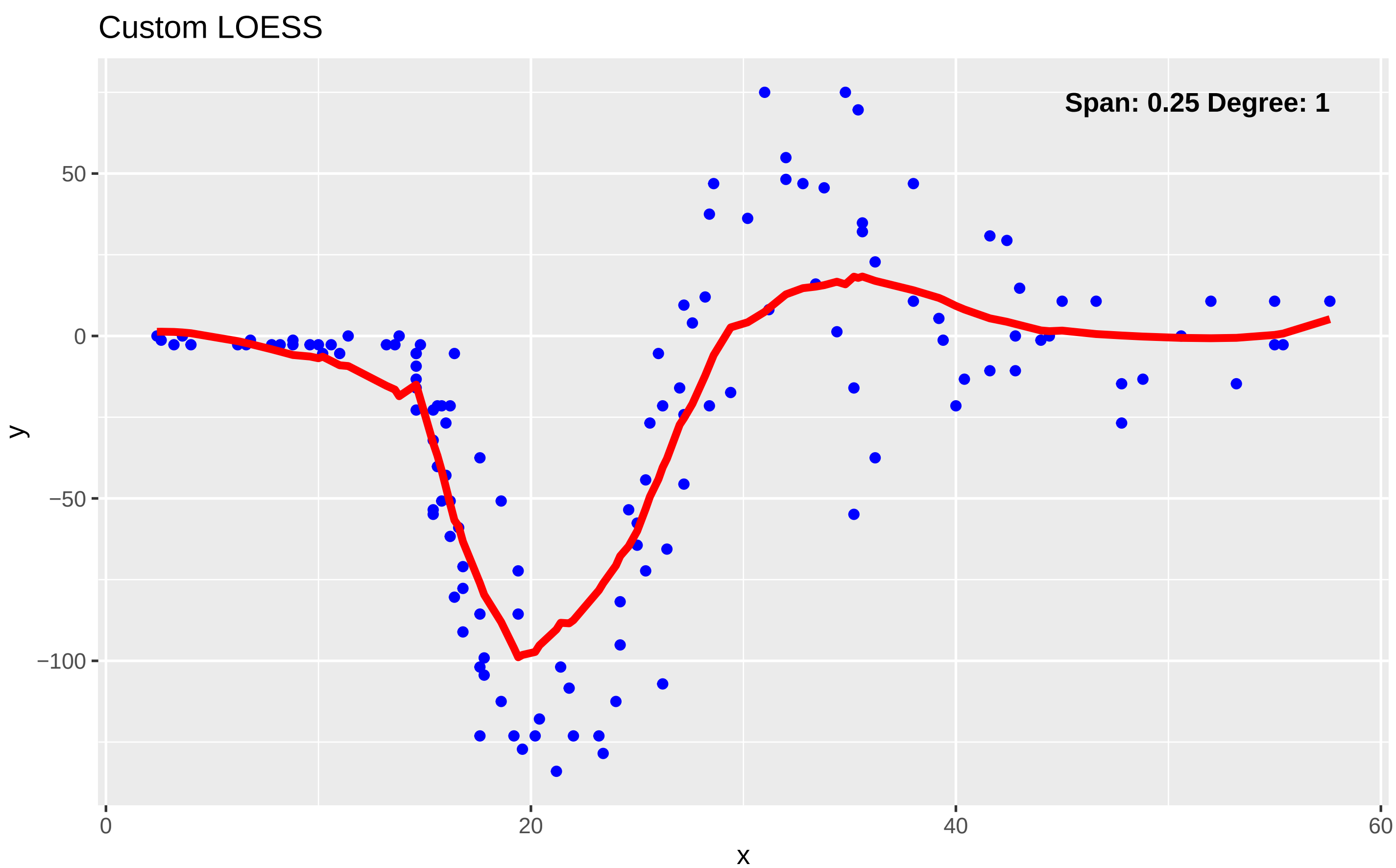
Looking at the MSE, span 0.25, 0.3, 0.35 has the lowest MSE thus I deem these 3 fits the best.

Let's plot these fits to see what they look like

```
best_spans_d1 <- c(0.25, 0.3, 0.35)
for (span in best_spans_d2){
  output = myloess(x, y, span = span, degree = 1, show.plot = TRUE)
}
```







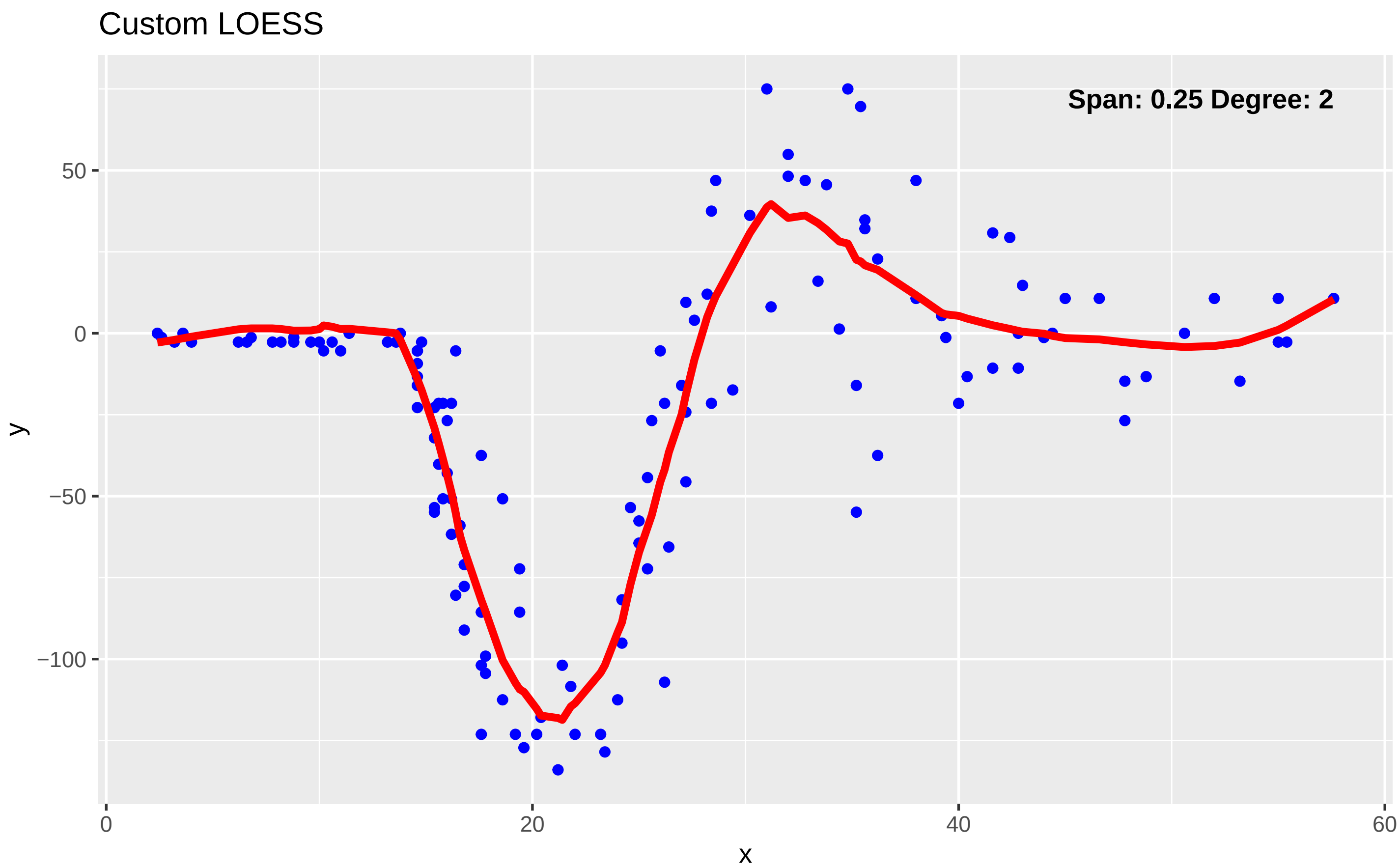
As we can see, all three spans in the plot does not capture enough curvature in the data. Let's try using degree = 2 to perhaps get better results.

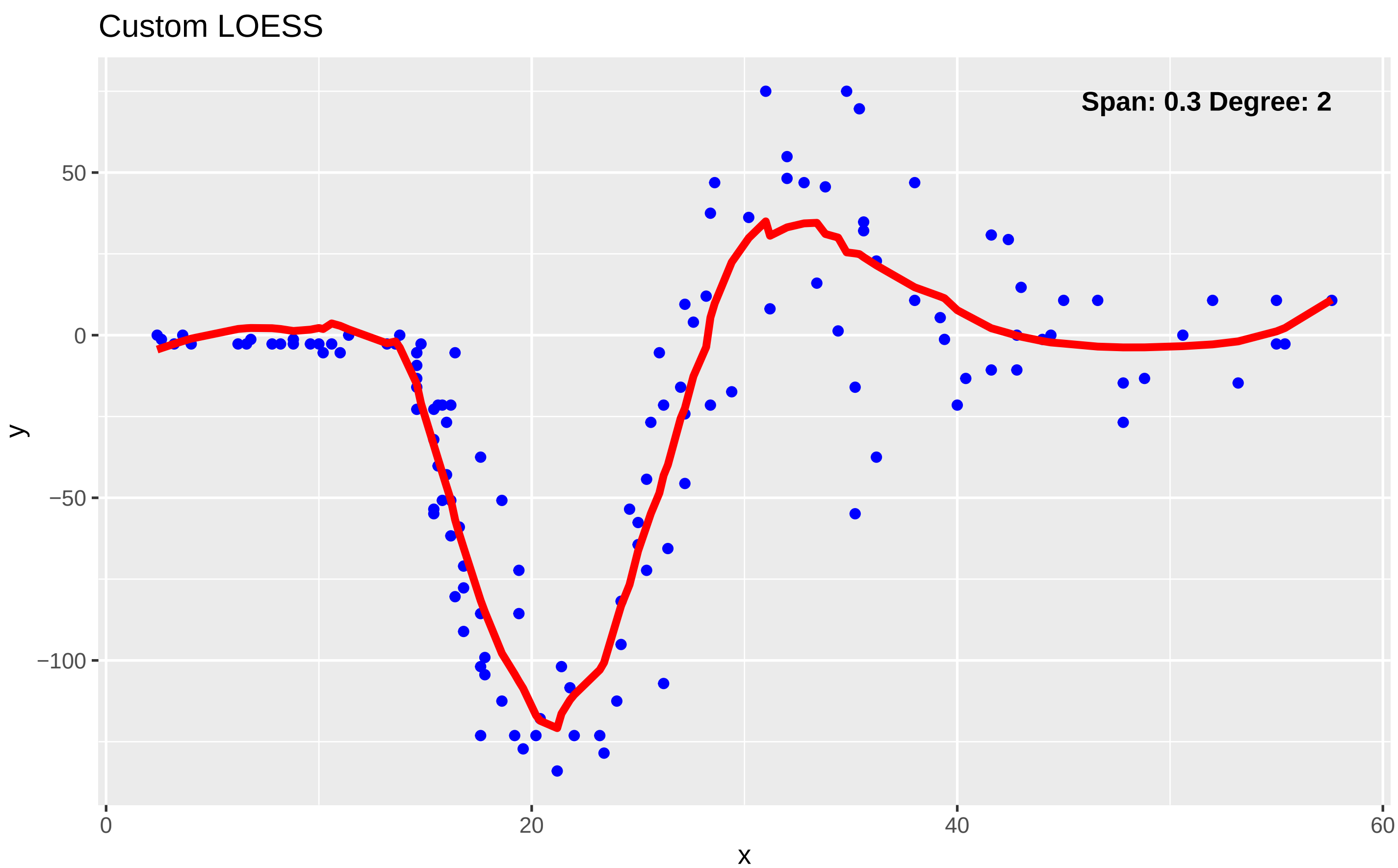
```r
#Degree 2 LOESS span: 0.25 to 0.75
for (i in seq(.25, .75, by = 0.05)) {
  output = myloess(x, y, span = i, degree = 2, show.plot = FALSE)
  MSE <- output$MSE
  cat("The MSE of myloess model degree 2, span ",i," is:", MSE, "\n")
}
```

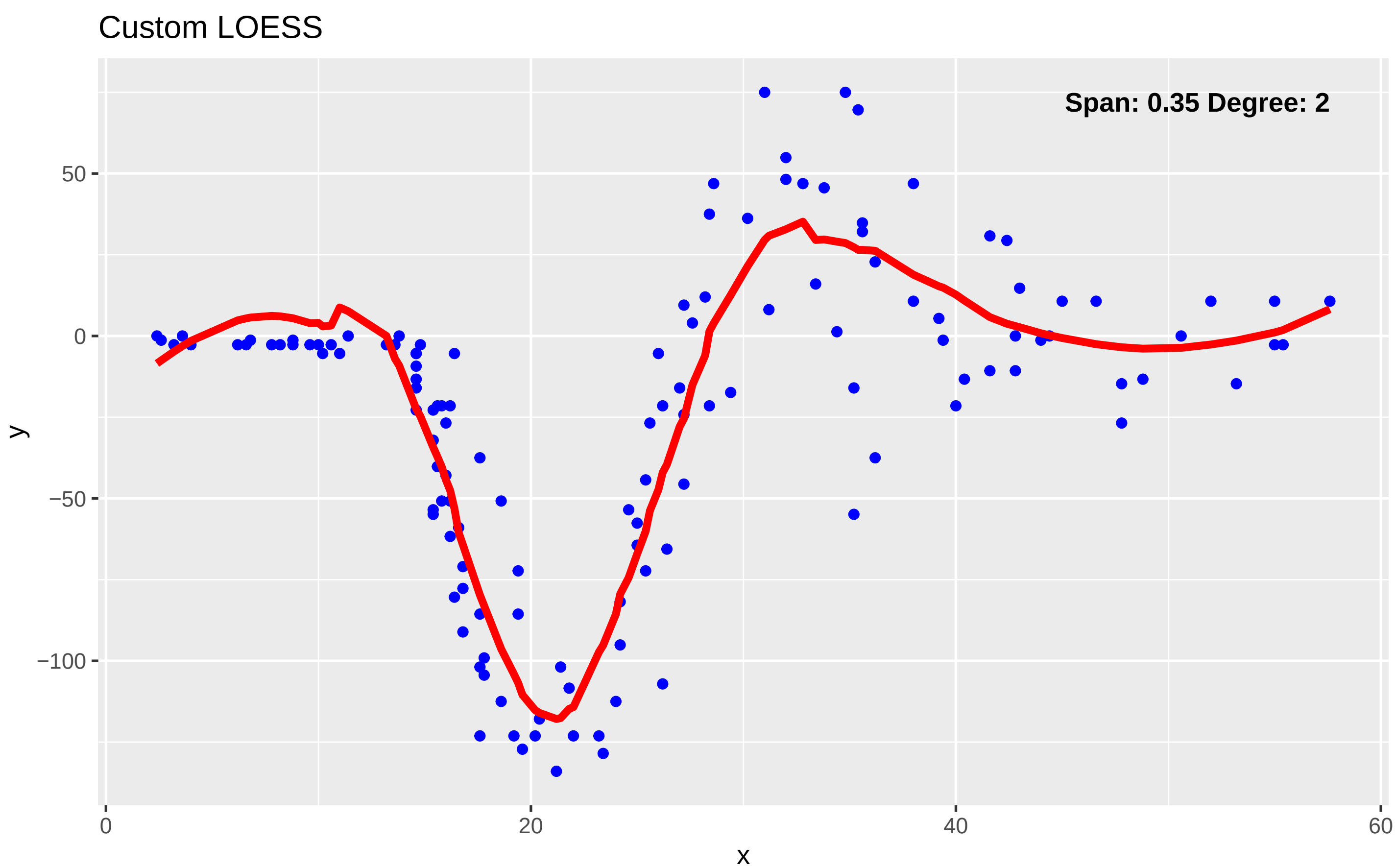
```
The MSE of myloess model degree 2, span  0.25  is: 458.4228
The MSE of myloess model degree 2, span  0.3  is: 471.6152
The MSE of myloess model degree 2, span  0.35  is: 497.4158
The MSE of myloess model degree 2, span  0.4  is: 551.0853
The MSE of myloess model degree 2, span  0.45  is: 602.6103
The MSE of myloess model degree 2, span  0.5  is: 682.5983
The MSE of myloess model degree 2, span  0.55  is: 793.1534
The MSE of myloess model degree 2, span  0.6  is: 913.4544
The MSE of myloess model degree 2, span  0.65  is: 1081.515
The MSE of myloess model degree 2, span  0.7  is: 1250.041
The MSE of myloess model degree 2, span  0.75  is: 1335.439
```

Our MSE has dropped drastically, again the best fits being 0.25, 0.3, 0.35 Let's plot these 3 best fits to compare with our plots from degree = 2

```r
best_spans_d2 <- c(0.25, 0.3, 0.35)
for (span in best_spans_d2){
  output = myloess(x, y, span = span, degree = 2, show.plot = TRUE)
}
```







We can see we get a much better fit with much more curvature in the regression line. This is due to using a quadratic regression thus introduces a more flexible shape that captures the curvature in the data.

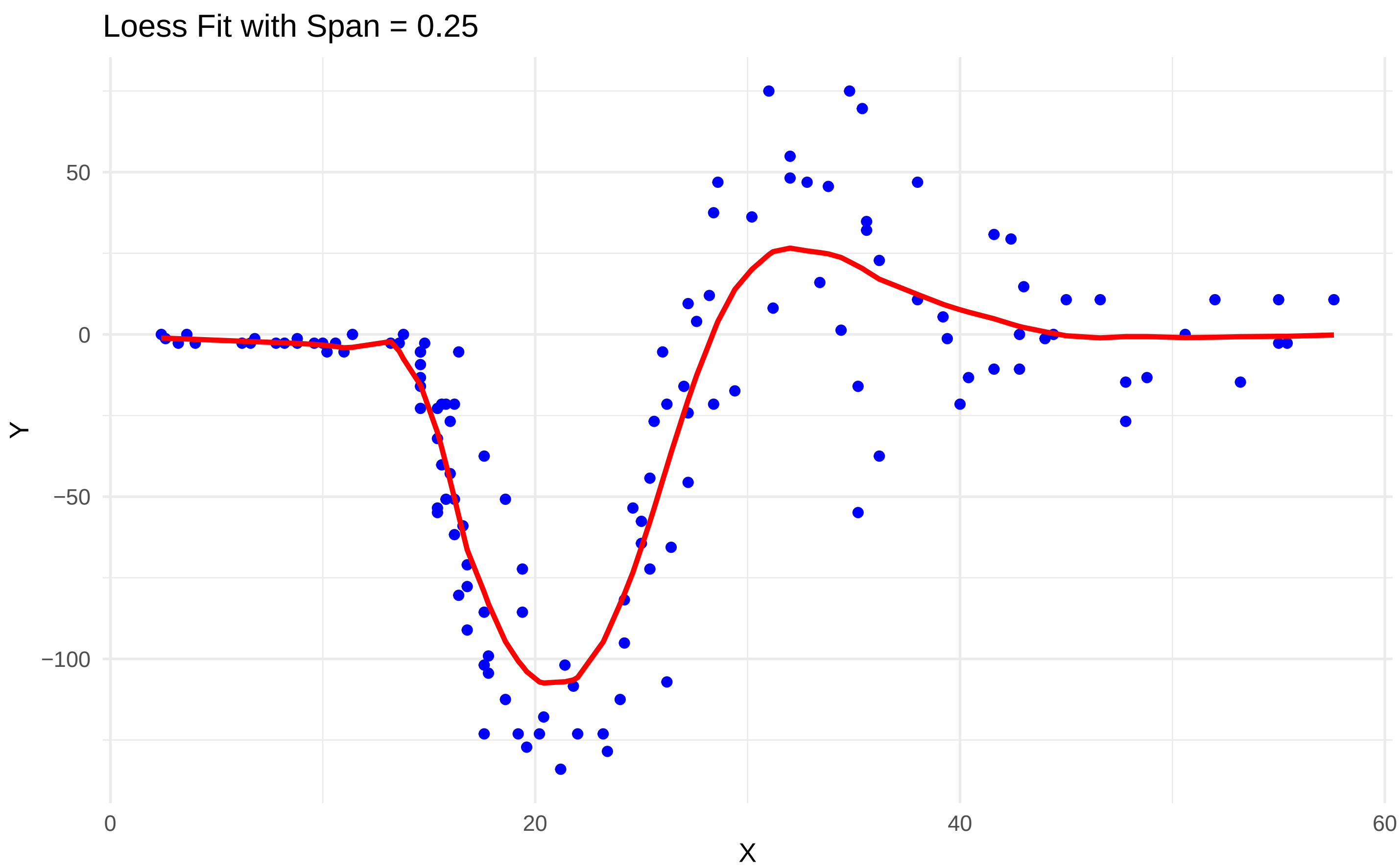
2.)

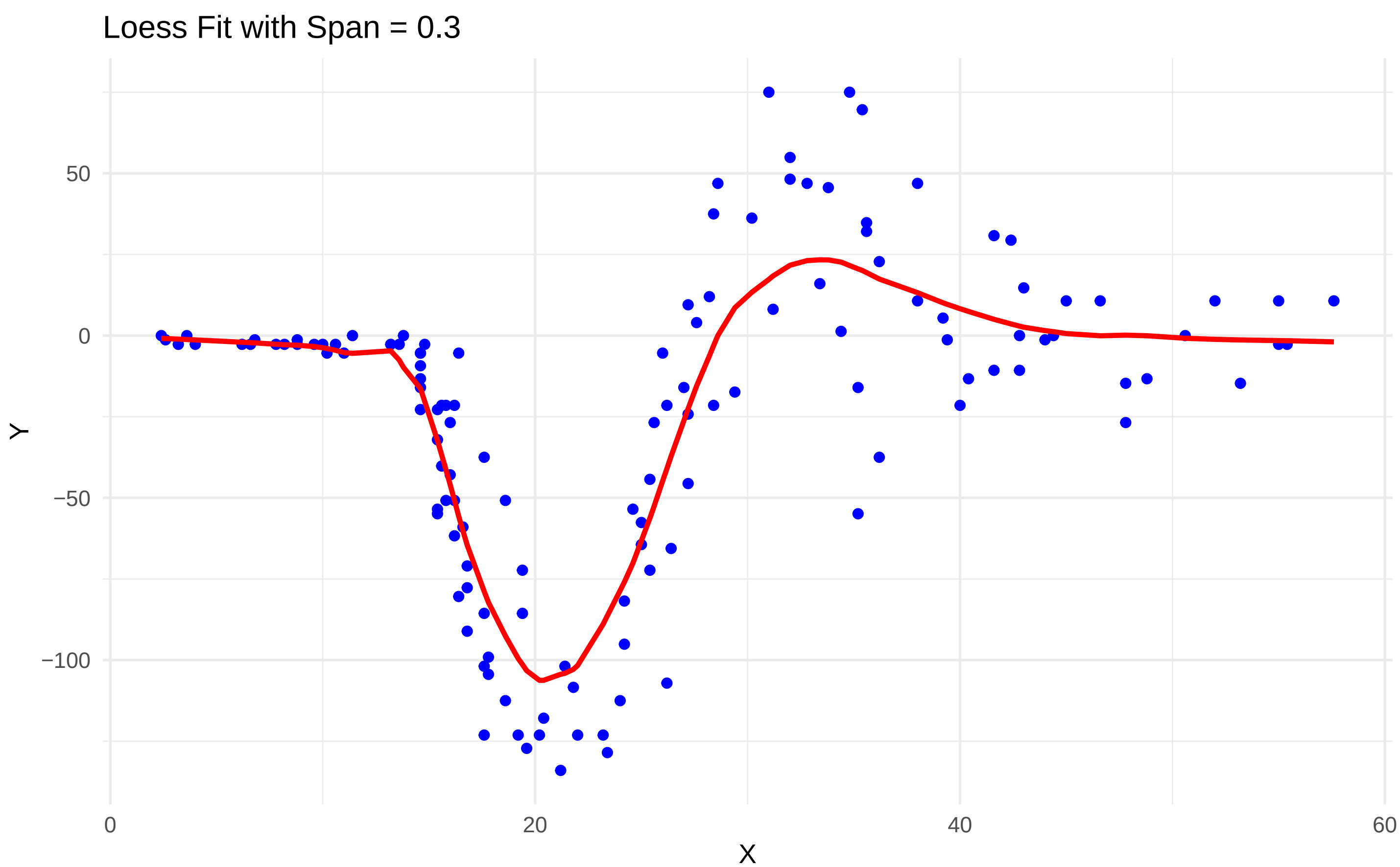
```r
#Using the built in loess function
for (i in seq(.25, .75, by = 0.05)) {
  output = loess(y ~ x, span = i, degree = 1, show.plot = FALSE)

  MSE <- mean(output$residuals^2)

  # Calculate the MSE

  cat("The MSE of loess model span ",i," is:", MSE, "\n")
  }
```

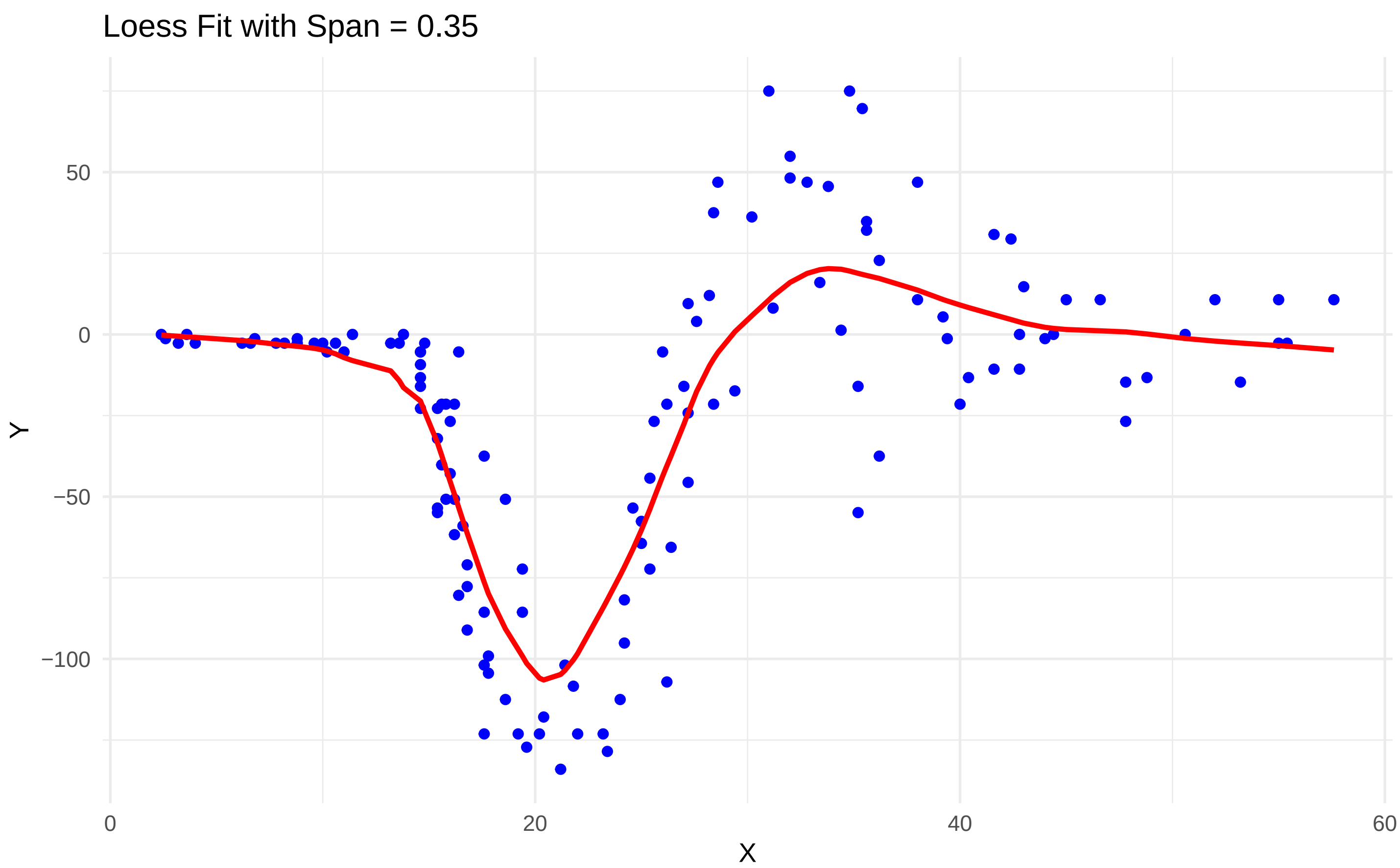
```
The MSE of loess model span  0.25  is: 491.0984
The MSE of loess model span  0.3  is: 518.9617
The MSE of loess model span  0.35  is: 561.0595
The MSE of loess model span  0.4  is: 623.5488
The MSE of loess model span  0.45  is: 682.3613
The MSE of loess model span  0.5  is: 784.8051
The MSE of loess model span  0.55  is: 869.9086
The MSE of loess model span  0.6  is: 990.8273
The MSE of loess model span  0.65  is: 1117.311
The MSE of loess model span  0.7  is: 1206.051
The MSE of loess model span  0.75  is: 1300.983
```

The built in loess() function still has consistently lower MSE thus providing a better fit especially considering degree = 1. Let's plot the 3 best fits (span = 0.25, 0.3, 0.35) of degree 1 for comparison and to get a visualization.

```r
span_values <- c(0.25, 0.3, 0.35) #Designate specific span values

df <- data.frame(x = x, y = y) #Use a dataframe to store x and y values

for (i in span_values) {
  loess_fit <- loess(y ~ x, span = i, degree = 1)  # Fit the loess model
  fitted_values <- predict(loess_fit)  #Make our predictions

  # Create a plot for each span value
  plot <- ggplot(df, aes(x = x, y = y)) +
    geom_point(color = "blue") +  # Original points
    geom_line(aes(y = fitted_values), color = "red", size = 1) +
    labs(title = paste("Loess Fit with Span =", i), x = "X", y = "Y") +
    theme_minimal()

  print(plot)
}
```


Loess Fit with Span = 0.25
50
0
−50
−100
Y
0
20
40
60
X


Loess Fit with Span = 0.3
50
0
−50
−100
Y
0
20
40
60
X



Next let's see how well the loess() function performs when degree = 2

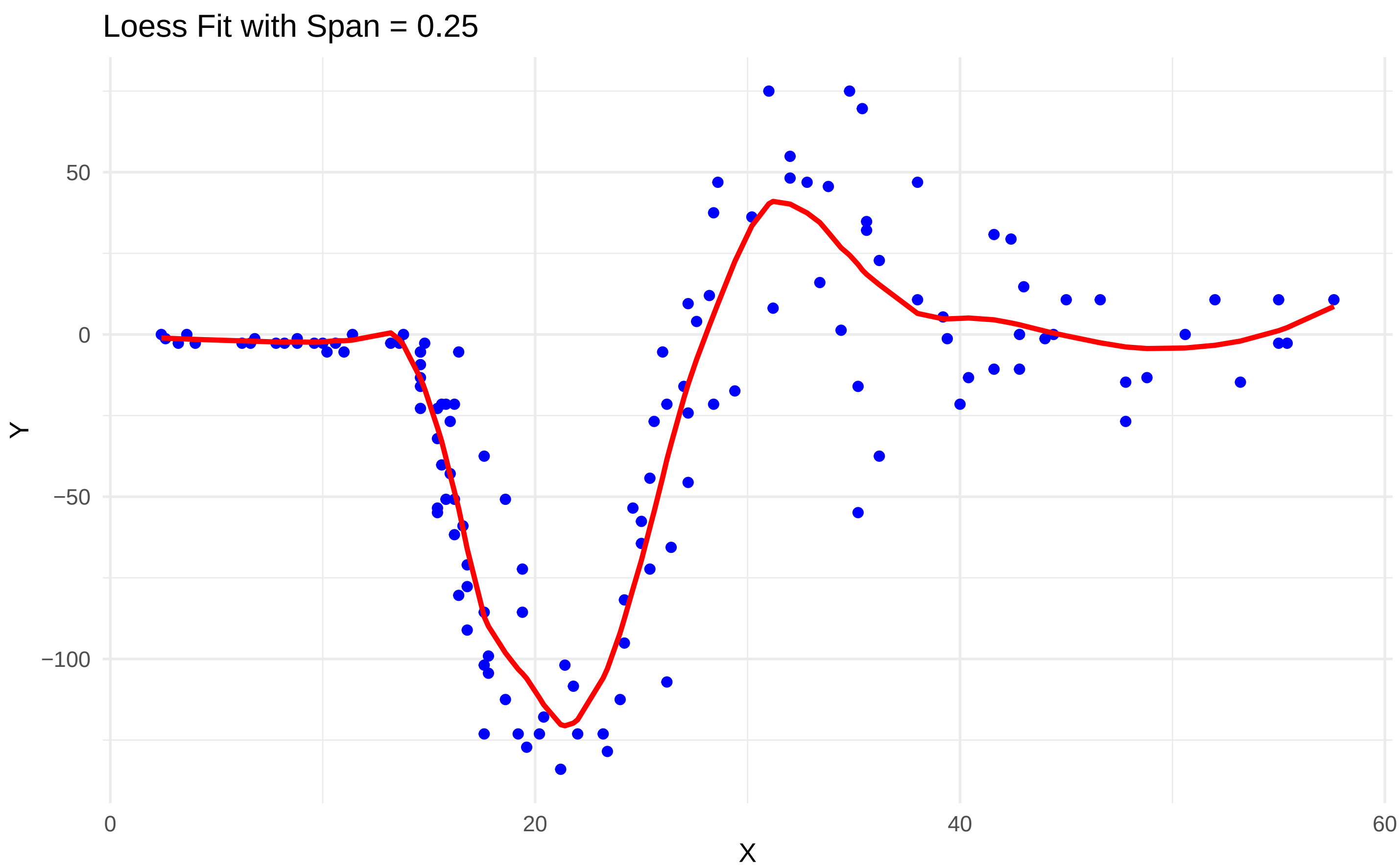
```
for (i in seq(.25, .75, by = 0.05)) {
  output = loess(y ~ x, span = i, degree = 2, show.plot = FALSE)

  MSE <- mean(output$residuals^2)

  # Calculate the MSE

  cat("The MSE of loess model span ",i," is:", MSE, "\n")
  }
```

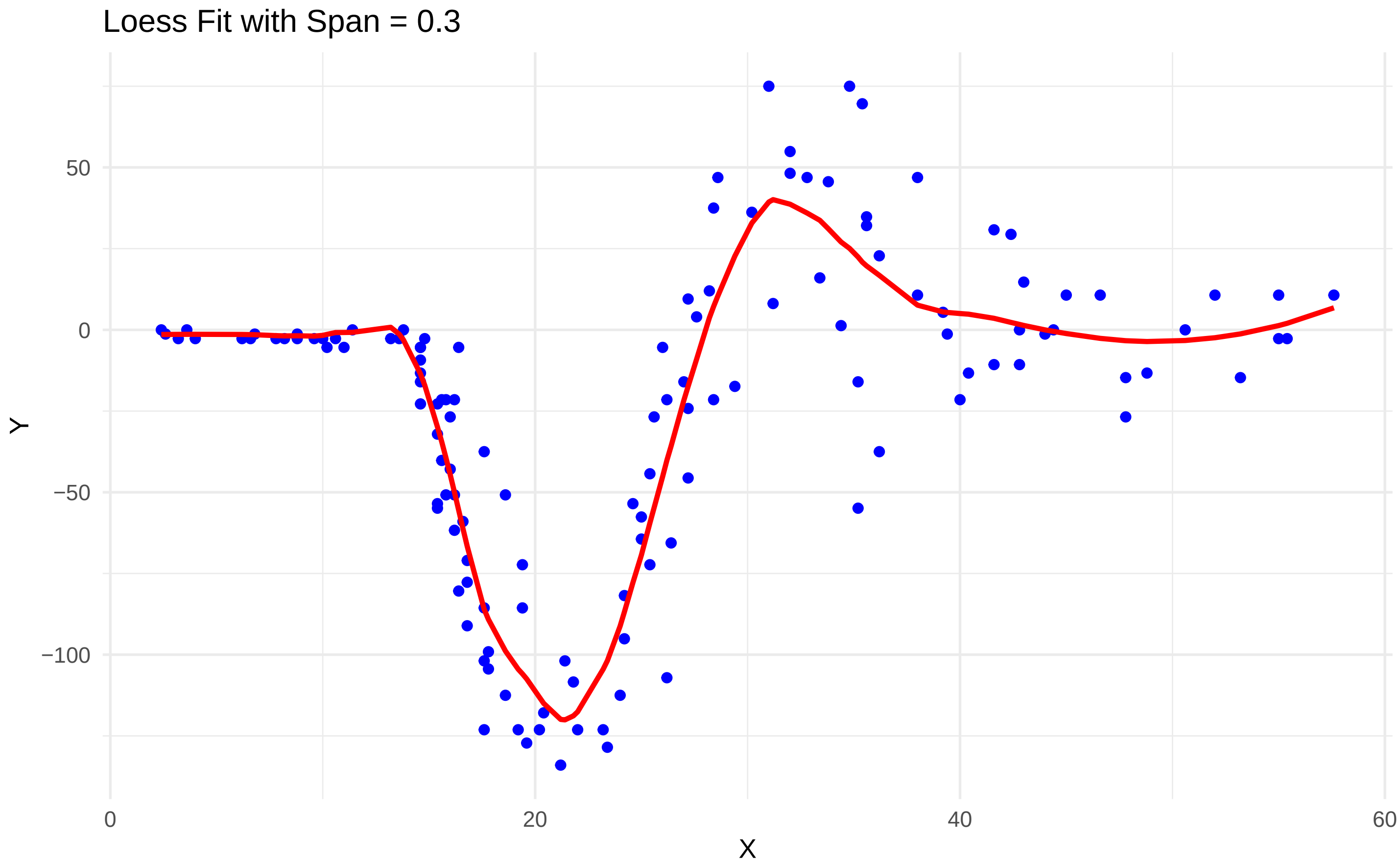
```
The MSE of loess model span  0.25  is: 454.2291
The MSE of loess model span  0.3  is: 456.1895
The MSE of loess model span  0.35  is: 465.3822
The MSE of loess model span  0.4  is: 482.7499
The MSE of loess model span  0.45  is: 501.8924
The MSE of loess model span  0.5  is: 535.2092
The MSE of loess model span  0.55  is: 570.1868
The MSE of loess model span  0.6  is: 616.2551
The MSE of loess model span  0.65  is: 727.6737
The MSE of loess model span  0.7  is: 840.7879
The MSE of loess model span  0.75  is: 979.736
```

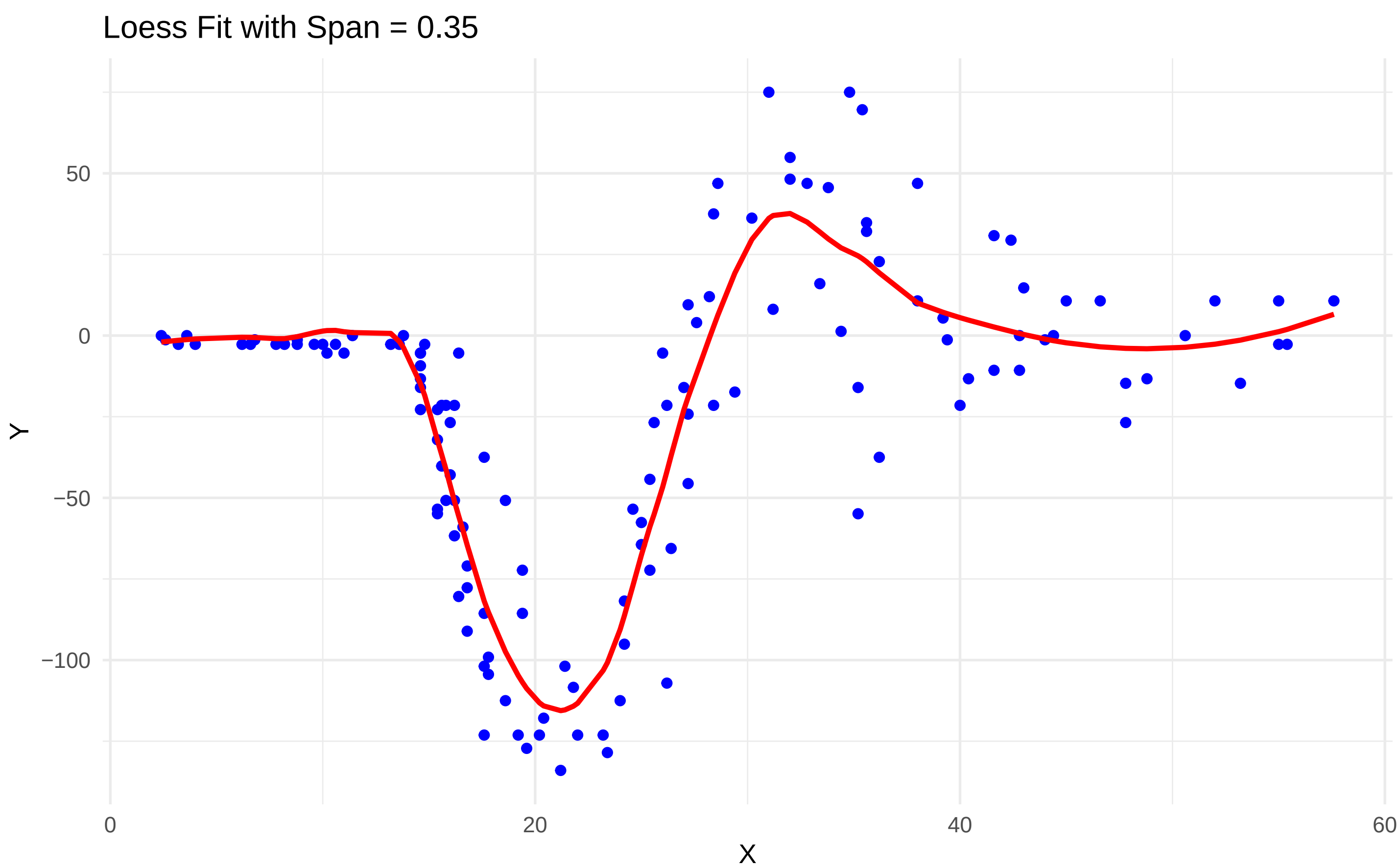
It provides a even better fit with a lower MSE. Let's look at the plots

```
span_values <- c(0.25, 0.3, 0.35) #Designate specific span values

df <- data.frame(x = x, y = y) #Use a dataframe to store x and y values
```

```
for (i in span_values) {
  loess_fit <- loess(y ~ x, span = i, degree = 2)  # Fit the loess model
  fitted_values <- predict(loess_fit)  #Make our predictions

  # Create a plot for each span value
  plot <- ggplot(df, aes(x = x, y = y)) +
    geom_point(color = "blue") +  # Original points
    geom_line(aes(y = fitted_values), color = "red", size = 1) +
    labs(title = paste("Loess Fit with Span =", i), x = "X", y = "Y") +
    theme_minimal()

  print(plot)
}
```







When degree = 2 the graph displays deeper troughs and higher peaks, likely due to the more flexible regression model thus allowing for more variation. However, I notice that values of X from 0 :approx 15, the loess model's line passes right through the data whereas my custom myloess() is offset and gets worse as the span increases. This is causing the higher MSE in myloess() predictions.

Part 2

Implementing a distance-weighted KNN

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


  n_test <- nrow(test)

  n_train <- nrow(train)


  # Vectorized computation of pairwise Euclidean distances

  distance_matrix <- as.matrix(dist(rbind(train, test)))

  train_test_distances <- distance_matrix[1:n_train, (n_train + 1):(n_train + n_test)]


  # Initialize yhat as a numeric vector

  yhat <- numeric(n_test)


  # Loop through each test point

  for (i in 1:n_test) {

    test_distances <- train_test_distances[, i]


    neighbors <- order(test_distances)[1:k]

    neighbor_distances <- test_distances[neighbors]

    neighbor_y <- y_train[neighbors]


    # weighted kNN

    if (weighted) {

      # Add .Machine$double.eps to prevent division by zero

      weights <- 1 / (neighbor_distances + .Machine$double.eps)

      weights[is.infinite(weights)] <- 0  # Handle zero distance by assigning 0 weights.
```

```
    if (is.factor(y_train)) {
      # Classification
      weighted_votes <- tapply(weights, neighbor_y, sum)
      yhat[i] <- names(which.max(weighted_votes))  # Class with the highest weighted votes
    } else {
      # Regression: Weighted average of neighbor responses
      yhat[i] <- sum(weights * neighbor_y) / sum(weights)
    }
  } else {
    # unweighted kNN
    if (is.factor(y_train)) {
      # Classification: Majority vote
      yhat[i] <- names(sort(table(neighbor_y), decreasing = TRUE))[1]
    } else {
      # Regression: Mean of neighbor responses
      yhat[i] <- mean(neighbor_y)
    }
  }
}

# Classification
if (is.factor(y_train)) {
  accuracy <- sum(yhat == y_test) / length(y_test)  # Calculate accuracy
  error_rate <- 1 - accuracy
  confusion_matrix <- table(yhat, y_test)  # Confusion matrix

  return(list(yhat = yhat, accuracy = accuracy, error_rate = error_rate, confusion_matrix = confusion_matrix
}
else {
  # Regression
```

```
    residuals <- y_test - yhat

    SSE <- sum(residuals^2)

    MSE <- SSE / length(y_test)

    RMSE <- sqrt(MSE)


    return(list(yhat = yhat, residuals = residuals, SSE = SSE, MSE = MSE, RMSE = RMSE, k = k, n_points = lengt

  }

}
```

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
```

```
# Set seed for reproducibility

set.seed(123)



# Split the data (70% training, 30% testing)

train_indices <- sample(1:nrow(Auto_new), 0.7 * nrow(Auto_new))

train_data <- Auto_new[train_indices, ]

test_data <- Auto_new[-train_indices, ]
```

```r
# Separate features (X) and target (Y)

train_x <- train_data[, -8]

test_x <- test_data[, -8]

train_y <- train_data$origin

test_y <- test_data$origin
```

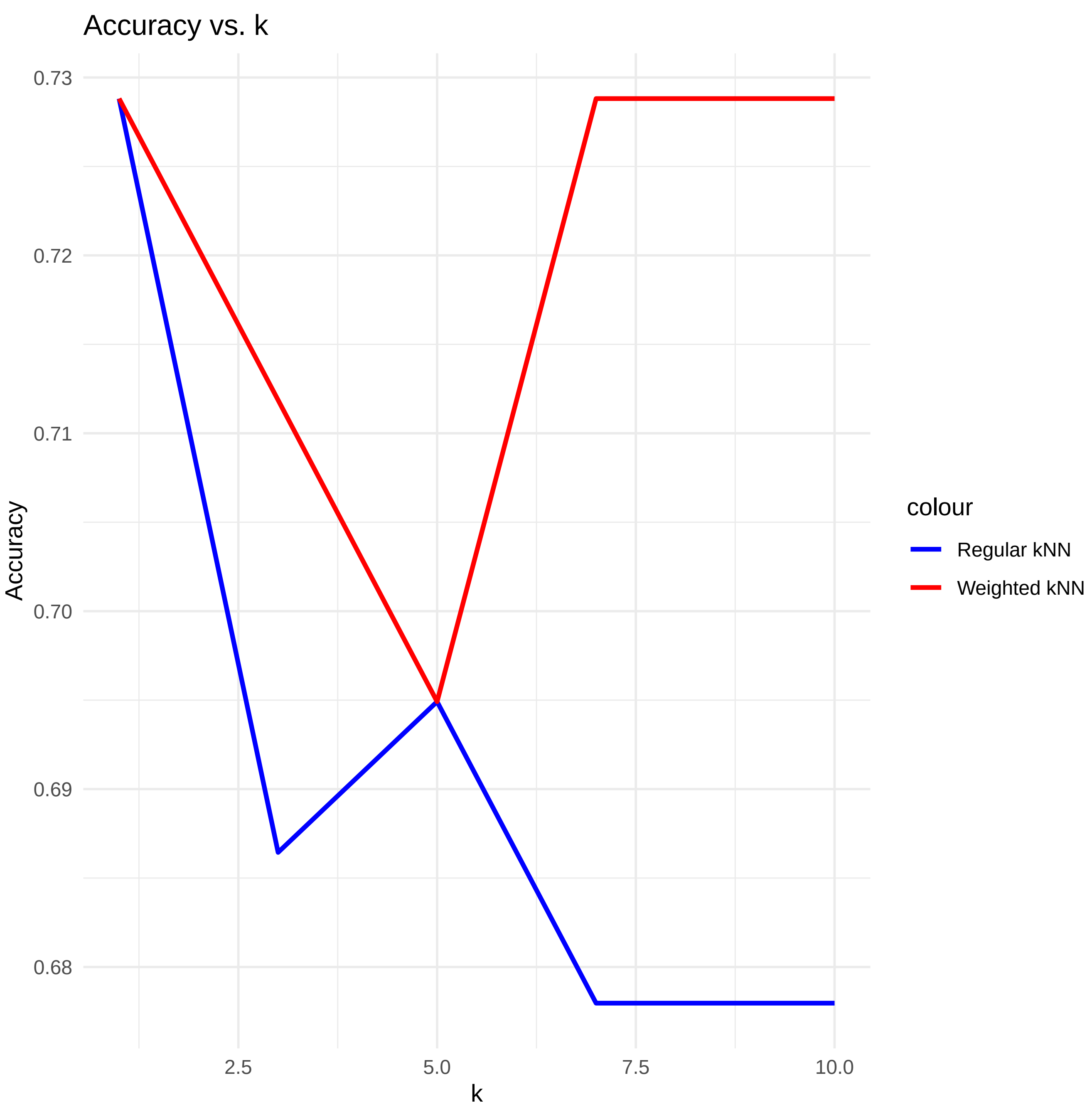
```r
library(knitr)

k_values <- c(1, 3, 5, 7, 10)

results_knn <- data.frame(k = k_values, accuracy_regular = numeric(length(k_values)), accuracy_weighted = nume


#Accuracy

for (k in k_values) {

  # Regular kNN

  result_regular <- mykNN(train_x, test_x, train_y, test_y, k = k, weighted = FALSE)

  results_knn[results_knn$k == k, "accuracy_regular"] <- result_regular$accuracy

  # Distance-weighted kNN

  result_weighted <- mykNN(train_x, test_x, train_y, test_y, k = k, weighted = TRUE)

  results_knn[results_knn$k == k, "accuracy_weighted"] <- result_weighted$accuracy

}


# Table

kable(results_knn, col.names = c("k", "Accuracy (Regular kNN)", "Accuracy (Weighted kNN)"), caption = "Accurac
```

Table 1: Accuracy for Regular and Weighted kNN

k	Accuracy (Regular kNN)	Accuracy (Weighted kNN)
1	0.7288136	0.7288136
3	0.6864407	0.7118644
5	0.6949153	0.6949153
7	0.6779661	0.7288136
10	0.6779661	0.7288136

```r
library(ggplot2)

# Plot accuracy vs k
```

```r
ggplot(results_knn, aes(x = k)) +
  geom_line(aes(y = accuracy_regular, color = "Regular kNN"), size = 1) +   # Line for regular kNN
  geom_line(aes(y = accuracy_weighted, color = "Weighted kNN"), size = 1) + # Line for weighted kNN
  labs(title = "Accuracy vs. k", x = "k", y = "Accuracy") +                # Labels
  scale_color_manual(values = c("Regular kNN" = "blue", "Weighted kNN" = "red")) + # Colors
  theme_minimal()
```




```r
#Confusion matrixes
best_k <- 1
```

```r
# Regular kNN with k = 1

best_regular_knn <- mykNN(train_x, test_x, train_y, test_y, k = best_k, weighted = FALSE)

cat("Confusion Matrix for Regular kNN with k =", best_k, ":\n")
```

```
Confusion Matrix for Regular kNN with k = 1 :
```

```r
print(best_regular_knn$confusion_matrix)
```

```
          y_test
yhat       USA European Japanese
  European   5        7        4
  Japanese   3        1       17
  USA       62       13        6
```

```r
cat("Accuracy for Regular kNN with k =", best_k, ":", best_regular_knn$accuracy, "\n")
```

```
Accuracy for Regular kNN with k = 1 : 0.7288136
```

```r
# Weighted kNN with k = 1

best_weighted_knn <- mykNN(train_x, test_x, train_y, test_y, k = best_k, weighted = TRUE)

cat("\nConfusion Matrix for Weighted kNN with k =", best_k, ":\n")
```

```

Confusion Matrix for Weighted kNN with k = 1 :
```

```r
print(best_weighted_knn$confusion_matrix)
```

```
          y_test
yhat       USA European Japanese
  European   5        7        4
  Japanese   3        1       17
  USA       62       13        6
```

```r
cat("Accuracy for Weighted kNN with k =", best_k, ":", best_weighted_knn$accuracy, "\n")
```

```
Accuracy for Weighted kNN with k = 1 : 0.7288136
```

```r
# Weighted knn for k = 7

best_k_weighted <- 7

best_weighted_knn <- mykNN(train_x, test_x, train_y, test_y, k = best_k_weighted, weighted = TRUE)



cat("\nConfusion Matrix for Weighted kNN with k =", best_k_weighted, ":\n")
```

```

Confusion Matrix for Weighted kNN with k = 7 :
```

```r
print(best_weighted_knn$confusion_matrix)
```

```
          y_test
yhat       USA European Japanese
  European   4        7        3
  Japanese   1        3       14
  USA       65       11       10
```

```
cat("Accuracy for Weighted kNN with k =", best_k_weighted, ":", best_weighted_knn$accuracy, "\n")
```

```
Accuracy for Weighted kNN with k = 7 : 0.7288136
```

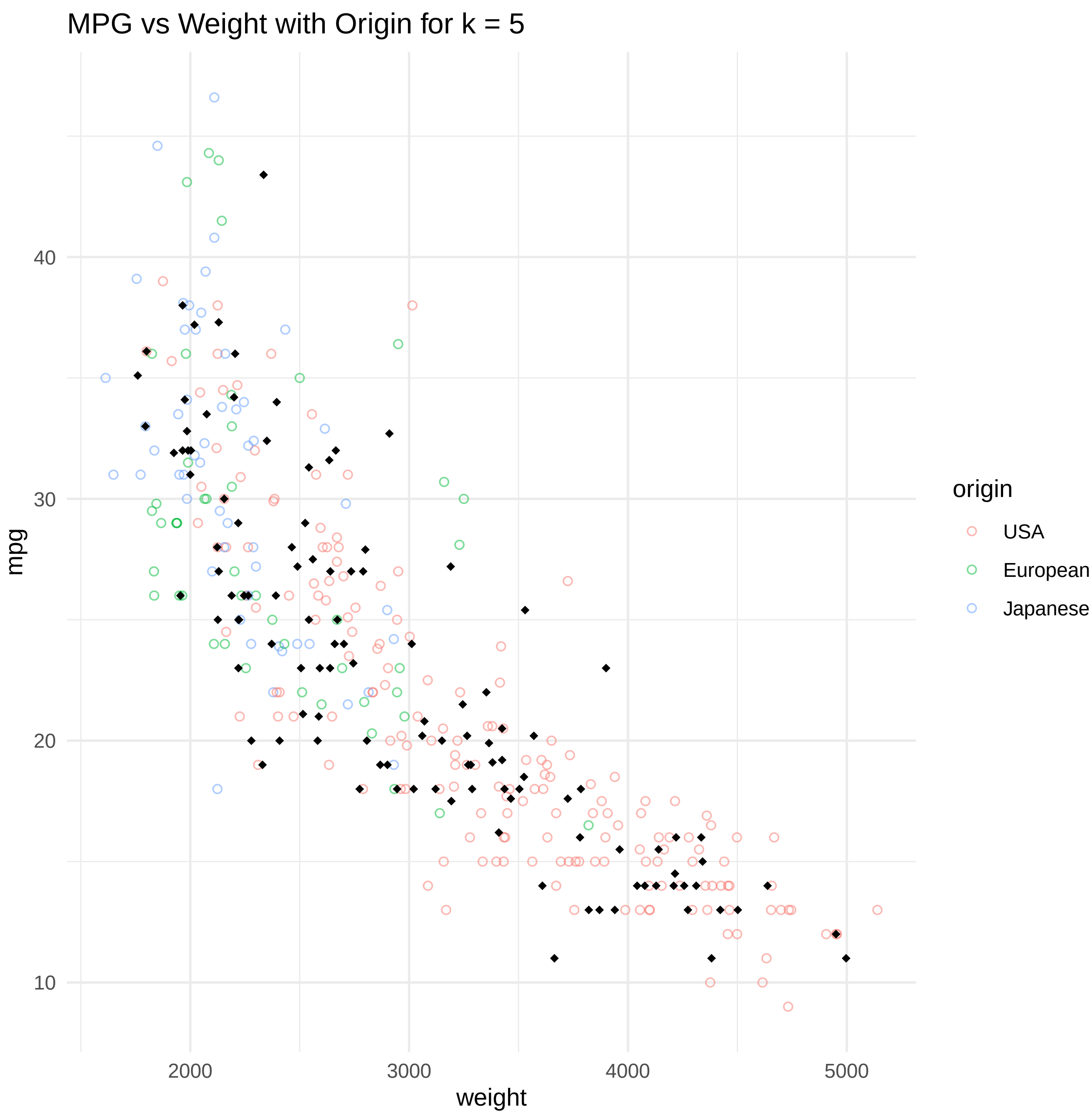
Observations:

The confusion matrix for k = 1 for regular kNN shows the classification performance on the test set, and the accuracy is 72.88%. Regular kNN performs well with smaller values of k, but its accuracy drops when more neighbors are taken into account. This shows that regular kNN is underfitting when more neighbors are considered, as it smooths out the classifications.

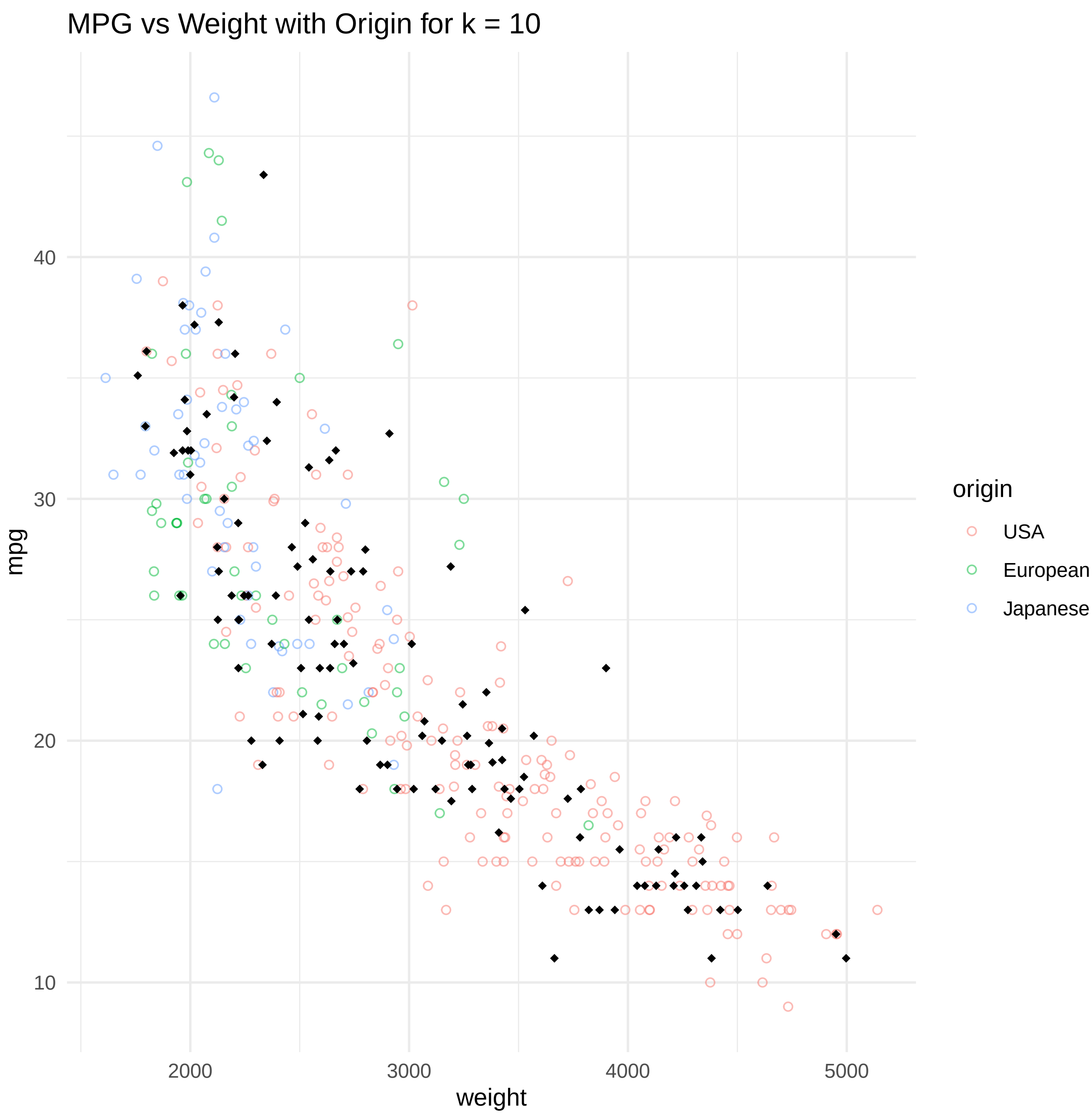
Weighted kNN (dnkNN), benefits from weighting the influence of neighbors by distance, allowing it to maintain the 72.88% accuracy even with more neighbors (k = 7). This suggests that distance-weighted kNN is a more robust approach.

Plots of mpg vs weight

```
# Plot for k = 5

ggplot(train_data, aes(x = weight, y = mpg, color = origin)) +

  geom_point(alpha = 0.5, shape = 21) +

  geom_point(data = test_data, aes(x = weight, y = mpg), shape = 18, color = "black") +

  ggtitle("MPG vs Weight with Origin for k = 5") +

  theme_minimal()
```




```r
# Plot for k = 10
ggplot(train_data, aes(x = weight, y = mpg, color = origin)) +
  geom_point(alpha = 0.5, shape = 21) +
  geom_point(data = test_data, aes(x = weight, y = mpg), shape = 18, color = "black") +
  ggtitle("MPG vs Weight with Origin for k = 10") +
  theme_minimal()
```



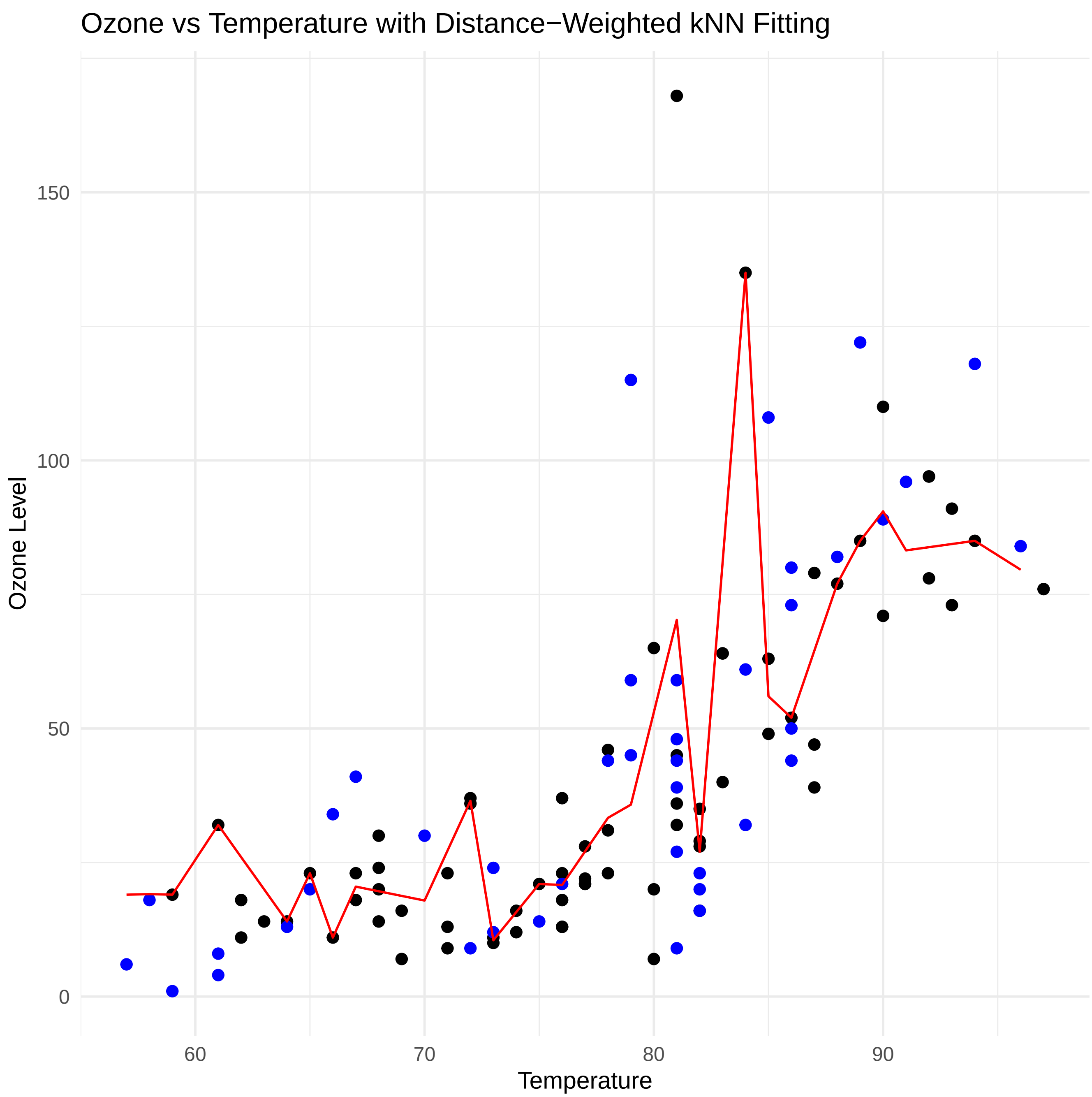
Problem 4

```r
# Set seed for reproducibility
set.seed(42)

# Split the data into training and testing data (70 obs for training, 41 for testing)
index <- sample(1:nrow(ozone), 70)
train_data <- ozone[index, ]
test_data <- ozone[-index, ]
```

Part a

```r
# Set variables appropriately
y_train <- train_data$ozone
y_test <- test_data$ozone
X_train <- train_data$temperature
X_test <- test_data$temperature

# k values
k_values <- c(1, 3, 5, 10, 20)

# Initialize DataFrame for results
results <- data.frame()

# Loop through k values to do dwkNN
for (k in k_values) {
  # Use the mykNN function to predict ozone levels
  predictions <- mykNN(train = as.matrix(X_train), test = as.matrix(X_test),
                       y_train = y_train, y_test = y_test, k = k, weighted = TRUE)

  # Calculate the MSE for the predictions
  MSE <- predictions$MSE

  # Append results to the DataFrame
  results <- rbind(results, data.frame(k = k, MSE = MSE))
}

# Plot training and testing data along with the fitted regression
ggplot() +
  geom_point(data = train_data, aes(x = temperature, y = ozone),
             color = "black", size = 2) +  # Training data (black points)
```

```
  geom_point(data = test_data, aes(x = temperature, y = ozone),
             color = "blue", size = 2) +    # Testing data (blue points)
  geom_line(data = data.frame(temperature = X_test, fitted_ozone = predictions$yhat),
            aes(x = temperature, y = fitted_ozone), color = "red") + # Fitted regression line
  ggtitle("Ozone vs Temperature with Distance-Weighted kNN Fitting") +
  theme_minimal() +
  labs(x = "Temperature", y = "Ozone Level")
```




```
# Display the results in a table using kable
```

```r
kable(results, col.names = c("k", "MSE"),

      caption = "MSE for Different k values in dwkNN (Ozone ~ Temp)")
```

Table 2: MSE for Different k values in dwkNN (Ozone ~ Temp)

k	MSE
1	993.5366
3	863.3389
5	1058.4382
10	1038.5638
20	1019.6554

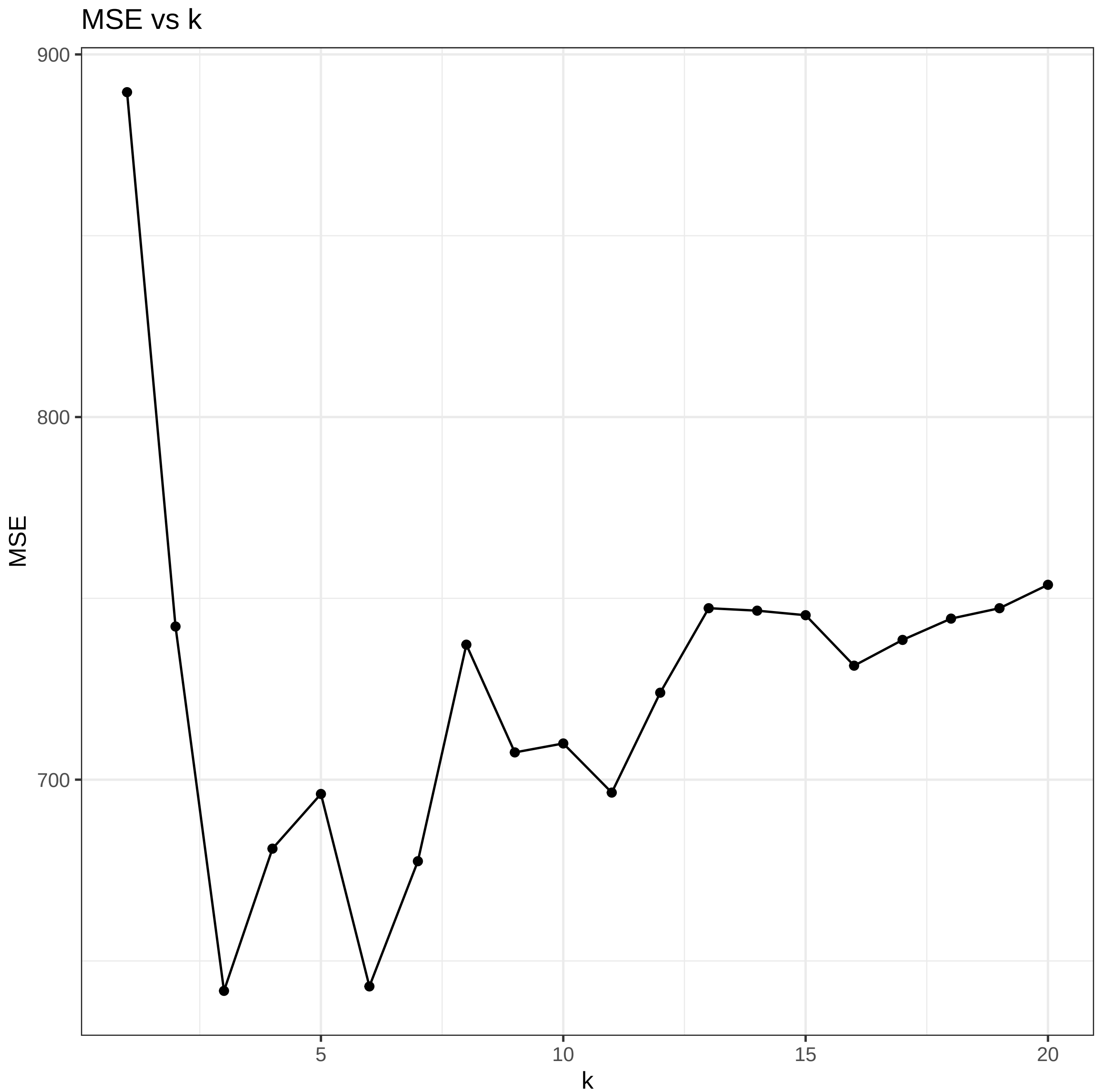
From the table, we can see that the best number of neighbors to use is k = 3, as it has the lowest MSE. This shows a decrease from k = 1, suggesting that using a small k can lead to overfitting. After k = 3, there's a noticeable increase in MSE values, with all of them going above 1000.

Looking at the graph, the red fitted line generally follows the data well. The data points aren't super tightly clustered around the line, but they're not too far off, either. There are a few outliers, particularly around a temperature of 80, where the points deviate from the fitted line. With all of this being said, there might be a other factors affecting the ozone, but the model seems to perform pretty well.

Part b

```r
# Set variables appropriately

y_train <- train_data$ozone

y_test <- test_data$ozone

X_train <- train_data[, -which(names(train_data) == "ozone")]

X_test <- test_data[, -which(names(test_data) == "ozone")]



# Initialize DataFrame for results

results2 <- data.frame()



# dwkNN for k = 1, ..., 20

for (k in 1:20) {

  predictions <- mykNN(train = as.matrix(X_train), test = as.matrix(X_test),

                       y_train = y_train, y_test = y_test, k = k, weighted = TRUE)



  # Get MSE

  MSE <- predictions$MSE
```

```
  # Append results

  results2 <- rbind(results2, data.frame(k = k, MSE = MSE))

}



# Plot MSE vs k

ggplot(results2, aes(x = k, y = MSE)) +

  geom_point() +

  geom_line() +

  theme_bw() +

  labs(title = "MSE vs k",

       x = "k",

       y = "MSE")
```



From looking at the graph, we see that as k increases, MSE initially increases. It then hits its absolute minimum value when k = 3 before starting to increase and then hitting a second local minimum when k = 6. Then, MSE increases for the next 2 k values, and staggers after that, with the MSE for each increasing k value being relatively similar to the last. Since our absolute minimum MSE value is when k = 3, we should use 3 nearest neighbors to have the best accuracy for our model. The next best option, and a very similar one regarding its MSE, would be k = 6.