CMDA-4654
Project1
Project 1

Xumanning Luo, Jack Jiang

10/26/2021

Problem 1

```
library(ggplot2)
library(gridExtra)
load("data/ozone.RData")
data("ozone")
```

1

```
for (D in 1:6){
  model_fit <- lm(ozone$ozone ~ poly(ozone$temperature,D))
  print(summary(model_fit))
}
```

```

Call:
lm(formula = ozone$ozone ~ poly(ozone$temperature, D))

Residuals:
    Min      1Q  Median      3Q     Max
-40.922 -17.459  -0.874  10.444 118.078

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)                   42.10       2.27   18.54   <2e-16 ***
poly(ozone$temperature, D)   243.79      23.92   10.19   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 23.92 on 109 degrees of freedom
Multiple R-squared:  0.488, Adjusted R-squared:  0.4833
F-statistic: 103.9 on 1 and 109 DF,  p-value: < 2.2e-16


Call:
lm(formula = ozone$ozone ~ poly(ozone$temperature, D))

Residuals:
    Min      1Q  Median      3Q     Max
-37.270 -12.462  -3.072   9.439 123.618

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                   42.099      2.156  19.529  < 2e-16 ***
poly(ozone$temperature, D)1  243.792     22.712  10.734  < 2e-16 ***
poly(ozone$temperature, D)2   81.600     22.712   3.593 0.000494 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.71 on 108 degrees of freedom
Multiple R-squared:  0.5426,    Adjusted R-squared:  0.5342
F-statistic: 64.07 on 2 and 108 DF,  p-value: < 2.2e-16


Call:
lm(formula = ozone$ozone ~ poly(ozone$temperature, D))

Residuals:
    Min      1Q  Median      3Q     Max
-36.771 -12.609  -2.017   9.834 122.229
```

```
Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                   42.099      2.153  19.552  < 2e-16 ***
poly(ozone$temperature, D)1  243.792     22.685  10.747  < 2e-16 ***
poly(ozone$temperature, D)2   81.600     22.685   3.597 0.000489 ***
poly(ozone$temperature, D)3  -25.366     22.685  -1.118 0.265996
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.69 on 107 degrees of freedom
Multiple R-squared:  0.5479,    Adjusted R-squared:  0.5352
F-statistic: 43.23 on 3 and 107 DF,  p-value: < 2.2e-16


Call:
lm(formula = ozone$ozone ~ poly(ozone$temperature, D))

Residuals:
    Min      1Q  Median      3Q     Max
-34.344 -10.655  -2.386   5.914 124.656

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                   42.099      2.101  20.038  < 2e-16 ***
poly(ozone$temperature, D)1  243.792     22.136  11.014  < 2e-16 ***
poly(ozone$temperature, D)2   81.600     22.136   3.686  0.00036 ***
poly(ozone$temperature, D)3  -25.366     22.136  -1.146  0.25439
poly(ozone$temperature, D)4  -55.924     22.136  -2.526  0.01300 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.14 on 106 degrees of freedom
Multiple R-squared:  0.5736,    Adjusted R-squared:  0.5575
F-statistic: 35.65 on 4 and 106 DF,  p-value: < 2.2e-16


Call:
lm(formula = ozone$ozone ~ poly(ozone$temperature, D))

Residuals:
    Min      1Q  Median      3Q     Max
-34.293 -11.126  -2.885   5.488 125.390

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                   42.099      2.109  19.966  < 2e-16 ***
poly(ozone$temperature, D)1  243.792     22.215  10.974  < 2e-16 ***
poly(ozone$temperature, D)2   81.600     22.215   3.673 0.000379 ***
poly(ozone$temperature, D)3  -25.366     22.215  -1.142 0.256110
poly(ozone$temperature, D)4  -55.924     22.215  -2.517 0.013333 *
poly(ozone$temperature, D)5  -10.950     22.215  -0.493 0.623109
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.21 on 105 degrees of freedom
Multiple R-squared:  0.5746,    Adjusted R-squared:  0.5543
F-statistic: 28.36 on 5 and 105 DF,  p-value: < 2.2e-16
```

```
Call:
lm(formula = ozone$ozone ~ poly(ozone$temperature, D))

Residuals:
    Min      1Q  Median      3Q     Max
-34.348 -11.132  -2.849   5.513 125.401

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                   42.0991     2.1187  19.871  < 2e-16 ***
poly(ozone$temperature, D)1 243.7919    22.3215  10.922  < 2e-16 ***
poly(ozone$temperature, D)2  81.5998    22.3215   3.656 0.000404 ***
poly(ozone$temperature, D)3 -25.3664    22.3215  -1.136 0.258396
poly(ozone$temperature, D)4 -55.9240    22.3215  -2.505 0.013783 *
poly(ozone$temperature, D)5 -10.9499    22.3215  -0.491 0.624776
poly(ozone$temperature, D)6   0.5638    22.3215   0.025 0.979898
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.32 on 104 degrees of freedom
Multiple R-squared:  0.5746,    Adjusted R-squared:   0.55
F-statistic: 23.41 on 6 and 104 DF,  p-value: < 2.2e-16
```

The polynomial fit with degree 4 appears to work the best since it explains 55.7% of the values and has the least residual standard error of 22.14.

2

```
spans <- seq(0.3, 0.75, by = 0.05)
fit1 <- myloess(ozone$temperature, ozone$ozone, span = 0.25, degree = 1, show.plot = FALSE)
fit2 <- myloess(ozone$temperature, ozone$ozone, span = 0.25, degree = 2, show.plot = FALSE)
result_1 <- data.frame(fit1[1:6])
result_2 <- data.frame(fit2[1:6])
for (span in spans){
  fit1 <- myloess(ozone$temperature, ozone$ozone, span = span, degree = 1, show.plot = FALSE)
  fit2 <- myloess(ozone$temperature, ozone$ozone, span = span, degree = 2, show.plot = FALSE)
  result_1[nrow(result_1) + 1,] <- fit1[1:6]
  result_2[nrow(result_2) + 1,] <- fit2[1:6]
}
```

```
result_1
```

```
   span degree N_total Win_total n_points      SSE
1  0.25      1     111        92       28 68404.65
2  0.30      1     111        91       33 68454.98
3  0.35      1     111        87       39 67713.29
4  0.40      1     111        89       44 67554.85
5  0.45      1     111        84       50 66910.37
6  0.50      1     111        82       56 67113.06
7  0.55      1     111        78       61 67085.47
8  0.60      1     111        76       67 66822.52
9  0.65      1     111        71       72 66274.14
10 0.70      1     111        69       78 65823.34
11 0.75      1     111        64       83 65540.09
```

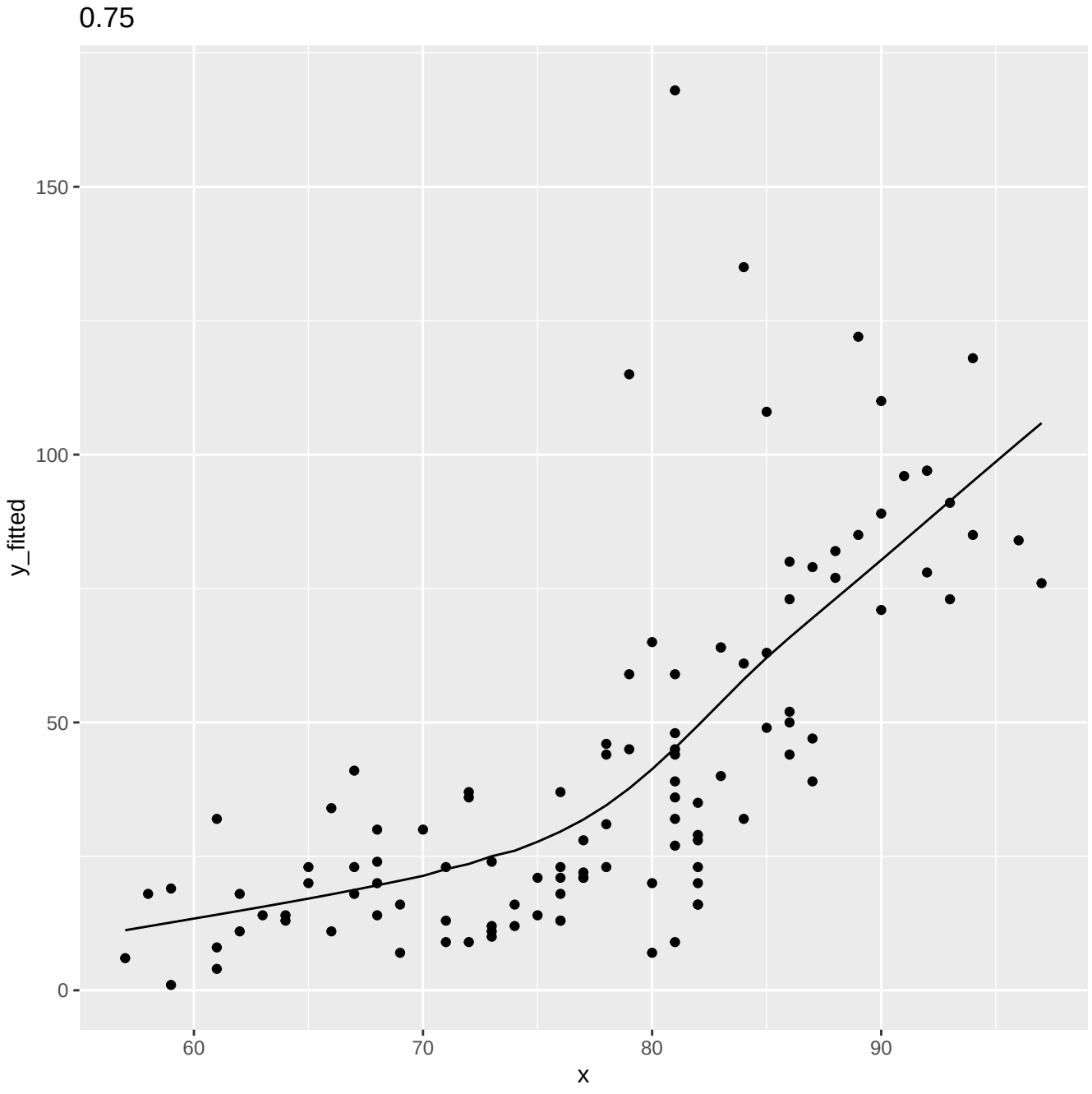
```
# double checking result_1 rows with least SSE
n1 <- length(result_1)
sort(result_1$SSE)[3]
```

```
[1] 66274.14
```

The three "best" degree = 1 fits, in order, are when span is equal to 0.75, 0.70 and 0.65 because they have the lowest SSE.

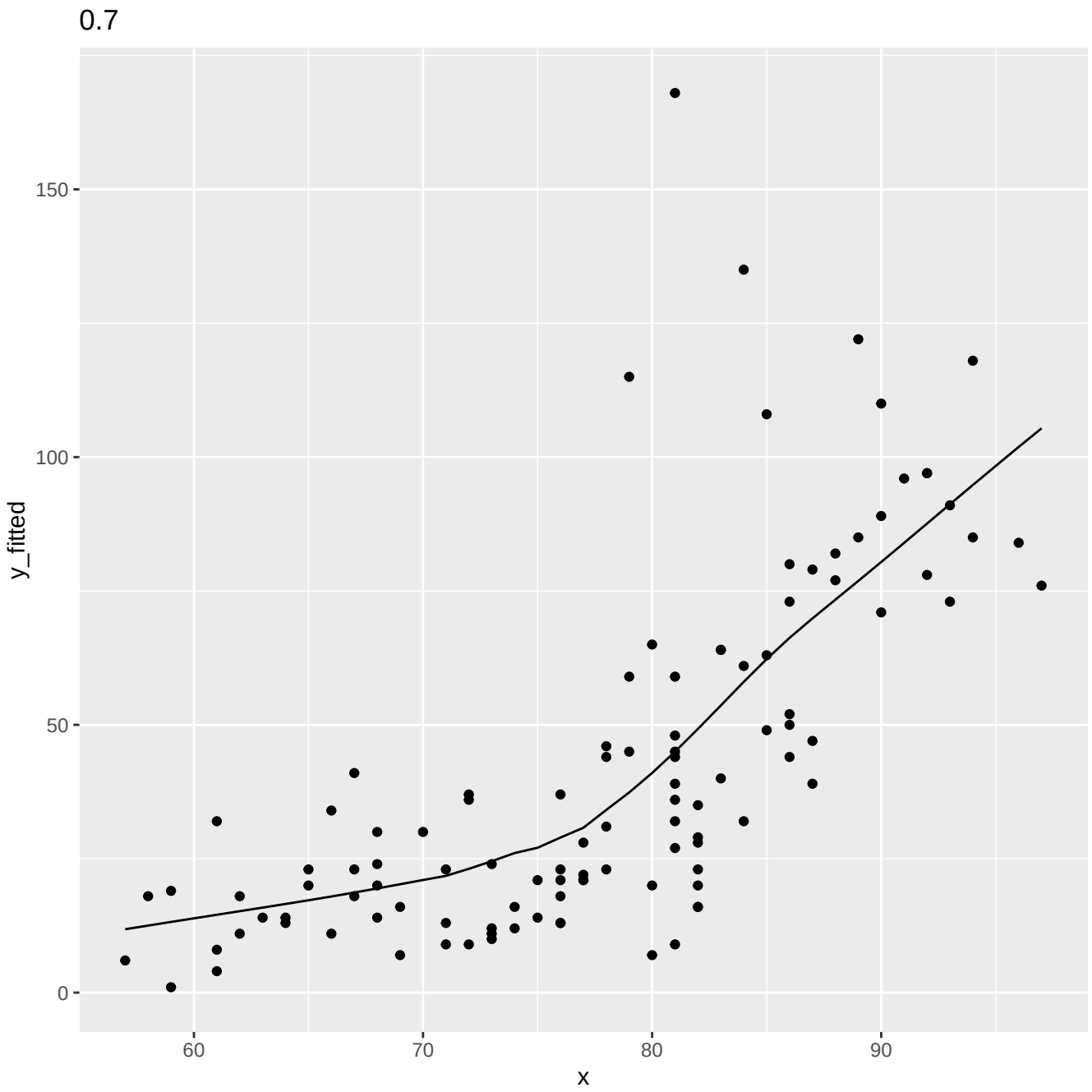
```
myloess(ozone$temperature, ozone$ozone, span = 0.75, degree = 1, show.plot = FALSE)[7]
```

```
$loessplot
```



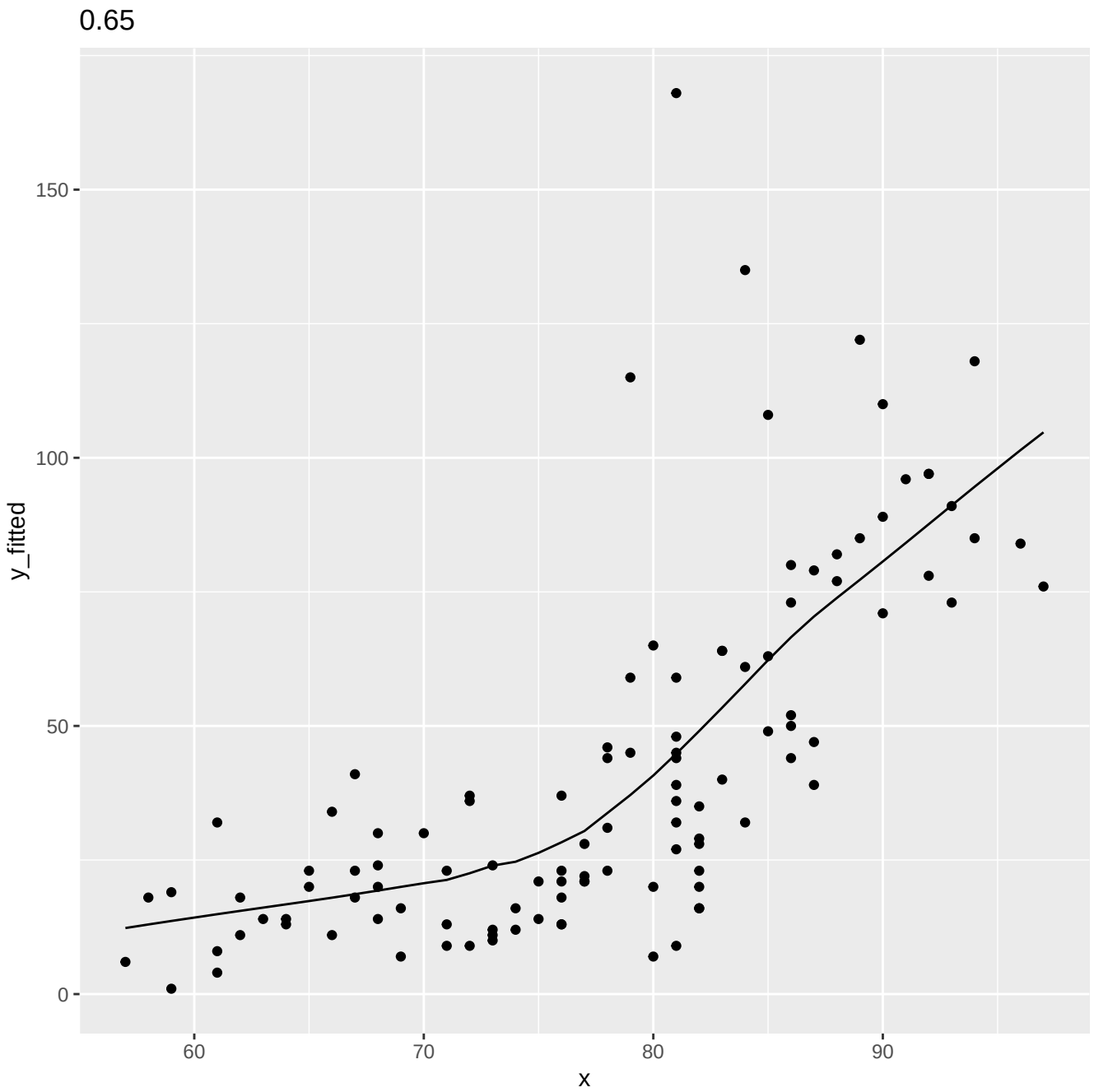

```
myloess(ozone$temperature, ozone$ozone, span = 0.70, degree = 1, show.plot = FALSE)[7]
```

```
$loessplot
```




```
myloess(ozone$temperature, ozone$ozone, span = 0.65, degree = 1, show.plot = FALSE)[7]
```

```
$loessplot
```




```
result_2
```

```
   span degree N_total Win_total n_points      SSE
1  0.25      2     111        92       28 75976.08
2  0.30      2     111        91       33 74105.68
3  0.35      2     111        87       39 70758.31
4  0.40      2     111        89       44 70249.67
5  0.45      2     111        84       50 69223.43
6  0.50      2     111        82       56 68669.44
7  0.55      2     111        78       61 68618.11
8  0.60      2     111        76       67 67456.79
9  0.65      2     111        71       72 67192.36
10 0.70      2     111        69       78 67386.45
11 0.75      2     111        64       83 67529.04
```

```
# double checking result_2 rows with least SSE
```

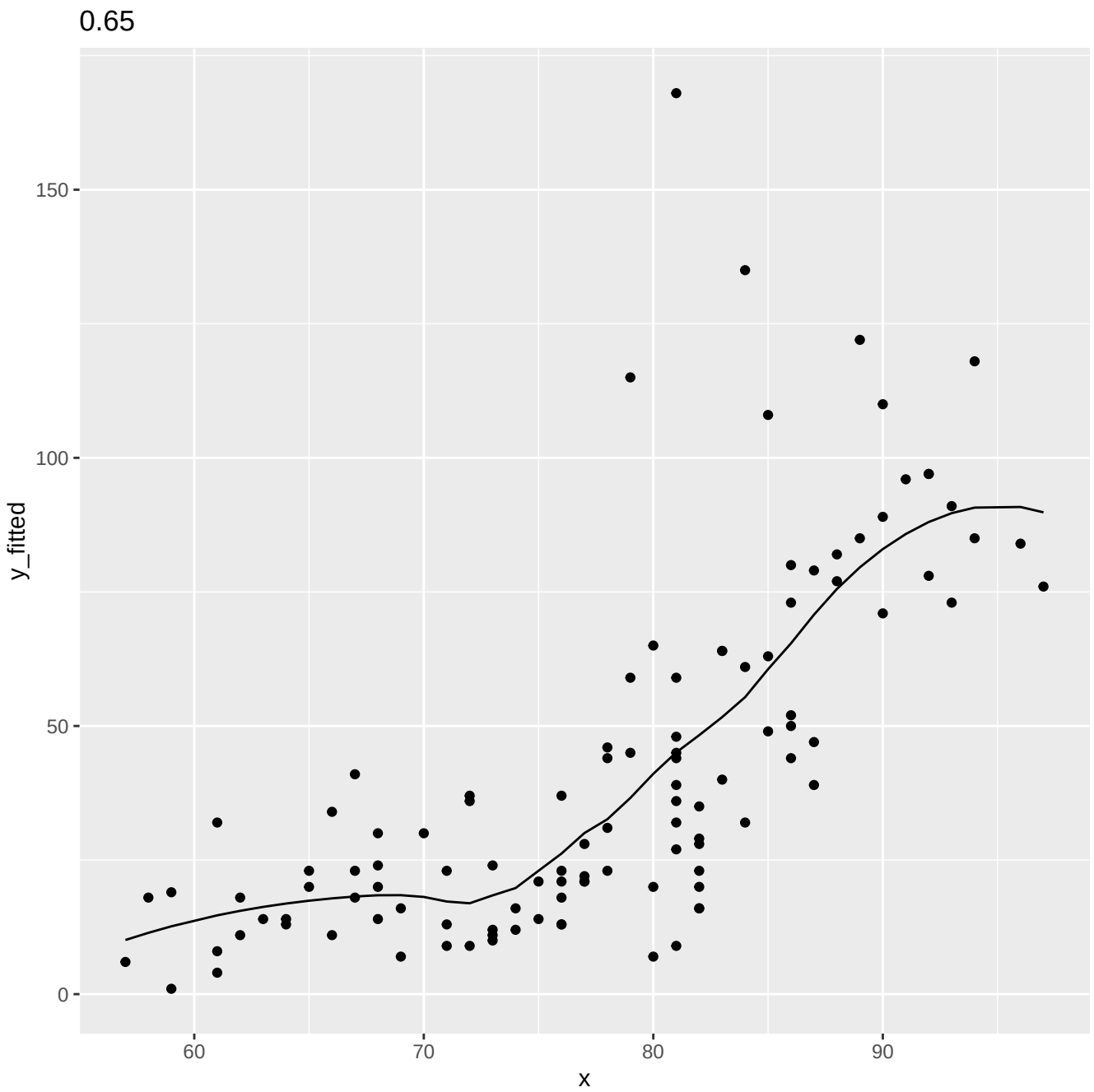
```
n2 <- length(result_2)
sort(result_2$SSE)[1]
```

```
[1] 67192.36
```

The three "best" degree = 2 fits, in order, are when span is equal to 0.65, 0.70 and 0.60 because they have the lowest SSE.

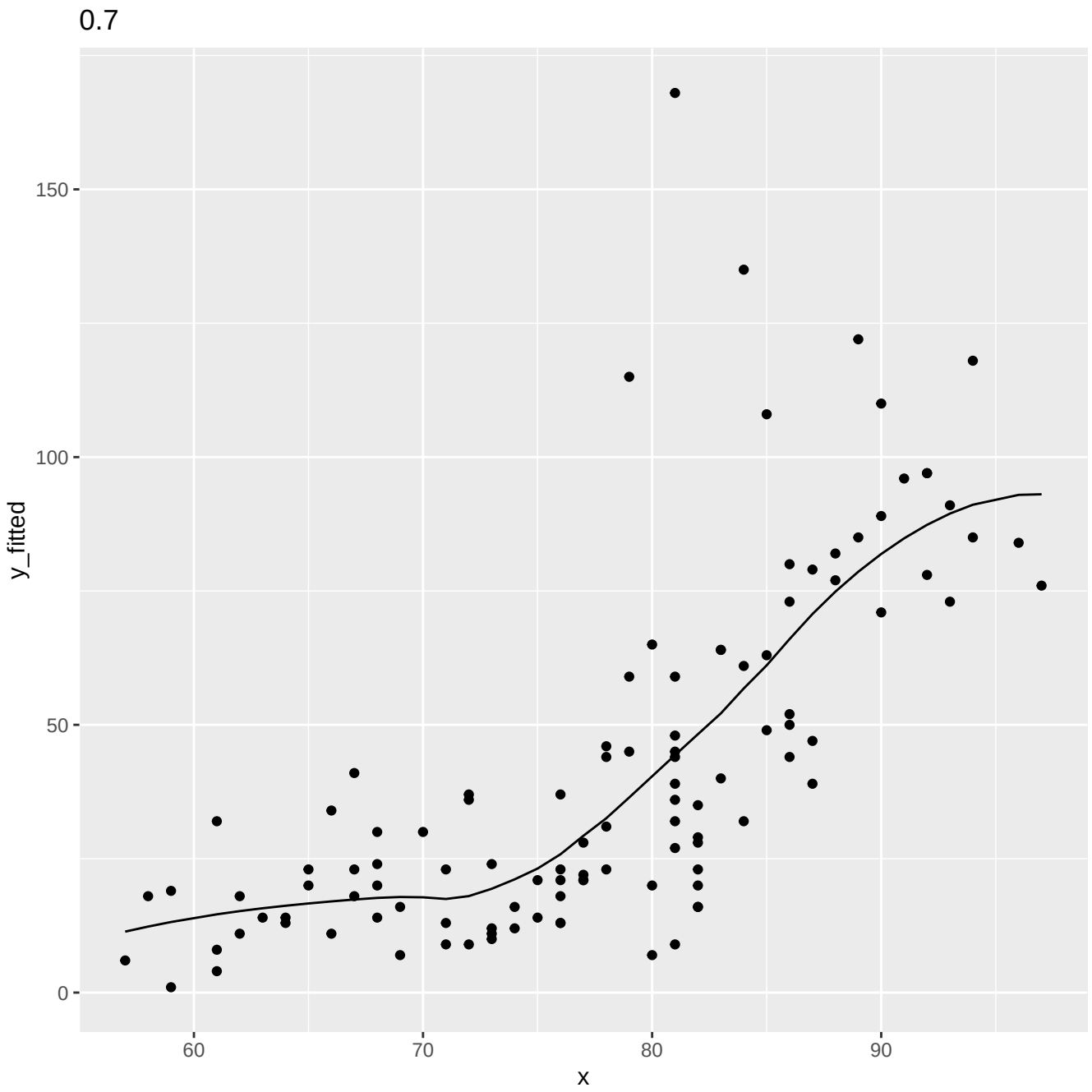
```
myloess(ozone$temperature, ozone$ozone, span = 0.65, degree = 2, show.plot = FALSE)[7]
```

```
$loessplot
```



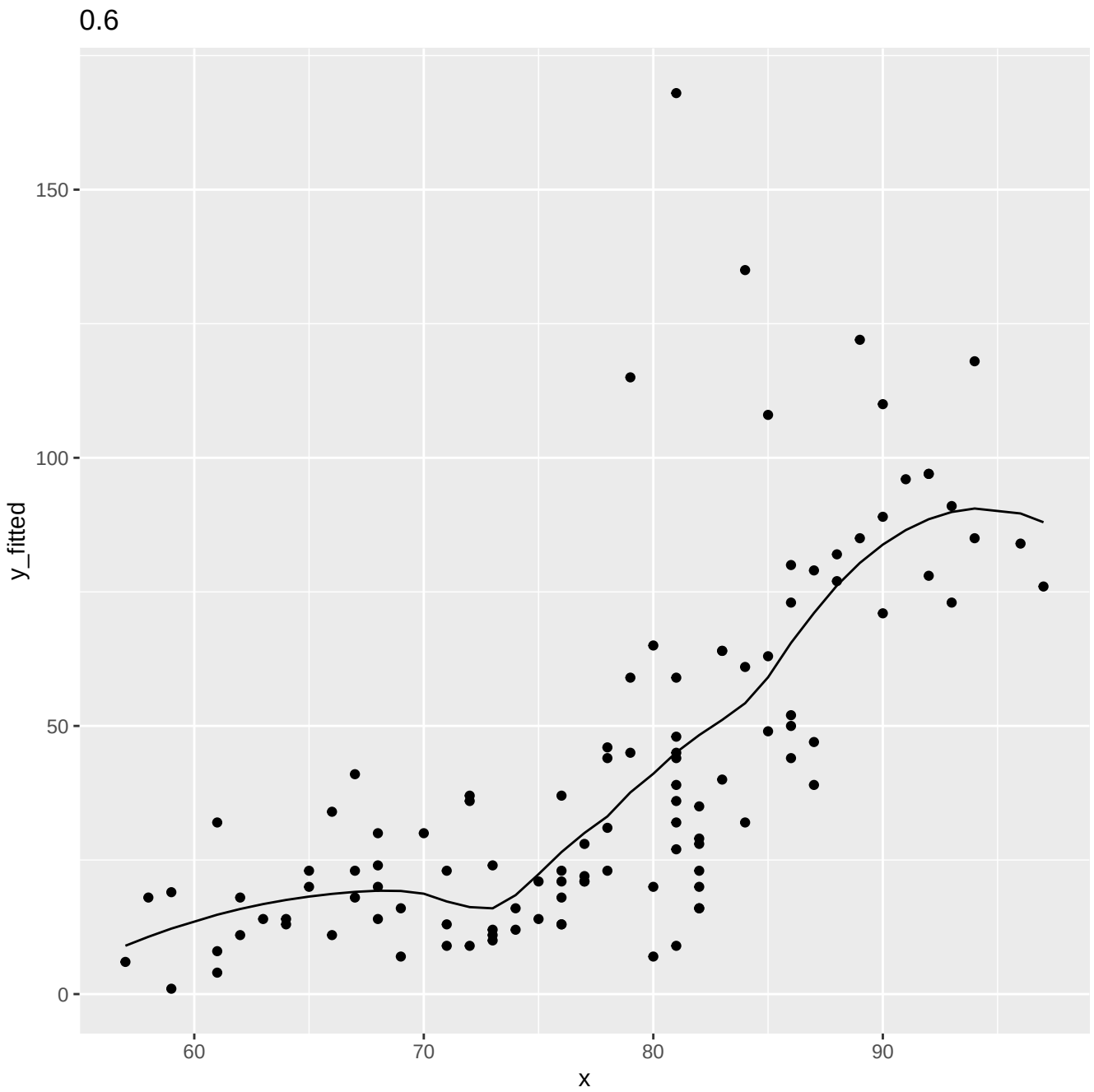

```
myloess(ozone$temperature, ozone$ozone, span = 0.70, degree = 2, show.plot = FALSE)[7]
```

```
$loessplot
```




```
myloess(ozone$temperature, ozone$ozone, span = 0.60, degree = 2, show.plot = FALSE)[7]
```

```
$loessplot
```



We did determine our "best" fits by picking the models with the lowest residual standard errors, using the SSE. For $degree = 1$, when visually inspecting the 2^{nd} ($span = 0.70$) and 3^{rd} ($span = 0.65$) "best" fits and comparing it to the best ($span = 0.75$), we believe our the data is not over-fit for our best fit as its regression fit line is smoothest and appears to fit best between all of the data points. For $degree = 2$, taking a look at the plots of the 3 "best" fits, when comparing our 2^{nd} ($span = 0.70$) and 3^{rd} ($span = 0.60$) to our 1^{st} ($span = 0.65$), there does appear to be over-fitting and so we believe the data may be over-fit.

3

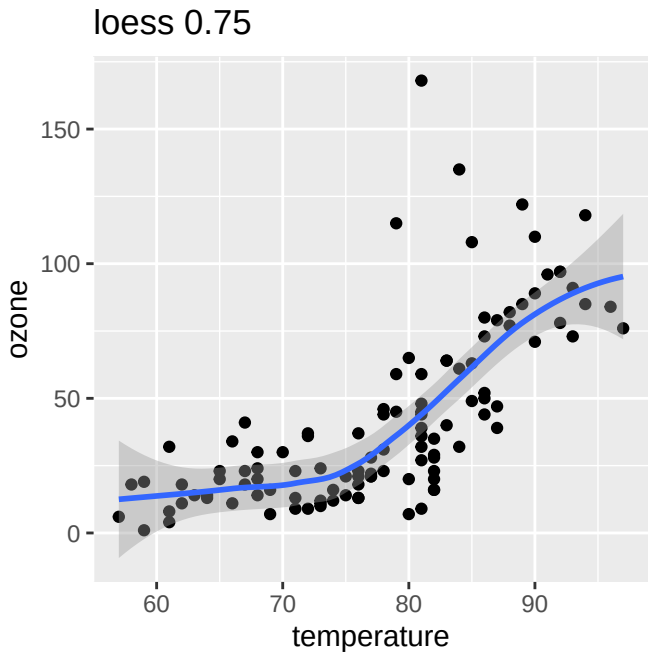
Now, we will compare our results and plots with the built-in *loess*() function.

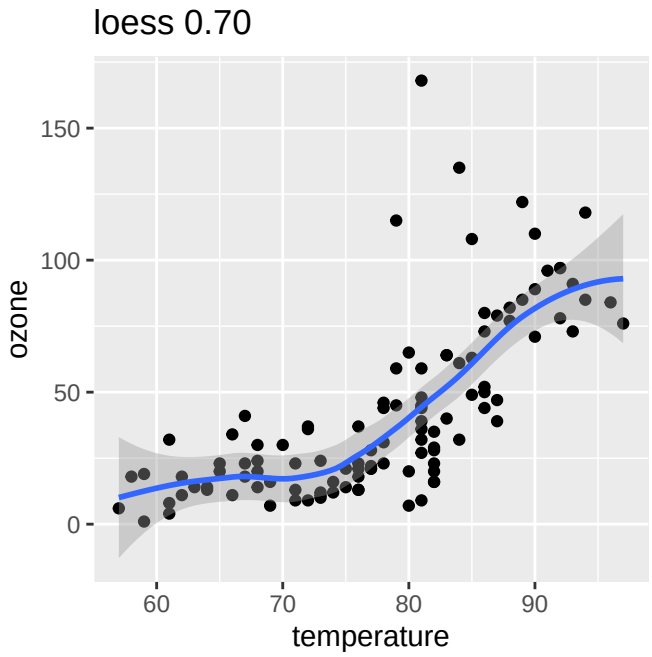
```
r_fit1_1 <- loess(ozone~temperature, data = ozone, span = 0.75, degree = 1, model = TRUE)
r_fit1_2 <- loess(ozone~temperature, data = ozone, span = 0.70, degree = 1, model = TRUE)
r_fit1_3 <- loess(ozone~temperature, data = ozone, span = 0.65, degree = 1, model = TRUE)
```

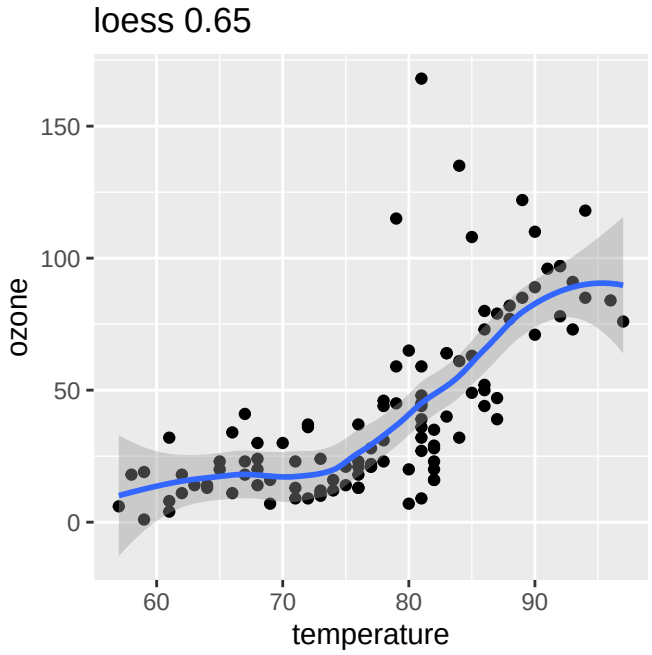
Plots for degree = 1:

```
r_fit1_1

Call:
loess(formula = ozone ~ temperature, data = ozone, model = TRUE,
    span = 0.75, degree = 1)

Number of Observations: 111
Equivalent Number of Parameters: 3.02
Residual Standard Error: 22.48

r_fit1_2

Call:
loess(formula = ozone ~ temperature, data = ozone, model = TRUE,
    span = 0.7, degree = 1)

Number of Observations: 111
Equivalent Number of Parameters: 3.26
Residual Standard Error: 22.44

r_fit1_3

Call:
loess(formula = ozone ~ temperature, data = ozone, model = TRUE,
    span = 0.65, degree = 1)

Number of Observations: 111
Equivalent Number of Parameters: 3.42
Residual Standard Error: 22.44
```

```
rp1_1 <- ggplot(ozone, aes(temperature, ozone)) + geom_point()
rp1_1 <- rp1_1 + geom_smooth(method="loess", span=0.75) + ggtitle("loess 0.75")

rp1_2 <- ggplot(ozone, aes(temperature, ozone)) + geom_point()
rp1_2 <- rp1_2 + geom_smooth(method="loess", span=0.70) + ggtitle("loess 0.70")

rp1_3 <- ggplot(ozone, aes(temperature, ozone)) + geom_point()
rp1_3 <- rp1_3 + geom_smooth(method="loess", span=0.65) + ggtitle("loess 0.65")


p1_1 <- myloess(ozone$temperature, ozone$ozone, span = 0.75, degree = 1, show.plot = FALSE)[7]
p1_2 <- myloess(ozone$temperature, ozone$ozone, span = 0.70, degree = 1, show.plot = FALSE)[7]
p1_3 <- myloess(ozone$temperature, ozone$ozone, span = 0.65, degree = 1, show.plot = FALSE)[7]

grid.arrange(rp1_1, rp1_2, rp1_3, heights = c(3, 3), widths = c(3, 3))
```







Plots for degree = 2:

```
r_fit2_1 <- loess(ozone~temperature, data = ozone, span = 0.65, degree = 1, model = TRUE)
r_fit2_2 <- loess(ozone~temperature, data = ozone, span = 0.70, degree = 1, model = TRUE)
r_fit2_3 <- loess(ozone~temperature, data = ozone, span = 0.60, degree = 1, model = TRUE)
r_fit2_1
```

```
Call:
loess(formula = ozone ~ temperature, data = ozone, model = TRUE,
    span = 0.65, degree = 1)

Number of Observations: 111
Equivalent Number of Parameters: 3.42
Residual Standard Error: 22.44
```

```
r_fit2_2
```

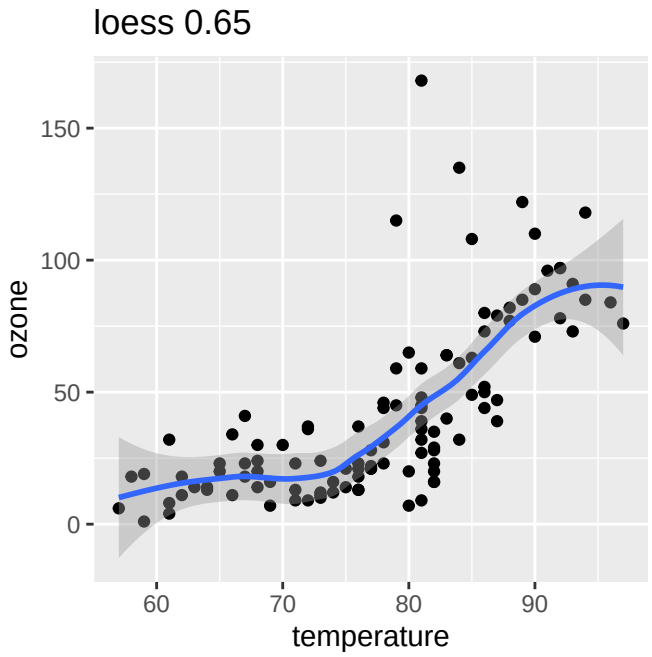
```
Call:
loess(formula = ozone ~ temperature, data = ozone, model = TRUE,
    span = 0.7, degree = 1)

Number of Observations: 111
Equivalent Number of Parameters: 3.26
Residual Standard Error: 22.44
```

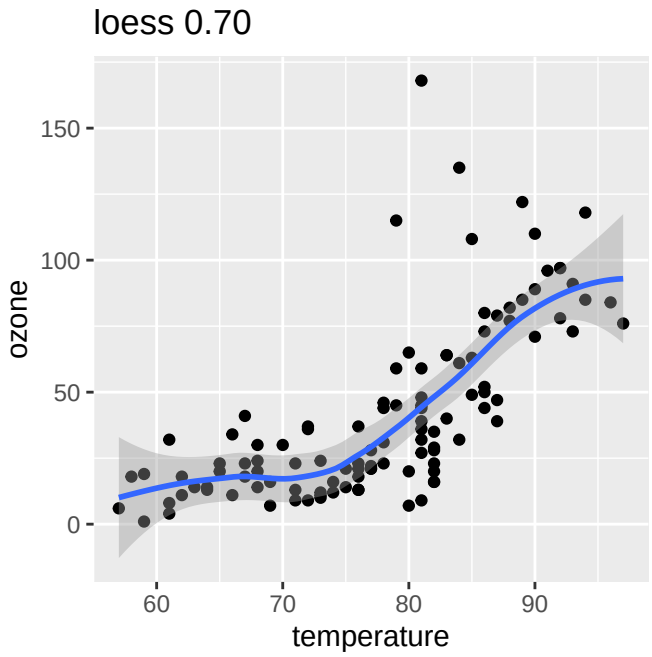
```
r_fit2_3
```

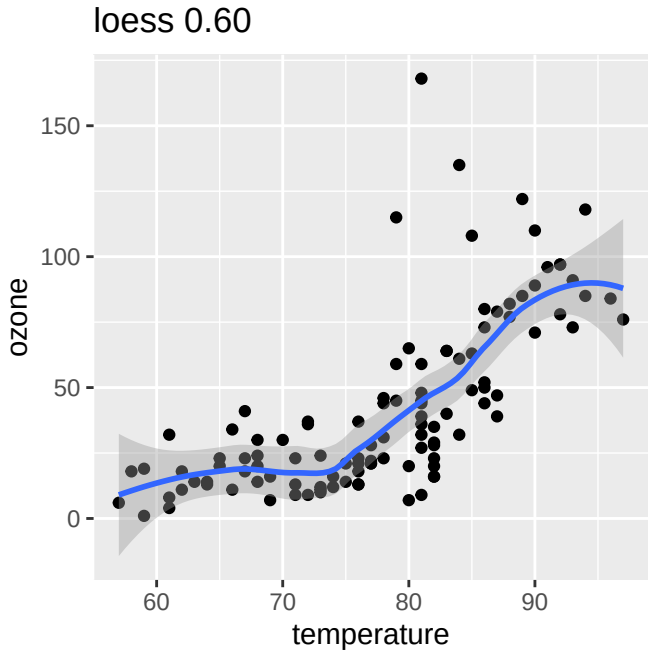
```
Call:
loess(formula = ozone ~ temperature, data = ozone, model = TRUE,
    span = 0.6, degree = 1)

Number of Observations: 111
Equivalent Number of Parameters: 3.64
Residual Standard Error: 22.45
```

```
rp2_1 <- ggplot(ozone, aes(temperature, ozone)) + geom_point()
rp2_1 <- rp2_1 + geom_smooth(method="loess", span=0.65) + ggtitle("loess 0.65")

rp2_2 <- ggplot(ozone, aes(temperature, ozone)) + geom_point()
rp2_2 <- rp2_2 + geom_smooth(method="loess", span=0.70) + ggtitle("loess 0.70")

rp2_3 <- ggplot(ozone, aes(temperature, ozone)) + geom_point()
rp2_3 <- rp2_3 + geom_smooth(method="loess", span=0.60) + ggtitle("loess 0.60")
```

```
p2_1 <- myloess(ozone$temperature, ozone$ozone, span = 0.65, degree = 2, show.plot = FALSE)[7]
p2_2 <- myloess(ozone$temperature, ozone$ozone, span = 0.70, degree = 2, show.plot = FALSE)[7]
p2_3 <- myloess(ozone$temperature, ozone$ozone, span = 0.60, degree = 2, show.plot = FALSE)[7]

grid.arrange(rp2_1, rp2_2, rp2_3, heights = c(3, 3), widths = c(3, 3))
```



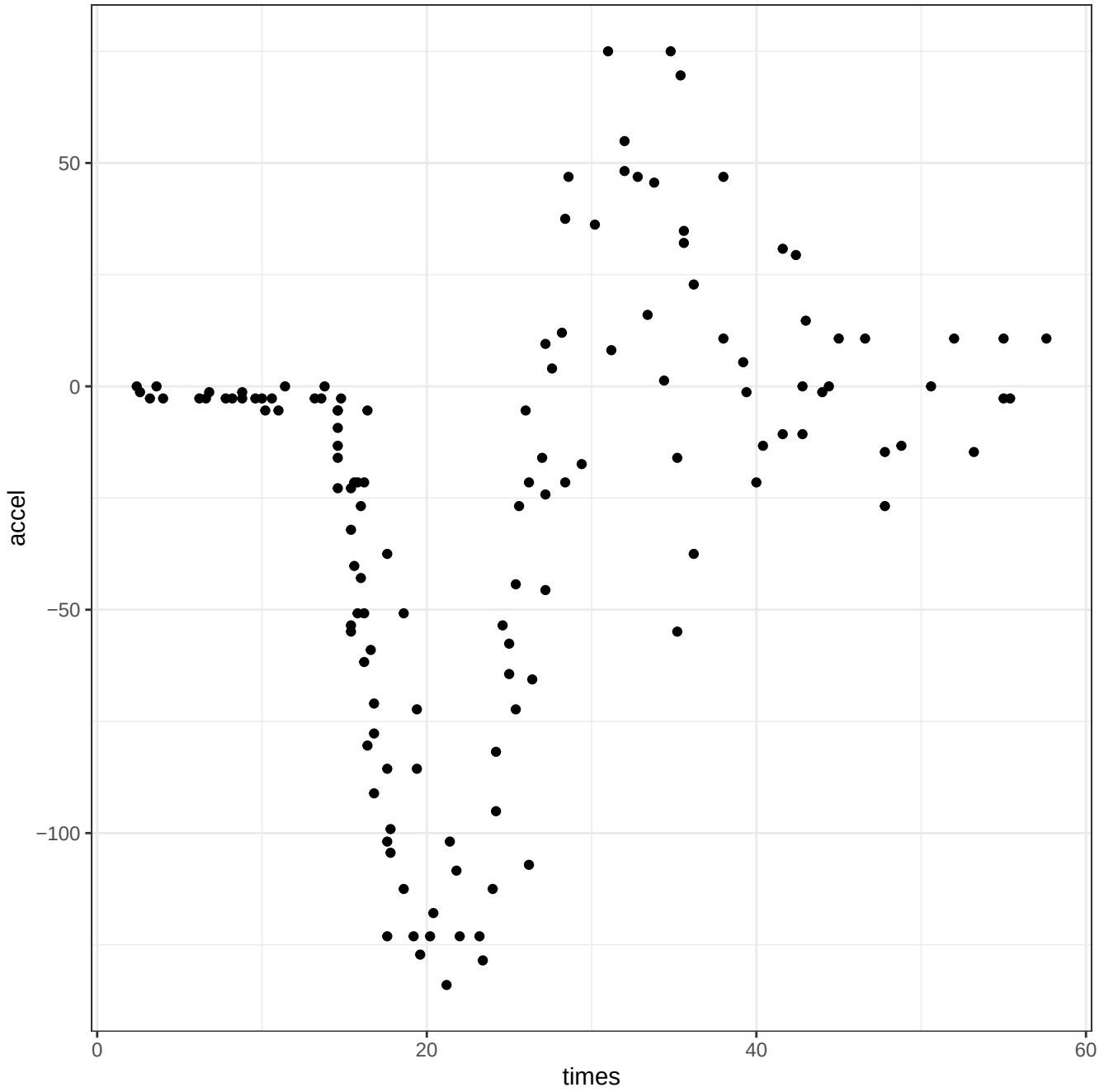




Observing the plots created from using *myloess*() and R's built in *loess*() function, they appear to be very similar. However, our loess function, *myloess*(), appears to be more smooth and fit the data points slightly better.

Problem 2

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

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



1

```
spans <- seq(0.3, 0.75, by = 0.05)
mass_fit1 <- myloess(mcycle$times, mcycle$accel, span = 0.25, degree = 1, show.plot = FALSE)
mass_fit2 <- myloess(mcycle$times, mcycle$accel, span = 0.25, degree = 2, show.plot = FALSE)
mass_result1 <- data.frame(mass_fit1[1:6])
mass_result2 <- data.frame(mass_fit2[1:6])
for (span in spans){
  mass_fit1 <- myloess(mcycle$times, mcycle$accel, span = span, degree = 1, show.plot = FALSE)
  mass_fit2 <- myloess(mcycle$times, mcycle$accel, span = span, degree = 2, show.plot = FALSE)
  mass_result1[nrow(mass_result1) + 1,] <- mass_fit1[1:6]
  mass_result2[nrow(mass_result2) + 1,] <- mass_fit2[1:6]
}
```

```
mass_result1
```

```
   span degree N_total Win_total n_points       SSE
1  0.25      1     133        51       33 207113.52
2  0.30      1     133        51       40 190794.64
3  0.35      1     133        52       47 173262.00
4  0.40      1     133        51       53 158280.19
5  0.45      1     133        42       60 142727.99
6  0.50      1     133        39       66 129762.76
7  0.55      1     133        37       73 113995.50
8  0.60      1     133        32       80 100481.33
9  0.65      1     133        33       86  91067.53
10 0.70      1     133        27       93  79378.97
11 0.75      1     133        26      100  66022.69
```

The three "best" *degree* = 1 LOESS regression fits are when span = 0.75, 0.70, and 0.65 because they have the lowest SSE values.

```
mass_result2
```

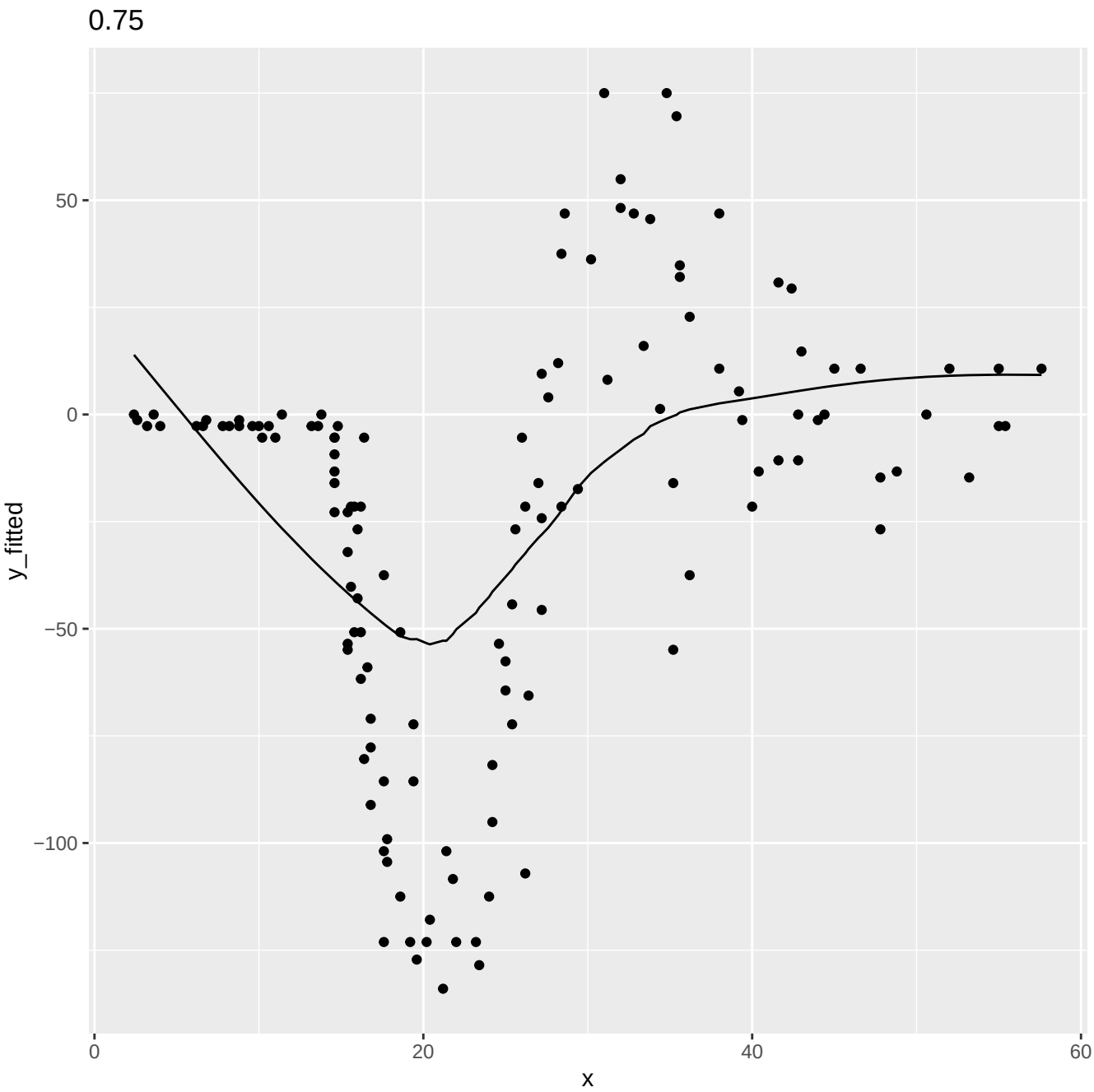
```
   span degree N_total Win_total n_points      SSE
1  0.25      2     133        51       33 246600.8
2  0.30      2     133        51       40 247040.5
3  0.35      2     133        52       47 241918.2
4  0.40      2     133        51       53 237238.5
5  0.45      2     133        42       60 235544.4
6  0.50      2     133        39       66 234815.3
7  0.55      2     133        37       73 233702.2
8  0.60      2     133        32       80 222187.7
9  0.65      2     133        33       86 206175.9
10 0.70      2     133        27       93 194146.2
11 0.75      2     133        26      100 187051.8
```

The three "best" *degree* = 2 LOESS regression fits are when span = 0.75, 0.70, and 0.65 because they have the lowest SSE values.

Here are the plots for our *myloess*() function for *degree* = 1:

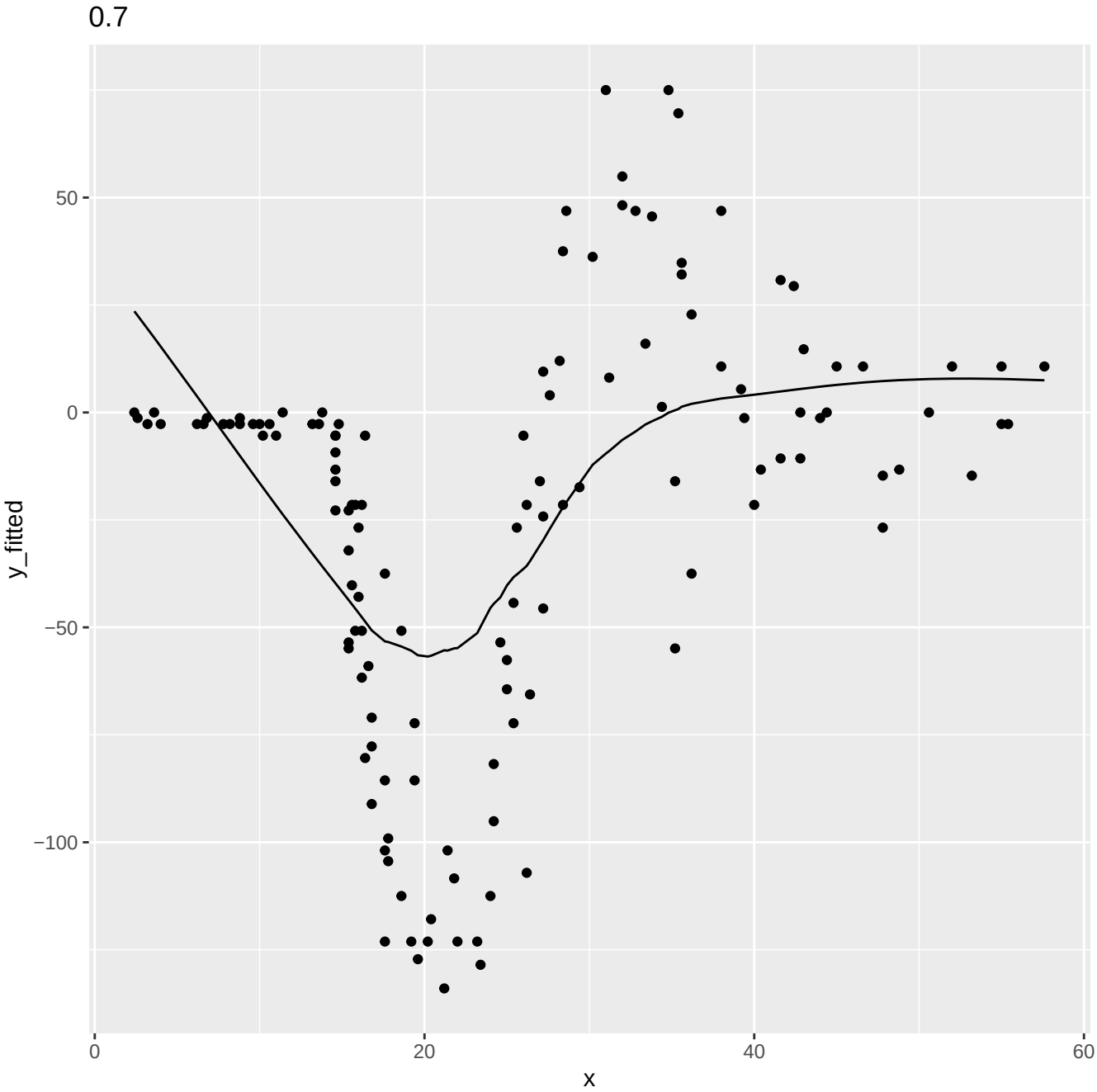
```
myloess(mcycle$times, mcycle$accel, span = 0.75, degree = 1, show.plot = FALSE)[7]
```

```
$loessplot
```



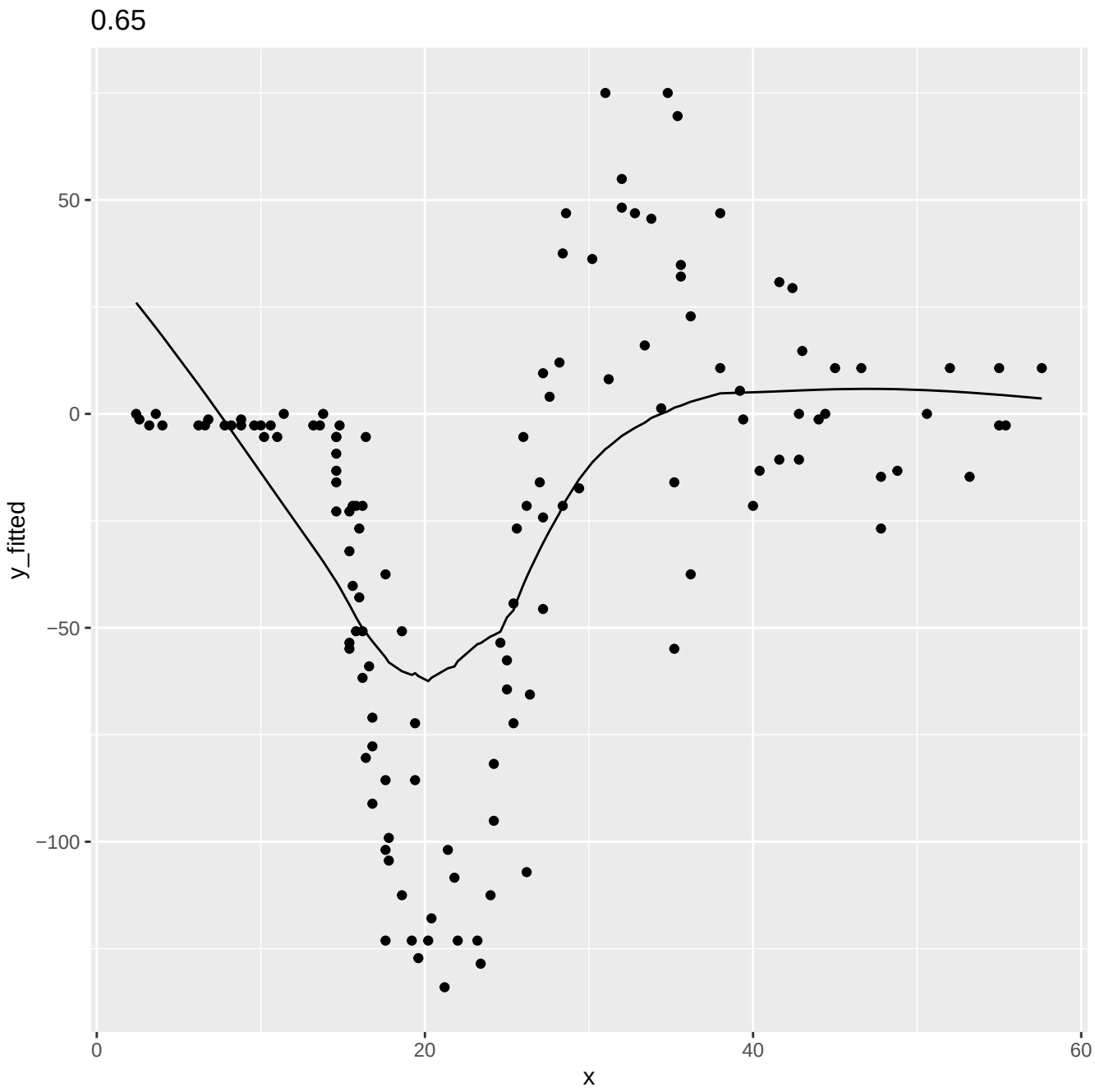

```
myloess(mcycle$times, mcycle$accel, span = 0.70, degree = 1, show.plot = FALSE)[7]
```

```
$loessplot
```




```
myloess(mcycle$times, mcycle$accel, span = 0.65, degree = 1, show.plot = FALSE)[7]
```

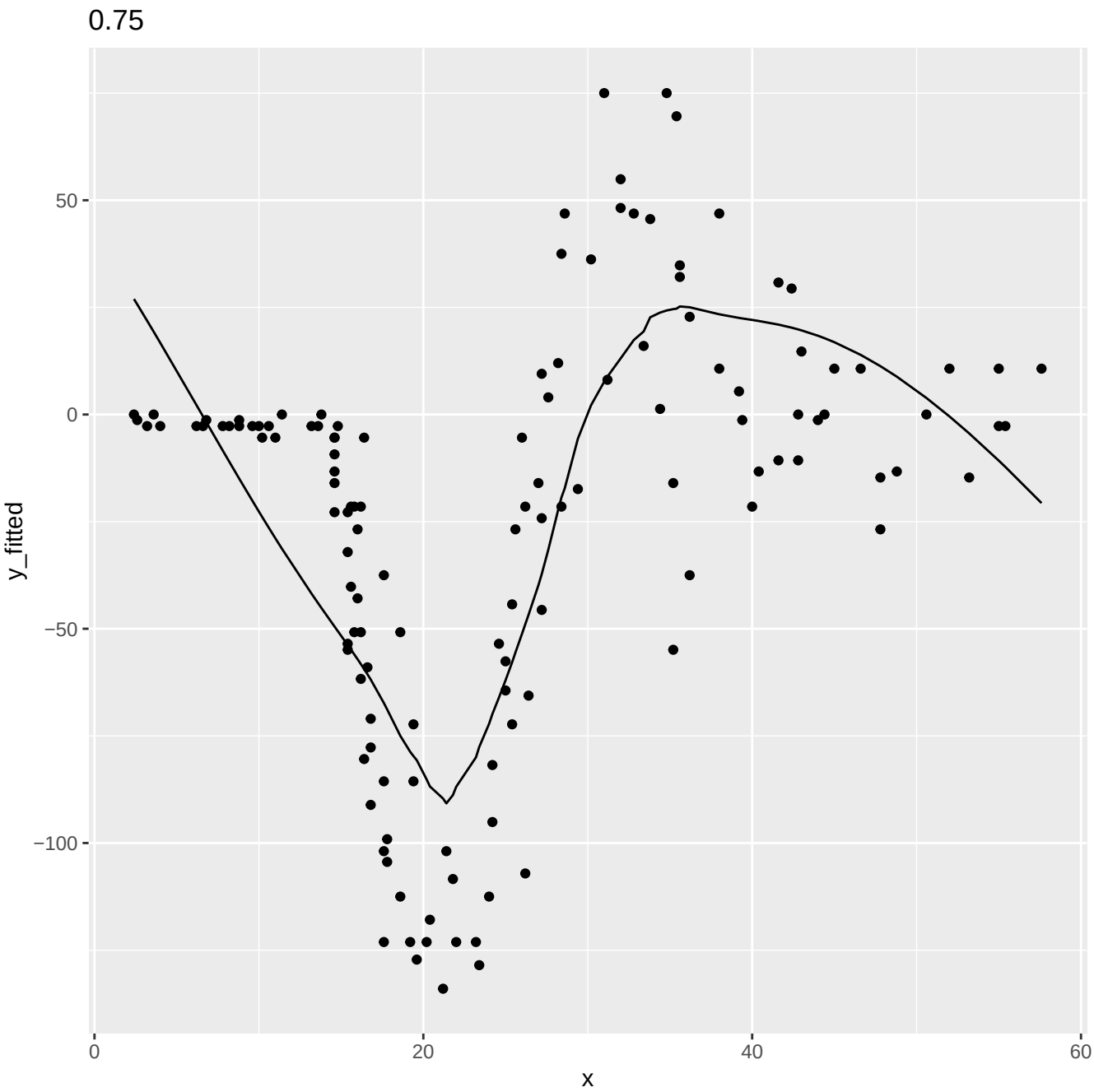
```
$loessplot
```



Here are the plots for our *myloess*() function for *degree* = 2:

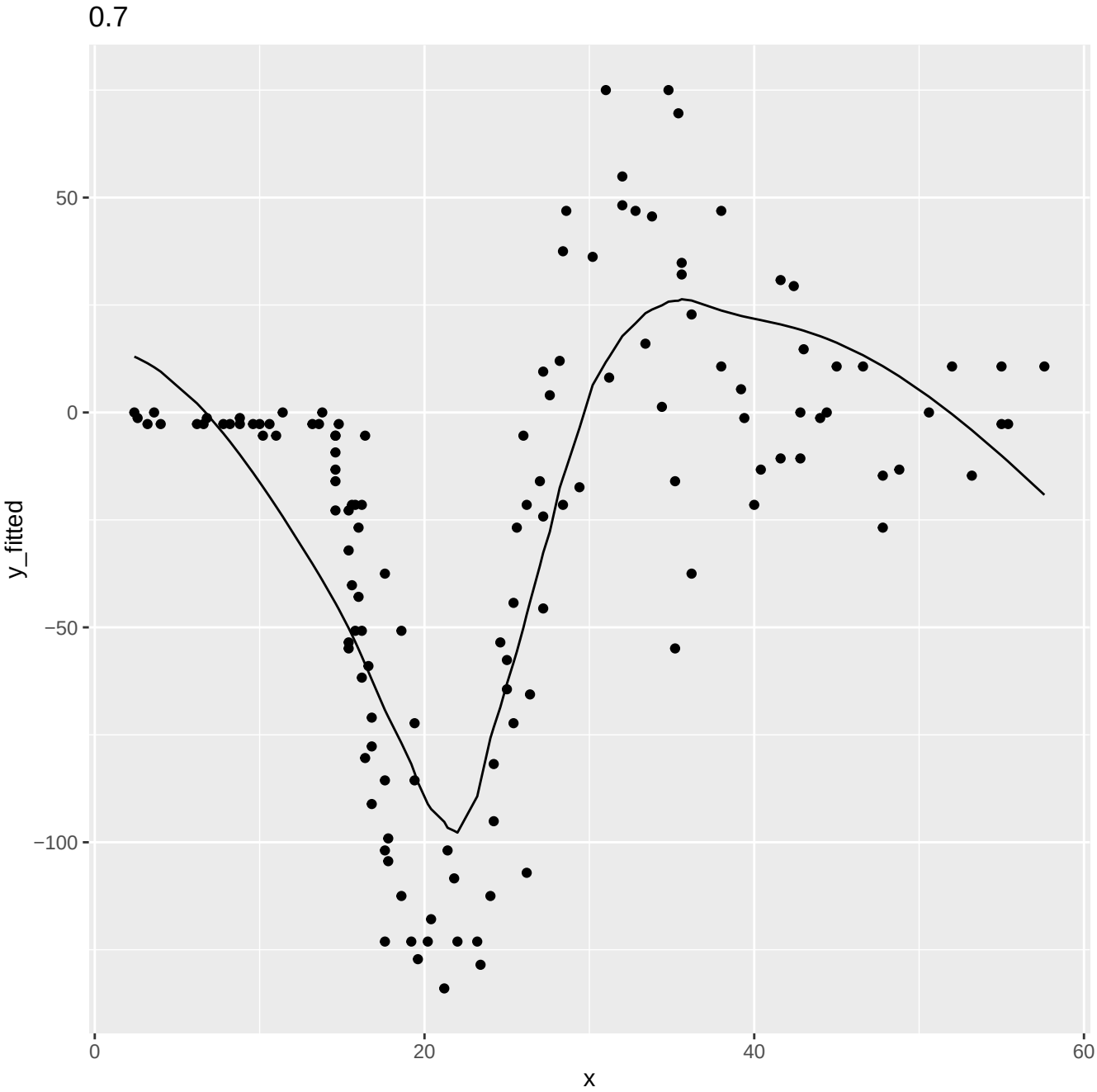
```
myloess(mcycle$times, mcycle$accel, span = 0.75, degree = 2, show.plot = FALSE)[7]
```

```
$loessplot
```



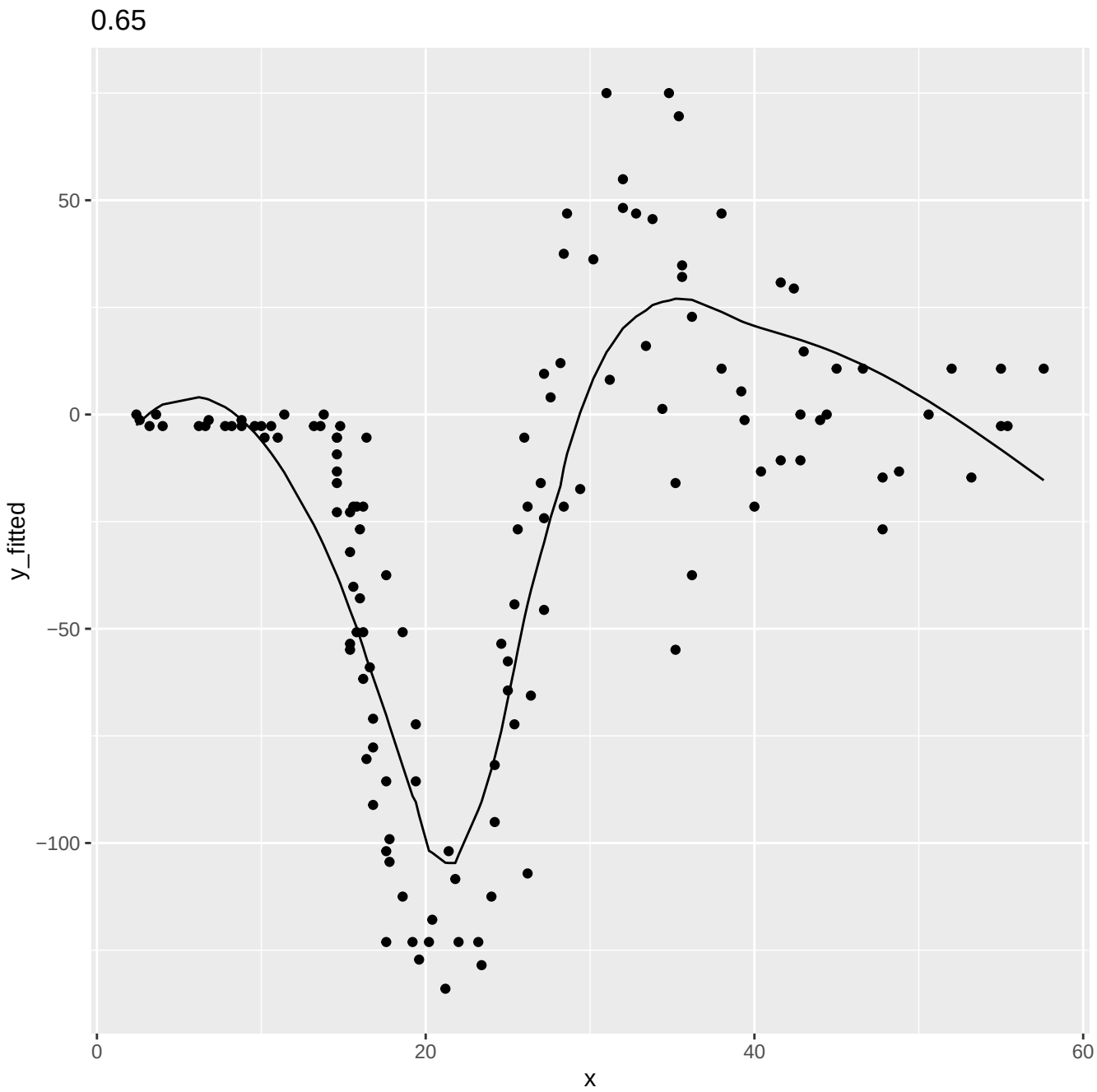

```
myloess(mcycle$times, mcycle$accel, span = 0.70, degree = 2, show.plot = FALSE)[7]
```

```
$loessplot
```




```
myloess(mcycle$times, mcycle$accel, span = 0.65, degree = 2, show.plot = FALSE)[7]
```

```
$loessplot
```



Upon visually inspecting the three best fits, for both $degree = 1$ and $degree = 2$, although when span = 0.75 has the lowest SSE, it fits the worst out of the 3 spans and span = 0.65 actually appears to fit the best. Thus, for $degree = 1$ and $degree = 2$, our model when span = 0.65 appears to provide the "best" fit.

2

Now, we will compare our results to the built-in *loess*() function.

Plots for degree = 1:

```
m_fit1_1 <- loess(times~accel, data = mcycle, span = 0.75, degree = 1, model = TRUE)
m_fit1_2 <- loess(times~accel, data = mcycle, span = 0.70, degree = 1, model = TRUE)
m_fit1_3 <- loess(times~accel, data = mcycle, span = 0.65, degree = 1, model = TRUE)

m_fit1_1
```

```
Call:
loess(formula = times ~ accel, data = mcycle, model = TRUE, span = 0.75,
    degree = 1)

Number of Observations: 133
Equivalent Number of Parameters: 3.33
Residual Standard Error: 12.52
```

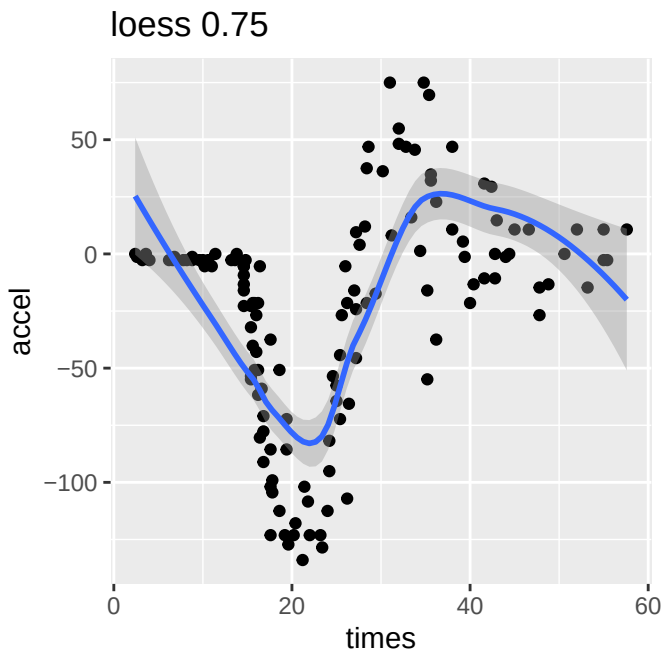
```
m_fit1_2
```

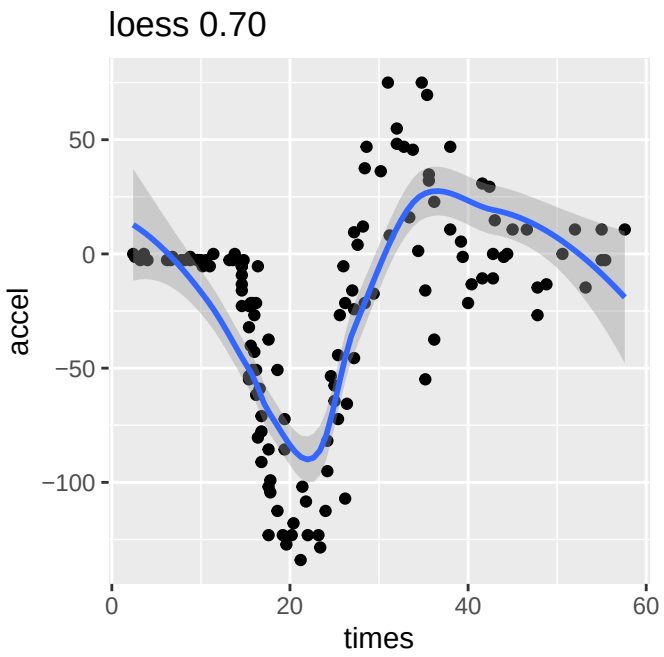
```
Call:
loess(formula = times ~ accel, data = mcycle, model = TRUE, span = 0.7,
    degree = 1)

Number of Observations: 133
Equivalent Number of Parameters: 3.57
Residual Standard Error: 12.53
```

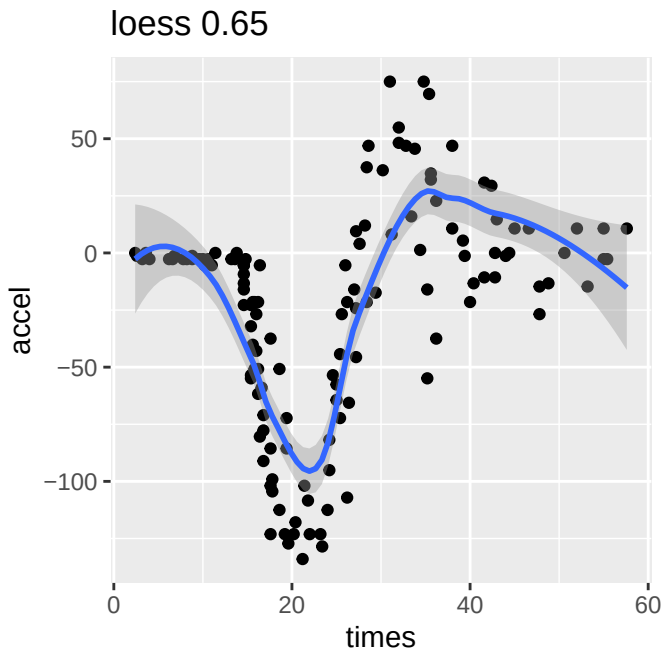
```
m_fit1_3
```

```
Call:
loess(formula = times ~ accel, data = mcycle, model = TRUE, span = 0.65,
    degree = 1)

Number of Observations: 133
Equivalent Number of Parameters: 3.88
Residual Standard Error: 12.49
```

```
mp1_1 <- ggplot(mcycle, aes(times, accel)) + geom_point()
mp1_1 <- mp1_1 + geom_smooth(method="loess", span=0.75) + ggtitle("loess 0.75")

mp1_2 <- ggplot(mcycle, aes(times, accel)) + geom_point()
mp1_2 <- mp1_2 + geom_smooth(method="loess", span=0.70) + ggtitle("loess 0.70")

mp1_3 <- ggplot(mcycle, aes(times, accel)) + geom_point()
mp1_3 <- mp1_3 + geom_smooth(method="loess", span=0.65) + ggtitle("loess 0.65")


my_mp1_1 <- myloess(mcycle$times, mcycle$accel, span = 0.75, degree = 1, show.plot = FALSE)[7]
my_mp1_2 <- myloess(mcycle$times, mcycle$accel, span = 0.70, degree = 1, show.plot = FALSE)[7]
my_mp1_3 <- myloess(mcycle$times, mcycle$accel, span = 0.65, degree = 1, show.plot = FALSE)[7]

grid.arrange(mp1_1, mp1_2, mp1_3, heights = c(3, 3), widths = c(3, 3))
```







Plots for degree = 2:

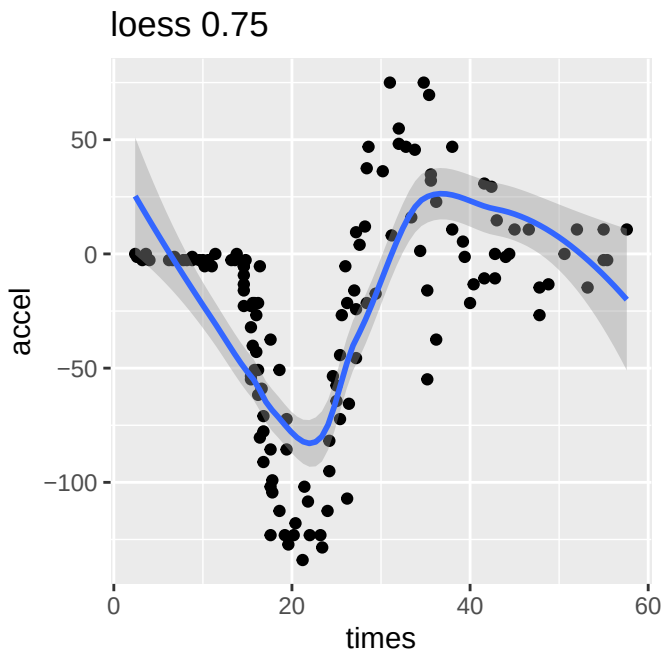
```
m_fit1_1 <- loess(times~accel, data = mcycle, span = 0.75, degree = 2, model = TRUE)
m_fit1_2 <- loess(times~accel, data = mcycle, span = 0.70, degree = 2, model = TRUE)
m_fit1_3 <- loess(times~accel, data = mcycle, span = 0.65, degree = 2, model = TRUE)

m_fit1_1

Call:
loess(formula = times ~ accel, data = mcycle, model = TRUE, span = 0.75,
    degree = 2)

Number of Observations: 133
Equivalent Number of Parameters: 4.96
Residual Standard Error: 12.5

m_fit1_2
```

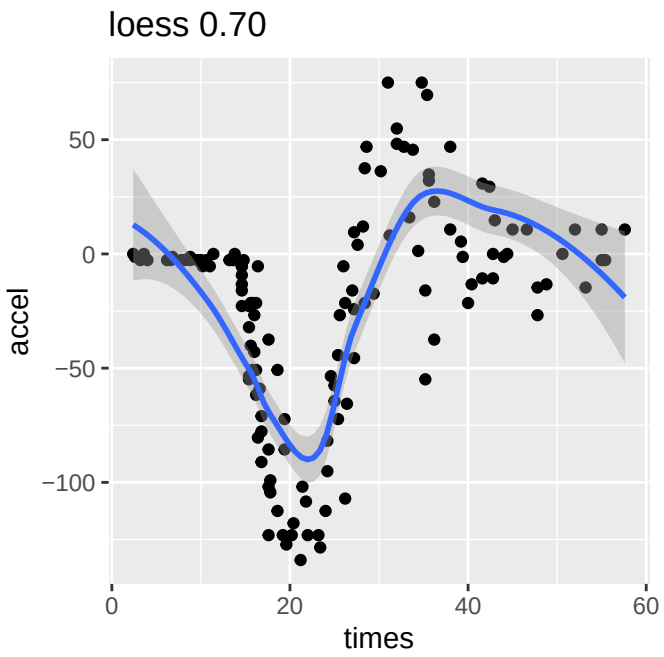
```
Call:
loess(formula = times ~ accel, data = mcycle, model = TRUE, span = 0.7,
    degree = 2)

Number of Observations: 133
Equivalent Number of Parameters: 5.3
Residual Standard Error: 12.46
```

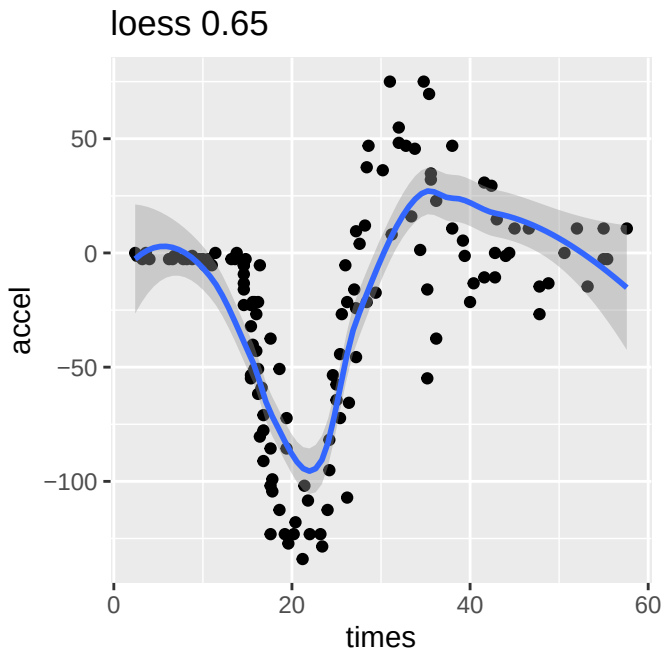
```
m_fit1_3
```

```
Call:
loess(formula = times ~ accel, data = mcycle, model = TRUE, span = 0.65,
    degree = 2)

Number of Observations: 133
Equivalent Number of Parameters: 5.82
Residual Standard Error: 12.37
```

```
mp1_1 <- ggplot(mcycle, aes(times, accel)) + geom_point()
mp1_1 <- mp1_1 + geom_smooth(method="loess", span=0.75) + ggtitle("loess 0.75")

mp1_2 <- ggplot(mcycle, aes(times, accel)) + geom_point()
mp1_2 <- mp1_2 + geom_smooth(method="loess", span=0.70) + ggtitle("loess 0.70")

mp1_3 <- ggplot(mcycle, aes(times, accel)) + geom_point()
mp1_3 <- mp1_3 + geom_smooth(method="loess", span=0.65) + ggtitle("loess 0.65")


my_mp1_1 <- myloess(mcycle$times, mcycle$accel, span = 0.75, degree = 2, show.plot = FALSE)[7]
my_mp1_2 <- myloess(mcycle$times, mcycle$accel, span = 0.70, degree = 2, show.plot = FALSE)[7]
my_mp1_3 <- myloess(mcycle$times, mcycle$accel, span = 0.65, degree = 2, show.plot = FALSE)[7]

grid.arrange(mp1_1, mp1_2, mp1_3, heights = c(3, 3), widths = c(3, 3))
```







Results for our *myloess*() function:

```
for (D in 1:6){
  mass_model_fit <- lm(mcycle$accel ~ poly(mcycle$times,D))
  print(summary(mass_model_fit))
}
```

```
Call:
lm(formula = mcycle$accel ~ poly(mcycle$times, D))

Residuals:
     Min       1Q   Median       3Q      Max
-104.114  -25.926    4.582   36.163   94.197

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)            -25.546      4.017  -6.359 3.11e-09 ***
poly(mcycle$times, D)  164.557     46.326   3.552 0.000532 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 46.33 on 131 degrees of freedom
Multiple R-squared:  0.08785,   Adjusted R-squared:  0.08089
F-statistic: 12.62 on 1 and 131 DF,  p-value: 0.0005318


Call:
lm(formula = mcycle$accel ~ poly(mcycle$times, D))

Residuals:
    Min      1Q  Median      3Q     Max
-96.527 -30.817   9.589  29.210 104.728

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)            -25.546      3.907  -6.539 1.29e-09 ***
poly(mcycle$times, D)1  164.557     45.058   3.652 0.000376 ***
poly(mcycle$times, D)2  131.227     45.058   2.912 0.004222 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 45.06 on 130 degrees of freedom
Multiple R-squared:  0.1437,    Adjusted R-squared:  0.1306
F-statistic: 10.91 on 2 and 130 DF,  p-value: 4.167e-05


Call:
lm(formula = mcycle$accel ~ poly(mcycle$times, D))

Residuals:
    Min      1Q  Median      3Q     Max
-83.083 -28.772   2.935  31.004  94.108

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)            -25.546      3.469  -7.365 1.87e-11 ***
poly(mcycle$times, D)1  164.557     40.002   4.114 6.89e-05 ***
poly(mcycle$times, D)2  131.227     40.002   3.280  0.00133 **
poly(mcycle$times, D)3 -239.790     40.002  -5.994 1.92e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 40 on 129 degrees of freedom
Multiple R-squared:  0.3303,    Adjusted R-squared:  0.3147
F-statistic: 21.21 on 3 and 129 DF,  p-value: 3.134e-11


Call:
lm(formula = mcycle$accel ~ poly(mcycle$times, D))

Residuals:
    Min      1Q  Median      3Q     Max
-82.860 -28.790   2.804  30.618  94.672

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)             -25.546      3.482  -7.337 2.22e-11 ***
poly(mcycle$times, D)1  164.557     40.154   4.098 7.34e-05 ***
poly(mcycle$times, D)2  131.227     40.154   3.268  0.00139 **
poly(mcycle$times, D)3 -239.790     40.154  -5.972 2.17e-08 ***
poly(mcycle$times, D)4   -6.738     40.154  -0.168  0.86701
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 40.15 on 128 degrees of freedom
Multiple R-squared:  0.3304,    Adjusted R-squared:  0.3095
F-statistic: 15.79 on 4 and 128 DF,  p-value: 1.572e-10


Call:
lm(formula = mcycle$accel ~ poly(mcycle$times, D))

Residuals:
    Min      1Q  Median      3Q     Max
-77.271 -21.285   0.975  25.386  82.371

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)             -25.546      2.940  -8.690 1.54e-14 ***
poly(mcycle$times, D)1  164.557     33.901   4.854 3.48e-06 ***
poly(mcycle$times, D)2  131.227     33.901   3.871 0.000173 ***
poly(mcycle$times, D)3 -239.790     33.901  -7.073 9.04e-11 ***
poly(mcycle$times, D)4   -6.738     33.901  -0.199 0.842779
poly(mcycle$times, D)5  245.799     33.901   7.250 3.60e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 33.9 on 127 degrees of freedom
Multiple R-squared:  0.5264,    Adjusted R-squared:  0.5078
F-statistic: 28.24 on 5 and 127 DF,  p-value: < 2.2e-16


Call:
lm(formula = mcycle$accel ~ poly(mcycle$times, D))

Residuals:
    Min      1Q  Median      3Q     Max
-76.816 -23.945   0.814  25.910  76.355

Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)             -25.546      2.879  -8.873 5.93e-15 ***
poly(mcycle$times, D)1  164.557     33.205   4.956 2.27e-06 ***
poly(mcycle$times, D)2  131.227     33.205   3.952 0.000128 ***
poly(mcycle$times, D)3 -239.790     33.205  -7.222 4.30e-11 ***
poly(mcycle$times, D)4   -6.738     33.205  -0.203 0.839527
poly(mcycle$times, D)5  245.799     33.205   7.403 1.67e-11 ***
poly(mcycle$times, D)6  -83.906     33.205  -2.527 0.012744 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 33.2 on 126 degrees of freedom
Multiple R-squared:  0.5493,    Adjusted R-squared:  0.5278
F-statistic: 25.59 on 6 and 126 DF,  p-value: < 2.2e-16
```

Comparing our *myloess*() function with R's built in *loess*() function, R's built in *loess*() function visually appears to fit the

data better. While observing the results, R's built in *loess*() function also appears to fit the data better because its standard residual error is lower than our functions fitted model.

Part 2

Problem 3

Given code to load data and clean it up.

```
library(ggplot2)
# 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
```

1

Normalize function

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
```

Here, we split the data set into training and testing data.

```
# Normalize the dataset
Auto_norm <- as.data.frame(lapply(Auto_new[,1:7], normalize))
set.seed(123)
# Randomize 70% of to
ran <- sample(1:nrow(Auto_new), size=nrow(Auto_new)*0.7, replace=FALSE)
# 70% training data set
train_auto <- Auto_norm[ran,]
# 30% testing data set
test_auto <- Auto_norm[-ran,]
# Training labels
train_labels_auto <- Auto_new[ran, 8]
# Testing labels
test_labels_auto <- Auto_new[-ran, 8]
```

Now we run our kNN function myKNN() for both regular kNN and for dnkNN and test different values of k and perform analysis.

2

For default kNN:

```
accuracyk_auto <- c()
for (k in 1:15) {
  classification_k <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=k, weighted=FALSE)
  accuracyk_auto[k] <- classification_k$accuracy
}
```

For distance weighted dwkNN:

```
accuracydwk_auto <- c()
for (k in 1:15) {
  classification_dwk <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=k, weighted=TRUE)
  accuracydwk_auto[k] <- classification_dwk$accuracy
}
```

3

For default kNN:

```
k_value <- 1:15
autodf_k <- data.frame(k_value, accuracyk_auto)
knitr::kable(autodf_k, "simple", col.names=c("k", "Accuracy"))
```

k	Accuracy
1	75.42373
2	75.42373
3	72.88136
4	72.03390
5	72.88136
6	72.03390
7	72.88136
8	72.88136
9	71.18644
10	73.72881
11	75.42373
12	73.72881
13	75.42373
14	73.72881
15	72.88136

For distance weighted dwkNN:

```
k_value <- 1:15
autodf_dwk <- data.frame(k_value, accuracydwk_auto)
knitr::kable(autodf_dwk, "simple", col.names=c("k", "Accuracy"))
```

k	Accuracy
1	75.42373
2	66.10169
3	65.25424

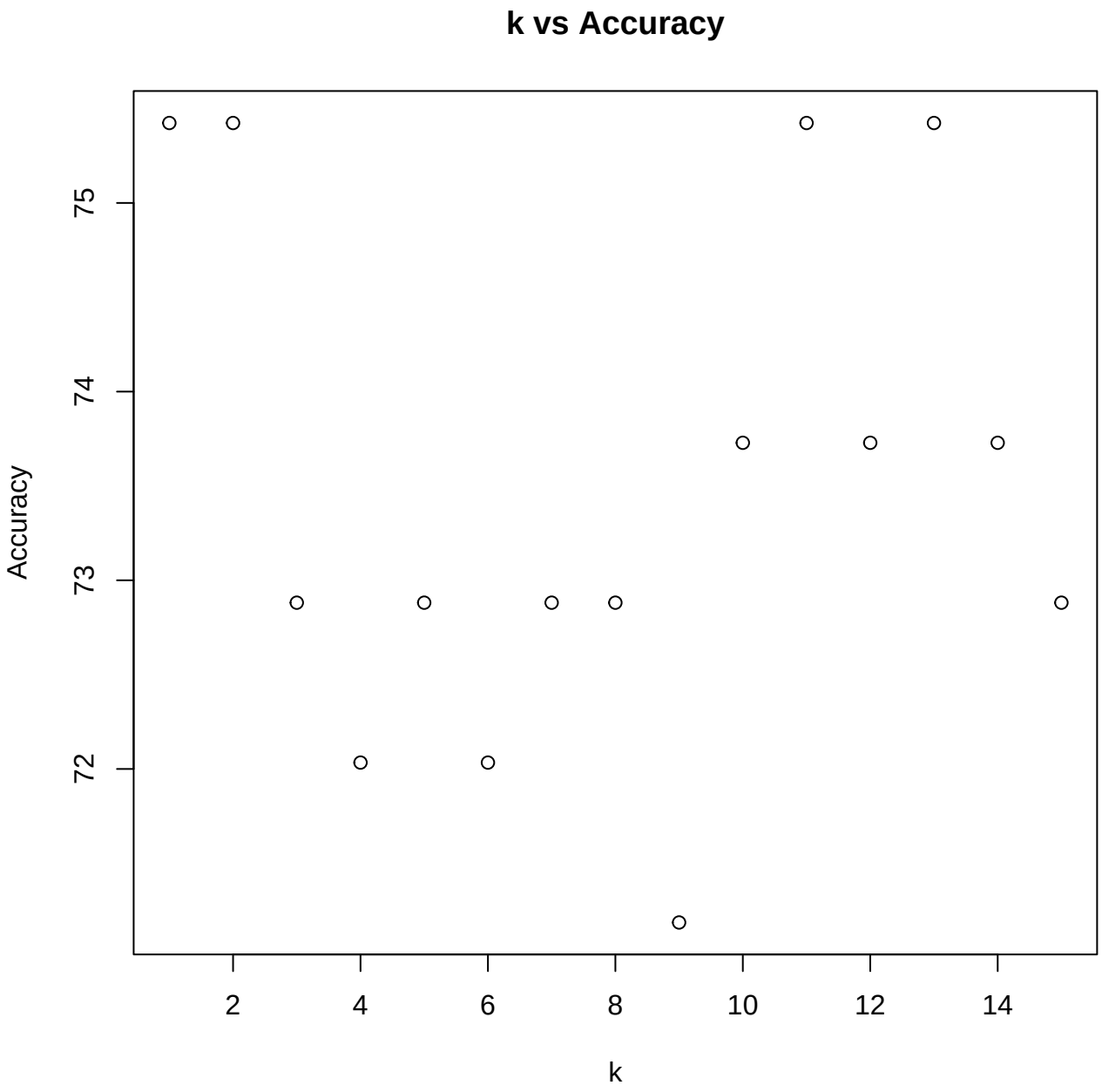
k	Accuracy
4	61.01695
5	61.86441
6	61.86441
7	61.01695
8	60.16949
9	60.16949
10	59.32203
11	59.32203
12	59.32203
13	59.32203
14	59.32203
15	59.32203

4

Here are plots of accuracy versus k, which will help us determine the best number of neighbors to use.

For default kNN:

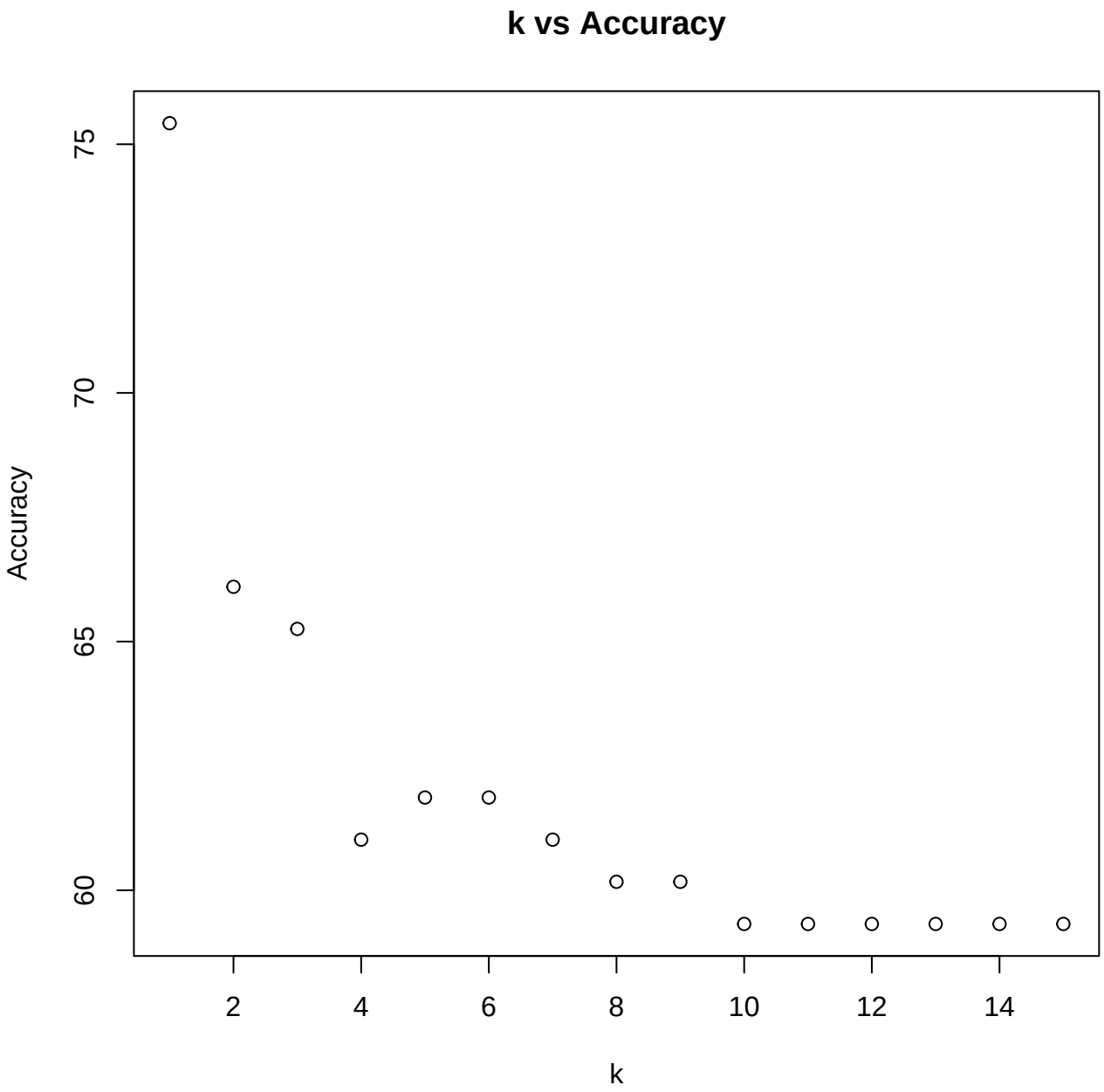
```
plot(autodf_k$k_value, autodf_k$accuracyk_auto, xlab='k', ylab='Accuracy', main='k vs Accuracy')
```



This plot shows that the best number of neighbors to use for default kNN is when k = 1, 2, 11, or 13.

For distance weighted dwkNN:

```
plot(autodf_dwk$k_value, autodf_dwk$accuracydwk_auto, xlab='k', ylab='Accuracy', main='k vs Accuracy')
```



This plot shows that the best number of neighbors to use for distance weighted dwkNN is when k = 1.

5

Now we will examine the confusion matrix.

For default kNN:

```
classification1 <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=1, weighted=FALSE)
classification2 <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=2, weighted=FALSE)
classification11 <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=11, weighted=FALSE)
classification13 <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=13, weighted=FALSE)
```

We have 4 values of k that have the same highest accuracy, so here is the confusion matrix of k = 1, 2, 11, and 13 in order.

```
classification1$confusion
```

```
          y_test
yhat       USA European Japanese
  USA       61        3        7
  European   2       11        3
  Japanese   7        7       17
```

```
classification2$confusion
```

```
          y_test
yhat       USA European Japanese
  USA       61        3        7
  European   2       11        3
  Japanese   7        7       17
```

```
classification11$confusion
```

```
          y_test
yhat       USA European Japanese
  USA       64        7       10
  European   4       11        3
  Japanese   2        3       14
```

```
classification13$confusion
```

```
          y_test
yhat       USA European Japanese
  USA       64        8        9
  European   4       10        3
  Japanese   2        3       15
```

For distance weighted dwkNN:

```
classification_w1 <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=1, weighted=TRUE)
```

We can see from the distance weighted dwkNN plot that k=1 has the highest accuracy. Here is its confusion matrix.

```
classification_w1$confusion
```

```
          y_test
yhat       USA European Japanese
  USA       61        3        7
  European   2       11        3
  Japanese   7        7       17
```

There are noticeable differences between kNN and dnkNN. The accuracy levels for each k value are very different from each other and each has different k values that model the data most accurately.

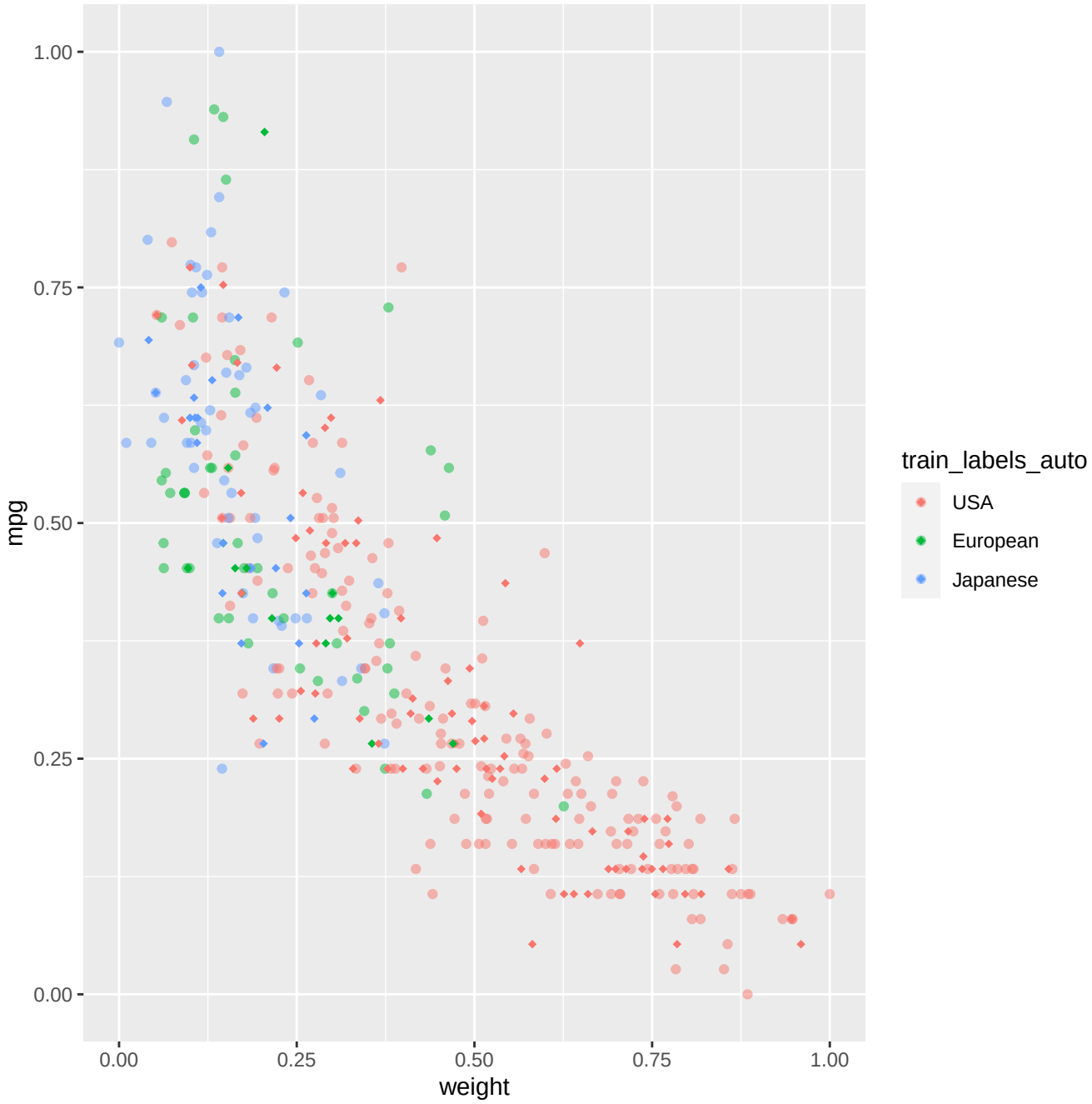
Now we will create a distinguished plot for k=5 and k=10 for both default kNN and distance weighted kNN.

Run default kNN for k=5 and k=10.

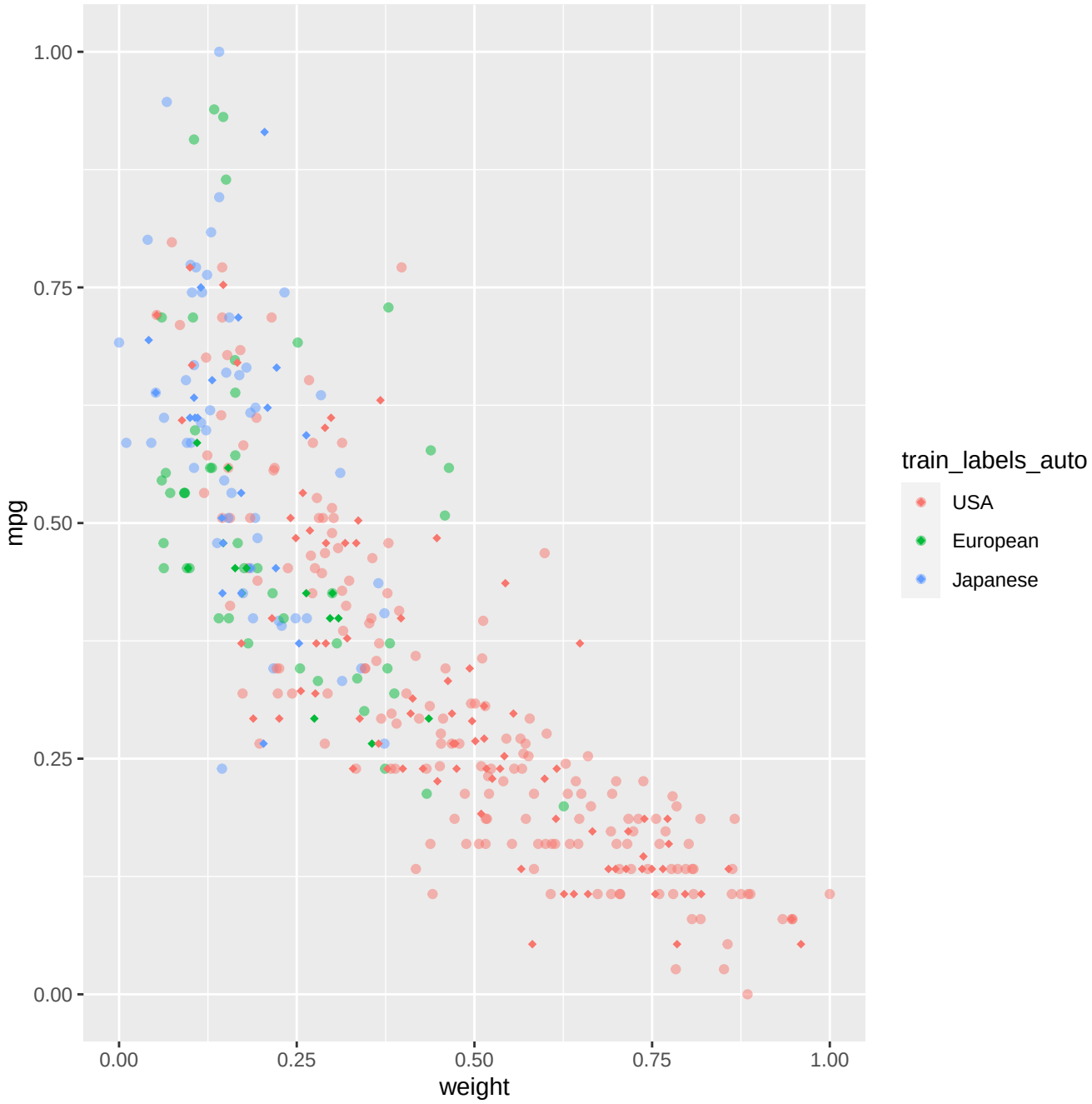
```
classification5 <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=5, weighted=FALSE)
classification10 <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=10, weighted=FALSE)
```

Plot for default kNN:

```
ggplot() + geom_point(data=train_auto, aes(x=weight, y=mpg, color=train_labels_auto), alpha=0.5) + geom_point(
```




```
ggplot() + geom_point(data=train_auto, aes(x=weight, y=mpg, color=train_labels_auto), alpha=0.5) + geom_point(
```

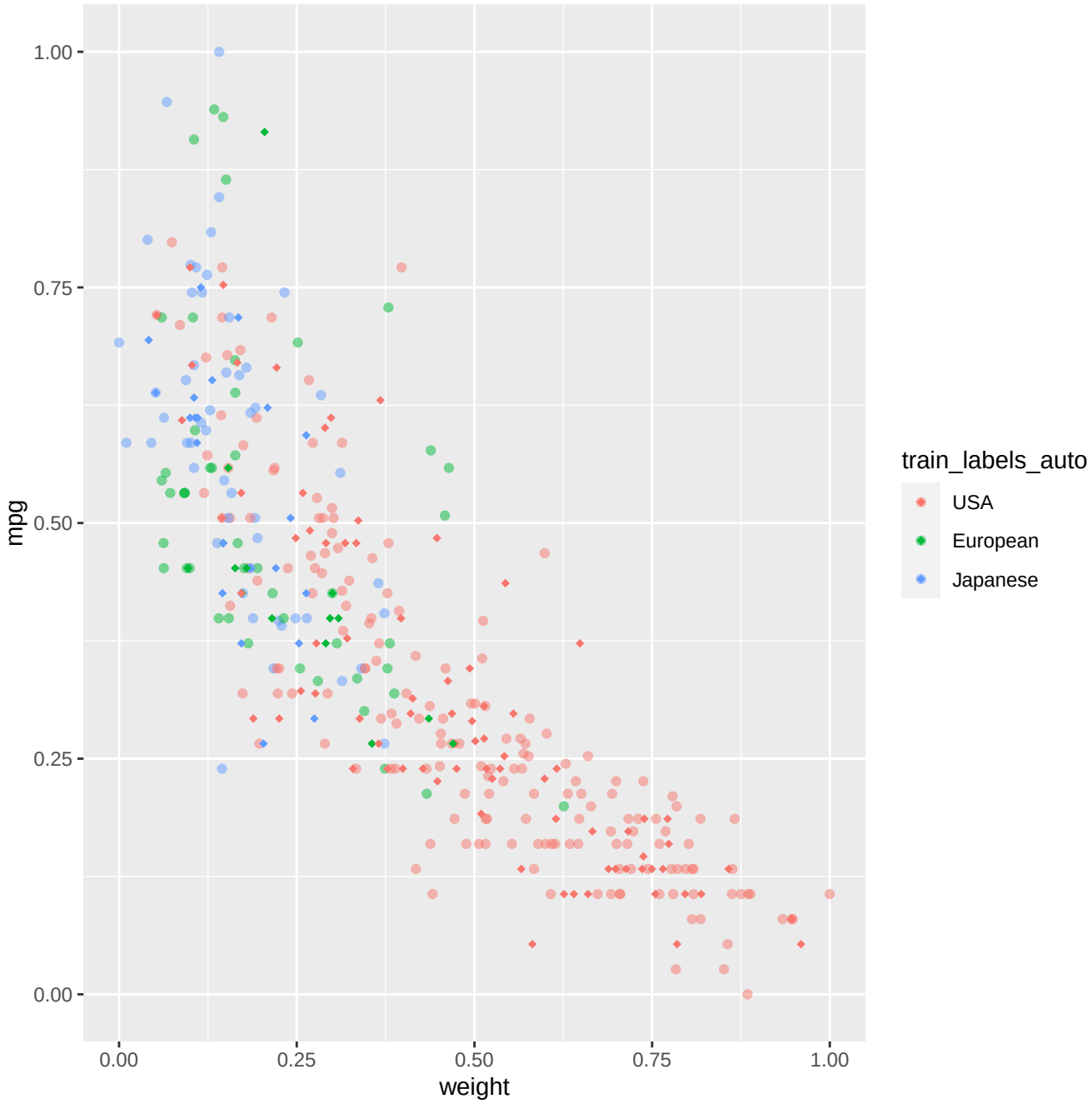


Run distance weighted dwkNN for k=5 and k=10.

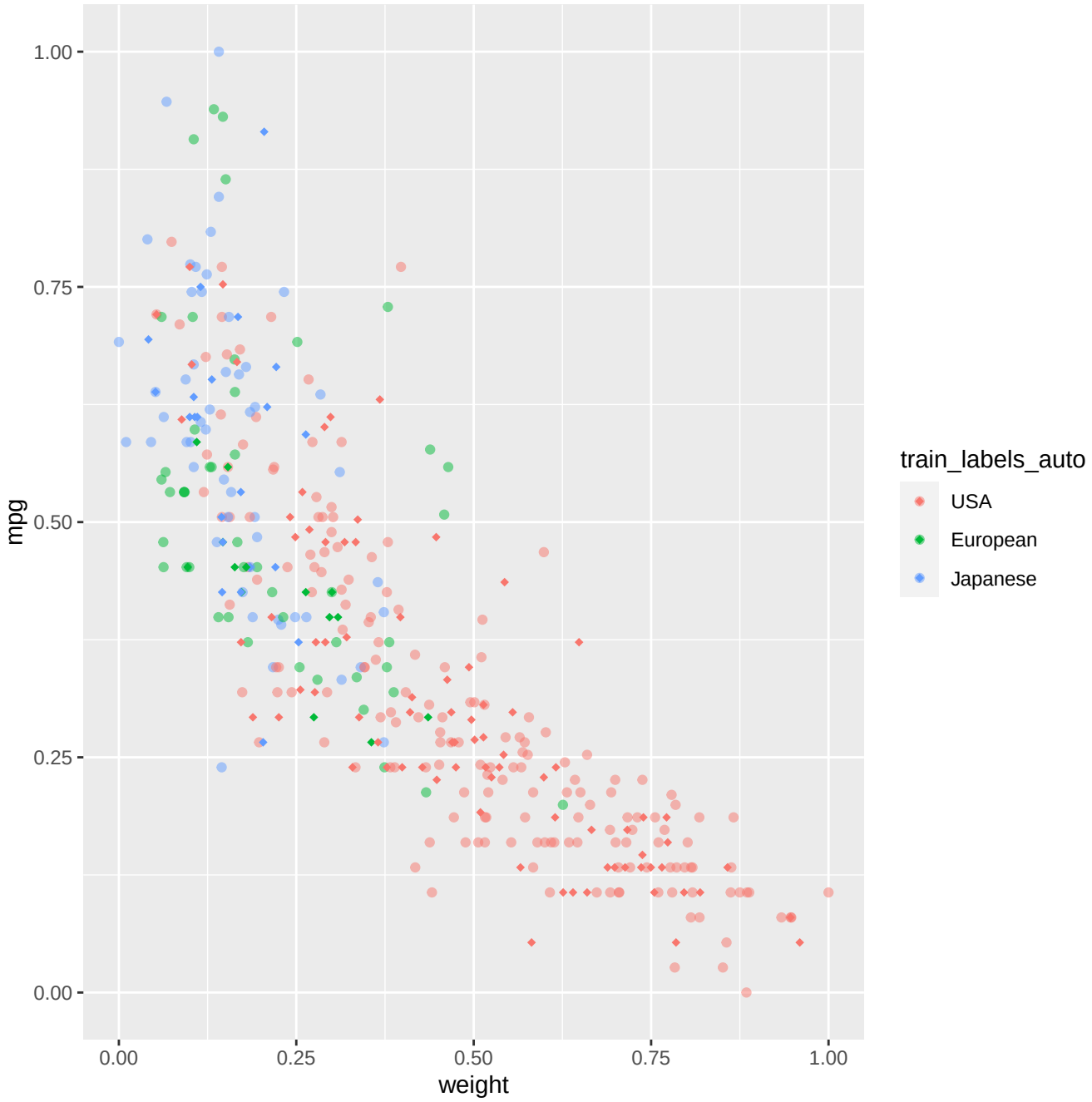
```
classification5w <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=5, weighted=FALSE)
classification10w <- mykNN(train_auto, test_auto, train_labels_auto, test_labels_auto, k=10, weighted=FALSE)
```

Plot for distance weighted dnkNN:

```
ggplot() + geom_point(data=train_auto, aes(x=weight, y=mpg, color=train_labels_auto), alpha=0.5) + geom_point(
```




```
ggplot() + geom_point(data=train_auto, aes(x=weight, y=mpg, color=train_labels_auto), alpha=0.5) + geom_point(
```



Problem 4

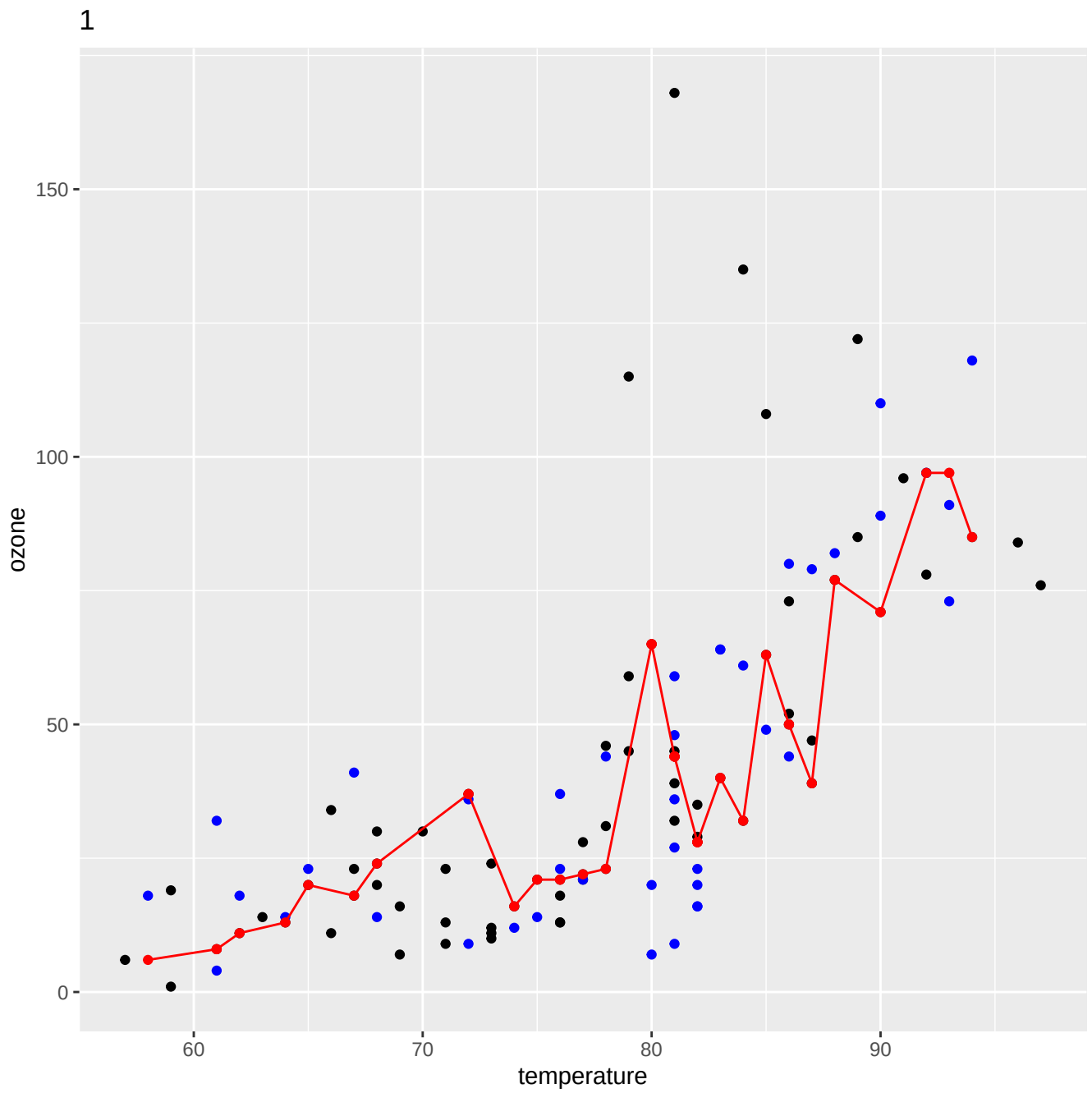
Here, we set up the ozone training and testing data. We select 70 random observations for the training data and the remaining 41 observations for our testing data. We use ozone as the response and temperature as the predictor.

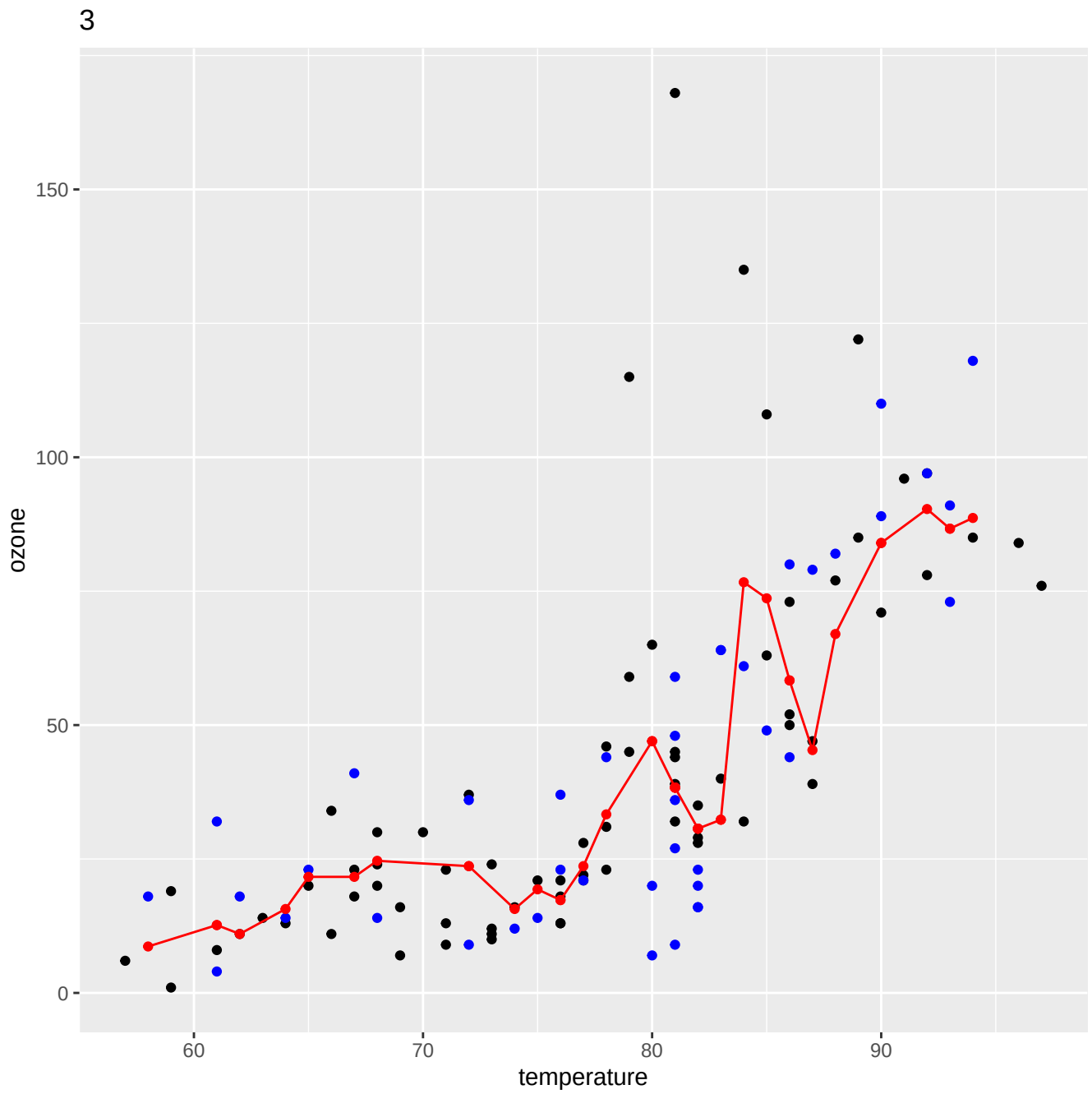
```
load("data/ozone.RData")
data("ozone")
set.seed(123)
ran_o <- sample(1:nrow(ozone), size=70)
train_ozone <- as.data.frame(ozone[ran_o, 3])
test_ozone <- as.data.frame(ozone[-ran_o, 3])
train_labels_ozone <- ozone[ran_o, 1]
test_labels_ozone <- ozone[-ran_o, 1]
```

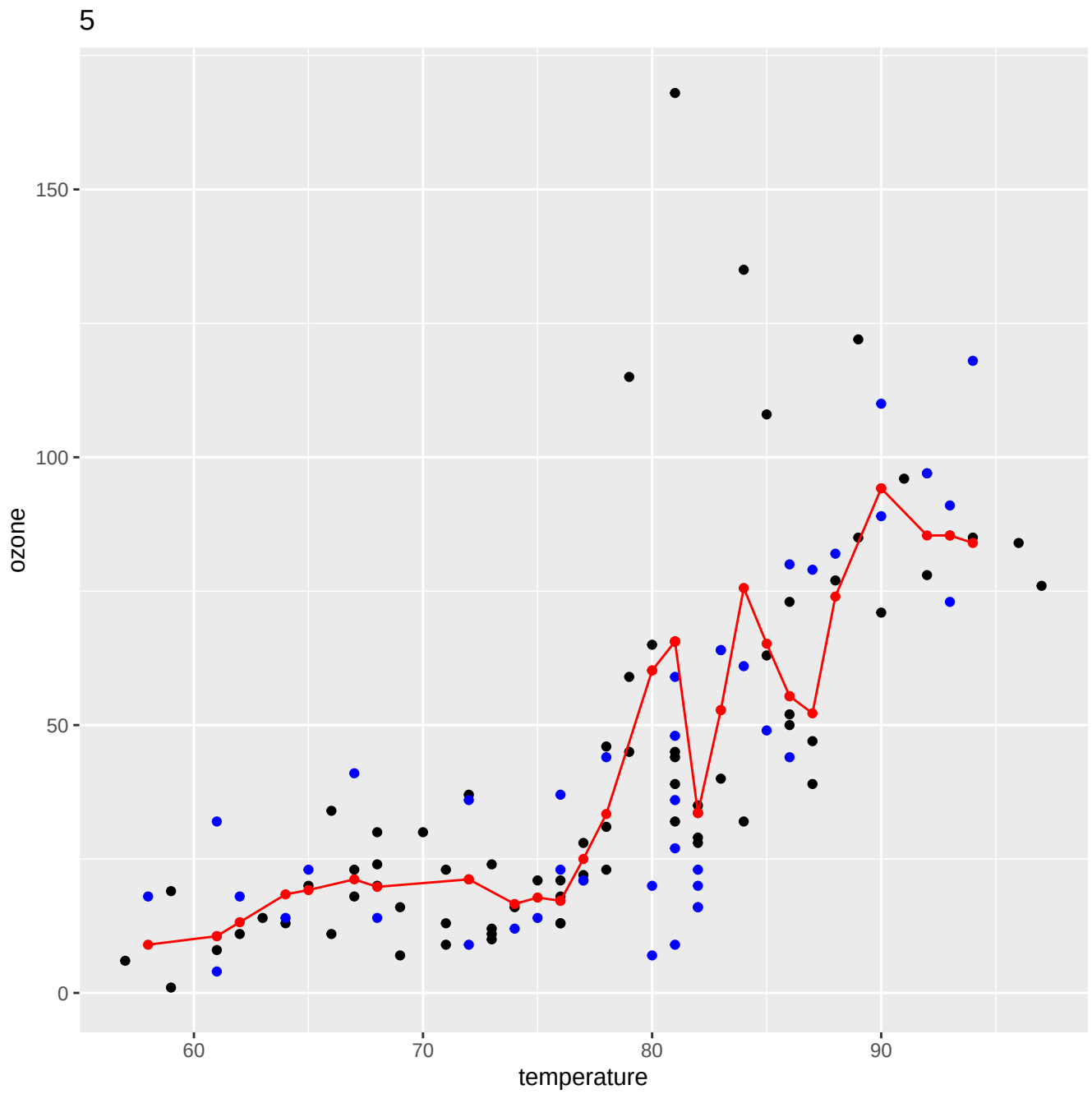
a

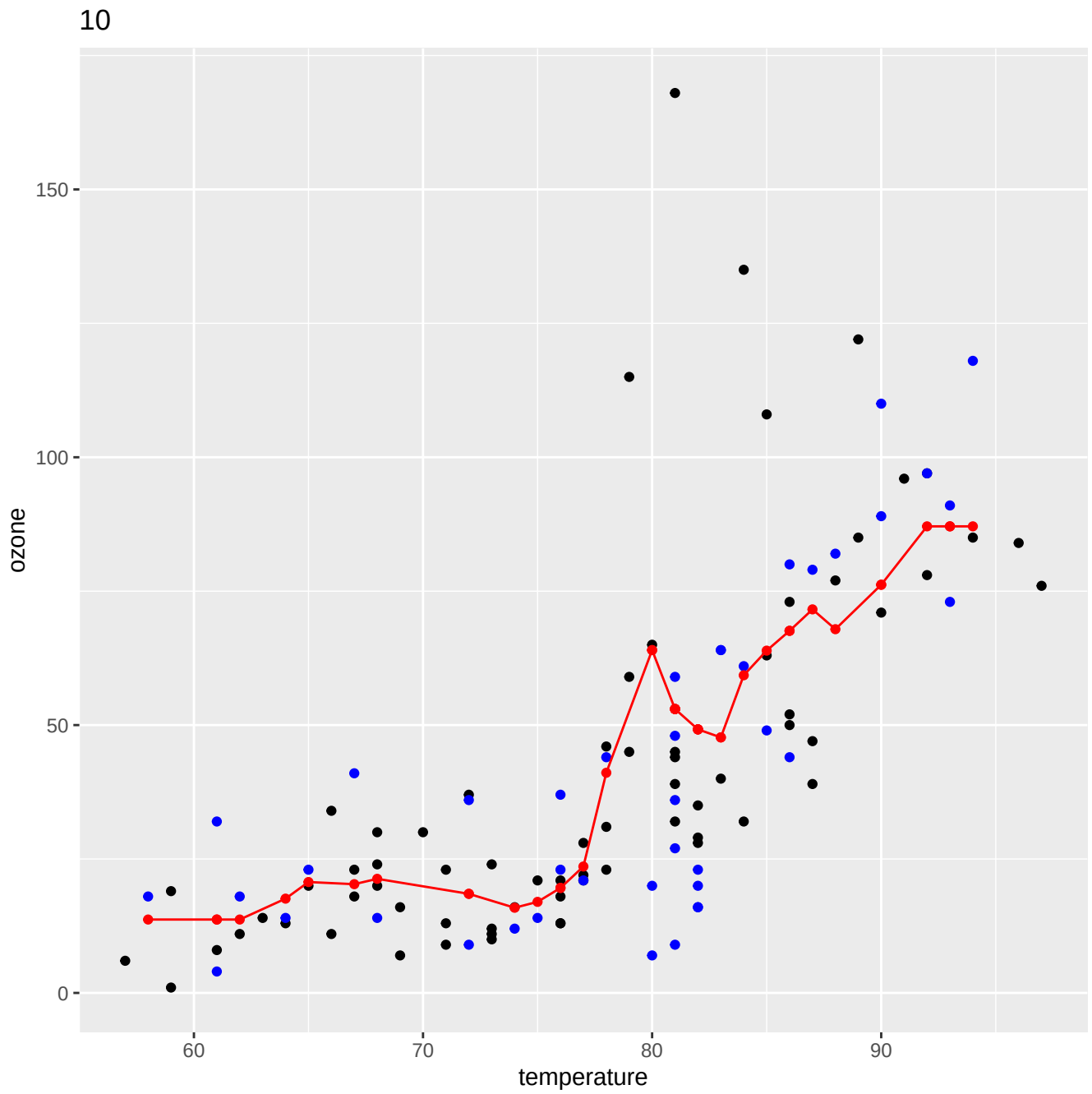
We use our function to do dwkNN for fitting a regression with ozone as the response and temperature as the predictor.

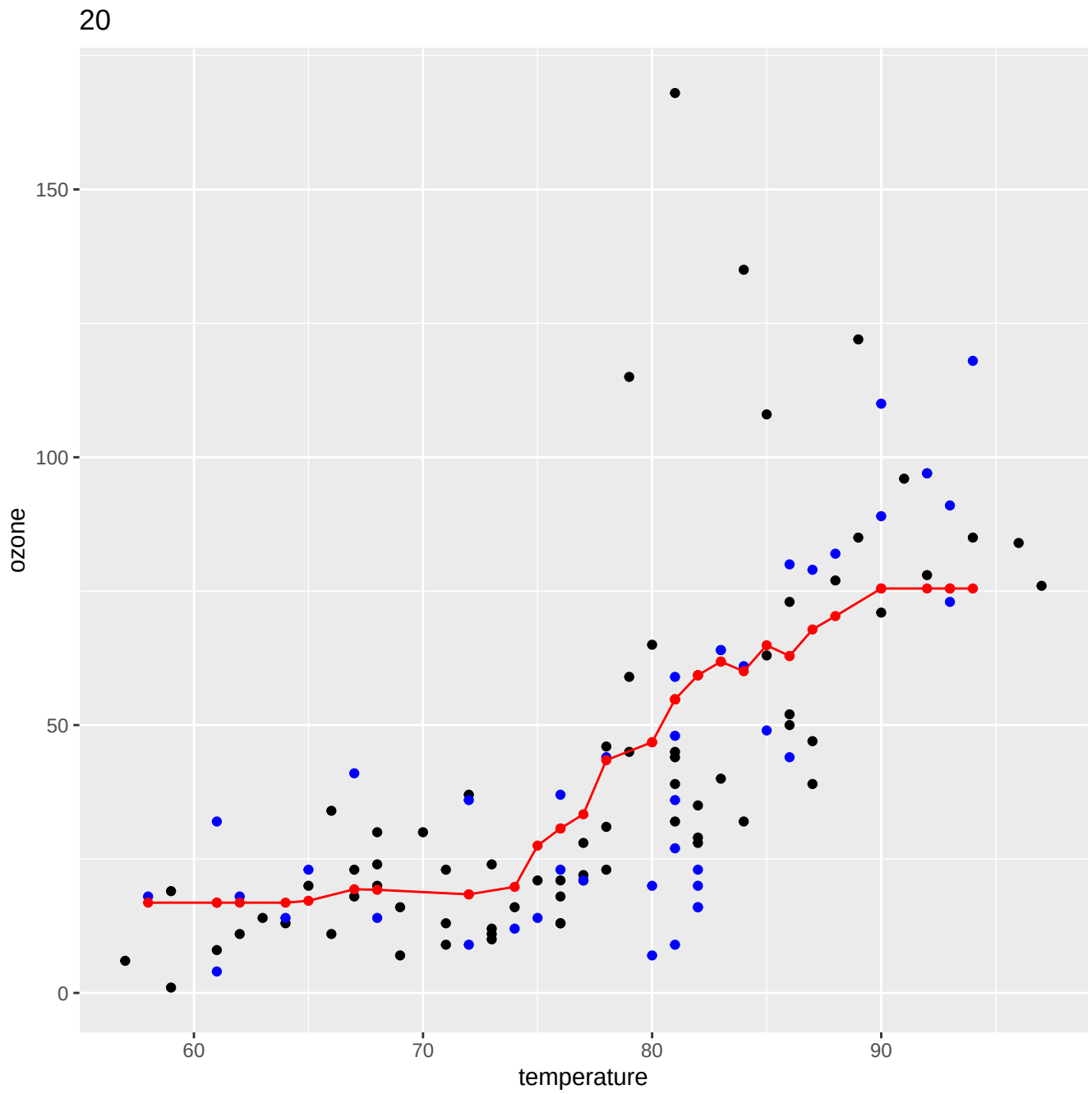
```
sse <- c()
k <- c(1,3,5,10,20)
for (i in k){
  dwknn_o <- mykNN(train_ozone, test_ozone, train_labels_ozone, test_labels_ozone, k=i, weighted=TRUE)
  # fitted regression points
  df <- data.frame(temperature = ozone[-ran_o,3], ozone = dwknn_o$yhat)
  # training data
  dtr <- data.frame(temperature = ozone[ran_o,3], ozone = ozone[ran_o,1])
  # testing data
  dt <- data.frame(temperature = ozone[-ran_o,3], ozone = ozone[-ran_o,1])
  # ggplot
  print(ggplot() + geom_point(data = dtr, aes(temperature,ozone), color = 'black') + geom_point(data = dt, aes
  sse <- c(sse,dwknn_o$SSE)
}
```


1
150
100
50
0
ozone
60
70
80
90
temperature


3
150
100
50
0
ozone
60
70
80
90
temperature


5
150
100
50
0
ozone
60
70
80
90
temperature


10
150
100
50
0
ozone
60
70
80
90
temperature



Here is the table of the results showing the SSE and value of k that was used.

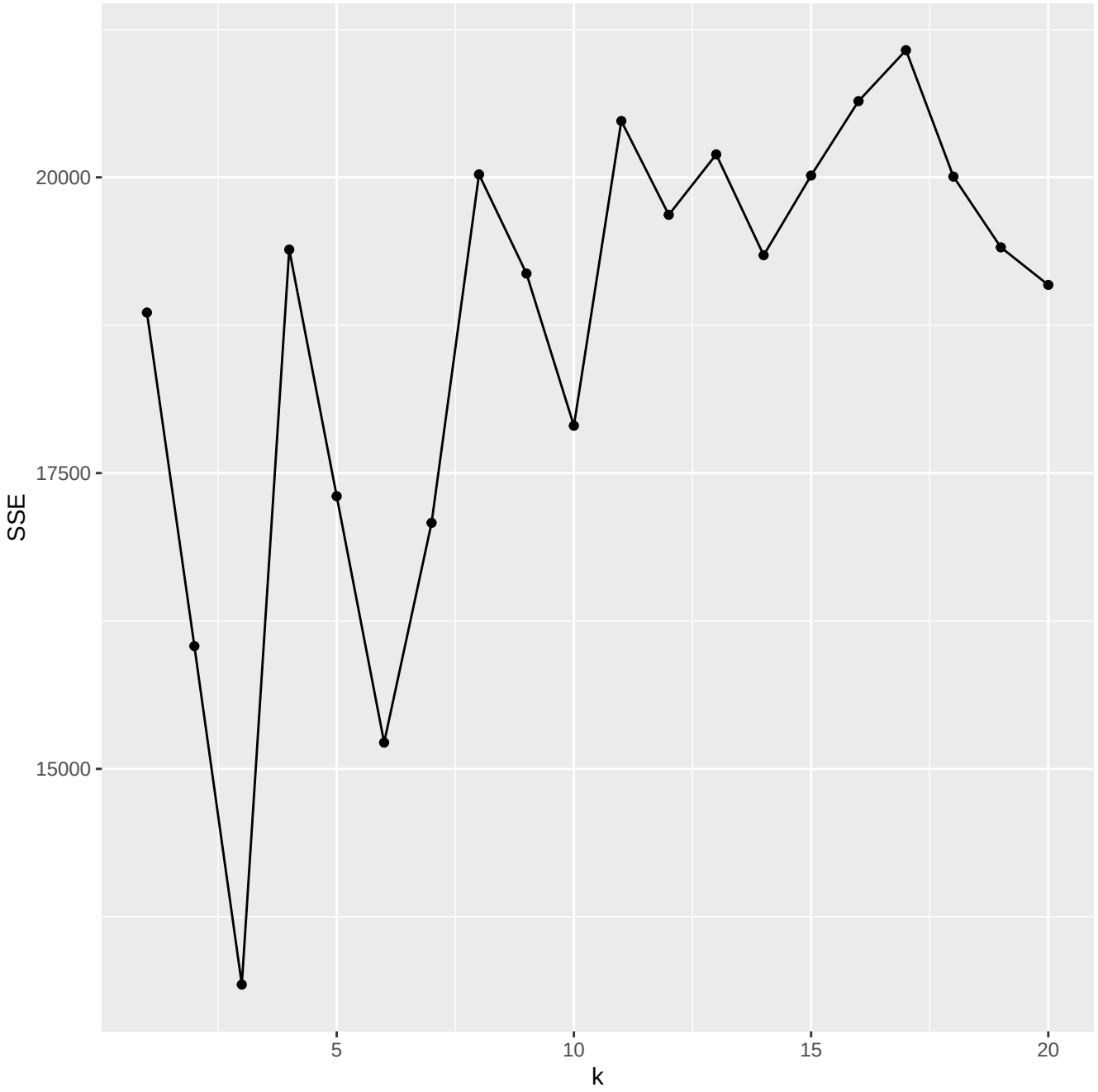
```
table <- data.frame(SSE = sse, k = k)
knitr::kable(table, "latex")
```

SSE	k
18857.00	1
13176.89	3
17304.88	5
17900.44	10
19090.51	20

When k = 3, the SSE is the smallest. Therefore, when choosing from $k = 1, 3, 5, 10, 20$, finding the nearest 3 points can help us find the most accurate regression for the dataset.

b

```
train_ozone_all <- as.data.frame(ozone[ran_o, c(2,3,4)])
test_ozone_all <- as.data.frame(ozone[-ran_o, c(2,3,4)])
train_labels_ozone <- ozone[ran_o, 1]
test_labels_ozone <- ozone[-ran_o, 1]
sse_all <- c()
k <- seq(1:20)
for (i in k){
  dwknn_o <- mykNN(train_ozone, test_ozone, train_labels_ozone, test_labels_ozone, k=i, weighted=TRUE)
  sse_all <- c(sse_all,dwknn_o$SSE)
}
df_all <- data.frame(SSE = sse_all, k = k)
ggplot(data = df_all, aes(k,SSE)) + geom_point() + geom_line()
```



When k becomes larger, the changing of SSE becomes smaller. When k = 3, the SSE is the smallest. Therefore, when choosing from $k = 1...20$, finding the nearest 3 points can help us find the most accurate regression for the data set.