C3M3_peer_review

March 27, 2022

1 C3M3: Peer Reviewed Assignment

1.0.1 Outline:

The objectives for this assignment:

1. Implement kernel smoothing in R and interpret the results.
2. Implement smoothing splines as an alternative to kernel estimation.
3. Implement and interpret the loess smoother in R.
4. Compare and contrast nonparametric smoothing methods.

General tips:

1. Read the questions carefully to understand what is being asked.
2. This work will be reviewed by another human, so make sure that you are clear and concise in what your explanations and answers.

[202]:
```
# Load Required Packages
library(ggplot2)
library(mgcv)
```

2 Problem 1: Advertising data

The following dataset containts measurements related to the impact of three advertising medias on sales of a product, P. The variables are:

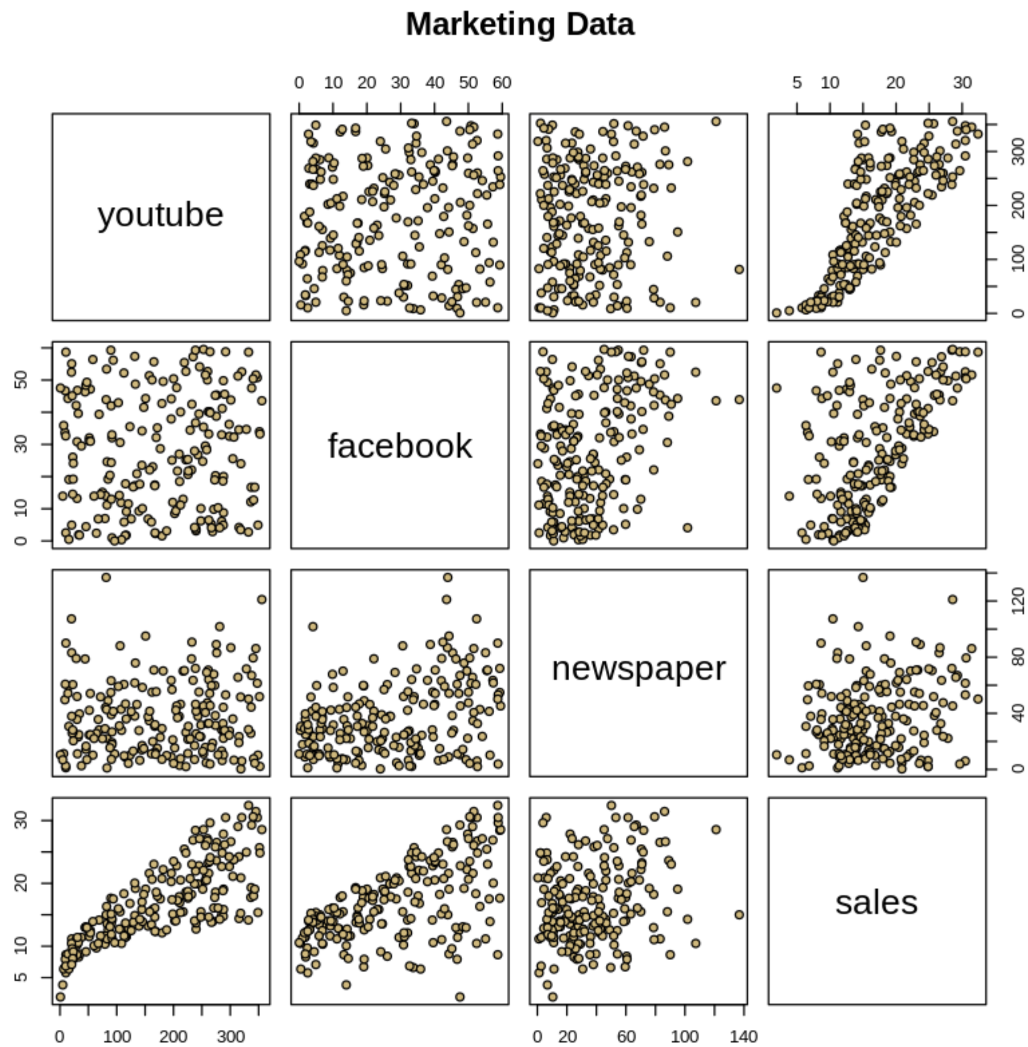
- `youtube`: the advertising budget allocated to YouTube. Measured in thousands of dollars;
- `facebook`: the advertising budget allocated to Facebook. Measured in thousands of dollars; and
- `newspaper`: the advertising budget allocated to a local newspaper. Measured in thousands of dollars.
- `sales`: the value in the i^{th} row of the sales column is a measurement of the sales (in thousands of units) for product P for company i.

The advertising data treat "a company selling product P" as the statistical unit, and "all companies selling product P" as the population. We assume that the $n = 200$ companies in the dataset were chosen at random from the population (a strong assumption!).

First, we load the data, plot it, and split it into a training set (`train_marketing`) and a test set (`test_marketing`).

[203]:
```r
# Load in the data
marketing = read.csv("marketing.txt", sep="")
summary(marketing)
pairs(marketing, main = "Marketing Data", pch = 21,
      bg = c("#CFB87C"))
```

```
    youtube          facebook        newspaper          sales      
 Min.   :  0.84   Min.   : 0.00   Min.   :  0.36   Min.   : 1.92  
 1st Qu.: 89.25   1st Qu.:11.97   1st Qu.: 15.30   1st Qu.:12.45  
 Median :179.70   Median :27.48   Median : 30.90   Median :15.48  
 Mean   :176.45   Mean   :27.92   Mean   : 36.66   Mean   :16.83  
 3rd Qu.:262.59   3rd Qu.:43.83   3rd Qu.: 54.12   3rd Qu.:20.88  
 Max.   :355.68   Max.   :59.52   Max.   :136.80   Max.   :32.40  
```




```
[204]: set.seed(1771) #set the random number generator seed.
n = floor(0.8 * nrow(marketing)) #find the number corresponding to 80% of the␣
 ↪data
index = sample(seq_len(nrow(marketing)), size = n) #randomly sample indicies to␣
 ↪be included in the training set

train_marketing = marketing[index, ] #set the training set to be the randomly␣
 ↪sampled rows of the dataframe
test_marketing = marketing[-index, ] #set the testing set to be the remaining␣
 ↪rows
dim(test_marketing) #check the dimensions
dim(train_marketing) #check the dimensions
```

1. 40 2. 4

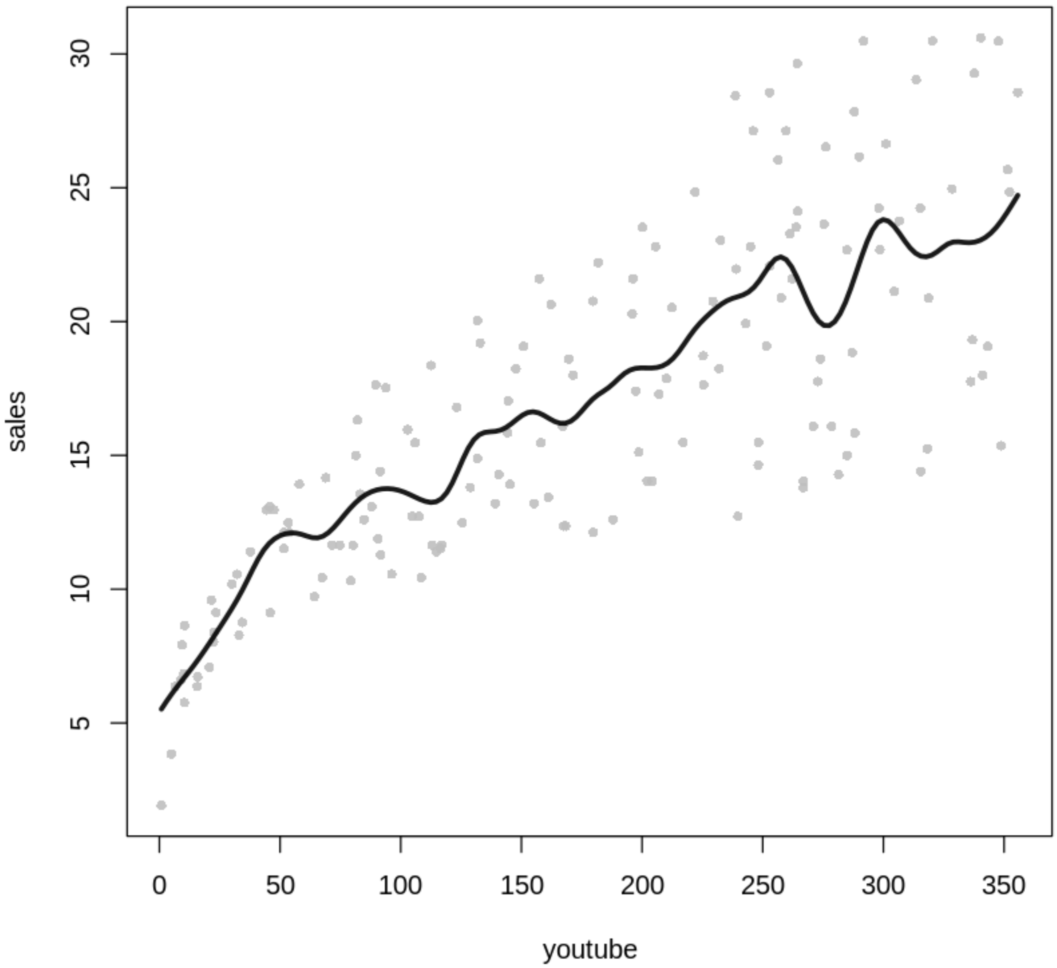
1. 160 2. 4

1.(a) Working with nonlinearity: Kernel regression

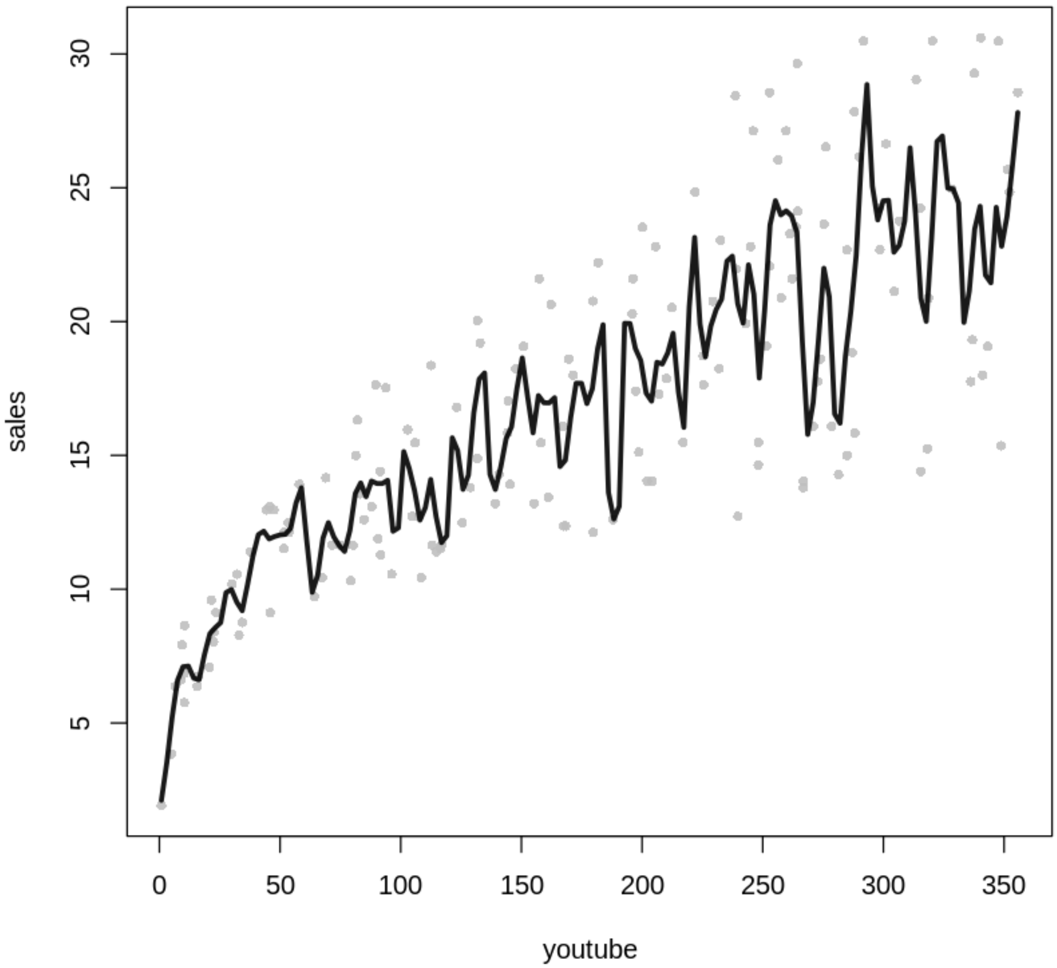
Note that the relationship between `sales` and `youtube` is nonlinear. This was a problem for us back in the first course in this specialization, when we modeled the data as if it were linear. For now, let's just focus on the relationship between `sales` and `youtube`, omitting the other variables (future lessons on generalized additive models will allow us to bring back other predictors).

Using the `train_marketing` set, plot `sales` (response) against `youtube` (predictor), and then fit and overlay a kernel regression. Experiment with the bandwidth parameter until the smooth looks appropriate, or comment why no bandwidth is ideal. Justify your answer.

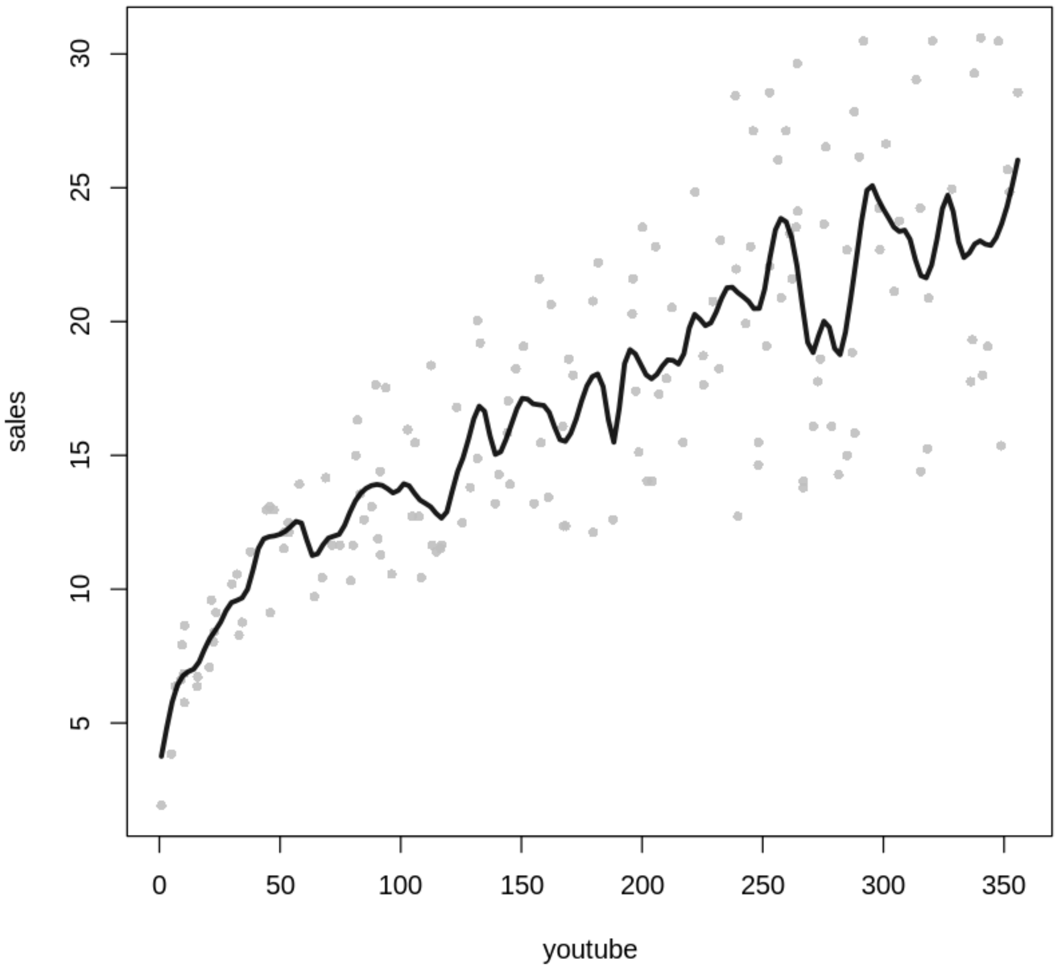
[205]:
```
z = ksmooth(train_marketing$youtube, train_marketing$sales, k="normal", 20)
plot(sales ~ youtube, pch=16, data=train_marketing, cex=0.8, col =␣
 ↪alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```



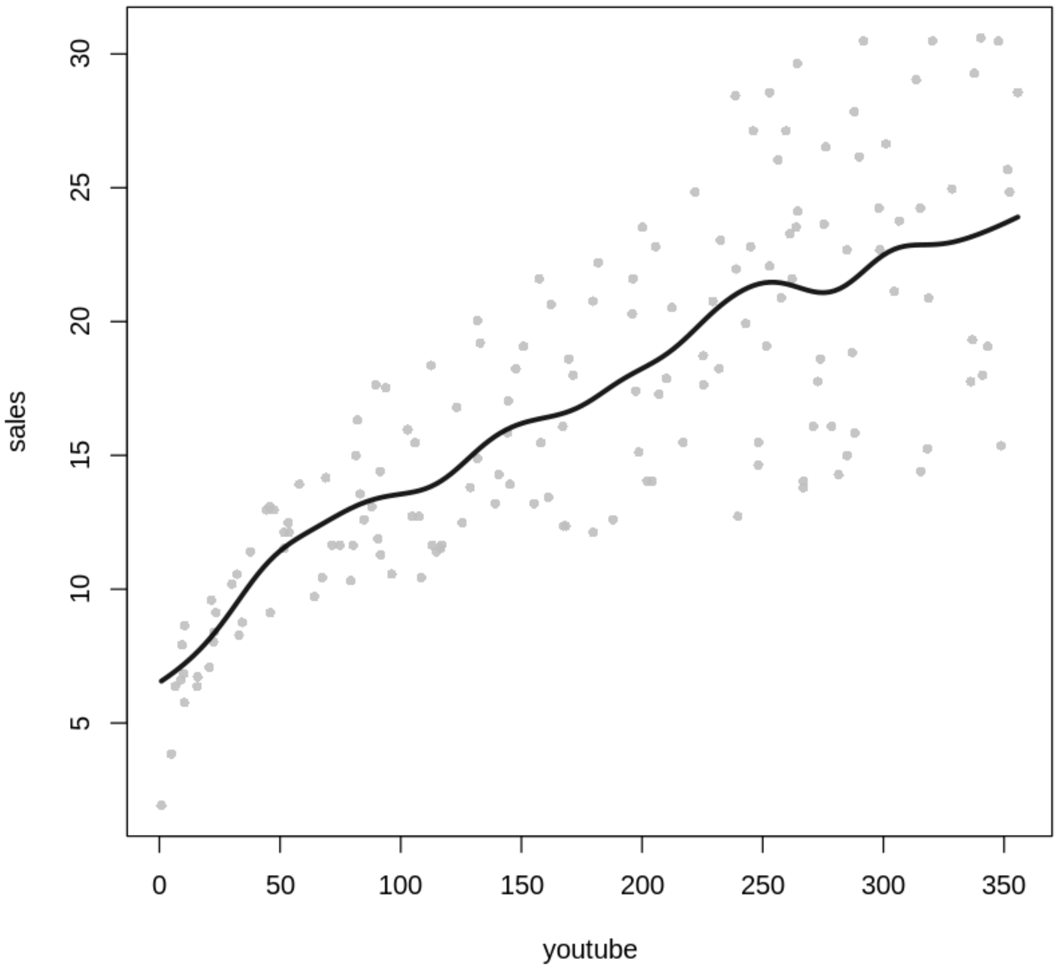

```
[206]: z = ksmooth(train_marketing$youtube, train_marketing$sales, k="normal", 5)
plot(sales ~ youtube, pch=16, data=train_marketing, cex=0.8, col =␣
 ↪alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```



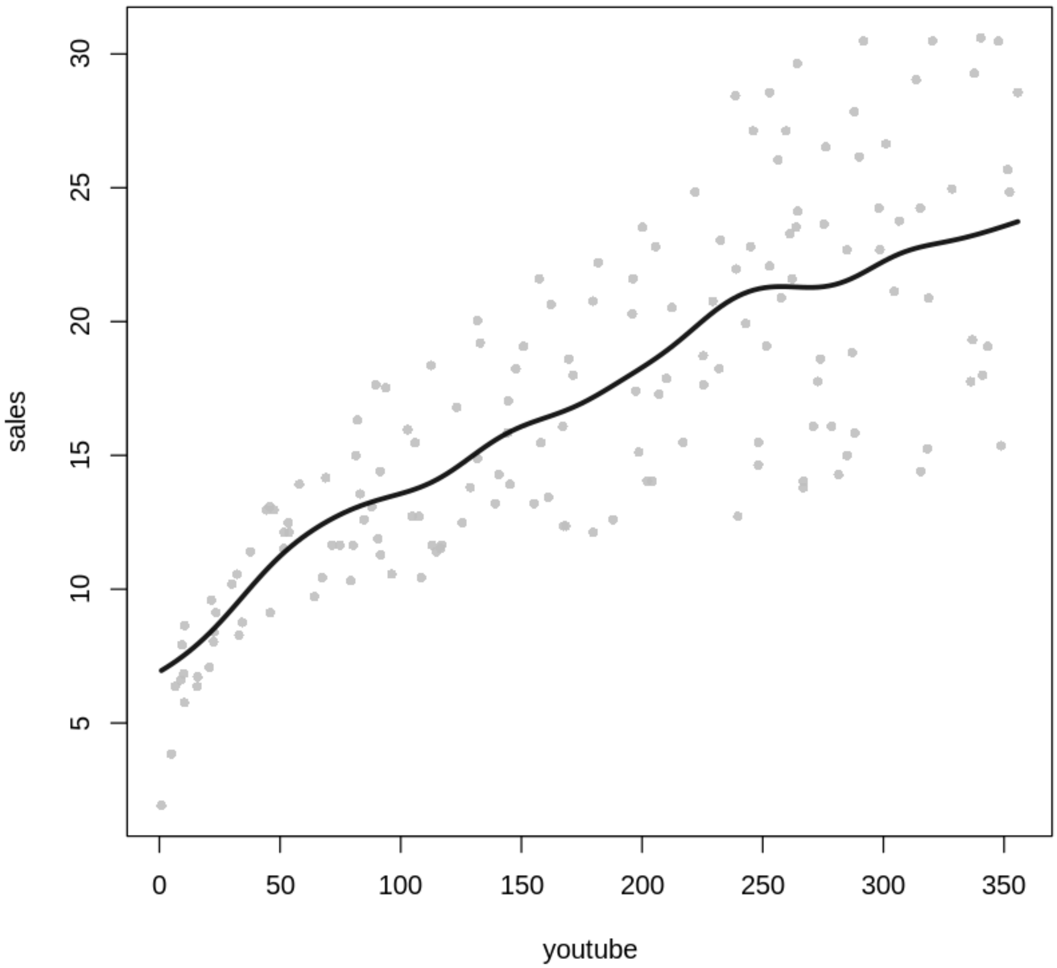
[207]:
```
z = ksmooth(train_marketing$youtube, train_marketing$sales, k="normal", 10)
plot(sales ~ youtube, pch=16, data=train_marketing, cex=0.8, col =
  alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```



[208]:
```
z = ksmooth(train_marketing$youtube, train_marketing$sales, k="normal", 40)
plot(sales ~ youtube, pch=16, data=train_marketing, cex=0.8, col =␣
 ↪alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```



[209]:
```
z = ksmooth(train_marketing$youtube, train_marketing$sales, k="normal", 50)
plot(sales ~ youtube, pch=16, data=train_marketing, cex=0.8, col =␣
↪alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```

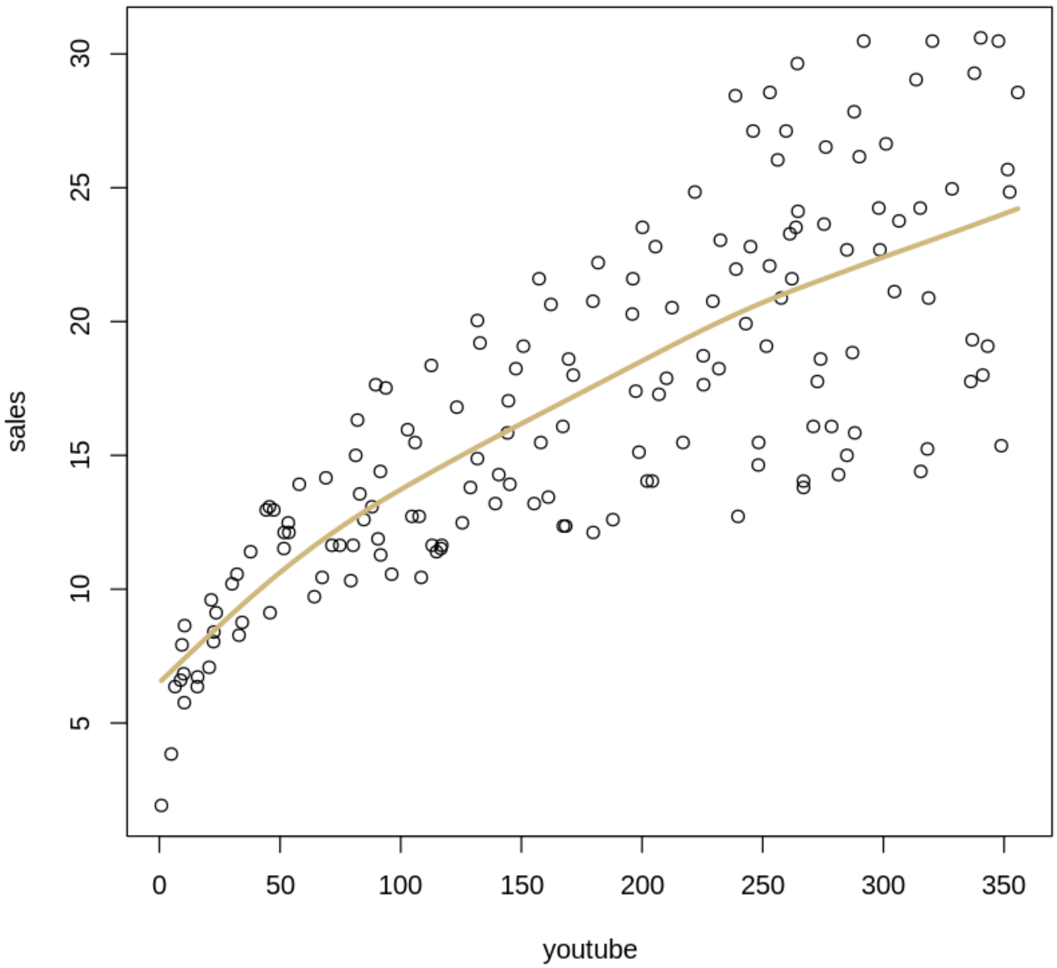


None of the bandwiths seem to fit the nonlinearity. The slope of the plot is not accurately shown as there is too much bumpiness that is not represented in the plot.

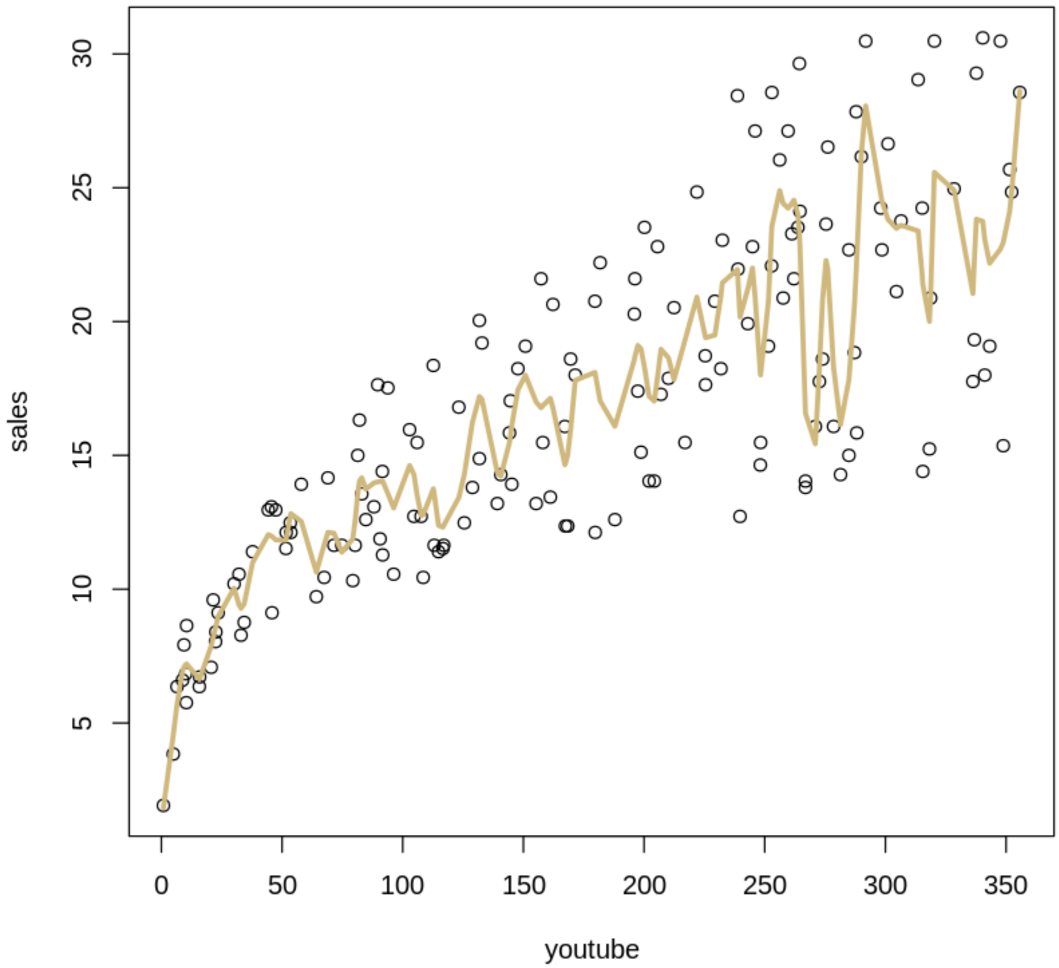
1.(b) Working with nonlinearity: Smoothing spline regression

Again, using the `train_marketing` set, plot `sales` (response) against `youtube` (predictor). This time, fit and overlay a smoothing spline regression model. Experiment with the smoothing parameter until the smooth looks appropriate. Explain why it's appropriate and justify your answer.

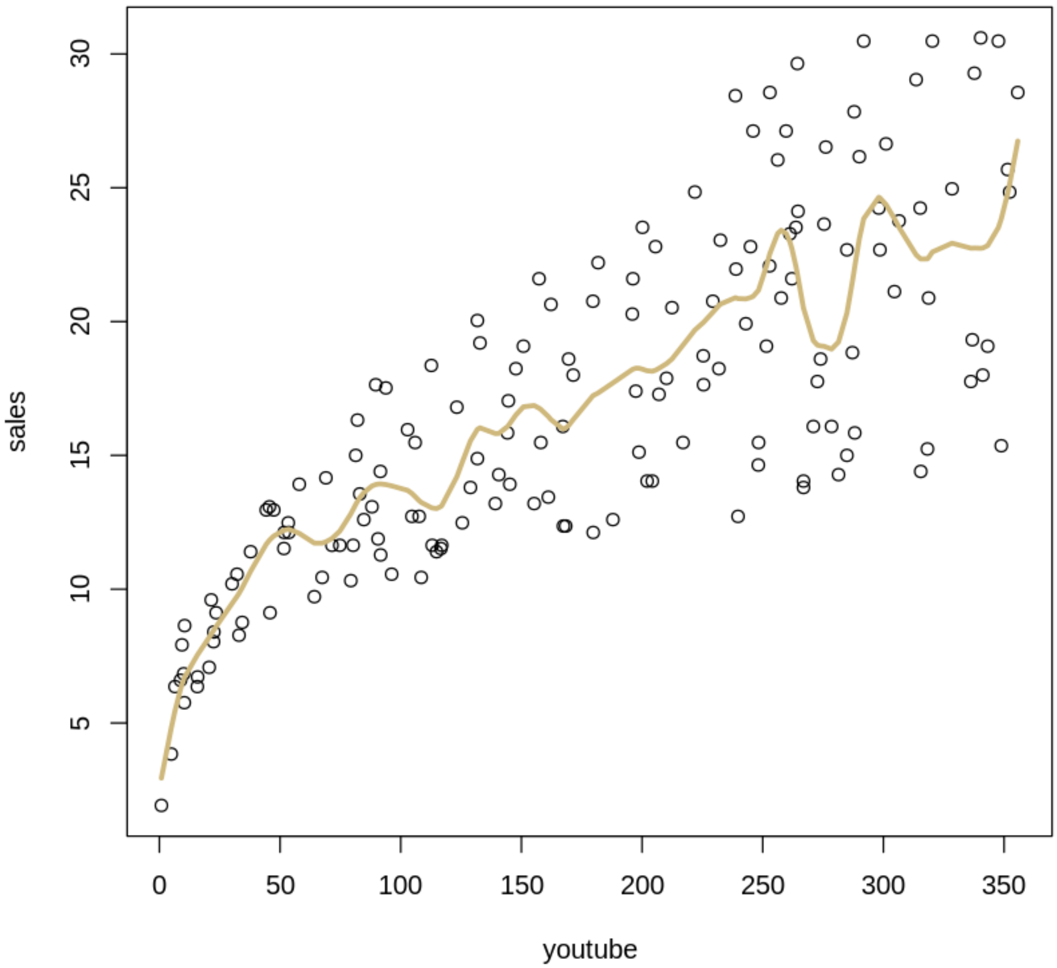
[210]:
```
lm_smooth = smooth.spline(train_marketing$youtube, train_marketing$sales,␣
 ↪spar=1)
plot(sales ~ youtube, data=train_marketing)
lines(lm_smooth, col="#CFB87C", lwd=3)
```



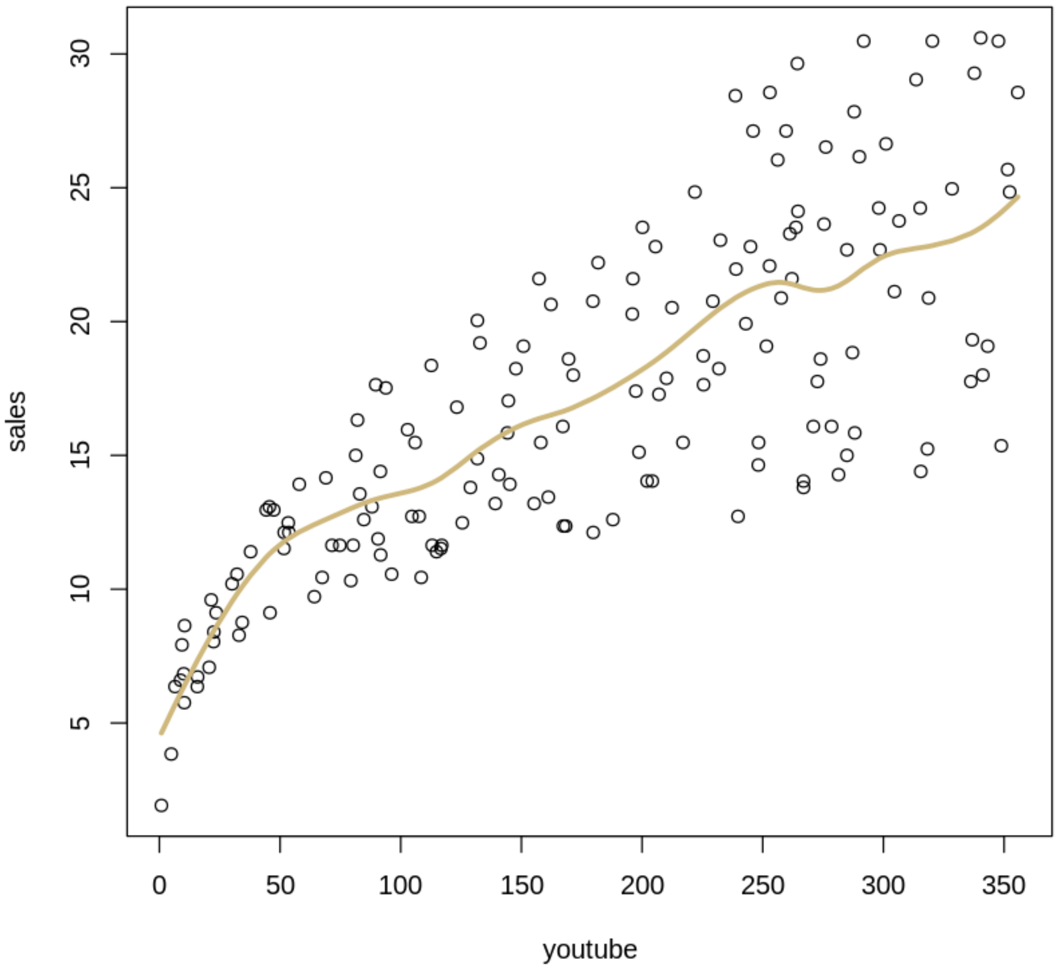
[211]:
```
lm_smooth = smooth.spline(train_marketing$youtube, train_marketing$sales,
    spar=0.25)
plot(sales ~ youtube, data=train_marketing)
lines(lm_smooth, col="#CFB87C", lwd=3)
```



[212]:
```
lm_smooth = smooth.spline(train_marketing$youtube, train_marketing$sales,␣
↪spar=0.5)
plot(sales ~ youtube, data=train_marketing)
lines(lm_smooth, col="#CFB87C", lwd=3)
```



[213]:
```
lm_smooth = smooth.spline(train_marketing$youtube, train_marketing$sales,
    spar=0.75)
plot(sales ~ youtube, data=train_marketing)
lines(lm_smooth, col="#CFB87C", lwd=3)
```



Having `spar=0.75` represents the data a bit better than the other models. The initial steepnes of the plot is picked up. Although there is a bit of inaccurate roughness (youtube betwee 250 and 300), the line represents the appropriate curvature throughout most of the plot.

1.(c) Working with nonlinearity: Loess

Again, using the `train_marketing` set, plot `sales` (response) against `youtube` (predictor). This time, fit and overlay a loess regression model. You can use the `loess()` function in a similar way as the `lm()` function. Experiment with the smoothing parameter (`span` in the `geom_smooth()` function) until the smooth looks appropriate. Explain why it's appropriate and justify your answer.

[214]:
```
#Issue with using plot, used ggplot instead

ggplot(train_marketing, aes(x=youtube, y=sales)) +
```

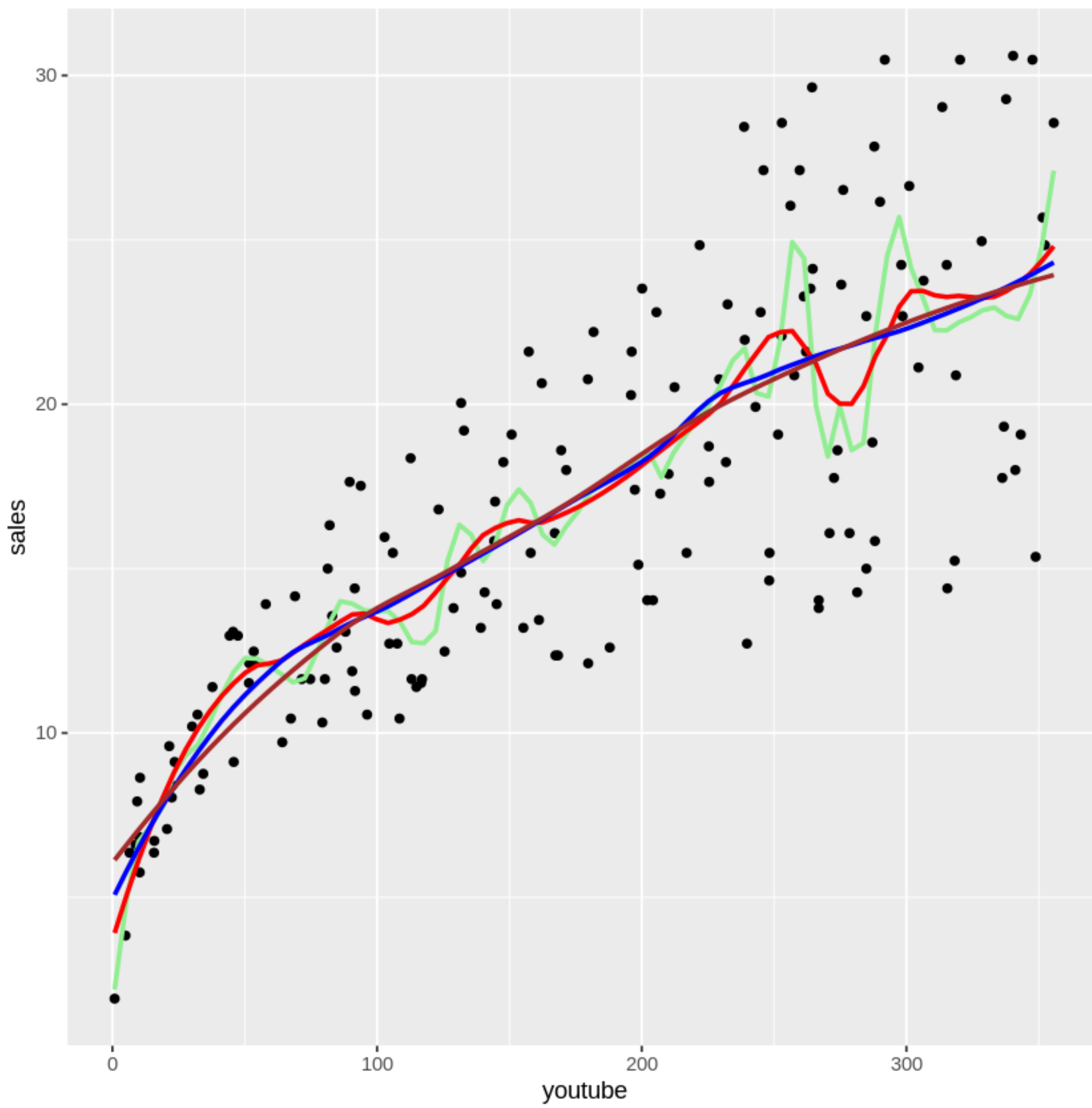
```
    geom_point() +
    geom_smooth(span=0.1, se=FALSE, color="light green") +
    geom_smooth(span=0.25, se=FALSE, color="red") +
    geom_smooth(span=0.50, se=FALSE, color = "blue") +
    geom_smooth(span=0.75, se=FALSE, color="brown")
```

```
`geom_smooth()` using method = 'loess' and formula 'y ~ x'

`geom_smooth()` using method = 'loess' and formula 'y ~ x'

`geom_smooth()` using method = 'loess' and formula 'y ~ x'

`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



[215]:
```
lm_loess = loess(sales ~ youtube, data= train_marketing, span=0.5)
```

Both `span = 0.5`(blue) and `span = 0.75`(brown) seems to represent the data well. However, `span = 0.5` seems to fit the initial curvature (0-100) a bit better. It continues to fit well as you go through the higher values of youtube.

1.(d) A prediction metric

Compare the models using the mean squared prediction error (MSPE) on the `test_marketing` dataset. That is, calculate the MSPE for your kernel regression, smoothing spline regression, and loess model, and identify which model is best in terms of this metric.

Remember, the MSPE is given by

$$MSPE = \frac{1}{k}\sum_{i=1}^{k}(y_i^\star - \widehat{y}_i^\star)^2$$

where $y_i^\star$ are the observed response values in the test set and $\widehat{y}_i^\star$ are the predicted values for the test set (using the model fit on the training set).

[216]:
```
#Kernel Regression
kreg_pred = ksmooth(train_marketing$youtube, train_marketing$sales,␣
↪kernel="normal",
                        40, x.points=test_marketing$youtube)
mean((test_marketing$sales - kreg_pred$y)^2)

#Kernel Smoothing
ksmooth_pred = predict(lm_smooth, x=test_marketing$youtube)
mean((test_marketing$sales - ksmooth_pred$y)^2)

#Loess
loess_pred = predict(lm_loess, test_marketing$youtube)
mean((test_marketing$sales - loess_pred)^2)
```

64.458818245407

18.1211942436849

18.1148323815482

The loess model has the smallest MSPE

3 Problem 2: Simulations!

Simulate data (one predictor and one response) with your own nonlinear relationship. Provide an explanation of how you generated the data. Then answer the questions above (1.(a) - 1.(d)) using your simulated data.

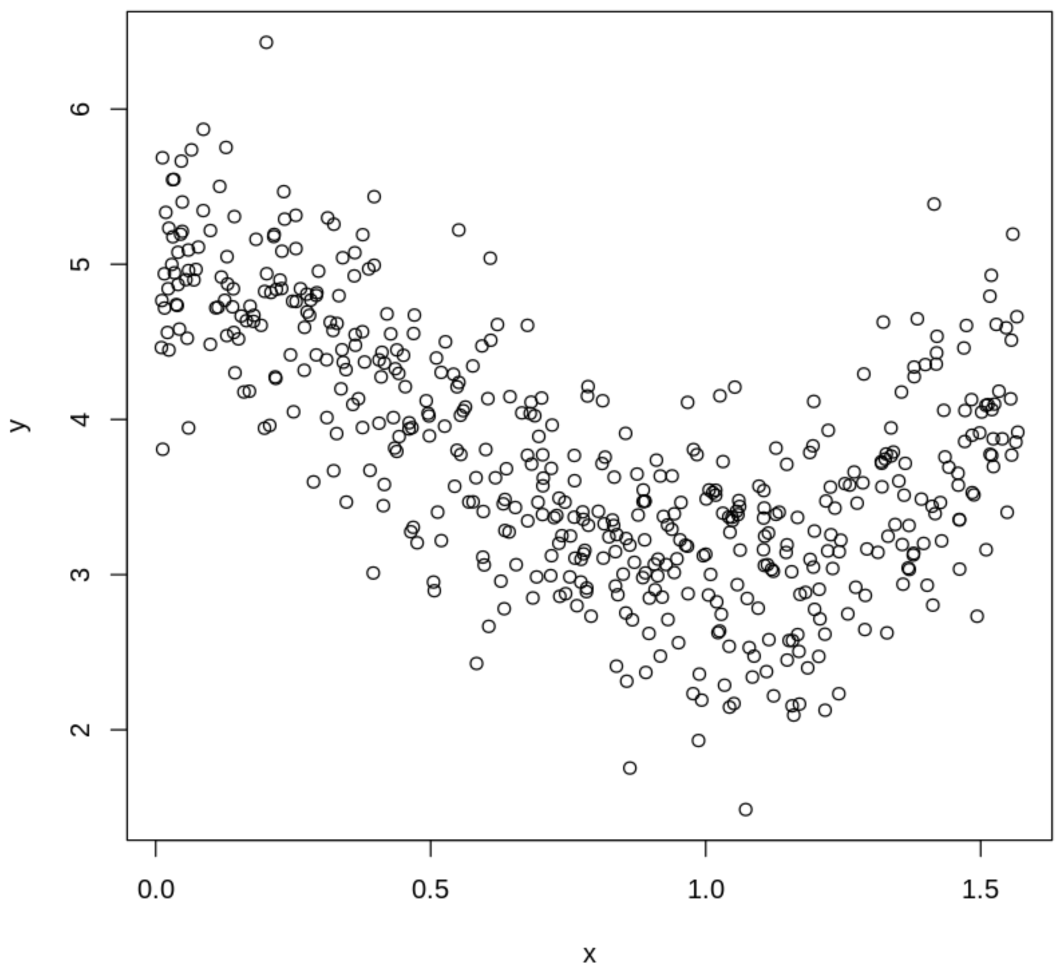
[217]:
```
#simulated data

n = 500
x = runif(n, 0, pi/2)
y = cos(pi*x) + rnorm(n, 0, 0.5) + 4
df_sim = data.frame(x = x, y = y)
head(df_sim)
```

A data.frame: 6 × 2

	x <dbl>	y <dbl>
1	1.012967	3.528781
2	1.426630	3.464510
3	1.287504	4.292812
4	1.018202	3.544160
5	0.453815	4.210996
6	1.420508	4.535766

[218]:
```
plot(y ~ x, data=df_sim)
```



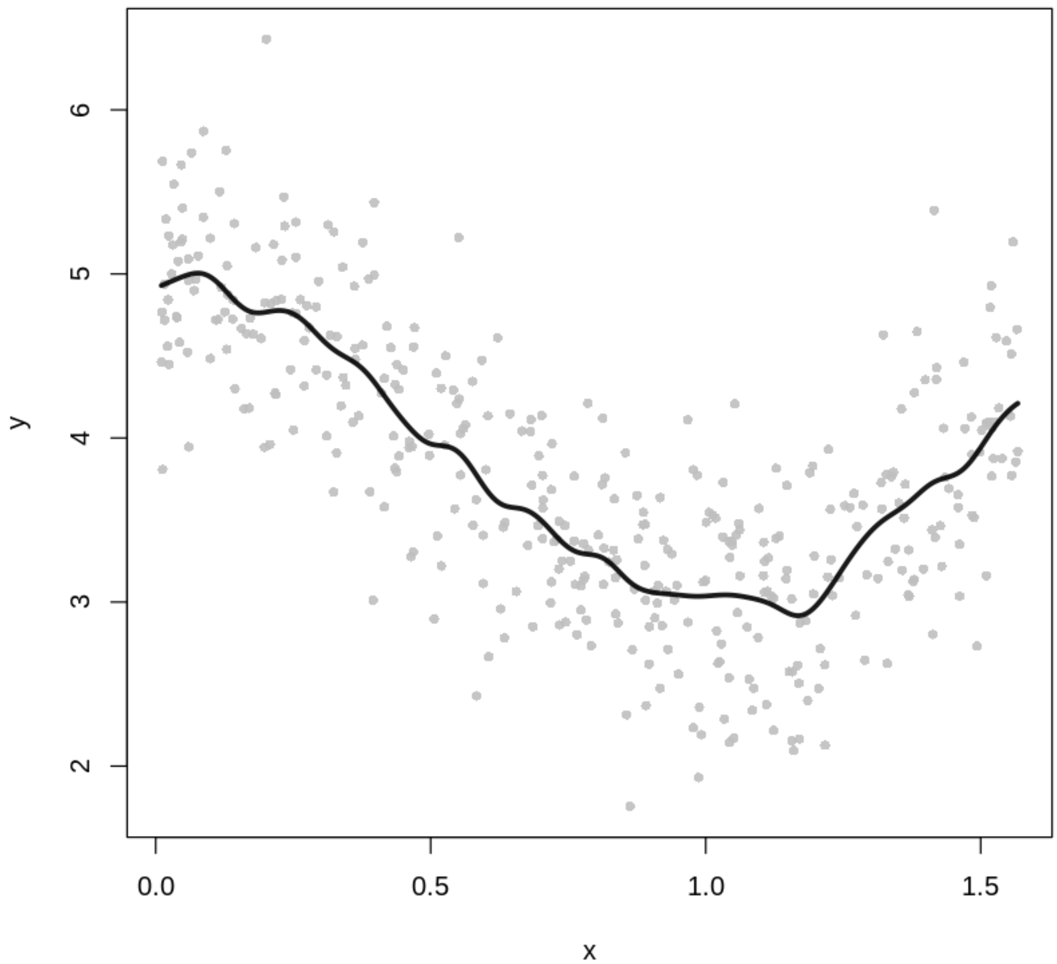
[219]:
```
set.seed(35243) #set the random number generator seed.
n_size = floor(0.8 * n) #find the number corresponding to 80% of the data
index = sample(seq_len(nrow(df_sim)), size = n_size) #randomly sample indicies␣
↪to be included in the training set

train_sim= df_sim[index, ] #set the training set to be the randomly sampled␣
↪rows of the dataframe
test_sim = df_sim[-index, ] #set the testing set to be the remaining rows
dim(test_sim) #check the dimensions
dim(train_sim) #check the dimensions
```

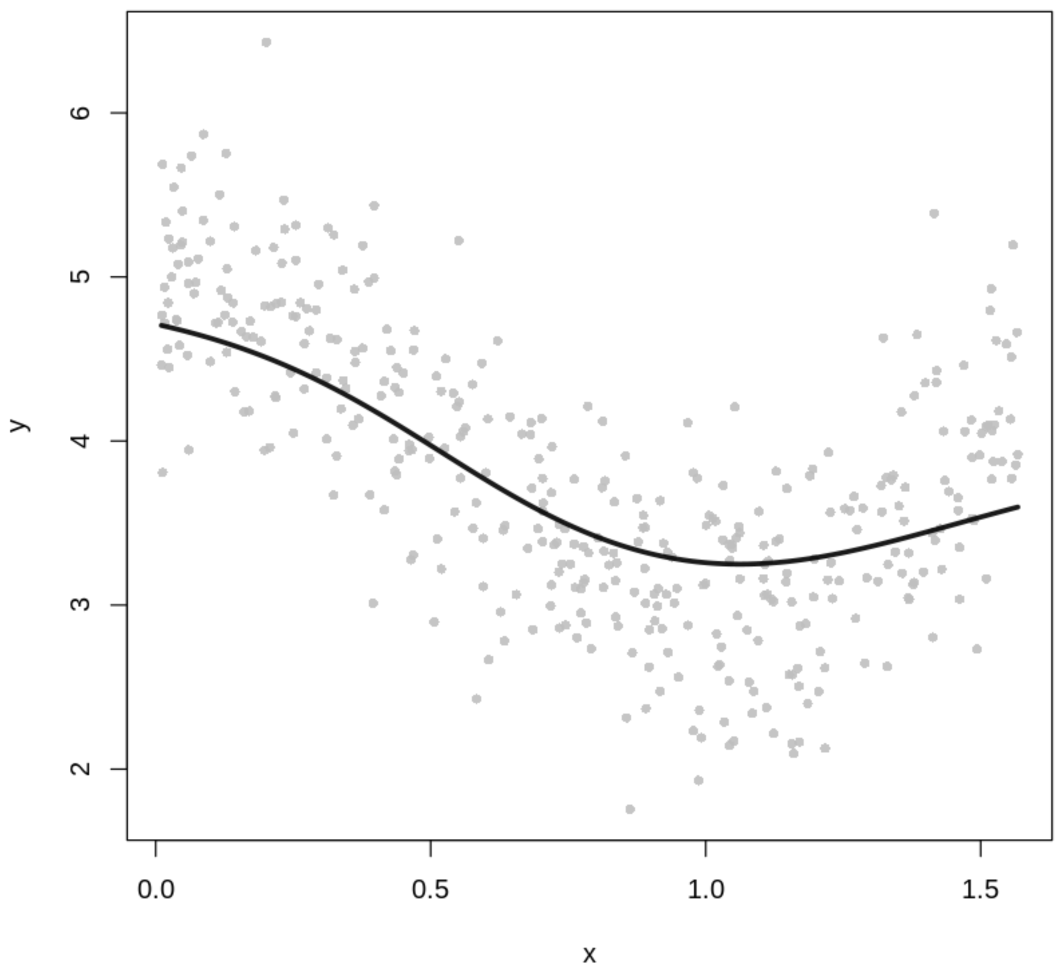
1. 100 2. 2

1. 400 2. 2

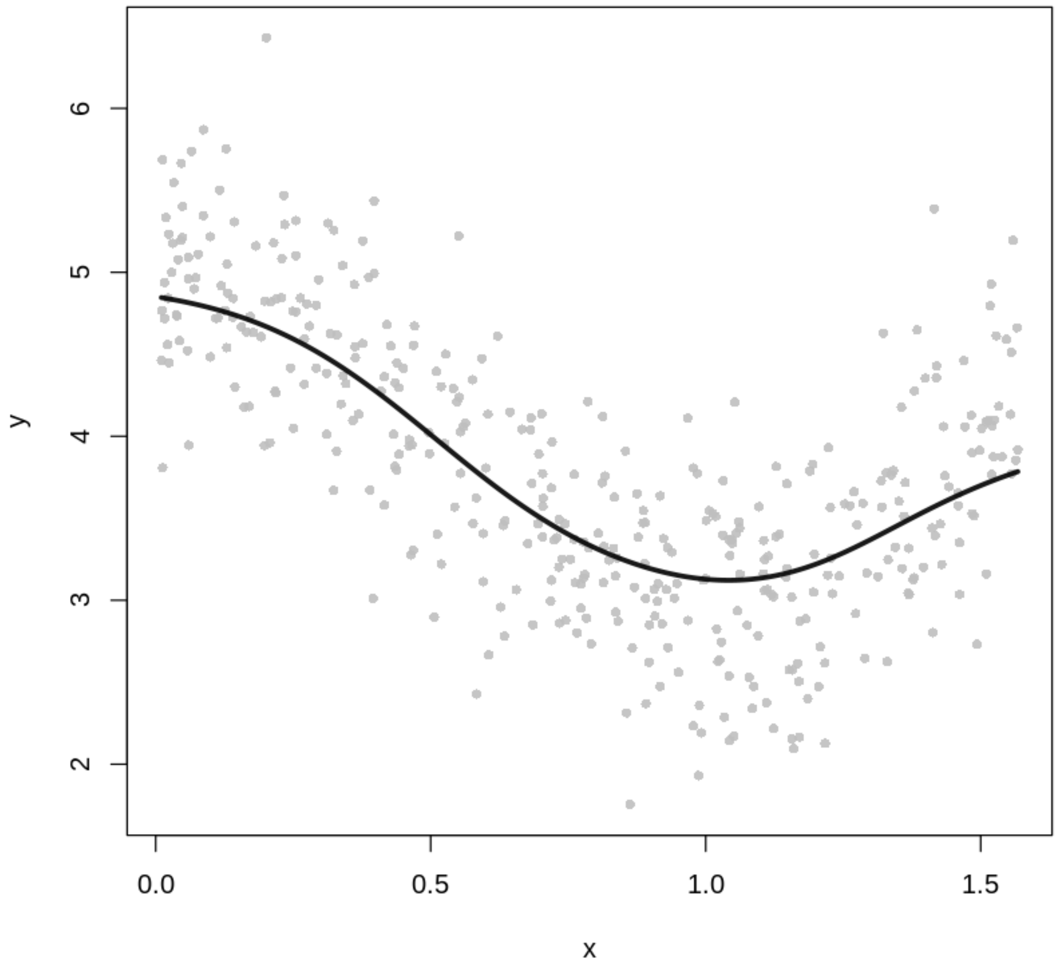
[220]:
```
z = ksmooth(train_sim$x, train_sim$y, "normal", 0.1)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```



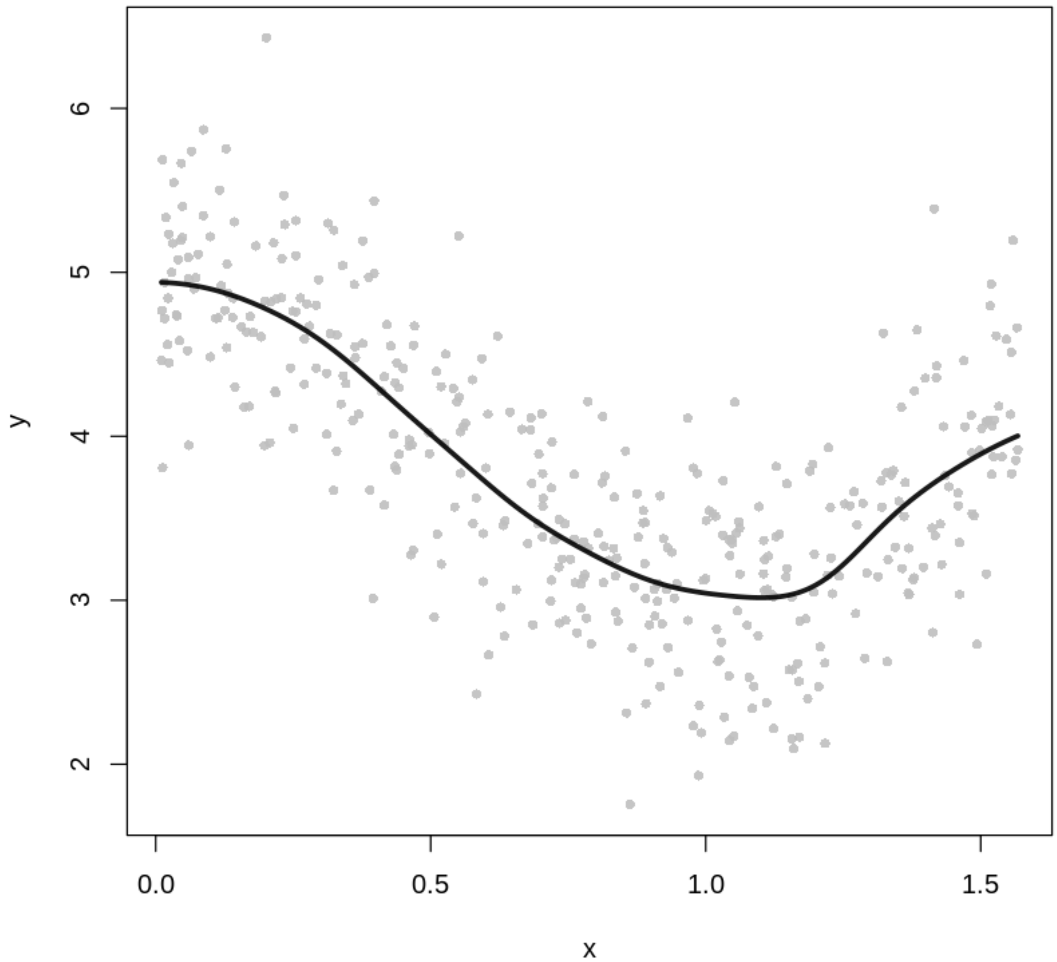
[221]:
```
z = ksmooth(train_sim$x, train_sim$y, "normal", 0.75)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```



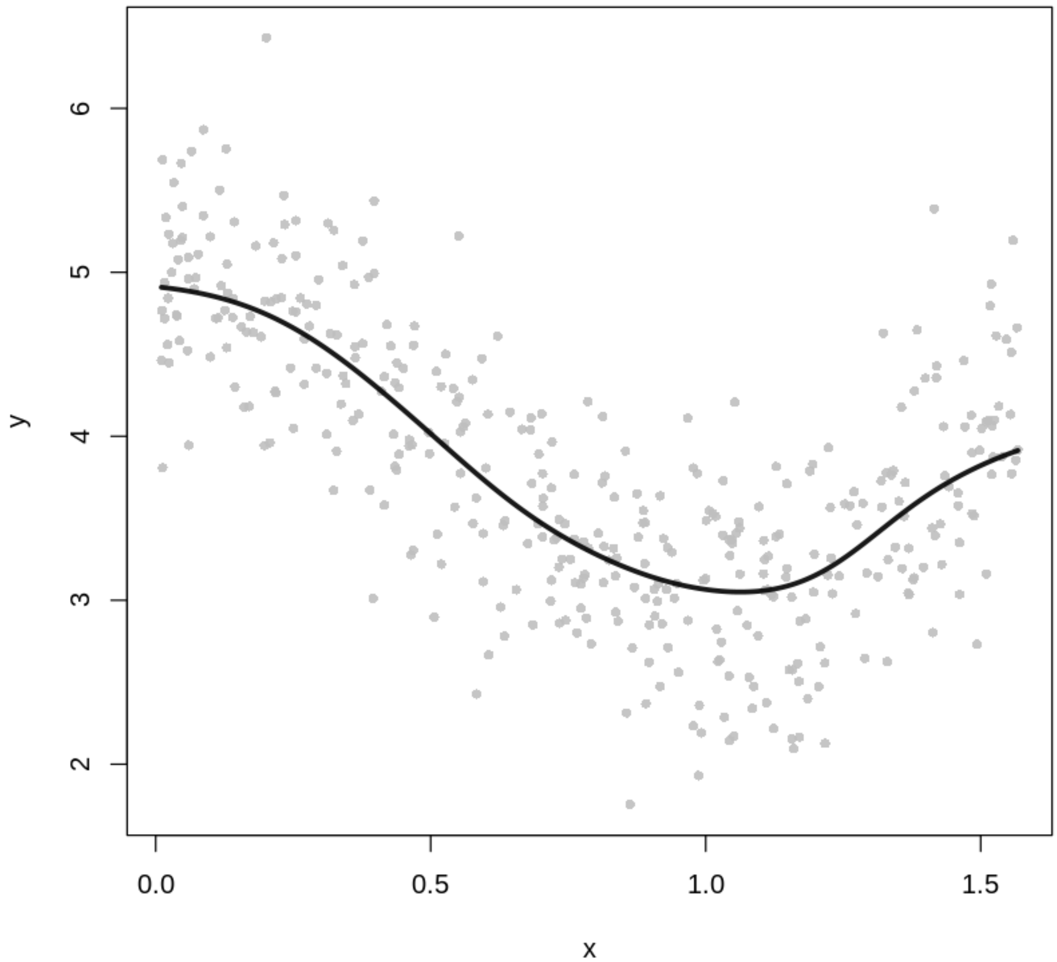
[222]:
```
z = ksmooth(train_sim$x, train_sim$y, "normal", 0.5)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```



[223]:
```
z = ksmooth(train_sim$x, train_sim$y, "normal", 0.25)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```



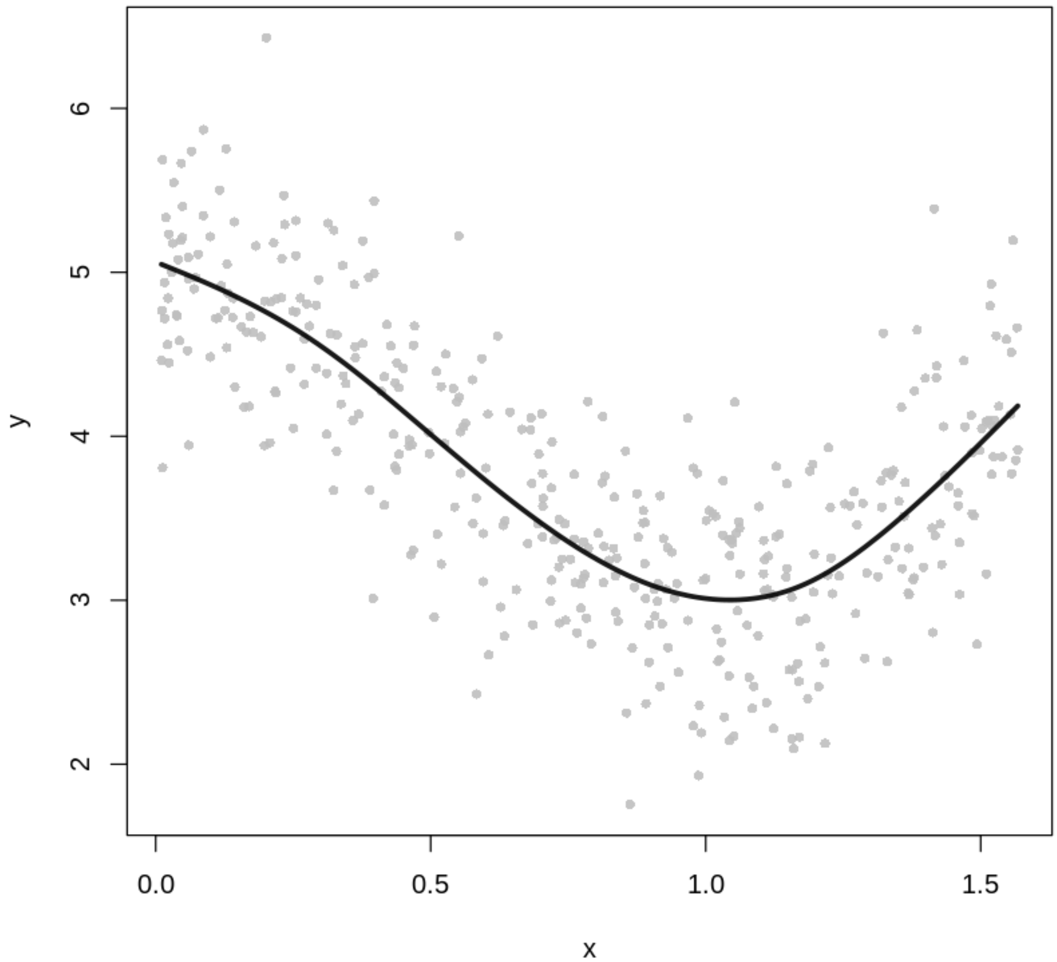
[224]:
```
#1.a
z = ksmooth(train_sim$x, train_sim$y, "normal", 0.35)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(z, lwd=3, col = alpha("black", 0.9))
```



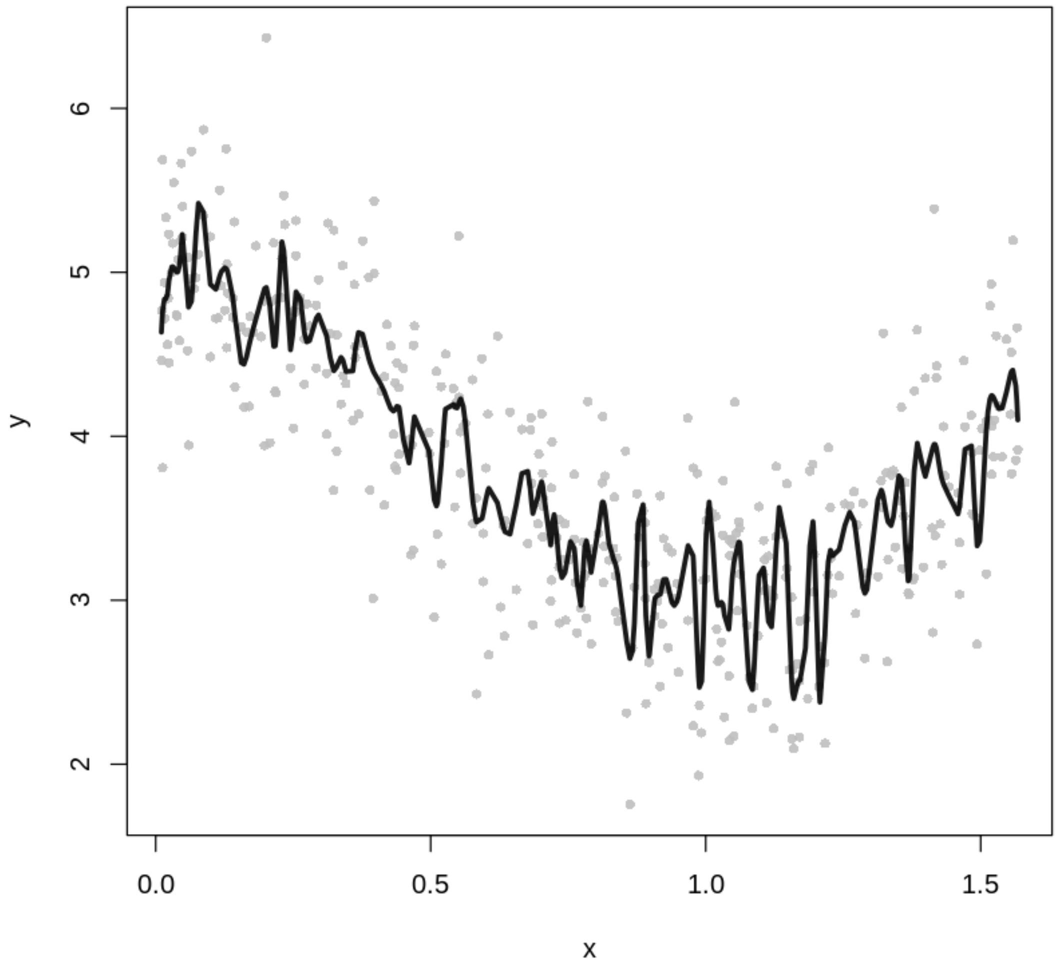
4 Answer 2.a

It seems all of the bandwidth fail to capture the large increase in slope towards the end of the plot. However, setting lambda equal to 0.35 allows us to capture the initial curvature at the beginning and towards the middle of the plot. Therefore, `lambda = 0.35` is out best choice.

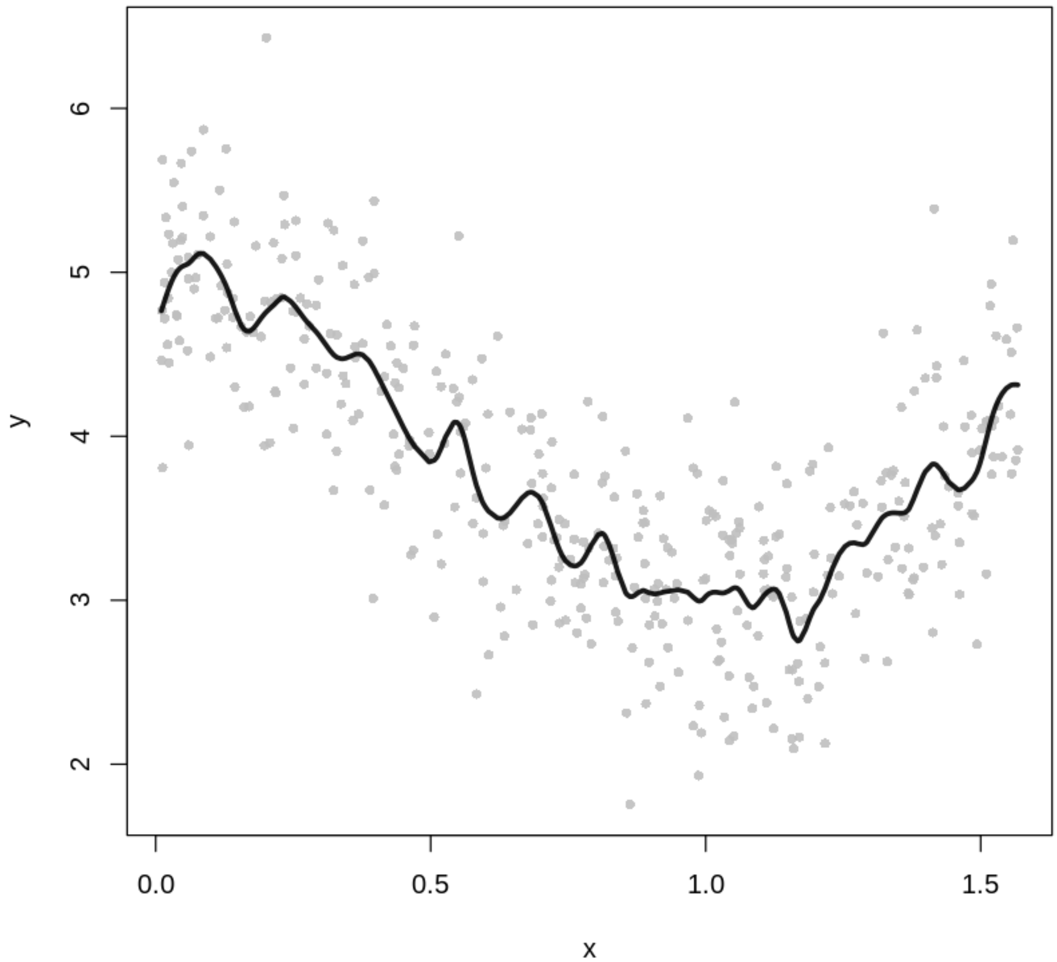
[225]:
```
#1.b
lm_smooth_sim = smooth.spline(train_sim$x, train_sim$y, spar=1)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(lm_smooth_sim, lwd=3, col = alpha("black", 0.9))
```



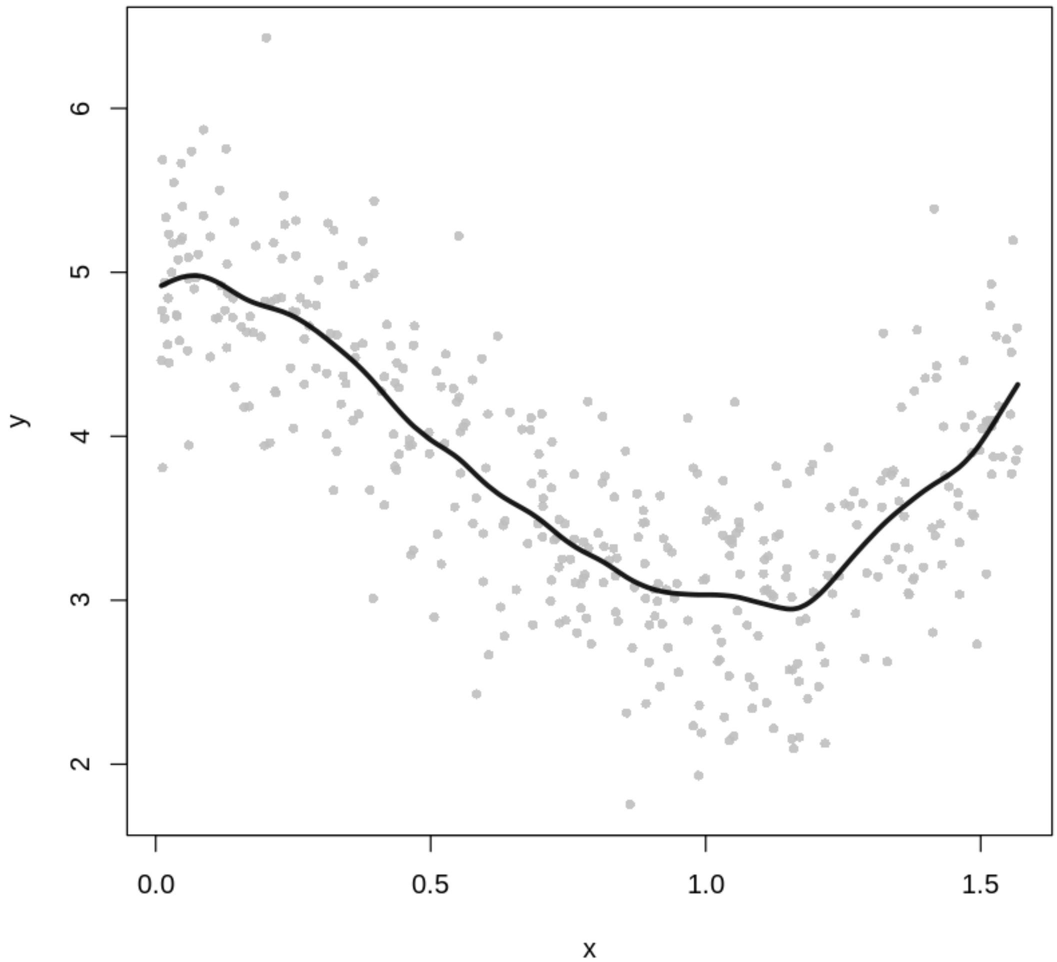
[226]:
```
lm_smooth_sim = smooth.spline(train_sim$x, train_sim$y, spar=0.1)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(lm_smooth_sim, lwd=3, col = alpha("black", 0.9))
```



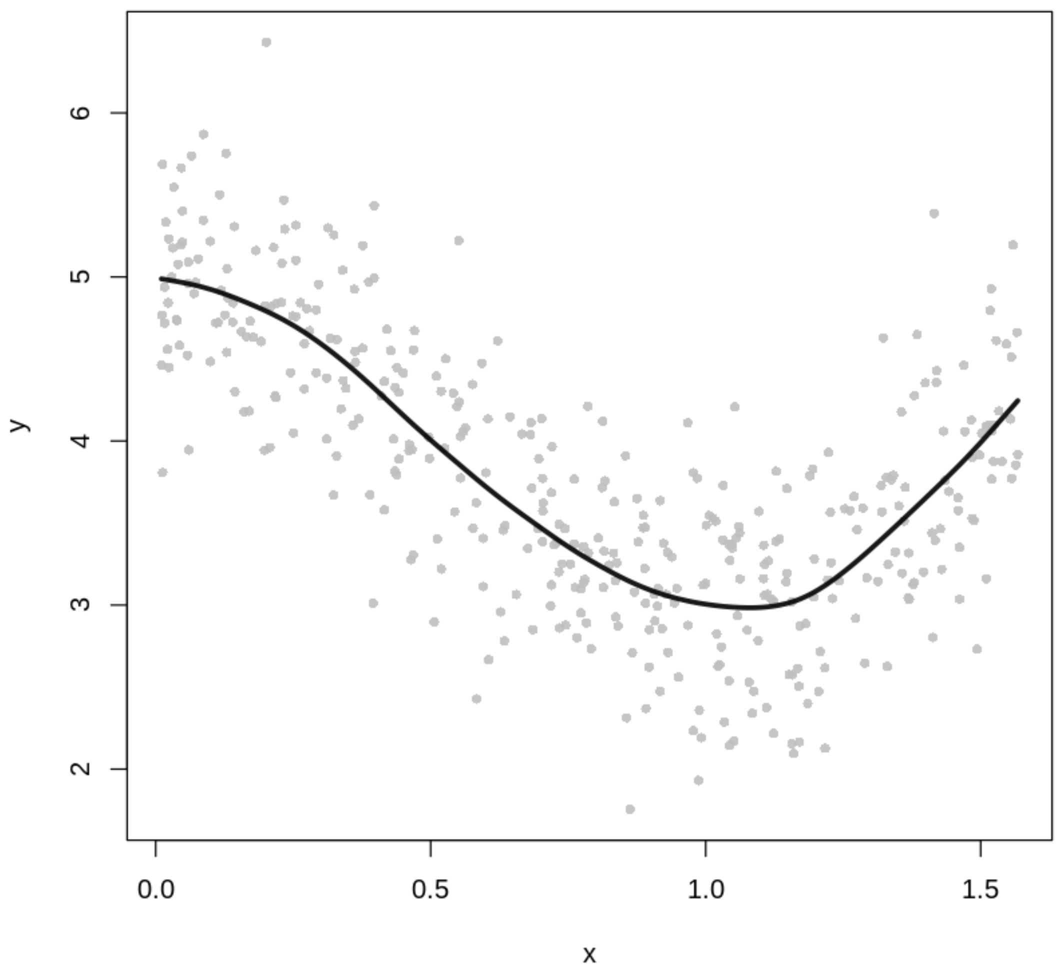
[227]:
```
lm_smooth_sim = smooth.spline(train_sim$x, train_sim$y, spar=0.5)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(lm_smooth_sim, lwd=3, col = alpha("black", 0.9))
```



[228]:
```
lm_smooth_sim = smooth.spline(train_sim$x, train_sim$y, spar=0.75)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(lm_smooth_sim, lwd=3, col = alpha("black", 0.9))
```



[229]:
```
lm_smooth_sim = smooth.spline(train_sim$x, train_sim$y, spar=0.9)
plot(y ~ x, pch=16, data=train_sim, cex=0.8, col = alpha("grey", 0.9))
lines(lm_smooth_sim, lwd=3, col = alpha("black", 0.9))
```



5 Answer 2.B

Having `spar=0.9` helps us fit a line that represents the curvature well. It captures the initial flatness and then drop in all the way until x = 1. Also, unlike kernel regression, it captures the increase in slope towards the end of the plot.

[230]:
```
#plot() not working with loess method

ggplot(train_sim, aes(x=x, y=y)) +
    geom_point(alpha=0.5) +
    geom_smooth(span=0.1, se=FALSE, color="light green") +
```

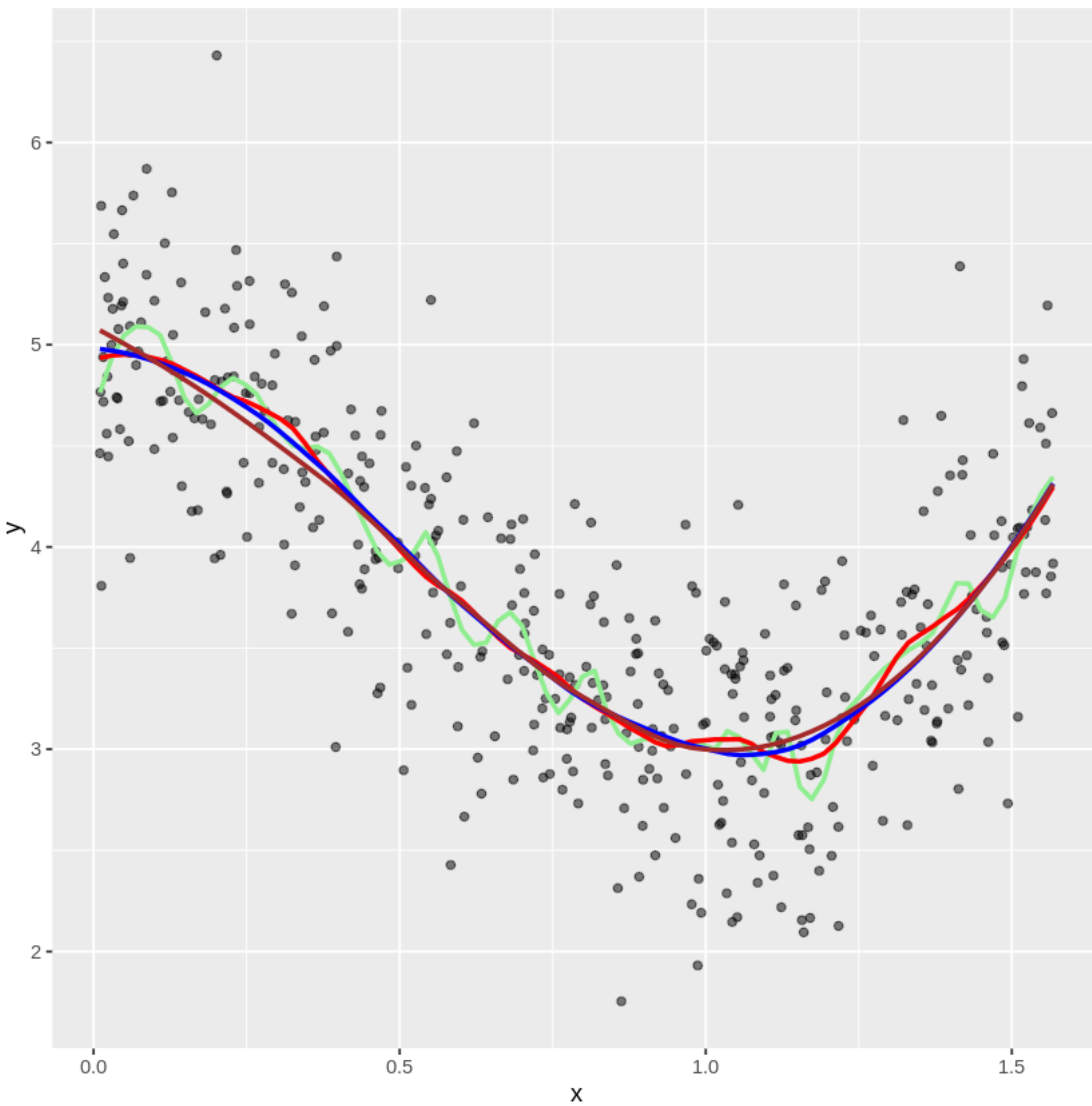
```r
  geom_smooth(span=0.25, se=FALSE, color="red") +
  geom_smooth(span=0.50, se=FALSE, color = "blue") +
  geom_smooth(span=0.75, se=FALSE, color="brown")
```

```
`geom_smooth()` using method = 'loess' and formula 'y ~ x'

`geom_smooth()` using method = 'loess' and formula 'y ~ x'

`geom_smooth()` using method = 'loess' and formula 'y ~ x'

`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



[231]:
```
lm_loess = loess(y ~ x, data= train_sim, span=0.9)
```

6 Answer 2.c

The best fit is `span = 0.75` (brown plot). The initial steepness works captured as well as treh increase in the slop towards the end of the plot.

[233]:
```
#1.d
#Kernel Regression
kreg_pred = ksmooth(train_sim$x, train_sim$y, kernel="normal",
                      40, x.points=test_sim$x)
mean((test_sim$y - kreg_pred$y)^2)

#Kernel Smoothing
ksmooth_pred = predict(lm_smooth_sim, x=test_sim$x)
mean((test_sim$y - ksmooth_pred$y)^2)

#Loess
loess_pred = predict(lm_loess, test_sim$x)
mean((test_sim$y - loess_pred)^2)
```

0.57521925799043

0.275852204625953

0.272164399565335

The Loess model has the smallest MSPE out of all of the models.

[ ]:
```
```