# C3M3_peer_review

November 20, 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.

[1]:
```
# Load Required Packages
library(ggplot2)
library(mgcv)
```

```
Loading required package: nlme

This is mgcv 1.8-31. For overview type 'help("mgcv-package")'.
```

# 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.

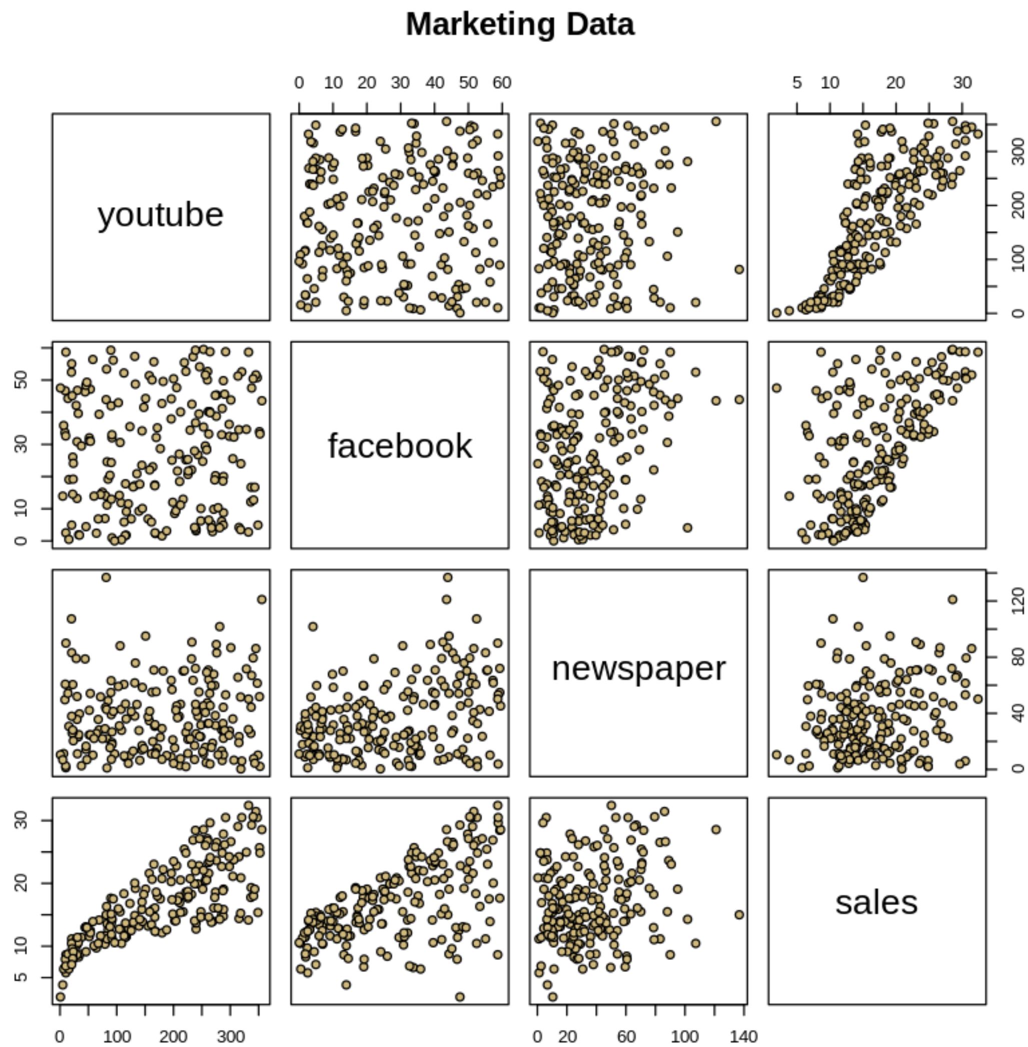
- `sales`: the value in the $i^{th}$ row of the sales column is a measurement of the sales (in thousands of units) for product $P$ for company $i$.

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`).

[2]:
```
# 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  
```




```
[3]: 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

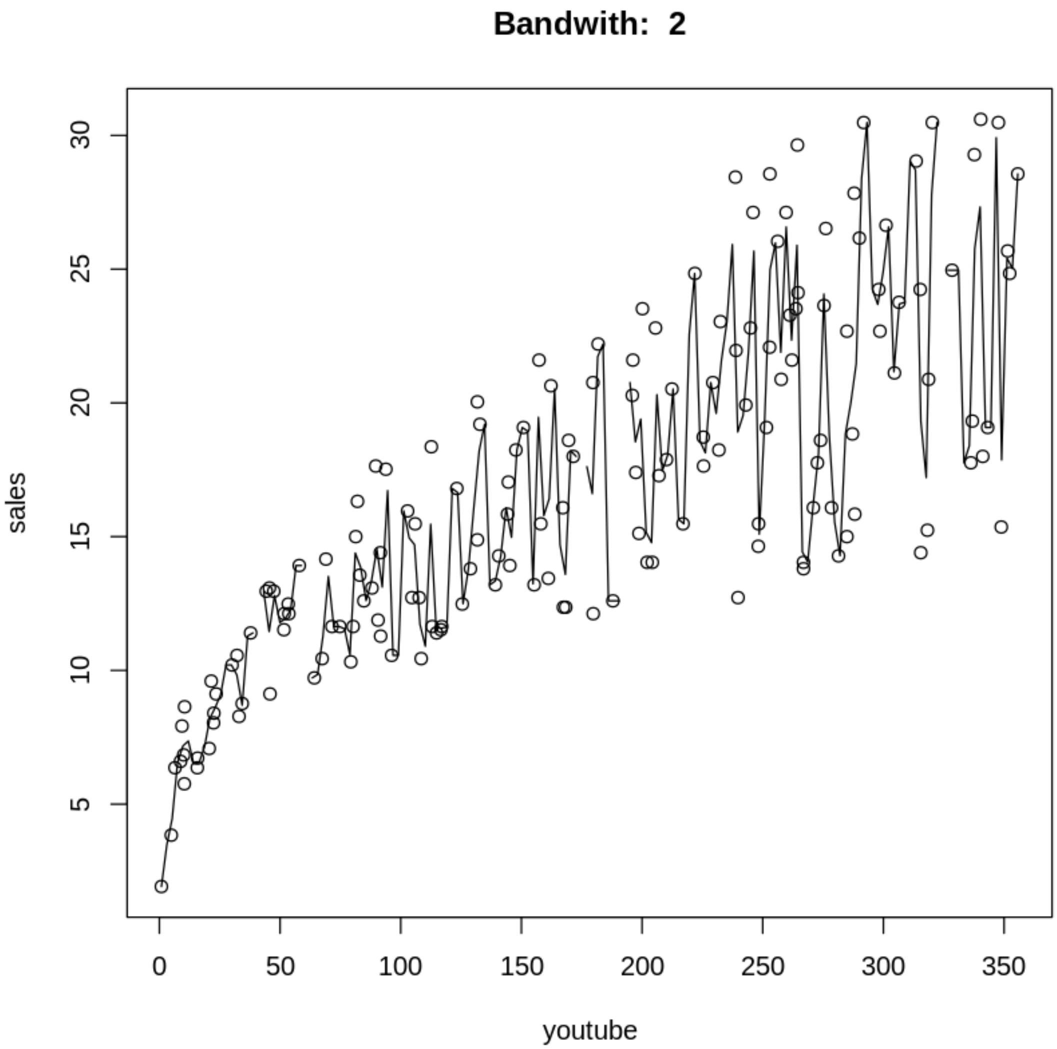
1. 160 2. 4

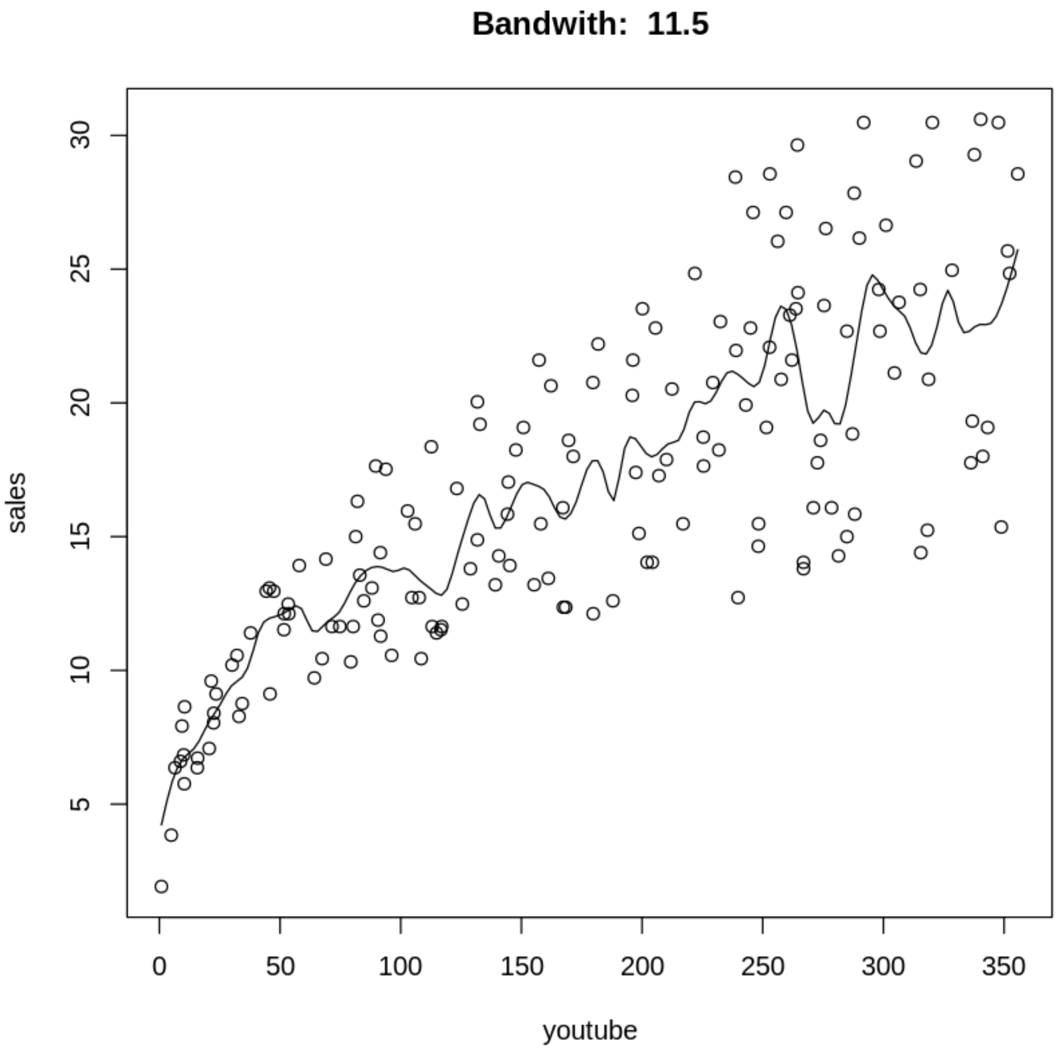
**1.(a) Working with nonlinearity: Kernel regression**

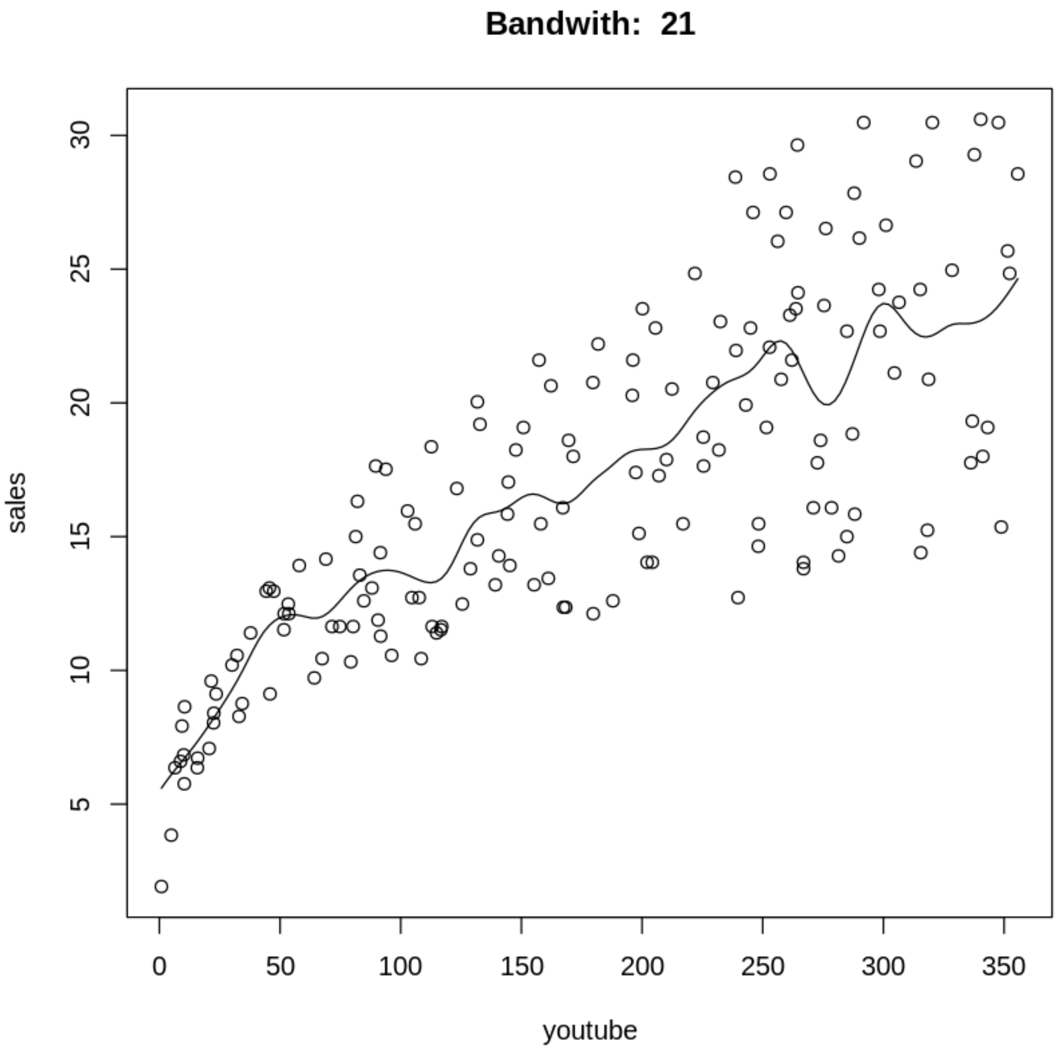
Note that the relationship between sales and youtube is nonlinear. This was a problem for us back in the first course in this specialization, when we modeled the data as if it were linear. For now, let's just focus on the relationship between sales and youtube, omitting the other variables (future lessons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

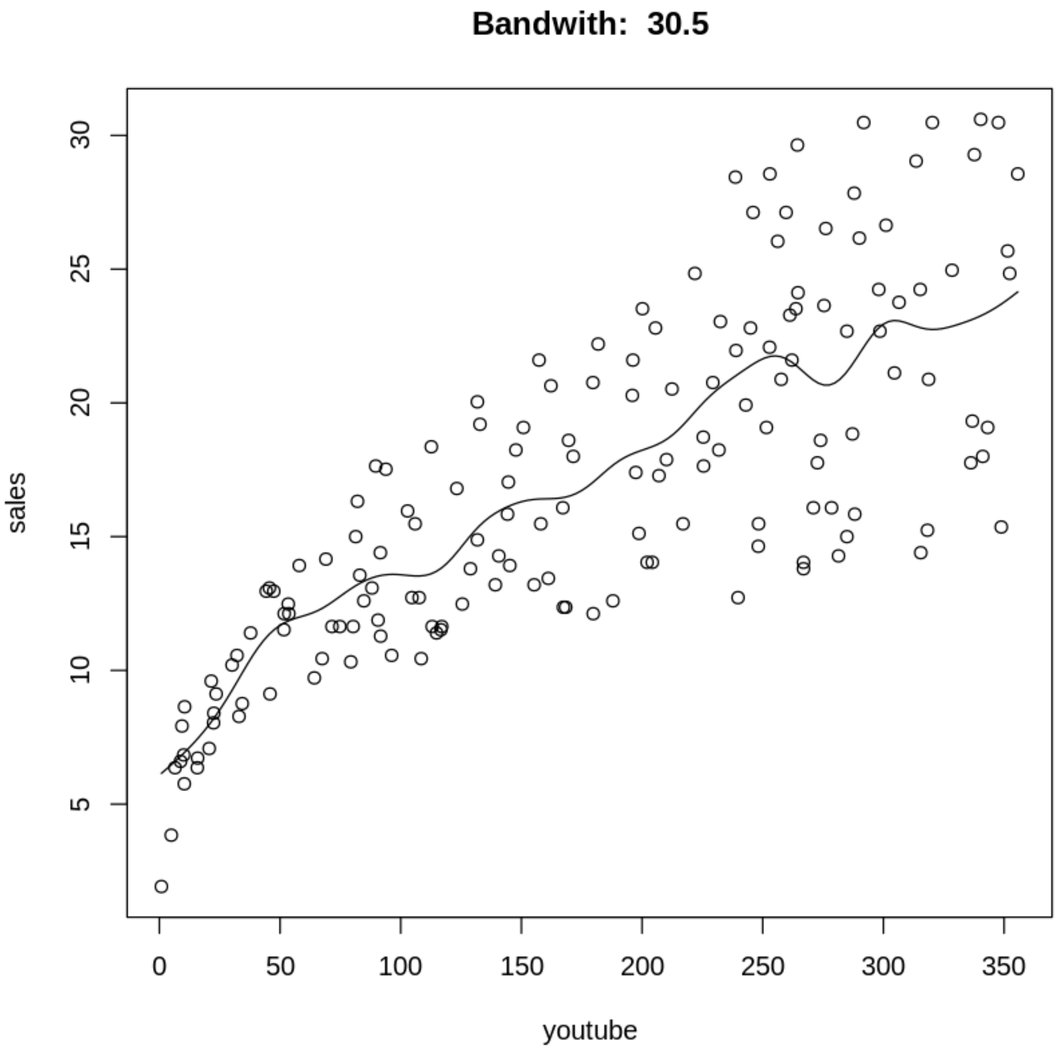
Using the train_marketing set, plot sales (response) against youtube (predictor), and then fit and overlay a kernel regression. Experiment with the bandwidth parameter until the smooth looks appropriate, or comment why no bandwidth is ideal. Justify your answer.

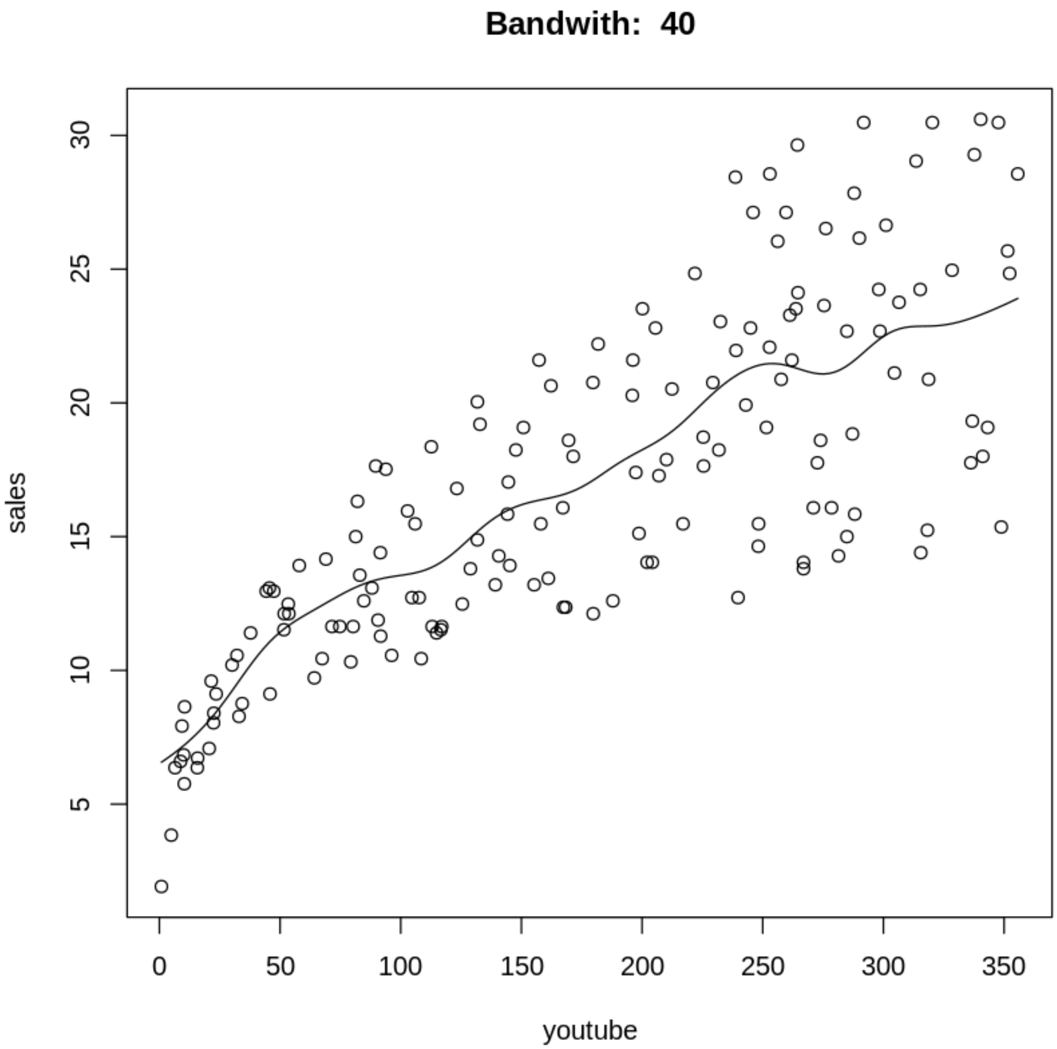
[13]:
```
bds = seq(2,40,length.out=5)
for(bd in bds){
    ks = ksmooth(train_marketing$youtube, train_marketing$sales,␣
 ↪kernel='normal', bd)
    text = paste('Bandwith: ', bd)
    with(train_marketing, plot(youtube, sales, main=text))
    lines(ks)
}
```


Bandwith: 2
sales
youtube
0
50
100
150
200
250
300
350
5
10
15
20
25
30


Bandwith: 11.5
sales
youtube
0
50
100
150
200
250
300
350
5
10
15
20
25
30


Bandwith: 21
sales
youtube
0
50
100
150
200
250
300
350
5
10
15
20
25
30


Bandwith: 30.5
sales
youtube
0
50
100
150
200
250
300
350
5
10
15
20
25
30




I fit the data with kernel estimation with various bandwith (from 2 to 40). It turned out that model with bandwith at 40 fit the data better than the others. Still the fit line is not very smooth accross different values of x.

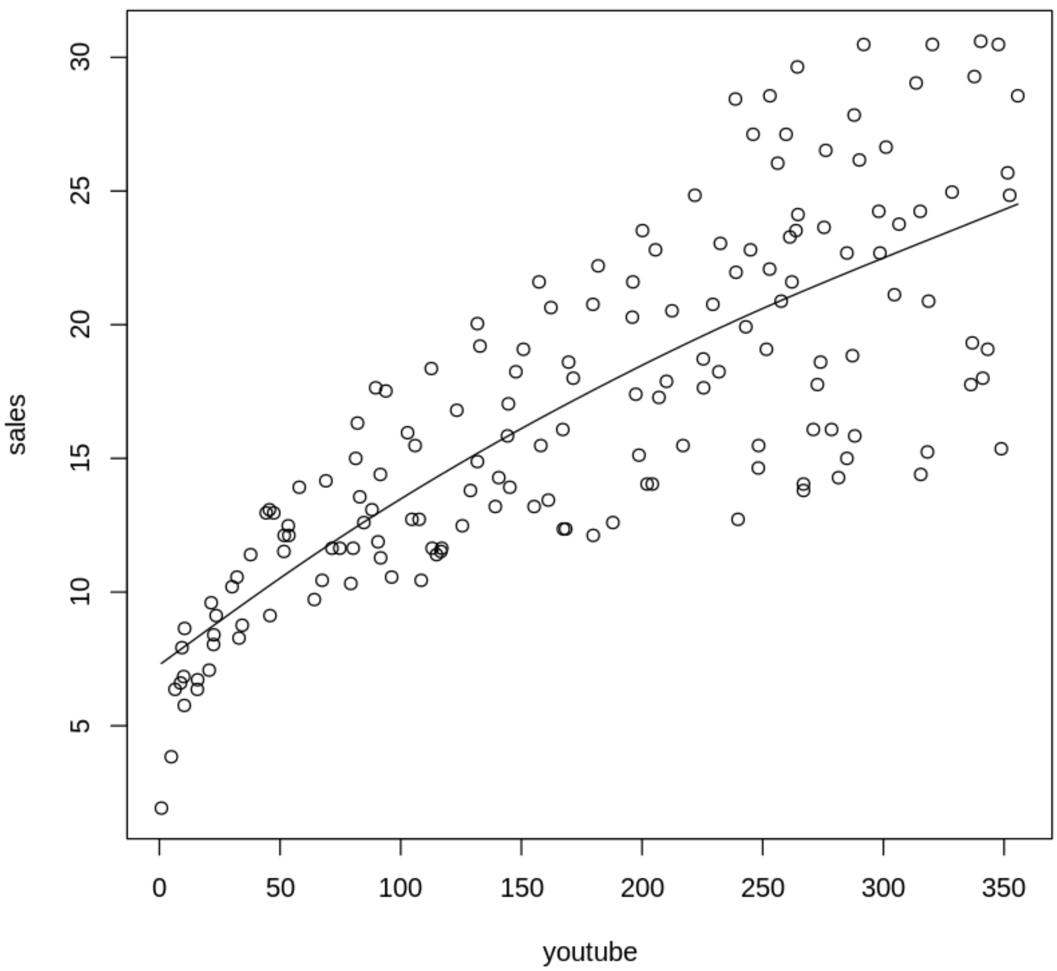
**1.(b) Working with nonlinearity: Smoothing spline regression**

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.

[14]:
```
ss = smooth.spline(train_marketing$youtube, train_marketing$sales)
with(train_marketing, plot(youtube, sales))
lines(ss)
```

```
ss
```

```
Call:
smooth.spline(x = train_marketing$youtube, y = train_marketing$sales)

Smoothing Parameter  spar= 1.111934  lambda= 0.1300704 (16 iterations)
Equivalent Degrees of Freedom (Df): 3.109666
Penalized Criterion (RSS): 2223.953
GCV: 14.73459
```

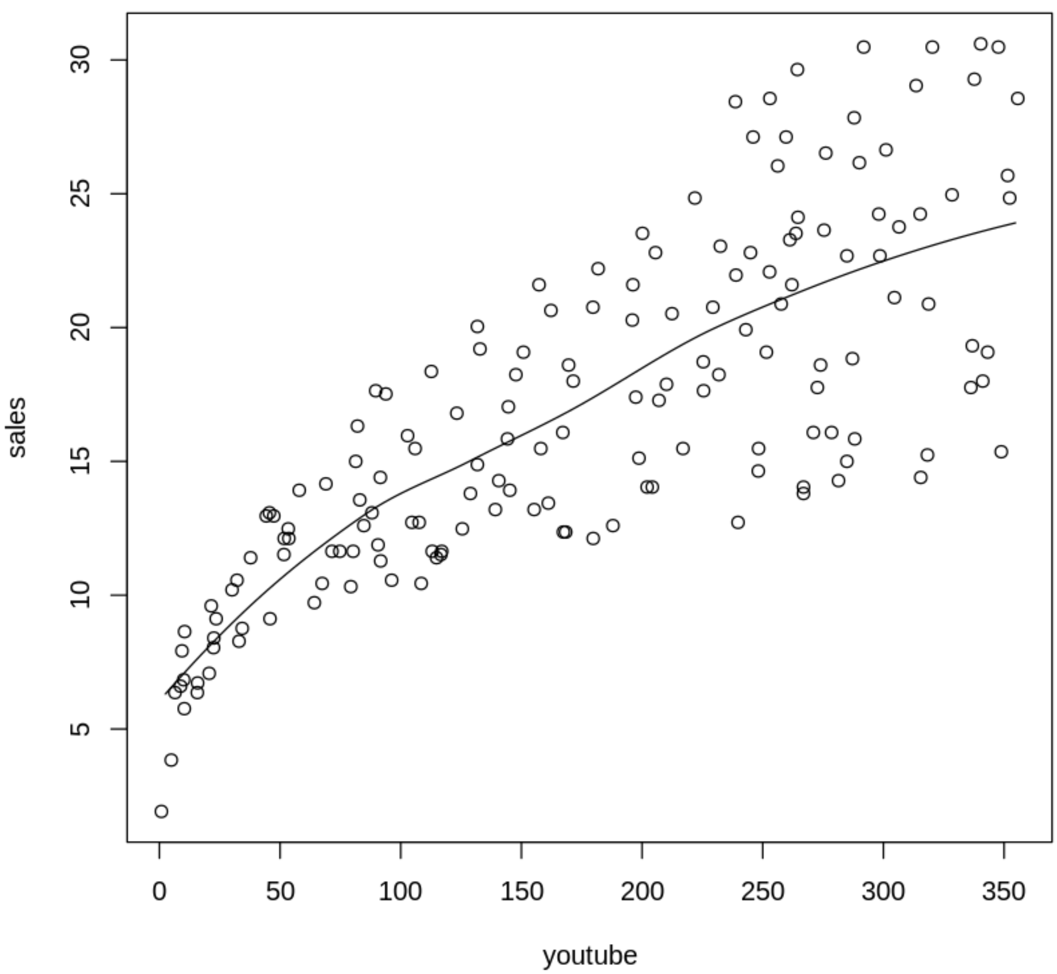



Compared with the kernel estimation, the smoothing spline model fit the data better. The R built-in function suggested that the spar value should be 1.112.

### 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.

[16]:
```
l = loess(sales~youtube, train_marketing)
with(train_marketing, plot(youtube, sales))
newdata = seq(0,400,length.out=dim(train_marketing)[1])
preds = predict(l, newdata)
lines(newdata, preds)
```




The LOESS model fit the data well.

### 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).

[17]:
```
mspe = function(preds){
    value = (preds - test_marketing$sales)**2
    return(mean(value))
}

# Kernel Estimation with Lambda = 30
ke_preds = ksmooth(train_marketing$youtube, train_marketing$sales, 'normal',␣
↪30, x.points=test_marketing$youtube)
ke_mspe = mspe(ke_preds$y)
text = 'MSPE of Kernel Estimation (with bandwith=30): '
paste(text, round(ke_mspe,2))

# Smoothing Spline with Spar = 1.111934
ss_preds = predict(ss, test_marketing$youtube)
ss_mspe = mspe(ss_preds$y)
text = 'MSPE of Smoothing Spline (with spar=1.112): '
paste(text, round(ss_mspe,2))

# Loess
l_preds = predict(l, test_marketing$youtube)
l_mspe = mspe(l_preds)
text = 'MSPE of LOESS: '
paste(text, round(l_mspe,2))
```

'MSPE of Kernel Estimation (with bandwith=30): 65.16'

'MSPE of Smoothing Spline (with spar=1.112): 17.54'

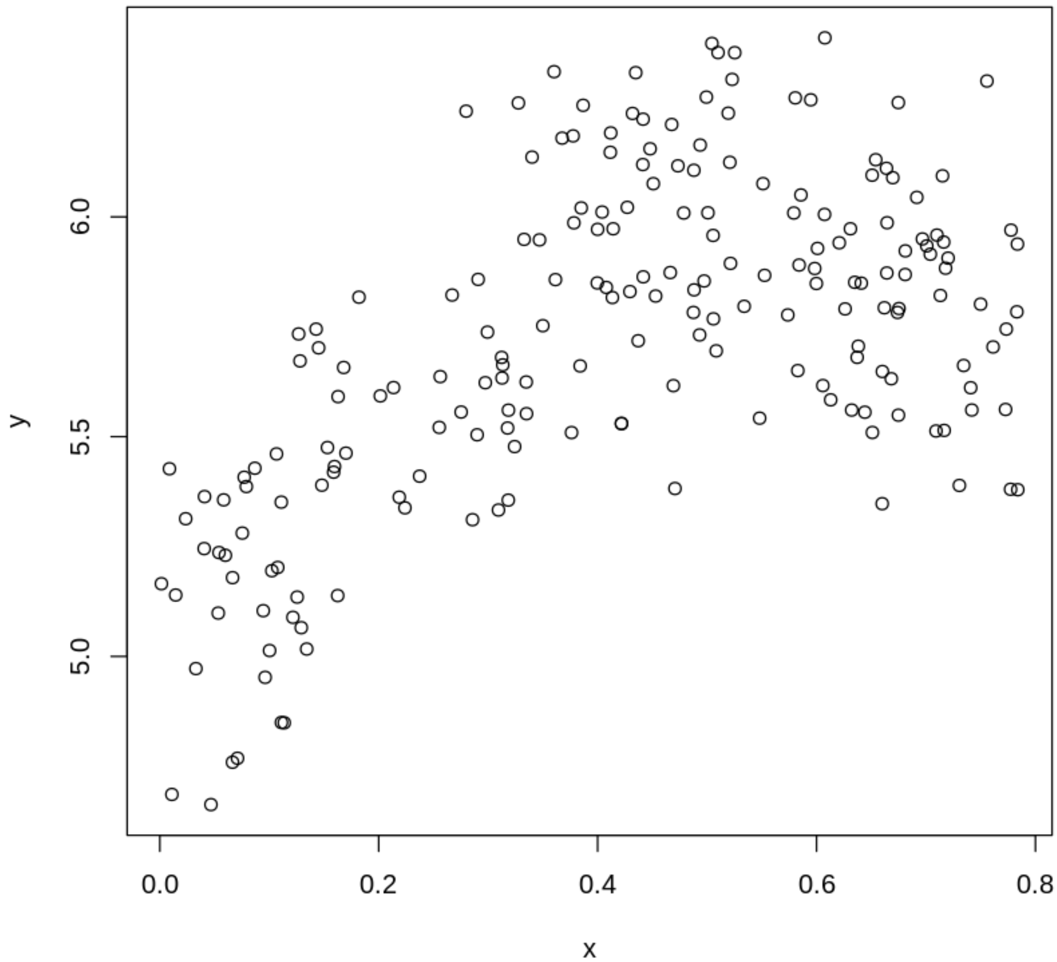
'MSPE of LOESS: 18.04'

The smoothing spline model has the smallest mean squared prediction error. Therefore we should use it for fitting the advertising dataset.

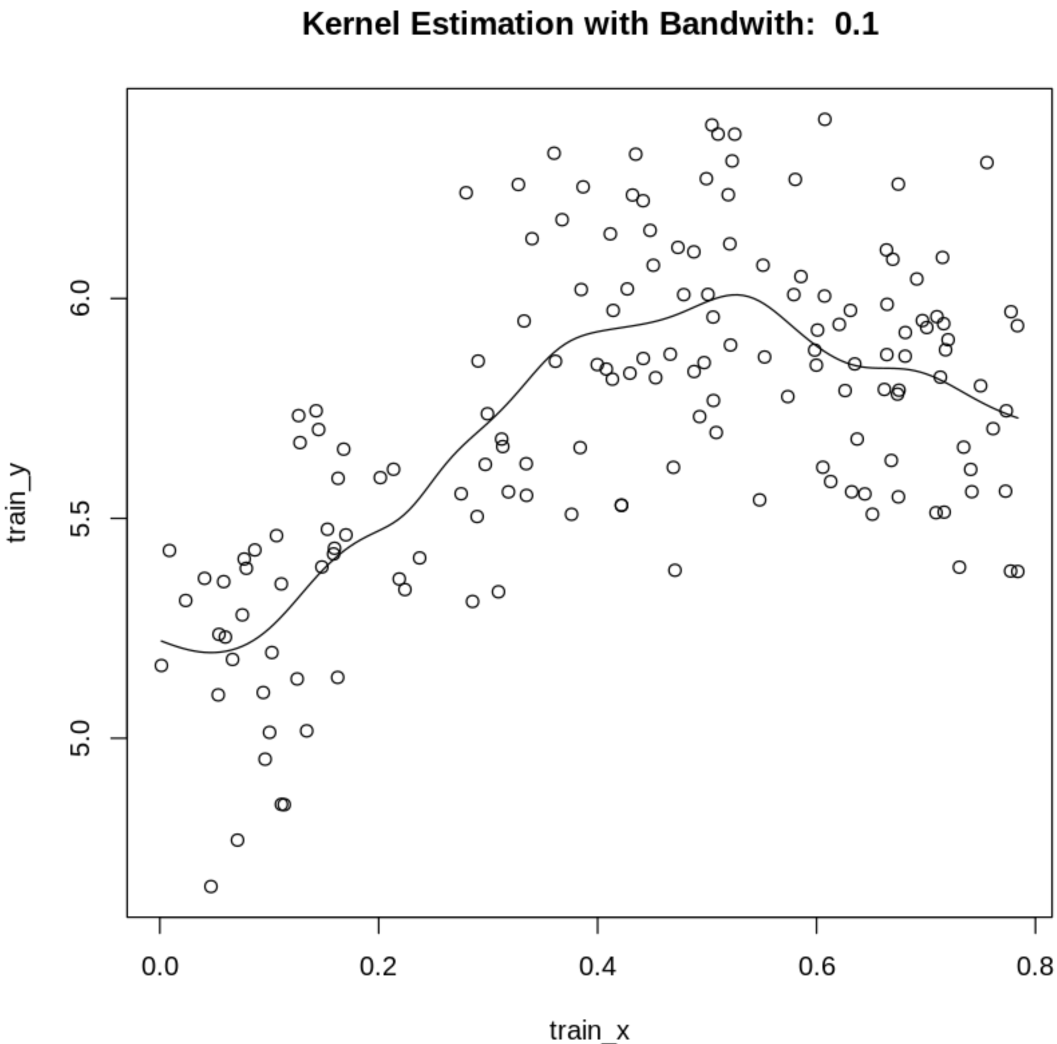
# 3 Problem 2: Simulations!

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.

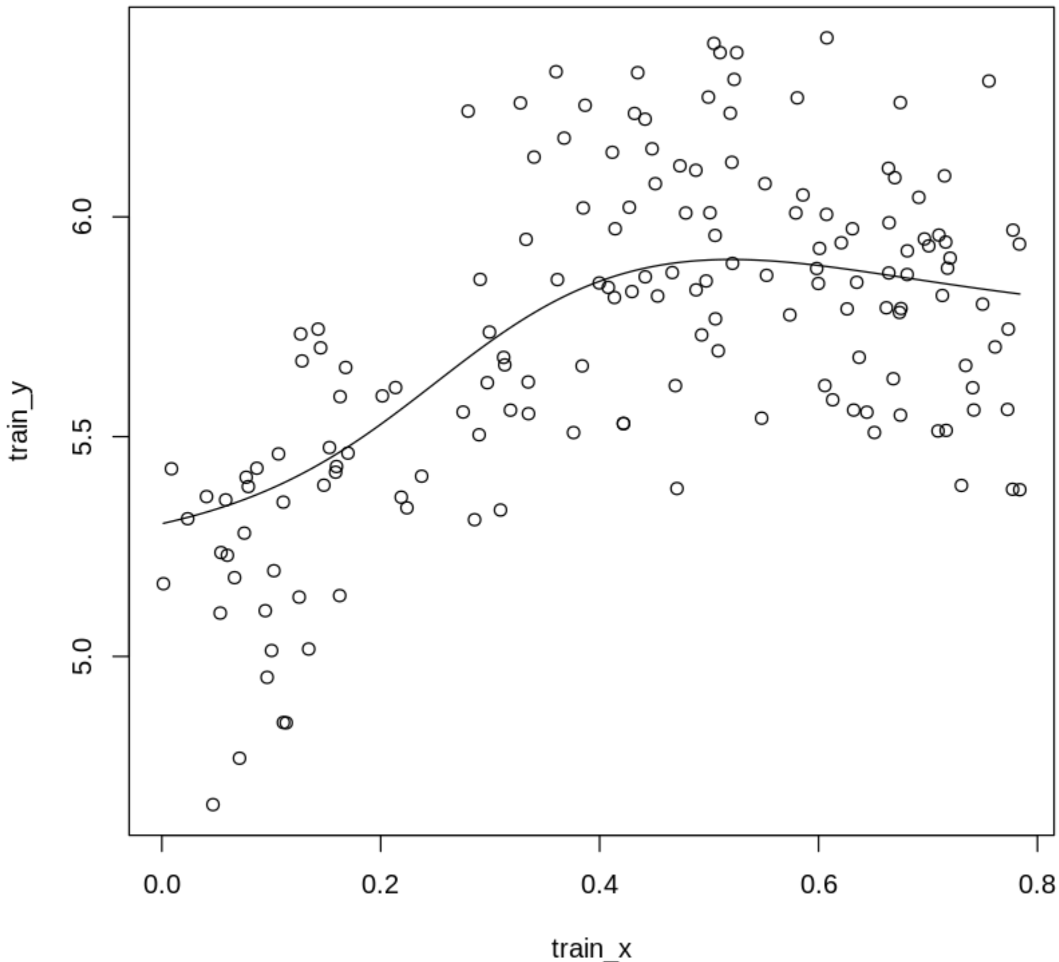
[18]:
```
#simulated data
set.seed(2022)
n = 200
x = runif(n, 0, pi/4)
y = sin(pi*x) + rnorm(n, 0, 0.25) + 5
plot(x,y)

train_index = sample(n, floor(n*0.8))
train_x = x[train_index]
train_y = y[train_index]
test_x = x[-train_index]
test_y = y[-train_index]
```



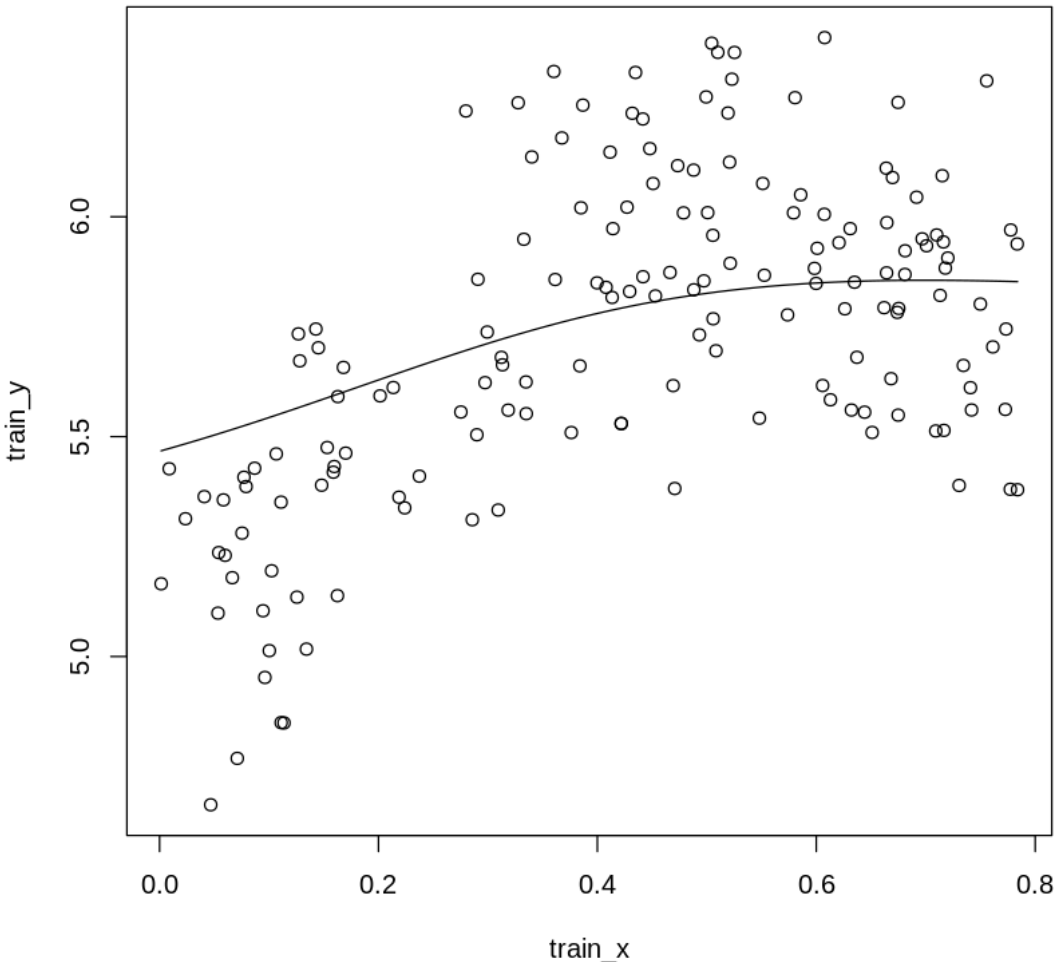

[19]:
```
#1.a
bds = seq(0.1,1.2,length.out=5)
for(bd in bds){
    ks = ksmooth(train_x, train_y, 'normal', bd)
    text = paste('Kernel Estimation with Bandwith: ', bd)
    plot(train_x, train_y, main=text)
    lines(ks)
}
```

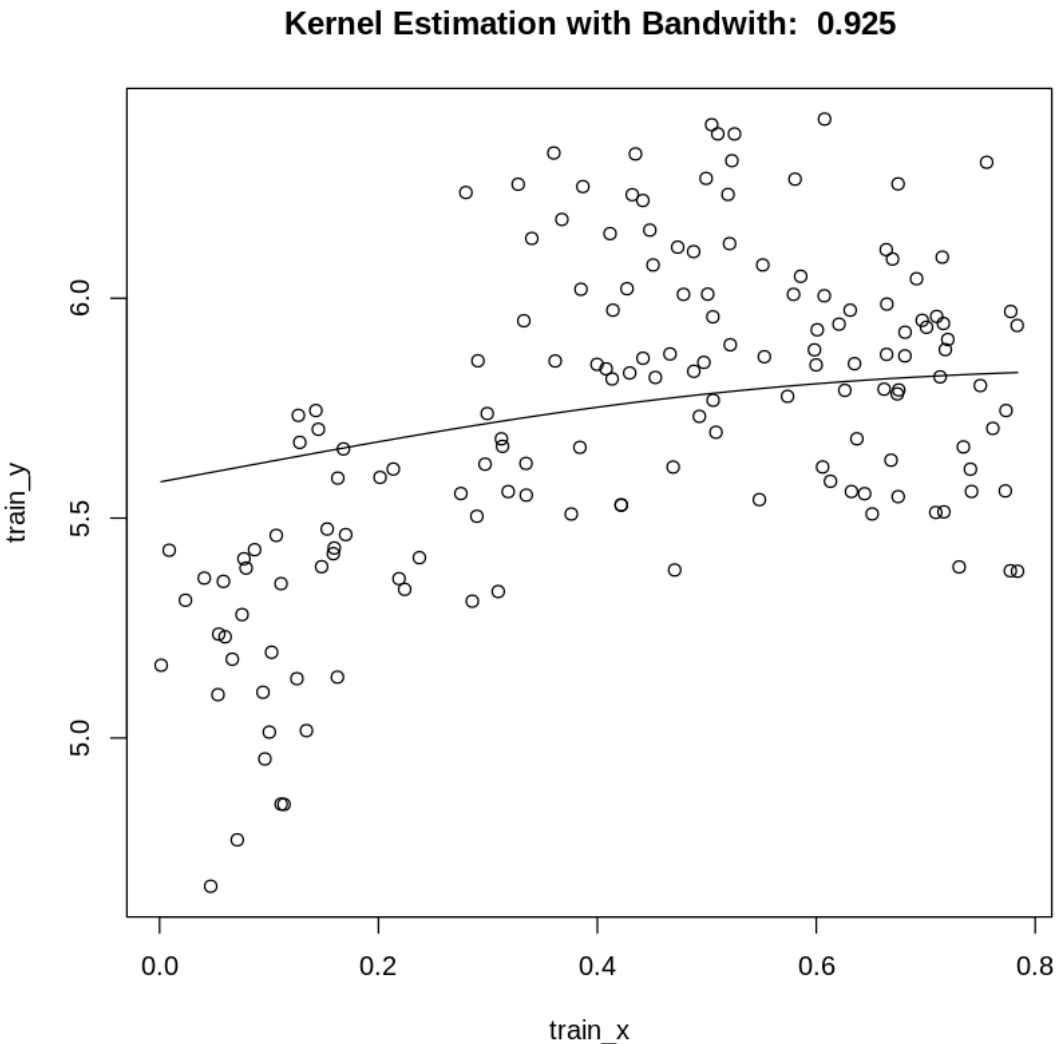

Kernel Estimation with Bandwith: 0.1
train_y
6.0
5.5
5.0
0.0
0.2
0.4
0.6
0.8
train_x

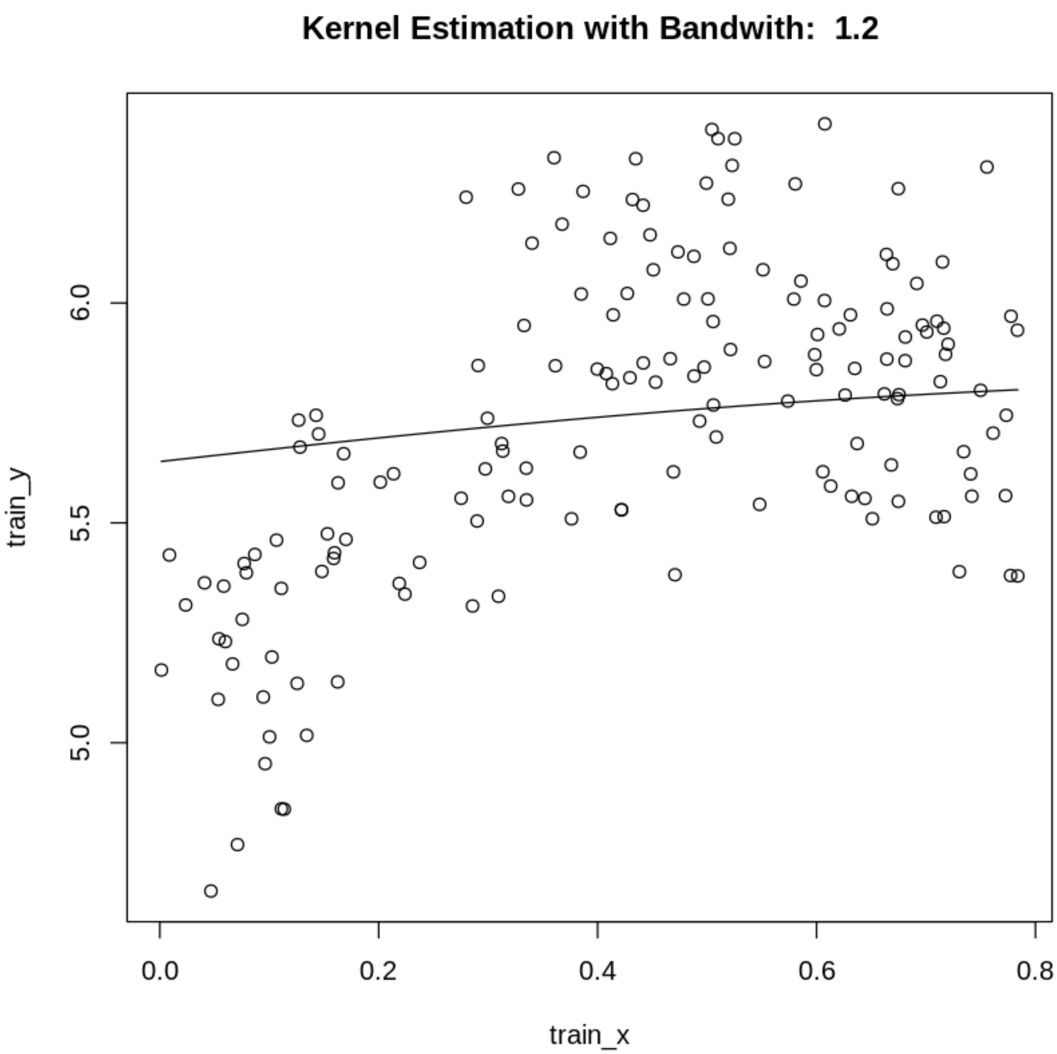
## Kernel Estimation with Bandwith:  0.375



Kernel Estimation with Bandwith: 0.65

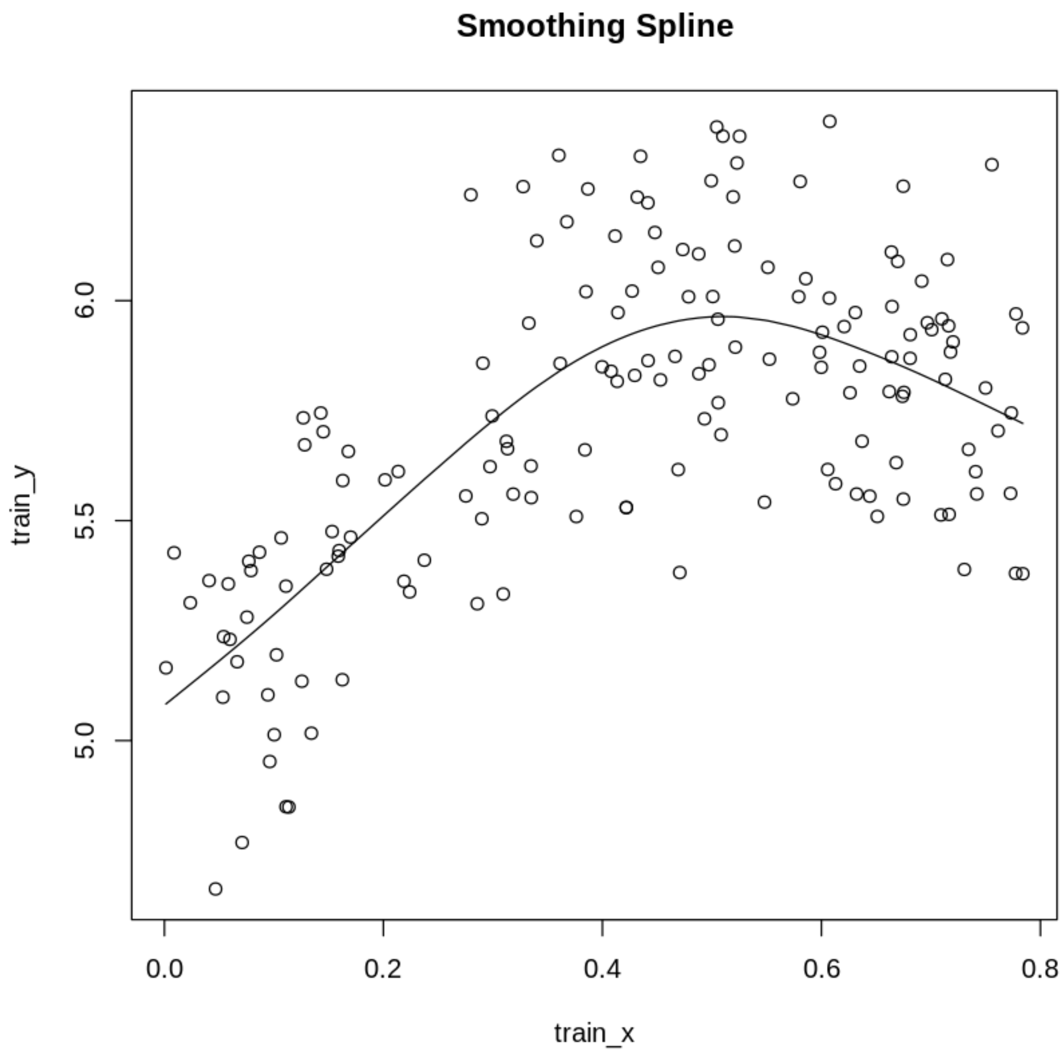



Kernel Estimation with Bandwith: 0.925
train_y
6.0
5.5
5.0
0.0
0.2
0.4
0.6
0.8
train_x




[20]:
```
#1.b
ss = smooth.spline(train_x, train_y)
plot(train_x, train_y, main='Smoothing Spline')
lines(ss)
ss
```

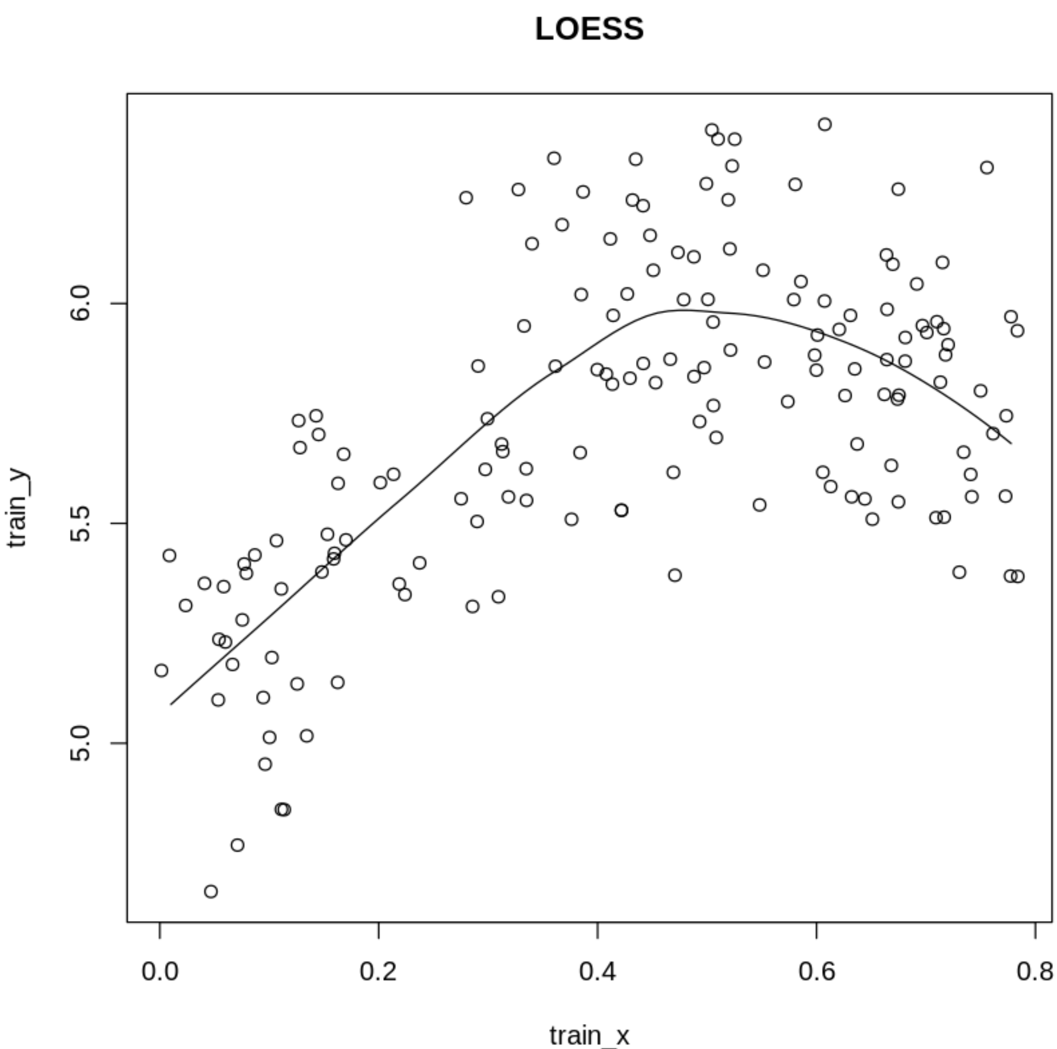
```
Call:
smooth.spline(x = train_x, y = train_y)

Smoothing Parameter  spar= 1.06295  lambda= 0.02350544 (14 iterations)
Equivalent Degrees of Freedom (Df): 4.155956
Penalized Criterion (RSS): 9.653389
GCV: 0.2039945
```




[21]:
```
#1.c
temp = data.frame(x=train_x, y=train_y)
l = loess(y~x, temp)
l
plot(train_x, train_y, main='LOESS')
newdata = seq(0 , 1,length.out=100)
preds = predict(l, newdata)
lines(newdata, preds)
```

```
Call:
loess(formula = y ~ x, data = temp)
```

```
Number of Observations: 160
Equivalent Number of Parameters: 4.26
Residual Standard Error: 0.2497
```




[22]:
```
#1.d
mspe = function(preds){
    value = (preds - y)**2
    return(mean(value))
}

# Kernel Estimation
ke_preds = ksmooth(train_x, train_y, 'normal', 0.925, x.points=test_x)
```

```r
ke_mspe = mspe(ke_preds$y)
text = 'MSPE of Kernel Estimation: '
paste(text, round(ke_mspe,4))

# Smoothing Spline
ss_preds = predict(ss, test_x)
ss_mspe = mspe(ss_preds$y)
text = 'MSPE of Smoothing Spline: '
paste(text, round(ss_mspe,4))

# Loess
l_preds = predict(l, test_x)
l_mspe = mspe(l_preds)
text = 'MSPE of LOESS: '
paste(text, round(l_mspe,4))
```

'MSPE of Kernel Estimation: 0.1397'

'MSPE of Smoothing Spline: 0.2185'

'MSPE of LOESS: 0.2232'

The kernel estimation model has the smallest mean squared prediction error. Therefore we should use it for fitting the simulated dataset.