C3M3_peer_review

June 22, 2023

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.

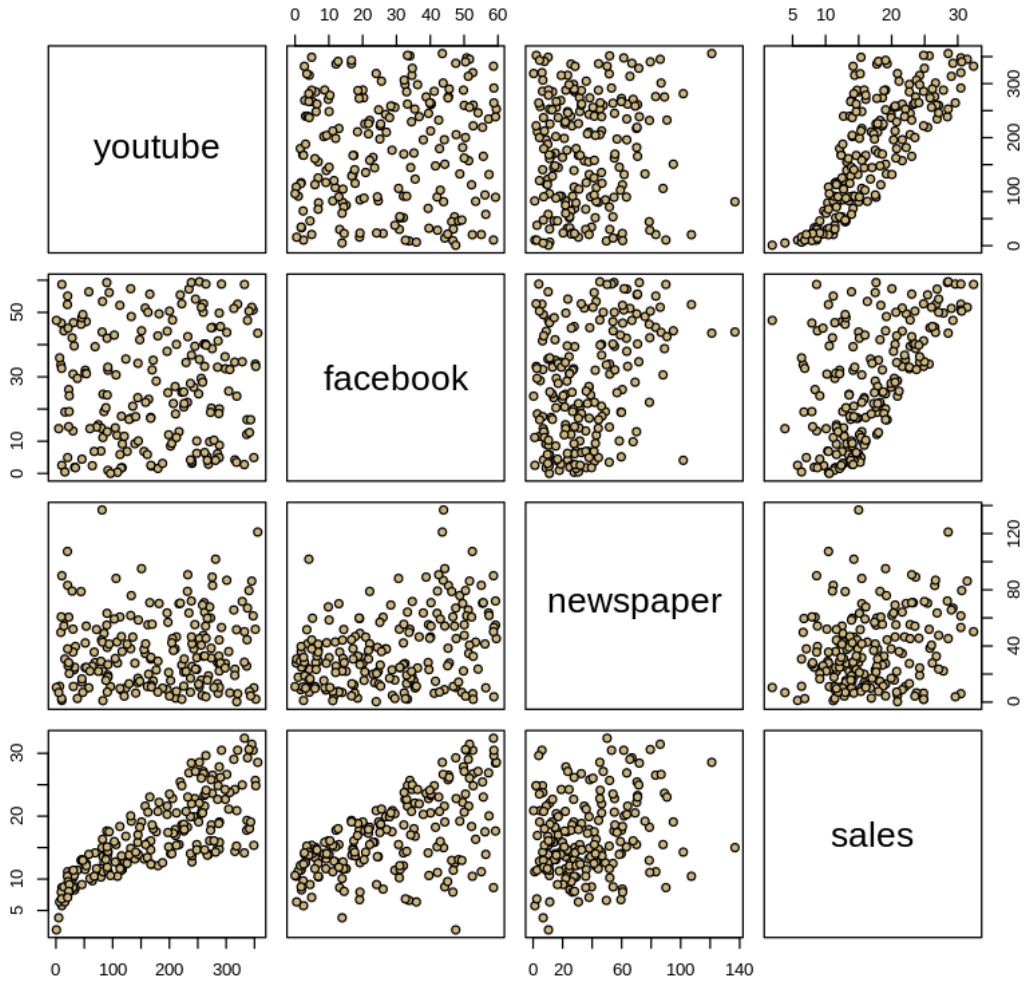
- `sales`: the value in the i^{th} row of the sales column is a measurement of the sales (in thousands of units) for product P for company i.

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]:
```
cor(marketing)
```

A matrix: 4 × 4 of type dbl

	youtube	facebook	newspaper	sales
youtube	1.00000000	0.05480866	0.05664787	0.7822244
facebook	0.05480866	1.00000000	0.35410375	0.5762226
newspaper	0.05664787	0.35410375	1.00000000	0.2282990
sales	0.78222442	0.57622257	0.22829903	1.0000000

[5]:
```
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

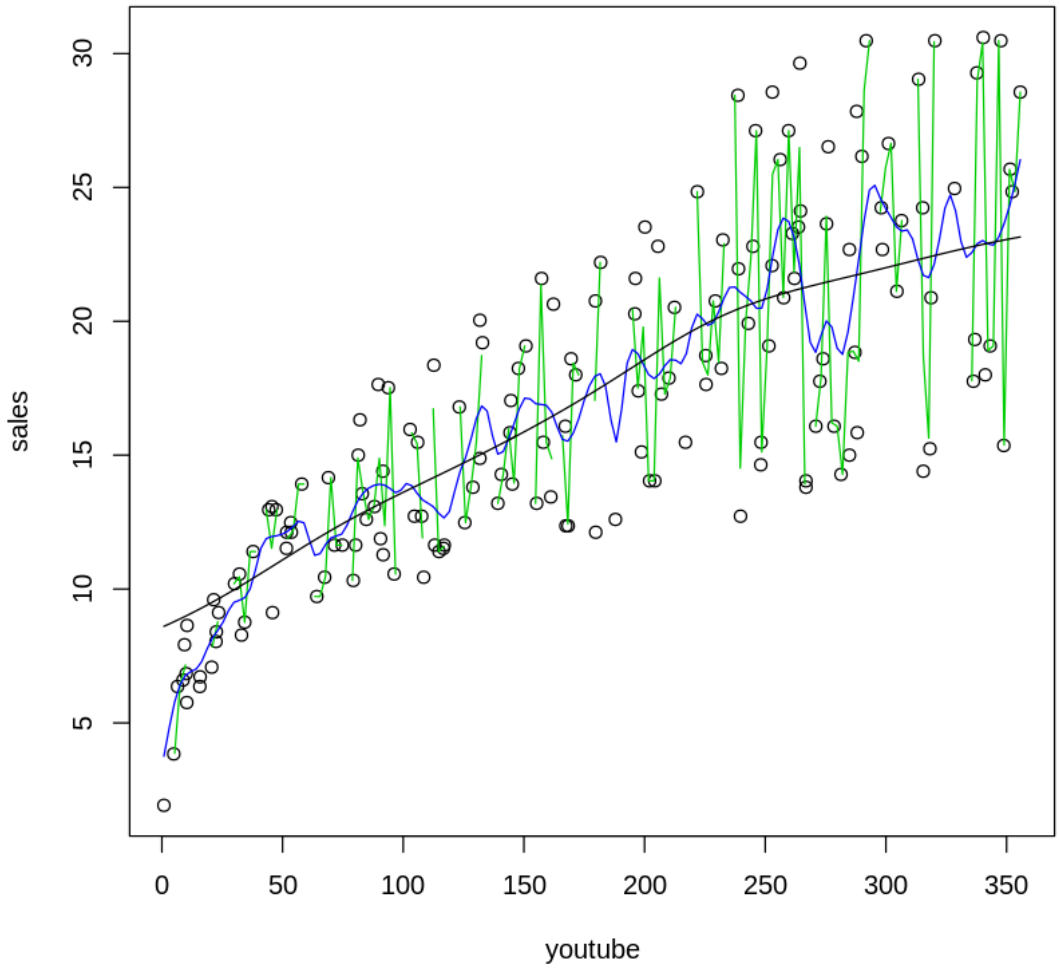
1. 160 2. 4

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.

[6]:
```
with(train_marketing, {
    plot(youtube, sales)
    lines(ksmooth(youtube, sales, kernel = "normal", bandwidth = 0.01), col = 2)
    lines(ksmooth(youtube, sales, kernel= "normal", bandwidth = 1), col = 3)
    lines(ksmooth(youtube, sales, kernel= "normal", bandwidth = 10), col = 4)
    lines(ksmooth(youtube, sales, kernel= "normal", bandwidth=100), col = 1)

})
```



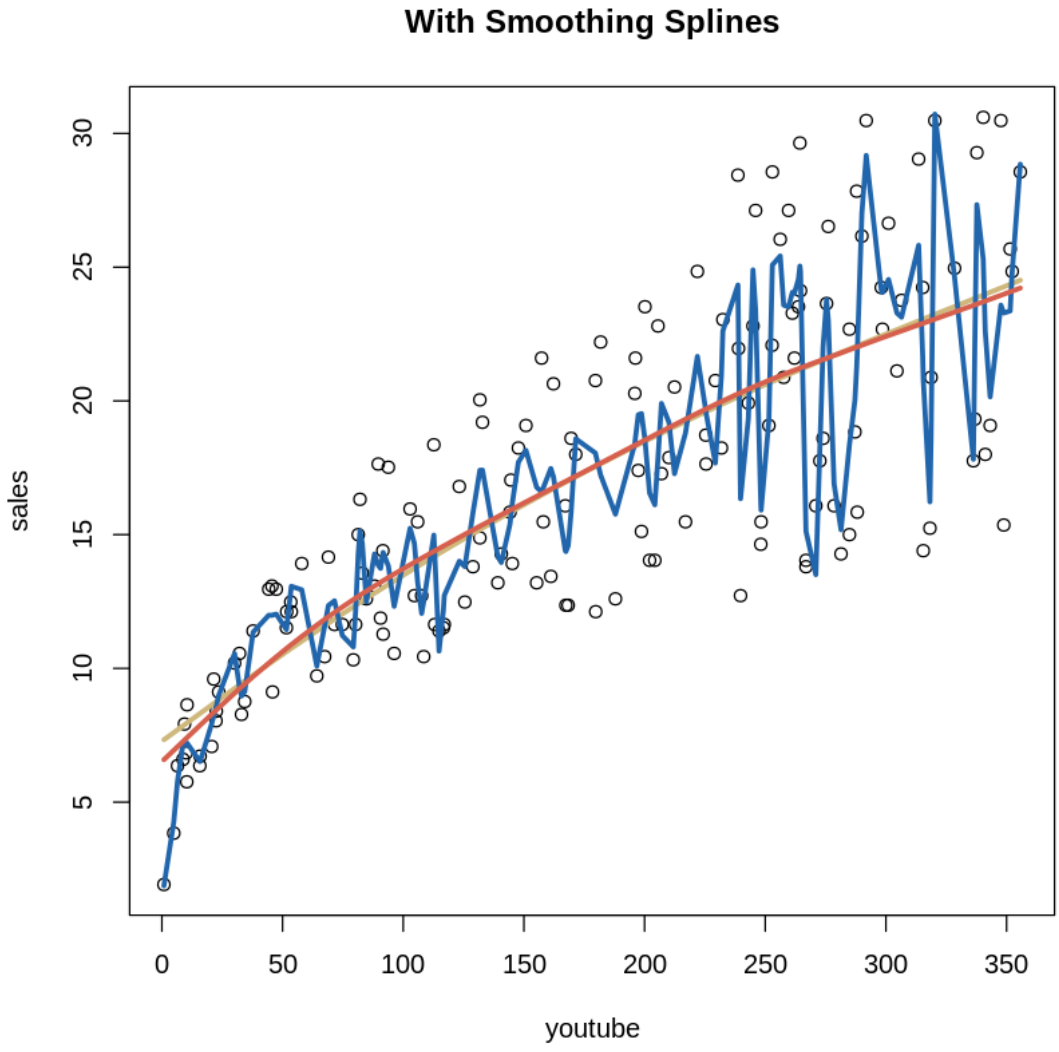
It looks like the kernel estimation method which is conceptually a kind of moving average suffers from the high variance in the data which is increasing as we go higher on the youtube spend scale. Even if we are experimenting with various bandwidth, the best MSPE is the average of the response variable.

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.

[7]:
```
smooth = smooth.spline(x=train_marketing$youtube,y = train_marketing$sales)
smooth2 = smooth.spline(x=train_marketing$youtube,y =␣
↪train_marketing$sales,spar=0.1)
```

```
smooth3 = smooth.spline(x=train_marketing$youtube,y =␣
↪train_marketing$sales,spar=1)
# your code here


plot(sales ~ youtube, data = train_marketing, main = "With Smoothing Splines")
lines(smooth, col = "#CFB87C", lwd=3)
lines(smooth2, col = "#2166AC", lwd=3)
lines(smooth3, col = "#D6604D", lwd=3)
```



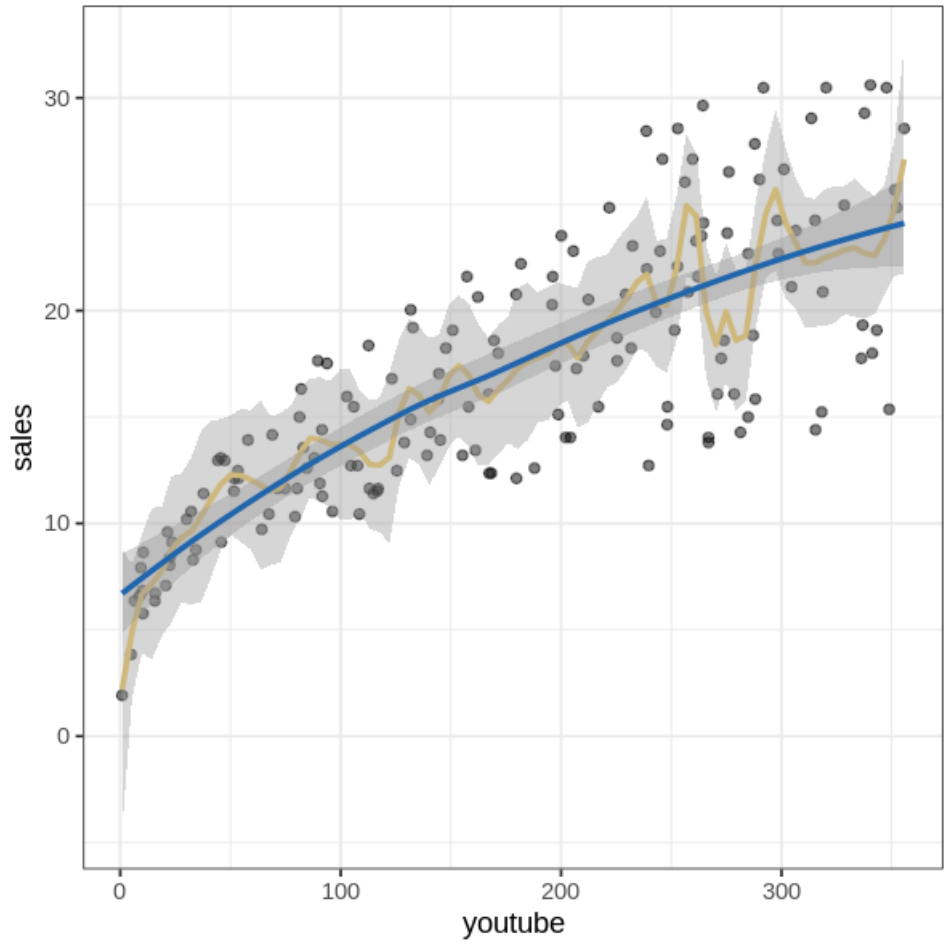
Smoothing splines look like a better alternative since it captures the upward non-linear trend, but penalize the model for very high curvature. The by default spar value gives a lot better MSPE than in the kernel estiamtion case.

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.

[8]:
```
options(repr.plot.width = 5, repr.plot.height = 5)
ggplot(train_marketing, aes(x = youtube, y = sales)) +
    geom_point(alpha = 0.5) +
    geom_smooth(method = "loess",col = "#CFB87C",span=0.1) +
    geom_smooth(method = "loess",col = "#2166AC",span=1) +
    #geom_smooth(method = "loess",col = "#D6604D")
    theme_bw()
```

```
`geom_smooth()` using formula 'y ~ x'

`geom_smooth()` using formula 'y ~ x'
```



[107]:
```
loess_marketing = loess(sales ~ youtube, data = train_marketing)
predict_loess = predict(loess_marketing, train_marketing$youtube)
```

Similarly to the smoothing splines method the increasing trend is captured well by the model again with reasonable wiggliness. This is expected as the method is close to linear regression models where you allow non-linearity in the data, but still accounting for linear relationships in the betas.

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).

[9]:
```
model_ksmooth = ksmooth(train_marketing$youtube, train_marketing$sales, kernel=␣
↪"normal", bandwidth=100)
```

[81]:
```
predict_ksmooth = ksmooth(x=test_marketing$youtube, y=test_marketing$sales,␣
↪kernel='box',bandwidth=10000)$y
```

[85]:
```
sum((test_marketing$sales - predict_ksmooth)^2)/length(test_marketing$sales)
```

```
Warning message in test_marketing$sales - predict_ksmooth:
"longer object length is not a multiple of shorter object length"
```

88.797258

[86]:
```
model_sp=smooth.spline(x=train_marketing$youtube,y = train_marketing$sales)
```

[87]:
```
predict_sp = predict(model_sp,test_marketing$youtube)
```

[88]:
```
sum((test_marketing$sales - predict_sp$y)^2)/length(test_marketing$sales)
```

17.5389997021376

[89]:
```
loess_marketing = loess(sales ~ youtube, data = train_marketing)
predict_loess = predict(loess_marketing, test_marketing$youtube)
```

[90]:
```
sum((test_marketing$sales - predict_loess)^2)/length(test_marketing$sales)
```

18.0402565819606

The best MSPE is for the smoothing splines closely followed by the Loess method. Kernel estimation stays behind.

[ ]:

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.

I have generated 10000 data points based on a simple periodic sin function with added standard Gaussian noise.

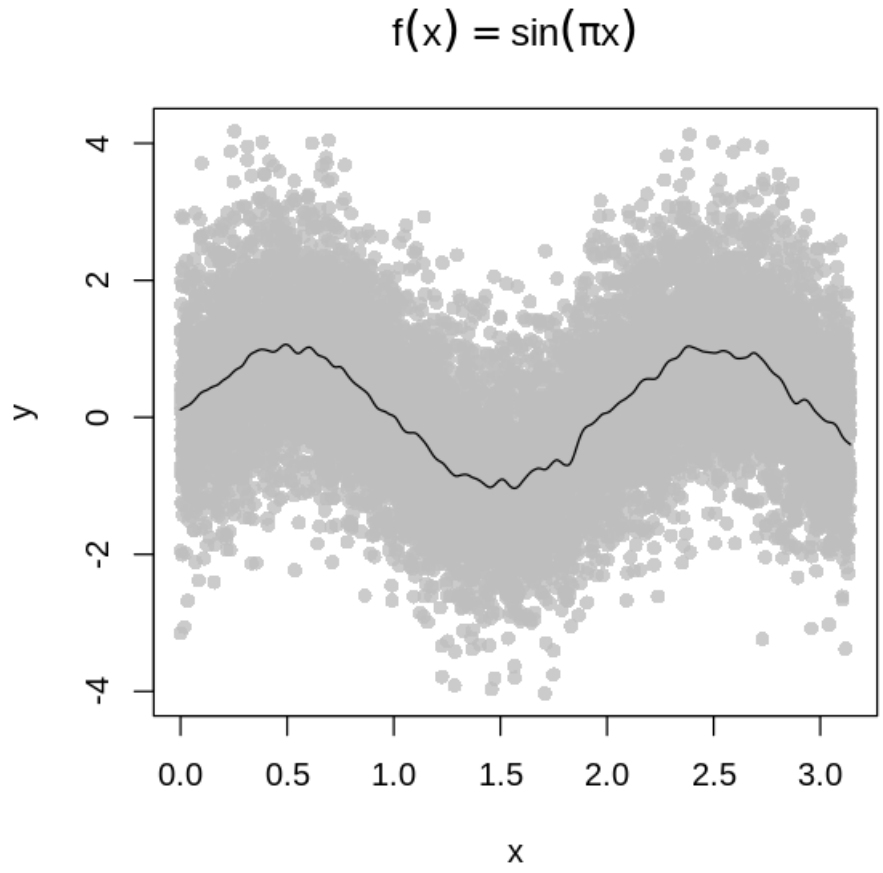
[91]:
```
#simulated data

set.seed(88888)

n = 10000
x = runif(n, 0, pi)
y = sin(pi*x) + rnorm(n, 0, 1)

df = data.frame(x = x, y = y)
head(df)
```

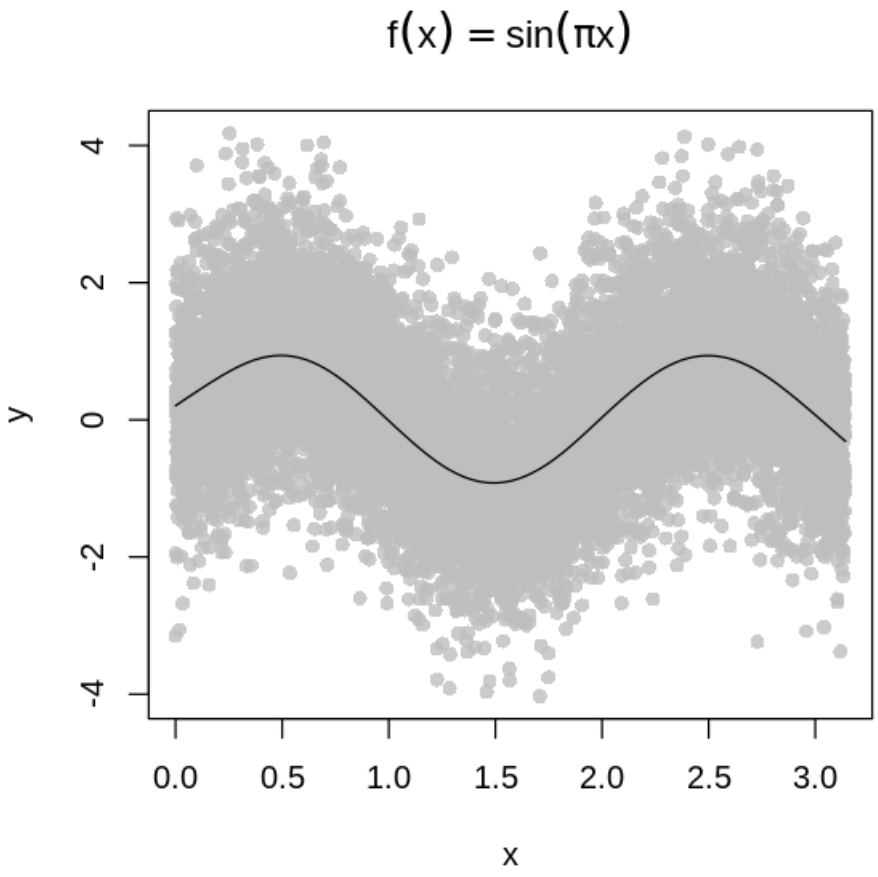
A data.frame: 6 × 2

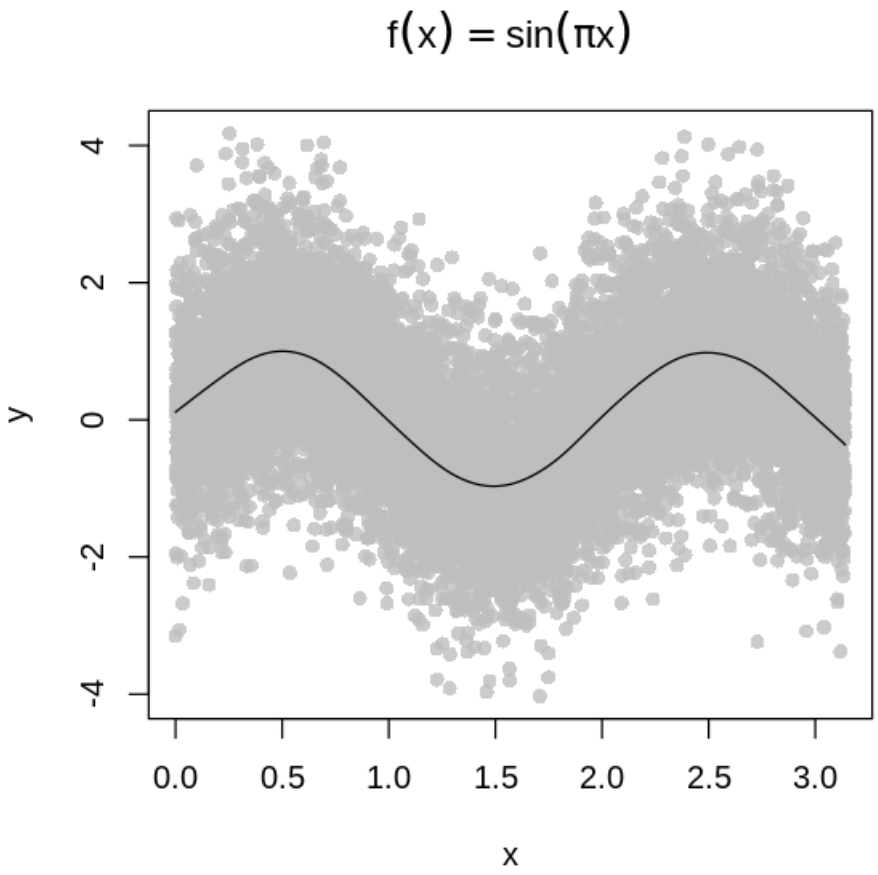
	x <dbl>	y <dbl>
1	0.7520398	-0.4668625
2	1.8033581	-0.5604260
3	2.4329813	-0.2149498
4	2.6215164	-1.1289802
5	0.8771843	0.1977056
6	2.0534615	-1.4692412

[92]:
```
library(ggplot2)
plot(y ~ x, main = expression(f(x) == sin(pi*x)), pch = 16, col = alpha("grey",␣
↪0.8))
lines(smooth.spline(x, y, spar = 0.5))

plot(y ~ x, main = expression(f(x) == sin(pi*x)), pch = 16, col = alpha("grey",␣
↪0.8))
lines(smooth.spline(x,y, spar = 1))

plot(y ~ x, main = expression(f(x) == sin(pi*x)), pch = 16, col = alpha("grey",␣
↪0.8))
```

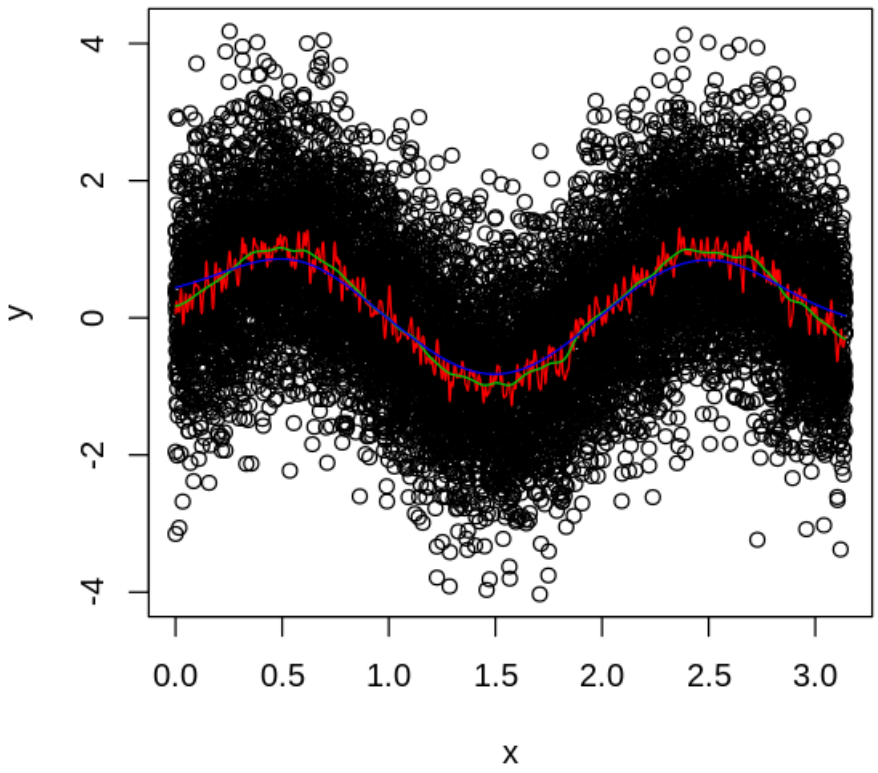
```
lines(smooth.spline(x,y))
```




f(x) = sin(πx)
y
4
2
0
-2
-4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x



[95]:
```
#1.a
plot(x, y)
    lines(ksmooth(x, y, kernel = "normal", bandwidth = 0.01), col = 2)
    lines(ksmooth(x, y, kernel = "normal", bandwidth = 0.1), col = 3)
    lines(ksmooth(x, y, kernel = "normal"), col = 4)
```



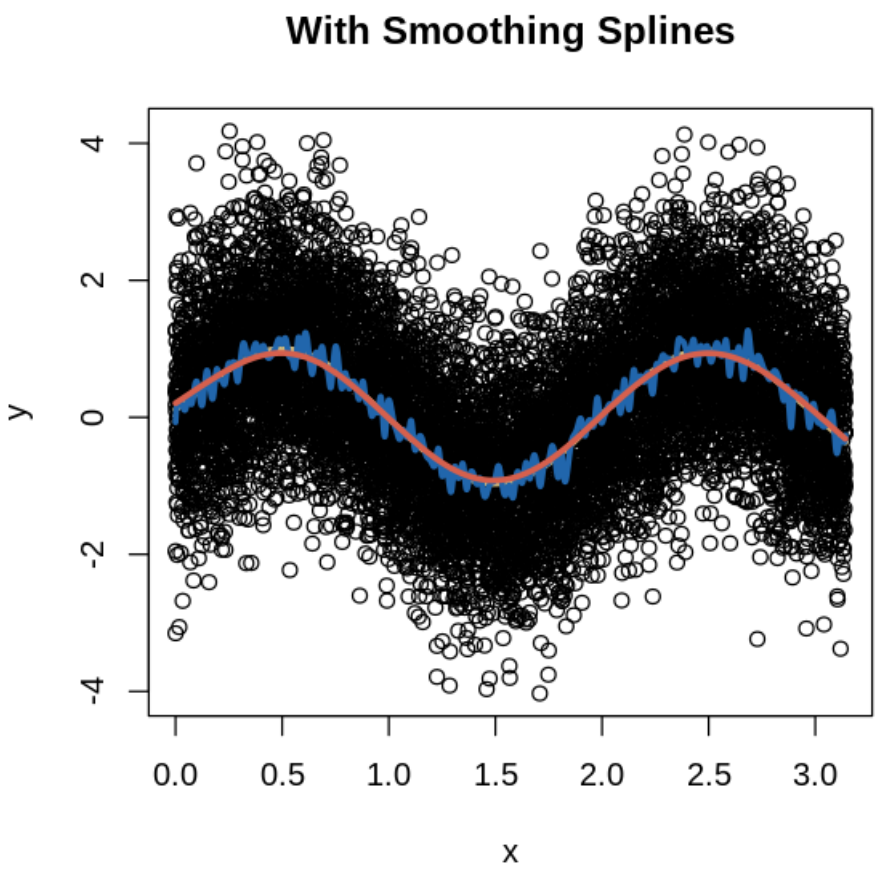
[97]:
```
ksmooth_own = ksmooth(x=x, y=y, kernel='normal')$y
sum((y - ksmooth_own)^2)/length(y)
```

1.74035498673092

[99]:
```
#1.b
smooth_own = smooth.spline(x = x, y = y)
smooth2_own = smooth.spline(x = x, y = y,spar=0.1)
smooth3_own = smooth.spline(x = x, y = y,spar=1)
# your code here


plot(y ~ x, main = "With Smoothing Splines")
lines(smooth_own, col = "#CFB87C", lwd=3)
lines(smooth2_own, col = "#2166AC", lwd=3)
lines(smooth3_own, col = "#D6604D", lwd=3)

sp_own=smooth.spline(x = x, y = y)
```

```
predict_sp_own = predict(sp_own,x)
sum((y - predict_sp_own$y)^2)/length(y)
```

1.00543909859376



[104]:
```
#1.c
options(repr.plot.width = 5, repr.plot.height = 5)
ggplot(df,aes(x = x, y = y)) +
    geom_point(alpha = 0.5) +
    geom_smooth(method = "loess",col = "#CFB87C",span=0.1) +
    geom_smooth(method = "loess",col = "#2166AC",span=1) +
    geom_smooth(method = "loess",col = "#D6604D")
    theme_bw()
```

```
`geom_smooth()` using formula 'y ~ x'

`geom_smooth()` using formula 'y ~ x'
```

```
`geom_smooth()` using formula 'y ~ x'


List of 93
 $ line                       :List of 6
  ..$ colour       : chr "black"
  ..$ size         : num 0.5
  ..$ linetype     : num 1
  ..$ lineend      : chr "butt"
  ..$ arrow        : logi FALSE
  ..$ inherit.blank: logi TRUE
  ..- attr(*, "class")= chr [1:2] "element_line" "element"
 $ rect                       :List of 5
  ..$ fill         : chr "white"
  ..$ colour       : chr "black"
  ..$ size         : num 0.5
  ..$ linetype     : num 1
  ..$ inherit.blank: logi TRUE
  ..- attr(*, "class")= chr [1:2] "element_rect" "element"
 $ text                       :List of 11
  ..$ family       : chr ""
  ..$ face         : chr "plain"
  ..$ colour       : chr "black"
  ..$ size         : num 11
  ..$ hjust        : num 0.5
  ..$ vjust        : num 0.5
  ..$ angle        : num 0
  ..$ lineheight   : num 0.9
  ..$ margin       : 'margin' num [1:4] 0pt 0pt 0pt 0pt
  .. ..- attr(*, "valid.unit")= int 8
  .. ..- attr(*, "unit")= chr "pt"
  ..$ debug        : logi FALSE
  ..$ inherit.blank: logi TRUE
  ..- attr(*, "class")= chr [1:2] "element_text" "element"
 $ title                      : NULL
 $ aspect.ratio               : NULL
 $ axis.title                 : NULL
 $ axis.title.x               :List of 11
  ..$ family       : NULL
  ..$ face         : NULL
  ..$ colour       : NULL
  ..$ size         : NULL
  ..$ hjust        : NULL
  ..$ vjust        : num 1
  ..$ angle        : NULL
  ..$ lineheight   : NULL
  ..$ margin       : 'margin' num [1:4] 2.75pt 0pt 0pt 0pt
  .. ..- attr(*, "valid.unit")= int 8
```

```
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ axis.title.x.top          :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : NULL
 ..$ vjust        : num 0
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 0pt 0pt 2.75pt 0pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ axis.title.x.bottom       : NULL
$ axis.title.y              :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : NULL
 ..$ vjust        : num 1
 ..$ angle        : num 90
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 0pt 2.75pt 0pt 0pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ axis.title.y.left         : NULL
$ axis.title.y.right        :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : NULL
 ..$ vjust        : num 0
 ..$ angle        : num -90
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 0pt 0pt 0pt 2.75pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
```

```
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ axis.text                 :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : chr "grey30"
 ..$ size         : 'rel' num 0.8
 ..$ hjust        : NULL
 ..$ vjust        : NULL
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : NULL
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ axis.text.x               :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : NULL
 ..$ vjust        : num 1
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 2.2pt 0pt 0pt 0pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ axis.text.x.top           :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : NULL
 ..$ vjust        : num 0
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 0pt 0pt 2.2pt 0pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ axis.text.x.bottom        : NULL
$ axis.text.y               :List of 11
```

```
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : num 1
 ..$ vjust        : NULL
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 0pt 2.2pt 0pt 0pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ axis.text.y.left          : NULL
$ axis.text.y.right         :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : num 0
 ..$ vjust        : NULL
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 0pt 0pt 0pt 2.2pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ axis.ticks                :List of 6
 ..$ colour       : chr "grey20"
 ..$ size         : NULL
 ..$ linetype     : NULL
 ..$ lineend      : NULL
 ..$ arrow        : logi FALSE
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_line" "element"
$ axis.ticks.x              : NULL
$ axis.ticks.x.top          : NULL
$ axis.ticks.x.bottom       : NULL
$ axis.ticks.y              : NULL
$ axis.ticks.y.left         : NULL
$ axis.ticks.y.right        : NULL
$ axis.ticks.length         : 'unit' num 2.75pt
 ..- attr(*, "valid.unit")= int 8
 ..- attr(*, "unit")= chr "pt"
$ axis.ticks.length.x       : NULL
```

```
$ axis.ticks.length.x.top   : NULL
$ axis.ticks.length.x.bottom: NULL
$ axis.ticks.length.y       : NULL
$ axis.ticks.length.y.left  : NULL
$ axis.ticks.length.y.right : NULL
$ axis.line                 : list()
 ..- attr(*, "class")= chr [1:2] "element_blank" "element"
$ axis.line.x               : NULL
$ axis.line.x.top           : NULL
$ axis.line.x.bottom        : NULL
$ axis.line.y               : NULL
$ axis.line.y.left          : NULL
$ axis.line.y.right         : NULL
$ legend.background         :List of 5
 ..$ fill         : NULL
 ..$ colour       : logi NA
 ..$ size         : NULL
 ..$ linetype     : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_rect" "element"
$ legend.margin             : 'margin' num [1:4] 5.5pt 5.5pt 5.5pt 5.5pt
 ..- attr(*, "valid.unit")= int 8
 ..- attr(*, "unit")= chr "pt"
$ legend.spacing            : 'unit' num 11pt
 ..- attr(*, "valid.unit")= int 8
 ..- attr(*, "unit")= chr "pt"
$ legend.spacing.x          : NULL
$ legend.spacing.y          : NULL
$ legend.key                :List of 5
 ..$ fill         : chr "white"
 ..$ colour       : logi NA
 ..$ size         : NULL
 ..$ linetype     : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_rect" "element"
$ legend.key.size           : 'unit' num 1.2lines
 ..- attr(*, "valid.unit")= int 3
 ..- attr(*, "unit")= chr "lines"
$ legend.key.height         : NULL
$ legend.key.width          : NULL
$ legend.text               :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : 'rel' num 0.8
 ..$ hjust        : NULL
 ..$ vjust        : NULL
 ..$ angle        : NULL
```

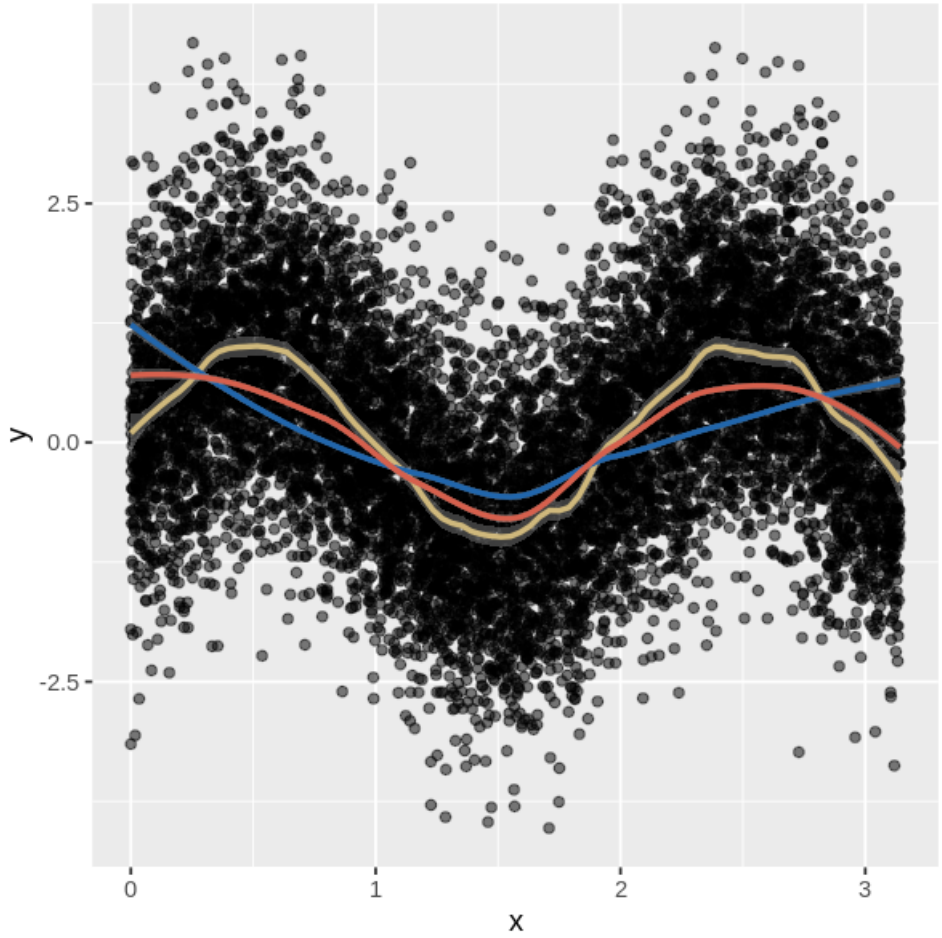
```
 ..$ lineheight   : NULL
 ..$ margin       : NULL
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ legend.text.align         : NULL
$ legend.title              :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : num 0
 ..$ vjust        : NULL
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : NULL
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ legend.title.align        : NULL
$ legend.position           : chr "right"
$ legend.direction          : NULL
$ legend.justification      : chr "center"
$ legend.box                : NULL
$ legend.box.just           : NULL
$ legend.box.margin         : 'margin' num [1:4] 0cm 0cm 0cm 0cm
 ..- attr(*, "valid.unit")= int 1
 ..- attr(*, "unit")= chr "cm"
$ legend.box.background     : list()
 ..- attr(*, "class")= chr [1:2] "element_blank" "element"
$ legend.box.spacing        : 'unit' num 11pt
 ..- attr(*, "valid.unit")= int 8
 ..- attr(*, "unit")= chr "pt"
$ panel.background          :List of 5
 ..$ fill         : chr "white"
 ..$ colour       : logi NA
 ..$ size         : NULL
 ..$ linetype     : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_rect" "element"
$ panel.border              :List of 5
 ..$ fill         : logi NA
 ..$ colour       : chr "grey20"
 ..$ size         : NULL
 ..$ linetype     : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_rect" "element"
$ panel.spacing             : 'unit' num 5.5pt
```

```
 ..- attr(*, "valid.unit")= int 8
 ..- attr(*, "unit")= chr "pt"
$ panel.spacing.x           : NULL
$ panel.spacing.y           : NULL
$ panel.grid                :List of 6
 ..$ colour       : chr "grey92"
 ..$ size         : NULL
 ..$ linetype     : NULL
 ..$ lineend      : NULL
 ..$ arrow        : logi FALSE
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_line" "element"
$ panel.grid.major          : NULL
$ panel.grid.minor          :List of 6
 ..$ colour       : NULL
 ..$ size         : 'rel' num 0.5
 ..$ linetype     : NULL
 ..$ lineend      : NULL
 ..$ arrow        : logi FALSE
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_line" "element"
$ panel.grid.major.x        : NULL
$ panel.grid.major.y        : NULL
$ panel.grid.minor.x        : NULL
$ panel.grid.minor.y        : NULL
$ panel.ontop               : logi FALSE
$ plot.background           :List of 5
 ..$ fill         : NULL
 ..$ colour       : chr "white"
 ..$ size         : NULL
 ..$ linetype     : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_rect" "element"
$ plot.title                :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : 'rel' num 1.2
 ..$ hjust        : num 0
 ..$ vjust        : num 1
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 0pt 0pt 5.5pt 0pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
```

```
$ plot.title.position       : chr "panel"
$ plot.subtitle             :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : num 0
 ..$ vjust        : num 1
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 0pt 0pt 5.5pt 0pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ plot.caption              :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : 'rel' num 0.8
 ..$ hjust        : num 1
 ..$ vjust        : num 1
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 5.5pt 0pt 0pt 0pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ plot.caption.position     : chr "panel"
$ plot.tag                  :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : 'rel' num 1.2
 ..$ hjust        : num 0.5
 ..$ vjust        : num 0.5
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : NULL
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ plot.tag.position         : chr "topleft"
$ plot.margin               : 'margin' num [1:4] 5.5pt 5.5pt 5.5pt 5.5pt
 ..- attr(*, "valid.unit")= int 8
```

```
 ..- attr(*, "unit")= chr "pt"
$ strip.background          :List of 5
 ..$ fill         : chr "grey85"
 ..$ colour       : chr "grey20"
 ..$ size         : NULL
 ..$ linetype     : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_rect" "element"
$ strip.background.x        : NULL
$ strip.background.y        : NULL
$ strip.placement           : chr "inside"
$ strip.text                :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : chr "grey10"
 ..$ size         : 'rel' num 0.8
 ..$ hjust        : NULL
 ..$ vjust        : NULL
 ..$ angle        : NULL
 ..$ lineheight   : NULL
 ..$ margin       : 'margin' num [1:4] 4.4pt 4.4pt 4.4pt 4.4pt
 .. ..- attr(*, "valid.unit")= int 8
 .. ..- attr(*, "unit")= chr "pt"
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ strip.text.x              : NULL
$ strip.text.y              :List of 11
 ..$ family       : NULL
 ..$ face         : NULL
 ..$ colour       : NULL
 ..$ size         : NULL
 ..$ hjust        : NULL
 ..$ vjust        : NULL
 ..$ angle        : num -90
 ..$ lineheight   : NULL
 ..$ margin       : NULL
 ..$ debug        : NULL
 ..$ inherit.blank: logi TRUE
 ..- attr(*, "class")= chr [1:2] "element_text" "element"
$ strip.switch.pad.grid     : 'unit' num 2.75pt
 ..- attr(*, "valid.unit")= int 8
 ..- attr(*, "unit")= chr "pt"
$ strip.switch.pad.wrap     : 'unit' num 2.75pt
 ..- attr(*, "valid.unit")= int 8
 ..- attr(*, "unit")= chr "pt"
$ strip.text.y.left         :List of 11
 ..$ family       : NULL
```

```
  ..$ face          : NULL
  ..$ colour        : NULL
  ..$ size          : NULL
  ..$ hjust         : NULL
  ..$ vjust         : NULL
  ..$ angle         : num 90
  ..$ lineheight    : NULL
  ..$ margin        : NULL
  ..$ debug         : NULL
  ..$ inherit.blank: logi TRUE
  ..- attr(*, "class")= chr [1:2] "element_text" "element"
 - attr(*, "class")= chr [1:2] "theme" "gg"
 - attr(*, "complete")= logi TRUE
 - attr(*, "validate")= logi TRUE
```



[105]:
```
loess_own = loess(y ~ x, data = df)
predict_loess_own = predict(loess_own, x)
```

```
sum((y - predict_loess_own)^2)/length(y)
```

1.08050785015133

[4]:
```
#1.d
```

The overall conclusions are very similar to what we reached in the advertising budget case. Kernel smoothing with moving average concept stays behing smoothing splines and Loess methods which two are closely comparable.

[ ]:
```

```