C3M3: Peer Reviewed Assignment

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.

In [172]:
```
# Load Required Packages
library(ggplot2)
library(mgcv)
```

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:

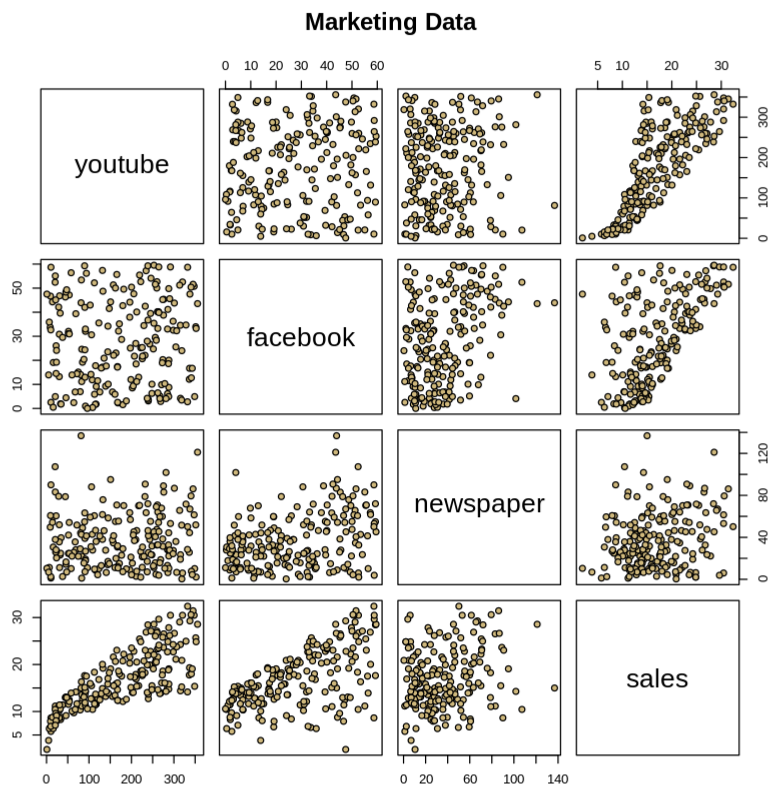
- `youtube` : the advertising budget allocated to YouTube. Measured in thousands of dollars;
- `facebook` : the advertising budget allocated to Facebook. Measured in thousands of dollars; and
- `newspaper` : the advertising budget allocated to a local newspaper. Measured in thousands of dollars.
- `sales` : the value in the ith 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`).

In [173]:
```
# 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  
```



In [174]:
```
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 indici
es to be included in the training set

train_marketing = marketing[index, ] #set the training set to be the rando
mly sampled rows of the dataframe
test_marketing = marketing[-index, ] #set the testing set to be the remain
ing rows
dim(test_marketing) #check the dimensions
dim(train_marketing) #check the dimensions
```

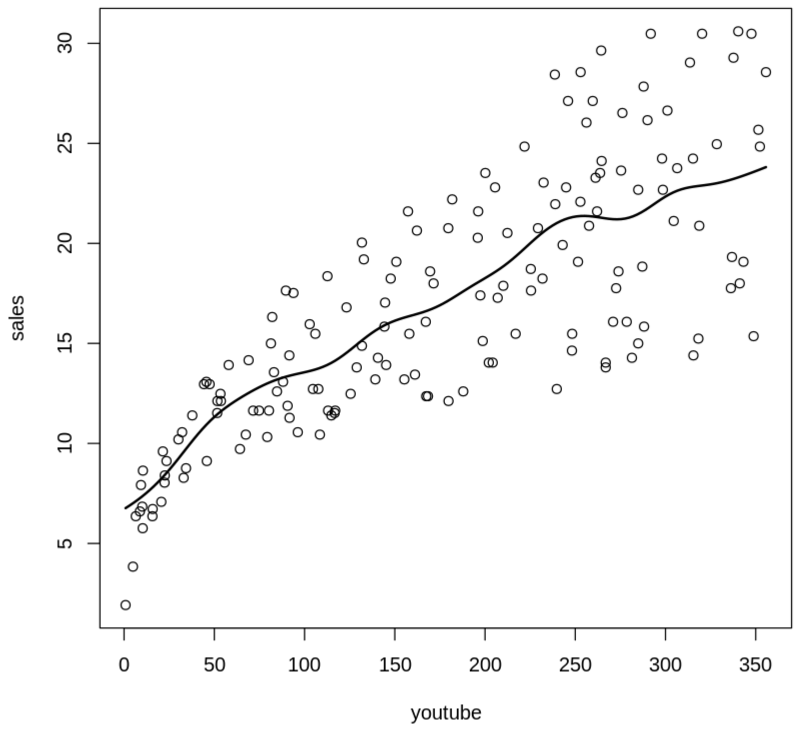
```
40 · 4

160 · 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.

In [175]:
```
z = ksmooth(train_marketing$youtube, train_marketing$sales, kernel = "norm
al", 45)
plot(sales ~ youtube, data = train_marketing)
lines(z, lwd = 2)
```



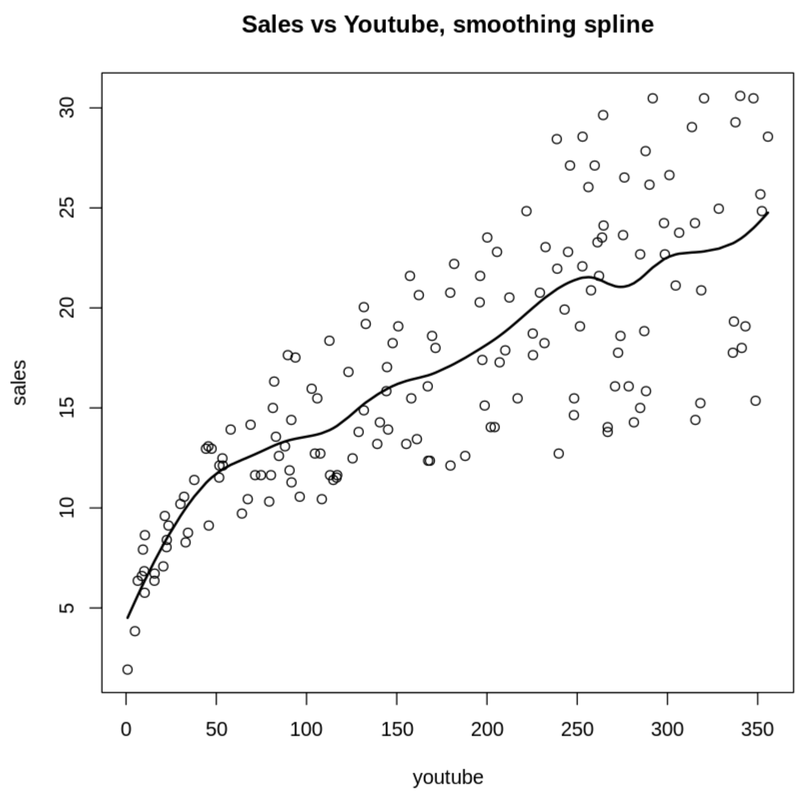
The smoothing process could not be 'perfected', so I attempted to try to shape the line to resemble the curvy/non-linear relationship while avoiding having too much 'curliness'. The result shows smoothing overall, with slightly more intense curls towards the upper right, or the higher number regions of both sales and youtube.

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.

In [208]:
```
y_sales = train_marketing$sales
x_youtube = train_marketing$youtube
spl = smooth.spline(y = train_marketing$sales, x = train_marketing$youtube
, spar = 0.73)
```

```
plot(sales ~ youtube, data = train_marketing, main = "Sales vs Youtube, sm
oothing spline")
lines(spl, lwd = 2)
```



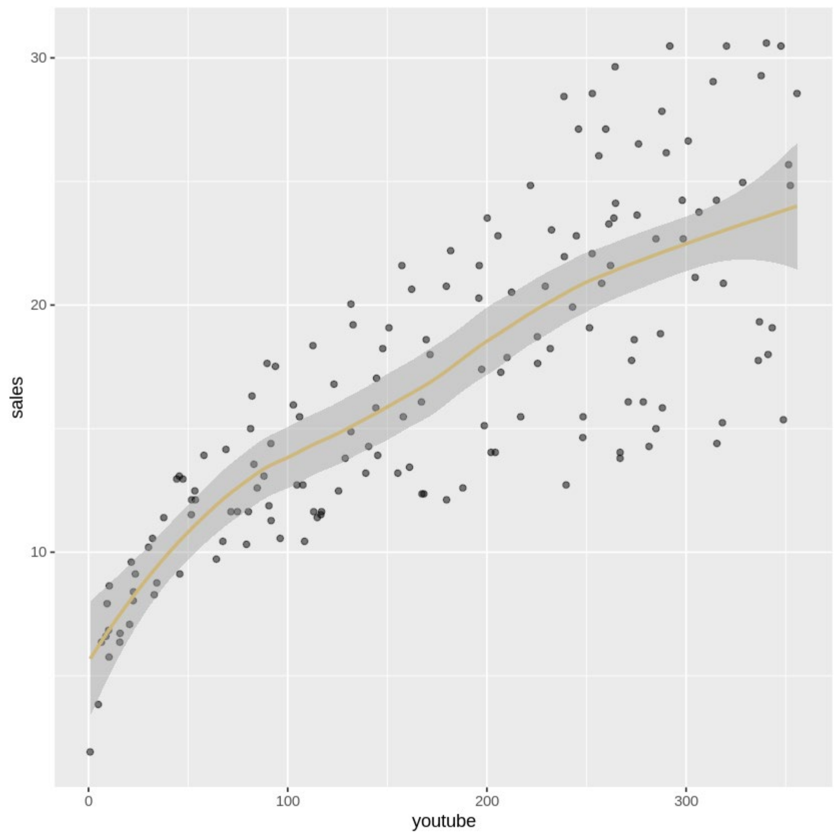
This produced a better result compared to the kernel regression. If 'spar' parameter is set lower, the curliness is significantly more dramatic. It becomes less apparent as the parameter value is increased, at which 0.8 seems to be the optimal value. We can see that the line explicitly shows varying steepness in slope as we move up on the youtube axis, starting off steeply positive and slightly shallowing out to some curly portions at the higher sales and youtube regions.

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.

In [209]:
```
loessreg1 = loess(sales ~ youtube, data = train_marketing, span = 0.35)
ggplot(train_marketing, aes(youtube, sales)) + geom_point(alpha = 0.5) + g
eom_smooth(method = "loess", color = "#CFB87C", span = 0.6) +
    xlab("youtube") +
    ylab("sales")
```

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



Adjusting the span parameter to 0.6 slightly improves smoothing, while keeping the apparent non-linearity at x ~ 80. At about x=80 we can still see the sudden change in slope while having a smooth upper-right region.

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 = $1kk\sum i=1\left(y_{\star i} - \hat{y}_{\star i}\right)^2$

where $y_{\star i}$ are the observed response values in the test set and $\hat{y}_{\star i}$ are the predicted values for the test set (using the model fit on the training set).

In [210]:
```
#x_youtube, y_sales
#kernel : z = ksmooth(train_marketing$youtube, train_marketing$sales, kern
el = "normal", 45)\
#spline: spl = smooth.spline(y = y_sales, x = x_youtube, spar = 0.8)
#loess: lossmod1 = loess(sales ~ youtube, data = train_marketing, span = 0
.35)

z2 = ksmooth(train_marketing$youtube, train_marketing$sales, kernel = "nor
mal", 45, x.points = test_marketing$youtube)
MSPE_kernel = sum((test_marketing$sales - z2$y)^2)/length(z2$y)
MSPE_kernel
```

```
spl2 = predict(spl, x = test_marketing$youtube)
MSPE_spl = sum((test_marketing$sales - spl2$y)^2)/length(spl2$y)
MSPE_spl

loessreg2 = predict(loessreg1, newdata = test_marketing$youtube)
MSPE_loess = sum((test_marketing$sales - loessreg2)^2)/length(loessreg2)
MSPE_loess
```

```
64.0929792868167

18.144205022685

18.1326266788043
```

The MSPE for kernel, smoothing spline, and loess regression are 64.09, 18.14, and 18.13, respectively. The MSPE of loess regression is the lowest among the three, although extremely close to that of smoothing spline regression.

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 am generating random data similar to the simulation method used in the "Loess Regression" lab, with data points following the cubic function $y = x^3$

In [211]:
```
#simulated data

set.seed(41412)
n = 250
x = runif(n, 0, pi/2)
y = cos((pi/2)*x) + rnorm(n, 0, 0.5) + 4
df = data.frame(x = x, y = y)
head(df)

set.seed(1111)
n = floor(0.8 * nrow(df))
index = sample(seq_len(nrow(df)), size = n)

train_data = df[index, ]
test_data = df[-index, ]
dim(test_data)
dim(train_data)

head(test_data)
head(train_data)

plot(y ~ x, main = expression(f(x) == cos((pi/2)*x)))

lsreg = loess(y~x, control=loess.control(surface = "direct"), train_data,
span = 0.5)
```

```
lsreg
```

A data.frame: 6 × 2

	x	y
	<dbl>	<dbl>
1	1.4939664	2.716523
2	1.1147989	3.969268
3	1.3306610	3.863216
4	0.8171156	4.925973
5	1.4298908	3.665921
6	1.5549821	3.958761

50 · 2

200 · 2

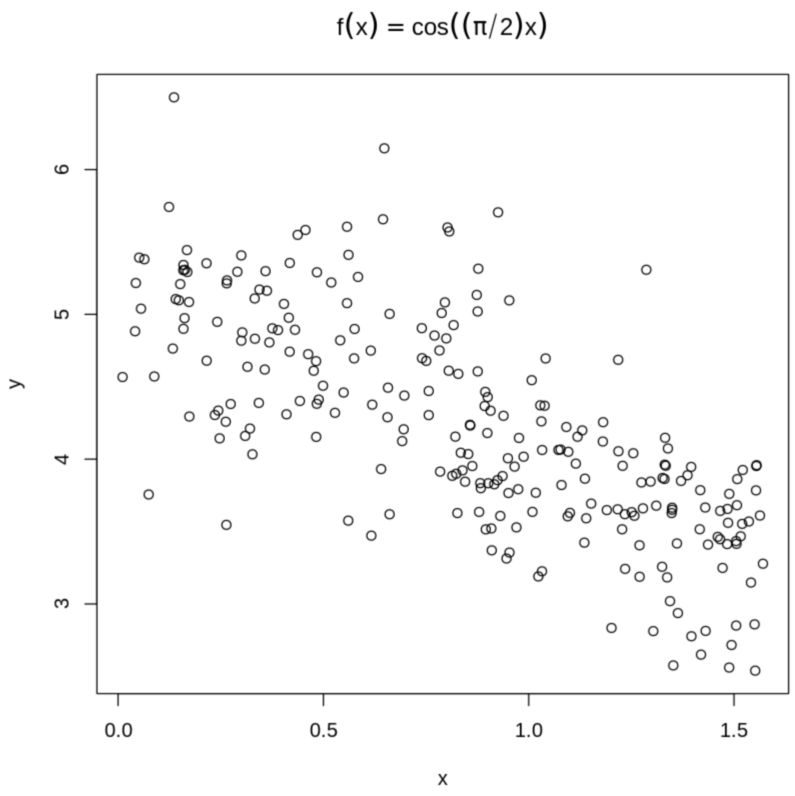
A data.frame: 6 × 2

	x	y
	<dbl>	<dbl>
3	1.3306610	3.863216
8	1.4178066	3.786010
28	0.3443584	5.171222
29	0.9763008	4.146514
35	0.8735083	5.134278
58	1.1521404	3.693163

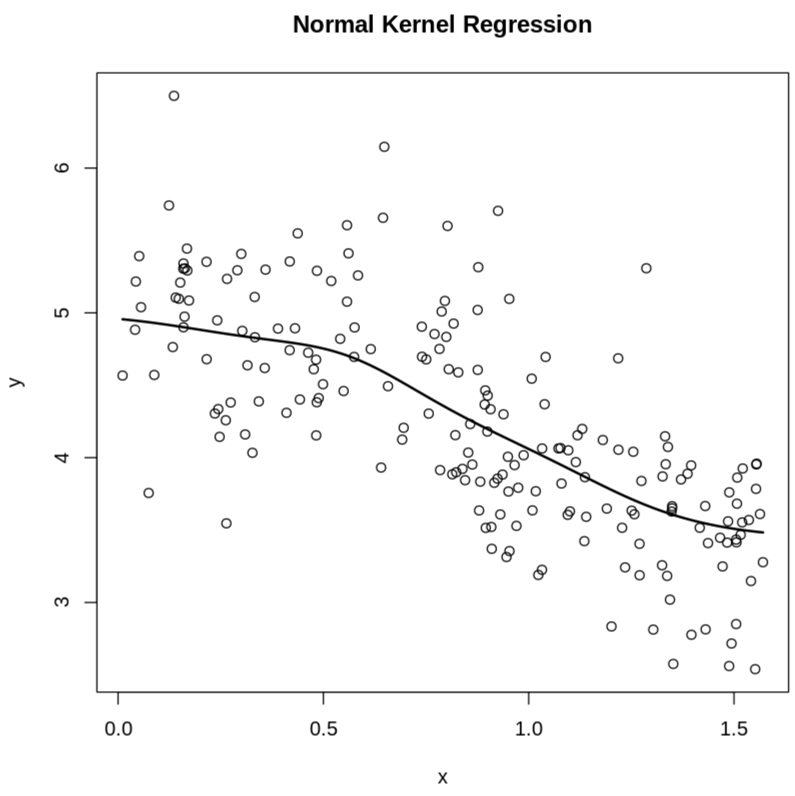
A data.frame: 6 × 2

	x	y
	<dbl>	<dbl>
44	0.30246065	4.875751
182	1.33364901	3.954559
50	0.87721083	5.315516
26	1.03268905	4.063355
36	0.32722317	4.033629
241	0.08822678	4.571223

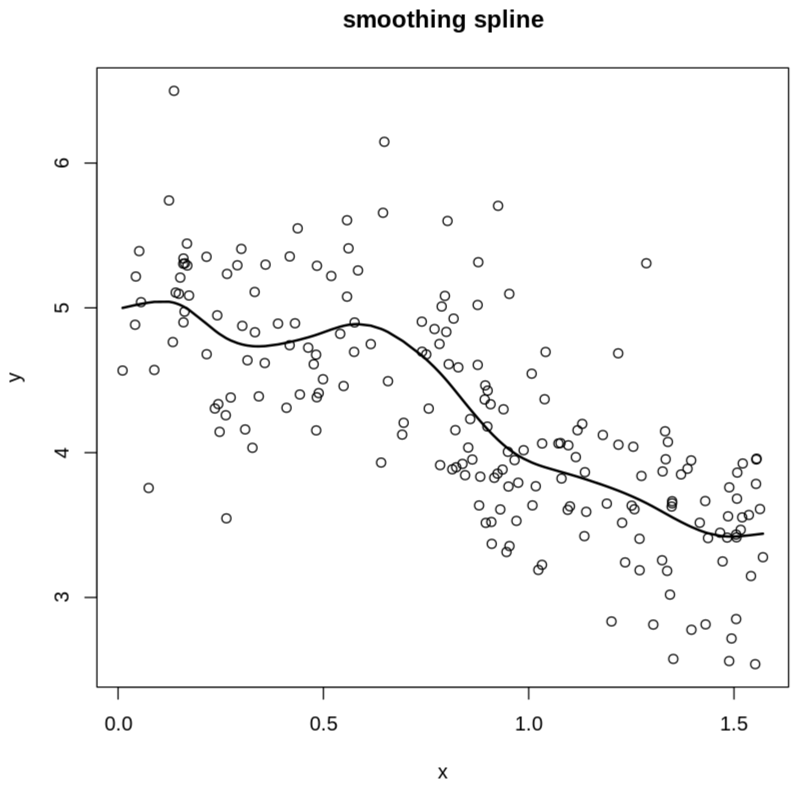
```
Call:
loess(formula = y ~ x, data = train_data, span = 0.5, control = loess.cont
rol(surface = "direct"))

Number of Observations: 200
Equivalent Number of Parameters: 6.32
Residual Standard Error: 0.5146
```



In [212]:
```
#1.a
kline = ksmooth(train_data$x, train_data$y, kernel = 'normal', 0.5)
plot(y ~ x, data = train_data, main = "Normal Kernel Regression")
lines(kline, lwd = 2)
```

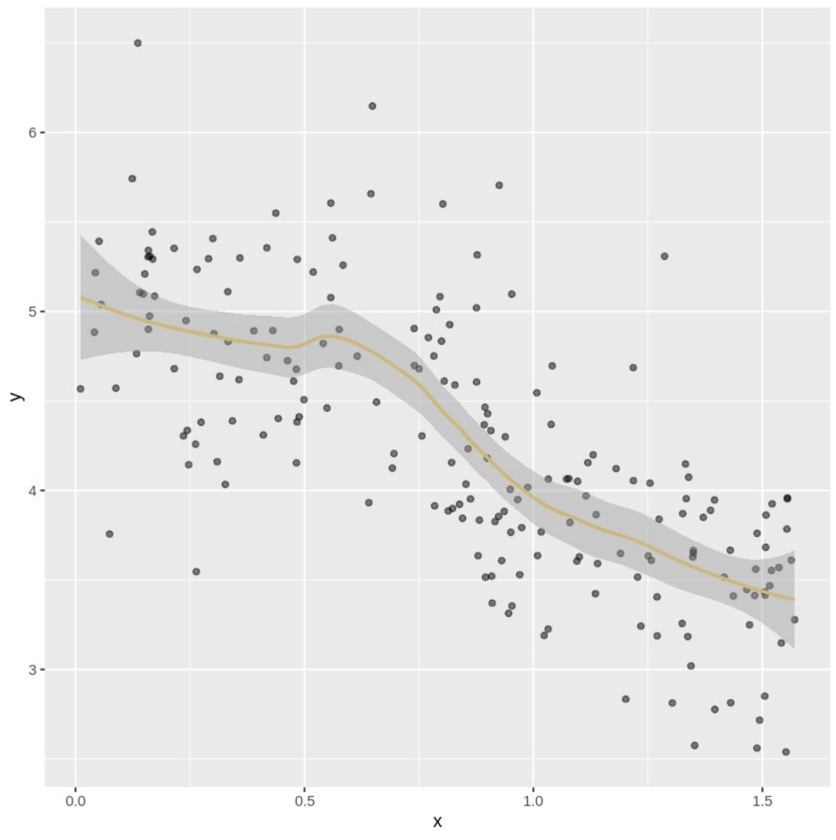


In [213]:
```
#1.b
y_data = train_data$y
x_data = train_data$x
spl_data = smooth.spline(y = y_data, x = x_data, spar = 0.9)
plot(y ~ x, data = train_data, main = "smoothing spline")
lines(spl_data, lwd = 2)
```



```
In [214]: #1.c
          ggplot(train_data, aes(x, y)) + geom_point(alpha = 0.5) + geom_smooth(meth
          od = "loess", color = "#CFB87C", span = 0.6) +
              xlab("x") +
              ylab("y")
```

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



```
In [215]: #1.d
          #loss_data = loess(y ~ x, data = train_data, span = 0.35)
          spl_data = smooth.spline(y = y_data, x = x_data, spar = 0.8)

          z3 = ksmooth(train_data$x, train_data$y, kernel = "normal", 45, x.points =
           test_data$x)
          MSPE_kernel_data = sum((test_data$y - z3$y)^2)/length(z3$y)
          MSPE_kernel_data

          spl_data2 = predict(spl_data, x = test_data$x)
          MSPE_spl_data = sum((test_data$y - spl_data2$y)^2)/length(spl_data2$y)
          MSPE_spl_data

          loss_data2 = predict(lsreg, newdata = test_data$x)
          #MSPE_loess_data = sum((test_data$y - lossmod_data2)^2)/length(lossmod_dat
          a2)
          lsregmspe = mean((test_data$y - loss_data2)^2)
          lsregmspe
```

```
0.480957119866116

0.294210589726354

0.279612240040191
```

The results are similar to the previous problem. The loess regression is the best with an MSPE of 0.28, followed by smoothing spline's 0.29, and finally 0.48 for the kernel regression.

```
In [ ]:
```

```
In [ ]:
```