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

```
Loading required package: nlme

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

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:

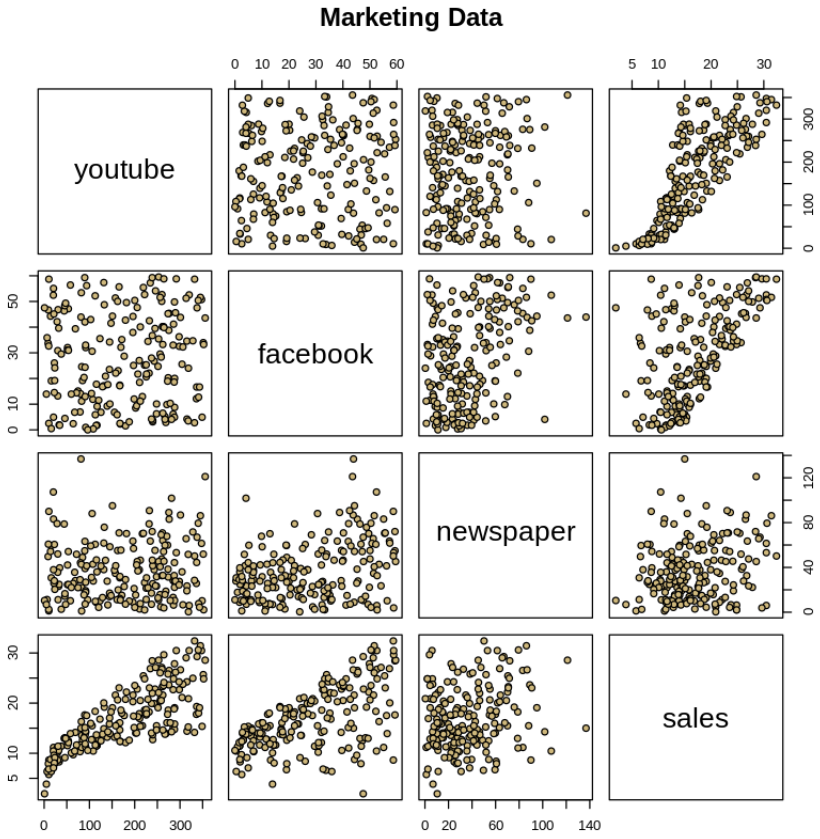
- `youtube` : the advertising budget allocated to YouTube. Measured in thousands of dollars;
- `facebook` : the advertising budget allocated to Facebook. Measured in thousands of dollars; and
- `newspaper` : the advertising budget allocated to a local newspaper. Measured in thousands of dollars.
- `sales`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

In [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  
```



In [3]:
```
set.seed(1771) #set the random number generator seed.
n = floor(0.8 * nrow(marketing)) #find the number corresponding to 80%
index = sample(seq_len(nrow(marketing)), size = n) #randomly sample in

train_marketing = marketing[index, ] #set the training set to be the r
test_marketing = marketing[-index, ] #set the testing set to be the re
dim(test_marketing) #check the dimensions
dim(train_marketing) #check the dimensions
```

40 · 4

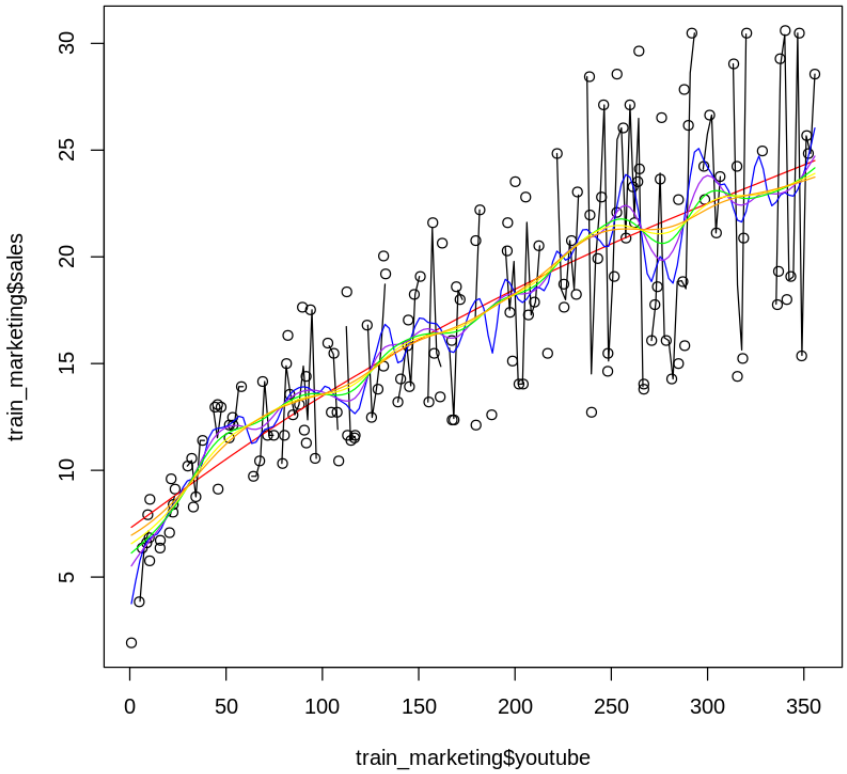
160 · 4

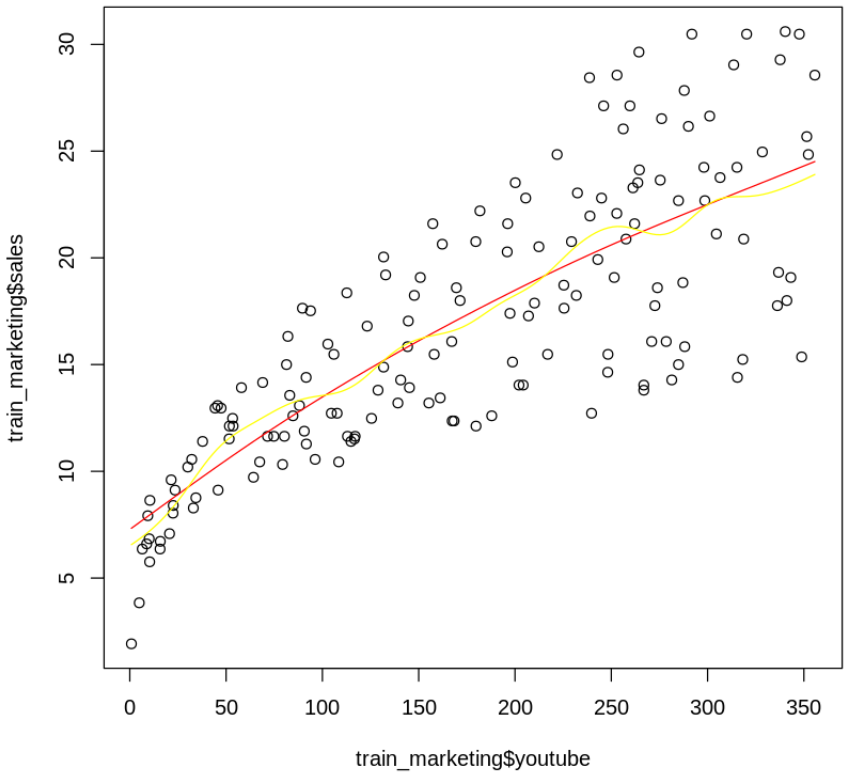
1.(a) Working with nonlinearity: Kernel regression

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 [4]:
```
ssp = smooth.spline(train_marketing$youtube,train_marketing$sales)
ssk2 = ksmooth(train_marketing$youtube,train_marketing$sales,'normal',

plot(train_marketing$youtube,train_marketing$sales)
lines(ssp, col="red")
lines(ssk2$x,ssk2$y, col="black")

ssk3 = ksmooth(train_marketing$youtube,train_marketing$sales,'normal',
ssk4 = ksmooth(train_marketing$youtube,train_marketing$sales,'normal',
ssk5 = ksmooth(train_marketing$youtube,train_marketing$sales,'normal',
ssk6 = ksmooth(train_marketing$youtube,train_marketing$sales,'normal',
ssk7 = ksmooth(train_marketing$youtube,train_marketing$sales,'normal',
lines(ssk3$x,ssk3$y, col="blue")
lines(ssk4$x,ssk4$y, col="purple")
lines(ssk5$x,ssk5$y, col="green")
lines(ssk6$x,ssk6$y, col="yellow")
lines(ssk7$x,ssk7$y, col="orange")

plot(train_marketing$youtube,train_marketing$sales)
lines(ssp, col="red")
lines(ssk6$x,ssk6$y, col="yellow")
```


train_marketing$sales
train_marketing$youtube
0
50
100
150
200
250
300
350
5
10
15
20
25
30


train_marketing$sales
train_marketing$youtube
0
50
100
150
200
250
300
350
5
10
15
20
25
30

This data is incredibly dense. More so than anything we've worked with prior. As such, the bandwidth aperature must be expanded considerably to get kernels to smooth the data. The red line is a smooth spline developed with general cross validation (leave one out) which mimics the ggplot smooth function most closely. To get a kernel to act in similar fashion we need a bandwidth parameter over 50. The green line (bandwidth at 30) and the yellow line (bandwidth at 40) are also acceptable because they are mostly smooth approximations of the data. I choose the yellow (bandwidth at 40) for a nice mix of smoothness and variable explanation.

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

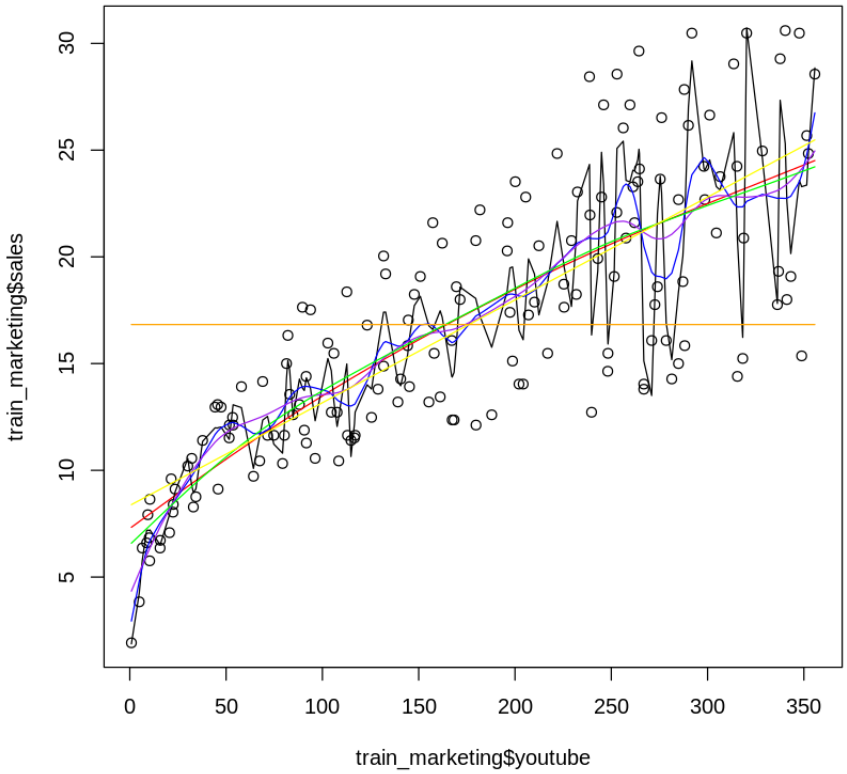
Again, using the `train_marketing` set, plot `sales` (response) against `youtube` (predictor). This time, fit and overlay a smoothing spline regression model. Experiment with the smoothing parameter until the smooth looks appropriate. Explain why it's appropriate and justify your answer.

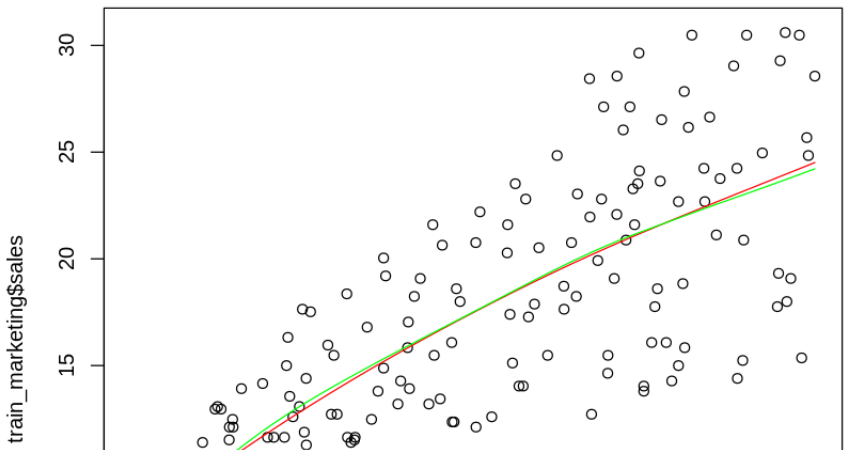
In [5]:
```
ssp = smooth.spline(train_marketing$youtube,train_marketing$sales)
summary(ssp)
ssp2 = smooth.spline(train_marketing$youtube,train_marketing$sales,spa
ssp3 = smooth.spline(train_marketing$youtube,train_marketing$sales,spa
ssp4 = smooth.spline(train_marketing$youtube,train_marketing$sales,spa
ssp5 = smooth.spline(train_marketing$youtube,train_marketing$sales,spa
ssp6 = smooth.spline(train_marketing$youtube,train_marketing$sales,spa
ssp7 = smooth.spline(train_marketing$youtube,train_marketing$sales,spa

plot(train_marketing$youtube,train_marketing$sales)
lines(ssp, col="red")
lines(ssp2$x,ssp2$y, col="black")
lines(ssp3$x,ssp3$y, col="blue")
lines(ssp4$x,ssp4$y, col="purple")
lines(ssp5$x,ssp5$y, col="green")
lines(ssp6$x,ssp6$y, col="yellow")
lines(ssp7$x,ssp7$y, col="orange")

plot(train_marketing$youtube,train_marketing$sales)
lines(ssp, col="red")
lines(ssp5$x,ssp5$y, col="green")
```

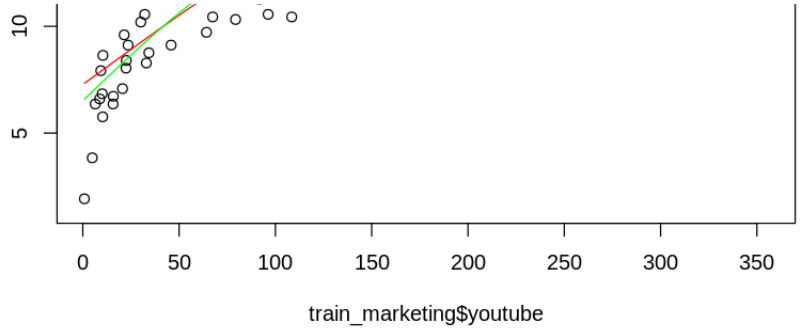
```
           Length Class              Mode
x          157    -none-             numeric
y          157    -none-             numeric
w          157    -none-             numeric
yin        157    -none-             numeric
tol          1    -none-             numeric
data         3    -none-             list
no.weights   1    -none-             logical
lev        157    -none-             numeric
cv.crit      1    -none-             numeric
pen.crit     1    -none-             numeric
crit         1    -none-             numeric
```

```
df           1    -none-            numeric
spar         1    -none-            numeric
ratio        1    -none-            numeric
lambda       1    -none-            numeric
iparms       5    -none-            numeric
auxM         0    -none-           NULL
fit          5    smooth.spline.fit list
call         3    -none-            call
```

```
Warning message in smooth.spline(train_marketing$youtube, train_marke
ting$sales, :
“smoothing parameter value too large
setting df = 1  __use with care!__”
```







Here, again, we see that a low spar value is unable to smooth the exceptionally dense data. Spar values under .7 are jagged representations of the data (black, blue, and purple lines). The green line (spar = 1.0) mimics the default spar value when letting the algorithm pick the most appropriate value for us (spar = 1.0) after 5 iterations, but fits the tails of the distribution more closely (why I chose this spar value). I was intrigued to see that a spar value of 2 (yellow line) was still close to the optimal value. Spar value of 3 (orange line) did not explain the data at all.

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 [9]:
```
ll.x = seq(min(train_marketing$youtube),max(train_marketing$youtube),l
print("MSPEs:")
ll5 = loess(sales~youtube, data = train_marketing,span=.07,degree=1)
ll5.y = predict(ll5,ll.x)
ll5MSE= sum((predict(ll5,train_marketing$youtube)-train_marketing$sale
ll5MSE

ll = loess(sales~youtube, data = train_marketing,span=.1,degree=1)
ll.y = predict(ll,ll.x)
llMSE= sum((predict(ll,train_marketing$youtube)-train_marketing$sales)
llMSE

ll2 = loess(sales~youtube, data = train_marketing,span=.3,degree=1)
ll2.y = predict(ll2,ll.x)
ll2MSE= sum((predict(ll2,train_marketing$youtube)-train_marketing$sale
ll2MSE

ll3 = loess(sales~youtube, data = train_marketing,span=.6,degree=1)
ll3.y = predict(ll3,ll.x)
ll3MSE= sum((predict(ll3,train_marketing$youtube)-train_marketing$sale
ll3MSE

ll4 = loess(sales~youtube, data = train_marketing,span=.9,degree=1)
```

```
ll4.y = predict(ll4,ll.x)
ll4MSE= sum((predict(ll4,train_marketing$youtube)-train_marketing$sale
ll4MSE



plot(train_marketing$youtube,train_marketing$sales)
lines(ll.x,ll.y,col="red")
lines(ll.x,ll2.y,col="green")
lines(ll.x,ll3.y,col="yellow")
lines(ll.x,ll4.y,col="purple")
lines(ll.x,ll5.y,col="blue")

plot(train_marketing$youtube,train_marketing$sales)

lines(ll.x,ll2.y,col="green")
```

```
[1] "MSPEs:"
```

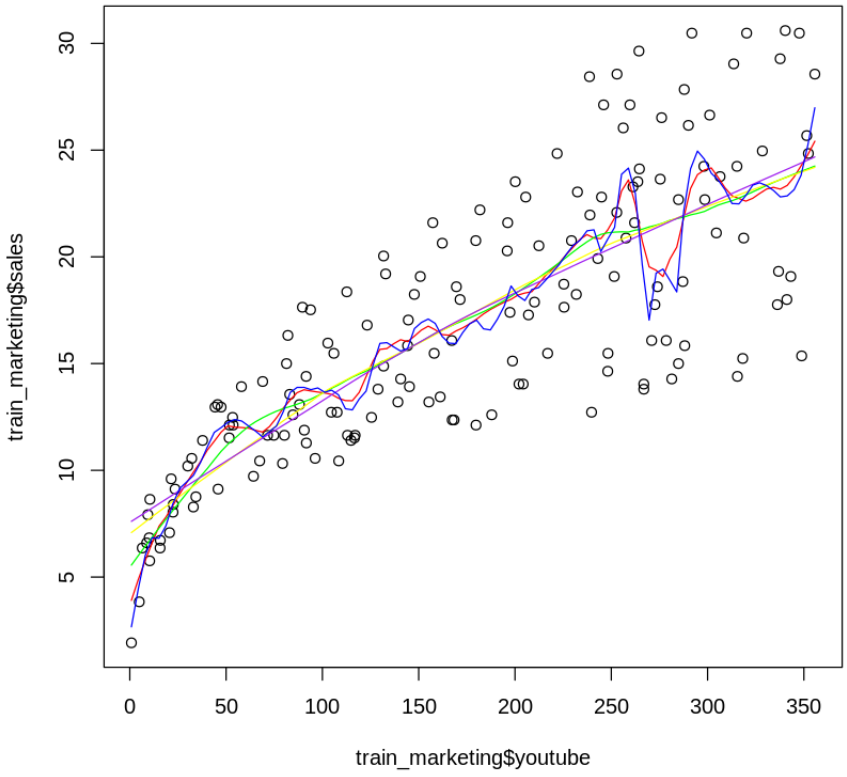
11.6245186800663

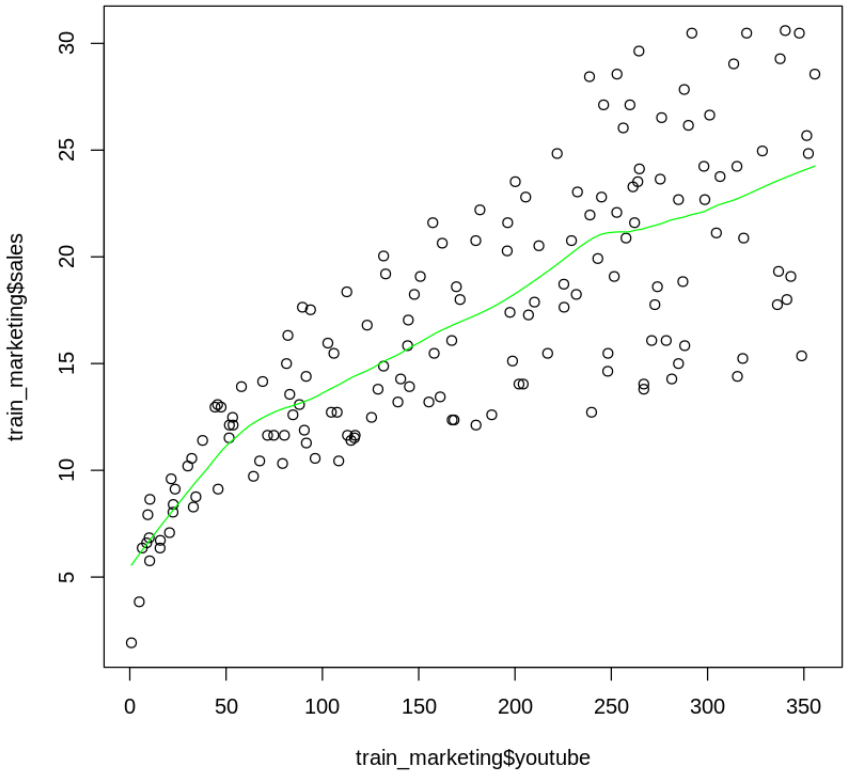
12.4831890984764

13.7802401829566

14.1369121019752

14.3094349502057





I believe a span of .3 (green line) is best for this data. The green line is the smallest span that results in a line without jagged edges. Additionally, this span selection has a lower MSE than any models with larger spans. I am attempting to weigh the tradeoff between smoothness and MSPE. Increasing the span size results in nearly linear estimations with larger MSE, despite the non-linearity of the data. Decreasing the span size results in jagged, nonsmooth, estimations of the data. The green line is the best of these two aspirations. To see why it fits the data best look for smoothness and accuracy in the tails of the estimation.

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

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

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

*Note that `ksmooth()` orders your designated `x.points`. Make sure to account for this in your MSPE calculation.

In [27]:
```
kernelpreds = ksmooth(train_marketing$youtube,train_marketing$sales,'n
# kernelpreds$x == test_marketing$youtube[order(test_marketing$youtube
kernelMSE = mean((kernelpreds$y - test_marketing$sales[order(test_mark
cat("\n Kernel MSE:", kernelMSE)

ll2MSE = sum((predict( ll2, newdata = test_marketing$youtube)-test_mar
cat("\n Loess MSPE:", ll2MSE)

sspreds = predict(ssp5,test_marketing$youtube)
#sspreds$x == test_marketing$youtube  #makes sure predictors line up..
sspMSE = mean((sspreds$y - test_marketing$sales)^2)
cat("\n Smoothing Spline MSPE:", sspMSE)

cat("\n\n Best Model is Smoothing Spline with MSPE of 17.8061")
```

```
 Kernel MSE: 18.26094
 Loess MSPE: 18.17186
 Smoothing Spline MSPE: 17.80605

 Best Model is Smoothing Spline with MSPE of 17.8061
```

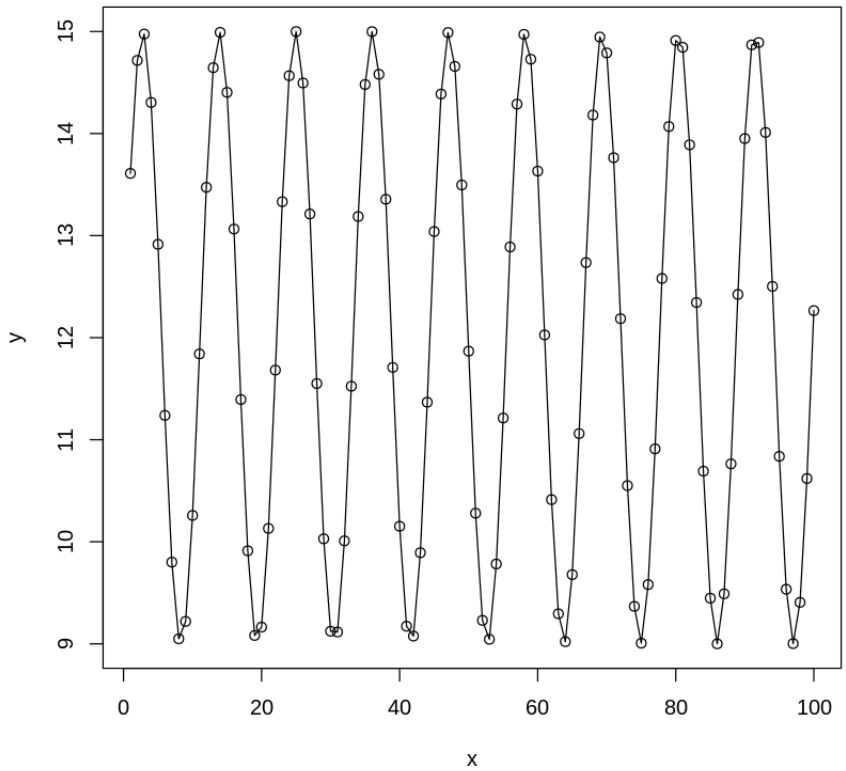
The above calculations return the Mean Square Prediction Error for each of the models chosen above. Although each is close (less than .5 difference between the three) the Smoothing Spline model has the lowest MSPE and is chosen for accuracy.

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.

To generate the data I will run a simple wave equation (y = A * sin(Ba-Cx) + D). I'll go ahead and plot the data so we can see it...

In [158]:
```
#simulated data
multiplier = seq(1,100) #rpois(1000,12)
base = rt(length(multiplier),1,0)

x = multiplier
y =   3 * sin(base * .000002 - 12 * multiplier) + 12
plot(x,y)
lines(x,y)
```



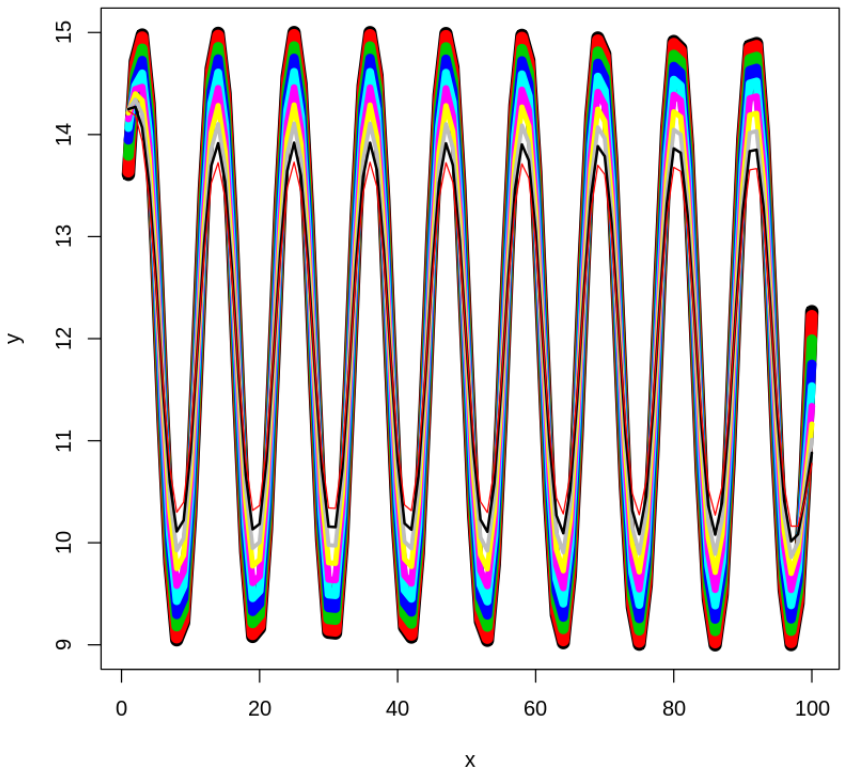
In [165]:
```
#1.a
print("Kernel Testing:")
library("RColorBrewer")
#display.brewer.all()
bws = seq(0.5,5,length.out=10)
```

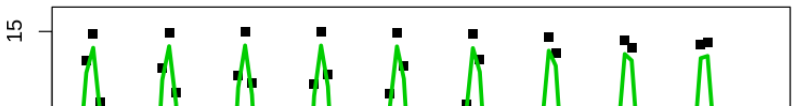
```
bws
cols <- data.frame(
  col = heat.colors(length(bws)),
  id = seq(1,length(bws),1)
)
plot(x,y,pch=15,cex=1)
for (i in 1:length(bws)){
    mod <- ksmooth(x,y,'normal',bws[i])
    lines(mod$x,mod$y, col=cols$col[match(i, cols$id)], lwd=11-i)
    }

print("Kernel Selected: Bandwidth = 1.5")
plot(x,y,pch=15,cex=1)
modfin <- ksmooth(x,y,'normal',1.5)
lines(modfin$x,modfin$y, col=cols$col[match(3 , cols$id)],lwd=3)
```

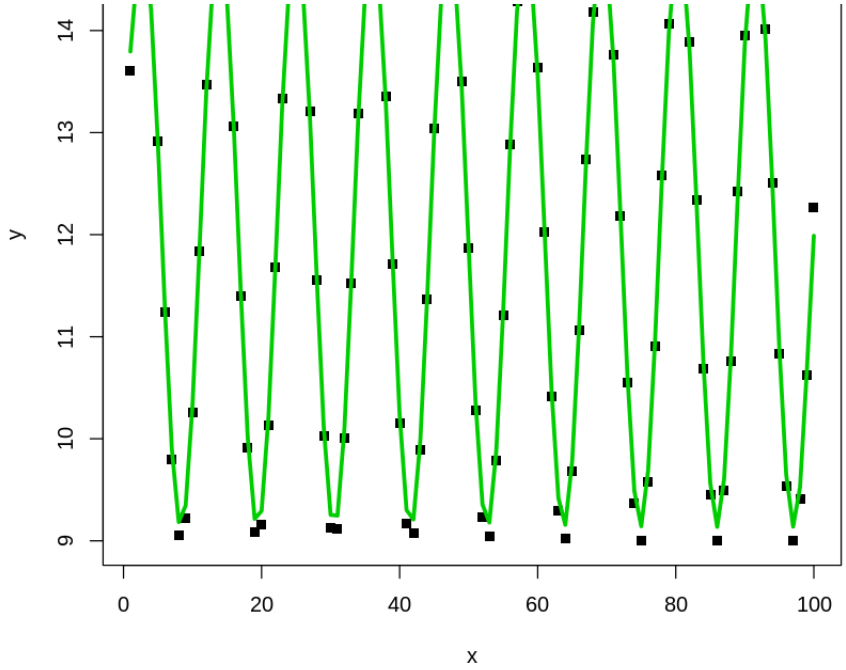
```
[1] "Kernel Testing:"
```

0.5 · 1 · 1.5 · 2 · 2.5 · 3 · 3.5 · 4 · 4.5 · 5

```
[1] "Kernel Selected: Bandwidth = 1.5"
```

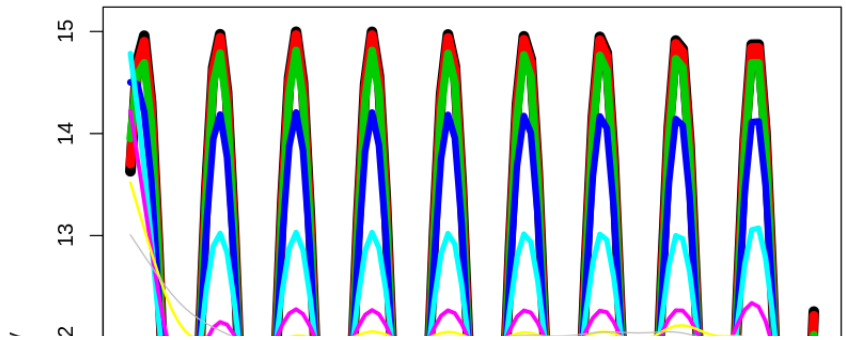


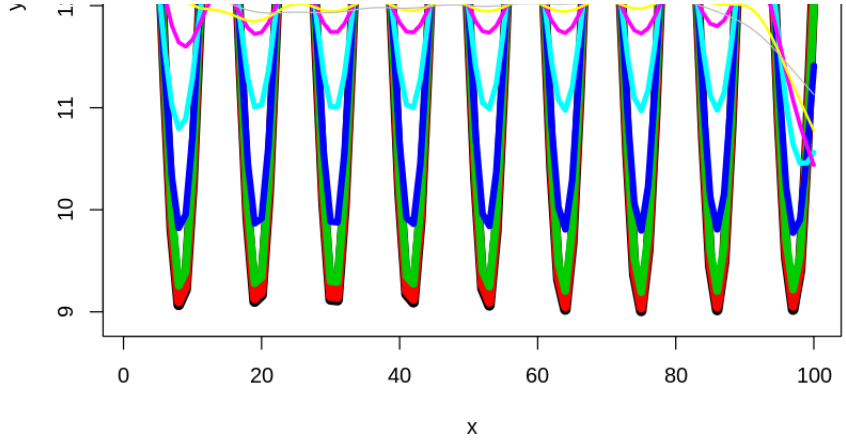


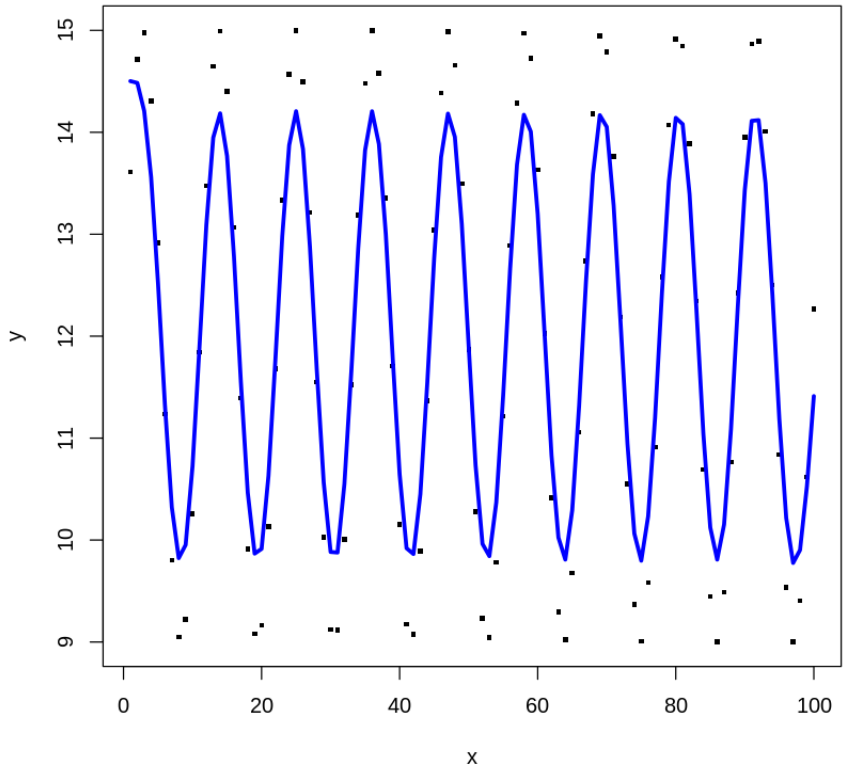


In [166]:
```
#1.b
spans = seq(.1,.8,.1)
cat("Spans tested:",spans)
cols <- data.frame(
  col = heat.colors(length(spans)),
  id = seq(1,length(spans),1)
)
plot(x,y,pch=15,cex=.5)
for (i in 1:length(spans)){
    mod <- smooth.spline(x,y,spar=spans[i])
    lines(mod$x,mod$y, col=cols$col[match(i, cols$id)], lwd=9-i)
    }

cat("\n\nSpan Chosen: 0.4")
plot(x,y,pch=15,cex=.5)
modssfin <- smooth.spline(x,y,spar=spans[4])
lines(modssfin$x,modssfin$y, col=cols$col[match(4 , cols$id)],lwd=3)
```

```
Spans tested: 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8

Span Chosen: 0.4
```



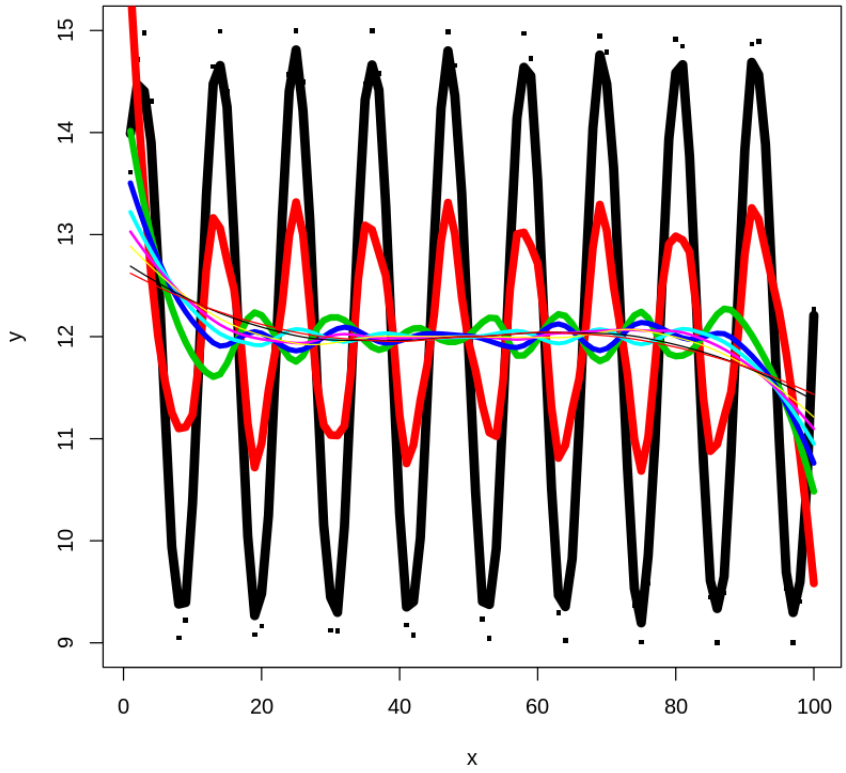


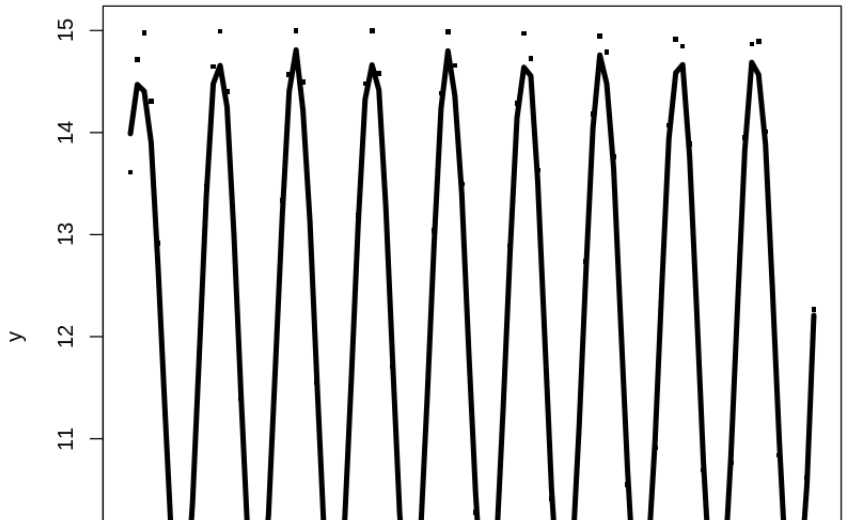


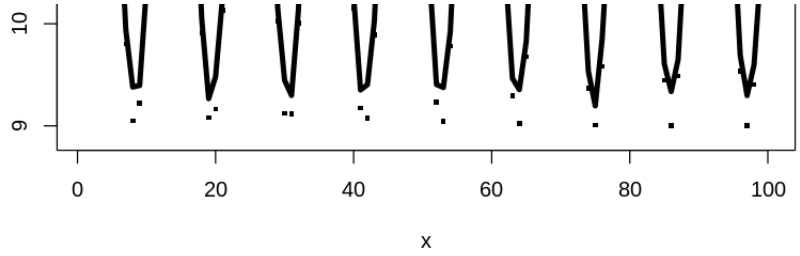
In [170]:
```
#1.c

ll.x = seq(min(x),max(x),length.out=100)

spans = seq(0.1,1,.1)
cat("Spans Tested:", spans)
cols <- data.frame(
  col = heat.colors(length(spans)),
  id = seq(1,length(spans),1)
)
plot(x,y,pch=15,cex=.5)
for (i in 1:length(spans)){
    lomod = loess(y~x, span=spans[i] )
    lines(ll.x,predict(lomod,ll.x), col=cols$col[match(i, cols$id)], l
```

```
    }


cat("\n\nSpan Chosen: 0.1")
plot(x,y,pch=15,cex=.5)
lomod = loess(y~x, span=.1 )
lines(ll.x,predict(lomod,ll.x), col=cols$col[match(1, cols$id)], lwd=4
```

```
Spans Tested: 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Span Chosen: 0.1
```







In [183]:
```r
#1.d
#Compute Kernel MSPE
kernelMSPE <- mean((modfin$y-y)^2)
cat("\n Kernel MSPE:", kernelMSPE)

#Compute Smooth Spline MSPE
#str(modssfin) #check order of x and y
smspMSPE <- mean((modssfin$y-y)^2)
cat("\n Smooth Spline MSPE:",smspMSPE)

#Compute Loess MSPE
loessMSPE <- mean((predict(lomod,x)-y)^2)
cat("\n Loess MSPE:",loessMSPE)

cat("\n\n Best fitting Model is the Kernel Model with span of 0.1")
```

```
 Kernel MSPE: 0.01035147
 Smooth Spline MSPE: 0.324685
 Loess MSPE: 0.03720567

 Best fitting Model is the Kernel Model with span of 0.1
```

For the Kernel set of models, I tested box and normal. None of the box outputs were smooth. I also tested spans ranging from 0.5 to 5 in increments of 0.5. I chose the span size of size 1.5 because of the smoothness of the line and the closeness of fit. Smaller spans were more jagged but fit almost perfectly, and larger spans were not as accurate. The MSPE for the kernel models was 0.103.

For the smooth.spline set of models, I tested spans ranging from .1 to .8 and chose .4 as the line of best fit. Smaller spans resulted in jagged lines. Larger spans resulted in linear more linear outputs. The choice of .4 was a the best, smoothest, choice. The MSPE of the smooth.spline output is 0.325.

For the loess function we tested spans ranging from .1 to 1 in increments of .1. Smaller spans result in an error, larger spans result in an almost linear regression output of our wave pattern. The loess function chosen (.1) has a MSPE of 0.0372.