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:

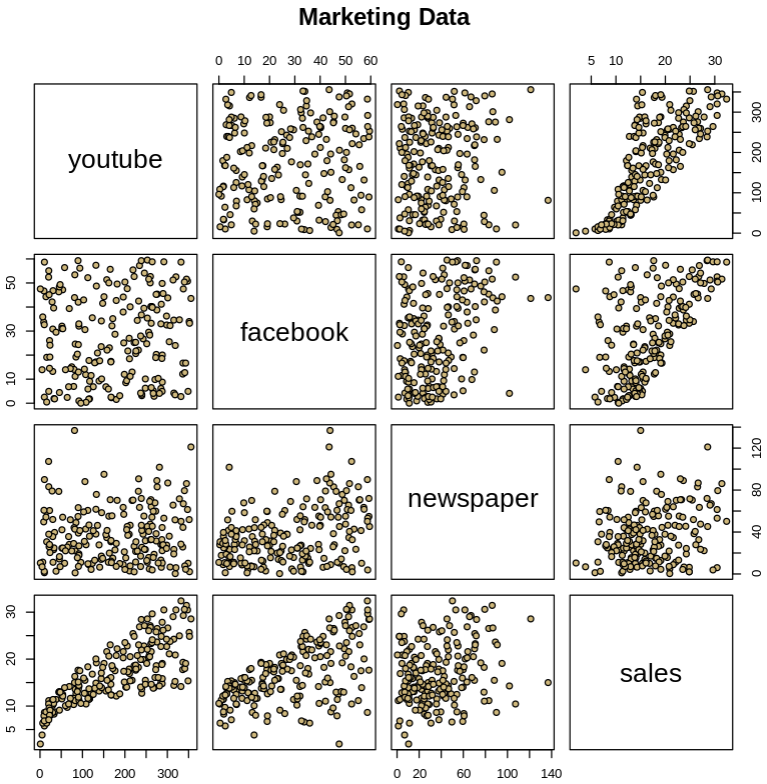
- `youtube` : the advertising budget allocated to YouTube. Measured in thousands of dollars;
- `facebook` : the advertising budget allocated to Facebook. Measured in thousands of dollars; and
- `newspaper` : the advertising budget allocated to a local newspaper. Measured in thousands of dollars.
- `sales` : 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 [3]:
```
# 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 [4]:
```
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 th
e 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
```

40 · 4

160 · 4

1.(a) Working with nonlinearity: Kernel regression

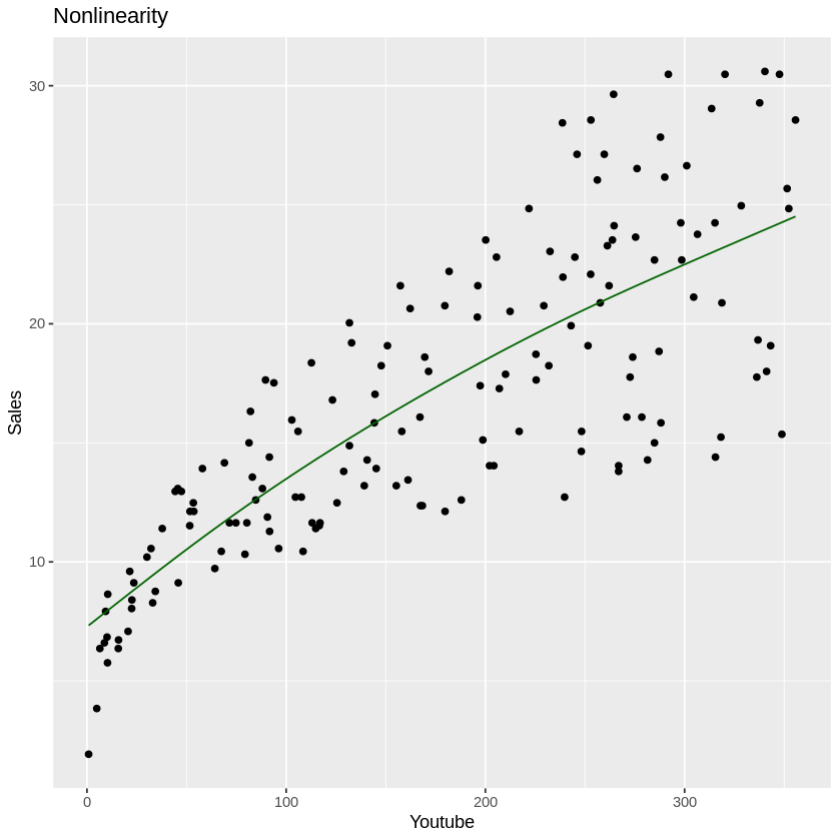
Note that the relationship between `sales` and `youtube` is nonlinear. This was a problem for us back in the first course in this specialization, when we modeled the data as if it were linear. For now, let's just focus on the relationship between `sales` and `youtube`, omitting the other variables (future 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 [21]:
```
install.packages("kernlab")
library(kernlab)
plot(train_marketing$youtube, train_marketing$sales, xlab = "YouTub
e", ylab = "Sales", main = "Kernel Regression")
bw = 1
fit = kpss(train_marketing$sales ~ train_marketing$youtube, bw = b
w, kernel = "rbfdot")
lines(sort(train_marketing$youtube), predict(fit, newdata = data.fr
ame(youtube = sort(train_marketing$youtube))), col = "red")
```

```
Warning message:
“unable to access index for repository https://cran.r-project.org/sr
c/contrib:
  cannot open URL 'https://cran.r-project.org/src/contrib/PACKAGES'”
Warning message:
“package ‘kernlab’ is not available (for R version 3.6.3)”

Error in library(kernlab): there is no package called ‘kernlab’
Traceback:

1. library(kernlab)
```

In [17]:
```
gam_model = gam(sales ~ s(youtube), data = train_marketing)
ggplot(train_marketing, aes(x = youtube, y = sales)) +
  geom_point() +
  geom_line(aes(y = predict(gam_model, newdata = data.frame(youtube
= youtube))),
  color = "darkgreen") +
  labs(x = "Youtube", y = "Sales", title = "Nonlinearity")
```



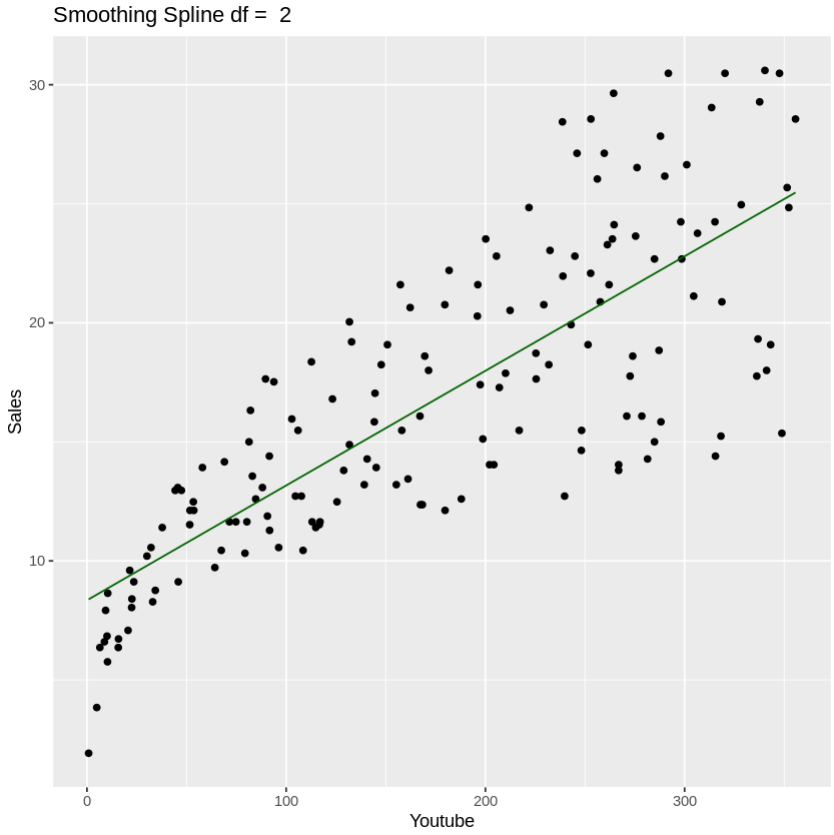
Was unable to get library(kernlab), or library(np) to load into Coursera, so I am using a gam model instead of a kernel regression.

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 [12]:

```
plot_smooth_spline_model = function(df) {
  ss_model = smooth.spline(train_marketing$youtube, train_marketin
g$sales, df = df)
  youtube_seq = seq(min(train_marketing$youtube), max(train_marketi
ng$youtube), length.out = 100)
  predicted_sales = predict(ss_model, x = youtube_seq)$y
  plot = ggplot(train_marketing, aes(x = youtube, y = sales)) +
    geom_point() +
    geom_line(data = data.frame(youtube = youtube_seq, sales = pred
icted_sales),
    aes(x = youtube, y = sales),
    color = "darkgreen") +
    labs(x = "Youtube", y = "Sales", title = paste("Smoothing Splin
e df = ", df))

  return(plot)
}
df_values <- c(2, 5, 10)
lapply(df_values, plot_smooth_spline_model)
```

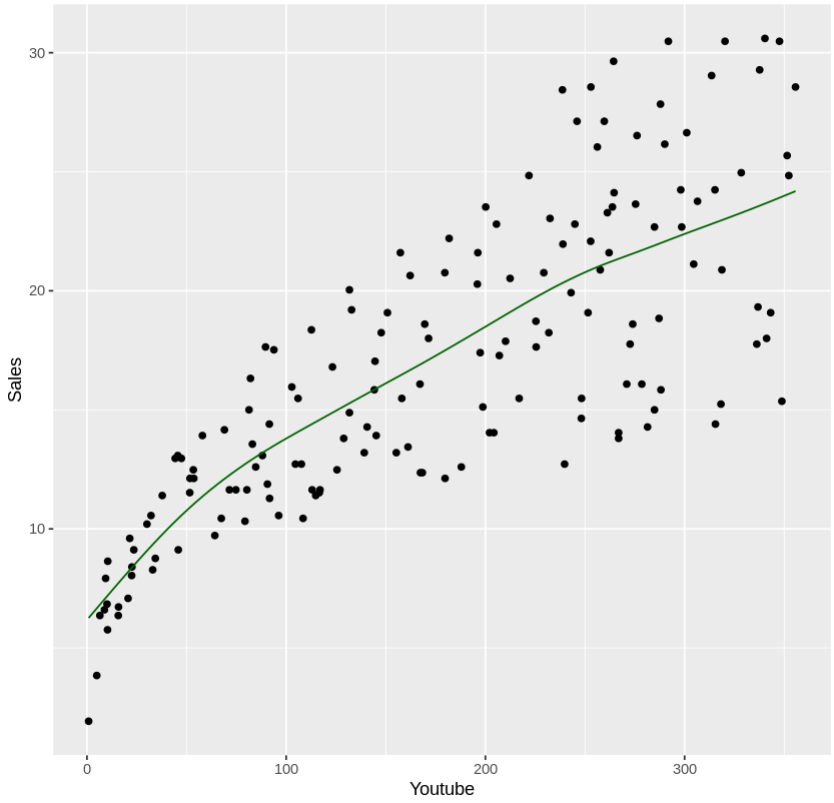


[[1]]

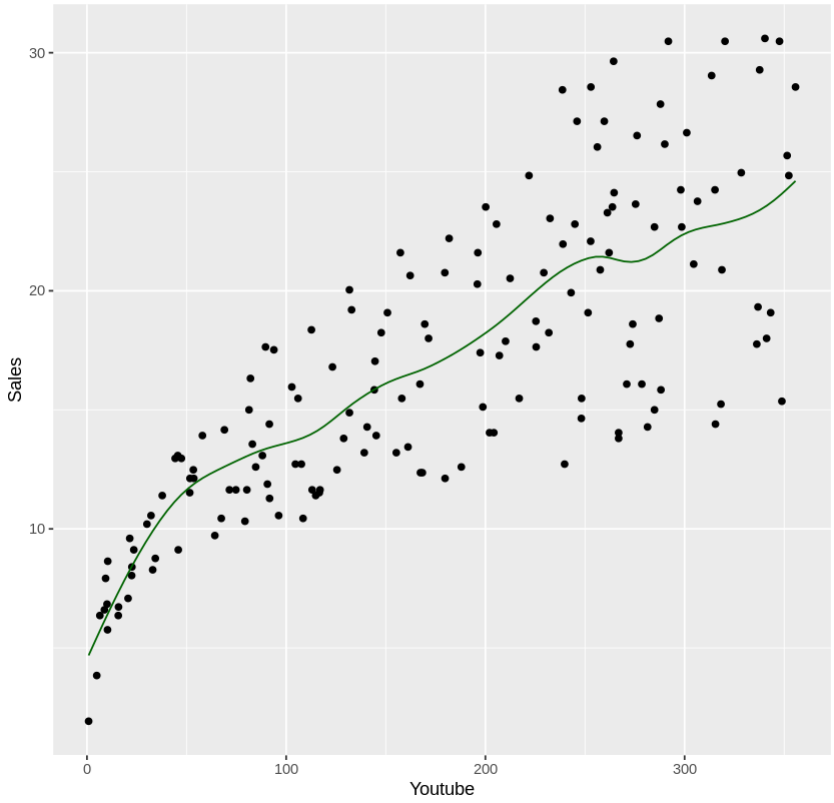
[[2]]

[[3]]

Smoothing Spline df = 5



Smoothing Spline df = 10



In our code we used the smoothing spline regression to reduce the amount of spanning, from out results we can see that at a df value of 2, we have the best fitting "smooth" line. It seems that as the df value increases, the smoothness decreases

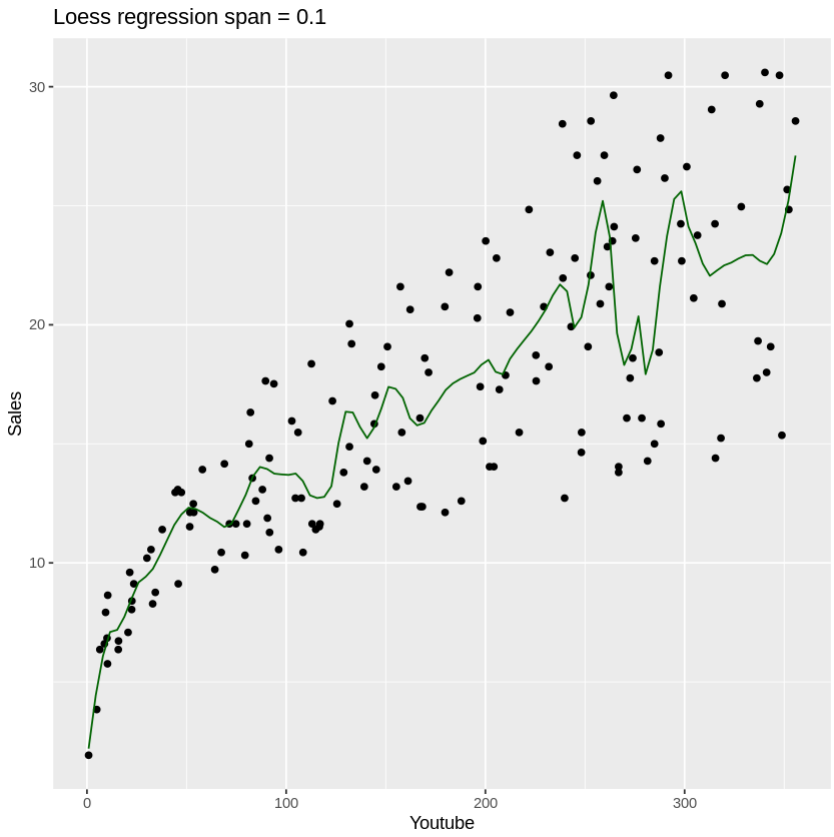
1.(c) Working with nonlinearity: Loess

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.

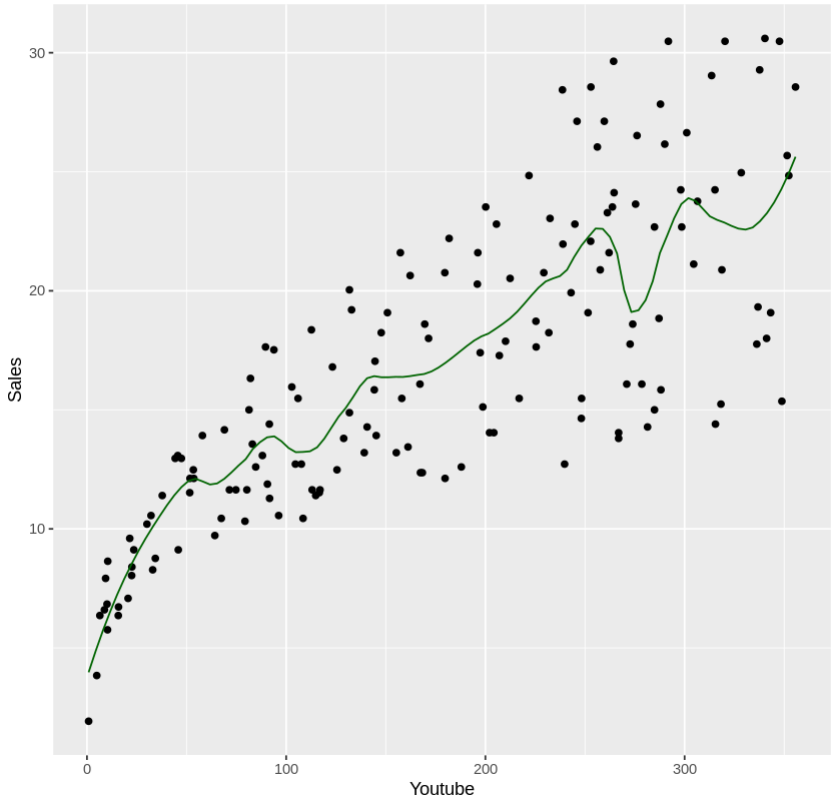
In [11]:

```
plot_loess_model = function(span) {
  loess_model = loess(sales ~ youtube, data = train_marketing, span
= span)
  youtube_seq = seq(min(train_marketing$youtube), max(train_marketi
ng$youtube), length.out = 100)
  predicted_sales = predict(loess_model, newdata = data.frame(youtu
be = youtube_seq))
  plot = ggplot(train_marketing, aes(x = youtube, y = sales)) +
    geom_point() +
    geom_line(data = data.frame(youtube = youtube_seq, sales = pred
icted_sales),
    aes(x = youtube, y = sales),
    color = "darkgreen") +
    labs(x = "Youtube", y = "Sales", title = paste("Loess regressio
n span =", span))
  cat("Loess span =", span, "\n")
  cat("Standard error:", sd(loess_model$residuals), "\n\n")

  return(plot)
}
span_values = seq(0.1, 1, by = 0.1)
lapply(span_values, plot_loess_model)
```

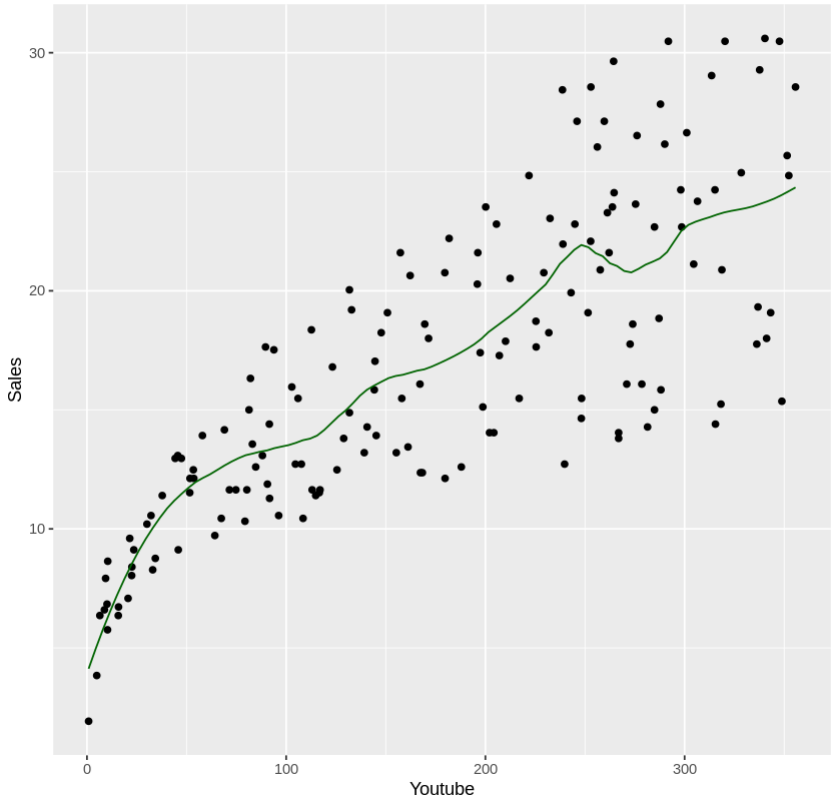
```
Loess span = 0.1
Standard error: 3.410586

Loess span = 0.2
Standard error: 3.584219

Loess span = 0.3
Standard error: 3.682007

Loess span = 0.4
Standard error: 3.708835

Loess span = 0.5
Standard error: 3.719997

Loess span = 0.6
Standard error: 3.734689

Loess span = 0.7
Standard error: 3.742815

Loess span = 0.8
Standard error: 3.749072

Loess span = 0.9
Standard error: 3.75797

Loess span = 1
Standard error: 3.763231
```



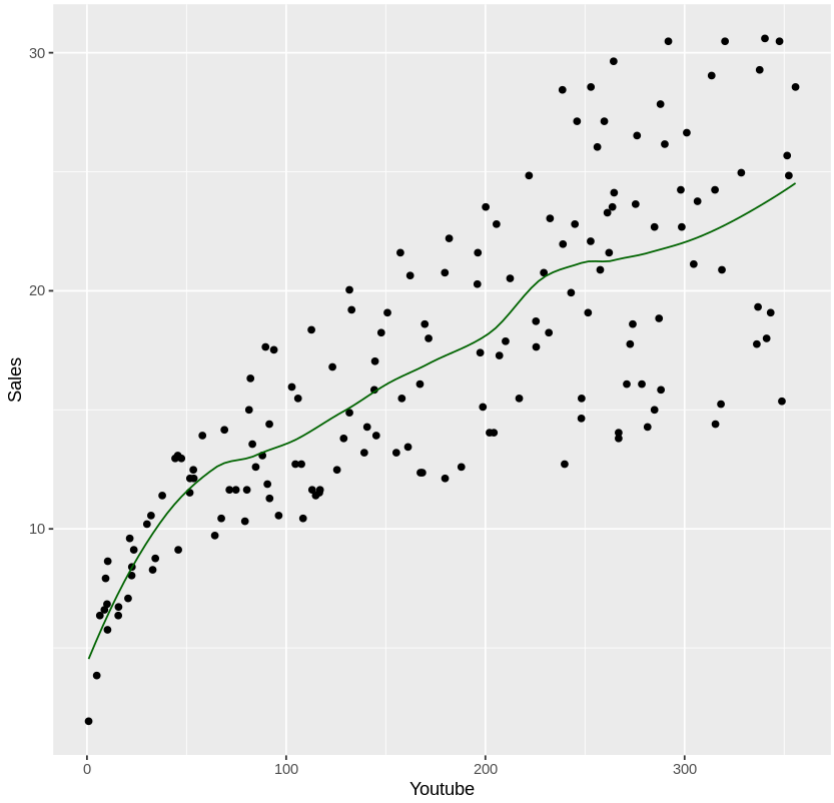
Loess regression span = 0.2



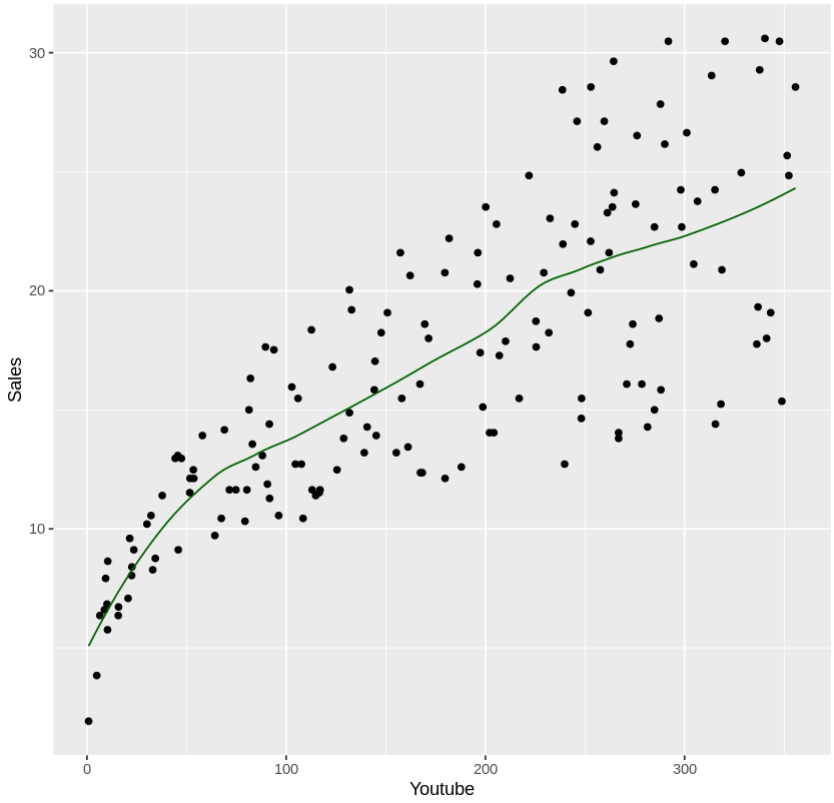
Loess regression span = 0.3



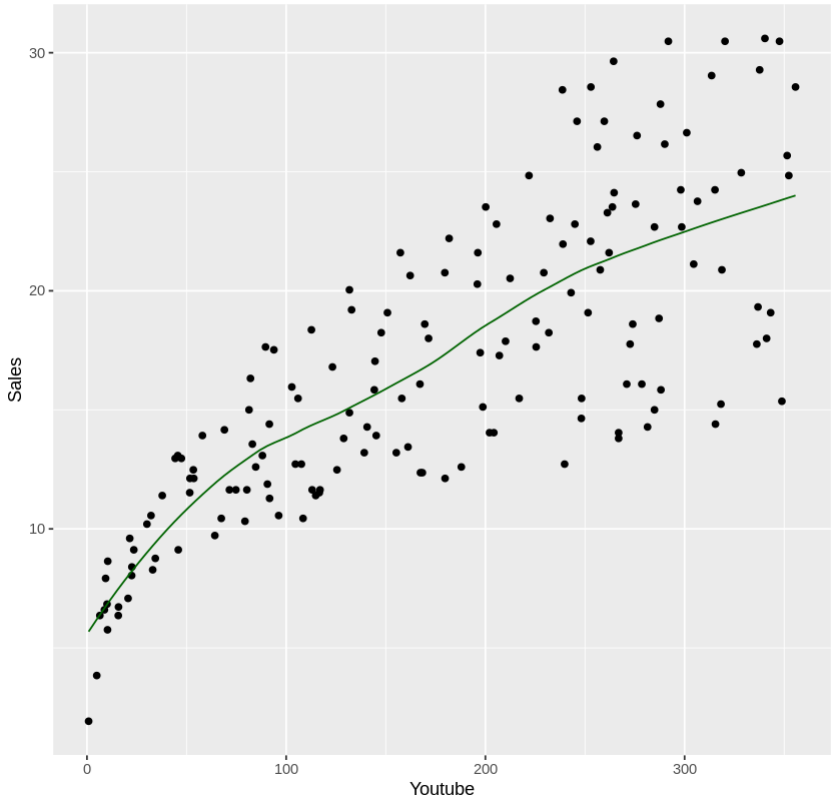
Loess regression span = 0.4



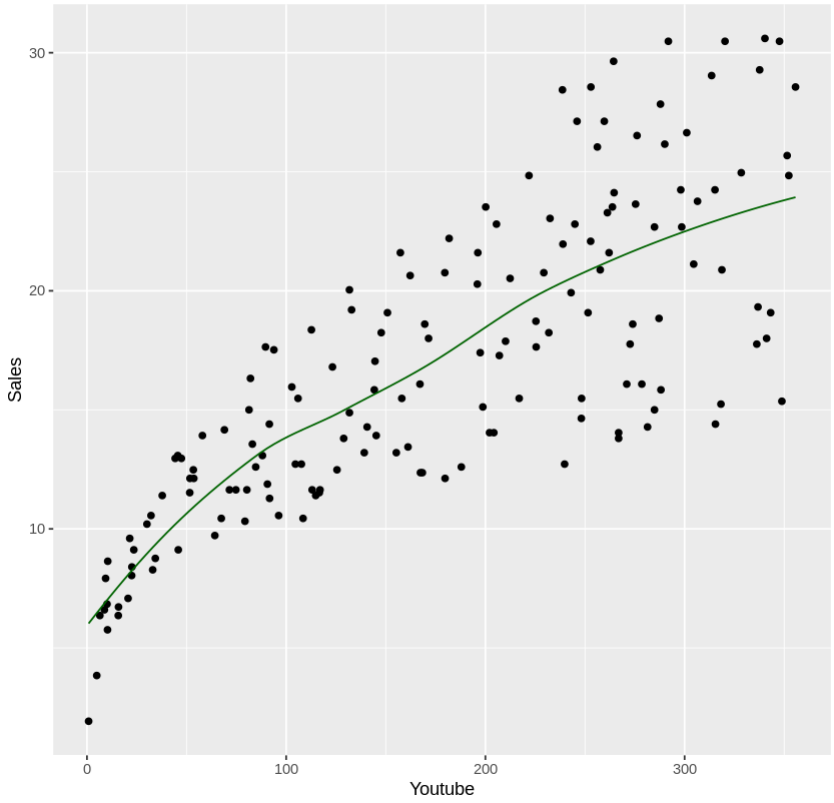
Loess regression span = 0.5

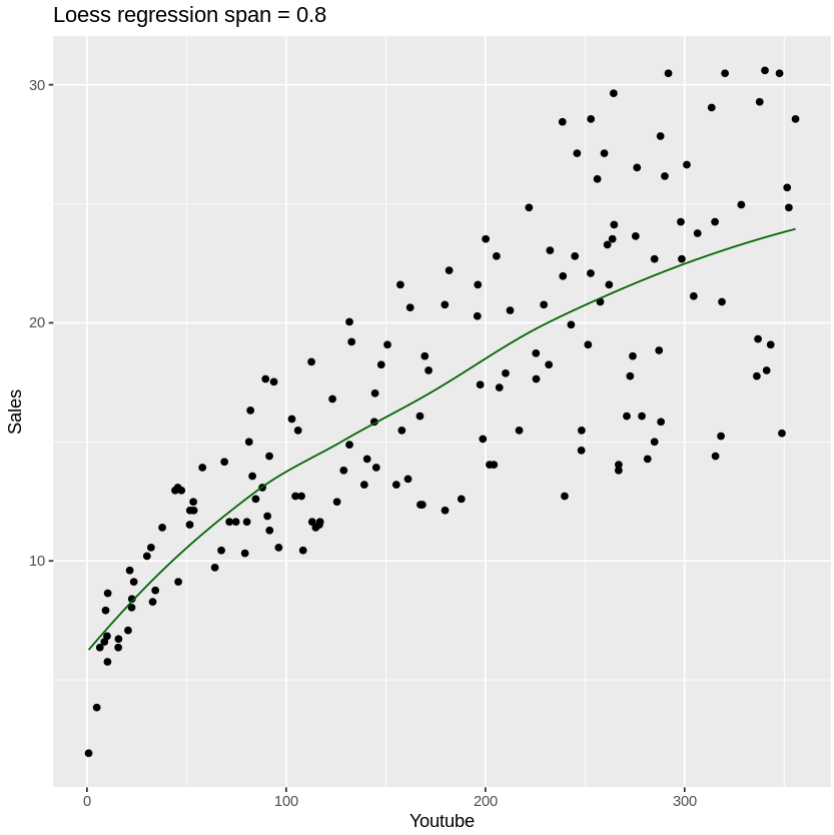


Loess regression span = 0.6



Loess regression span = 0.7





[[1]]

[[2]]

[[3]]

[[4]]

[[5]]

[[6]]

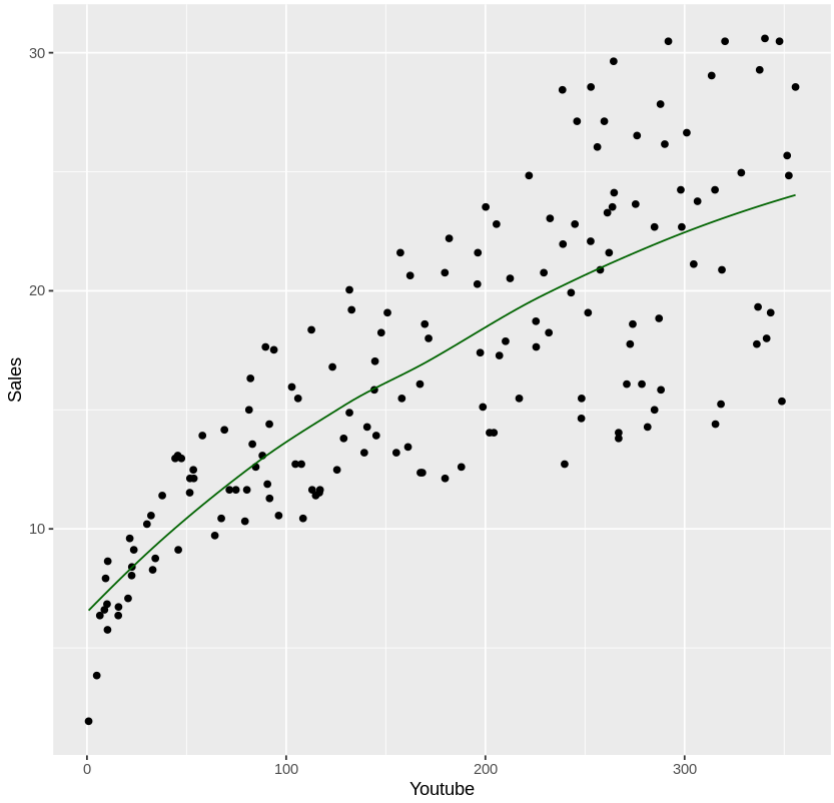
[[7]]

[[8]]

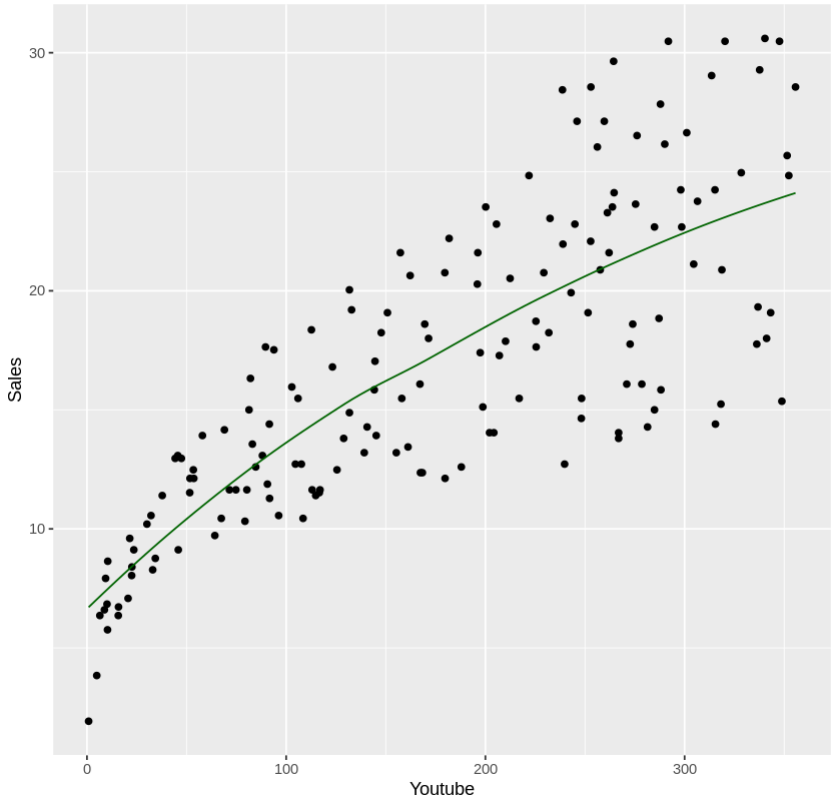
[[9]]

[[10]]

Loess regression span = 0.9



Loess regression span = 1



In the above code, we use the loess function to run a regression model, by controlling the span we are able to add smoothness to the curve, we incremented our span by .1 until it reached 1. We could have encluded more, or gone into more detial within our code, added different diagnostic models, etc, but I felt this smoothness looked appropriate.

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).

In [20]:
```
calc_mspe = function(actual, predicted) {
  mean((actual - predicted) ^ 2)
}

# GAM
gam_pred = predict(models[[which.min(deviances)]], newdata = test_m
arketing)
gam_mspe <- calc_mspe(test_marketing$sales, gam_pred)

# Smoothing Spline
ss_model = smooth.spline(train_marketing$youtube, train_marketing$s
ales, df = 5)
ss_pred = predict(ss_model, x = test_marketing$youtube)$y
ss_mspe = calc_mspe(test_marketing$sales, ss_pred)

# Loess
loess_model = loess(sales ~ youtube, data = train_marketing, span =
which.min(span_values))
loess_pred = predict(loess_model, newdata = test_marketing)
loess_mspe = calc_mspe(test_marketing$sales, loess_pred)

cat("MSPE for GAM Model: ", gam_mspe, "\n")
cat("MSPE for Smoothing Spline: ", ss_mspe, "\n")
cat("MSPE for Loess: ", loess_mspe, "\n")
```

```
MSPE for GAM Model:  17.55009
MSPE for Smoothing Spline:  17.89612
MSPE for Loess:  17.7954
```

Based off this data, the model with the lowest MSPE, is that of part a, GAM model, thus this is the best model in terms of our caculating metric

Problem 2: Simulations!

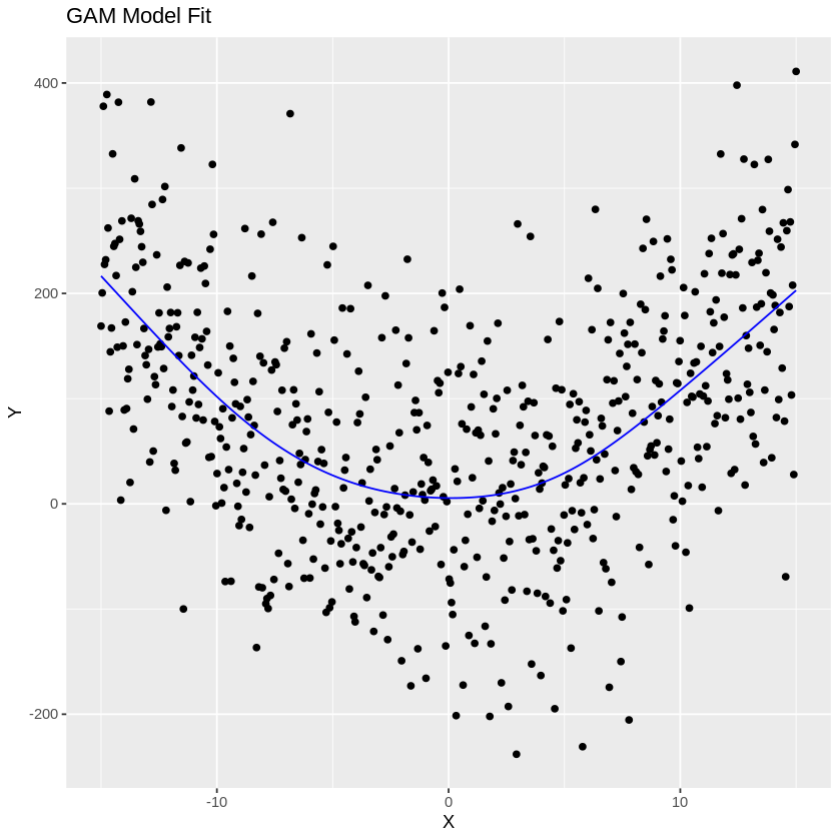
Simulate data (one predictor and one response) with your own nonlinear relationship. Provide an explanation of how you generated the data. Then answer the questions above (1.(a) - 1.(d)) using your simulated data.

Firstly, we will need to simulate our data, can generate it with a seed, get our population size n, get our x, and y, and finally plot this data to get a feel for it:

In [32]:
```
#simulated data
set.seed(123)
n = 600
x = seq(-15, 15, length.out = n)
y = x^2 + rnorm(n, 0, 100)
simulated_data <- data.frame(x = x, y = y)
```

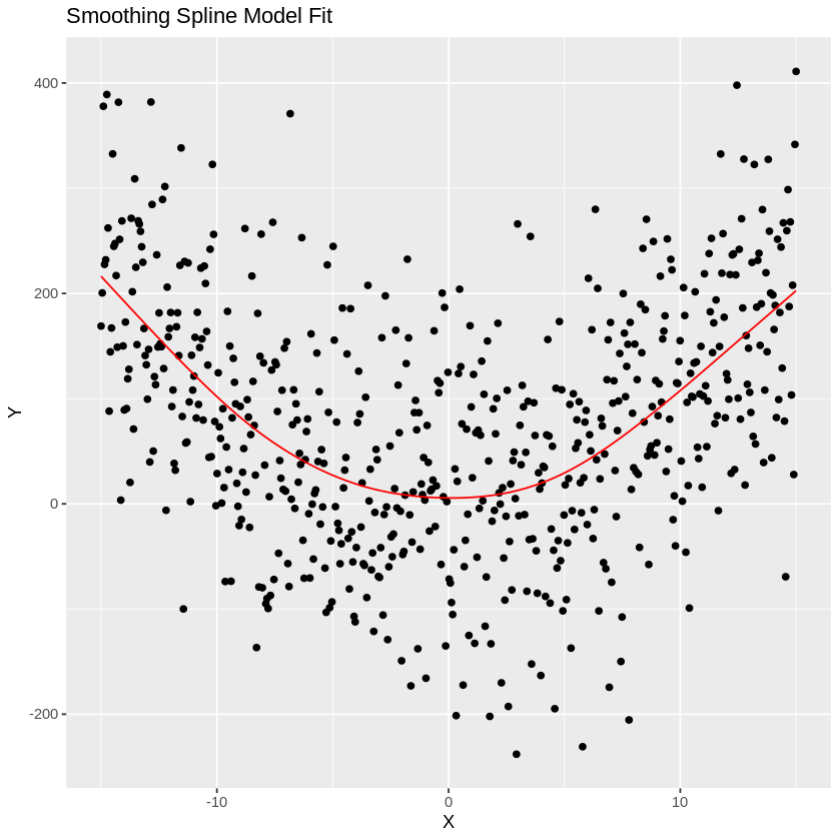
GAM Model:

In [33]:
```
#1.a
gam_model = gam(y ~ s(x), data = simulated_data)
ggplot(simulated_data, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = predict(gam_model, newdata = data.frame(x =
x))), color = "blue") +
  labs(x = "X", y = "Y", title = "GAM Model Fit")
```



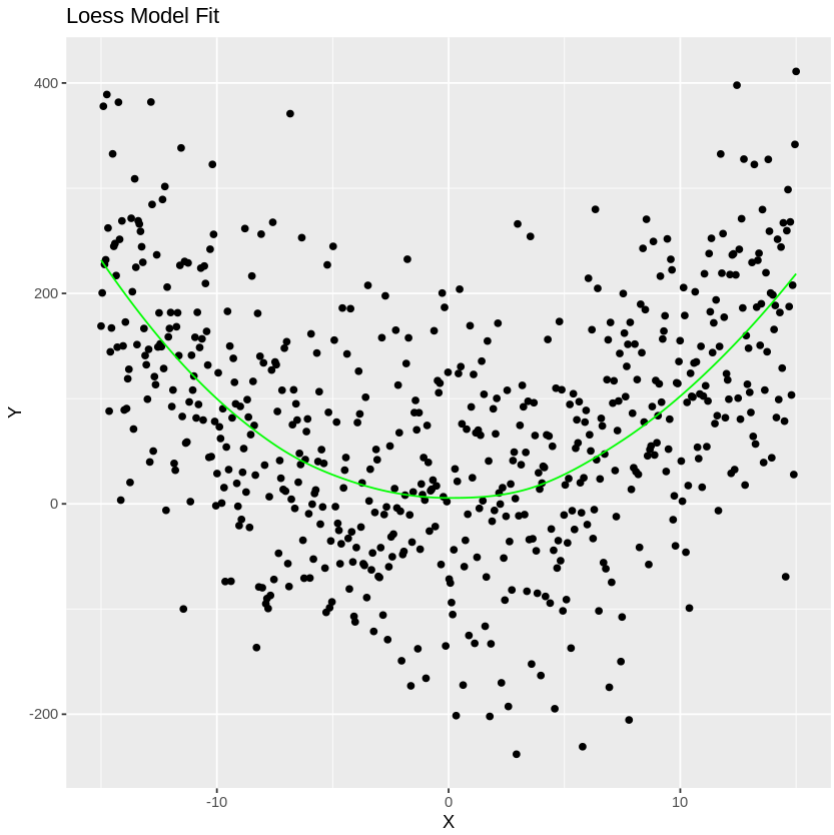
Smoothing Spline Model:

In [34]:
```
#1.b
ss_model = smooth.spline(simulated_data$x, simulated_data$y)
ggplot(simulated_data, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = predict(ss_model, x = x)$y), color = "red") +
  labs(x = "X", y = "Y", title = "Smoothing Spline Model Fit")
```



Loess Model:

In [35]:
```
#1.c
loess_model = loess(y ~ x, data = simulated_data)
ggplot(simulated_data, aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = predict(loess_model, newdata = data.frame(x =
x))), color = "green") +
  labs(x = "X", y = "Y", title = "Loess Model Fit")
```



Predictive Metric:

In [36]:
```
#1.d
set.seed(456)
n_test = 100
x_test = runif(n_test, -10, 10)
y_test = x_test^2 + rnorm(n_test, 0, 100)
test_data = data.frame(x = x_test, y = y_test)

gam = mean((y_test - predict(gam_model, newdata = test_data))^2)
ss = mean((y_test - predict(ss_model, x = x_test)$y)^2)
loess = mean((y_test - predict(loess_model, newdata = test_data))^
2)
cat("GAM: ", gam, "\n")
cat("Smoothing Spline:", ss, "\n")
cat("Loess: ", loess, "\n")
```

```
GAM:  9080.417
Smoothing Spline: 9078.144
Loess:  9075.595
```