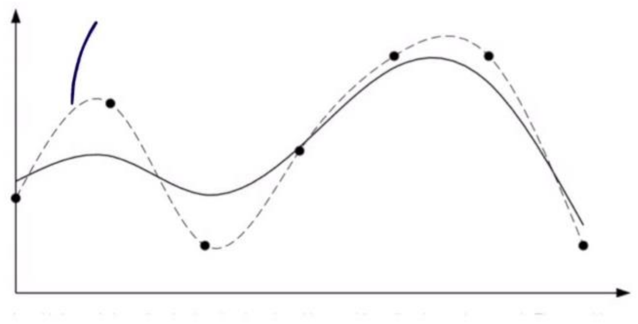
MTH 325, Optional Lab #1, Spring 2024 Name ____

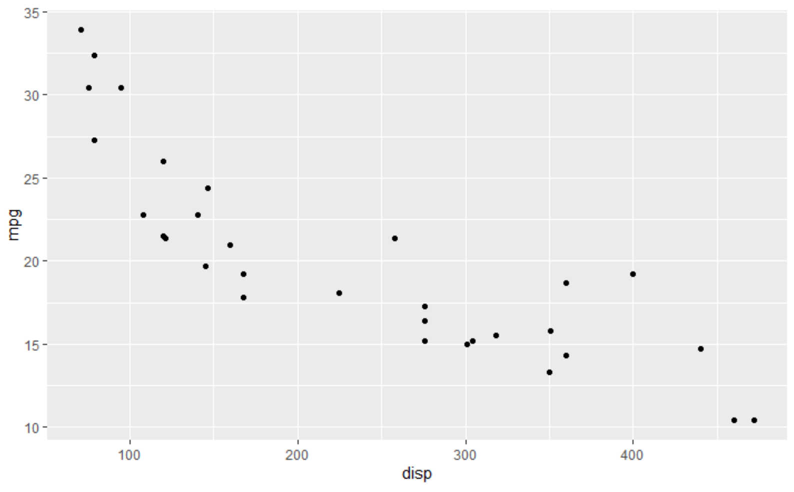
Instructions: Follow along with the tutorial portion of the lab. Replicate the code examples in R on your own, along with the demonstration. Then use those examples as a model to answer the questions/perform the tasks that follow. Copy and paste the results of your code to answer questions where directed. Submit your response file and the code used (both for the tutorial and part two). Your code file and your lab response file should each include your name inside.

We're going to look at three types of non-parametric (or sometimes called semi-parametric) regression methods. We'll start with spline methods. Splines are curves that try to balance fit with smoothness. They are composed of simpler curves that may or may not join at the ends. They can be represented by piecewise functions. A smoothing spline does require the endpoints to join, particularly with conditions of continuity or possibly also with a continuous first (or second) derivative.



We will use the mtcars dataset and model mpg with the disp(lacement) variable.

```
data(mtcars)
library(ggplot2)
ggplot(data=mtcars, aes(x=disp,y=mpg))+geom_point()
```



The trend is nonlinear. There are several packages that allow us to apply splines to model our data. The package npreg has two smoothing splines methods. The knots are the number of breaks in the function. Typically these are selected by quantiles.

```
library(npreg)
mod.ss <- ss(mtcars$disp, mtcars$mpg, nknots = 5)
mod.ss

mod.smsp <- smooth.spline(mtcars$disp, mtcars$mpg, nknots = 5)
mod.smsp
```

We can plot the results on the same graph to compare the results. These two functions produce similar results, though not identical results.

```
plot(mtcars$disp, mtcars$mpg)
lines(mod.ss$x, mod.ss$y, lty = 2, col = 1, lwd = 2)
lines(mod.smsp$x, mod.smsp$y, lty = 3, col = 2, lwd = 2)
legend("topright",
       legend = c("ss", "smooth.spline"),
       lty = 1:2, col = 1:2, lwd = 2, bty = "n")
```

The ss function can also plot error bars easily.

```
plot(mod.ss)
```

We can use the summary() function to see a detailed analysis of the model.

```
summary(mod.ss)
```

We can adjust parameters of the smoothing spline to adjust the results. If λ is set to a value close to zero, there will be less smoothing and the function will be jumpier, but if it is set to a large value, then the model will tend toward linear as the value grows. We can also modify the parameter m that adjusts the penalty. The joins will be smoother if m is larger, and more cusp-like if m is 1 (the default is 2).

```
mod.ss0 <- ss(mtcars$disp, mtcars$mpg, all.knots = TRUE, lambda = 10E-7, m=2)
plot(mod.ss0, ylim = c(0, 40))
points(mtcars$disp, mtcars$mpg)
```

Test the plots with lambda = {10E-12, 10E-7, and 1}, and separately, m = {1,2,3}.

The splines2 package allows you to specify the knots specifically (as values) rather than just by count. You can also use basis splines (with a specified number of non-zero derivatives). These are referred to as B-splines. This package can also apply periodic functions and other types of splines called M-splines, I-splines, and C-splines. We will focus here on the B-splines.

Load the splines2 package. There are two ways to create the bSpline knots (use the default or specify with the knots function). In this case, both models produce the same results. You can create second prediction by repeating the predict() function with mod2. Compare the plots.

```r
library(splines2)

mod1 <- lm(mpg ~ bSpline(disp, degree = 1, df = 3), data = mtcars)
iknots <- with(mtcars, knots(bSpline(disp, degree = 1, df = 3)))
mod2 <- lm(mpg ~ bSpline(disp, degree = 1, knots = iknots), data = mtcars)
mtcars2<-mtcars
mtcars2$disp<-seq(min(mtcars$disp),max(mtcars$disp),(max(mtcars$disp)-min(mtcars$disp))/31)
pred1 <- predict(mod1, mtcars2)

plot(mtcars2$disp,pred1,type="l")
points(mtcars$disp, mtcars$mpg,col="blue")
```

You can update the degree and the number of knots to change the shape of the graph.

```r
mod1 <- lm(mpg ~ bSpline(disp, degree = 2, df = 5), data = mtcars)
pred1 <- predict(mod1, mtcars2)

plot(mtcars2$disp,pred1,type="l")
points(mtcars$disp, mtcars$mpg,col="blue")
```

Let’s now look at LOESS models. LOESS is more computationally complex than splines, though it has some conceptual similarities. LOESS models are built on low-degree polynomial functions, but rather than a small number of knots (breaks) that must be joined together, the LOESS model takes a window of the data (defined by a percentage of the nearest available points). In principle, the model could (in principle) be constructed as a piecewise function, but with many more pieces that would depend on the number of data points being modeled. It is therefore usually just expressed as a set of fitted values.

One advantage LOESS models in R have over splines is that it is built into ggplot, making visualizing the model much easier. We can also plot it manually in base R functions. We’ll look at both.

We’ll start by comparing several versions of LOESS using different span values. Span should be a number between 0 and 1 that gives the proportion of the data that is going to be used at each value to build the model at that point. The minimum value of the span is not really 0. Rather it depends on the number of data points and the degree of the polynomial used for the pieces (typically quadratic). For example, the span for a quadratic piece cannot include fewer than three values (to get an exact fit), or four values for any sort of error estimation. You generally don’t want to go that close to the minimum because you end of modeling noise (and overfitting), but the algorithm won’t work at all smaller than these counts. The percentage will therefore be smaller as the number of points in the dataset increases. The default values for span is in the 0.67 to 0.75 range.

```r
loess50 <- loess(mpg ~ disp, data=mtcars, span=.5)
smooth50 <- predict(loess50, newdata=mtcars2, se=FALSE)

loess75 <- loess(mpg ~ disp, data=mtcars, span=.75)
smooth75 <- predict(loess75, newdata=mtcars2, se=FALSE)

loess90 <- loess(mpg ~ disp, data=mtcars, span=.9)
smooth90 <- predict(loess90, newdata=mtcars2, se=FALSE)
```

We are using the same sequence of inputs here that we used above for the splines case in order to plot the curves in base R.

```r
plot(mtcars$disp, mtcars$mpg, pch=19, main='Loess Regression Models')
lines(smooth50, x=mtcars2$disp, col='red')
lines(smooth75, x=mtcars2$disp, col='purple')
lines(smooth90, x=mtcars2$disp, col='blue')
legend('topright', legend=c('.5', '.75', '.9'),
       col=c('red', 'purple', 'blue'), pch=19, title='Smoothing Span')
```

The 75% span and the 90% are pretty similar except for a small section in the middle, but the 50% span is much more nonlinear.

```r
summary(loess50)
summary(loess75)
```

You can also compare the models using the summary function. Here, we'll just look at two of them. The summary provides a value for the residual standard error, and the equivalent number of parameters (recall, for a polynomial, the number of parameters is one more than the degree). The default degree is listed here as 2 for each span of values, but this is an attribute that can be modified by specifying it in the model. Inside loess(), the degree can be specified to be 0, 1, or 2, but using a constant function is uncommon. The loess() function also allows you to specify a weighting scheme.

In ggplot, the LOESS model can be specified directly in the graph.

```r
ggplot(data=mtcars, aes(x=disp,y=mpg))+geom_point()+
  geom_smooth(formula="y~x", method="loess", se=TRUE)
```

We can adjust the span inside the geom_smooth() piece.

```r
ggplot(data=mtcars, aes(x=disp,y=mpg))+geom_point()+
  geom_smooth(formula="y~x", method="loess", se=TRUE, span=0.5)
```

We can plot the spline model and the LOESS model on the same graph to compare them visually.

```r
plot(mtcars$disp, mtcars$mpg, pch=19, main='Compare Splines & Loess Regression Models')
lines(smooth50, x=mtcars2$disp, col='purple')
lines(mod.ss$x, mod.ss$y, lty = 2, col = 1, lwd = 2)
legend("topright", legend = c("LOESS", "spline"),
       lty = 1:2, col = 1:2, lwd = 2, bty = "n")
```

While our examples have only one independent variable, there are packages for R than can build a two-independent variable model.

Our last type of non-parametric model we'll consider here is Gaussian process regression. Because the default kernel in the package we are using is exponential, the model works better on small values rather than large ones, so here, I've rescaled the disp(lacement) variable so that everything falls between 0 and 1. We could set the minimum to 0 and max to 1, or use a -1 to 1 scaling with the mean at 0, or use a z-score range (roughly -3 to 3). But large values will generally produce an error. The output values don't need to be scaled here.

```
library(RobustGaSP)
input<-as.matrix(data.frame(mtcars['disp']/500))
output<-as.matrix(data.frame(mtcars['mpg']))
model<- rgasp(design = input, response = output, zero.mean="No",nugget.est=TRUE)
model
```

The model output includes a mean and variance and other parameter settings. One advantage of the Gaussian process model is that the error is less about the mean model and more about the data itself. The error range will encompass most of the data, while that is generally not the case in other types of models.

We can plot the graph using the example below.

```
testing_input = as.matrix(seq(0,1,1/100))
model.predict<-predict(model,testing_input)
plot(testing_input,model.predict$mean,type='l',col='blue',
     xlab='input',ylab='output',main="GP regression")
polygon(c(testing_input,rev(testing_input)),
        c(model.predict$lower95,
          rev(model.predict$upper95)),col = "grey80", border = FALSE)
lines(testing_input,model.predict$mean,type='l',col='blue')
lines(input, output,type='p')
```

To plot this in ggplot, we could rescale to match the original data, save the output and input as a dataframe.

The last thing we want to do is compare all three models to each other on one graph.

```
plot(testing_input,model.predict$mean,type='l',col=1,
     xlab='input',ylab='output',main="GP vs. LOESS vs. spline regression")
polygon(c(testing_input,rev(testing_input)),
        c(model.predict$lower95,
          rev(model.predict$upper95)),col = "grey80", border = FALSE)
lines(testing_input,model.predict$mean,type='l',col=1)
lines(input, output,type='p')
lines(smooth50, x=mtcars2$disp/500, col=2)
lines(mod.ss$x/500, mod.ss$y, lty = 2, col = 3, lwd = 2)
legend("topright", legend = c("GP", "LOESS", "spline"),
       lty = 1:3, col = 1:3, lwd = 2, bty = "n")
```

From the previous graph, I added the two other models and adjusted the colors, and updated the title and legend. You can also look at the individual graphs side by side where we plotted the errors and look at the error bars relative to the data to see how the Gaussian process regression error is much wider than in the other models.

Tasks:

1. Install the package nlraa and load the dataset swpg from that package. Use the variables x=ftsw and y=lfgr (ftsw stands for Fraction Transpirable Soil Water and lfgr stands for Relative Leaf Growth). Plot the data (it is strongly nonlinear). Construct the following models:

a. A spline model (either a smoothing spline or a bSpline)
b. A LOESS model (use span = 0.5)
c. Gaussian process model

Plot all the models individually, and on the same graph. Describe what you see. Assess which model you like best for this data and why.

Resources:

1. http://users.stat.umn.edu/~helwig/notes/smooth-spline-notes.html
2. https://cran.r-project.org/web/packages/splines2/vignettes/splines2-intro.html
3. https://www.statology.org/loess-regression-in-r/
4. https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/predict.loess
5. https://rpubs.com/kaz_yos/loess1
6. https://andrewirwin.github.io/data-visualization/smoothing.html
7. https://bookdown.org/rbg/surrogates/chap5.html
8. https://cran.r-project.org/web/packages/RobustGaSP/RobustGaSP.pdf
9. https://cran.r-project.org/web/packages/nlraa/vignettes/nlraa.html