Regression Techniques Homework

Subhrajyoty Roy (MB1911)

September 1, 2019

Ridge, Lasso and Elastic Net

To illustrate the idea of Ridge regression, Lasso regression and Elastic net regression techniques, we shall use the **Communities and Crime Dataset**, which is available at the link https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime.

The dataset consists of 128 variables with 1994 observations. There are different measurements on the demographics consititution of a community and the goal is to predict the number of violent crimes happening within that community area. According to the description provided at the website, there are 5 non-predictive variables (like the name of the community, their state code etc.) which should be removed before applying the regression methods.

```
crimedata = read.table("communities.data", sep = ",", na.strings = "?")
dim(crimedata)
```

```
[1] 1994  128
```

```
# there are 1994 rows and 128 columns

# first 5 columns contain meta information which should be removed
crimedata <- crimedata[, -c(1:5)]
# remove the NA values
crimedata <- crimedata[complete.cases(crimedata),]

dim(crimedata)
```

```
[1] 319 123
```

Note that, there were many missing observations, hence we only get 319 many sample observations containing all the information on 123 many variables.

Now, we seperate a training and testing data from the whole dataset.

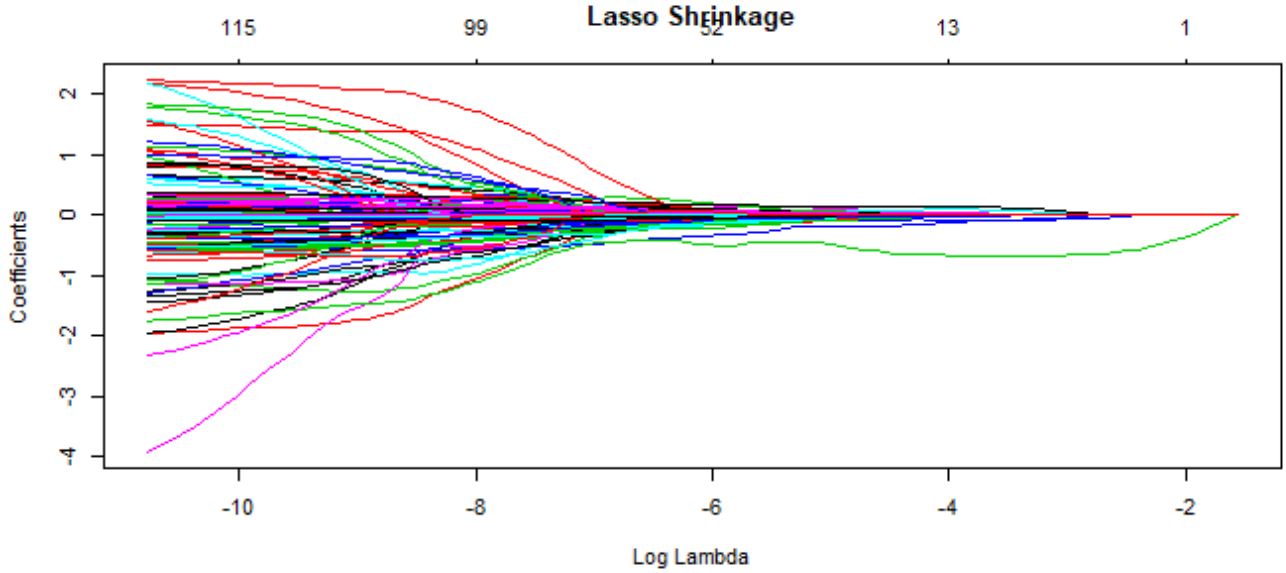
```
set.seed(1911)
# choose 2/3-rd of data as training set
train_rows <- sample(1:319, 0.67*319)

x.train <- as.matrix(crimedata[train_rows, -123])
y.train <- crimedata[train_rows, 123]

x.test <- as.matrix(crimedata[-train_rows, -123])
y.test <- crimedata[-train_rows, 123]
```

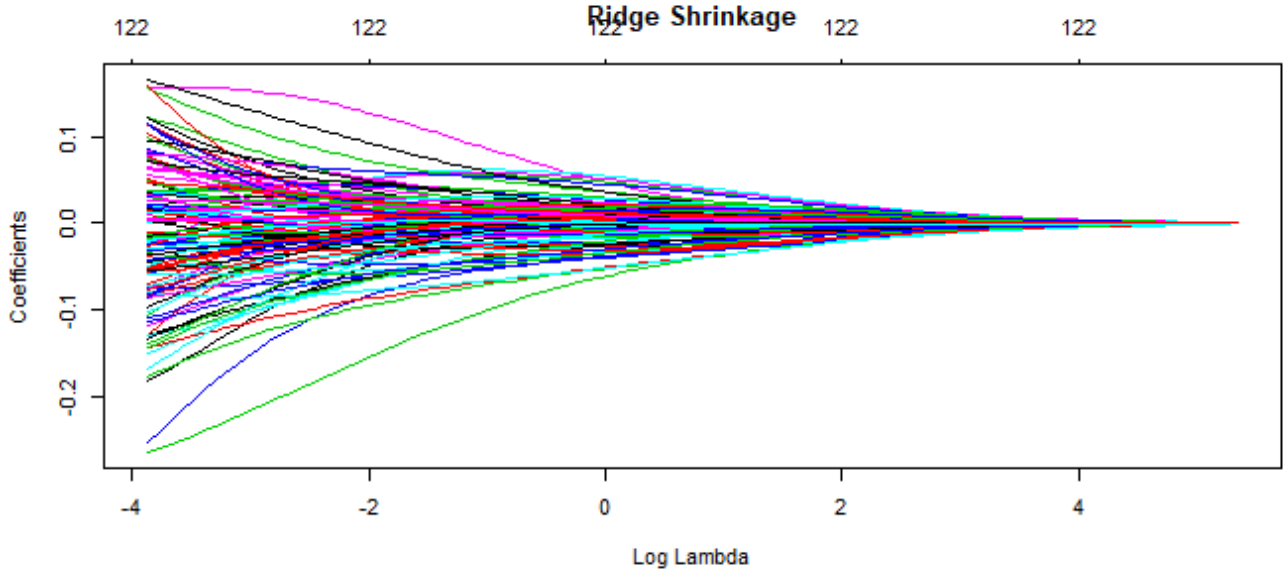
Now, we use **glmnet** package to fit Ridge, Lasso and Elastic net regression model.

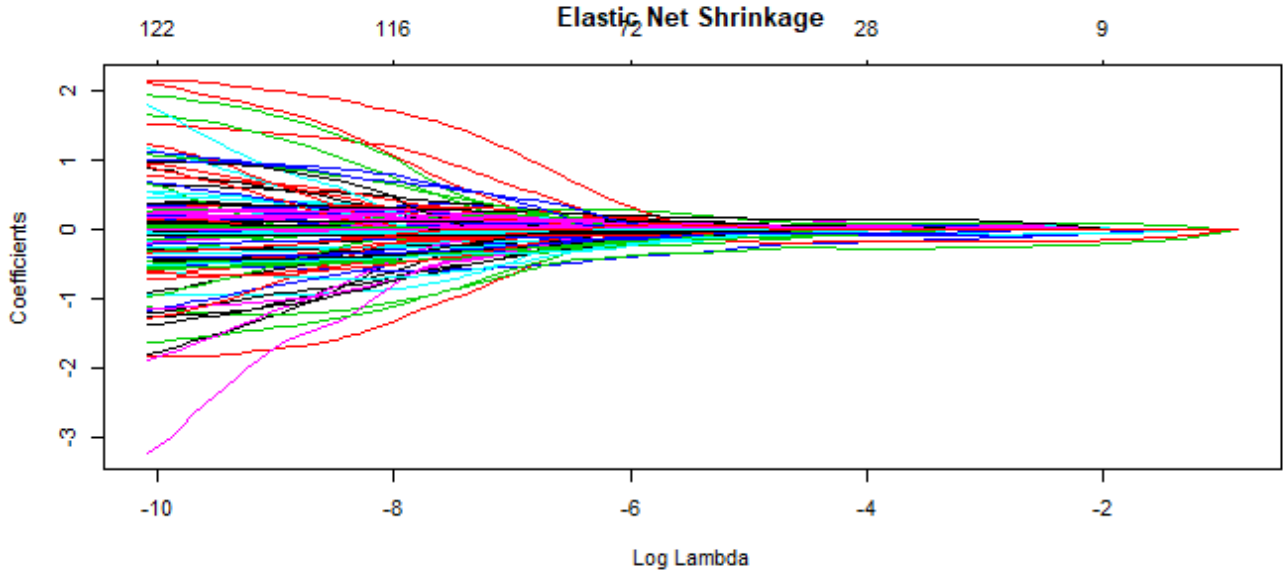
```
library(glmnet)  # load the glmnet package

fit.lasso <- glmnet(x.train, y.train,
                    family = "gaussian", alpha = 1)
fit.ridge <- glmnet(x.train, y.train,
                    family = "gaussian", alpha = 0)
fit.elnet <- glmnet(x.train, y.train,
                    family = "gaussian", alpha = 0.5)
```

Now, we make the plot to see the effect of shrinkage as penalty parameter λ is increased.

```
par(mfrow = c(3,1))
plot(fit.lasso, xvar = "lambda", main = "Lasso Shrinkage")
plot(fit.ridge, xvar = "lambda", main = "Ridge Shrinkage")
plot(fit.elnet, xvar = "lambda", main = "Elastic Net Shrinkage")
```





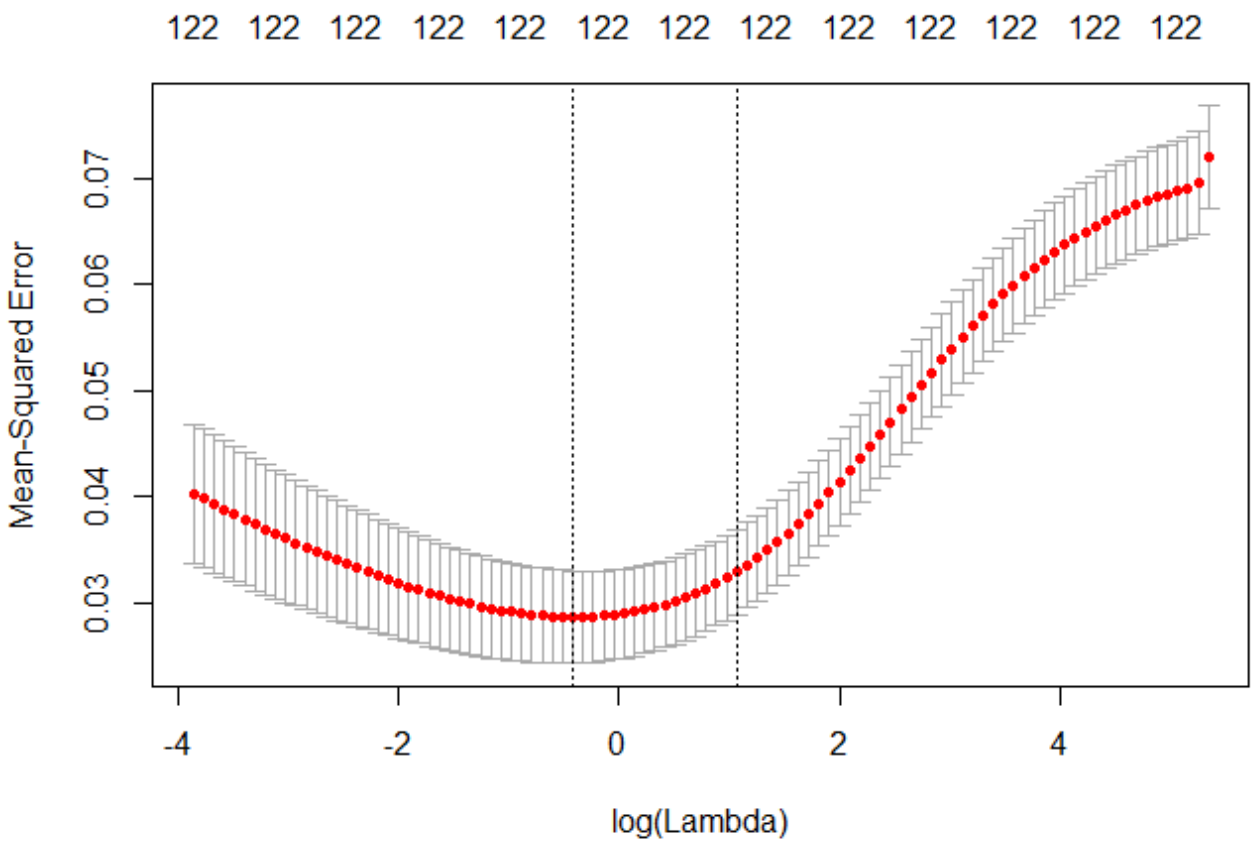


Note that, as λ increases the coefficients in all type of regression shrinks towards 0. However, such shrinkage is rapid in case of Lasso and Elastic net, while is

slower in case of Ridge regression. To find the optimal value for penalty parameter, in classical framework, we use cross validation.

We first perform 10-fold cross validation for Ridge regression model.

```
cvfit1 <- cv.glmnet(x.train, y.train, alpha = 0, nfolds = 10)
plot(cvfit1)
```




```
print(cvfit1$lambda.min)
```

```
[1] 0.6563151
```

Rather than printing out all the 123 coefficients of the ridge regression with optimal λ we consider the quantiles of the absolute value of the coefficients, which should give us a rough idea about how small these coefficients are going to be.

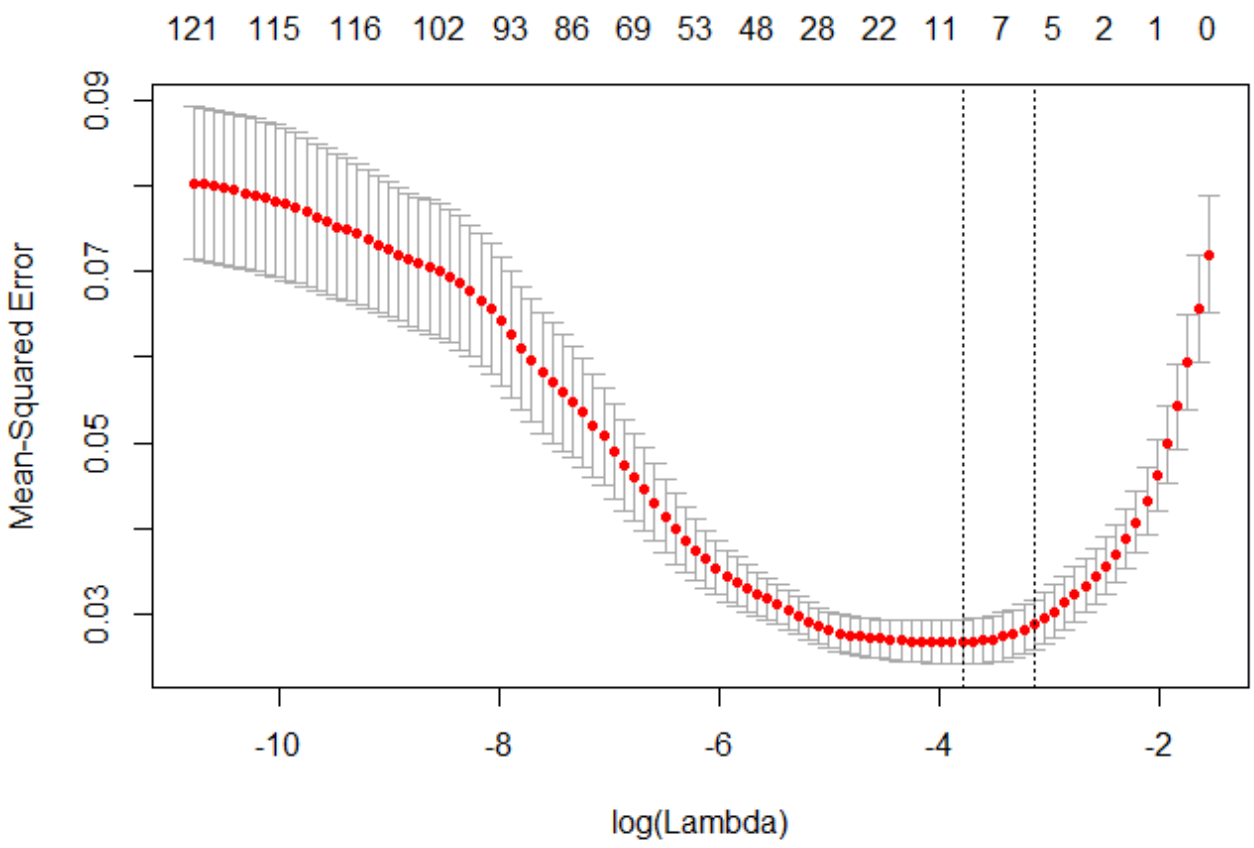
```
a <- coef(fit.ridge, cvfit1$lambda.min)
quantile(abs(as.vector(a)))
```

```
          0%          25%          50%          75%         100% 
0.0003879408 0.0056334003 0.0117365668 0.0246035814 0.4625599860 
```

Note that, about 75% of the coefficients have absolute value less than 0.025, which is pretty small. However, no coefficient has been set exactly equal to 0 due to ridge regression.

A similar thing is performed for Lasso also.

```
cvfit2 <- cv.glmnet(x.train, y.train, alpha = 1, nfolds = 10)
plot(cvfit2)
```




```
print(cvfit2$lambda.min)
```

```
[1] 0.02251469
```

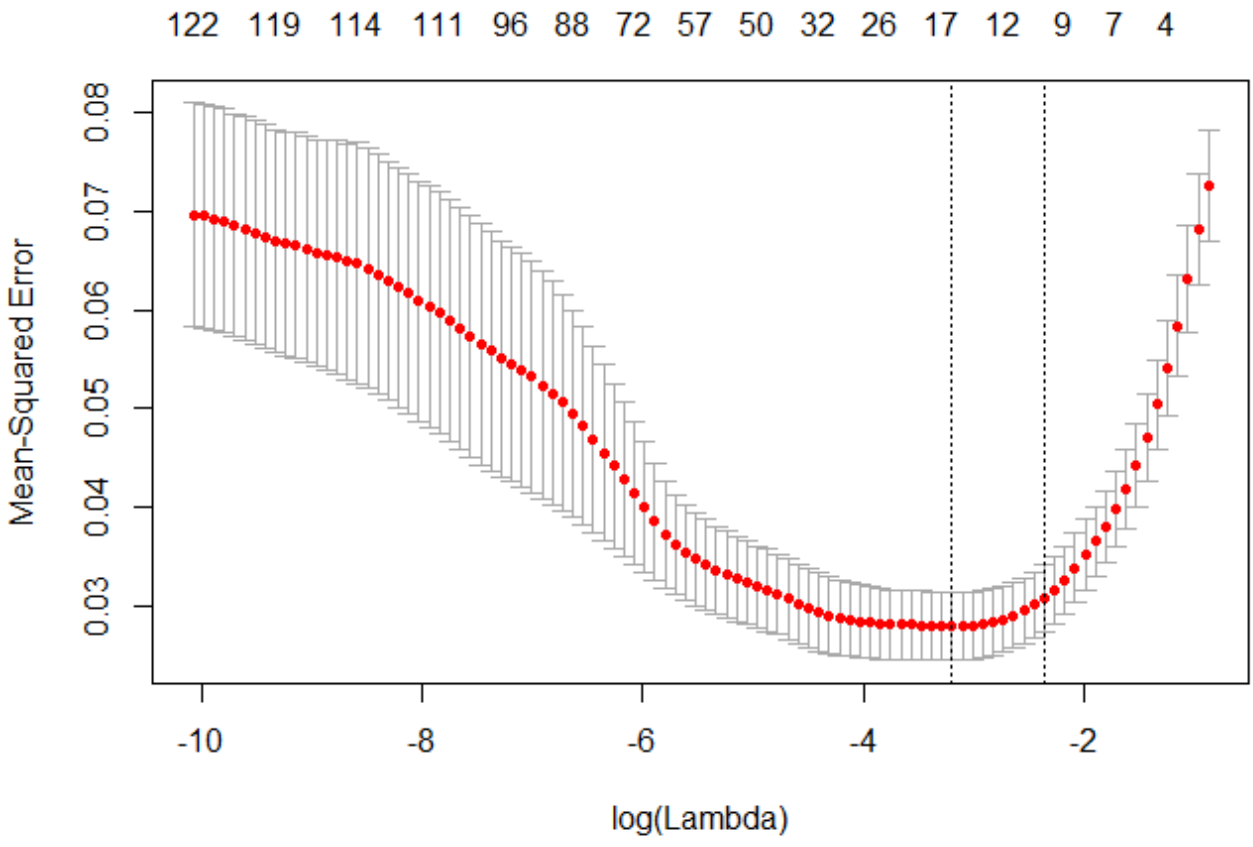
```
a <- coef(fit.lasso, cvfit2$lambda.min)
quantile(abs(as.vector(a)), c(0, 0.25, 0.5, 0.75, 0.9, 0.95, 1))
```

```
        0%         25%         50%         75%         90%         95%
0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.03214922
      100%
0.79261488
```

Note that, compared to Ridge regression, Lasso sets 90% of the coefficients exactly equal to 0, which helps a lot in subset selection.

Similar thing is also performed for Elastic net also.

```
cvfit3 <- cv.glmnet(x.train, y.train, alpha = 0.5, nfolds = 10)
plot(cvfit3)
```



```
print(cvfit3$lambda.min)
```

```
[1] 0.04102909
```

```
a <- coef(fit.elnet, cvfit3$lambda.min)
quantile(abs(as.vector(a)), c(0, 0.25, 0.5, 0.75, 0.9, 0.95, 1))
```

```
        0%        25%        50%        75%        90%        95%
0.00000000 0.00000000 0.00000000 0.00000000 0.02574237 0.07642714
      100%
0.69045548
```

Note that, compared to Ridge and Lasso regression, Elastic net sets 75% of the coefficients exactly equal to 0, which is midway between Ridge & Lasso performances.

To compare between these three models, the test data will be used.

```
fit.ridge_final <- glmnet(x.train, y.train, family = "gaussian",
                          alpha = 0, lambda = cvfit1$lambda.min)
fit.lasso_final <- glmnet(x.train, y.train, family = "gaussian",
                          alpha = 1, lambda = cvfit2$lambda.min)
fit.elnet_final <- glmnet(x.train, y.train, family = "gaussian",
                          alpha = 0.5, lambda = cvfit3$lambda.min)
```

Now we predict responses for the test data using these models, and evaluate the value of MSE.

```
pred_ridge <- predict(fit.ridge_final, x.test)
# MSE for Ridge Regression Prediction
mean((y.test - pred_ridge)^2)
```

```
[1] 0.03451028
```

```
# MSE for Lasso Regression Prediction
pred_lasso <- predict(fit.lasso_final, x.test)
```

```
mean((y.test - pred_lasso)^2)
```

```
[1] 0.03371714
```

```
# MSE for Elastic Net Prediction
pred_elnet <- predict(fit.elnet_final, x.test)
mean((y.test - pred_elnet)^2)
```

```
[1] 0.03332301
```

Although the MSE values are very close to each other, Elastic Net seems to outperform the Ridge and Lasso. Lasso is performing better than Ridge regression in terms of MSE in test dataset.

Mixed Effect Modelling of Longitudinal Data

In this section, we shall use the **gababies** dataset available in the following link: http://www.biostat.ucsf.edu/vgsm/1st_ed/data/gababies.txt. The dataset is about Georgian infant birth weights. Some features of the data is as follows:

- Birth weight measured for each of m = 5 children of n = 200 mothers.
- Birth weight for infants j comprise repeated measures on mothers i.
- Interested in the association between birth order and birth weight.

```
library(lme4)
data <- read.table("gababies.txt", header = T)
head(data)
```

```
  momid birthord momage timesnc delwght lowbrth bweight lastwght initage
1    39        1     15       0   -1240       0    3720     2480      15
2    39        2     17       2   -1240       0    3260     2480      15
3    39        3     19       4   -1240       0    3910     2480      15
4    39        4     24       9   -1240       0    3320     2480      15
5    39        5     25      10   -1240       1    2480     2480      15
6    62        1     17       0    -170       1    2381     2211      17
  initwght cinitage
1     3720   -2.545
```

```
2     3720   -2.545
3     3720   -2.545
4     3720   -2.545
5     3720   -2.545
6     2381   -0.545
```

Note that, in this data, the birth order gives a temporal effect on birth weight. We also take the age of the mom as a fixed effect covariate.

```
fit <- lmer(bweight ~ birthord + momage + (birthord | momid), data)
summary(fit)
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: bweight ~ birthord + momage + (birthord | momid)
   Data: data

REML criterion at convergence: 15297.6

Scaled residuals:
    Min      1Q  Median      3Q     Max
-5.0881 -0.4407  0.0439  0.5290  3.8485

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 momid    (Intercept)  85028.2 291.60
          birthord       519.4  22.79   1.00
 Residual             196784.3 443.60
Number of obs: 1000, groups:  momid, 200

Fixed effects:
            Estimate Std. Error t value
(Intercept) 2824.958     75.224  37.554
birthord      23.746     13.246   1.793
momage        11.060      4.175   2.649

Correlation of Fixed Effects:
         (Intr) brthrd
birthord  0.295
momage   -0.857 -0.651
convergence code: 0
boundary (singular) fit: see ?isSingular
```

From the summary above, we find that the final fitted model looks as follows;

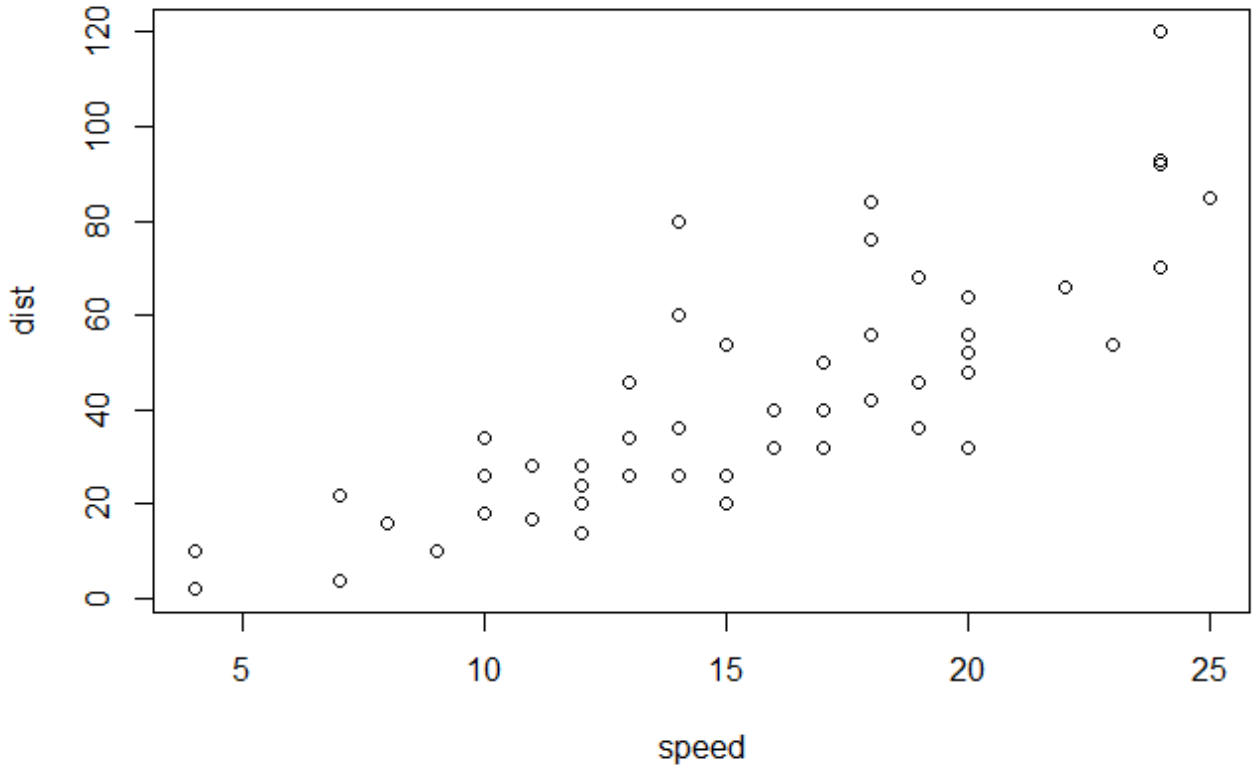
$$y_{it} = 2824.958 + 23.746t + 11.06a_i + (b_{0,it} + b_{1,it}t)$$

where y_{it} is the weight of the t-th child of i-th mother, a_i is the age of the mother at the time of giving birth, while $b_{0,it}$ and $b_{1,it}$ are random effects drawn from the normal distribution with mean 0 and variance 85028.2 and 519.4 respectively.

Non-parametric Regression Technique

For this section, we shall use **cars** dataset from **datasets** package in R. This dataset contains data on speed and the stopping distance of different cars. The goal is to model the stopping distance as a function of the speed. Before proceeding, let us first take a look at the data.

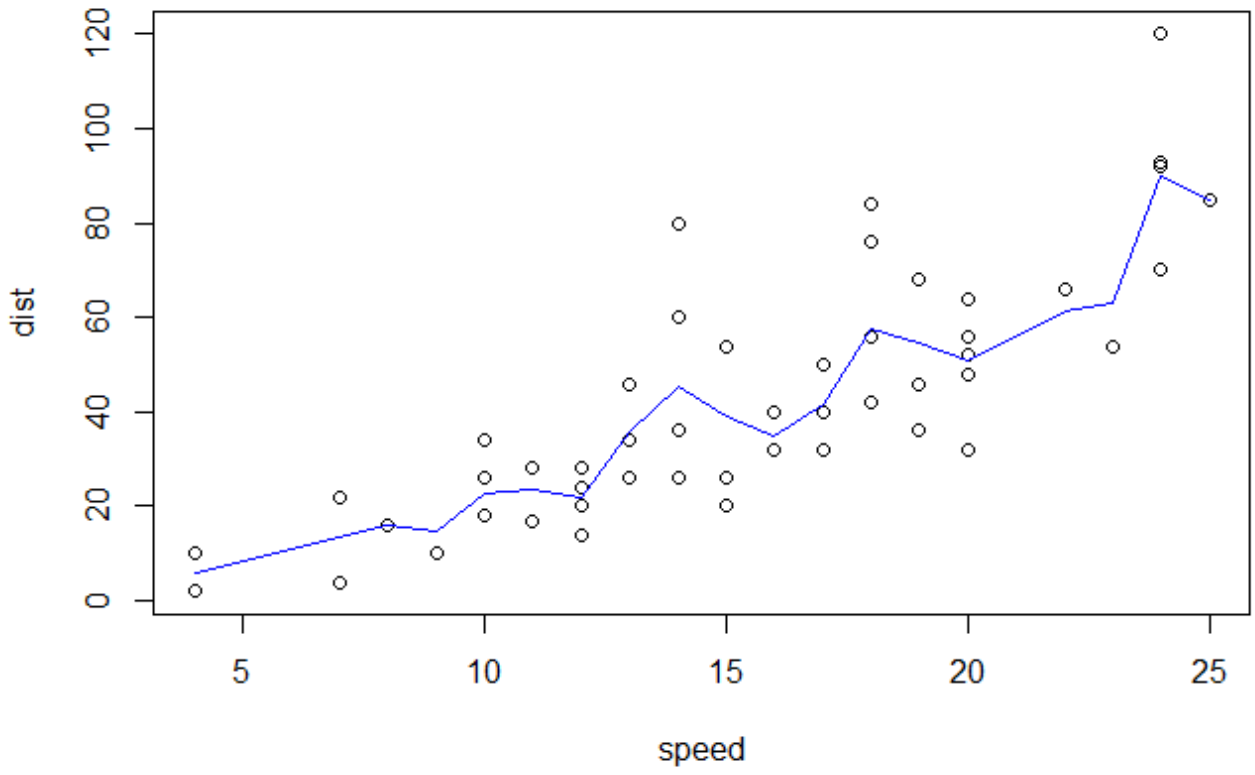
```
data(cars)
plot(dist ~ speed, cars)
```



Note that, there is a positive correlation between speed and stopping distance as suggested by the figure. However, a nonlinear function (possibly quadratic) would be able to capture a general trend of the relationship.

Firstly, we fit a local nearest neighbour regression curve.

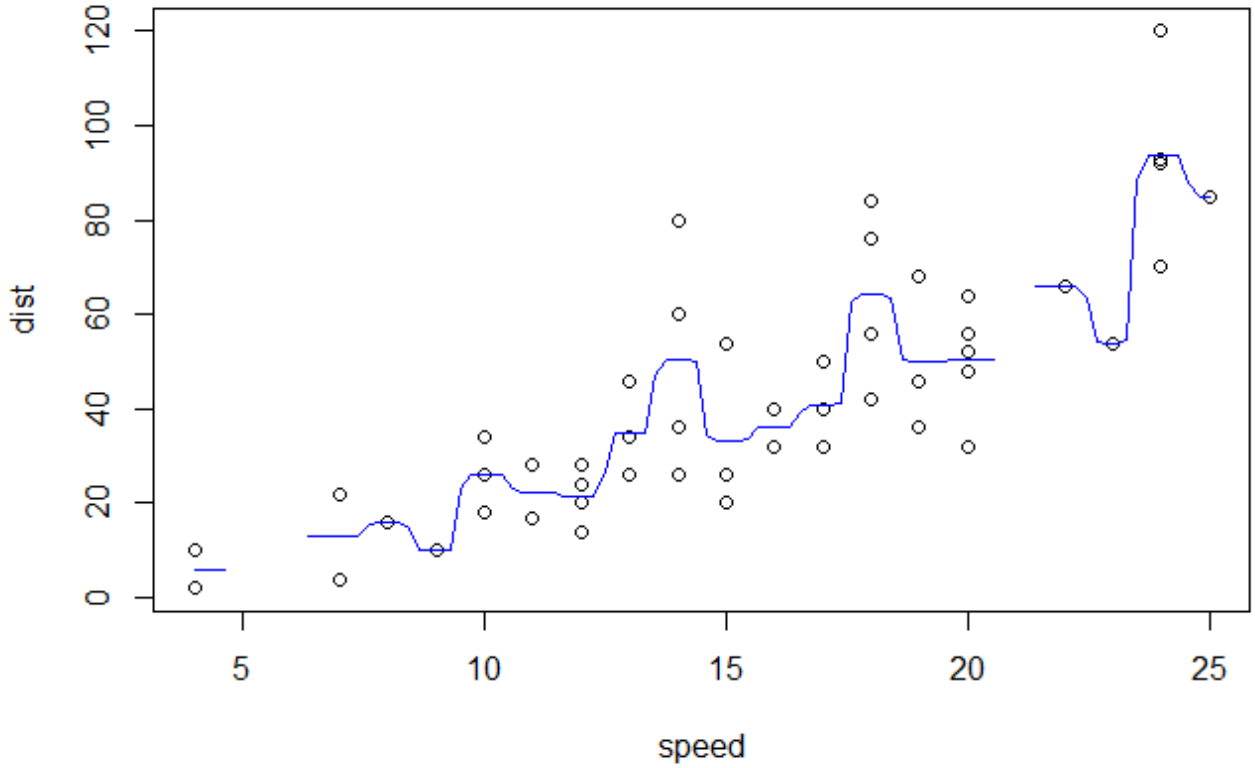
```
library(locfit)
fit1 <- locfit(dist ~ speed, data = cars, alpha = c(0,2))
#alpha denotes the number of nearest neighbour

plot(dist ~ speed, cars)
lines(cars$speed, fitted(fit1), col = "blue")
```



Note that, the above curve passes through the cluser of the data, it approximately the general trend well, but it is not very smooth.

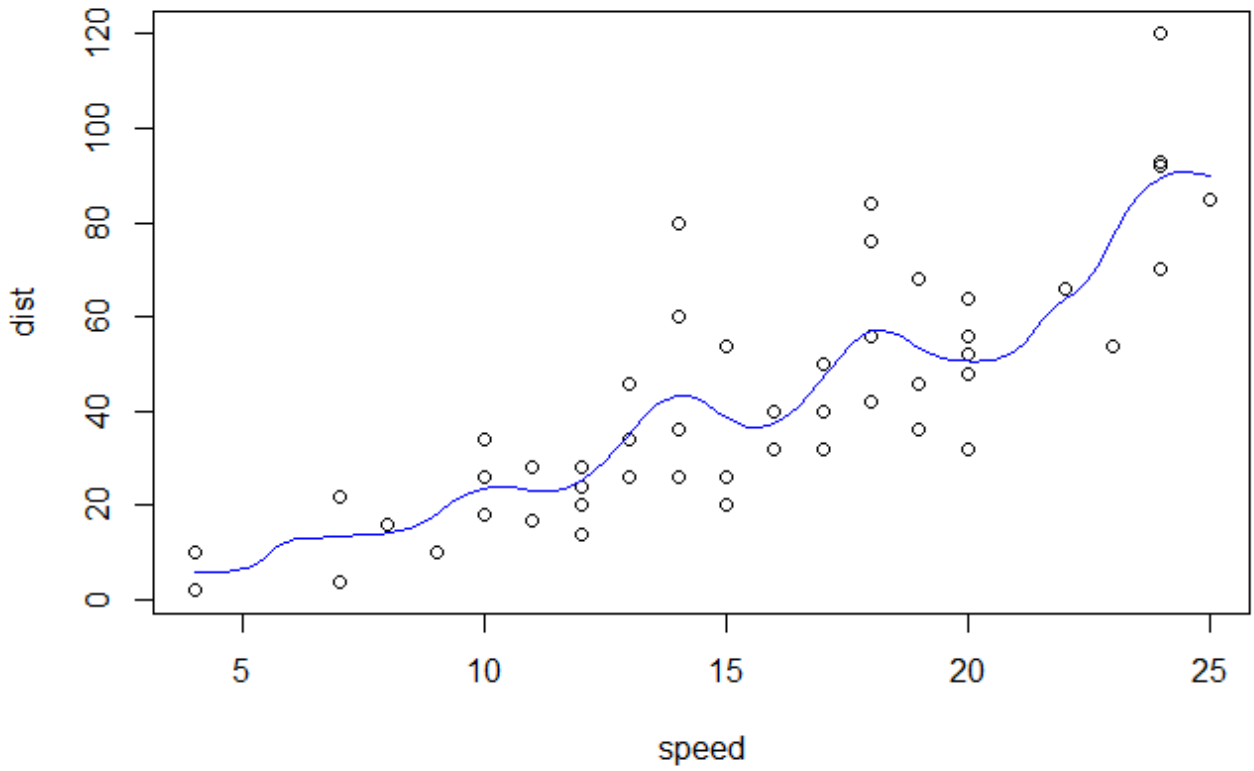
Therefore, we use a Kernel smoothing technique to estimate the functional form of the mean function.

```
fit2 <- ksmooth(cars$speed, cars$dist, kernel = "normal")
plot(dist ~ speed, cars)
lines(fit2, col = "blue")
```



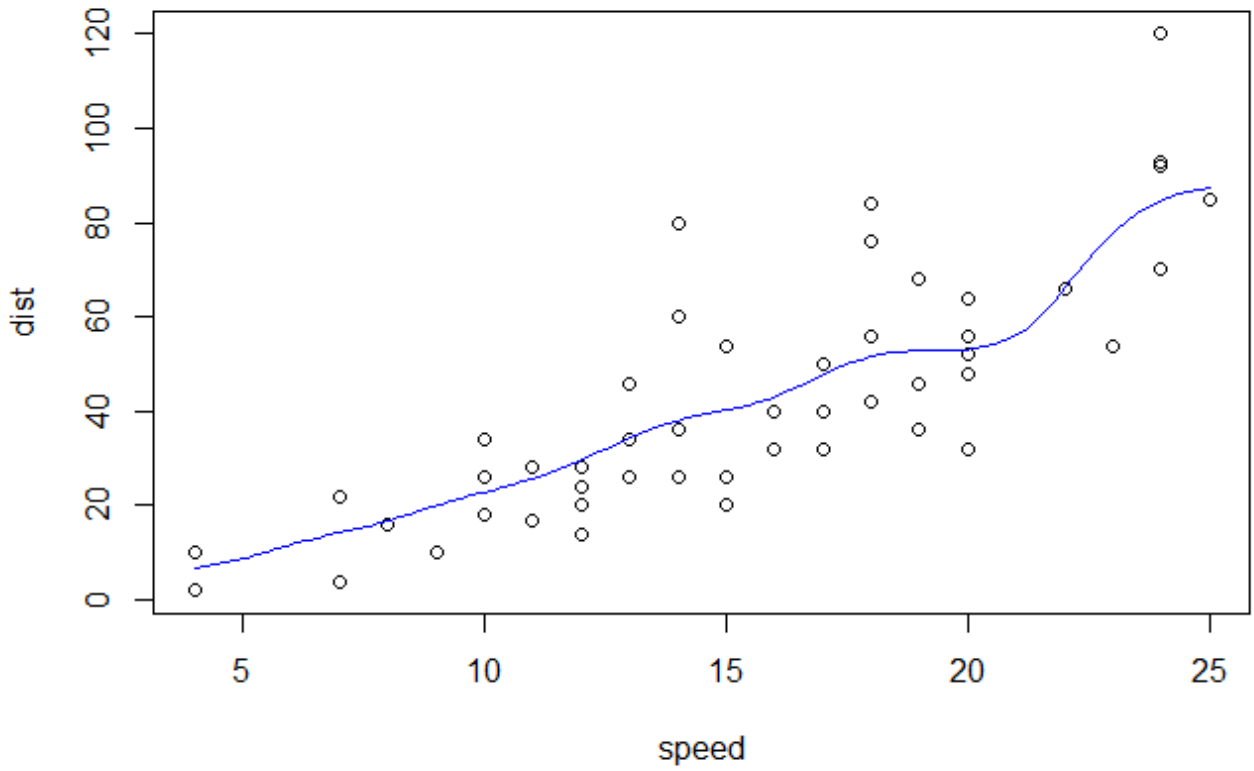
We find that having a lower bandwidth (default 0.5) can even make the kernel smoother a discontinuous function. Therefore, we increase the bandwidth to make it more smoother and continuous.

```
fit3 <- ksmooth(cars$speed, cars$dist,
                kernel = "normal", bandwidth = 2)
plot(dist ~ speed, cars)
lines(fit3, col = "blue")
```



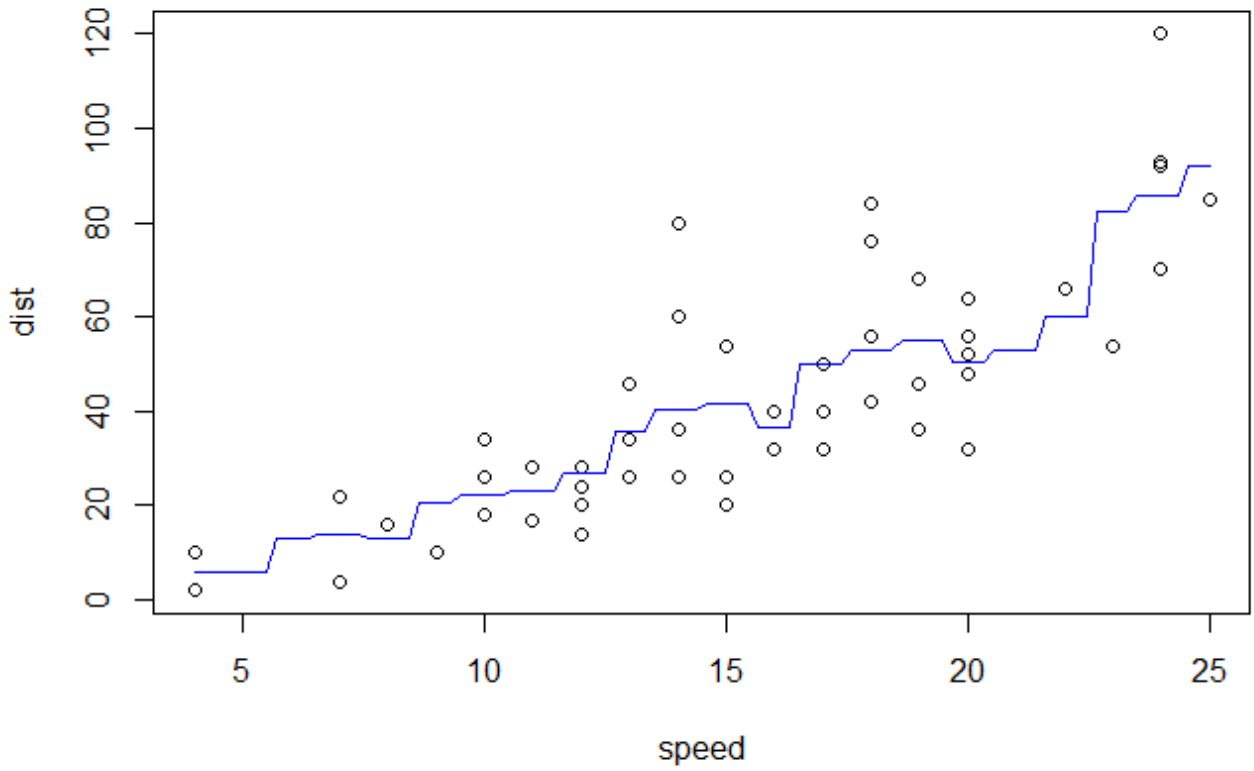
We have a good smooth curve, but it is still not simple enough to be able to be used for prediction purpose. So, we increase the bandwidth even more to see the smoothing effect.

```
fit4 <- ksmooth(cars$speed, cars$dist,
                kernel = "normal", bandwidth = 4)
plot(dist ~ speed, cars)
lines(fit4, col = "blue")
```



We find that, increasing the bandwidth makes the curve more close to a linear curve, at the same time, it fails to capture the highly varying part between speed 12 to 15.

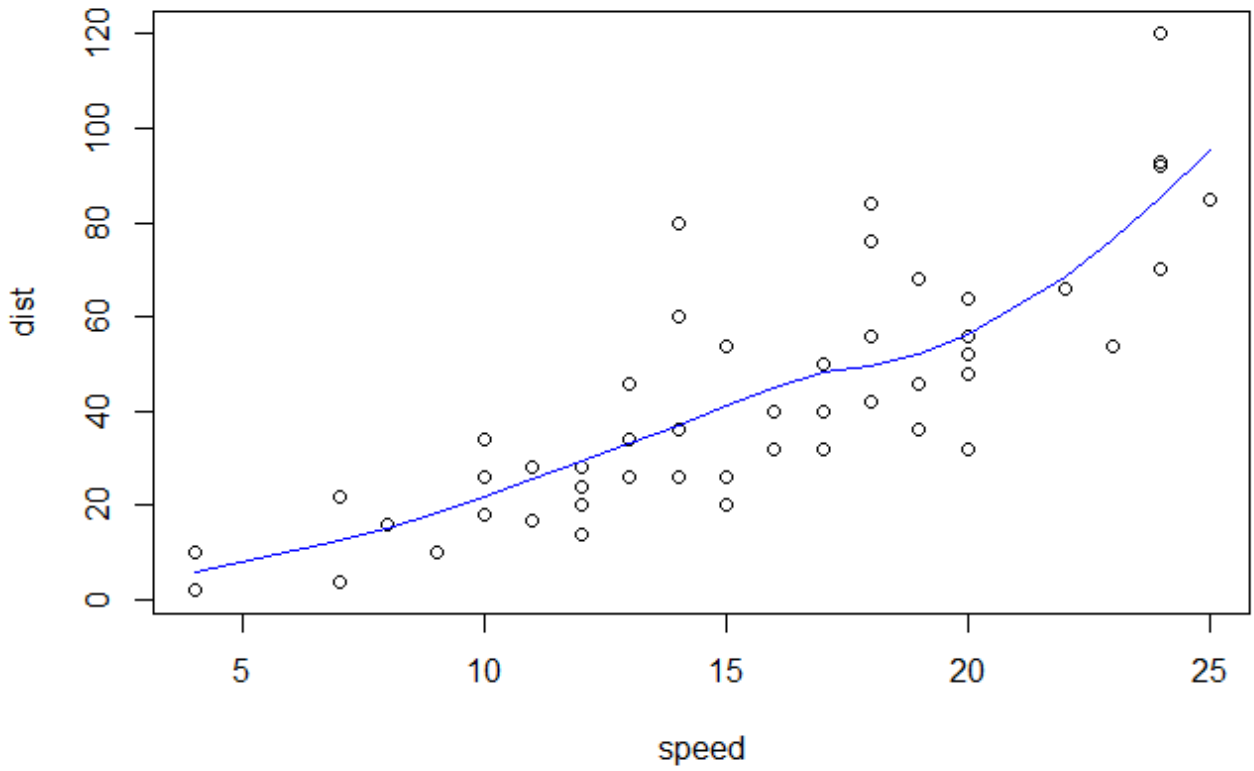
```
fit5 <- ksmooth(cars$speed, cars$dist,
                kernel = "box", bandwidth = 3)
plot(dist ~ speed, cars)
lines(fit5, col = "blue")
```



We find that, usage of box kernel makes a more jagged kind of curve in order to fit through the data.

Finally, we use a Local polynomial regression method to model the data.

```
fit <- loess(dist ~ speed, data = cars, family = "gaussian")
plot(dist ~ speed, cars)
lines(cars$speed, fitted(fit), col = "blue")
```



We find that this results in a much smoother curve as well as being able to capture the mean function of the data pretty well.

THANK YOU