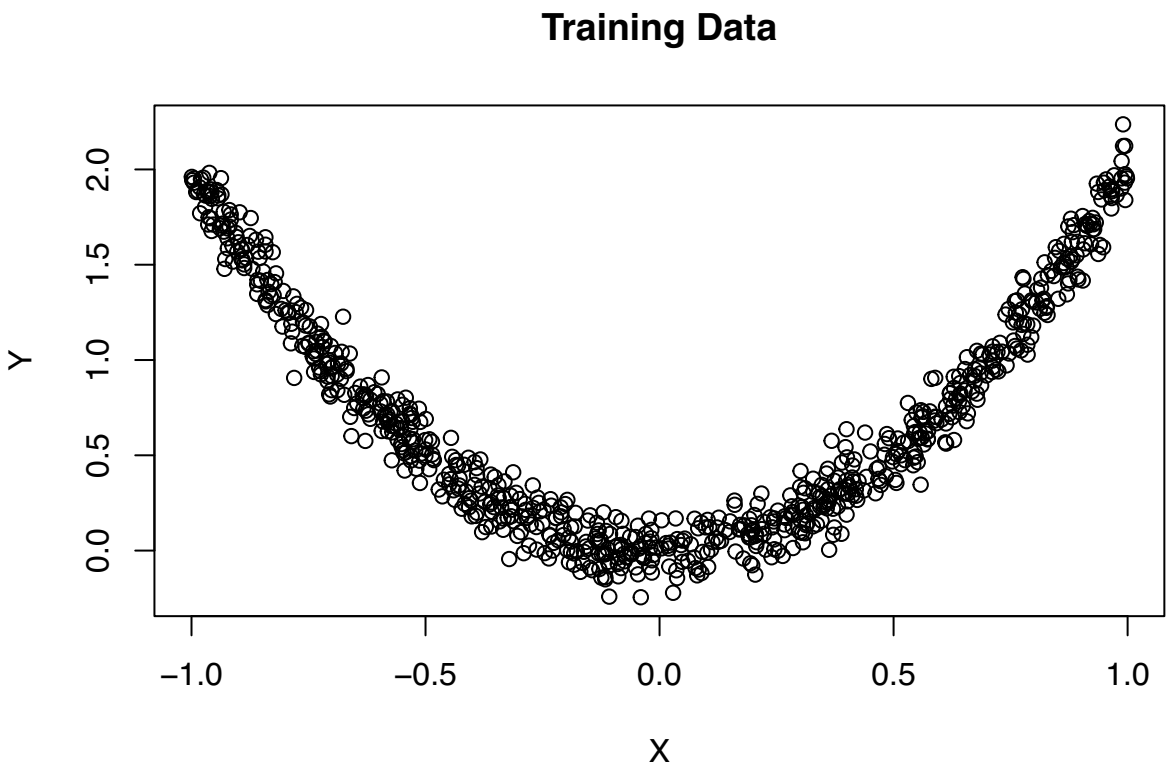
hw6 - R

Question 1

a)

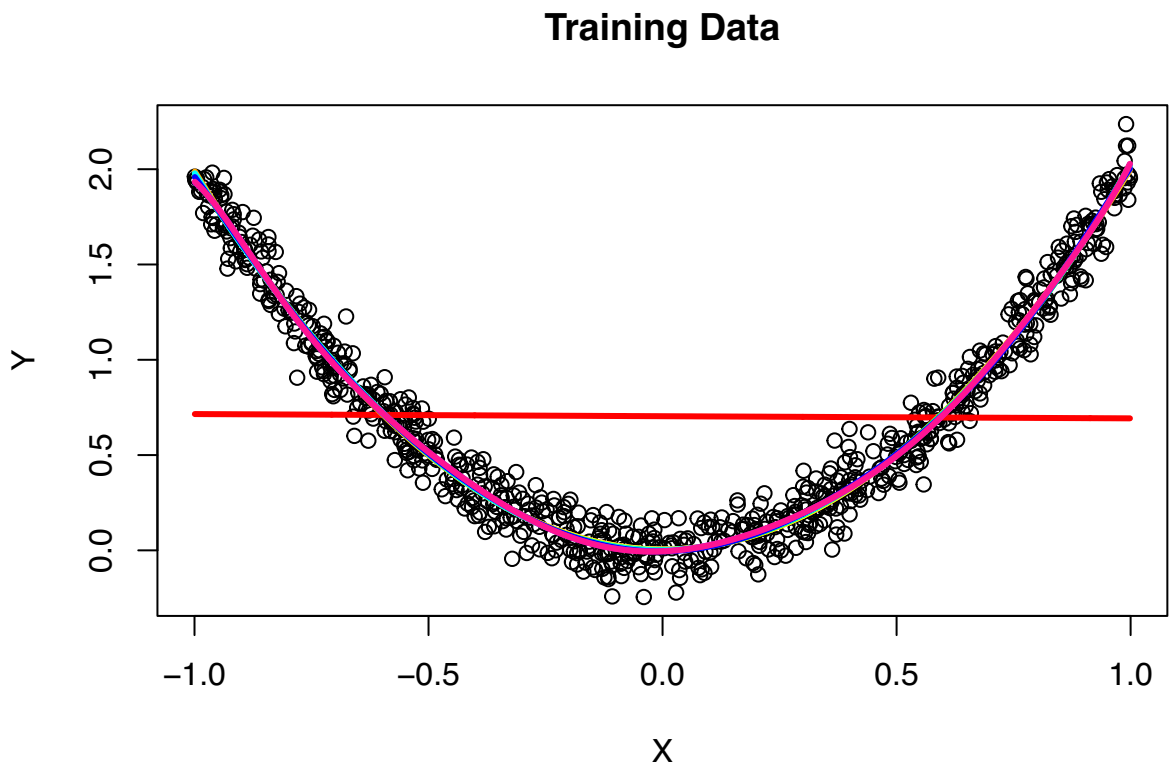
```
set.seed(101)

n = 1000
X = runif(n,-1,1)
epsilon = rnorm(n,0,.1)
Y = 2*X^2 + epsilon

sample = sample(c(rep(0, 200), rep(1, 800)), replace=FALSE)
train = data.frame(X=X[sample == 1], Y=Y[sample == 1])
test = data.frame(X=X[sample == 0], Y=Y[sample == 0])

plot(train$X, train$Y, xlab="X", ylab="Y", main="Training Data")
```



b)

```
out1 = lm(Y ~ poly(X, 1), data=train)
out2 = lm(Y ~ poly(X, 2), data=train)
out3 = lm(Y ~ poly(X, 3), data=train)
out4 = lm(Y ~ poly(X, 4), data=train)
out5 = lm(Y ~ poly(X, 5), data=train)
out6 = lm(Y ~ poly(X, 6), data=train)
out7 = lm(Y ~ poly(X, 7), data=train)
out8 = lm(Y ~ poly(X, 8), data=train)
out9 = lm(Y ~ poly(X, 9), data=train)
out10 = lm(Y ~ poly(X, 10), data=train)

plot(train$X, train$Y, xlab="X", ylab="Y", main="Training Data")

lines(sort(train$X), fitted(out1)[order(train$X)], col="red", lwd=3)
lines(sort(train$X), fitted(out2)[order(train$X)], col="orange", lwd=3)
lines(sort(train$X), fitted(out3)[order(train$X)], col="coral", lwd=3)
lines(sort(train$X), fitted(out4)[order(train$X)], col="yellow", lwd=3)
lines(sort(train$X), fitted(out5)[order(train$X)], col="darkolivegreen1", lwd=3)
lines(sort(train$X), fitted(out6)[order(train$X)], col="green", lwd=3)
lines(sort(train$X), fitted(out7)[order(train$X)], col="cyan", lwd=3)
lines(sort(train$X), fitted(out8)[order(train$X)], col="blue", lwd=3)
lines(sort(train$X), fitted(out9)[order(train$X)], col="purple", lwd=3)
```

```
lines(sort(train$X), fitted(out10)[order(train$X)], col="deeppink", lwd=3)
```

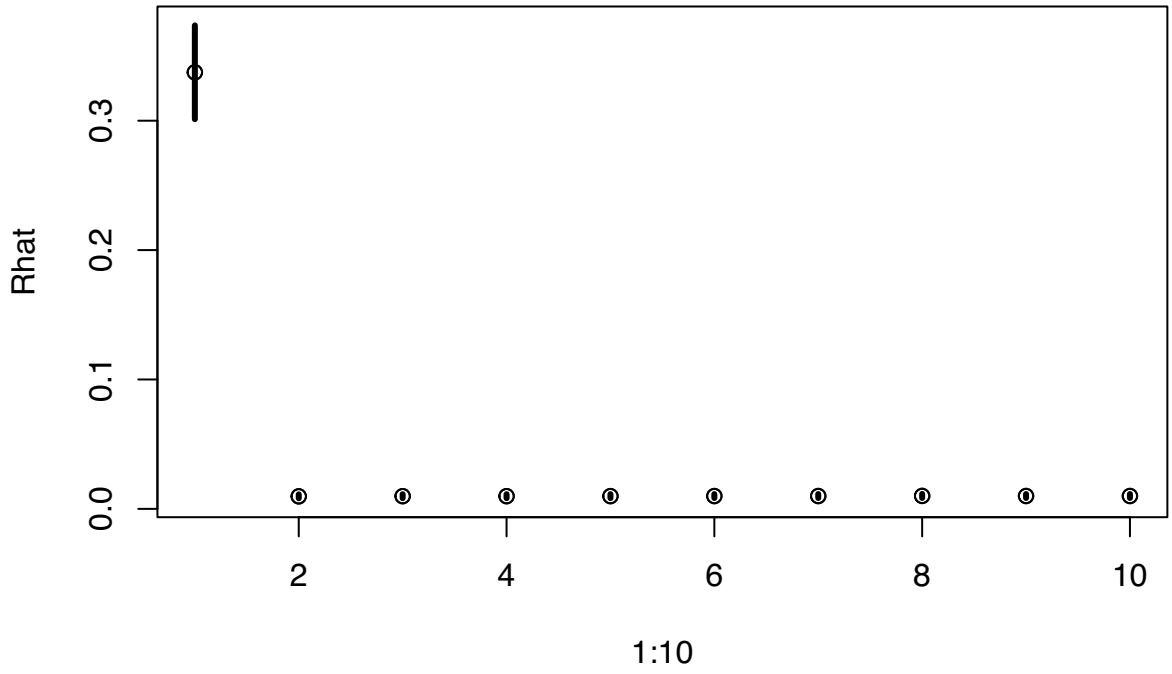


c)

```
newx = data.frame(X=test$X)
tmp1 = predict(out1,se.fit=TRUE,newdata=newx)
tmp2 = predict(out2,se.fit=TRUE,newdata=newx)
tmp3 = predict(out3,se.fit=TRUE,newdata=newx)
tmp4 = predict(out4,se.fit=TRUE,newdata=newx)
tmp5 = predict(out5,se.fit=TRUE,newdata=newx)
tmp6 = predict(out6,se.fit=TRUE,newdata=newx)
tmp7 = predict(out7,se.fit=TRUE,newdata=newx)
tmp8 = predict(out8,se.fit=TRUE,newdata=newx)
tmp9 = predict(out9,se.fit=TRUE,newdata=newx)
tmp10 = predict(out10,se.fit=TRUE,newdata=newx)

B1 = (test$Y - tmp1$fit)^2
B2 = (test$Y - tmp2$fit)^2
B3 = (test$Y - tmp3$fit)^2
B4 = (test$Y - tmp4$fit)^2
B5 = (test$Y - tmp5$fit)^2
B6 = (test$Y - tmp6$fit)^2
B7 = (test$Y - tmp7$fit)^2
```

```
B8 = (test$Y - tmp8$fit)^2
B9 = (test$Y - tmp9$fit)^2
B10 = (test$Y - tmp10$fit)^2

R1 = mean(B1)
R2 = mean(B2)
R3 = mean(B3)
R4 = mean(B4)
R5 = mean(B5)
R6 = mean(B6)
R7 = mean(B7)
R8 = mean(B8)
R9 = mean(B9)
R10 = mean(B10)

s1 = sqrt(sum((B1 - R1)^2)/199)
s2 = sqrt(sum((B2 - R2)^2)/199)
s3 = sqrt(sum((B3 - R3)^2)/199)
s4 = sqrt(sum((B4 - R4)^2)/199)
s5 = sqrt(sum((B5 - R5)^2)/199)
s6 = sqrt(sum((B6 - R6)^2)/199)
s7 = sqrt(sum((B7 - R7)^2)/199)
s8 = sqrt(sum((B8 - R8)^2)/199)
s9 = sqrt(sum((B9 - R9)^2)/199)
s10 = sqrt(sum((B10 - R10)^2)/199)

low1 = R1 - 1.645*s1/sqrt(200)
low2 = R2 - 1.645*s2/sqrt(200)
low3 = R3 - 1.645*s3/sqrt(200)
low4 = R4 - 1.645*s4/sqrt(200)
low5 = R5 - 1.645*s5/sqrt(200)
low6 = R6 - 1.645*s6/sqrt(200)
low7 = R7 - 1.645*s7/sqrt(200)
low8 = R8 - 1.645*s8/sqrt(200)
low9 = R9 - 1.645*s9/sqrt(200)
low10 = R10 - 1.645*s10/sqrt(200)

up1 = R1 + 1.645*s1/sqrt(200)
up2 = R2 + 1.645*s2/sqrt(200)
up3 = R3 + 1.645*s3/sqrt(200)
up4 = R4 + 1.645*s4/sqrt(200)
up5 = R5 + 1.645*s5/sqrt(200)
up6 = R6 + 1.645*s6/sqrt(200)
up7 = R7 + 1.645*s7/sqrt(200)
up8 = R8 + 1.645*s8/sqrt(200)
up9 = R9 + 1.645*s9/sqrt(200)
up10 = R10 + 1.645*s10/sqrt(200)

Rhat = c(R1, R2, R3, R4, R5, R6, R7, R8, R9, R10)
low = c(low1, low2, low3, low4, low5, low6, low7, low8, low9, low10)
up = c(up1, up2, up3, up4, up5, up6, up7, up8, up9, up10)

a = min(low)
```

```
b = max(up)

plot(1:10, Rhat, ylim=c(a,b))
for(i in 1:10) {
  segments(i, low[i], i, up[i], lwd=3)
}
points(1:10, Rhat)
```



d)

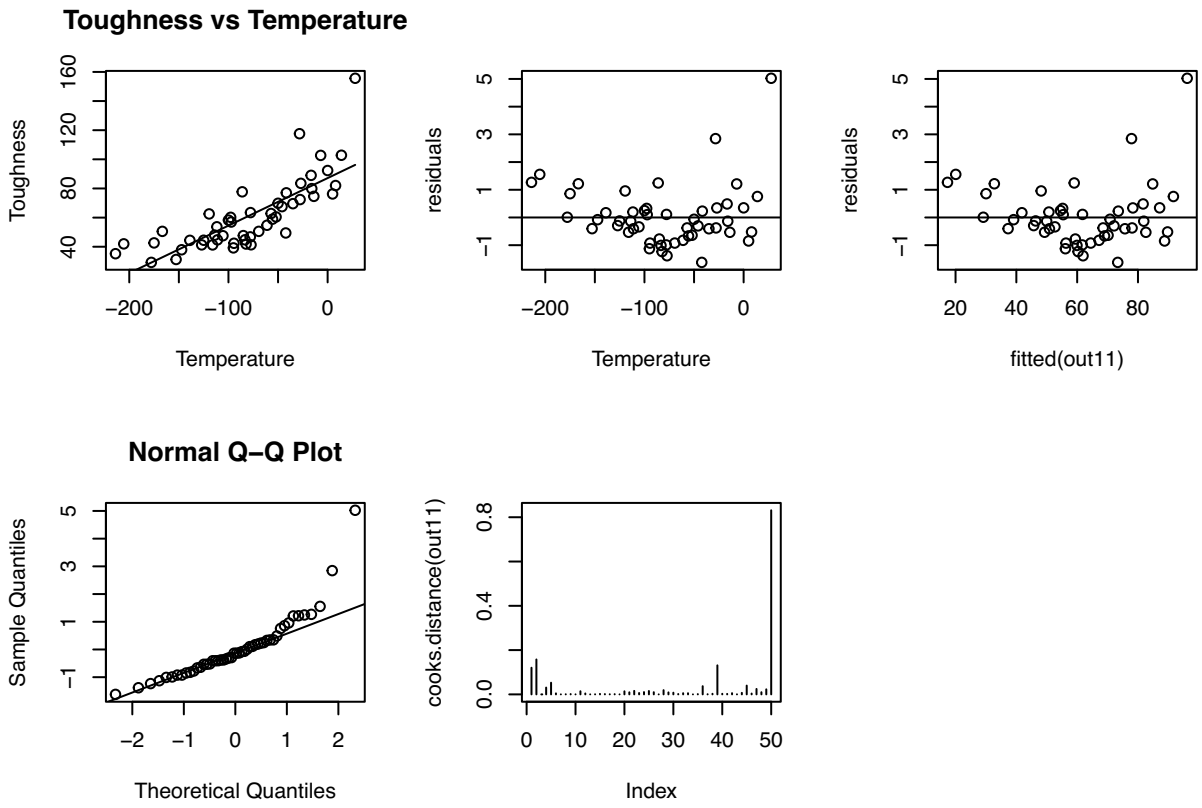
We see from the plot in 1c that the models with a polynomial degree of 2 or higher have lower estimated prediction errors than the model with a polynomial degree of 1. Because we want a low prediction error, we conclude that the models with a polynomial degree of 2 or higher are best. We note that the models with a polynomial degree of 2 or higher do not seem to have much difference between them in terms of their estimated prediction errors or their confidence intervals of their prediction errors. As such, we specifically conclude that the model with a polynomial degree of 2 is the best model because it has one of the lowest estimated prediction errors and because it has the lowest degree out of the models that had all relatively similar, low estimated prediction errors. We choose the lowest degree to keep our selected model simple and avoid overfitting.

Question 2

```
df = read.table("nuclear.txt", header=TRUE)
attach(df)
Y = split(toughness,temperature)
X = split(temperature,temperature)

X = sapply(X,"mean")
Y = sapply(Y,"mean")
```

a)

```
par(mfrow=c(2,3))

out11 = lm(Y ~ X)
plot(X, Y, xlab = "Temperature", ylab="Toughness",
     main="Toughness vs Temperature")
lines(X, fitted(out11))

r = rstudent(out11)
plot(X, r, xlab = "Temperature", ylab="residuals")
abline(h=0)
plot(fitted(out11), r, ylab="residuals")
abline(h=0)
qqnorm(r)
qqline(r)
plot(cooks.distance(out11), type="h")
```



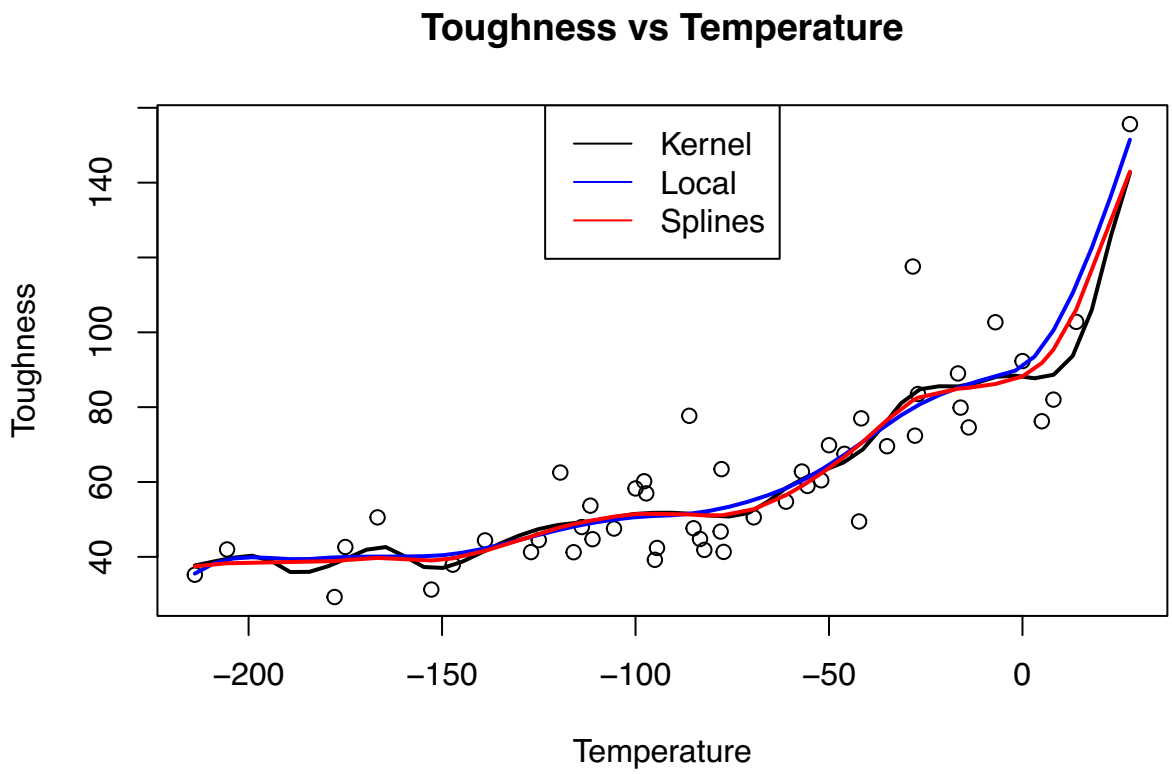
Looking at the plot of Toughness vs Temperature, we note that the linear regression model is not a good summary of the relationship between Temperature and Toughness given that many data points are not on or relatively close to the regression line. We see that the residuals plots have curving upwards, non-linear patterns, which indicates that the linear assumption is wrong. The plot also demonstrates to us that the residuals seem to slightly fan out more and more, so there seems to be increasing or non-constant variance, which indicates that the assumption of homoscedasticity is violated. We also observe that there are a couple of large positive residuals that are potential outliers. From the qqplot, we see that the points get farther and farther away from the line in the upper tail. Because the points do not look roughly like a straight line, we conclude that the residuals are not approximately normal. There do not seem to be any influential points given that none of the values for cook's distance are greater than 1.

b)

```
H = seq(3,100)
grid = seq(min(temperature),max(temperature),length=50)

Kernel = function(x,y,grid,h){
  n = length(x)
  k = length(grid)
  m.hat = rep(0,k)
```

```
  for(i in 1:k){
    w = dnorm(grid[i],x,h)
    m.hat[i] = sum(y*w)/sum(w)
  }
  return(m.hat)
}

Cross_Validation = function(x,y,H){
  n = length(x)
  nH = length(H)
  CV = rep(0,nH)
  GCV = rep(0,nH)
  Y = as.matrix(y)
  df = rep(0,nH)
  for(j in 1:nH){
    h = H[j]
    L = matrix(0,n,n)
    for(i in 1:n){
      w = dnorm(x[i],x,h)
      w = w/sum(w)
      L[i,] = w
    }
    fit = L %*% Y
    d = diag(L)
    CV[j] = mean( ((y - fit)/(1 - d))^2 )
    df[j] = sum(d)
    GCV[j] = mean((y - fit)^2)/(1 - df[j]/n)^2
  }
  return(list(CV=CV,GCV=GCV,df=df))
}

kernel_cv = Cross_Validation(X, Y, H)
kernel_h = H[which.min(kernel_cv$GCV)]
kernel = Kernel(X, Y, grid, kernel_h)

H = seq(14,100)
nH = length(H)
A = cbind(rep(0,nH),H)
local_gcv = gcvplot(Y ~ X,alpha=A,deg=1)
h = H[which.min(local_gcv$values)]
local_out = locfit(Y ~ lp(X,h=h,deg=1))
tmp = predict(local_out,se.fit=TRUE,newdata=grid)
f = tmp$fit

s = smooth.spline(X, Y)

plot(X, Y, xlab = "Temperature", ylab="Toughness",
     main="Toughness vs Temperature")
lines(grid, kernel, lwd=2)
lines(grid,f,col="blue", lwd=2)
lines(s$x, s$y, col="red", lwd=2)
legend(x = "top",
       col = c("black", "blue", "red"), lty = 1, lwd = 1,
```

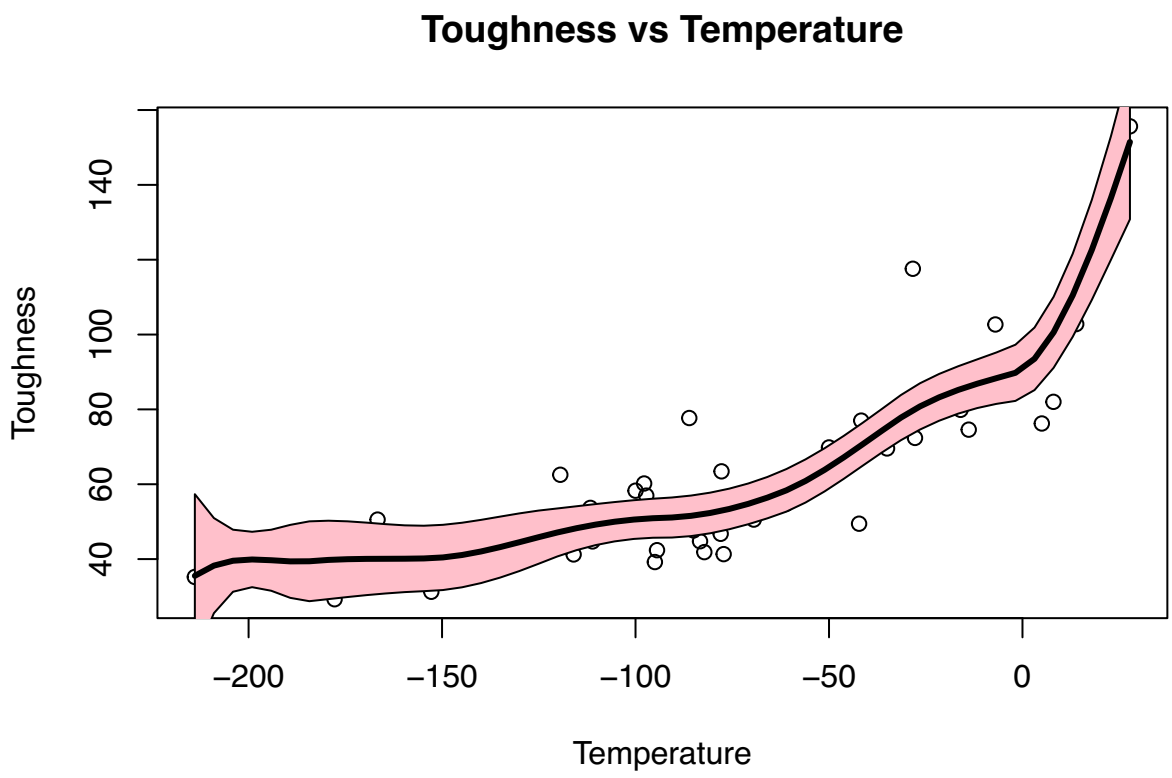
```
       legend = c('Kernel', 'Local', 'Splines'))
```



From the plot, we see that the three curves all look roughly similar. The kernel regression curve seems to be the least smooth out of the three, indicating it has less bias and more variance than the other two models. The curves for the local linear regression and the smoothing splines look almost identical until the second half of the plot, where the smoothing splines curve seems to start curving more and matching more of the small aspects of the data. Thus, it seems that the curve for the local linear regression is the most smooth out of the three, indicating that the local linear regression model has more bias and less variance than the smoothing splines model and the kernel regression model.

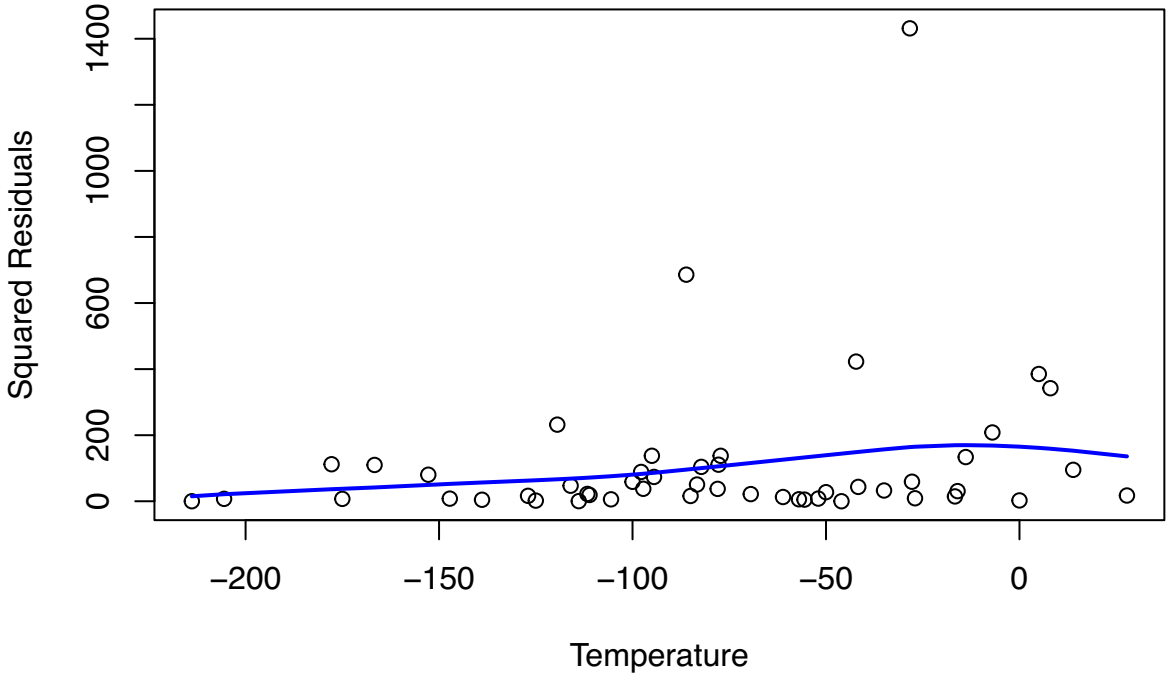
c)

```
up = tmp$fit + 1.96*tmp$se.fit
low = tmp$fit - 1.96*tmp$se.fit
plot(X,Y, xlab = "Temperature", ylab="Toughness",
     main="Toughness vs Temperature")
polygon(c(grid,rev(grid)),c(low,rev(up)),col="pink")
lines(grid,f,lwd=3)
```



d)

```
m.hat = predict(local_out,se.fit=TRUE,newdata=data.frame(X))
Z = (Y - m.hat$fit)^2

H = seq(14,100)
nH = length(H)
A = cbind(rep(0,nH),H)
sigma_gcv = gcvplot(Z ~ X,alpha=A,deg=1)
h = H[which.min(sigma_gcv$values)]
sigma_out = locfit(Z ~ lp(X,h=h,deg=1))
sigma_tmp = predict(sigma_out,se.fit=TRUE,newdata=data.frame(X))
sigma_f = sigma_tmp$fit
plot(X, Z, xlab = "Temperature", ylab="Squared Residuals")
lines(X,sigma_f,col="blue", lwd=2)
```



e)

Yes, the analysis does support the belief that the true regression function is an increasing function. We see that the kernel regression, the local linear regression and the smoothing splines estimates of the true regression function all were increasing functions. The confidence band for the local linear regression has roughly the same increasing trend as well, so we conclude that the true regression function is likely also an increasing function.