Project V: Parametric/Nonparametric Nonlinear Regression

DS 5494 - Statistical Data Mining II

Willliam Ofosu Agyapong[*]   University of Texas at El Paso (UTEP)

November 08, 2022

Contents

[*]woagyapong@miners.utep.edu

1 Introduction

We consider a data set jaws.txt, which is concerned about the association between jaw bone length (y = bone) and age in deer (x = age). We are going try out several parametric/nonparametric nonlinear regression models in this low-dimensional (p = 1) setting.

1.1 Bringing in the data

```
jaws <- read.table("jaws.txt", header = T)
# head(jaws)
# tail(jaws)
# dim(jaws)
jaws %>%
  slice_sample(n=6) %>%
  kable(booktabs=T, linesep="",
        caption = "10 random samples from the jaws data set") %>%
  kable_styling(latex_options = c("HOLD_position")) %>%
  kable_classic()
```

Table 1: 10 random samples from the jaws data set

age	bone
47.700	112.43
16.076	105.97
17.063	112.26
13.998	85.77
1.633	26.15
49.330	142.00

1.2 Relationship between Bone and Age

```
ggplot(jaws, aes(age, bone)) +
    geom_point() +
    geom_smooth(method = "lm", se=F, size=0.7, aes(color="Linear")) + # linear fit
    geom_smooth(method = "loess", se=F, size=0.7, aes(color="Loess")) + # nonlinear loess fit
    scale_color_manual(name="Fit", values = c("blue", "red")) +
    labs(x="Age", y="Jaw bone length") +
    theme_bw()
```

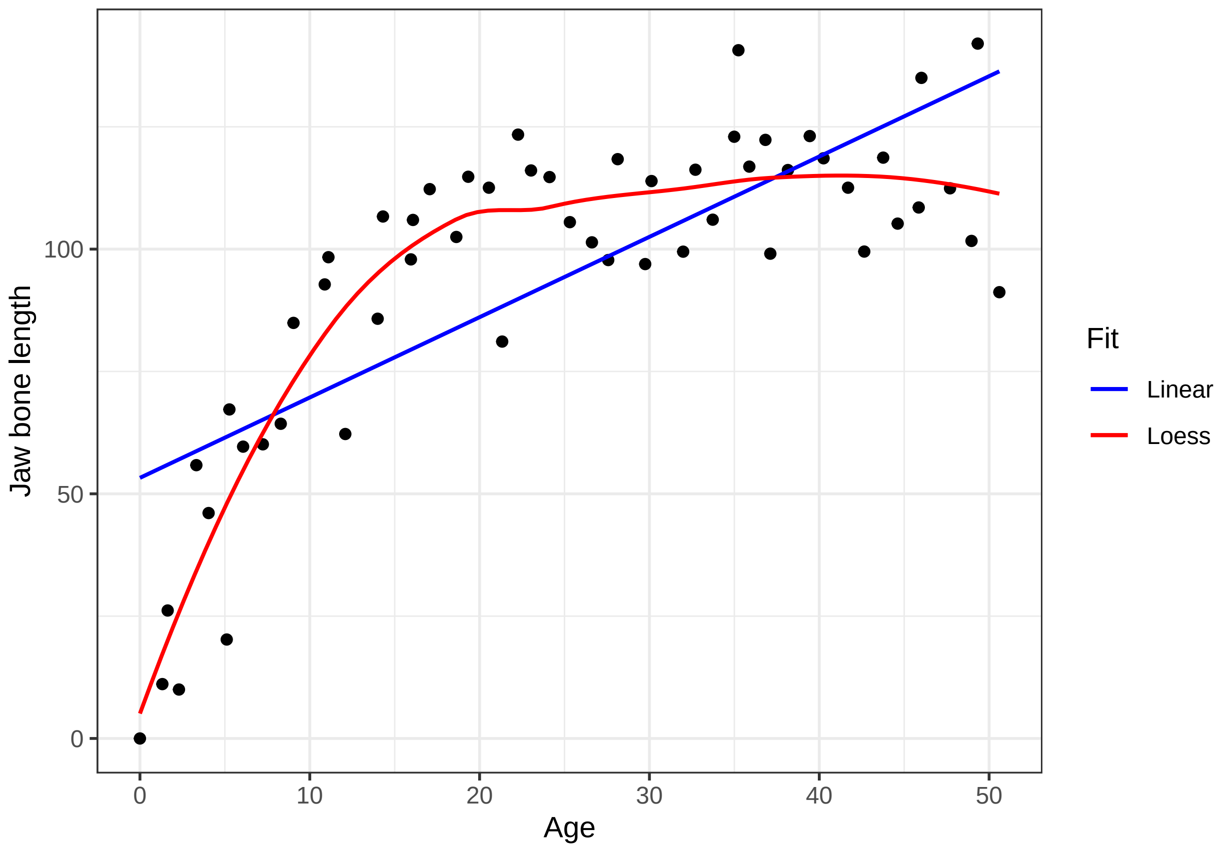


Figure 1: Relationship between jaw bone length and age in deer

The association between Bone and Age look non-linear. Therefore, a nonlinear fit may be more appropriate to model the relationship between the two variables.

2 Data Partitioning

2.1 Part (a): Randomly splitting data into training and test sets

First, we randomly partition the data D into the training set D1 and the test set D2 with a ratio of approximately 2:1 on the sample size. The output below shows that 36 observations and 18 observations were allocated to resulting training and test sets, respectively.

```
set.seed(9940)
ratio <- 2/3
train_ind <- sample(1:NROW(jaws), size = NROW(jaws)*ratio)
train_set <- jaws[train_ind, ]
test_set <- jaws[-train_ind,]
dim(train_set); dim(test_set)
```

```
## [1] 36  2
```

```
## [1] 18  2
```

2.2 Part (b): Preventing extrapolation when it comes to prediction

Next, we make sure that the observations with the minimum and maximum ages end up in the training set.

```
min_max_age1 <- jaws %>% summarise(min_age=min(age), max_age=max(age))
min_max_age2 <- train_set %>% summarise(min_age=min(age), max_age=max(age))
```

```
min_max_age3 <- test_set %>% summarise(min_age=min(age), max_age=max(age))
df <- rbind(min_max_age1, min_max_age2, min_max_age3)
rownames(df) <- c("Original data", "Train set", "Test set")
kable(df, booktabs=T, col.names = c("Minimum age", "Maximum age")) %>%
 kable_styling(latex_options = c("HOLD_position")) %>%
 kable_classic()
```

	Minimum age	Maximum age
Original data	0.00	50.60
Train set	0.00	50.60
Test set	1.32	48.96

Luckily for us, with our choice of seed for reproducible results, it turns out that the minimum and maximum ages in the original `jaws` dataset ended up in our training set, so no further action is needed to avoid extrapolation when making predictions on the test data.

3 Parametric nonlinear models

3.1 Part (a): An Asymptotic exponential model

We fit an asymptotic exponential model of the form

$$y = \beta_1 - \beta_2 e^{-\beta_3 x} + \epsilon$$

We use the `nls()` with the Gauss-Newton algorithm and starting values $\beta_1 = 150$, $\beta_2 = 100$, and $\beta_3 = 0.6$.

```
full_expo_fit <- nls(bone ~ beta1 - beta2*exp(-beta3*age), trace = F, start =
                     list(beta1=150, beta2=100, beta3=.6), data = train_set)
```

```
summary(full_expo_fit)
```

```
##
## Formula: bone ~ beta1 - beta2 * exp(-beta3 * age)
##
## Parameters:
##       Estimate Std. Error t value Pr(>|t|)
## beta1 117.2347     3.5824   32.73  < 2e-16 ***
## beta2 121.3008    10.1270   11.98  1.5e-13 ***
## beta3   0.1201     0.0187    6.43  2.8e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.1 on 33 degrees of freedom
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 4.9e-06
```

From the output, the model coefficient estimates are $\hat{\beta}_1 = 117.2347$, $\hat{\beta}_2 = 121.3008$, and $\hat{\beta}_3 = 117.2347$. The mean square error (MSE) is $13.06^2 = 170.5636$ on 33 degrees of freedom.

Looking at the p-values, we can see that, at a reasonable significance level say 5%, all the coefficients are statistically significant.

3.2 Part (b): Hypothesis testing

We are interested in testing the hypothesis: $H_0 : \beta_1 = \beta_2$. Let $\beta_1 = \beta_2 = \beta$. Then the reduced model under H_0 is of the form

$$y = \beta\left(1 - e^{-\beta_3 x}\right) + \epsilon$$

```
reduced_mod <- nls(bone ~ beta*(1-exp(-beta3*age)), trace = F, start =
                     list(beta=100, beta3=.6), data = train_set)
summary(reduced_mod)
```

```
##
## Formula: bone ~ beta * (1 - exp(-beta3 * age))
##
## Parameters:
##       Estimate Std. Error t value Pr(>|t|)
## beta  117.5294     3.5257   33.34  < 2e-16 ***
## beta3   0.1158     0.0144    8.06  2.2e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.9 on 34 degrees of freedom
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 7.34e-07
```

From the above results, we observe that the estimate for the parameters are $\hat{\beta} = 117.5294$, and $\hat{\beta}_3 = 0.1158$. And once again, all the coefficients are statistically significant at the level of $\alpha = 0.05$ compared to the p—values.

3.3 Comparing the two models

```
anova(full_expo_fit, reduced_mod)
```

```
## Analysis of Variance Table
##
## Model 1: bone ~ beta1 - beta2 * exp(-beta3 * age)
## Model 2: bone ~ beta * (1 - exp(-beta3 * age))
##   Res.Df Res.Sum Sq Df Sum Sq F value Pr(>F)
## 1     33       5635
## 2     34       5662 -1  -27.5    0.16   0.69
```

At $\alpha = 0.05$ siginificance level compared to the p-value of 0.69, we fail to reject H_0, and conclude that there is no statistically significant difference between β_1 and β_2, which suggests that the reduced model is not statistically significantly different from the original model. We therefore conclude that the reduced model is better than the original model due to its simple form involving fewer model parameters.

For another level of model comparison, we use `AIC()` and `BIC()` functions in R with their default settings, and Table 2 shows the results for each model being considered. Both AIC and BIC values suggest that the reduced model is preferable to the full model. This is consistent with our previous result involving the test of hypothesis, therefore, we conclude that the reduced model is indeed better.

```
result_df <- data.frame(AIC=c(AIC(full_expo_fit), AIC(reduced_mod)),
           BIC = c(BIC(full_expo_fit), BIC(reduced_mod)))
rownames(result_df) <- c("Full exponential model", "Reduced exponential model")

kable(result_df, booktabs=T,linesep="",
      caption ="Model comparison based on AIC and BIC")%>%
    kable_styling(latex_options =c("HOLD_position"))%>%
    kable_classic()
```

Table 2: Model comparison based on AIC and BIC

	AIC	BIC
Full exponential model	292.1	298.4
Reduced exponential model	290.3	295.0

3.4 Part (c): Scatter plot with fitted curve from best nls model

3.4.1 Plotting functions

Two functions, `myscatterplot()` and `obs_pred_plot()`, were created to help with the generation of scatter plots with fitted curves from individual models and plots of the observed and predicted response values with reference line $y = x$, respectively. This was done to avoid unnecessary repetition of codes for pretty much the same routines.

```
myscatterplot <- function(model=NULL, fitted=NULL, df=train_set, fitted_col='blue',
                          title = '') {

    if(is.null(fitted)) fitted <- predict(model, newdata = df)
    if(title == '') title <- "Fitted curve together with the scatterplot of the data"
    # augment the data set with the fitted values from our selected model
    df$fitted_values <- fitted

    # recreate scatter plot with the fitted values
    ggplot(df, aes(age, bone)) +
        geom_point() +
        geom_line(aes(age, fitted_values), color=fitted_col) +
        labs(x="Age", y="Jaw bone length", title = title) +
        theme_bw()

}

obs_pred_plot <- function (pred, obs=NULL, col = "red", title='') {
    # obs: observed values
    # pred: predicted values

    if(is.null(obs)) obs <- test_set$bone

    if(title == '') title <- "Observed versus  predicted response values with reference line y=x"

    # PREDICTED VS. OBSERVED
    ggplot(mapping= aes(obs, pred)) +
        geom_point() +
        geom_abline(slope = 1, intercept = 0, color=col) +
        labs(y="Predicted jaw bone length", x="Observed jaw bone length",
             title = title)
}
```

Next, we create the required scatter plot with our custom function. This can be achieved by either specifying the trained model object or directly supplying the fitted values from the trained model, in which case we chose the former.

```
# create required scatter plot with our custom functions
myscatterplot(reduced_mod)
```

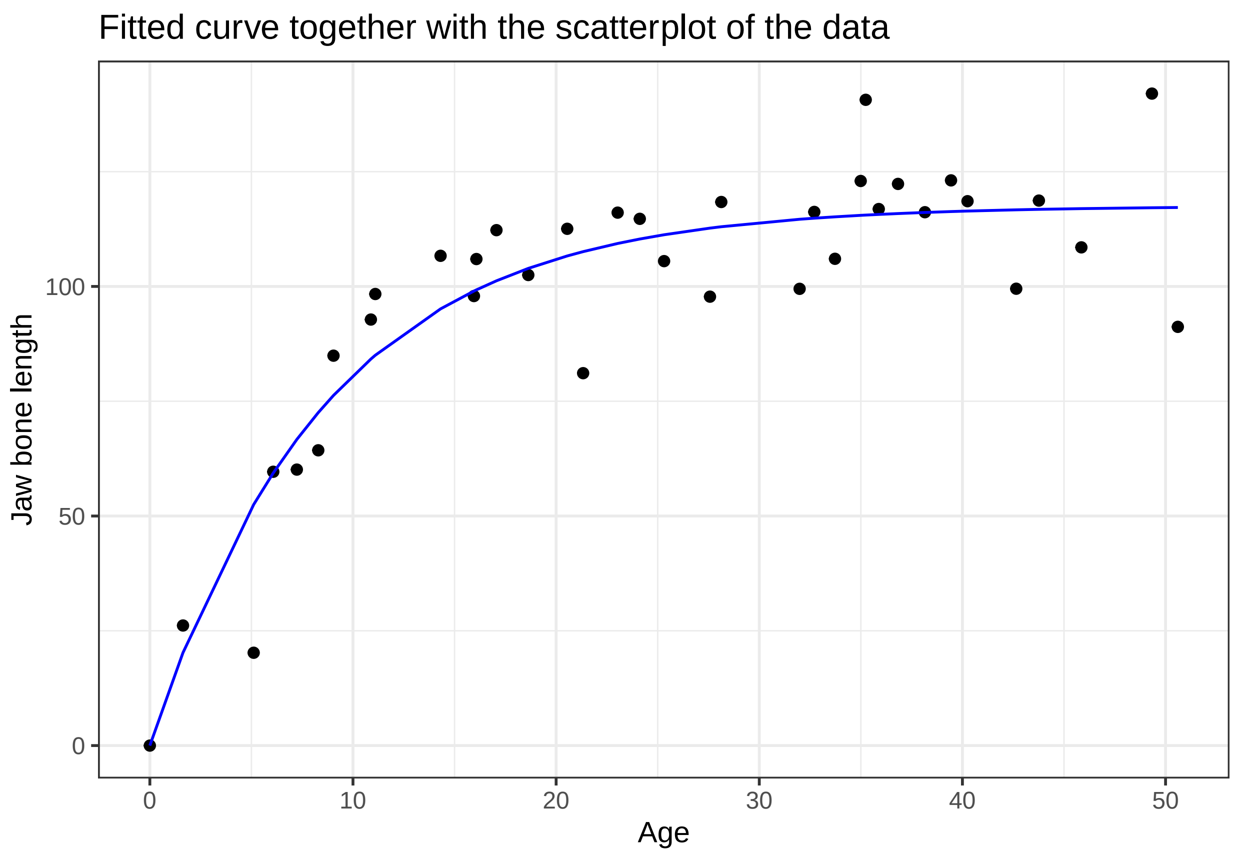


Figure 2: Relationship between jaw bone length and age in deer with the reduced nls fitted curve

Figure 2 shows that the reduced nonlinear spline model appear to fit the data well as it is able to capture the non-linear relationship we observed earlier.

3.5 Part (d): Apply selected model to the test data

Similarly, we first apply the selected model in part (b) to the test data and use our custom `obs_pred_plot()` function to generate a plot of the predicted response values against the observed response values as shown in Figure 3.

```r
# observed vs. predicted plot
nls_predicted <- predict(reduced_mod, test_set)
obs_pred_plot(pred = nls_predicted)
```

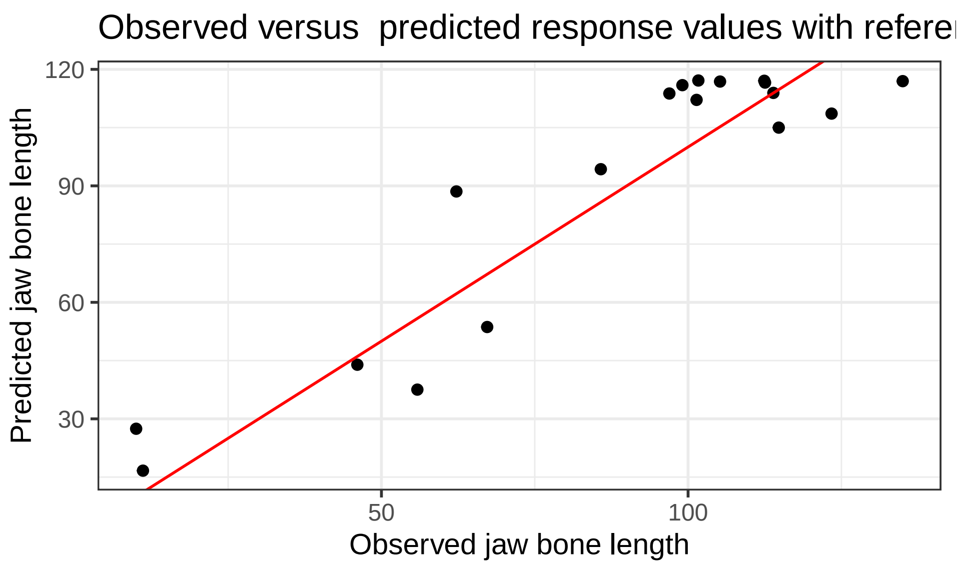


Figure 3: Observed versus predicted response

The red diagonal line represents the reference line $y = x$. Most of the data points deviate from this reference line, indicating that the predictions are not all that good. However, this seems a bit reasonable.

```r
# prediction mean square error
(reduced_nls_mse <- mean((test_set$bone - nls_predicted)^2))
```

```
## [1] 186.7
```

From the above output, the prediction mean square error (MSE) is **186.7497**.

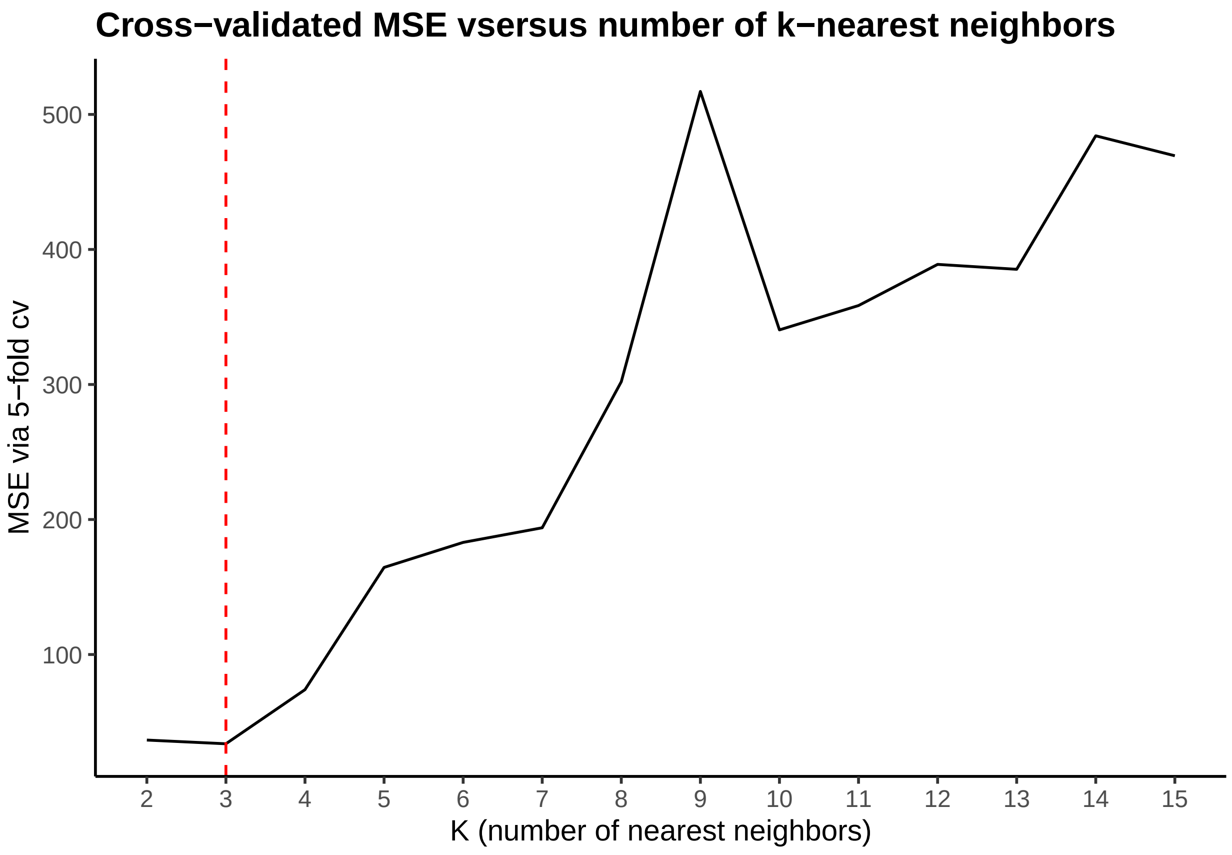
4 Local regression methods

4.1 Part (a): KNN Regression

Here, we implement a KNN regression model using the `knn.reg()` function in the `FNN` package. A 4-fold cross-validation was employed to obtain the optimal number of nearest neighbors, K from a set of integer values ranging from 2 to 15. The value corresponding to the minimum cross-validated mean square errors (MSE) was considered optimal.

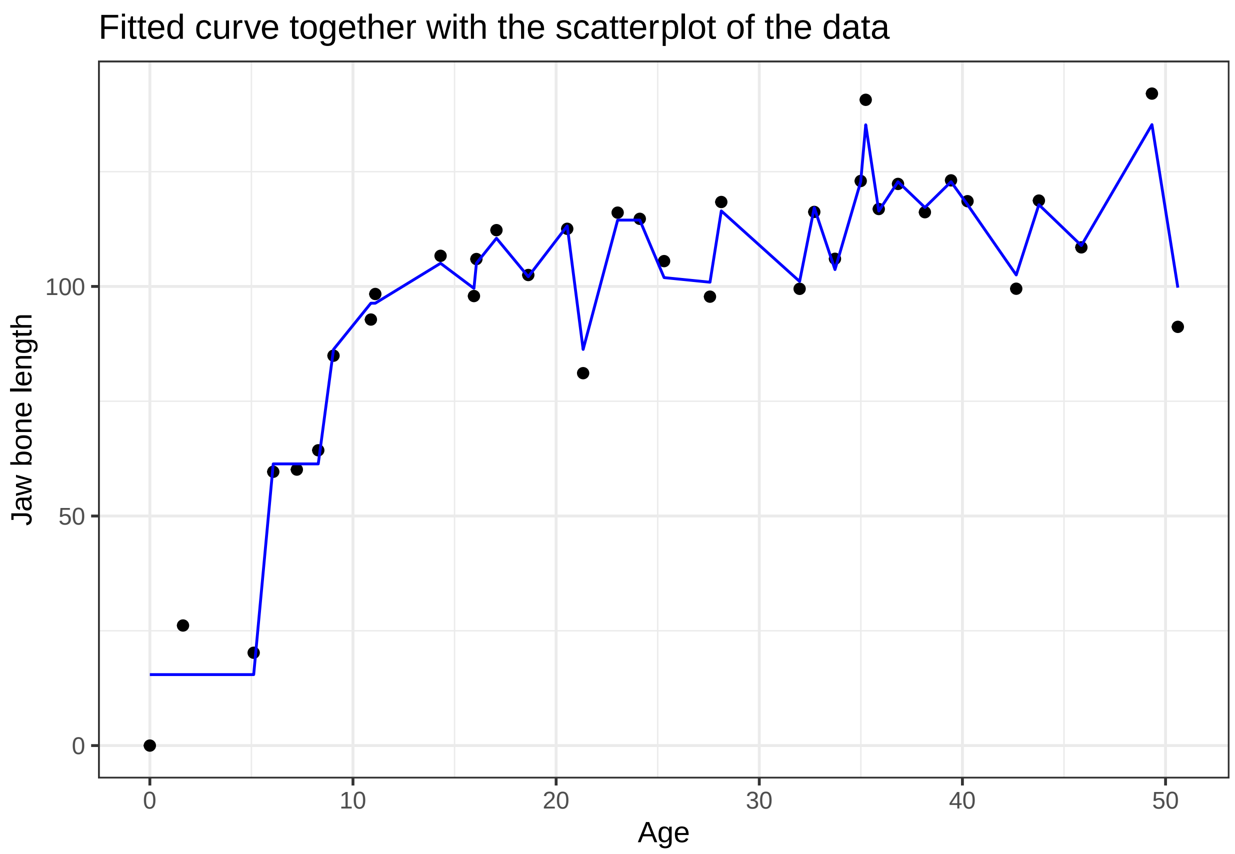
```r
# V-FOLD CV FOR SELECTING K
library(FNN)
set.seed(126)
V <- 5; n <- NROW(train_set)
id.fold <- sample(rep(1:V, each=(n%/%V)))
K <- 2:15; SSE <- rep(0, length(K))
for (k in seq_along(K)){
    id.fold <- sample(rep(1:V, each=(n%/%V)))
    for (v in 1:V){
        cv_train <- train_set[id.fold!=v,]
        cv_test <- train_set[id.fold==v,]
        yhat <- knn.reg(train=cv_train, test=cv_test, y=cv_train$bone,
                        k=k)$pred
        SSE[k] <- SSE[k] + sum((cv_test$bone-yhat)^2)
    }
}

cv_mse <- data.frame(K, MSE=SSE/n)
```

```r
# Get the optimal k
opt_k <- K[which.min(cv_mse$MSE)]

# Plot MSE versus k
ggplot(cv_mse) +
geom_line(aes(x=K, y=MSE)) +
geom_vline(xintercept=opt_k, color="red", linetype="dashed") +
scale_x_continuous(breaks = K) +
labs(x= "K (number of nearest neighbors)", y="MSE via 5-fold cv",
     title = "Cross-validated MSE vsersus number of k-nearest neighbors") +
theme_classic() +
theme(plot.title = element_text( face = "bold"))
```



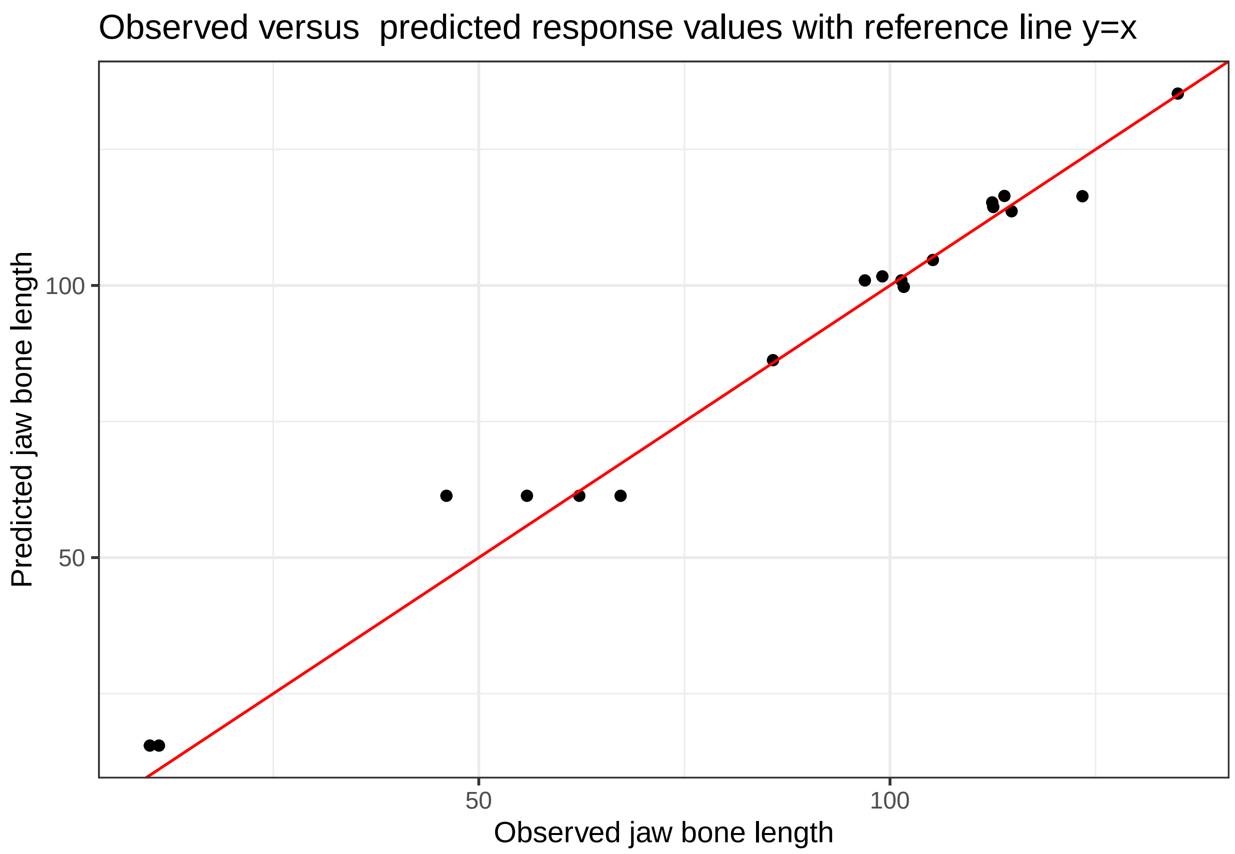
The red dashed line indicates that, with a 5-fold cross-validation, the optimal number of nearest neighbors occurs at $K = 3$, and this will be our choice of K for training a KNN regression model. 3 is quite low so this will results in a somewhat complex model.

```r
# fit knn with the optimal k
knn_fitted <- knn.reg(train_set,y=train_set$bone, test = train_set, k=opt_k,
                      algorithm = "kd_tree")$pred

# make predictions on test data
knn_predicted <- knn.reg(train_set,y=train_set$bone, test = test_set, k=opt_k,
                      algorithm = "kd_tree")$pred

# scatter plot of data with fitted
myscatterplot(fitted = knn_fitted)
```



The above figure shows a wiggly pattern indicating a potential overfitting problem by the KNN model with $K = 3$.

```
# plot of observed vs. predicted
obs_pred_plot(pred = knn_predicted)
```



The figure shows that the KNN regression model made reasonable predictions on the test data.

```
# prediction mean square error
(knn_mse <- mean((test_set$bone - knn_predicted)^2))
```

```
## [1] 24.63
```

4.2 Part (b): Kernel regression

We apply kernel regression to obtain a nonlinear fit via kernel regression smoothing with **local plug-in bandwidth** through the help of the `lokern` R package, where a local plugin bandwidth function is used to automatically obtain the bandwidths. The plugin method try to estimate the optimal bandwidths by estimating the asymptotically optimal mean squared error optimal bandwidths. Thus, using the default settings in `lokerns()`, a kernel regression was fitted to the training data.

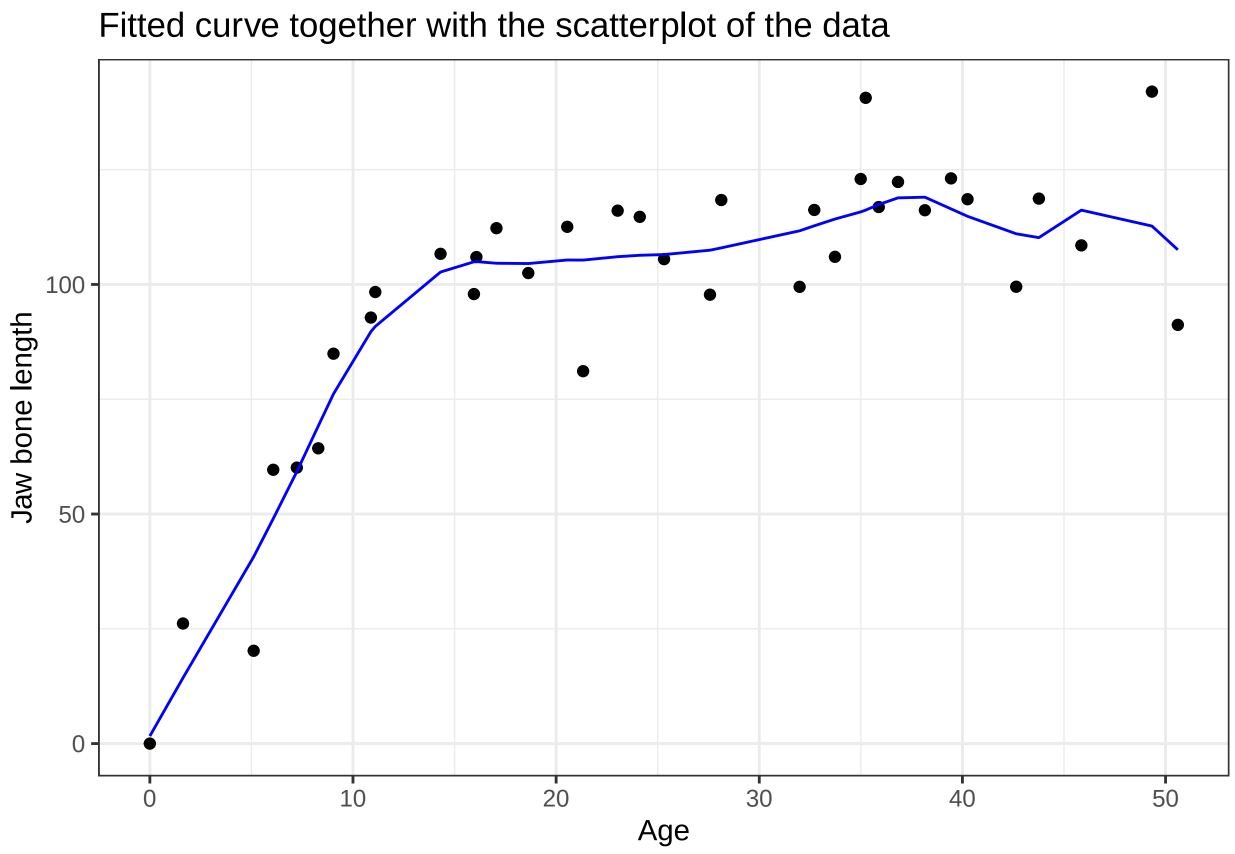
```
# Kernel regression smoothing with adaptive local plug-in bandwidth selection.
library(lokern)

kern_reg <- lokerns(train_set$age, train_set$bone)

# distribution of bandwidths
summary(kern_reg$bandwidth)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    3.45    5.06    5.47    5.52    6.32    8.20
```

```
# boxplot(kern_reg$bandwidth)
```

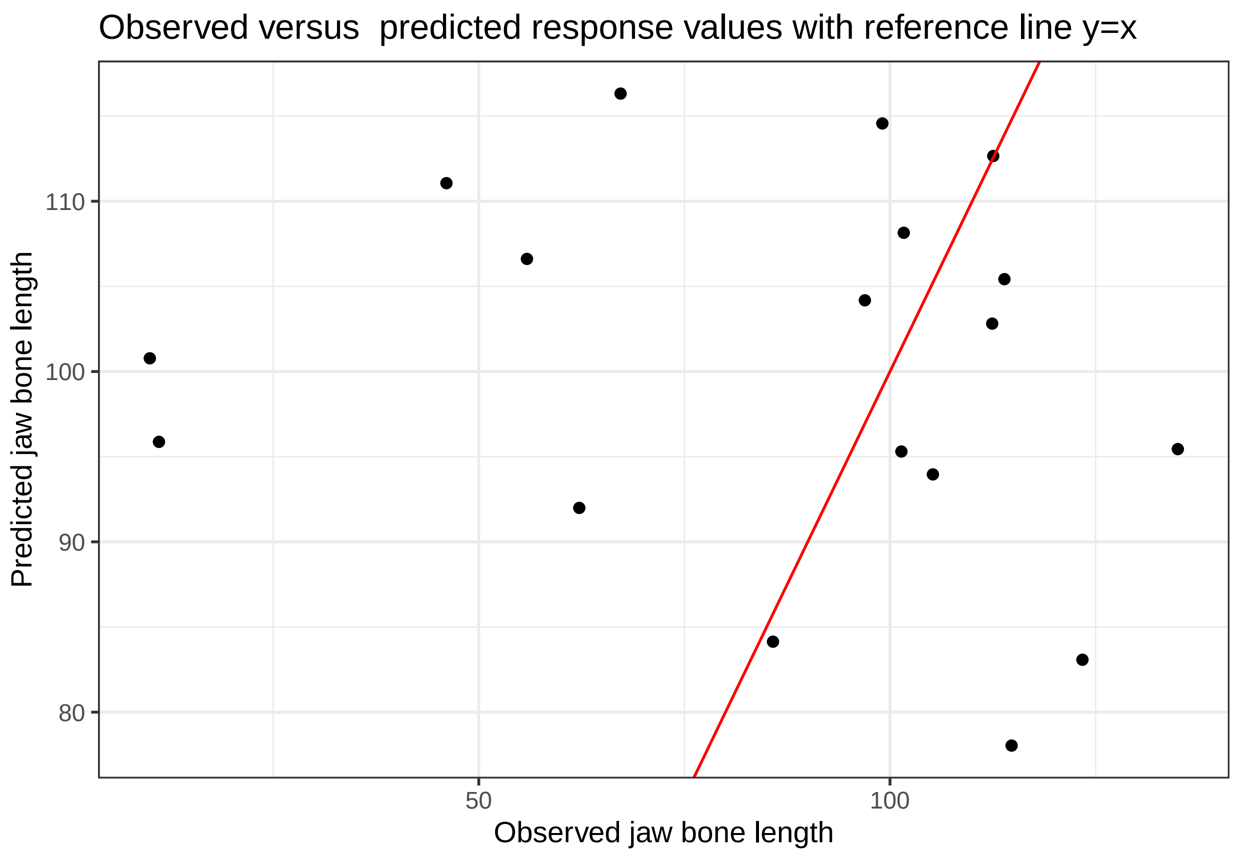
The above result shows the summary of the adaptive local plug-in bandwidth values used for the kernel regression estimation, from which we see that the optimal bandwidth for this problem ranges between 3.45 and 8.20 with a median value of 5.47.

```
# scatter plot with the fitted y values
myscatterplot(fitted = predict(kern_reg, newdata= train_set)$y)
```



By the above figure, we can see that the kernel regression model was able to capture the nonlinear relationship between jaw bone length and age in deer. However, it appear to have learned too much from the training data to be able to make good predictions on an unseen data.

```
kern_reg_predicted <- predict(kern_reg, newdata=test_set)$y

# plot of observed vs. predicted
obs_pred_plot(kern_reg_predicted)
```



The points are randomly scattered about, with most deviating from the reference line. This shows how unreasonable the predictions made by the the kernel regression model are.

```
(kern_reg_mse <- mean((test_set$bone - kern_reg_predicted)^2))
```

```
## [1] 1707
```

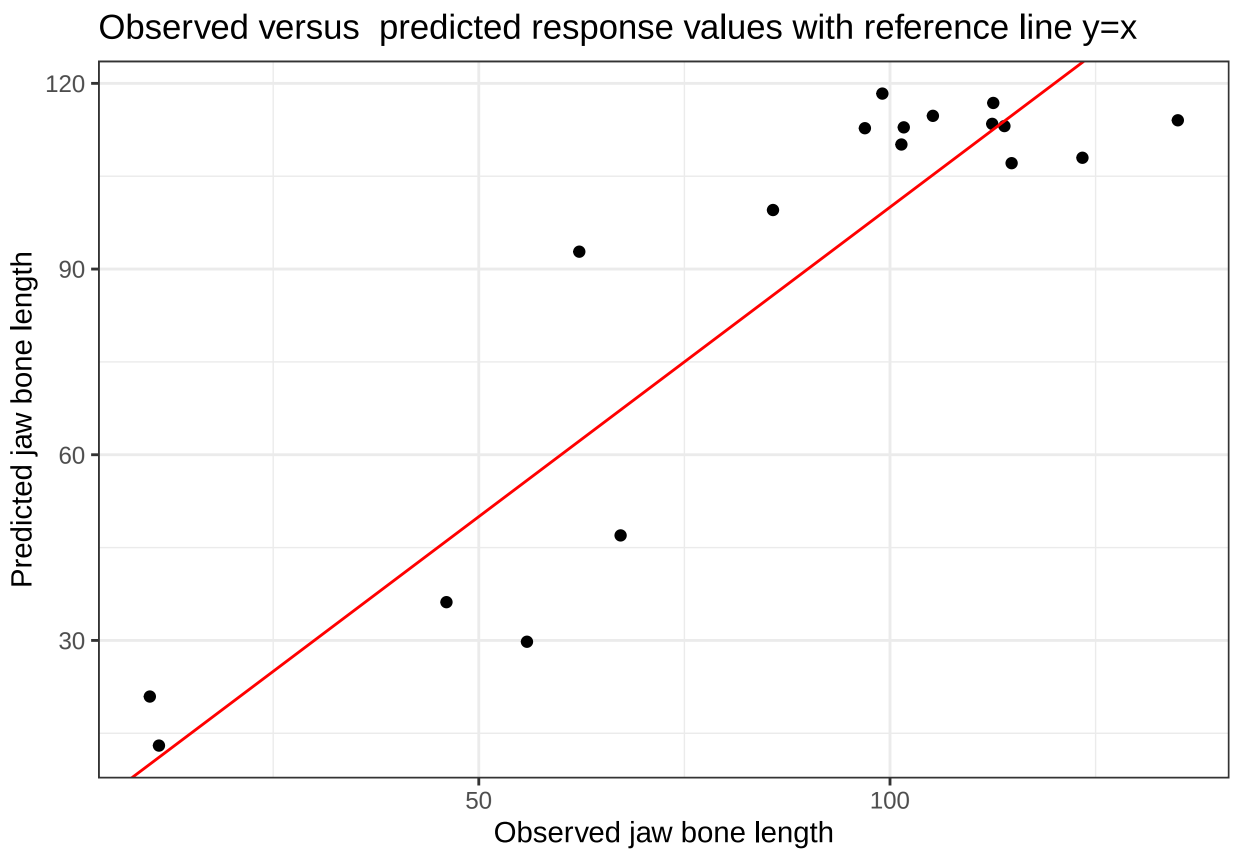
We have here a very large prediction MSE of 1707.0242, confirming our previous observation that the predictions do not seem reasonable.

4.3 Part (c): Local (cubic) polynomial regression

Using the `locpol()` function with a **gaussian** kernel, we applied a local (cubic) polynomial regression to the training data. For the choice of bandwidth the `bw.nrd()` function available in the `stats` package was applied on the age predictor, yielding a value of **7.429**.

```
library(locpol)

# fit a local cubic polynomial:
h <- bw.nrd(train_set$age) # get best bandwidth
local_poly_fit <- locpol(bone ~ age, data = train_set, deg = 3, bw=h, kernel = gaussK)
```

```
# observed vs. predicted on test set
lopredicted <- locpol(bone ~ age, data = train_set, xeval = test_set$age,
                      deg = 3, bw=h, kernel = gaussK)

obs_pred_plot(lopredicted$lpFit$bone)
```



The above figure shows that the local polynomial regression model could not predict most of the observations in the test set correctly, but the predictions do not appear too unreasonable.

```
(lomse <- mean((test_set$bone - lopredicted$lpFit$bone)^2))
```

```
## [1] 228.2
```

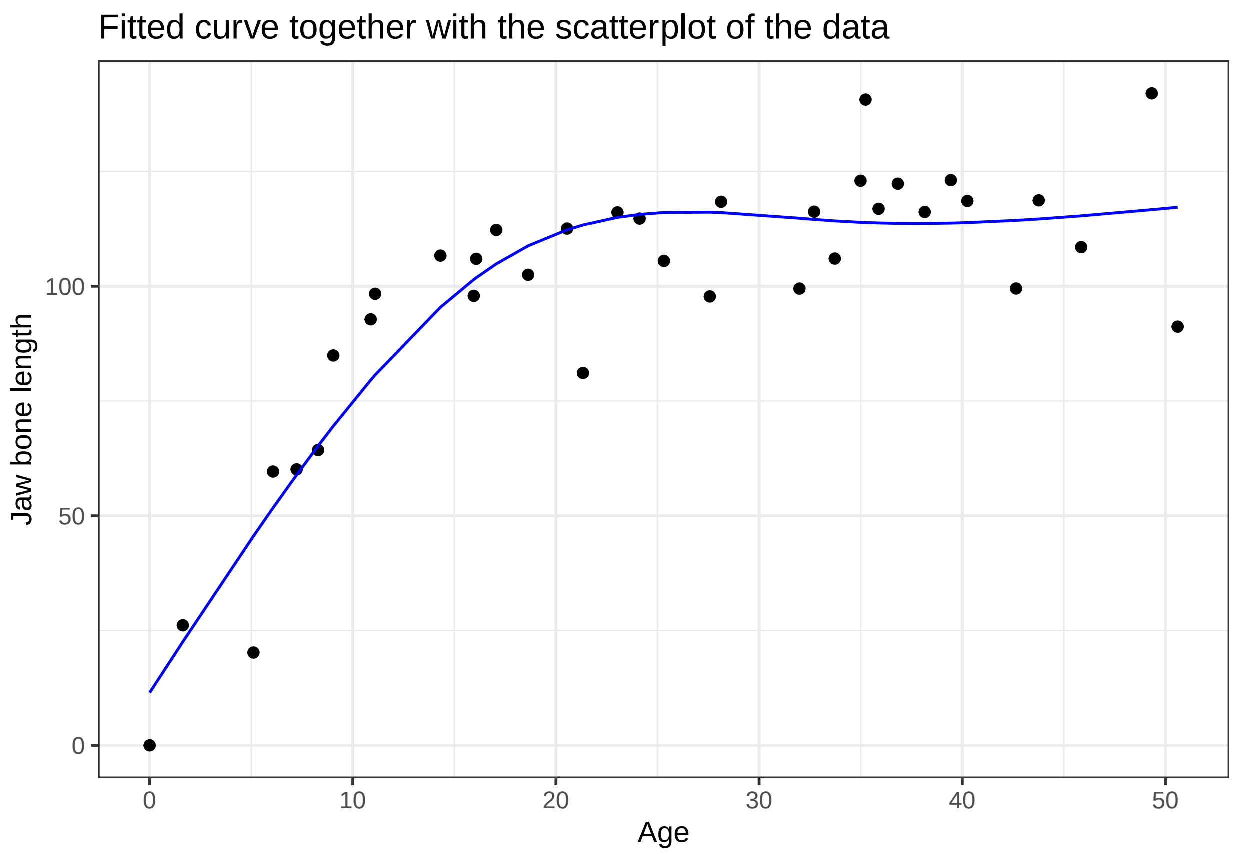
The prediction MSE is obtained as `rlomse`.

5 Regression/smoothing splines

5.1 Part (a): Regression spline (natural cubic spline)

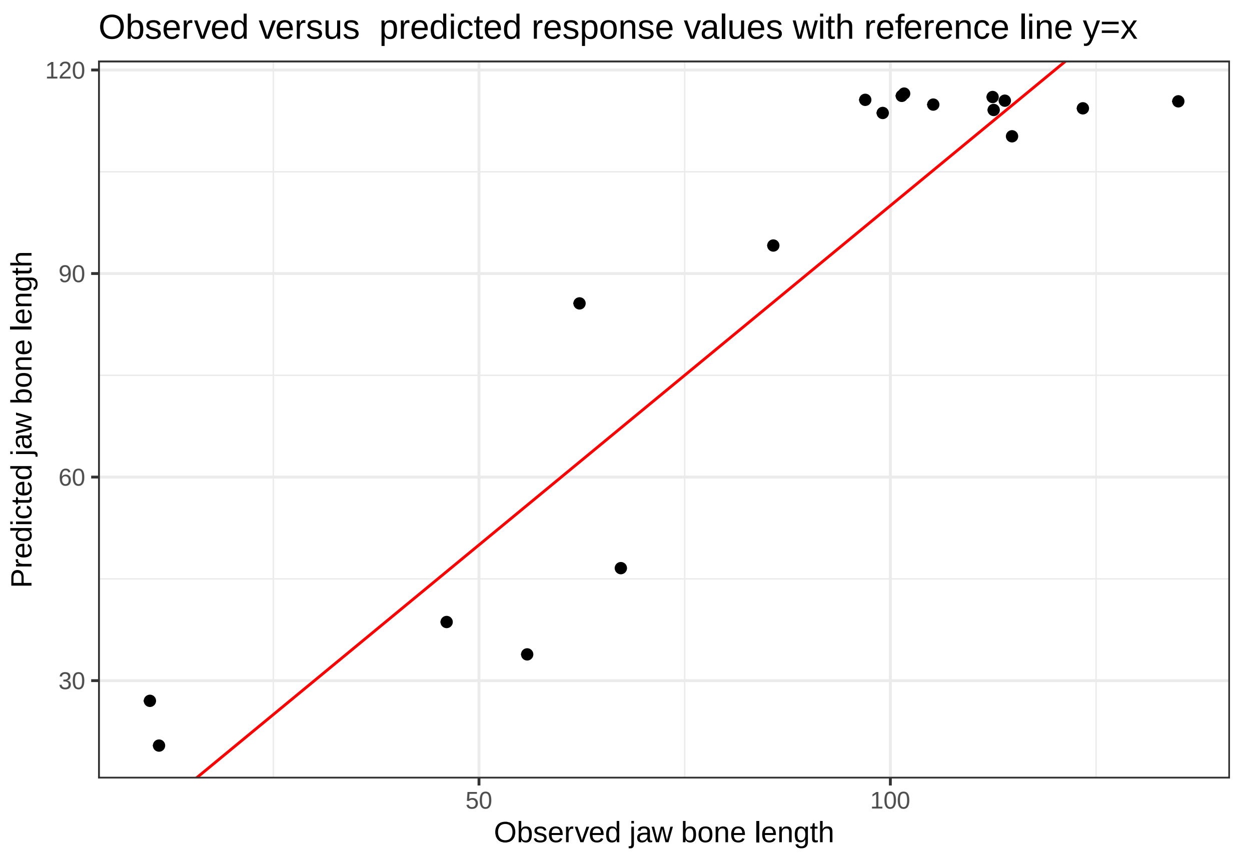
We used `ns()` function to generate the B-spline basis matrix for a natural cubic spline with 3 degrees of freedom resulting in a suitably chosen quantiles of the age variable as knots. No intercept was included in the basis functions.

```
library(splines)
cubic_spline_fit <- lm(bone ~ ns(age, df=3), data = train_set)

myscatterplot(cubic_spline_fit)
```



The above figure suggests a good fit as the fitted curve seem to capture the nonlinear relationship reasonably well.

```r
c_spline_predicted <- predict(cubic_spline_fit, newdata = test_set)

obs_pred_plot(c_spline_predicted)
```



Given the above plot, it can be said that the regression spline model was not able to most of the observations in the test set correctly since a good number of the data points are far from the reference line. However, the predictions do not appear extremely unreasonable.

```
(c_spline_mse <- mean((test_set$bone - c_spline_predicted)^2))
```

```
## [1] 197.7
```

For this model, the prediction MSE is 197.672.

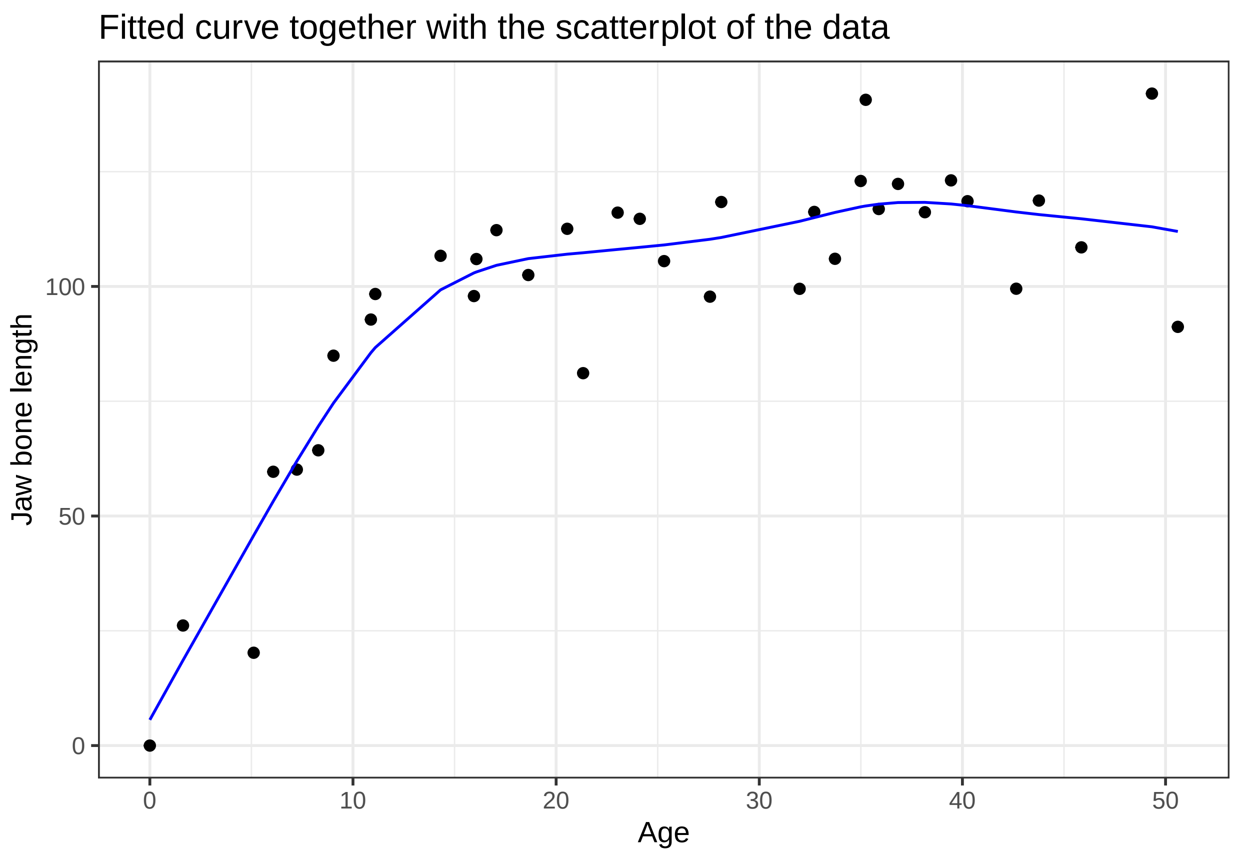
5.2 Part (b): Smoothing Splines

By setting the `cv` argument of the `smooth.spline()` function to **false**, the generalized cross-validation (GCV) procedure was used to determine the optimal *smoothing parameter* at $ spar =0.7175$ and a tuning parameter $df = 5.865$.

```
ss_fit <- smooth.spline(train_set$age, train_set$bone, cv=F)

# checking the optimal tuning parameter
ss_fit
```

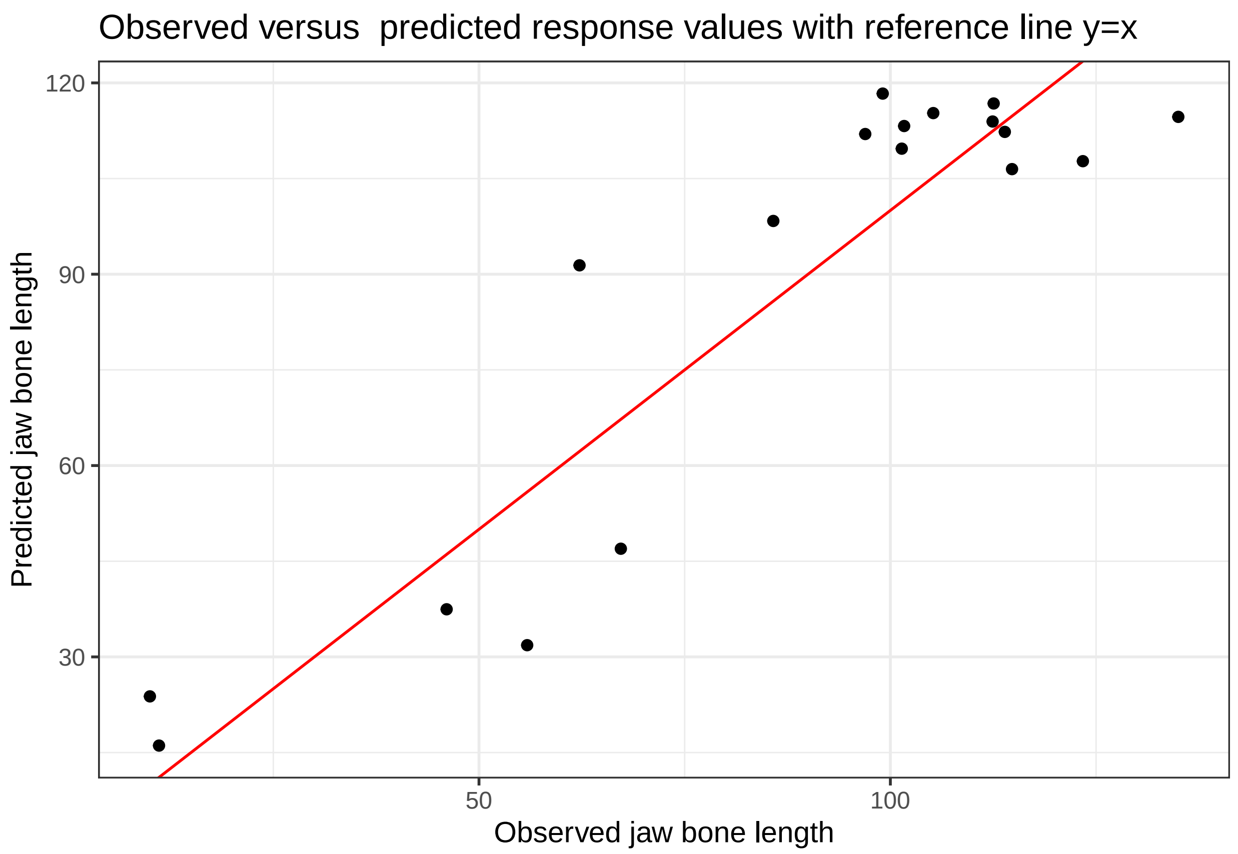
```
## Call:
## smooth.spline(x = train_set$age, y = train_set$bone, cv = F)
##
## Smoothing Parameter  spar= 0.7175  lambda= 0.001073 (14 iterations)
## Equivalent Degrees of Freedom (Df): 5.865
## Penalized Criterion (RSS): 4854
## GCV: 2560
```

```
# scatter plot of data with fitted curve
myscatterplot(fitted = predict(ss_fit, train_set$age)$y)
```



It is clear from the above figure that the smoothing splines appear to fit the data well.

```
ss_predicted <- predict(ss_fit, test_set$age)

obs_pred_plot(ss_predicted$y)
```



Although a good number of the points deviate from the reference line, they are not extremely far away to suggest unreasonable predictions. This result is comparable to the one in part (a).

```
(ss_mse <- mean((test_set$bone - ss_predicted$y)^2))
```

```
## [1] 218.8
```

The prediction MSE is `rss_mse`.

6 Prediction MSE measures

```
data.frame(Model = c("Exponential", "KNN regressiion", "Kernel regression",
                     "Local cubic polynomial regression", "Regression splines",
                     "Smoothing splines"),
           `Prediction MSE` = c(reduced_nls_mse,knn_mse, kern_reg_mse, lomse,
                                c_spline_mse, ss_mse)) %>%
    kable(booktabs=T, linesep="") %>%
    kable_styling(latex_options = c("HOLD_position")) %>%
    kable_classic()
```

Model	Prediction.MSE
Exponential	186.75
KNN regressiion	24.63
Kernel regression	1707.02
Local cubic polynomial regression	228.20
Regression splines	197.67
Smoothing splines	218.76

From the above table of results, it is clear that KNN gives the best result for the given data since its prediction MSE is the lowest.