APA_Assignment_2

AUTHOR
Sam Ryder, 18317496

Library Installation

```r
#Install libraries
suppressWarnings(library(COMPoissonReg))
```

```
Loading required package: Rcpp

Loading required package: numDeriv
```

```r
suppressWarnings(library(AER))
```

```
Loading required package: car

Loading required package: carData

Loading required package: lmtest

Loading required package: zoo


Attaching package: 'zoo'

The following objects are masked from 'package:base':

    as.Date, as.Date.numeric

Loading required package: sandwich

Loading required package: survival
```

```r
suppressWarnings(library(MASS))
suppressWarnings(library("xlsx"))
suppressWarnings(library("nlme"))
suppressWarnings(library(lme4))
```

```
Loading required package: Matrix


Attaching package: 'lme4'

The following object is masked from 'package:nlme':

    lmList
```

```r
suppressWarnings(library(splines))
```

Question 1

Part A: Poisson GLM

```
#Data
data(couple)

#Fit Poisson GLM
fit.pois <- glm(UPB ~ EDUCATION + ANXIETY, data = couple, family=poisson)
summary(fit.pois)
```

```
Call:
glm(formula = UPB ~ EDUCATION + ANXIETY, family = poisson, data = couple)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.81695    0.04386  18.628   <2e-16 ***
EDUCATION   -0.21579    0.07047  -3.062   0.0022 **
ANXIETY      0.42169    0.03333  12.651   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2478.3  on 386  degrees of freedom
Residual deviance: 2310.8  on 384  degrees of freedom
AIC: 2782.4

Number of Fisher Scoring iterations: 6
```

Both variables significant.

Education to a significance level of 0.01 and Anxiety to a significance level of 0.001.

Coefficient Estimates show that those with a bachelor’s degree are less likely to have high UPB.

Those with Anxiety are more likely to have high UPB.

Part B: Overdispersion Test

```
#Overdispersion test
dispersiontest(fit.pois, trafo = function(x) x^2)
```

```
    Overdispersion test

data:  fit.pois
z = 3.0033, p-value = 0.001335
alternative hypothesis: true alpha is greater than 0
sample estimates:
   alpha
3.186317
```

Reject h0 as alpha (0.001335) < 0.05.

This means that data is overdispersed.

To deal with overdispersion in poisson model one could use ZIP (Zero Inflated Poisson) model. ZIP model allows for modelling where large proportion of values of variable are 0 as is the case in this dataset.

Part C: Negative binomial regression model on overdispersed data

The data has been shown to be overdispersed.

This means the use of a negative binomial regression model is suitable.

```
fit.nb <- glm.nb(UPB ~ EDUCATION + ANXIETY, data = couple)
summary(fit.nb)
```

```

Call:
glm.nb(formula = UPB ~ EDUCATION + ANXIETY, data = couple, init.theta = 0.1937364127,
    link = log)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.8553     0.1549   5.521 3.36e-08 ***
EDUCATION    -0.3532     0.2498  -1.414    0.157
ANXIETY       0.4856     0.1217   3.991 6.58e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(0.1937) family taken to be 1)

    Null deviance: 303.28  on 386  degrees of freedom
Residual deviance: 288.22  on 384  degrees of freedom
AIC: 1285.9

Number of Fisher Scoring iterations: 1


              Theta:  0.1937
          Std. Err.:  0.0220

 2 x log-likelihood:  -1277.9190
```

```
SE<-coef(summary(fit.nb))[,2] # standard errors
inf<-fit.nb$coef-qnorm(1-0.05/2)*SE # inferior bound
sup<-fit.nb$coef+qnorm(1-0.05/2)*SE # superior bound
round(inf,5)
```

```
(Intercept)   EDUCATION     ANXIETY
    0.55172    -0.84270     0.24713
```

```
round(sup,5)
```

```
(Intercept)   EDUCATION     ANXIETY 
    1.15897     0.13633     0.72413 
```

Covariate education is non-significant as confidence interval contains the value zero.

This represents a change from the Poisson modelling as Education was significant to a 0.01 level.

Part D: Repeated Poisson model runs

Set up data.

```
x <- couple[, c('EDUCATION', 'ANXIETY')]
y <- couple$UPB
#Split data
set.seed(123)
n <- nrow(x)
train_indices <- sample(1:n, n * 0.8)  # Randomly sample 80% of the indices
x_train <- x[train_indices, ]  # Subset the data using the sampled indices for traini
x_test <- x[-train_indices, ]  # Subset the remaining data for testing
y_train <- y[train_indices]  # Subset the corresponding labels for training
y_test <- y[-train_indices]  # Subset the corresponding labels for testing
```

Train and predict using Poisson model and negative binomial model

```
#Training
#Poisson model
pois.mod <- glm(y_train ~ ., data = x_train, family=poisson)
#Negative Binomial model
fit.nb <- glm.nb(y_train ~ ., data = x_train)

#Predictions
#Poisson model
poisson_pred <- predict(pois.mod, newdata = x_test, type = "response")
#Negative Binomial model
nb_pred <- predict(fit.nb, newdata = x_test, type = "response")
```

Compute the MSE

```
#Compute MSE
#Poisson model
mse_poisson <- mean((y_test-poisson_pred)**2)
#Negative Binomial model
mse_binomial <- mean((y_test-nb_pred)**2)

# Print the results
cat("Mean Squared Prediction Error (MSPE) for Poisson Model:", mse_poisson, "\n")
```

```
Mean Squared Prediction Error (MSPE) for Poisson Model: 27.75891
```

```
cat("Mean Squared Prediction Error (MSPE) for Negative Binomial Model:", mse_binomial
```

```
Mean Squared Prediction Error (MSPE) for Negative Binomial Model: 27.57095
```

Repeat the process 100 times

```
#Repeat 100 times
#Result vectors
mse_poisson_vec <- numeric(100)
mse_nb_vec <- numeric(100)


# Repeat the process 100 times
for (i in 1:100) {
  # Randomly split the data into training and test sets
  train_indices <- sample(1:nrow(x), nrow(x) * 0.5)  # 50% for training
  train <- x[train_indices, ]
  test <- x[-train_indices, ]
  y_train <- y[train_indices]
  y_test <- y[-train_indices]

  # Fit Poisson model to the training set
  pois.mod <- glm(y_train ~ ., data = train, family = poisson)

  # Fit Negative Binomial model to the training set
  fit.nb <- glm.nb(y_train ~ ., data = train)

  # Predictions for Poisson model on the test set
  poisson_pred <- predict(pois.mod, newdata = test, type = "response")

  # Predictions for Negative Binomial model on the test set
  nb_pred <- predict(fit.nb, newdata = test, type = "response")

  # Compute MSE for Poisson model
  mse_poisson <- mean((y_test - poisson_pred)^2)

  # Compute MSE for Negative Binomial model
  mse_nb <- mean((y_test - nb_pred)^2)

  # Store MSE values in vectors
  mse_poisson_vec[i] <- mse_poisson
  mse_nb_vec[i] <- mse_nb
}

# Compute the mean of MSE values for both models
mean_mse_poisson <- mean(mse_poisson_vec)
mean_mse_nb <- mean(mse_nb_vec)

# Print the results
cat("Mean Squared Prediction Error (MSPE) for Poisson Model:", mean_mse_poisson, "\n"
Mean Squared Prediction Error (MSPE) for Poisson Model: 23.25776
```

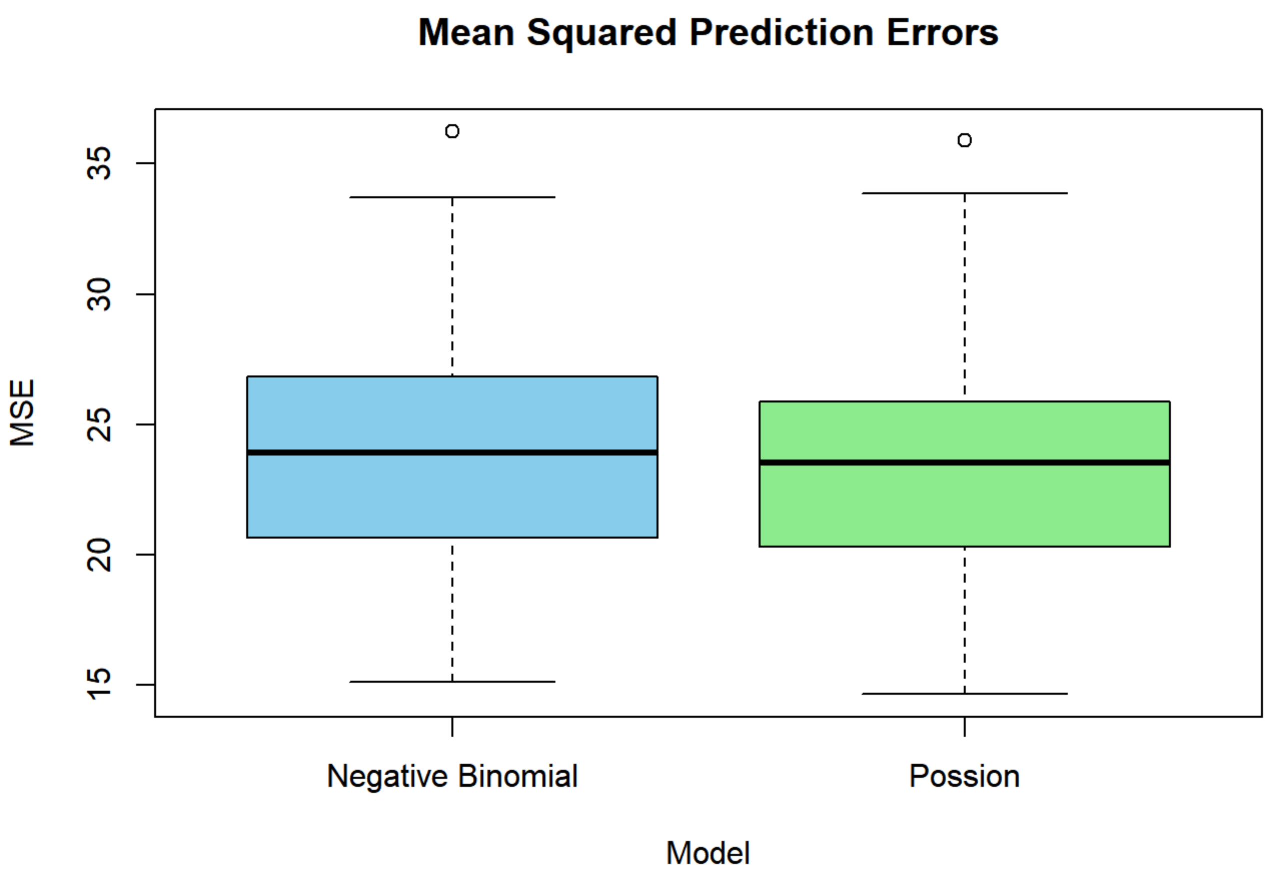
```
cat("Mean Squared Prediction Error (MSPE) for Negative Binomial Model:", mean_mse_nb,
```

```
Mean Squared Prediction Error (MSPE) for Negative Binomial Model: 23.77415
```

Collect results into dataframe and plot Mean Squared Error.

```
#Create a results dataframe
mse_data <- data.frame(
  Model_name = rep(c('Possion', 'Negative Binomial'), each=100),
  MSE = c(mse_poisson_vec, mse_nb_vec)
)

# Create boxplot
boxplot(MSE ~ Model_name, data = mse_data,
        main = "Mean Squared Prediction Errors",
        xlab = "Model", ylab = "MSE",
        col = c("skyblue", "lightgreen"))
```



Boxplot shows similar performance of models across 100 iterations.

The two models have similar inter-qartile ranges and ranges with both having a positive outlier of over 35.

The Poisson model seems to have a marginally lower average mean squared error and so is the preferred model when it comes to prediction.

Question 2

```
#Data set up
data<-read.xlsx(file='HSAB.xlsx', sheetIndex=1, header=TRUE)
math.achieve<-data$math.achieve
school<-as.factor(data$school)
sampled_schools <- sample(unique(school), 5)
data_sampled <- subset(data, school %in% sampled_schools)
```

Part A: Normal Regression model

```
#normal regression model
fit <- lm(math.achieve ~ school, data=data_sampled)
summary(fit)
```

```
Call:
lm(formula = math.achieve ~ school, data = data_sampled)

Residuals:
     Min       1Q   Median       3Q      Max 
-18.1283  -4.3748   0.6267   5.0338  11.0920 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) 16.1440173  1.0404832  15.516   <2e-16 ***
school      -0.0003059  0.0002034  -1.504    0.134    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.419 on 254 degrees of freedom
Multiple R-squared:  0.008826,	Adjusted R-squared:  0.004924 
F-statistic: 2.262 on 1 and 254 DF,  p-value: 0.1338
```

School is not a significant covariate.

Part B: Fixed Effects model

```
#Fixed effects model
fit2 <- lmer(math.achieve~1+(1|school))
summary(fit2)
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: math.achieve ~ 1 + (1 | school)

REML criterion at convergence: 47116.8
```

```
Scaled residuals:
    Min      1Q  Median      3Q     Max
-3.0631 -0.7539  0.0267  0.7606  2.7426

Random effects:
 Groups   Name        Variance Std.Dev.
 school   (Intercept)  8.614   2.935
 Residual             39.148   6.257
Number of obs: 7185, groups:  school, 160

Fixed effects:
            Estimate Std. Error t value
(Intercept)  12.6370     0.2444   51.71
```

Random effect for school indicates variance of 8.614 and SD of 2.935.

This shows there is significant variability between schools.

While school was not significant as a fixed factor in part a), it has become significant in the random effects model.

Part C: Intraclass coefficient and Parameter CIs

```
#Intraclass correlation coefficient
#p^ = intercept variance / (intercept variance + residual_variance)
phat = 8.614/(8.614 + 39.148)
phat
```

```
[1] 0.1803526
```

0.18 is total variance in outcome that can be attributed to differences between schools.

Nearly one fifth of math achievement is down to the school they go to.

```
#Confidence intervals for parameters
confint(fit2, method="boot")
```

```
Computing bootstrap confidence intervals ...

                2.5 %    97.5 %
.sig01       2.570205  3.309252
.sigma       6.147864  6.364829
(Intercept) 12.136551 13.144458
```

95% confidence interval for SD of random intercepts: [2.579, 3.268].

95% confidence interval for SD of residual: [6.147, 6.359].

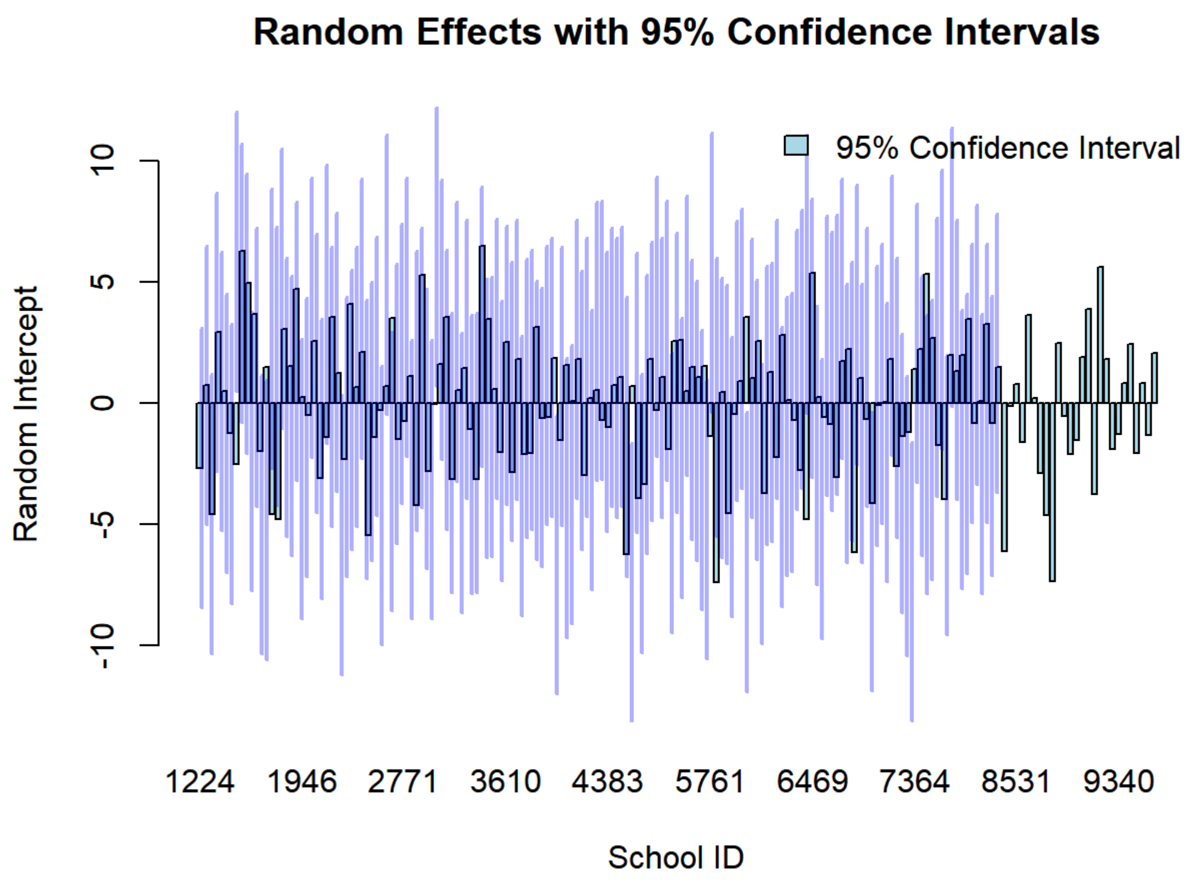
Part D: Predict random effects

```
#Predict the random effects
random_effects <- ranef(fit2)
# Extracting school IDs
```

```
school_ids <- as.numeric(rownames(random_effects[[1]]))
# Extracting random intercepts
intercepts <- random_effects[[1]][, "(Intercept)"]
# Computing lower and upper bounds of confidence intervals
ci_lower <- intercepts - 1.96 * sqrt(VarCorr(fit2)$school[1, 1])
ci_upper <- intercepts + 1.96 * sqrt(VarCorr(fit2)$school[1, 1])
# Creating a dataframe to present the results
random_effects_df <- data.frame(School_ID = school_ids,
                                Random_Intercept = intercepts,
                                Lower_CI = ci_lower,
                                Upper_CI = ci_upper)
# Printing the results
print(head(random_effects_df, 3))
```

```
  School_ID Random_Intercept   Lower_CI Upper_CI
1      1224       -2.6639348  -8.416468 3.088598
2      1288        0.7394098  -5.013123 6.491943
3      1296       -4.5684656 -10.320999 1.184068
```

```
# Plotting
par(mar = c(5, 4, 4, 4))
barplot(random_effects_df$Random_Intercept,
        ylim = range(c(random_effects_df$Lower_CI, random_effects_df$Upper_CI)),
        names.arg = random_effects_df$School_ID,
        ylab = "Random Intercept", xlab = "School ID",
        col = "lightblue",
        main = "Random Effects with 95% Confidence Intervals")
segments(x0 = 1:nrow(random_effects_df),
         y0 = random_effects_df$Lower_CI,
         x1 = 1:nrow(random_effects_df),
         y1 = random_effects_df$Upper_CI,
         lwd = 2, col = rgb(0, 0, 1, alpha = 0.3))
legend("topright", legend = "95% Confidence Interval",
       fill = "lightblue", bty = "n")
```



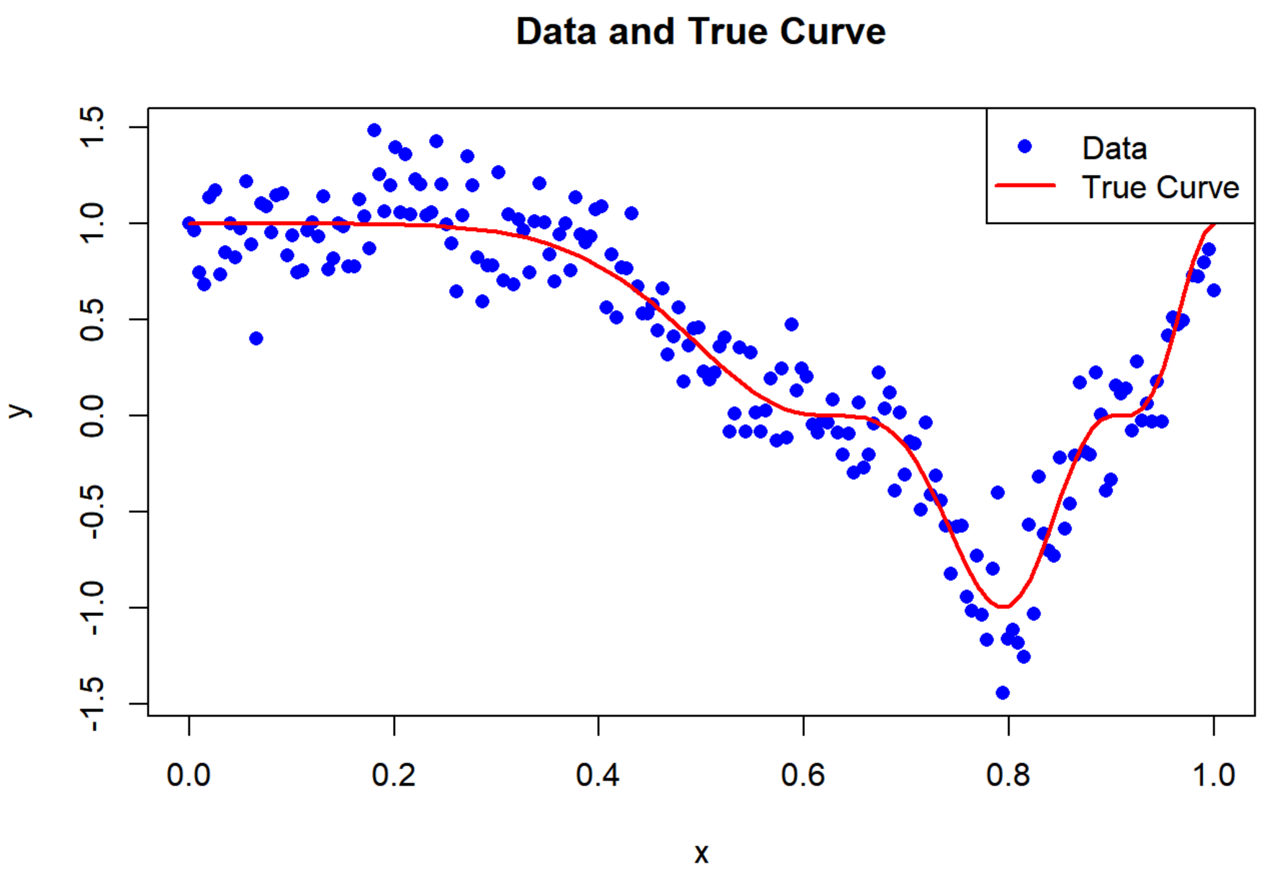
Question 3

Part A: Generate data sample and plot curve

```
# Define the  function f(x)
f <- function(x) {
  return((cos(2 * pi * x^3))^3)
}

# 200 samples
n <- 200
# Equally spaced over the interval [0, 1]
x <- seq(0, 1, length.out = n)
# Variance of the errors
sigma_sq <- 0.04
# Generate random errors
epsilon <- rnorm(n, mean = 0, sd = sqrt(sigma_sq))
# Generate the observed y values, from given formula
y <- f(x) + epsilon
# Plot the data
plot(x, y, col = "blue", pch = 16, xlab = "x", ylab = "y", main = "Data and True Curv
#Add true curve, f
curve(f, add = TRUE, col = "red", lwd = 2)
```

```
#Legend
legend("topright", legend = c("Data", "True Curve"), col = c("blue", "red"), pch = c(
```



Part B: Fit kernel smoothing spline

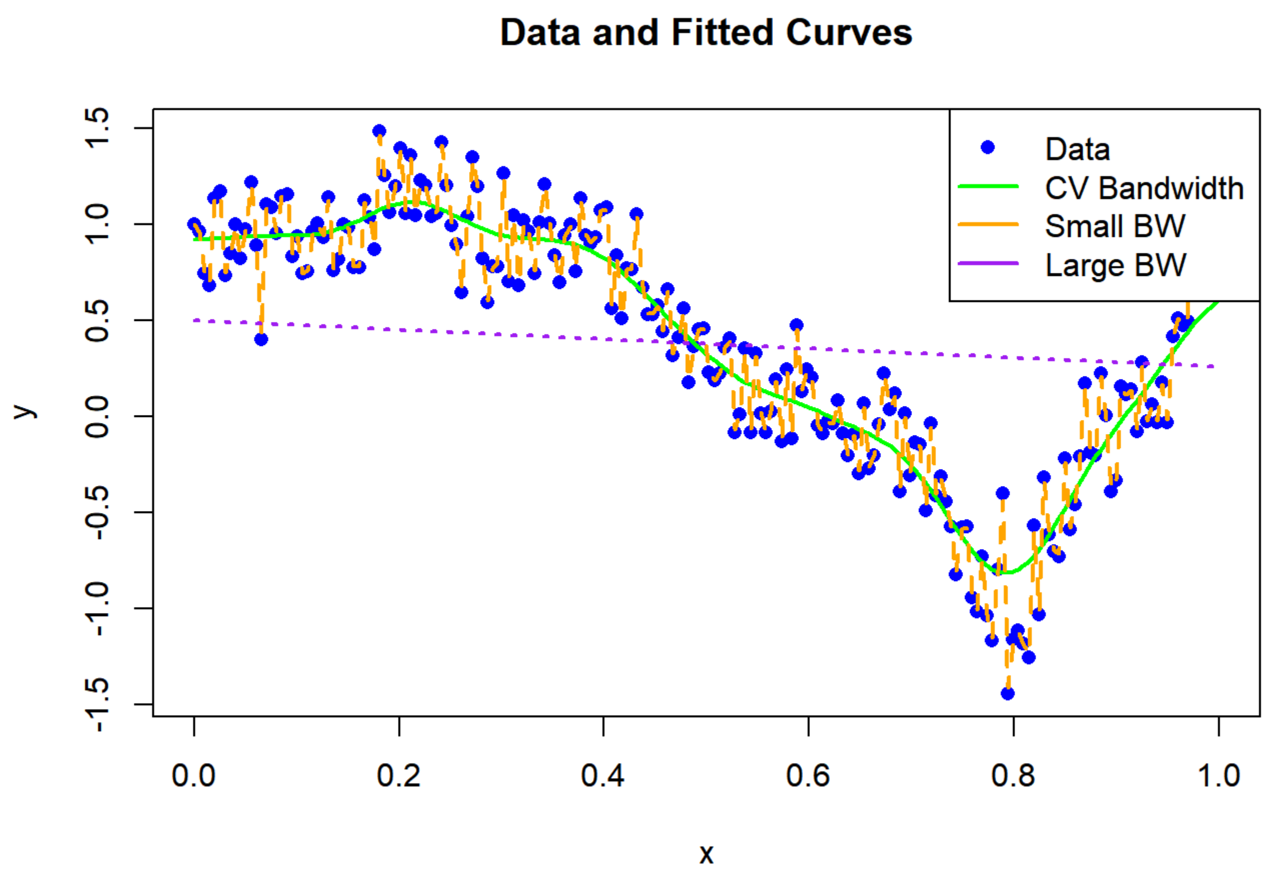
```
#kernel smoothing
LOOCV_error <- function(x, y, bandwidth) {
  n <- length(x)
  cv_error <- 0
  for (i in 1:n) {
    x_i <- x[-i]
    y_i <- y[-i]
    f_hat_i <- ksmooth(x_i, y_i, kernel = "normal", bandwidth = bandwidth)$y[i]
    cv_error <- cv_error + (y[i] - f_hat_i)^2
  }
  return(cv_error / n)
}

# Define a sequence of bandwidth values to evaluate
bw_values <- seq(0.01, 10, length.out = 100)
# Compute LOOCV error for each bandwidth value
cv_errors <- sapply(bw_values, function(bw) LOOCV_error(x, y, bw))
# Find the bandwidth that minimizes the LOOCV error
optimal_bw <- bw_values[which.min(cv_errors)]
# Check if there are NA values in the fitted values
```

```
if (any(is.na(ksmooth(x, y, kernel = "normal", bandwidth = optimal_bw)$y))) {
  # If NA values are present, the bandwidth might be too small. Adjust it.
  optimal_bw <- 0.1  # Or choose another reasonable value
}
# Fit a curve using kernel smoothing with the optimal bandwidth
fit_cv <- ksmooth(x, y, kernel = "normal", bandwidth = optimal_bw)

# Fit curves with small and large bandwidths for comparison
fit_small_bw <- ksmooth(x, y, kernel = "normal", bandwidth = 0.005)
fit_large_bw <- ksmooth(x, y, kernel = "normal", bandwidth = 2)

# Plot the data and the fitted curves
plot(x, y, col = "blue", pch = 16, xlab = "x", ylab = "y", main = "Data and Fitted Cu
lines(fit_cv, col = "green", lwd = 2, lty = 1)  # Cross-validated bandwidth
lines(fit_small_bw, col = "orange", lwd = 2, lty = 2)  # Small bandwidth
lines(fit_large_bw, col = "purple", lwd = 2, lty = 3)  # Large bandwidth
legend("topright", legend = c("Data", "CV Bandwidth", "Small BW", "Large BW"), col =
```



The fit in this case looks satisfactory.

```
# Print out the selected bandwidth
cat("Selected bandwidth after cross-validation:", optimal_bw, "\n")
```

```
Selected bandwidth after cross-validation: 0.1
```

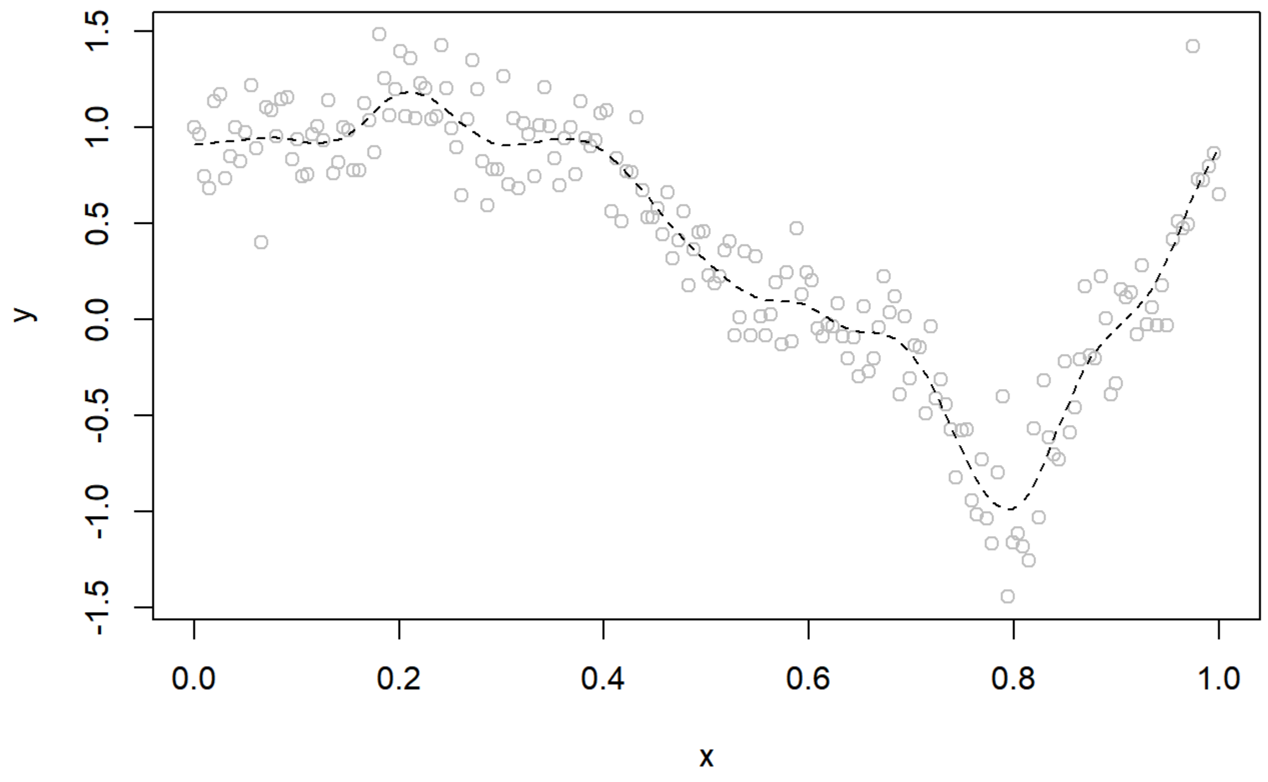
We can see in the graph that a bandwith of 0.005 overfits the data.

A bandwith of 2 underfits the data.

A value of 0.1, achieved through cross-validation, looks to be a good fit.

Part C: Fit smoothing spline

```
#Smoothing spline
plot(y ~ x, col = gray(0.75))
fit <- smooth.spline(x, y)
lines(fit, lty = 2)
```



```
# Get the effective degrees of freedom
df <- fit$df
print(paste("Effective Degrees of Freedom:", df))
```

```
[1] "Effective Degrees of Freedom: 18.8683711230771"
```

A relatively high degrees of freedom and a well fit curve to the data indicate that the automatic choice for the degrees of freedom was satisfactory.

Part D: Fit regression splines

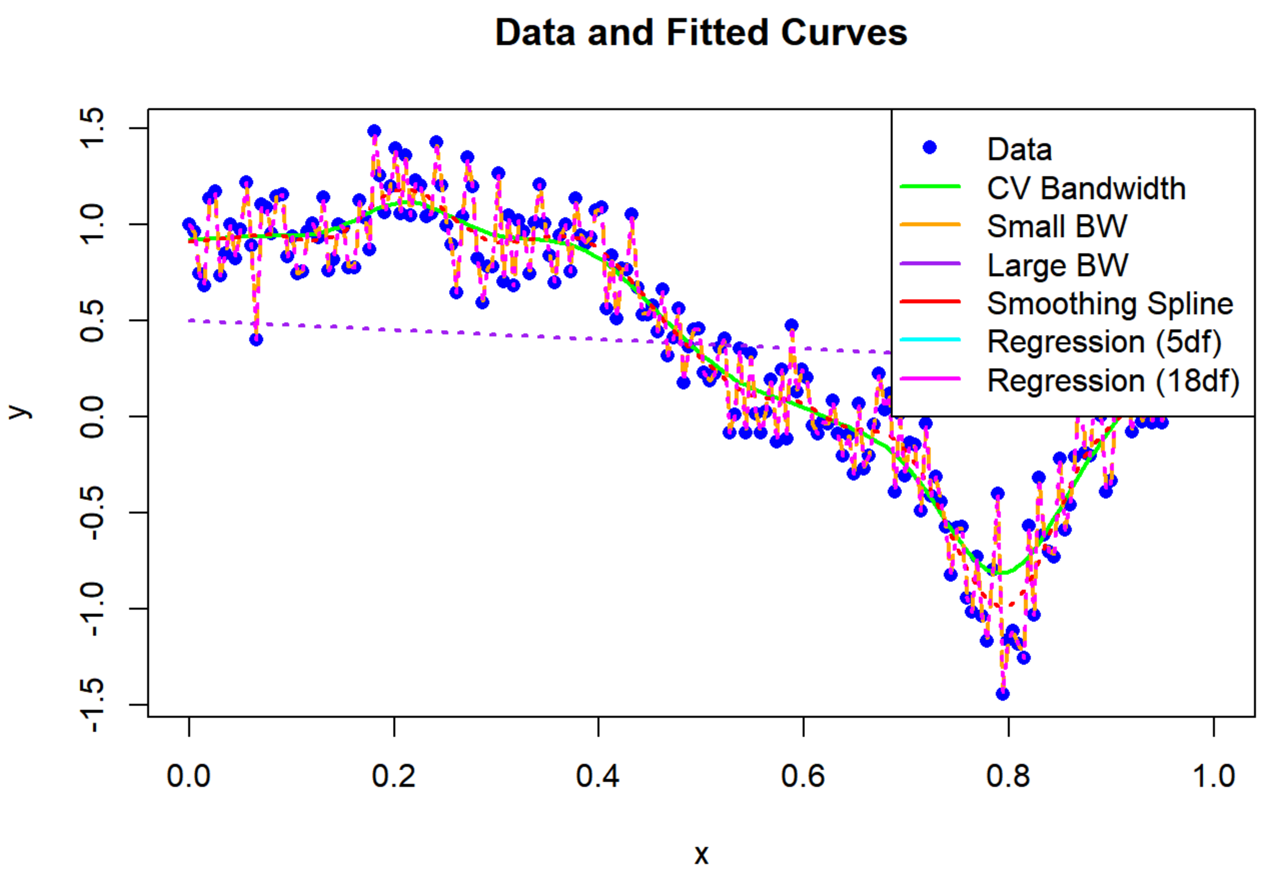
```
#Regression splines
xtilde5 <-bs(y, df = 5)
xtilde18 <-bs(y, df = 18)
```

```
# Fit regression splines with 5 and 18 degrees of freedom using xtilde5 and xtilde18
fit_5_df <- lm(y ~ xtilde5)
fit_18_df <- lm(y ~ xtilde18)
# Generate fitted values for the splines
fitted_5_df <- predict(fit_5_df)
fitted_18_df <- predict(fit_18_df)
```

Plot regression splines with all previous curve estimates

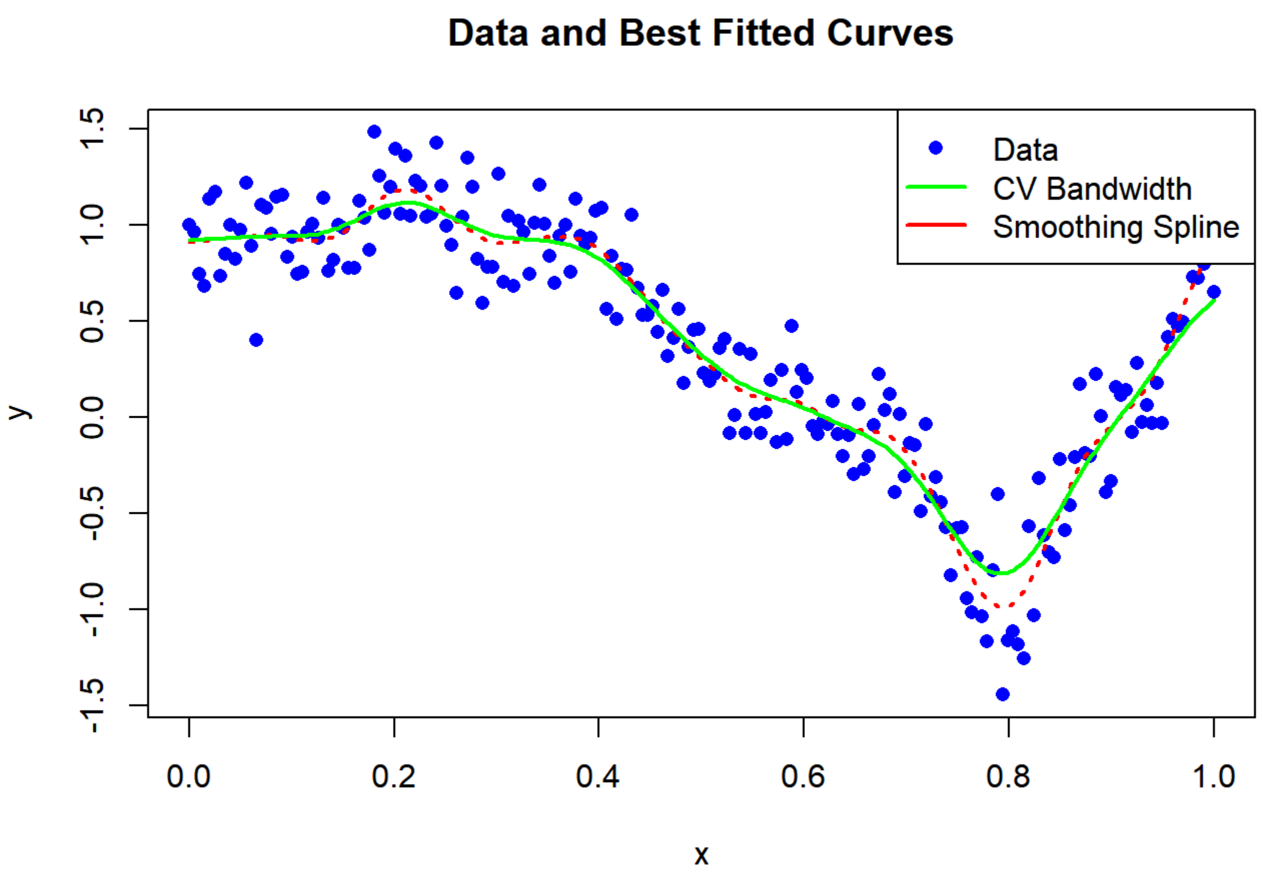
```
#Plot all togteher
plot(x, y, col = "blue", pch = 16, xlab = "x", ylab = "y", main = "Data and Fitted Cu
lines(fit_cv, col = "green", lwd = 2, lty = 1)  # Cross-validated bandwidth
lines(fit_small_bw, col = "orange", lwd = 2, lty = 2)  # Small bandwidth
lines(fit_large_bw, col = "purple", lwd = 2, lty = 3)  # Large bandwidth
lines(fit, col = "red", lwd = 2, lty = 3) # Smoothing spline
lines(x, fitted_5_df, col = "cyan", lwd = 2, lty = 3) #5df
lines(x, fitted_18_df, col = "magenta", lwd = 2, lty = 3) #18df

legend("topright",
       legend = c("Data", "CV Bandwidth", "Small BW", "Large BW", "Smoothing Spline",
       col = c("blue", "green", "orange", "purple", "red", "cyan", "magenta"),
       pch = c(16, NA, NA, NA, NA, NA, NA),
       lwd = c(NA, 2, 2, 2, 2, 2, 2))
```



```
#Isolating the best fitting curves
plot(x, y, col = "blue", pch = 16, xlab = "x", ylab = "y", main = "Data and Best Fitt
```

```r
lines(fit, col = "red", lwd = 2, lty = 3) # Smoothing spline
lines(fit_cv, col = "green", lwd = 2, lty = 1)  # Cross-validated bandwidth
legend("topright",
       legend = c("Data", "CV Bandwidth", "Smoothing Spline"),
       col = c("blue", "green", "red"),
       pch = c(16, NA, NA ),
       lwd = c(NA, 2, 2))
```



CV bandwith and smoothing spline perform the best.