APA_Assignment_2

AUTHOR
Sam Ryder, 18317496

Library Installation

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

```
Loading required package: Rcpp

Loading required package: numDeriv
```

```
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
```

```
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
```

```
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 training
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, "\n")
```

```
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
```

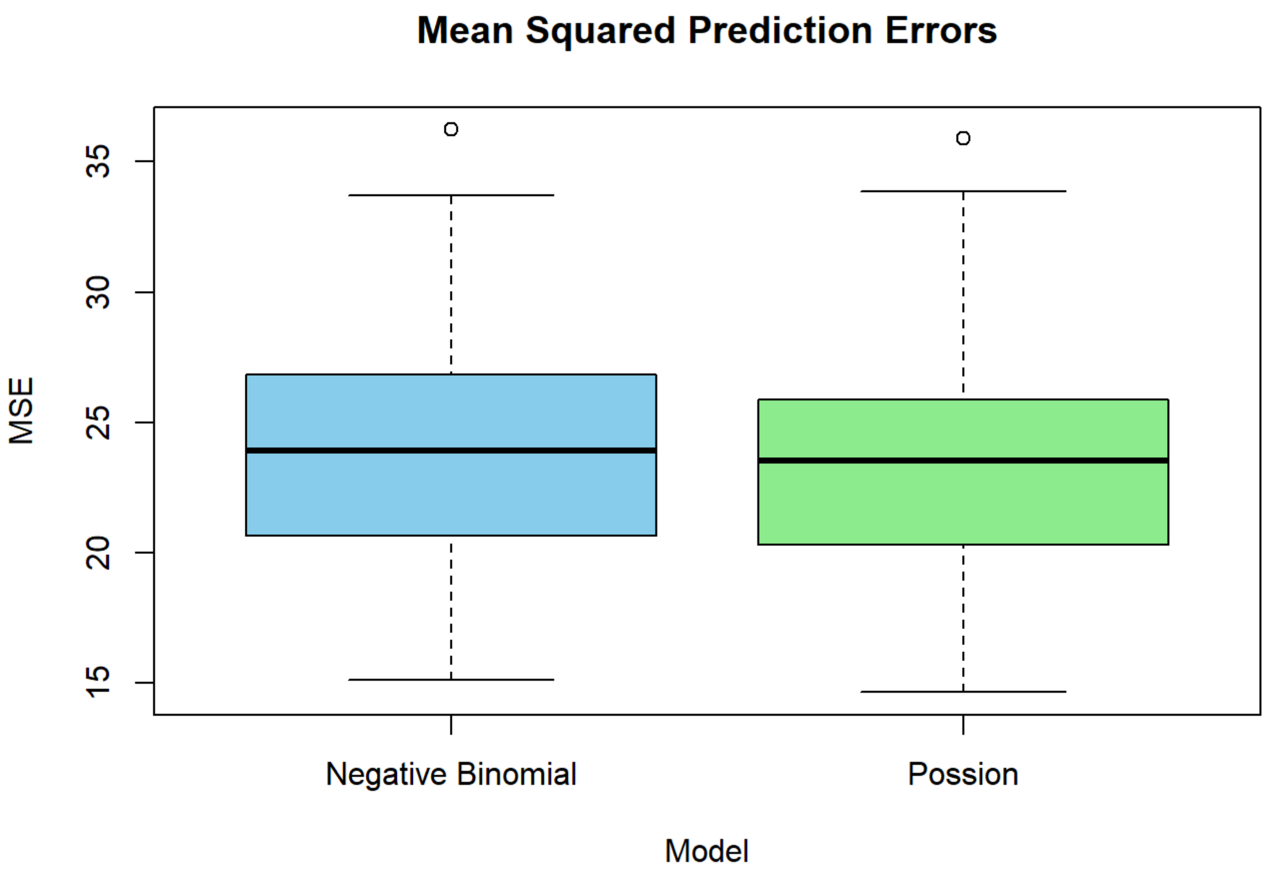
```
cat("Mean Squared Prediction Error (MSPE) for Negative Binomial Model:", mean_mse_nb, "\n")
```

```
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-qaurtile 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)
```

```r
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

```r
#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

```r
#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(random_effects_df)
```

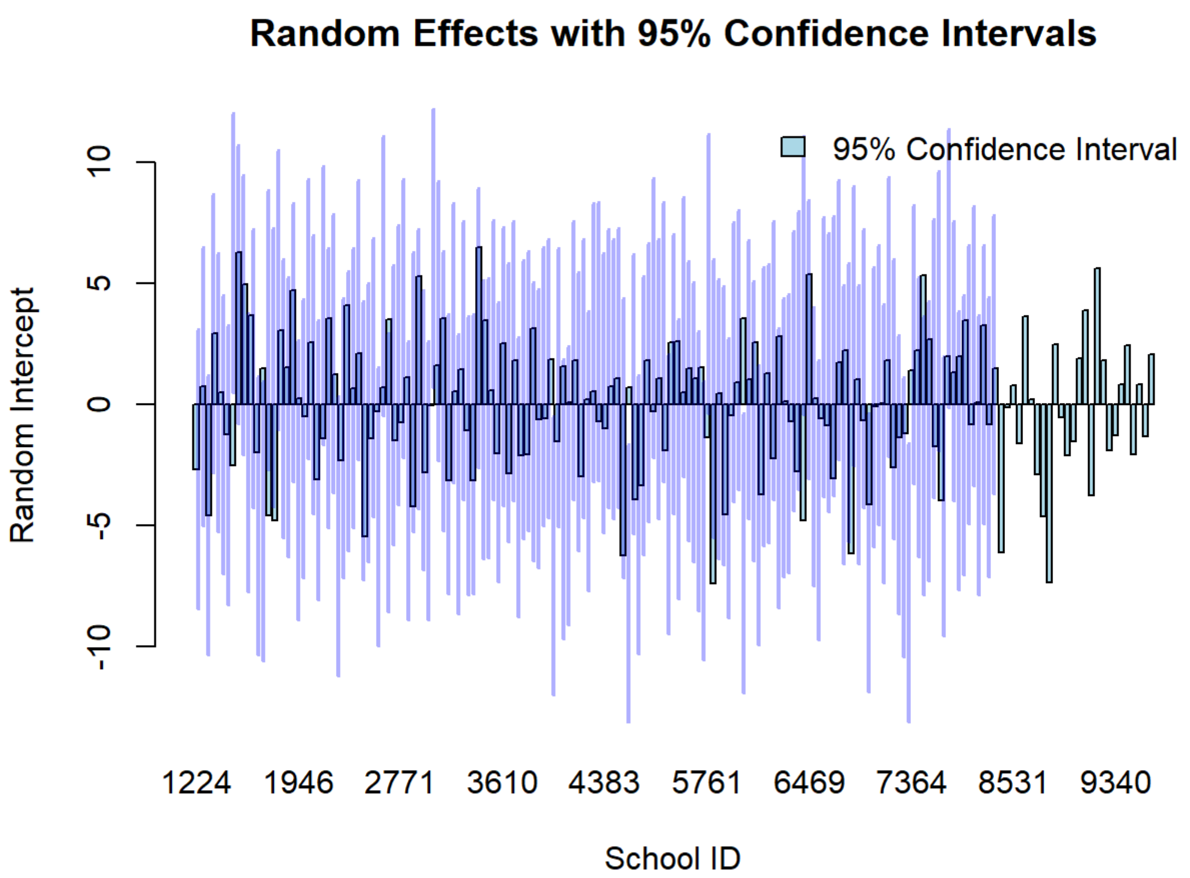
```
   School_ID Random_Intercept    Lower_CI   Upper_CI
1       1224      -2.66393477  -8.4164679  3.0885984
2       1288       0.73940980  -5.0131234  6.4919430
3       1296      -4.56846564 -10.3209988  1.1840675
4       1308       2.94851711  -2.8040161  8.7010503
5       1317       0.49394606  -5.2585871  6.2464792
6       1358      -1.24251161  -6.9950448  4.5100216
7       1374      -2.50234967  -8.2548828  3.2501835
8       1433       6.26824322   0.5157101 12.0207764
9       1436       4.96210829  -0.7904249 10.7146415
10      1461       3.69657486  -2.0559583  9.4491080
11      1462      -1.98328158  -7.7358147  3.7692516
12      1477       1.48280172  -4.2697314  7.2353349
13      1499      -4.58357634 -10.3361095  1.1689568
14      1637      -4.80420489 -10.5567381  0.9483283
15      1906       3.08192292  -2.6706102  8.8344561
16      1909       1.53689246  -4.2156407  7.2894256
17      1942       4.73230256  -1.0202306 10.4848357
18      1946       0.24312986  -5.5094033  5.9956630
19      2030      -0.50951426  -6.2620474  5.2430189
20      2208       2.57281423  -3.1797189  8.3253474
21      2277      -3.10782327  -8.8603564  2.6447099
22      2305      -1.40397847  -7.1565116  4.3485547
23      2336       3.53856291  -2.2139703  9.2910961
24      2458       1.24911599  -4.5034172  7.0016492
25      2467      -2.28936744  -8.0419006  3.4631657
26      2526       4.08992838  -1.6626048  9.8424615
27      2626       0.67848598  -5.0740472  6.4310191
28      2629       2.10311291  -3.6494203  7.8556461
29      2639      -5.43354661 -11.1860798  0.3189866
30      2651      -1.38679973  -7.1393329  4.3657334
31      2655      -0.26834757  -6.0208807  5.4841856
32      2658       0.68954228  -5.0629909  6.4420754
33      2755       3.50100329  -2.2515299  9.2535365
34      2768      -1.48085163  -7.2333848  4.2716815
35      2771      -0.73234974  -6.4848829  5.0201834
36      2818       1.11518774  -4.6373454  6.8677209
37      2917      -4.21276807  -9.9653012  1.5397651
38      2990       5.30834045  -0.4441927 11.0608736
39      2995      -2.81295056  -8.5654837  2.9395826
40      3013      -0.02407887  -5.7766120  5.7284543
41      3020       1.63254393  -4.1199892  7.3850771
42      3039       3.55707769  -2.1954555  9.3096109
43      3088      -3.12676214  -8.8792953  2.6257710
44      3152       0.52608559  -5.2264476  6.2786188
45      3332       1.46586925  -4.2866639  7.2184024
46      3351      -1.04949532  -6.8020285  4.7030378
47      3377      -3.13377136  -8.8863045  2.6187618
48      3427       6.47780566   0.7252725 12.2303388
```

49	3498	3.45703987	-2.2954933	9.2095730
50	3499	0.57123412	-5.1812990	6.3237673
51	3533	-2.03523294	-7.7877661	3.7173002
52	3610	2.53776919	-3.2147640	8.2903024
53	3657	-2.86086000	-8.6133932	2.8916732
54	3688	1.82626226	-3.9262709	7.5787954
55	3705	-2.09382198	-7.8463551	3.6587112
56	3716	-2.04196948	-7.7945026	3.7105637
57	3838	3.15989926	-2.5926339	8.9124324
58	3881	-0.61912615	-6.3716593	5.1334070
59	3967	-0.55352009	-6.3060533	5.1990131
60	3992	1.84962912	-3.9029040	7.6021623
61	3999	-1.54071082	-7.2932440	4.2118223
62	4042	1.56716196	-4.1853712	7.3196951
63	4173	0.07947625	-5.6730569	5.8320094
64	4223	1.80350561	-3.9490276	7.5560388
65	4253	-2.98983646	-8.7423696	2.7626967
66	4292	0.21252084	-5.5400123	5.9650540
67	4325	0.55540089	-5.1971323	6.3079341
68	4350	-0.68694444	-6.4394776	5.0655887
69	4383	-0.99111944	-6.7436526	4.7614137
70	4410	0.75258064	-4.9999525	6.5051138
71	4420	1.08332583	-4.6692073	6.8358590
72	4458	-6.23521734	-11.9877505	-0.4826842
73	4511	0.71596003	-5.0365731	6.4684932
74	4523	-3.90739873	-9.6599319	1.8451344
75	4530	-3.34031074	-9.0928439	2.4122224
76	4642	1.82601435	-3.9265188	7.5785475
77	4868	-0.28826542	-6.0407986	5.4642677
78	4931	1.06999472	-4.6825384	6.8225279
79	5192	-1.91641743	-7.6689506	3.8361157
80	5404	2.57286889	-3.1796643	8.3254021
81	5619	2.60021914	-3.1523140	8.3527523
82	5640	0.48450124	-5.2680319	6.2370344
83	5650	1.48643852	-4.2660946	7.2389717
84	5667	1.06212367	-4.6904095	6.8146568
85	5720	1.51538472	-4.2371484	7.2679179
86	5761	-1.37844240	-7.1309756	4.3740908
87	5762	-7.40281891	-13.1553521	-1.6502857
88	5783	0.44381828	-5.3087149	6.1963514
89	5815	-4.54024780	-10.2927810	1.2122854
90	5819	-0.45657382	-6.2091070	5.2959593
91	5838	0.91804951	-4.8344837	6.6705827
92	5937	3.57823119	-2.1743020	9.3307644
93	6074	1.05638390	-4.6961493	6.8089171
94	6089	2.57761589	-3.1749173	8.3301491
95	6144	-3.70076588	-9.4532990	2.0517673
96	6170	1.26936413	-4.4831690	7.0218973
97	6291	-2.23893571	-7.9914689	3.5135975
98	6366	2.80002088	-2.9525123	8.5525540
99	6397	0.14792180	-5.6046114	5.9004550
100	6415	-0.71647087	-6.4690040	5.0360623
101	6443	-2.74551993	-8.4980531	3.0070132
102	6464	-4.79405539	-10.5465886	0.9584778

103	6469	5.38906503	-0.3634681	11.1415982
104	6484	0.24377254	-5.5087606	5.9963057
105	6578	-0.59470973	-6.3472429	5.1578234
106	6600	-0.86304032	-6.6155735	4.8894928
107	6808	-3.03715524	-8.7896884	2.7153779
108	6816	1.75614970	-3.9963835	7.5086829
109	6897	2.25180532	-3.5007278	8.0043385
110	6990	-6.13417910	-11.8867123	-0.3816459
111	7011	1.03417645	-4.7183567	6.7867096
112	7101	-0.67713803	-6.4296712	5.0753951
113	7172	-4.14230120	-9.8948344	1.6102320
114	7232	-0.08675679	-5.8392900	5.6657764
115	7276	0.03907201	-5.7134612	5.7916052
116	7332	1.82622077	-3.9263124	7.5787539
117	7341	-2.61019710	-8.3627303	3.1423361
118	7342	-1.36370396	-7.1162371	4.3888292
119	7345	-1.20095583	-6.9534890	4.5515773
120	7364	1.39144182	-4.3610913	7.1439750
121	7635	2.22984899	-3.5226842	7.9823822
122	7688	5.33623561	-0.4162976	11.0887688
123	7697	2.70117929	-3.0513539	8.4537125
124	7734	-1.72167669	-7.4742099	4.0308565
125	7890	-3.94438335	-9.6969165	1.8081498
126	7919	1.97091161	-3.7816216	7.7234448
127	8009	1.32010092	-4.4324322	7.0726341
128	8150	2.00798675	-3.7445464	7.7605199
129	8165	3.49050825	-2.2620249	9.2430414
130	8175	-0.82523286	-6.5777660	4.9273003
131	8188	0.09031151	-5.6622217	5.8428447
132	8193	3.25161510	-2.5009181	9.0041483
133	8202	-0.81829089	-6.5708241	4.9342423
134	8357	1.49348952	-4.2590436	7.2460227
135	8367	-6.10301152	-11.8555447	-0.3504784
136	8477	-0.10217980	-5.8547130	5.6503534
137	8531	0.80272915	-4.9498040	6.5552623
138	8627	-1.61478957	-7.3673227	4.1377436
139	8628	3.62158235	-2.1309508	9.3741155
140	8707	0.22560321	-5.5269300	5.9781364
141	8775	-2.89579516	-8.6483283	2.8567380
142	8800	-4.64179687	-10.3943300	1.1107363
143	8854	-7.35291363	-13.1054468	-1.6003805
144	8857	2.48360018	-3.2689330	8.2361333
145	8874	-0.51671482	-6.2692480	5.2358183
146	8946	-2.09753091	-7.8500641	3.6550023
147	8983	-1.51038057	-7.2629137	4.2421526
148	9021	1.90507890	-3.8474543	7.6576121
149	9104	3.87494381	-1.8775894	9.6274770
150	9158	-3.76864491	-9.5211781	1.9838883
151	9198	5.62994498	-0.1225882	11.3824781
152	9225	1.80277345	-3.9497597	7.5553066
153	9292	-1.90231629	-7.6548495	3.8502169
154	9340	-1.26083157	-7.0133647	4.4917016
155	9347	0.83518933	-4.9173438	6.5877225
156	9359	2.42565070	-3.3268825	8.1781839

157	9397	-2.08034443	-7.8328776	3.6721887
158	9508	0.82991908	-4.9226141	6.5824522
159	9550	-1.33813136	-7.0906645	4.4144018
160	9586	2.06746599	-3.6850672	7.8199992

```r
# 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")
```

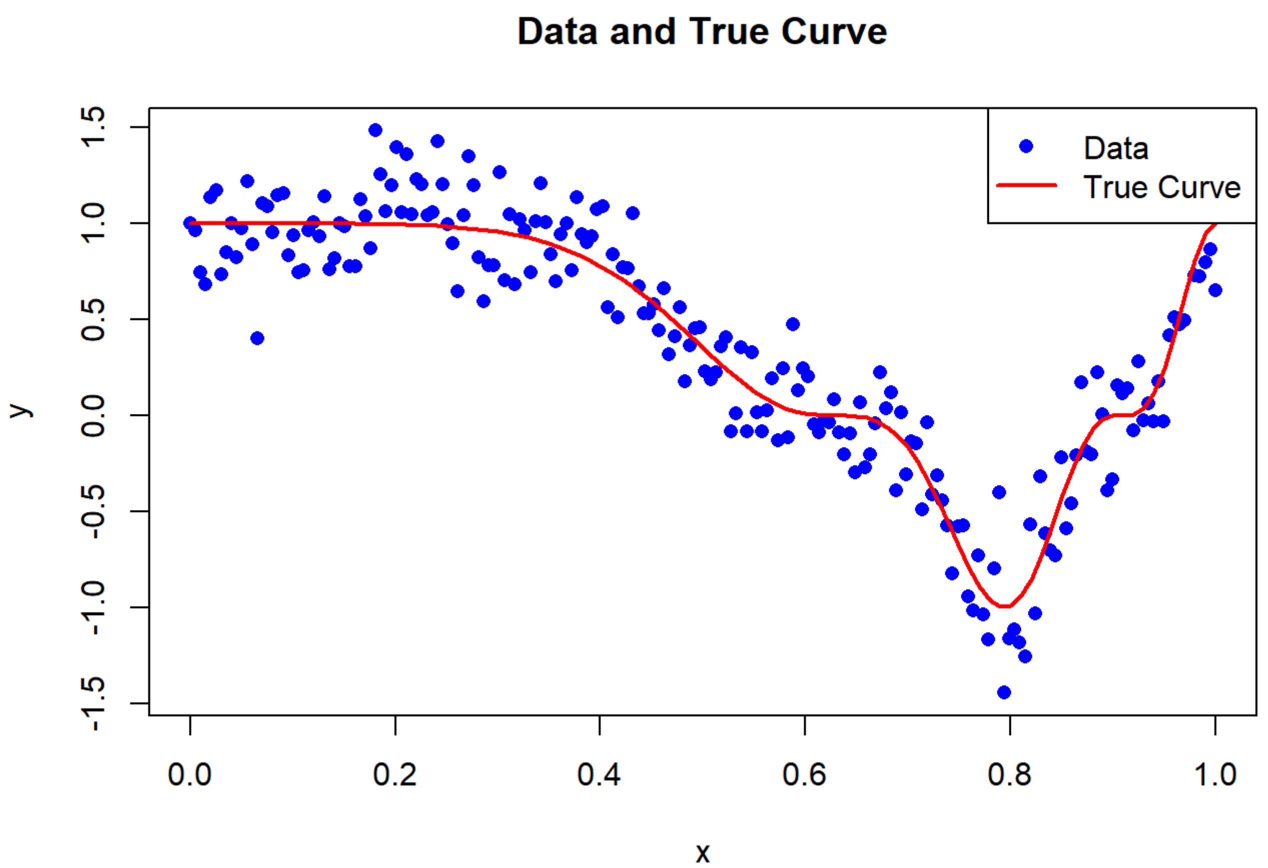


Question 3

Part A: Generate data sample and plot curve

```r
# 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 Curve")
#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(16, NA),
```



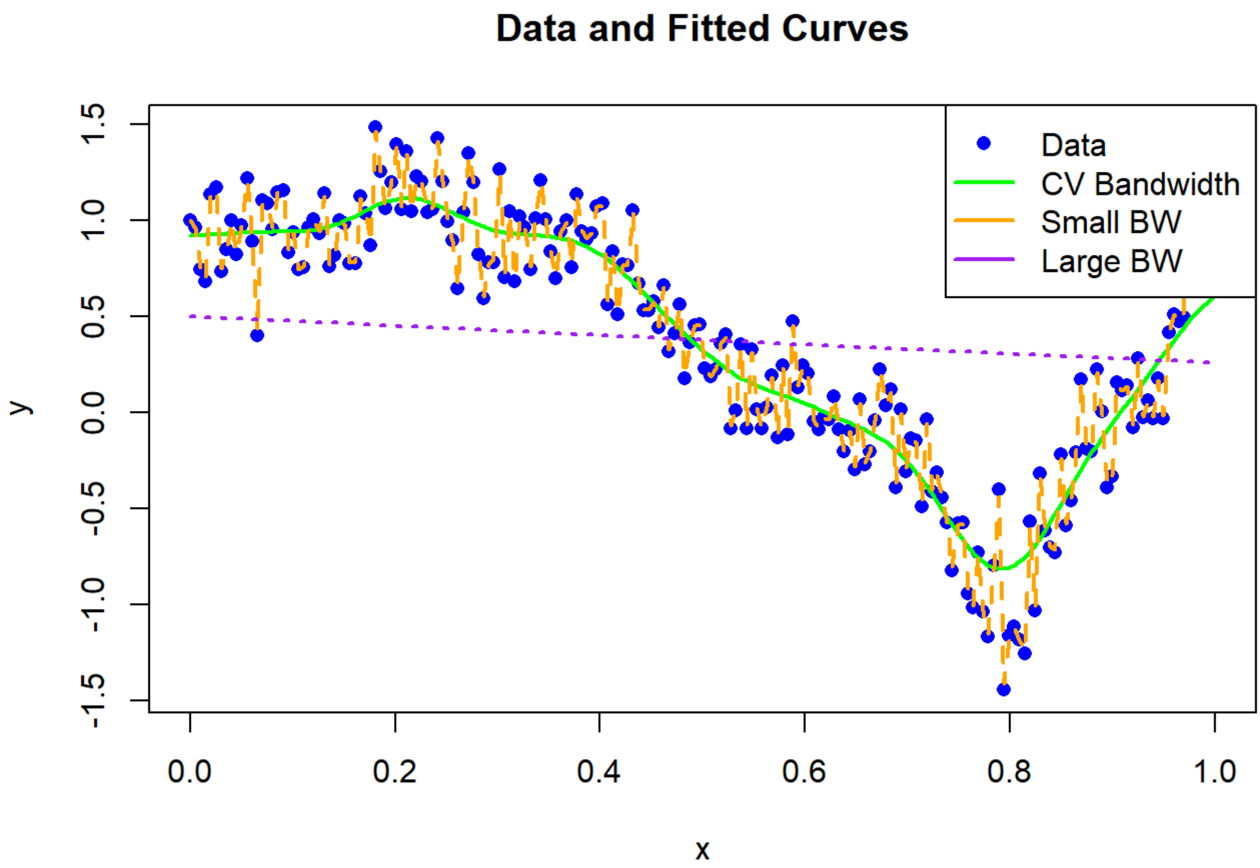
Part B: Fit kernel smoothing spline

```
#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 Curves")
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 = c("blue",
```



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
```

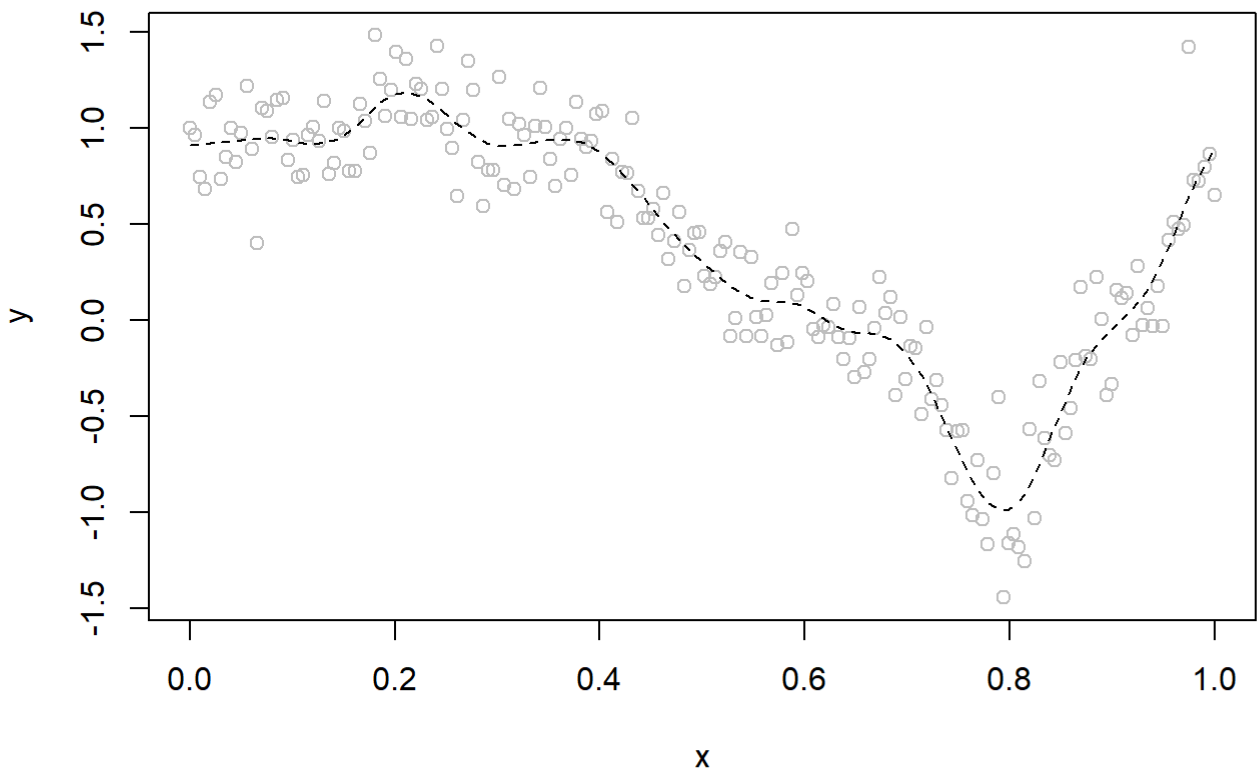
We can see in the graph that a bandwith of 0.005 overfits the data.

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

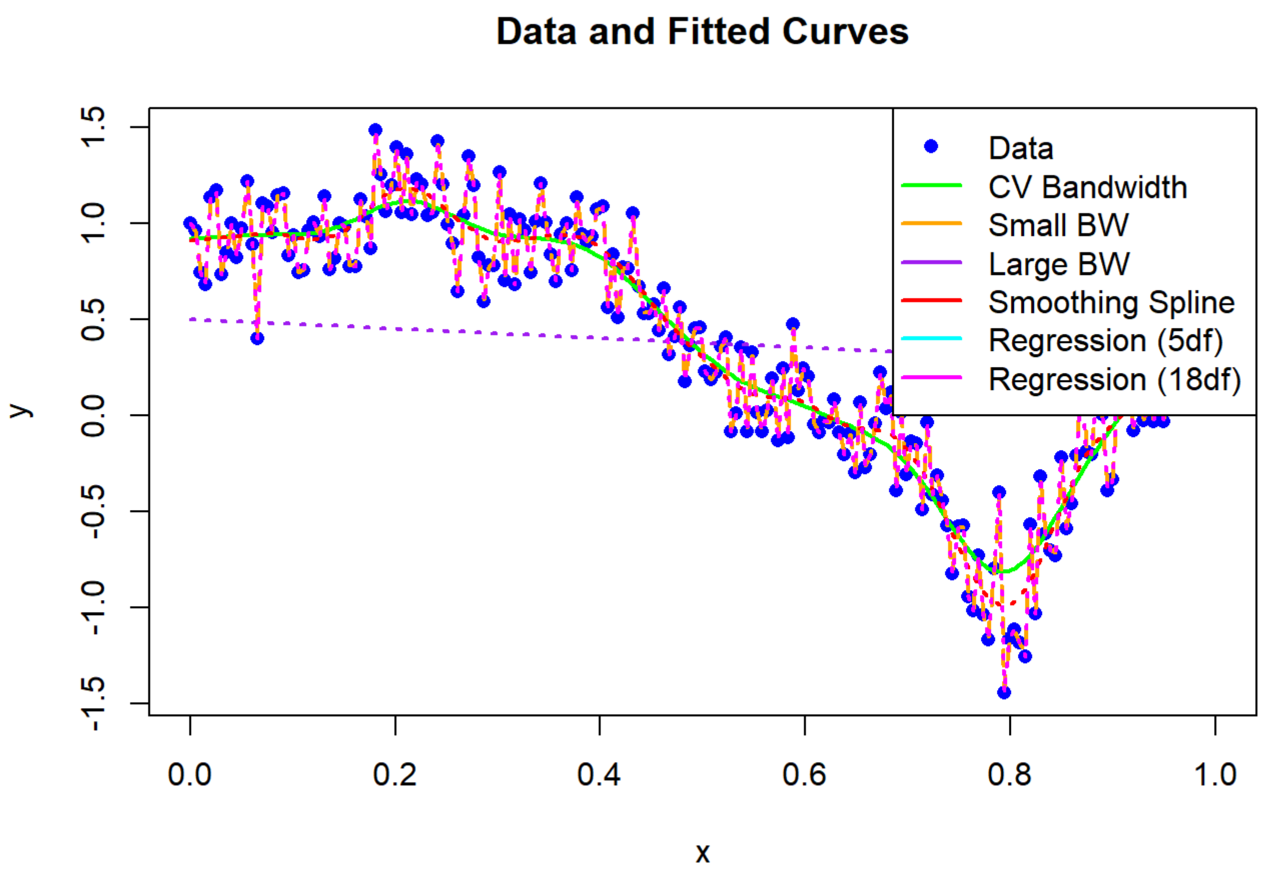
```
#Regression splines
xtilde5 <-bs(y, df = 5)
xtilde18 <-bs(y, df = 18)
# Fit regression splines with 5 and 18 degrees of freedom using xtilde5 and xtilde18
fit_5_df <- lm(y ~ xtilde5)
fit_18_df <- lm(y ~ xtilde18)
# Generate fitted values for the splines
fitted_5_df <- predict(fit_5_df)
fitted_18_df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

Plot regression splines with all previous curve estimates

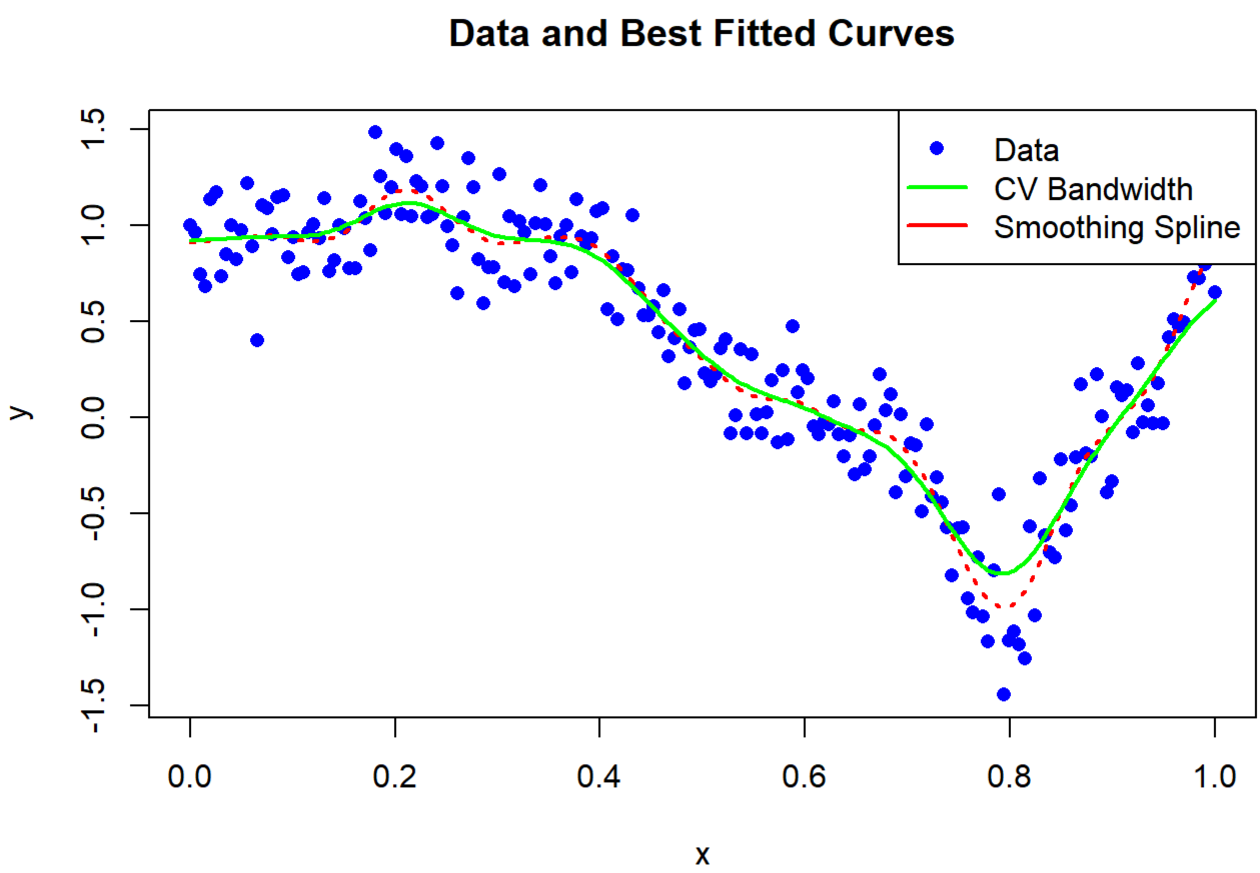
```
#Plot all togteher
plot(x, y, col = "blue", pch = 16, xlab = "x", ylab = "y", main = "Data and Fitted Curves")
lines(fit_cv, col = "green", lwd = 2, lty = 1)  # Cross-validated bandwidth
```

```
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", "Regress
       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 Fitted Curves
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.