Data 558: Statistical Machine Learning

Spring 2023

Homework – 5

Arjun Sharma

I. Conceptual Questions

Problem 1 (20 points): Please determine whether following practices are good or not. Please explain why in either case. [Grading criterion: each question subpart is worth 5 points with 2 point assigned for the answer and 3 points for the explanation.]

(a) We take a look at the data and see that a (possibly) different cubic model fits many regions of my data. So I go ahead and fit a cubic spline to my data and show our boss this result.

(b) We are fitting a smoothing spline to the data for a particular value of λ. We notice that the data is a bit wiggly so we increase λ and obtain another smoothing spline.

(c) We fit a natural quintic spline to the data and see that it is overfitting. We think that we have two options: we could adjust the number of knots or the complexity of the natural spline. We think quintic functions are too complicated so we decrease the degree of the natural spline to cubic.

(d) Our boss cares particularly about a natural spline fit that has low variance, meaning that if fit it to another dataset coming from the same mechanism, our model doesn't change very much. Our boss thinks also k number of knots is appropriate, but our task is to now place these knots appropriately. So we decide to space them equally in our x domain.

Solution:

(a) It may be good practise depending on the situation:

If the boss is technically sound and understands splines, interpretability would not be too much of a concern, otherwise interpretability is an issue.

If the underlying relationship between the data is complex, it may be considered good practise to do so, otherwise we risk overfitting, especially with a higher number of knots

(b) This is bad practise. We are changing lambda simply based on the 'wiggliness' of the data, and not based on any other criterion. We are penalizing the spline harshly just to alter the shape of the spline. We should use other methods to determine the optimal value for lambda and not simply increase it. An alternative could be k-fold cross validation.

(c) This may be bad practise depending on the kind of data we are working with.

While the measure taken is fair, to reduce overfitting by reducing complexity is the right thing to do via the bias-variance tradeoff. However, we must also account for the regions where our quintic model overfits the data and where it generalizes well. In order to determine the optimal complexity for different regions in the data, we can use alternatives like changing the number of knots and where they lie. This can help us assign a suitable model for each region of data.

(d) This is bad practise. By equally spacing our knots, we have not optimized knot placement for the complexity in the data by the region. Although I would argue that a predetermined

number of knots is not good practise, It would be good practise to follow the boss' instructions but also evaluate the regions for which the placement of a knot would be justified. This can be done through methods like K-fold cross validation as an example. By equally spacing our knots, we are optimizing for the complexity in the data across the region.

Problem 2 (12 points): Recall that a smoothing spline solves the optimization problem for some m

$$\arg\min_g \sum_{i=1}^{n} (y^{(i)} - g(x^{(i)}))^2 + \lambda \int_a^b \left(g^{(m)}x\right)^2 \partial x,$$

with $g^{(m)}$ representing the m-th derivative of g. Please describe what the optimal function in the optimization would be in each of these settings: [Grading criterion: each question subpart is worth 4 points with 1 point assigned for the answer and 3 points for the explanation.]

(a) $\lambda = \infty$ and $m = 0$

(b) $\lambda = \infty$ and $m = 2$

(c) $\lambda = \infty$ and $m = 3$

(d) $\lambda = 0$ and $m = 3$

a. λ is infinite, and hence leads to extremely high model complexity irrespective of the value of the g term. This leads to a piecewise constant model that overfits the data in a very extreme manner.

b. λ is infinite but m is 2, hence we have the 2nd derivative of the g term. This means that our optimal function is a model that is piecewise cubic, where a third-degree curve connecting each datapoint, so each datapoint is also a knot.

c. λ is infinite but m is 3, hence we have the 3rd derivative of the g term. Our optimal function is once again a model which is piecewise cubic like in b, where once again, each datapoint is a knot.

d. The λ is 0, hence the penalty does not exist. In this case, the optimal function is just the RSS from the data.

Problem 3 (8 points): We want to build a spline with two knots using the R code function $lm()$. We want to have two knots: one at $x = 0$, another at $x = 1$. We want the function on $(-\infty, 0)$ to be linear, the one on $(0, 1)$ to be quadratic and the one on $(1, \infty)$ to be cubic. Construct the set of basis functions for fitting to this model and explain why the basis function put together would satisfy the assumptions needed for a spline. Write down the one-line R code with the $lm(\cdot)$ function you would use to fit such a spline. [Grading criterion: 4 points for the answer and 4 points for the explanation.]

Given that we have 2 knots, we require 3 basis functions. The 3 basis functions are described as:

Function 1: Linear when < 1 - f1(x) = x for x ∈ (−∞, 0), 0 otherwise

Function 2: quadratic when >= 0 and <= 1 - $f_2(x) = x^2$ for x ∈ (0, 1), 0 otherwise

Function 3: cubic when > 1 - $f_3(x) = (x - 1)^3$ for x ∈ (1, ∞), 0 otherwise

The three functions are continuous for the specified conditions, they also incorporate the knots as in the question. Also, these functions can be described as a piece-wise equation. Hence, we can verify that the assumptions for a spline are satisfied.

The spline can be written in R as follows (y is our target): Spline_fit <- lm(y ~ f1 + f2 + f3, data = data)

Problem 4 (4 BONUS points): Suppose we are learning a spline with 3 knots, where the complexity of the functions from the leftmost to the rightmost regions are linear, quadratic, cubic, quadratic. What are the total number of parameters in this model? [Grading criterion: 2 points for the answer and 2 points for the explanation.]

Number of parameters by complexity:

1. Linear: 2 parameters (slope, intercept)

2. Quadratic: 3 parameters (intercept, linear coefficient, quadratic coefficient)

3. Cubic: 4 parameters (intercept, linear, quadratic and cubic coefficients)

Now, we have 4 functions, of which 2 are quadratic, and 1 each is linear and cubic.

Hence, we have 2+3+3+4 parameters = 12 parameters.

II. Applied Questions

Problem 1 (30 points): Generate data $Y = \sin(1/2 * X) + \epsilon$ where $X \sim \mathcal{N}(0, 1)$ and ϵ is standard normal. Draw $n = 100$ samples from (X, Y). Throughout, we will use cubic splines to fit a regression model. You may use R or Python functions for this task.

(a) Fit a cubic polynomial to the data and plot the the fitted polynomial with the data. Comment on the quality of the fit, especially at the extreme ends of the domain. [Grading criterion: 3 points for the answer and 3 points for the explanation.]

(b) We will consider $R = \{2, 3, 4, 5, 6\}$ knots placed at the quantiles of the data. Use 5-fold cross-validation to determine the number of knots, where the validation performance is based on RSS. Once you find the number of knots, refit to the entire data and plot the fitted cubic spline with data. Comment visually on the quality of the fit and its comparison to the polynomial fit from part a. [Grading criterion: 4 points for the answer and 4 points for the explanation.]

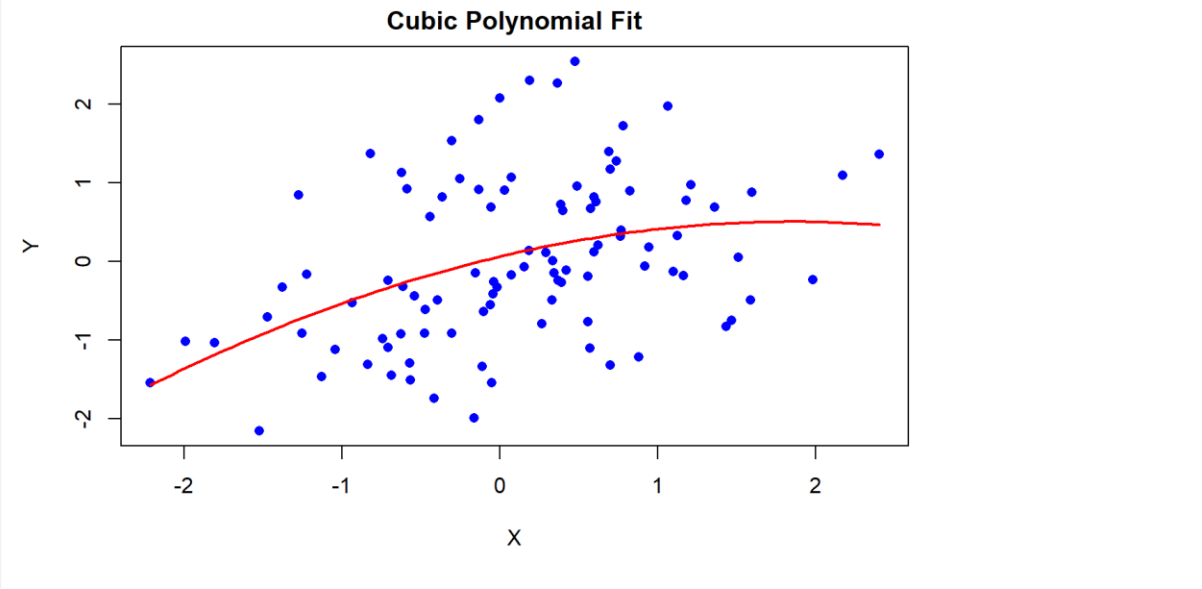
(c) Repeat the previous step but with a natural spline. Do you notice any improvements over a standard spline and a polynomial fit? [Grading criterion: 4 points for the answer and 4 points for the explanation.]

(d) Finally, use again 5-fold cross validation to fit a smoothing spline. Choose the regularization parameter via cross validation, and refit the entire data via this regularization parameter. Plot the fitted smoothing spline with the data. Compare its performance with the approaches in a-c. [Grading criterion: 4 points for the answer and 4 points for the explanation.]

```{r}
set.seed(1)
n <- 100
X <- rnorm(n)
epsilon <- rnorm(n)
Y <- sin(0.5 * X) + epsilon

```

(a)

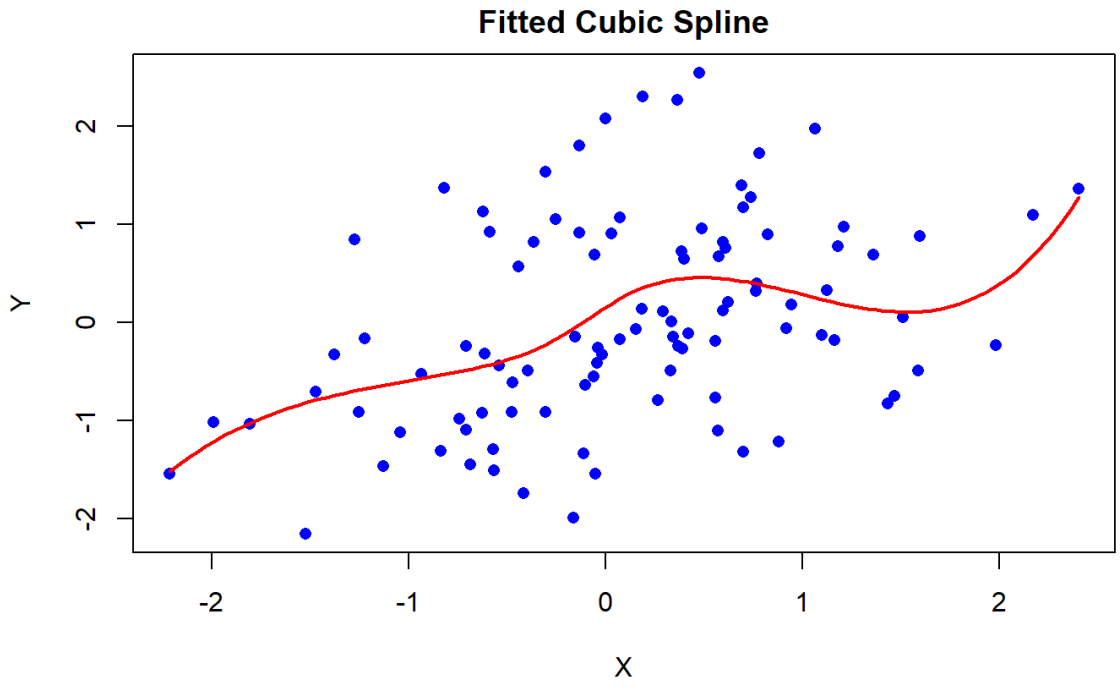
```r
# Fit a cubic polynomial
fit <- lm(Y ~ poly(X, 3, raw = TRUE))

x_plot <- seq(min(X), max(X), length.out = 100)
y_plot <- predict(fit, newdata = data.frame(X = x_plot))
plot(X, Y, pch = 16, col = "blue", xlab = "X", ylab = "Y", main = "Cubic Polynomial Fit")
lines(x_plot, y_plot, col = "red", lwd = 2)


```



Even though the function is in an approximate region with the distribution, it is a poor fit because the function foes not fit most of the points.

(b)

```r
library(splines)
num_knots <- 5

# Compute the quantiles
knots <- quantile(X, probs = seq(0, 1, length.out = num_knots + 2))[2:(num_knots + 1)]

# Perform 5-fold cross-validation to select the optimal number of knots
cv_errors <- numeric(num_knots)
folds <- cut(seq(1, nrow(data)), breaks = 5, labels = FALSE)

for (i in 1:num_knots) {
  errors <- numeric(5)
  for (j in 1:5) {
    train_indices <- which(folds != j)
    valid_indices <- which(folds == j)

    spline_fit <- lm(Y ~ bs(X, knots = knots[1:(i+2)]), data = data[train_indices, ])

    # Compute RSS
    errors[j] <- sum((Y[valid_indices] - predict(spline_fit, newdata = data.frame(X = X[valid_indices])))^2)
  }

  cv_errors[i] <- mean(errors)
}

optimal_num_knots <- which.min(cv_errors)

# Refit the spline to the entire data
final_fit <- lm(Y ~ bs(X, knots = knots[1:(optimal_num_knots)]), data = data)

x_plot <- seq(min(X), max(X), length.out = 100)

y_plot <- predict(final_fit, newdata = data.frame(X = x_plot))

# Plot the data and fitted spline
plot(X, Y, pch = 16, col = "blue", xlab = "X", ylab = "Y", main = "Fitted Cubic Spline")
lines(x_plot, y_plot, col = "red", lwd = 2)
```



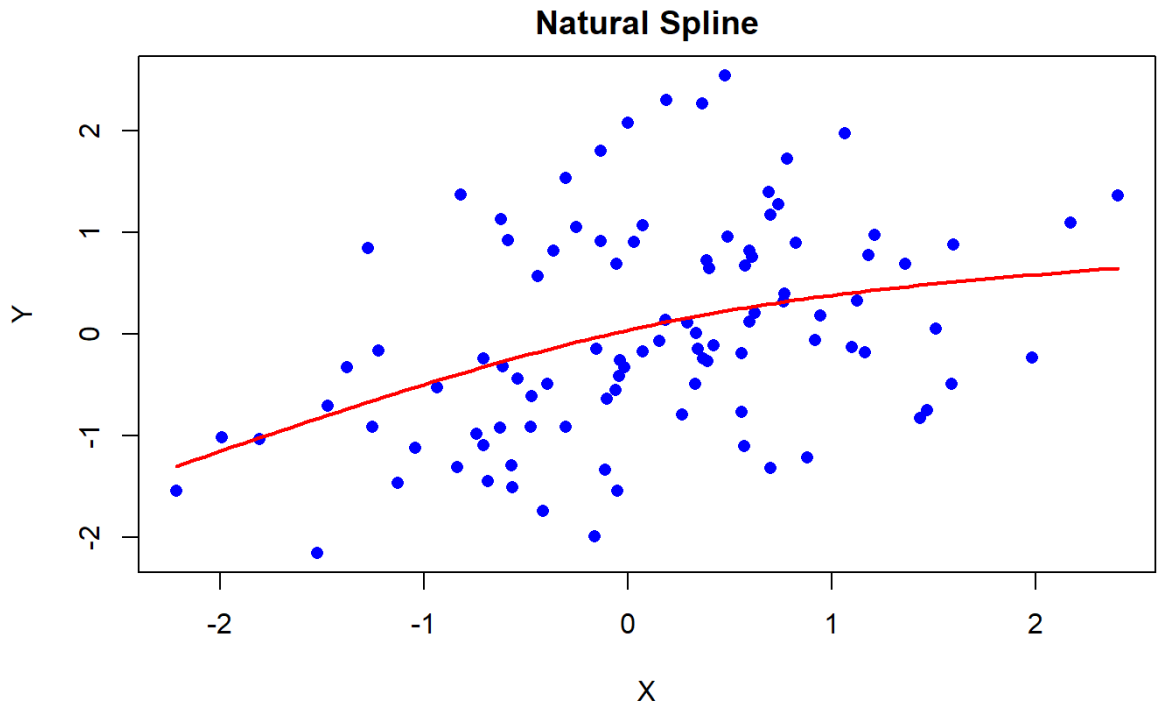

```r
cat("Optimal number of knots: ", optimal_num_knots)
```

```
Optimal number of knots:  3
```

This function still does not generalize well. Although the addition of knots allows for the function to better fit some regions of the data, it still does not fit most of the points.

c)

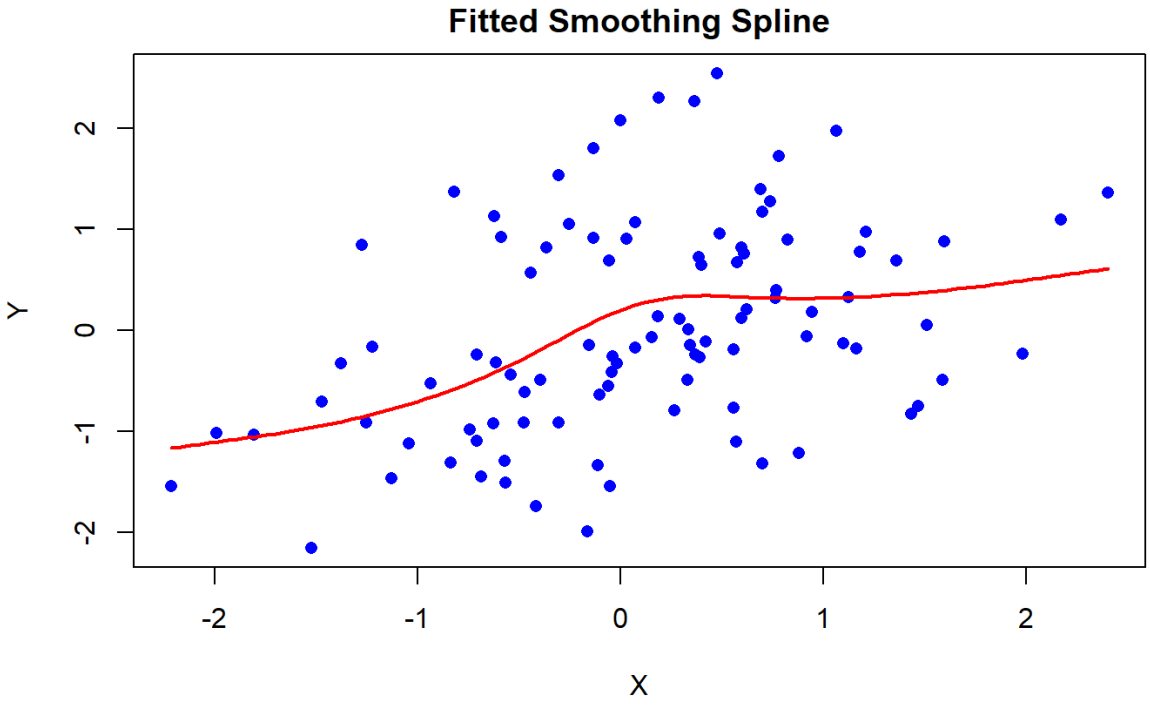
```{r}
num_knots <- 5

knots <- quantile(X, probs = seq(0, 1, length.out = num_knots + 2))[2:(num_knots + 1)]

cv_errors_natural <- numeric(num_knots)
for (i in 1:num_knots) {

  spline_fit_natural <- lm(Y ~ ns(X, df = i + 1, intercept = TRUE), data = data.frame(X = X, Y = Y))
  cv_errors_natural[i] <- sum((Y - predict(spline_fit_natural, newdata = data.frame(X = X)))^2)
}
optimal_num_knots_natural <- which.min(cv_errors_natural)
final_fit_natural <- lm(Y ~ ns(X, df = optimal_num_knots_natural, intercept = TRUE), data = data.frame(X = X, Y = Y))

x_plot <- seq(min(X), max(X), length.out = 100)

y_plot_standard <- predict(final_fit_standard, newdata = data.frame(X = x_plot))
y_plot_natural <- predict(final_fit_natural, newdata = data.frame(X = x_plot))

plot(X, Y, pch = 16, col = "blue", xlab = "X", ylab = "Y", main = "Natural Spline")
lines(x_plot, y_plot_natural, col = "red", lwd = 2, lty = 1)
```



The natural spline does not fit the data better than the previous fits, but it does appear smoother

(d)

```{r}

library(splines)
library(mgcv)

cv_errors <- numeric(5)

for (i in 1:knots) {
  gam_fit <- gam(Y ~ s(X, k = i, bs = "cr"), data = data.frame(X = X, Y = Y),
                 method = "REML", select = TRUE)

  cv_errors[i] <- sum((Y - predict(gam_fit, newdata = data.frame(X = X)))^2)
}

optimal_sp <- which.min(cv_errors)

# Refit the smoothing spline to the entire data
final_fit_smooth <- gam(Y ~ s(X, k = optimal_sp, bs = "cr"), data = data.frame(X = X, Y = Y),
                        method = "REML", select = TRUE, sp = optimal_sp)

x_plot <- seq(min(X), max(X), length.out = 100)


y_plot_smooth <- predict(final_fit_smooth, newdata = data.frame(X = x_plot))

plot(X, Y, pch = 16, col = "blue", xlab = "X", ylab = "Y", main = "Fitted Smoothing Spline")
lines(x_plot, y_plot_smooth, col = "red", lwd = 2)

```



We see that the smoothened spline tends to fit the data better than the other fits

Problem 2 (30 points): This problem considers daily air quality measurements in New York, from May to September in 1973 (in "airQuality.csv"). There are 153 observations on 6 variables: Ozone (ppb), Solar.R (solar radiation, measured in Langleys), Wind (mph), Temperature (degrees F), Month $(5-9)$, Day of month $(1-31)$. The response variable we are interested in is the cube root of Ozone. (Note: you need to transform the response variable yourself.)

(a) Use a smoothing cubic spline to fit each predictor to the response variable. Choose the regularization parameter via 5-fold cross validation. Report the regularization parameter for each predictor and plot the fitted smoothing spline for each predictor against the response. Comment on the fit of each predictor on the response. [Grading criterion: 5 points for the answer and 5 points for the explanation.]

(b) Consider all 5 predictors. Fit a GAM model with a smoothing spline on each parameter with the degrees of freedom set based on part a. Plot the resulting 1-D smoothing curve on each variable and comment on its relationship with the response variable. Compare to the plots in part a. [Grading criterion: 5 points for the results, 5 points for the explanations.]
Hint: You may find the "gam()" function in "mgcv" package helpful.

(c) Examine the adequacy of the GAM model you fit, using residual plots of residuals vs. each predictor and the fitted value respectively. Comment on how you would modify the GAM model (for example by trying different complexity models for each predictor) to obtain a better prediction model. [Grading criterion: 5 points for the results, 5 points for the explanations.]

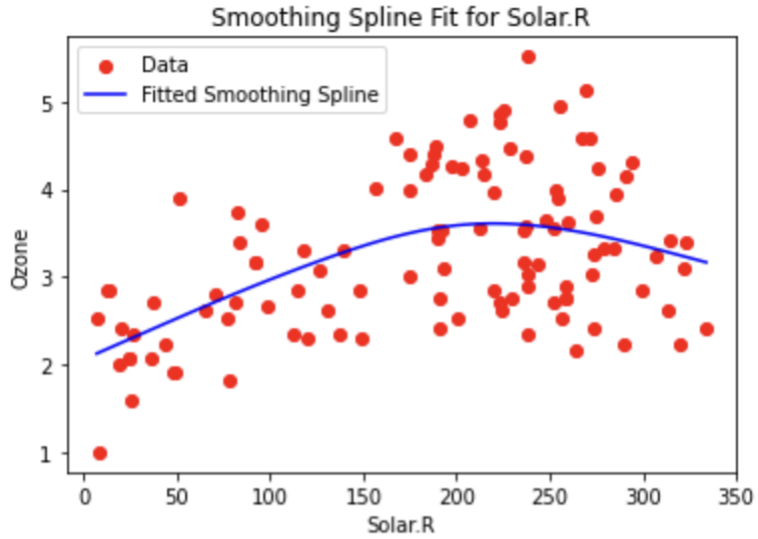
a)

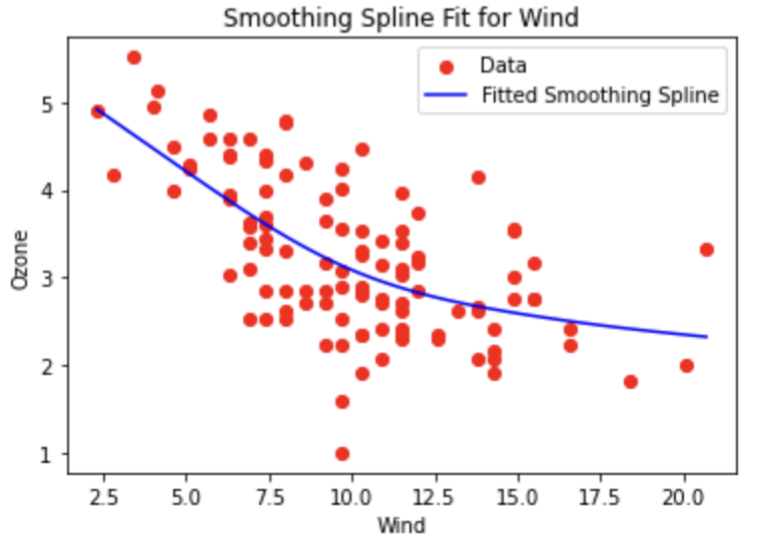
```
import numpy as np
import pandas as pd
from pygam import LinearGAM, s
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
```

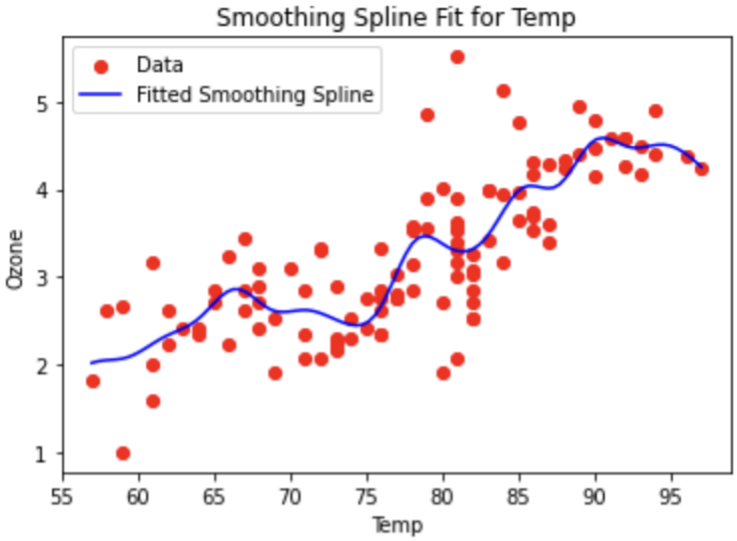
```
np.random.seed(1)
data = pd.read_csv("C://Users//arjun//Downloads//airQuality.csv")
data['Ozone'] = np.cbrt(data['Ozone'])
data = data.dropna()
X = data[['Solar.R', 'Wind', 'Temp', 'Month', 'Day']]
Y = data['Ozone']
```

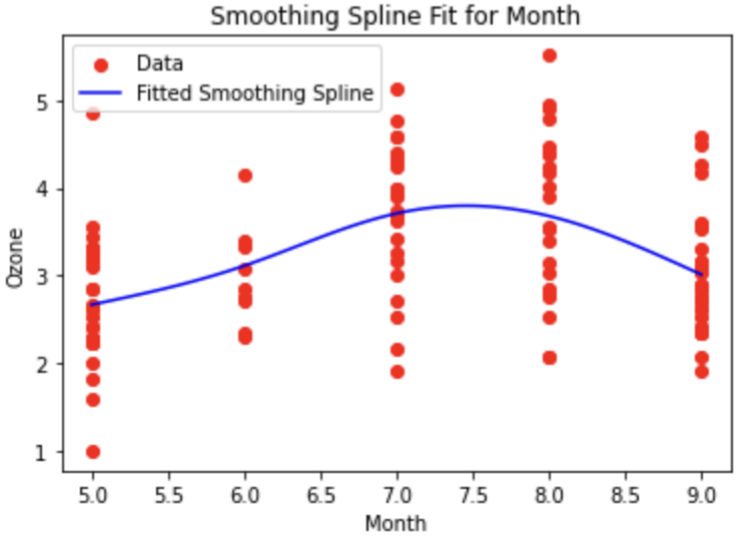
```
lam_range = np.logspace(-3, 4, 10)
cv_lam = []
cv_df = []
kf = KFold(n_splits=5)
```

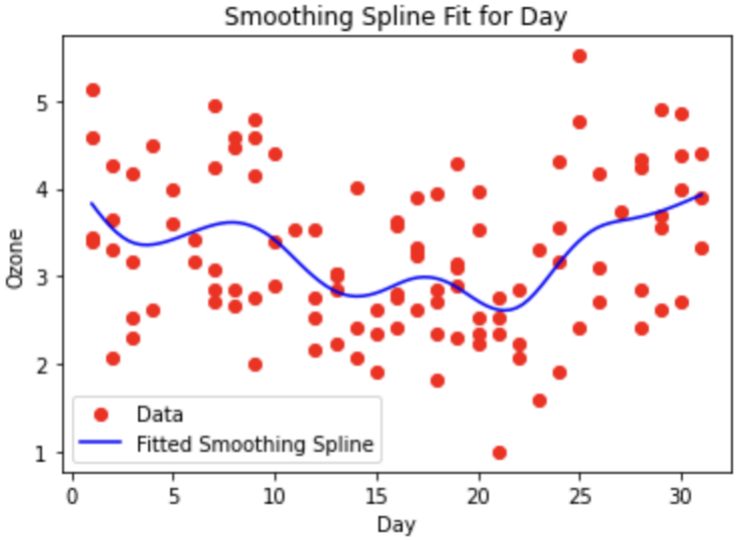
```
for col in X.columns:
    rss = []
    df = []
    for reg in lam_range:
        cv_rss = []
        cv_dff = []
        for i_train, i_val in kf.split(X):
            train_X, val_X = X.iloc[i_train], X.iloc[i_val]
            train_Y, val_Y = Y.iloc[i_train], Y.iloc[i_val]
            gam = LinearGAM(s(0, lam = reg))
            gam.fit(train_X[col].values.reshape((-1, 1)), train_Y)
            val_Y_pred = gam.predict(val_X[col].values.reshape((-1, 1)))
            cv_rss.append(np.mean((val_Y - val_Y_pred)**2))
            cv_dff.append(gam.statistics_['edof'])
        rss.append(np.mean(cv_rss))
        df.append(np.mean(cv_dff))
    opt_reg = lam_range[np.argmin(rss)]
    cv_lam.append(opt_reg)
    cv_df.append(df[np.argmin(rss)])
    gam = LinearGAM(s(0, lam = opt_reg))
    gam.fit(X[col].values.reshape((-1, 1)), Y)
    X_plot = np.linspace(X[col].min(), X[col].max(), 100)
    Y_plot = gam.predict(X_plot.reshape((-1, 1)))
    plt.scatter(X[col], Y, c = 'red', label = 'Data')
    plt.plot(X_plot, Y_plot, color = 'blue', label = 'Fitted Smoothing Spline')
    plt.xlabel(col)
    plt.ylabel('Ozone')
    plt.legend()
    plt.title(f'Smoothing Spline Fit for {col}')
    plt.show()
```


Smoothing Spline Fit for Solar.R
Data
Fitted Smoothing Spline
Ozone
Solar.R
1
2
3
4
5
0
50
100
150
200
250
300
350


Smoothing Spline Fit for Wind
Data
Fitted Smoothing Spline
Ozone
Wind
1
2
3
4
5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Smoothing Spline Fit for Temp
Data
Fitted Smoothing Spline
Ozone
Temp
1
2
3
4
5
55
60
65
70
75
80
85
90
95


Smoothing Spline Fit for Month
Data
Fitted Smoothing Spline
Ozone
Month
1
2
3
4
5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0


Smoothing Spline Fit for Day
Data
Fitted Smoothing Spline
Ozone
Day
1
2
3
4
5
0
5
10
15
20
25
30

```
for i in range(len(X.columns)):
    print(f"Predictor: {X.columns[i]}")
    print(f"Regularization parameter: {cv_lam[i]}")
    print(f"Degrees of freedom: {cv_df[i]}", "\n")
```

```
Predictor: Solar.R
Regularization parameter: 278.2559402207126
Degrees of freedom: 3.159605174475877

Predictor: Wind
Regularization parameter: 278.2559402207126
Degrees of freedom: 3.0096447385191047

Predictor: Temp
Regularization parameter: 0.03593813663804628
Degrees of freedom: 14.840030595892552

Predictor: Month
Regularization parameter: 46.41588833612782
Degrees of freedom: 3.999421915333012

Predictor: Day
Regularization parameter: 1.291549665014884
Degrees of freedom: 8.632075809852346
```

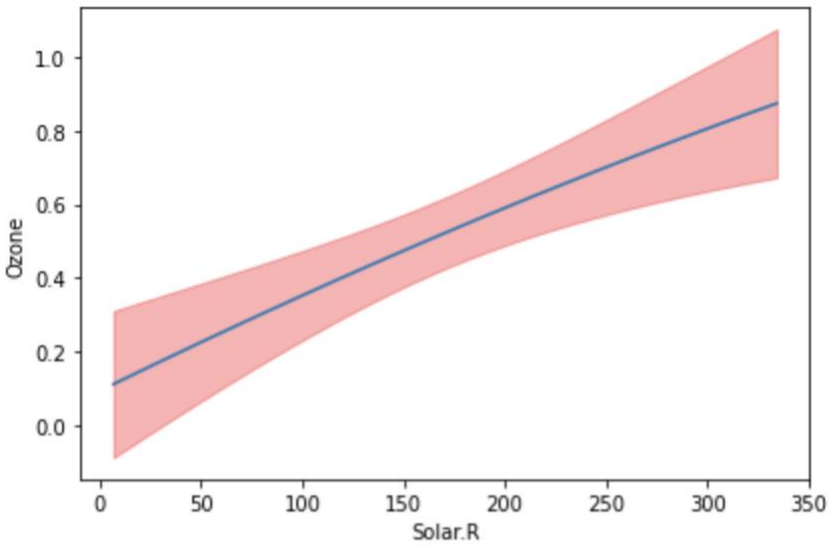
The functions for all predictors represent the trend to an appreciable extent in the data.

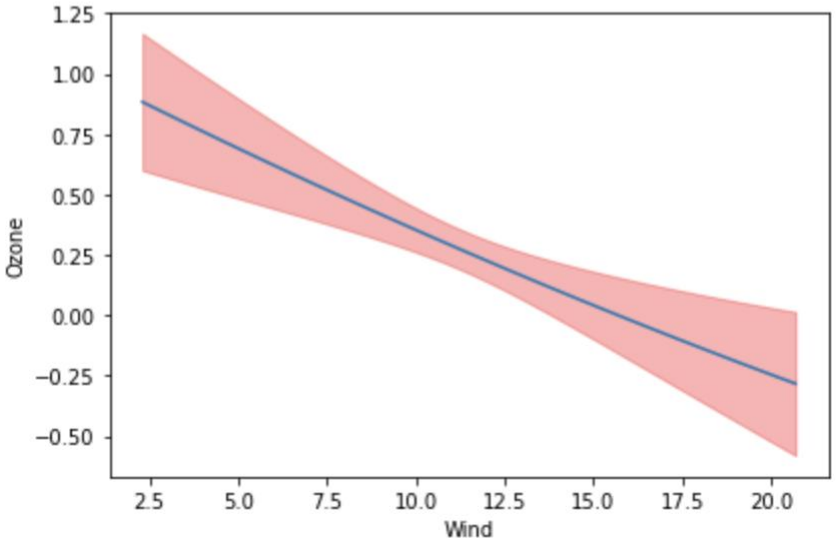
The fit for Temp seems to fit the data well. The same can be said for Wind.

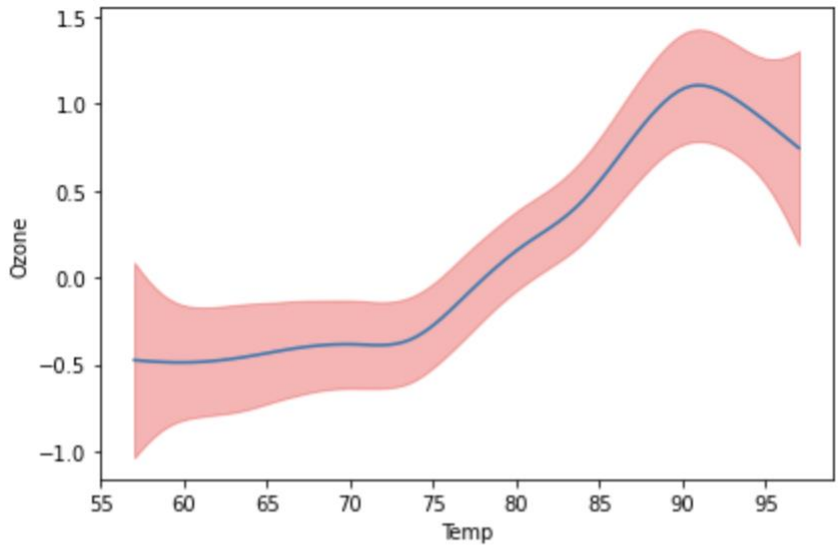
The fit for month, day and Solar.R although representative of the trend in the data, it does not fit the data appropriately.

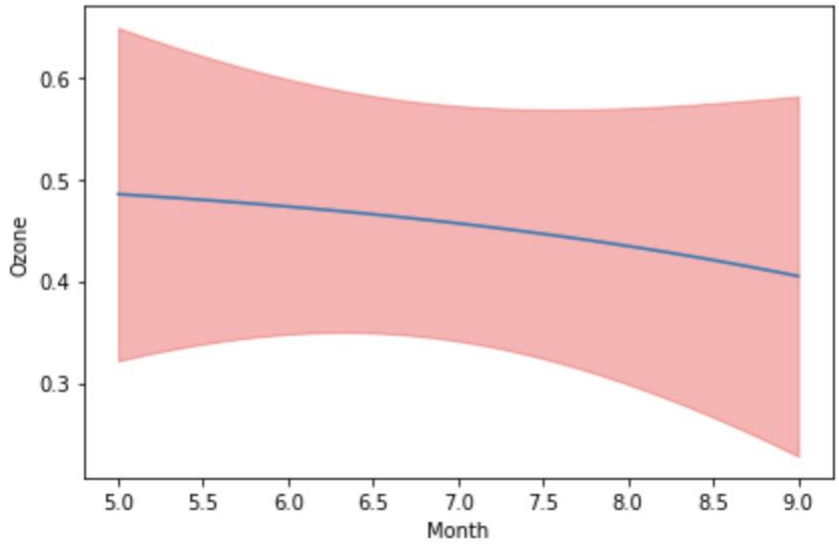
b)

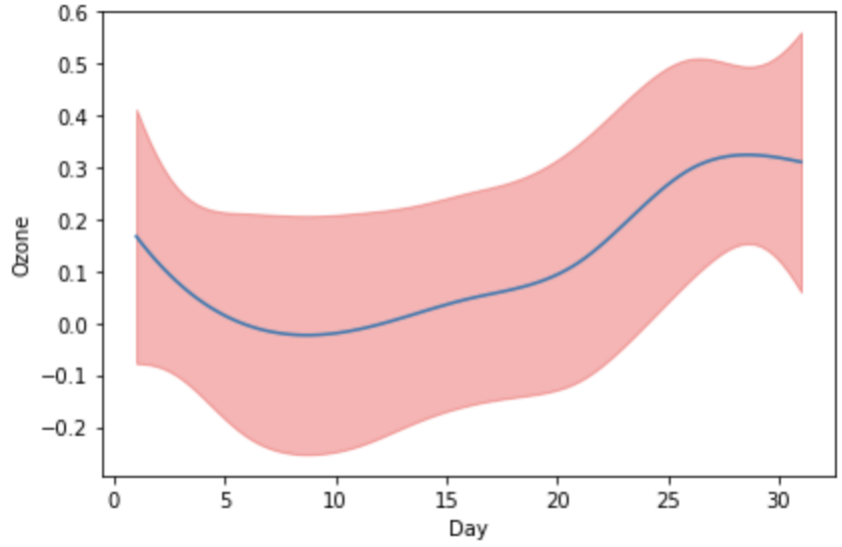
```
Solar_df = 4
Wind_df = 4
Temp_df = 15
Month_df = 4
Day_df = 9

gam_fit = LinearGAM(
s(0, n_splines=Solar_df) +
s(1, n_splines=Wind_df) +
s(2, n_splines=Temp_df) +
s(3, n_splines=Month_df) +
s(4, n_splines=Day_df), fit_intercept=True)

gam_fit.fit(X, Y)

for i in range(1, 6):
     X_gen = gam_fit.generate_X_grid(term=i-1)
     partial_dep, confidence_intervals = gam_model.partial_dependence(term=i-1,X=X_gen, width=0.95)
     plt.plot(X_gen[:, i-1], partial_dep)
     plt.fill_between(X_gen[:, i-1], confidence_intervals[:, 0], confidence_intervals[:, 1], color='red', alpha=0.3)
     plt.xlabel(data.columns[i+1])
     plt.ylabel("Ozone")
     plt.tight_layout()
     plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
Ozone
0
50
100
150
200
250
300
350
Solar.R


1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
Ozone
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Wind


Ozone
Temp
1.5
1.0
0.5
0.0
−0.5
−1.0
55
60
65
70
75
80
85
90
95


Ozone
Month
0.6
0.5
0.4
0.3
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0

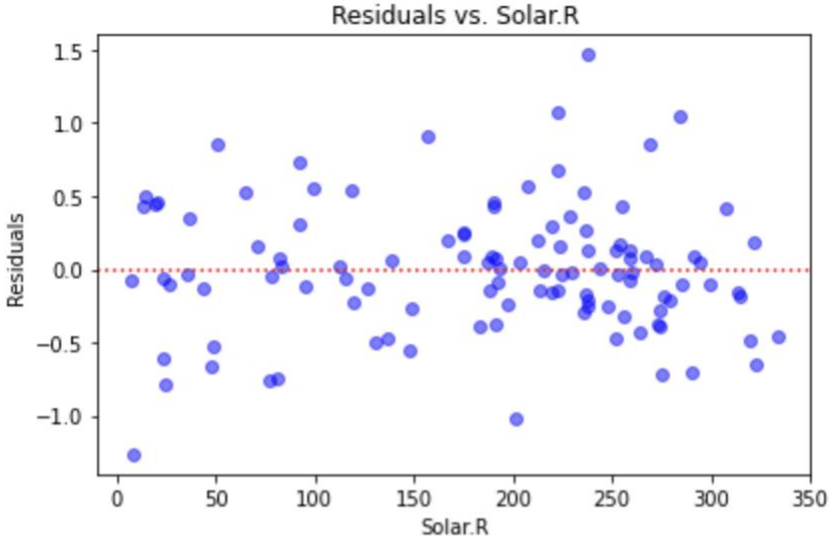

Ozone
Day
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
0
5
10
15
20
25
30

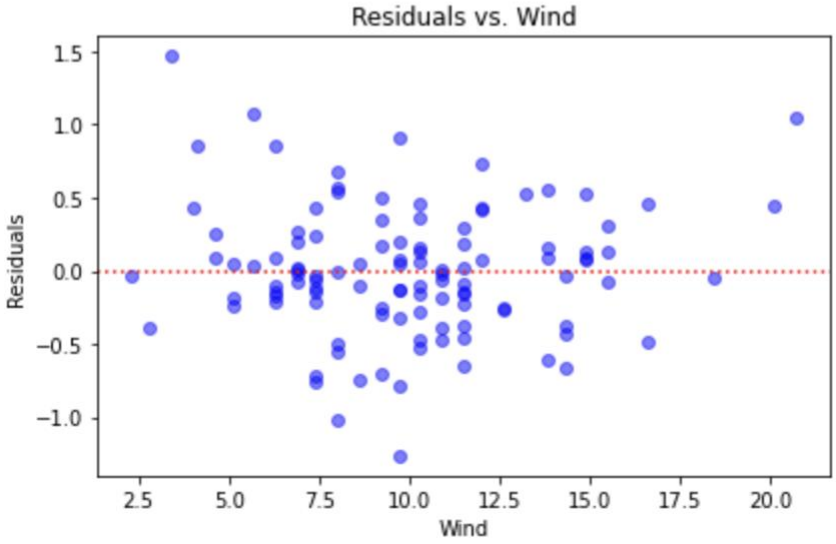
Except for Day, all predictors have wider residuals at the edges and narrower residuals in the middle values of the predictor. Month and Day have much wider ranges as compared to the other predictors.

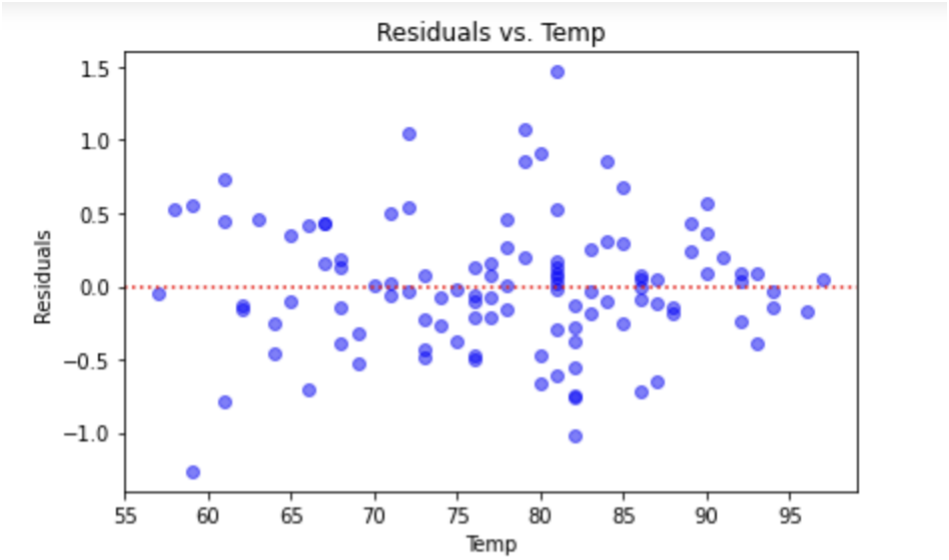
The quality of fit is consistent as in a. However, I would say that Solar.R's smoothing curve also denotes good fit.

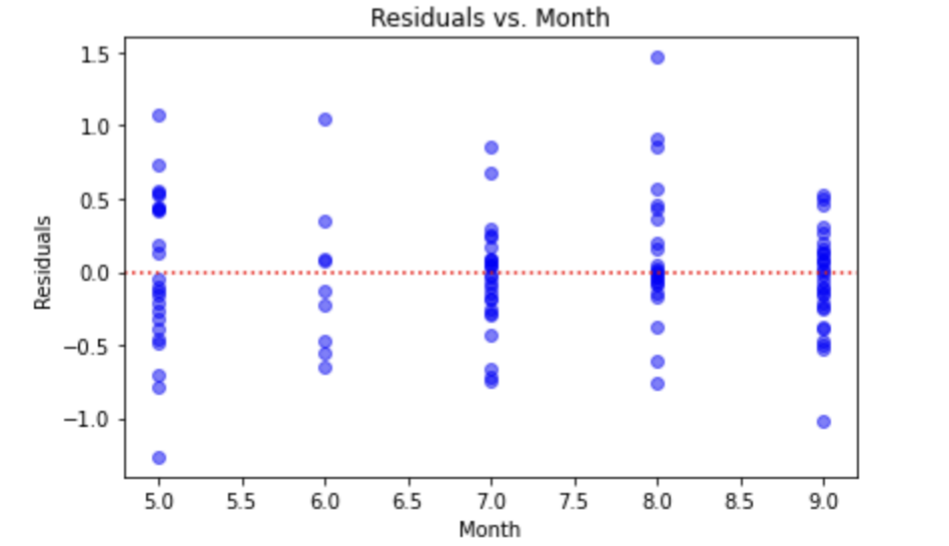
c)

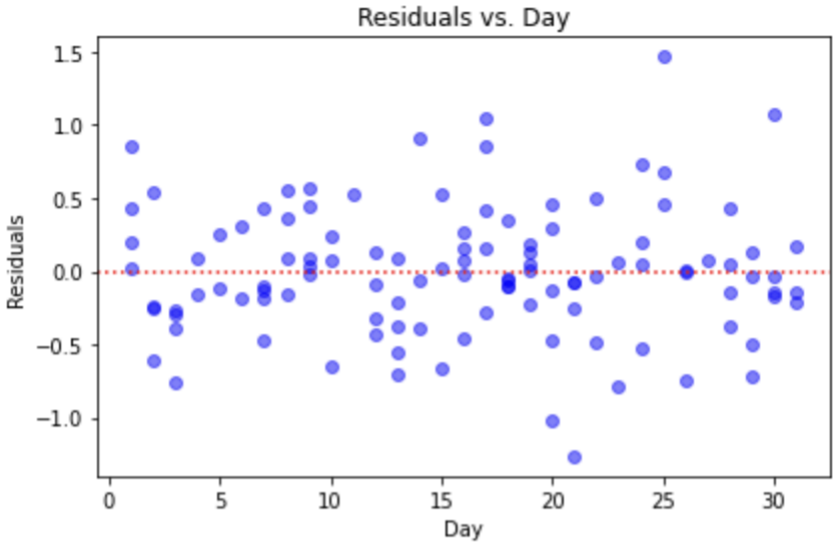
```
residuals = Y - gam_model.predict(X)
for i, predictor in enumerate(["Solar.R", "Wind", "Temp", "Month", "Day"]):
     plt.scatter(data[predictor], residuals, facecolor="blue", alpha=0.5)
     plt.xlabel(predictor)
     plt.ylabel("Residuals")
     plt.title(f"Residuals vs. {predictor}")
     plt.axhline(0, color="red", linestyle=":")
     plt.tight_layout()
     plt.show()
plt.scatter(gam_model.predict(X), residuals, facecolor="blue", alpha=0.5)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs. Fitted Values")
plt.axhline(0, color="red", linestyle="--")
plt.tight_layout()
plt.show()
```

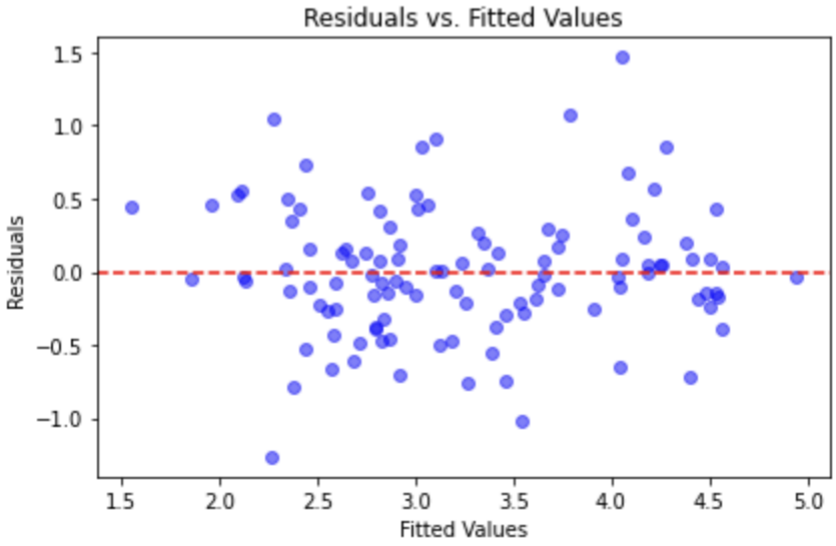

Residuals vs. Solar.R
Residuals
Solar.R
1.5
1.0
0.5
0.0
−0.5
−1.0
0
50
100
150
200
250
300
350


Residuals vs. Wind
Residuals
Wind
1.5
1.0
0.5
0.0
−0.5
−1.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Residuals vs. Temp
Residuals
Temp
1.5
1.0
0.5
0.0
−0.5
−1.0
55
60
65
70
75
80
85
90
95


Residuals vs. Month
Residuals
Month
1.5
1.0
0.5
0.0
−0.5
−1.0
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0





In order to improve model performance, we would require different complexities for each predictor at different regions as well. I notice the following trends in residuals across the predictors which necessitate different complexities in the said conditions:

1. Solar.R has higher residuals at the extremes, lower residuals in the middle.
2. Day has higher residuals for higher values
3. I don’t see any specific trend in the residuals for month
4. Residuals are higher for smaller and moderate values in Temp and Wind.

The residuals vs the fitted values are also relatively higher at the lower and higher values, this suggests the need for varying levels of complexity.