A. Suppose we want to estimate the value of the regression function y^* at some new point x^*, denoted $\hat{f}(x^*)$. Assume for the moment that $f(x)$ is linear, and that y and x have already had their means subtracted, in which case $y_i = \beta x_i + \epsilon_i$.

Return to your least-squares estimator for multiple regression. Show that for the one-predictor case, your prediction $\hat{y}^* = f(x^*) = \hat{\beta}x^*$ may be expressed as a *linear smoother* of the following form:

$$\hat{f}(x^*) = \sum_{i=1}^{n} w(x_i, x^*) y_i$$

for any x^*. Inspect the weighting function you derived. Briefly describe your understanding of how the resulting smoother behaves, compared with the smoother that arises from an alternate form of the weight function $w(x_i, x^*)$:

$$w_K(x_i, x^*) = \begin{cases} 1/K, & x_i \text{ one of the } K \text{ closest sample points to } x^*, \\ 0, & \text{otherwise.} \end{cases}$$

This is referred to as *K-nearest-neighbor smoothing*.

Recall the two following things we need to solve this problem:

$$\hat{\beta} = (X^T X)^{-1} X^T Y,$$
$$X^T Y = \sum_{i=1}^{n} x_i^T y_i,$$

where the first is a known result from multiple linear regression and the second is the expression of a product of matrices as the sum of outer products. With these two, we obtain the following:

$$\begin{aligned} \hat{\beta}x^* &= (X^T X)^{-1} X^T Y x^* \\ &= \left(\sum_{i=1}^{n} x_i^T x_i \right)^{-1} \sum_{i=1}^{n} x_i^T y_i x^* \\ &= \sum_{i=1}^{n} \frac{x_i^T x^*}{\sum_{i=1}^{n} x_i^T x_i} y_i \end{aligned}$$

From the above, we see that

$$w(x_i, x^*) = \frac{x_i^T x^*}{\sum_{i=1}^{n} x_i^T x_i},$$

which smooths the new x^* by scaling it with the ratio $\frac{x_i^T}{x_i^T x_i}$; this effectively takes the proximity of x^* to each x_i into account when summing over all x_i. In comparison, the K-nearest-neighbor smoothing simply scales x^* uniformly across all x_i that is "close" to x_i. We can do this by defining a neighborhood of x_i near x^*, then applying the uniform scale depending on how many x_i belong to this neighborhood.

B. A *kernel function* $K(x)$ is a smooth function satisfying

$$\int_{\mathbb{R}} K(x)dx = 1\,, \quad \int_{\mathbb{R}} xK(x)dx = 0\,, \quad \int_{\mathbb{R}} x^2K(x)dx > 0\,.$$

A very simple example is the uniform kernel,

$$K(x) = \frac{1}{2}I(x) \quad \textbf{where} \quad I(x) = \begin{cases} 1, & |x| \leq 1 \\ 0, & \textbf{otherwise}\,. \end{cases}$$

Another common example is the Gaussian kernel:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\,.$$

Kernels are used as weighting functions for taking local averages. Specifically, define the weighting function

$$w(x_i, x^*) = \frac{1}{h} K\left(\frac{x_i - x^*}{h}\right)\,,$$

where h is the bandwidth. Using this weighting function in a linear smoother is called *kernel regression*. (The weighting function gives the unnormalized weights; you should normalize the weights so that they sum to 1.)

Write your own R function that will fit a kernel smoother for an arbitrary set of x-y pairs and arbitrary choice of (positive real) bandwidth h. You choose the kernel. Set up an R script that will simulate noisy data from some nonlinear function, $y = f(x) + \epsilon$; subtract the sample means from the simulated x and y; and use your function to fit the kernel smoother for some choice of h. Plot the estimated functions for a range of bandwidths wide enough to yield noticeable differences in the qualitative behavior of the prediction functions.

I chose to work with the Gaussian kernel since I have read papers that mention it. For the bandwidth, I used $h = 1, 2, 5, 10$ to yield noticeable differences in the

behavior of the estimated function. Thus, with these bandwidths, the Gaussian kernel, $f_1(x) = 4x^3 + x^2 - 12$, and $x^* \in [-10, 10]$, I obtain Figure 1. Notice that as h increases, our estimated function grows increasingly linear, which is the behavior we would expect to see from our linear smoothing. This behavior can also be seen in Figure 2, where I change the original function to $f_2(x) = x\sin(x)$.

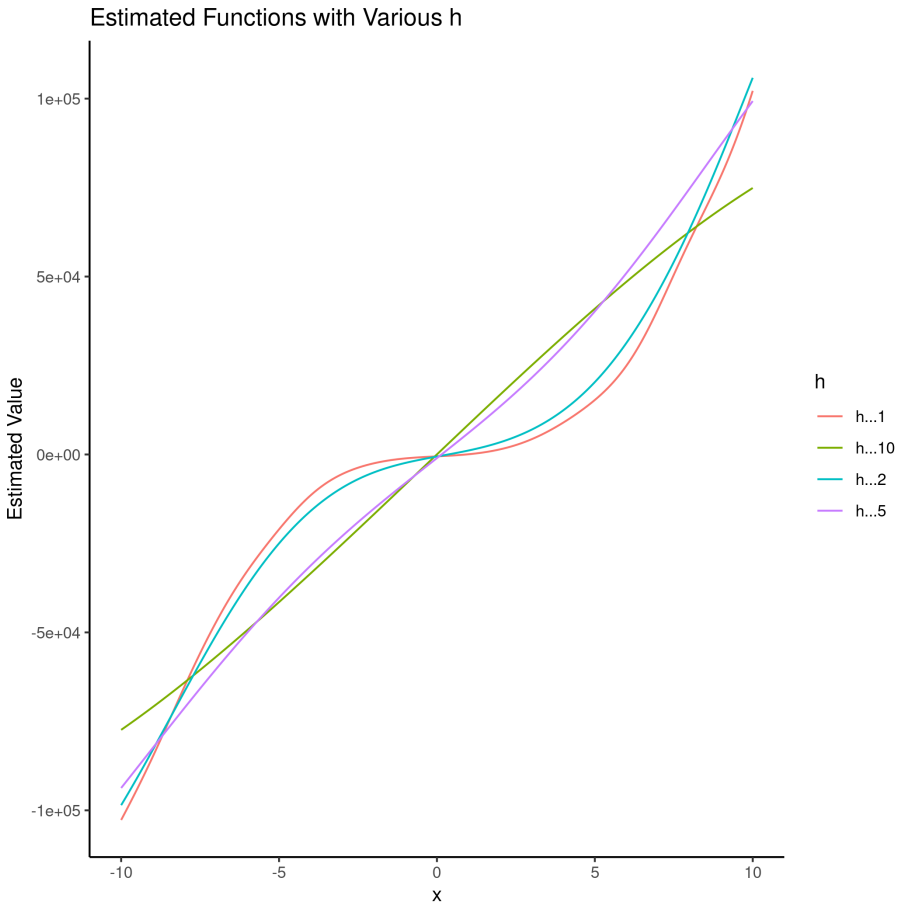

Figure 1: Estimated Functions Under Gaussian Kernel with Various Bandwidths and $f_1(x)$.

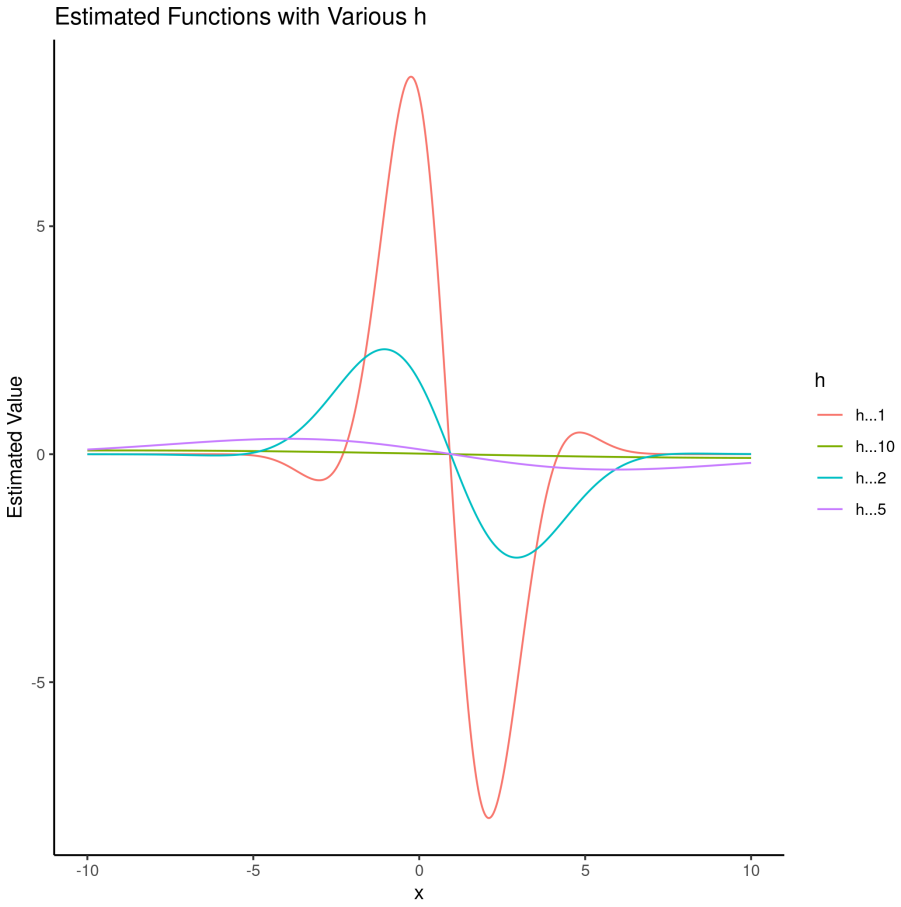

Figure 2: Estimated Functions Under Gaussian Kernel with Various Bandwidths and $f_2(x)$.

A. Presumably a good choice of h would be one that led to smaller predictive errors on fresh data. Write a function or script that will: (1) accept an old ("training") data set and a new ("testing") data set as inputs; (2) fit the kernel-regression estimator to the training data for specified choices of h; and (3) return the estimated functions and the realized prediction error on the testing data for each value of h. This should involve a fairly straightforward "wrapper" of the function you've already written.

Using $f_2(x)$ and $h = 1, 2, 5, 10$ from the previous section, I obtain Figure 3, where the black dots are data in the testing data set. Visually, we can see that the testing data best matches with the kernel-regression estimator when $h = 2$. This conclusion is *validated* when looking at the mean square prediction errors (MSPE) for $h = 1, 2, 5, 10$ which were $MSPE_h = 2871.7169, 504.5653, 777.8502,$ and 791.7469, respectively. Since we want to choose the estimated function with the lowest MSPE, our choice for bandwidth is surely $h = 2$.

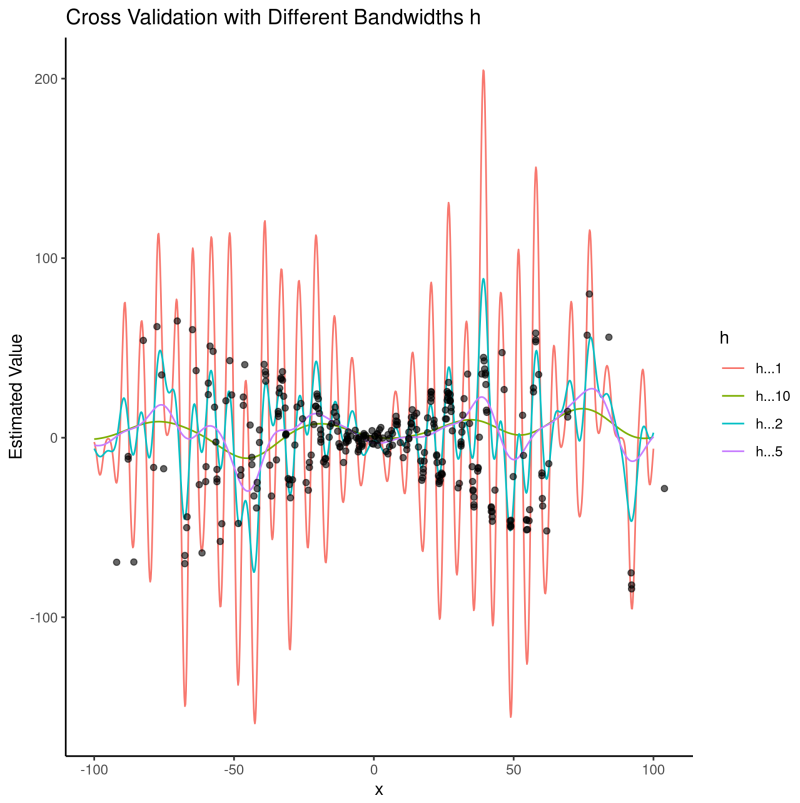

Figure 3: Cross Validation with Different Bandwidths h.

B. Imagine a conceptual two-by-two table for the unknown, true state of affairs. The rows of the table are "wiggly function" and "smooth function," and the columns are "highly noisy observations" and "not so noisy observations." Simulate one data set (say, 500 points) for each of the four cells of this table, where the x's take values in the unit interval. Then split each data set into training and testing subsets. You choose the functions. Apply your method to each case, using the testing data to select a bandwidth parameter. Choose the estimate that minimizes the average squared error in prediction, which estimates the mean-squared error:

$$L_n(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n^*}(y_i^* - \hat{y}_i^*)^2 \,,$$

where (y_i^*, x_i^*) are the points in the test set, and $\hat{y}_i^*$ is your predicted value arising from the model you fit using only the training data. Does your out-of-sample predictive validation method lead to reasonable choices of h for each case?

I used the following functions for this problem:

	High Noise	**Low Noise**
Wiggly Function	$f(x) = \cos(10x) + N(0, 0.8)$	$f(x) = \cos(10x) + N(0, 0.25)$
Smooth Function	$f(x) = x^{5/3} + N(0, 0.8)$	$f(x) = x^{5/3} + N(0, 0.25)$

Once again, I used the Gaussian kernel with bandwidths $h = 0.1, 0.2, 0.5, 1$. I split my 500 observations into 375 for training and 125 for testing. Note that, in the unit interval, the "high noise" significantly scales the observations, which is evident in the left column of Figure 4.

Under these four scenarios, the optimal h for the "wiggly" function and the "smooth" function under low noise was the lowest bandwidth $h = 0.1$. This makes sense since the wiggly function is best "matched" by a linear smoother that does the least amount of smoothing. Visually, we can see in the top row of Figure 4 that the less we smooth, the closer the testing data is to our estimated function. Meanwhile, when starting with a relatively smoother function such as in the bottom row, our optimal smoother may come in the form of a small value for h. In fact, for the "smooth" function with high noise, we find that $h = 0.2$ is the optimal bandwidth under seed *702*. Therefore, it seems that our out-of-sample predictive validation does lead to reasonable choices of h for each case since we have a decent amount of data.

	Wiggly/High	Wiggly/Low	Smooth/High	Smooth/Low
h=0.1	**0.638**	**0.139**	0.616	**0.066**
h=0.2	0.861	0.400	**0.614**	0.074
h=0.5	0.975	0.532	0.641	0.114
h=1	0.979	0.536	0.665	0.139

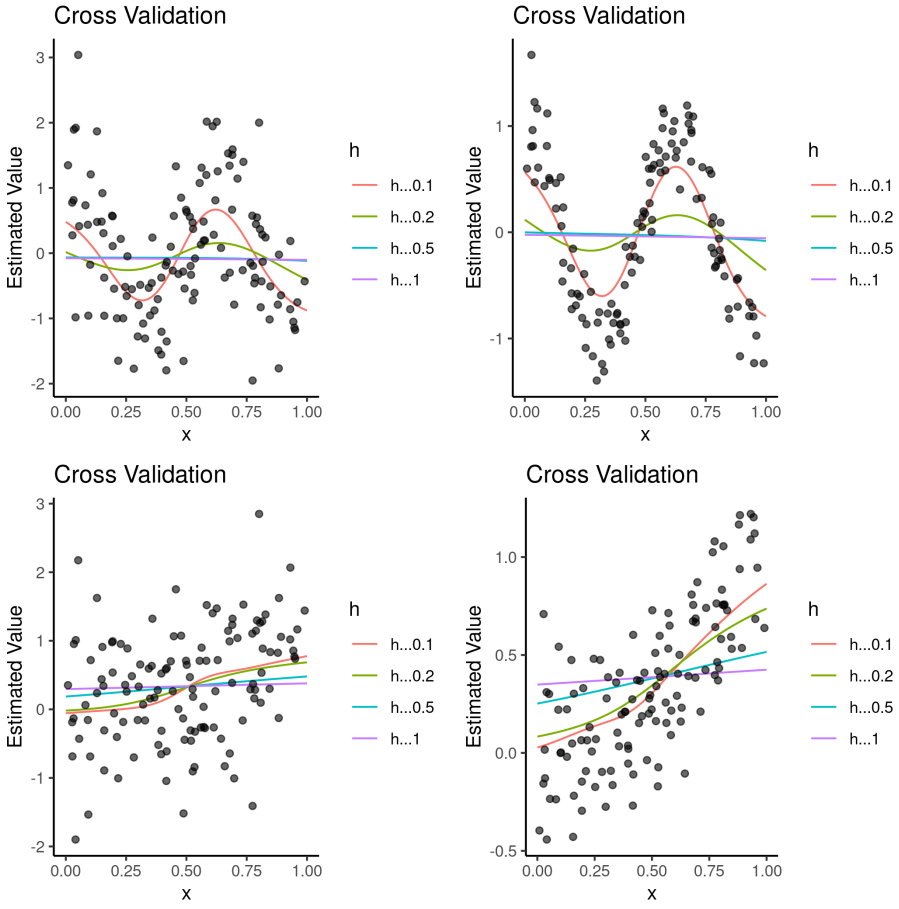

Figure 4: Comparing Optimal Bandwidths Across Different Scenarios in Out-of-Sample Cross Validation.

C. Use the leave-one-out lemma to revisit the examples you simulated in Part B, using leave-one-out cross validation to select h in each case. Because of the leave-one-out lemma, you won't need to actually refit the model N times!

Under linear smoothers, computation of LOOCV simplifies greatly to

$$\text{LOOCV} = \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - H_{ii}} \right)^2$$

This is convenient, since we can easily obtain some metric to determine the optimal bandwidth without having to split our data set into two. Note that with the out-of-sample approach, we are only able to use 75% of the data to fit an estimated function, whereas with this method, we are able to use all of the data!

In Figure 5, we can see the same smoothed estimated functions from Figure 4 with all of the data points overlaid. Note that the estimated functions may differ slightly in shape because we were able to use all of the data to fit them. Using the LOOCV criterion, the optimal bandwidth for both the "wiggly" and "smooth" function under both high and low noise was $h = 0.1$, indicating that when we use all of the data to obtain our estimated function in a variety of conditions, it's most optimal not to smooth the data given a sufficient amount of data.

	Wiggly/High	**Wiggly/Low**	**Smooth/High**	**Smooth/Low**
h=0.1	**0.684**	**0.150**	**0.594**	**0.068**
h=0.2	0.988	0.426	0.601	0.074
h=0.5	1.139	0.564	0.644	0.114
h=1	1.143	0.569	0.672	0.139

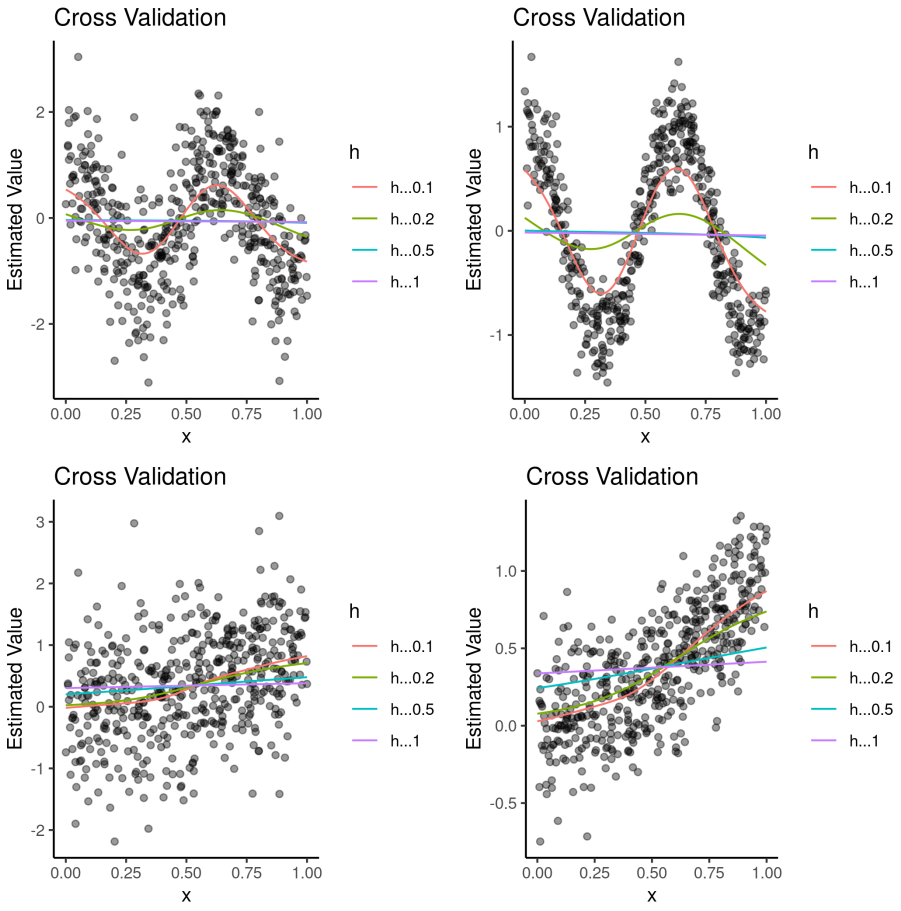

Figure 5: Comparing Optimal Bandwidths Across Different Scenarios in Leave-One-Out Cross Validation.