Homework 5

Advanced Methods for Data Analysis (36-402)

Due Friday February 25, 2022 *at 3:00 pm ET*

Solutions – not to be posted online or shared, even after the end of the semester.

You should **always show all your work** and submit both a writeup and *R* code.

- Assignments must be submitted through Gradescope as a PDF. Follow the instructions here: `https://www.cmu.edu/teaching/gradescope/`
- Gradescope will ask you to mark which parts of your submission correspond to each homework problem. This is mandatory; if you do not, grading will be slowed down, and your assignment will be penalized.
- Make sure your work is legible in Gradescope. You may not receive credit for work the TAs cannot read. **Note:** If you submit a PDF with pages much larger than 8.5×11", they will be blurry and unreadable in Gradescope.
- For questions involving R code, we strongly recommend using R Markdown. The relevant code should be included with each question, rather than in an appendix. A template Rmd file is provided on Canvas.

1. **Housing Data Revisited Redux.** In Homework 4, you worked with the housing data you used in Homework 2, building additional models and comparing their performance. Now we'll return to those models and compare them using cross-validation. In the problems below, the models we refer to are the ones defined in Homeworks 2 and 4.

   (a) It might be naïve to assume, as we did in Homework 4, that `Longitude` affects housing prices linearly over the long distance from California to Pennsylvania. For example, if house prices in Philadelphia (`Longitude` about -75 and `Latitude` about 40) are higher than house prices in Pittsburgh (`Longitude` about -80 and `Latitude` about 40.5), a linear contribution of `Longitude` that picks up the difference in price will be making a huge negative contribution to prices in California (`Longitude` about -120). But prices in California are also higher than in Pittsburgh. To avoid such considerations, a nonparametric model in which the conditional mean of `Median_house_value` given the predictors is allowed to be a nonlinear function of the predictors. Fit two additional models:

   Model 5: A kernel regression of `Median_house_value` on `Median_household_income` and `Mean_household_income`.

   Model 6: A kernel regression of `Median_house_value` on `Median_household_income`, `Mean_household_income`, `Latitude`, and `Longitude`.

   Once again, you will need to load the `np` package with `library(np)`. You need a different bandwidth for each predictor. For bandwidths, use the sample standard deviations of each predictor divided by $n^{1/5}$, where n is the number of data points used to fit the model.[1] For example, in

[1] The choices of bandwidths in these exercises was made based on some theory that will appear in a later lecture. The specific choices are not "optimal," but they are popular rule-of-thumb choices.

Model 6, the predictors in the order stated are variables 5, 6, 2, and 3 in the supplied data files. So the bandwidths can be supplied as the argument

`bws <- apply(housetrain[ ,c(5,6,2,3)], 2, sd) / n^(0.2)`

to the `npreg` command, where n is the size of the training data, (5303 in this case but *different* in part (b).)

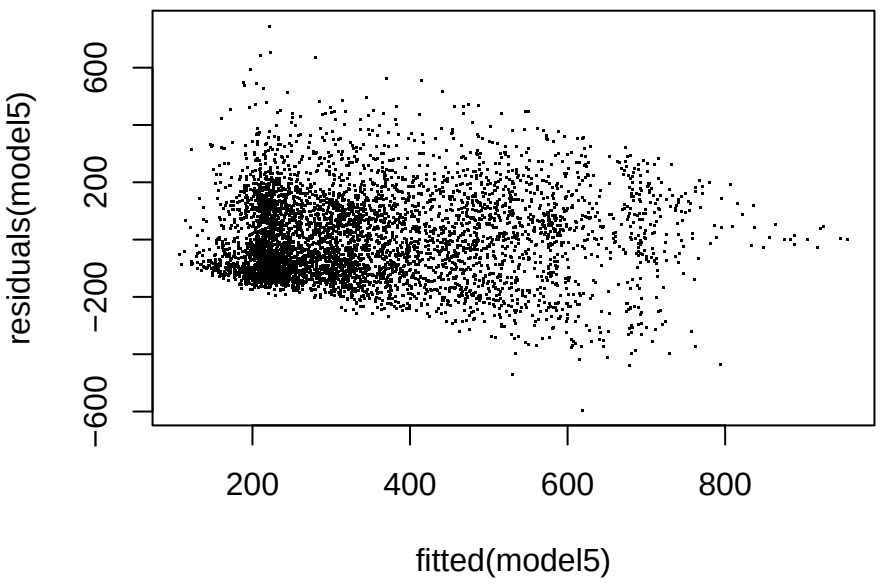
Plot residuals against the fitted values and predictors, and comment on any patterns you see. The residuals will be in the fitted object in an item called `resid` if you add the argument `residuals=TRUE` when you call `npreg`. The fitted values can be obtained by calling the `fitted` method on the object returned by `npreg`.

Solution:

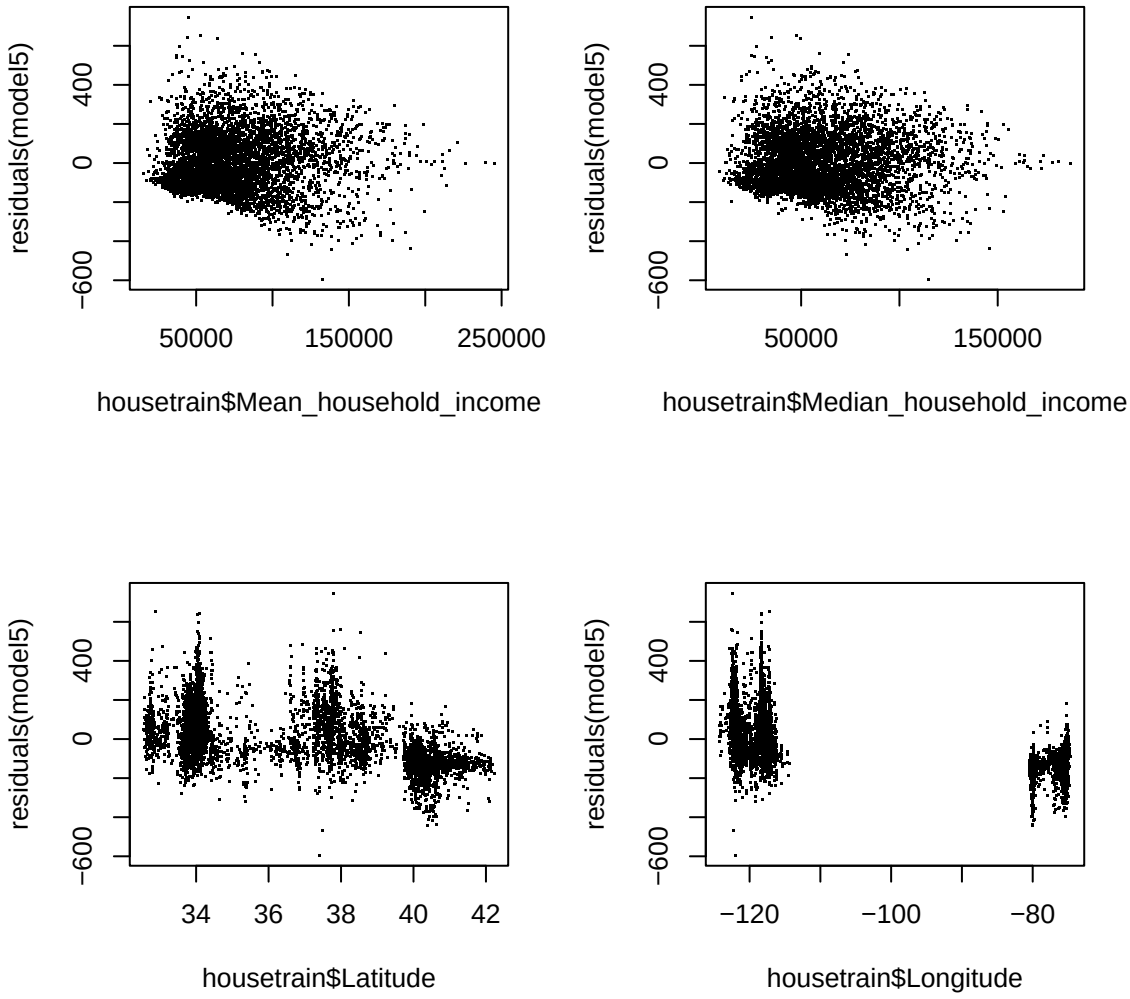
```
# Load the np library, but make it shut up
suppressMessages(library(np))

# Load the data
housetrain <- read.csv("housetrain.csv")
housetest <- read.csv("housetest.csv")
housedata <- rbind(housetrain, housetest)

# Compute the bandwidths
bw <- apply(housetrain[,c(5,6,2,3)], 2, sd) / nrow(housetrain)^(0.2)

# Model 5
model5 <- npreg(Median_house_value ~ Mean_household_income +
                Median_household_income, bws = bw[c(1,2)],
                data = housetrain, residuals=TRUE)
```

Now we plot the residuals:

```
plot(fitted(model5), residuals(model5), pch=".")
```




```
par(mfrow=c(2,2))
plot(housetrain$Mean_household_income, residuals(model5), pch=".")
plot(housetrain$Median_household_income, residuals(model5), pch=".")
plot(housetrain$Latitude, residuals(model5), pch=".")
plot(housetrain$Longitude, residuals(model5), pch=".")
```



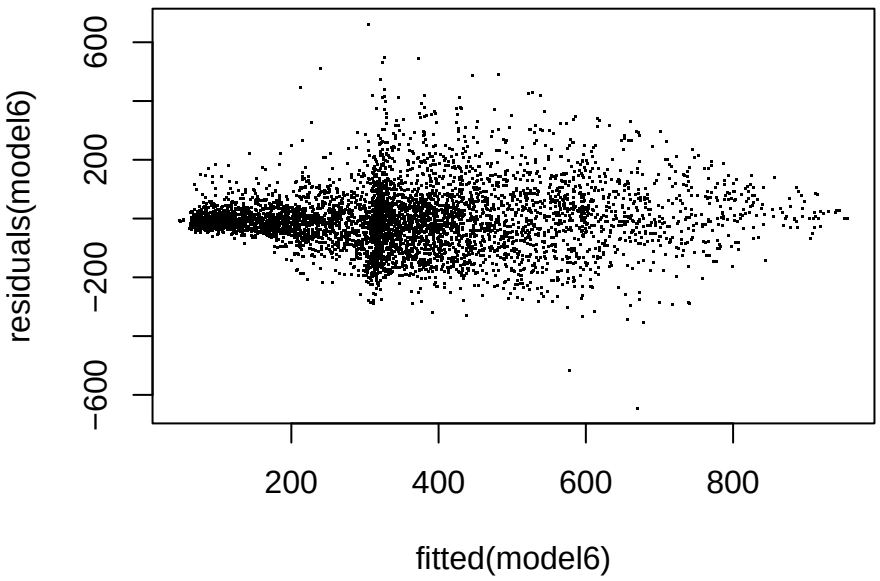
The Model 5 plots look a lot like the Model 3 plots, especially reproducing the facts that the residuals from different locations have different means and variances.

Now let's fit Model 6:

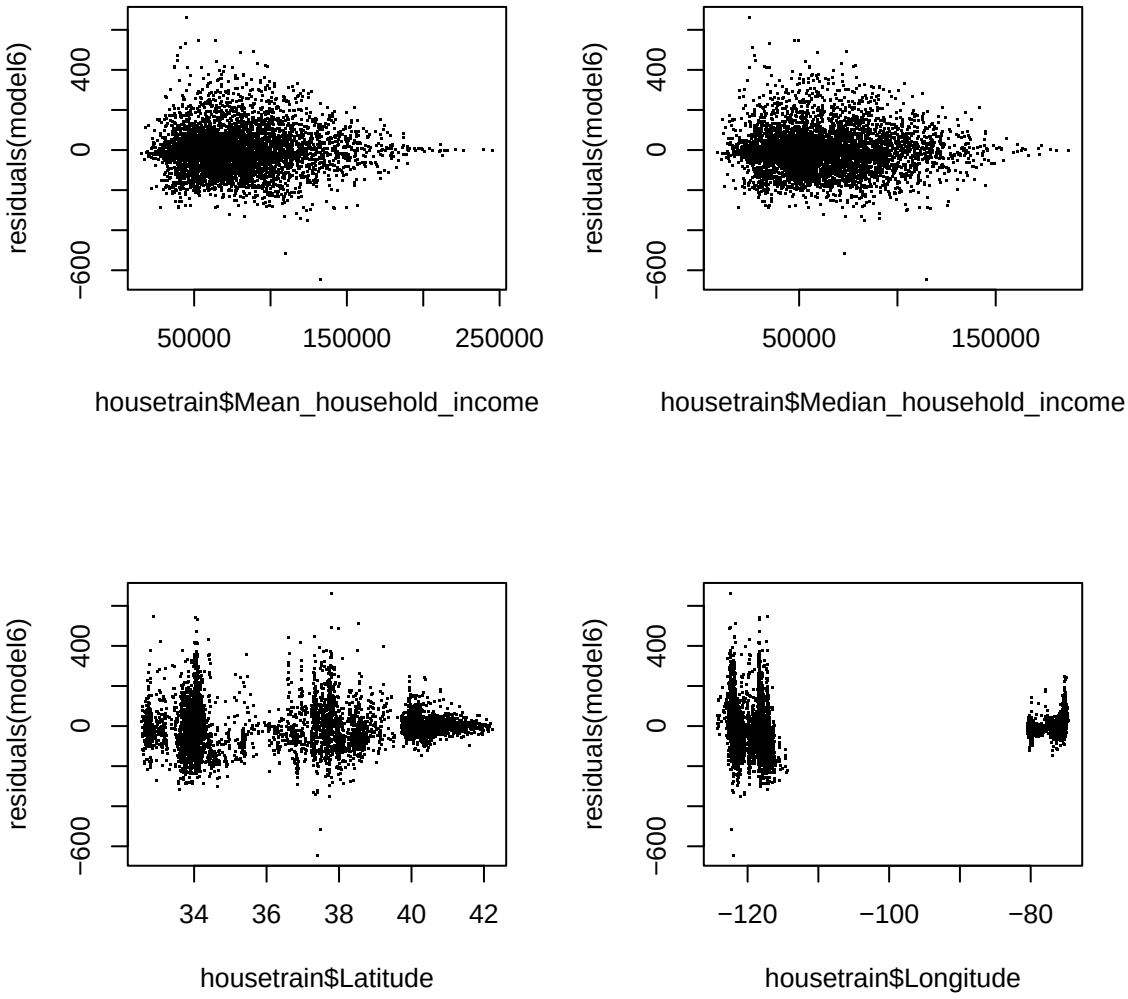
```
model6 <- npreg(Median_house_value ~ Mean_household_income +
                  Median_household_income + Latitude + Longitude,
                  data = housetrain, bws = bw, residuals = TRUE)
```

The residuals:

```
plot(fitted(model6), residuals(model6), pch=".")
```




```
par(mfrow=c(2,2))

plot(housetrain$Mean_household_income, residuals(model6), pch=".")
plot(housetrain$Median_household_income, residuals(model6), pch=".")
plot(housetrain$Latitude, residuals(model6), pch=".")
plot(housetrain$Longitude, residuals(model6), pch=".")
```



The Model 6 plots look better in that the residuals from the different locations are centered closer to zero, and the plot of residuals against fitted values is the closest to a random scatter amongst all four models. The residuals from different locations still have different variances.

(b) Next we will perform 10-fold cross-validation to choose between Models 3–6 as predictors. Combine the original training and test data sets into one data set of size $n = 10605$ by using `housedata <- rbind(housetrain, housetest)`, where `housetrain` and `housetest` are respectively the names of your training and test data sets. For each fold of cross-validation, create the test set by randomly assigning each row to a fold k for $k = 1, \ldots, 10$. The test folds and corresponding training portions can be created in a manner similar to Demo 3.1:

```
samp <- sample(rep(1:10, length.out = nrow(housedata)), replace = FALSE)
for (k in 1:10) {
  testd <- housedata[samp == k, ]
  traind <- housedata[!(samp == k), ]
```

The `npreg` function will compute predictions while fitting the model if you run it with a command like

```
model5 <- npreg(Median_house_value ~ Median_household_income +
```

```
                Mean_household_income,
                data = traind, newdata = testd,
                bws = apply(traind[, c(5,6)], 2, sd) / n^(0.2))
```

This time n is the size of the `traind` data set, which is different from the n in part (a). The predictions will then be in `model5$mean`. For each fold k and each model $m = 3, 4, 5, 6$, report the resulting values of the average squared prediction error within fold k for Model m. You can write a loop to do all the calculations and store the results in a 10×4 matrix.

Solution: Let's create the data and make a matrix for the results, giving it column names so we can keep track of everything.

```
samp <- sample(rep(1:10, length.out = nrow(housedata)), replace = FALSE)

prederr <- matrix(NA, nrow = 10, ncol = 4)
colnames(prederr) <- c("Model 3", "Model 4", "Model 5", "Model 6")
```

Next, we need to loop through the five folds, fitting the four models once with each fold, and computing the sum of squares of the prediction errors. For each fold, we need to recompute the bandwidths for the kernel regressions using the new training data.

```
for (k in 1:10) {
  testd <- housedata[samp == k, ]
  traind <- housedata[!(samp == k), ]
  model <- lm(Median_house_value ~ Mean_household_income +
              Median_household_income, data = traind)

  prederr[k,1] <- mean((predict(model, newdata = testd) -
                        testd$Median_house_value)^2)

  model <- lm(Median_house_value ~ Mean_household_income +
              Median_household_income + Latitude + Longitude,
              data=traind)
  prederr[k,2] = mean((predict(model, newdata = testd) -
                       testd$Median_house_value)^2)

  bw <- apply(traind[,c(5,6,2,3)], 2, sd) / nrow(traind)^(0.2)

  model <- npreg(Median_house_value ~ Median_household_income +
                 Mean_household_income, data = traind, newdata = testd,
                 bws = bw[c(1,2)])
  prederr[k,3] <- mean((model$mean - testd$Median_house_value)^2)

  model <- npreg(Median_house_value ~ Median_household_income +
                 Mean_household_income + Latitude + Longitude,
                 data = traind, newdata = testd, bws = bw)
  prederr[k,4] <- mean((model$mean - testd$Median_house_value)^2)
}
```

All of the action is in the summary statistics that we compute for part (f).

As a note, one could have used the `predict` function to compute the predictions from models 5 and 6 instead of including the parameter `newdata = testd` in the `npreg` function. To use `predict`, one would have used `mean((predict(model, newdata = testd) -`

`testd$Median_house_value)^2)` to compute the `prederr` values for models 5 and 6.

(c) Compute the average of the ten cross-validation error values for each model along with the corresponding standard error as defined in lecture. Comment on which model or models are clearly better than the others. How much better is the best model than the second best compared to the standard errors? Comment on what this says about how good the models are compared to each other.

Solution:

```
prederr
##         Model 3  Model 4  Model 5  Model 6
##  [1,] 24868.47 14639.26 24018.12 12098.56
##  [2,] 22243.10 13886.78 22359.36 12319.43
##  [3,] 22366.45 13967.45 21856.41 11724.31
##  [4,] 22727.73 13496.64 22726.98 11836.60
##  [5,] 22845.94 14427.54 22878.44 12384.32
##  [6,] 22910.99 14152.05 22384.06 12206.19
##  [7,] 22782.41 14972.03 22584.26 12542.05
##  [8,] 22537.69 13760.17 22121.68 12175.92
##  [9,] 22535.06 14847.51 21411.31 12397.09
## [10,] 22859.77 13893.25 22396.66 11747.62
apply(prederr, 2, mean)
##  Model 3  Model 4  Model 5  Model 6
## 22867.76 14204.27 22473.73 12143.21
apply(prederr, 2, sd) / sqrt(10)
##   Model 3   Model 4   Model 5   Model 6
## 233.17333 156.22601 218.22757  91.02635
```

We see that the two models (4 and 6) that make use of location have much smaller cross-validation error than the other two, many times both of the standard errors. The other two models are closer, but Model 6 has cross-validation error that is about 2000 smaller than Model 4, while the standard errors are only a few hundred each. It looks like Model 6 predicts better than the others and its residual plots are better as well. There still seem to be different variances for different locations. Perhaps a weighted kernel regression fit could deal with that.

2. **Optimism and Effective Degrees of Freedom.** In lecture (Lectures 1 and 3), we presented the following result about the relationship between in-sample prediction risk and expected training error:

Let the data consist of $(x_1, Y_1), \ldots, (x_n, Y_n)$, a set of n (predictor, response) pairs with

(i) $Y_i = r(x_i) + \varepsilon_i$ for each i,

(ii) $\varepsilon_1, \ldots, \varepsilon_n$ uncorrelated random variables with mean 0 and common variance σ^2,

(iii) $x_1, \ldots, x_n$ all non-random known values, and

(iv) $r(\cdot)$ a non-random unknown function.

Let $\hat{r}(\cdot)$ be an estimator of $r(\cdot)$ that is determined from the data. Let Y_i' have the same distribution as Y_i for each i with $Y_1', \ldots, Y_n'$ independent of $Y_1, \ldots, Y_n$. Then

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\{Y_i' - \hat{r}(x_i)\}^2\right] - \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\{Y_i - \hat{r}(x_i)\}^2\right] = \frac{2}{n}\sum_{i=1}^{n}\mathrm{Cov}(Y_i, \hat{r}(x_i)). \quad (1)$$

Prove that (1) is true. You may want to review some of the lectures where prediction risk was discussed, especially the associated hand-written notes.

Solution: We are asked to prove that

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\{Y_i' - \hat{r}(x_i)\}^2\right] - \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\{Y_i - \hat{r}(x_i)\}^2\right] = \frac{2}{n}\sum_{i=1}^{n}\mathrm{Cov}(Y_i, \hat{r}(x_i)).$$

It suffices to prove that, for each i,

$$\mathbb{E}\left[\{Y_i' - \hat{r}(x_i)\}^2 - \{Y_i - \hat{r}(x_i)\}^2\right] = 2\,\mathrm{Cov}(Y_i, \hat{r}(x_i)). \tag{2}$$

As was done in the earlier lectures, we add and subtract the means of the random variables that appear inside of each of the squared terms $\{Y_i' - \hat{r}(x_i)\}^2$ and $\{Y_i - \hat{r}(x_i)\}^2$ so that we can expand the squares into terms whose means are easier to deal with, including several whose means are the same.

$$\begin{aligned}
(Y_i' - \hat{r}(x_i))^2 &= \big(Y_i' - r(x_i) + r(x_i) - \hat{r}(x_i)\big)^2 \\
&= (Y_i' - r(x_i))^2 + 2(Y_i' - r(x_i))(r(x_i) - \hat{r}(x_i)) + (r(x_i) - \hat{r}(x_i))^2 \\
(Y_i - \hat{r}(x_i))^2 &= (Y_i - r(x_i) + r(x_i) - \hat{r}(x_i))^2 \\
&= (Y_i - r(x_i))^2 + 2(Y_i - r(x_i))(r(x_i) - \hat{r}(x_i)) + (r(x_i) - \hat{r}(x_i))^2
\end{aligned}$$

It follows that

$$\begin{aligned}
(Y_i' - \hat{r}(x_i))^2 - (Y_i - \hat{r}(x_i))^2 = \underbrace{(Y_i' - r(x_i))^2}_{1} - \underbrace{(Y_i - r(x_i))^2}_{2} + \\
\underbrace{2(Y_i' - r(x_i))(r(x_i) - \hat{r}(x_i))}_{3} - \underbrace{2(Y_i - r(x_i))(r(x_i) - \hat{r}(x_i))}_{4}.
\end{aligned} \tag{3}$$

To prove (2), we need to show that the expectations of these four terms add up to $2\,\mathrm{Cov}(Y_i, \hat{r}(x_i))$. Let's go through them each in turn.

For terms 1 and 2, note that the only random variables are Y_i and Y_i', and these are identically distributed. Hence their expectations are equal and cancel out. Alternately, you could notice that

$$\begin{aligned}
\mathbb{E}[(Y_i' - r(x_i))^2] &= \sigma^2 \\
\mathbb{E}[(Y_i - r(x_i))^2] &= \sigma^2,
\end{aligned}$$

because $\mathrm{Var}(Y_i - r(x_i)) = \mathrm{Var}(\varepsilon) = \sigma^2 = \mathbb{E}[(Y_i - r(x_i))^2] - \mathbb{E}[Y_i - r(x_i)]^2$ by definition, and $\mathbb{E}[Y_i - r(x_i)] = 0$ (because $r(x_i)$ is the mean of Y_i by definition). The same reasoning applies to $\mathrm{Var}(Y_i' - r(x_i))$. Hence terms 1 and 2 cancel out when we take the expectation of everything.

When we take the expectation of term 3, notice that the two factors are independent. (The only random parts are $\hat{r}(x_i)$, trained on the training data, and Y_i', a new and *independent* observation.) Hence we can break the expectation into the product:

$$\mathbb{E}\left[2(Y_i' - r(x_i))(r(x_i) - \hat{r}(x_i))\right] = 2\,\mathbb{E}[Y_i' - r(x_i)]\,\mathbb{E}[r(x_i) - \hat{r}(x_i)].$$

And because $\mathbb{E}[Y'|X = x_i] = r(x_i)$ by definition, the left-hand expectation is zero, and hence term 3 is zero.

That leaves term 4 as the only nonzero term in (3). We need to get $\text{Cov}(Y_i, \hat{r}(x_i))$ from it, so let's start with the covariance and apply the definition that $\text{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y]$:

$$\begin{aligned}\text{Cov}(Y_i, \hat{r}(x_i)) &= \mathbb{E}[Y_i \hat{r}(x_i)] - \mathbb{E}[Y_i]\,\mathbb{E}[\hat{r}(x_i)] \\ &= \mathbb{E}[Y_i \hat{r}(x_i)] - r(x_i)\,\mathbb{E}[\hat{r}(x_i)].\end{aligned}$$

Now let's take the expectation of term 4 and expand it:

$$\begin{aligned}-2\,\mathbb{E}\left[(Y_i - r(x_i))(r(x_i) - \hat{r}(x_i))\right] &= -2\,\mathbb{E}\left[Y_i r(x_i) - Y_i \hat{r}(x_i) - r(x_i)^2 + r(x_i)\hat{r}(x_i)\right] \\ &= -2\left(r(x_i)\,\mathbb{E}[Y_i] - \mathbb{E}[Y_i \hat{r}(x_i)] - r(x_i)^2 + r(x_i)\,\mathbb{E}[\hat{r}(x_i)]\right) \\ &= -2\left(r(x_i)^2 - \mathbb{E}[Y_i \hat{r}(x_i)] - r(x_i)^2 + r(x_i)\,\mathbb{E}[\hat{r}(x_i)]\right) \\ &= 2(\mathbb{E}[Y_i \hat{r}(x_i)] - r(x_i)\,\mathbb{E}[\hat{r}(x_i)]).\end{aligned}$$

This is the same as 2 times the covariance above, and hence we have achieved the desired result.

3. **The Parametric Bootstrap in Regression.** The file `parametric-bootstrap.csv` contains 100 observations of two variables, X and Y. These variables are linearly related, but one of the simple linear regression assumptions is not met. (See Shalizi section 2.3 for more on the simple linear regression assumptions.) Specifically, while the residuals are normally distributed, their variance is not constant in x:

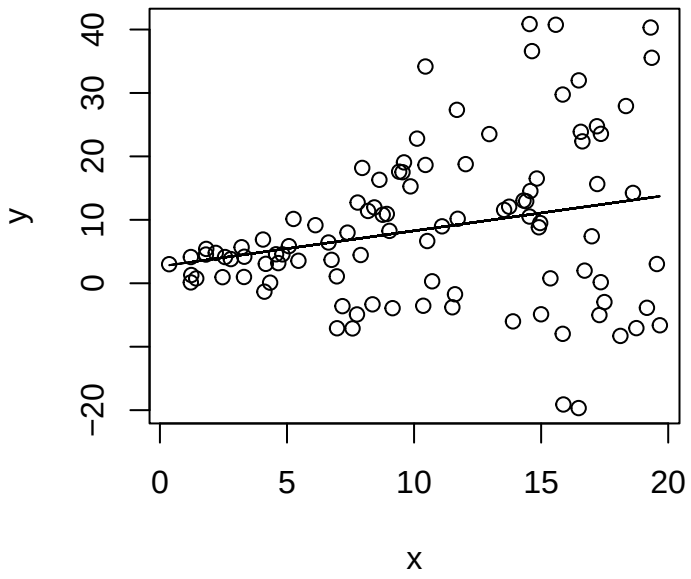
$$\begin{aligned}Y &= r(X) + \varepsilon = \beta_0 + \beta_1 X + \varepsilon \\ \mathbb{E}[\varepsilon \mid X = x] &= 0 \text{ for all } x \\ \text{Var}(\varepsilon \mid X = x) &= x^2 + 1.\end{aligned}$$

In this problem, we'll explore the parametric bootstrap and how it helps us estimate uncertainty in our fitted model $\hat{r}(x)$ when standard assumptions are not met.

(a) First, use R to fit a linear regression model to the data. Plot the residuals and other diagnostics, and indicate which plots would have revealed the assumption violation if you didn't know it was present.

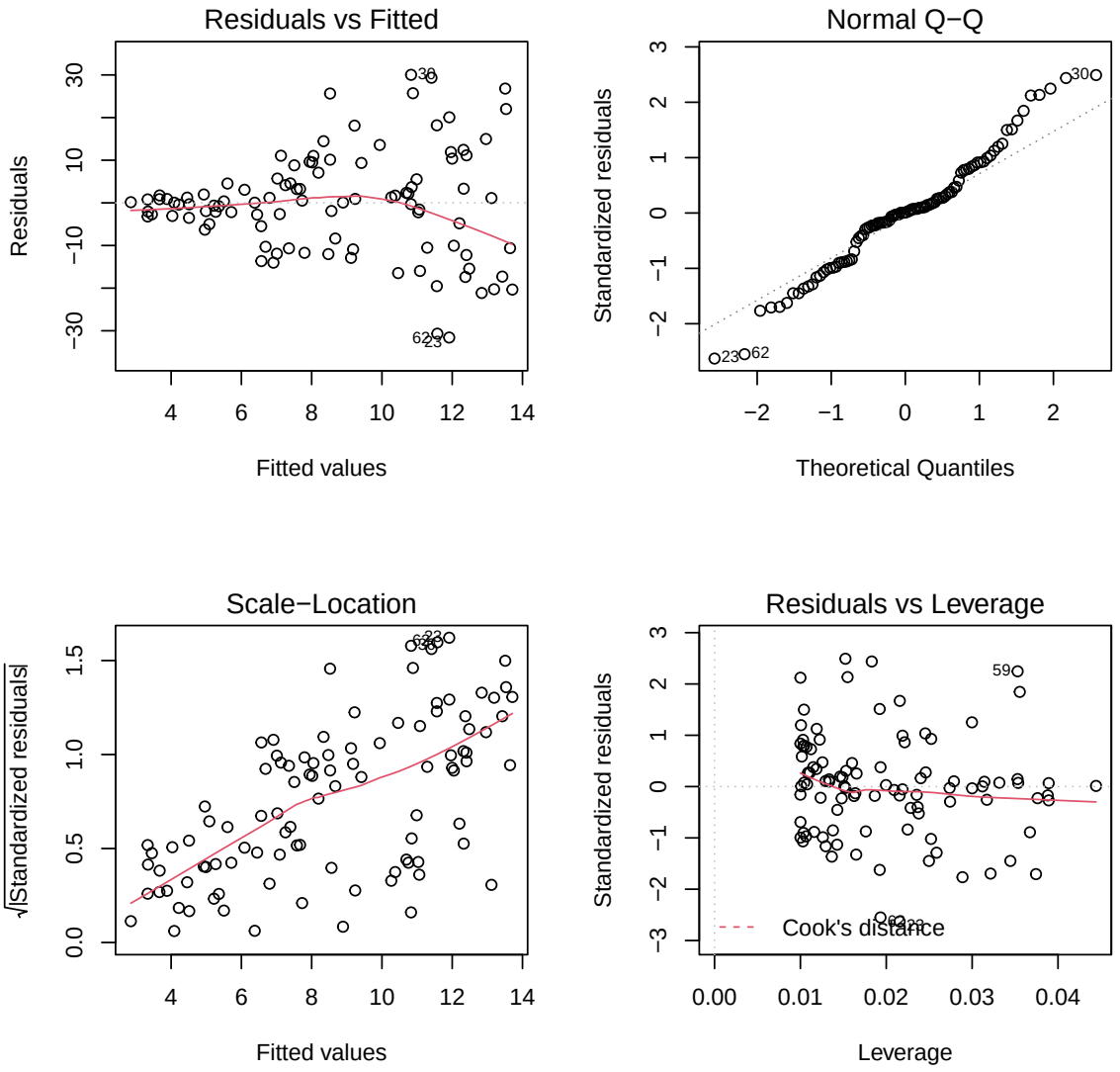
Solution: We start by fitting the model and making a plot of the data and the fitted line to make sure it looks reasonable:

```
orig_data <- read.csv("parametric-bootstrap.csv")

fit <- lm(y ~ x, data = orig_data)

plot(y ~ x, data = orig_data,
     xlab = "x", ylab = "y")
lines(orig_data$x, fitted(fit))
```



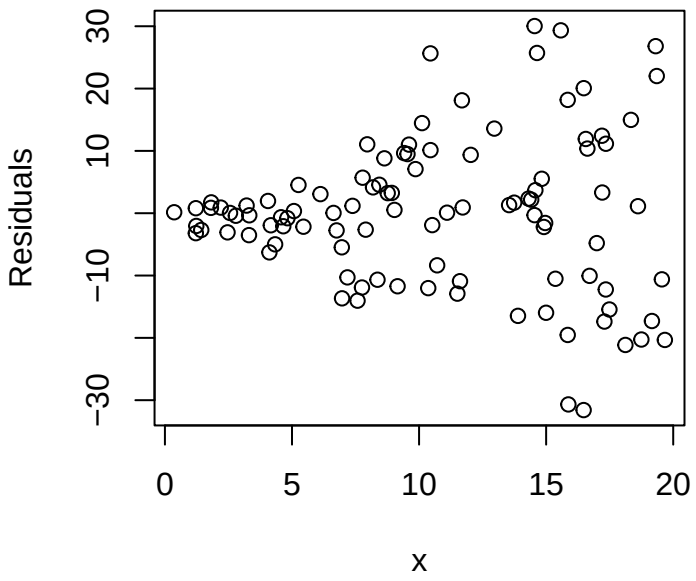
This is good to see, since the line appears reasonable. (It's always good to plot your data just to see that your fit worked correctly and you didn't accidentally do something weird.) Now let's make R's standard diagnostics:

```
par(mfrow = c(2,2))
plot(fit)
```



We can see in the Residuals vs Fitted plot that the spread of the residuals increases from left to right, suggesting the variance is not constant. We a similar trend in the scale-location plot. And if we plot the residuals against X, we see it again:

```
plot(orig_data$x, residuals(fit),
     xlab = "x", ylab = "Residuals")
```



(Of course, the fitted values are just a linear transformation of X, so this plot should look quite similar to the Residuals vs Fitted plot.)

Normality of the residuals looks dubious from the Q-Q plot, but that is because the Q-Q plot is comparing the residuals against a distribution with fixed variance—but our data comes from normal distributions with different variances.

(b) A scientist wants your best estimate of the mean of Y when $X = 15$. That is, you are asked to estimate $r(15) = \mathbb{E}[Y \mid X = 15]$. Using the `predict` function and its `se.fit` argument, make an estimate and give its standard error. What assumptions must be true for that standard error to be accurate, and are they all met?

Solution: We predict:

```
predict(fit, data.frame(x = 15), se.fit = TRUE)
## $fit
##        1
## 11.08485
##
## $se.fit
## [1] 1.560567
##
## $df
## [1] 98
##
## $residual.scale
## [1] 12.14641
```

The assumptions are

1. the relationship is linear
2. the errors are independent of each other, with mean zero and constant variance
3. the errors are independent of X

Or, written mathematically,

1. $Y = \beta_0 + \beta_1 X + \epsilon$
2. $\mathbb{E}[\epsilon] = 0$ and $\text{Var}(\epsilon) = \sigma^2$, i.i.d.
3. $\epsilon \perp X$

(Normality of the residuals is not required; it's used to derive that $\hat{\beta}$ is normally distributed, but we don't need that here.)
The key fact is that the error distribution must be the same regardless of X, but in our case it is not, because its variance increases with X.
See Shalizi section 2.3 for more detail on these linear regression assumptions.

(c) As you may recall from courses like 36-226, the standard error is an estimate of the width of the *sampling distribution* of your estimate. That is, if we were to repeatedly obtain new samples from the population, fit a linear model to them, and use it to estimate $r(15) = \mathbb{E}[Y \mid X = 15]$, then calculate the standard deviation of thousands of those estimates, that is the standard error. Using mathematical assumptions, we can calculate a standard error without needing repeated samples from the population.
But when those mathematical assumptions are not met, the parametric bootstrap can help us estimate the standard error instead. As with all bootstrap methods, it involves *resampling*, i.e. creating new datasets and refitting the model to see how the estimates vary. All bootstrapping methods attempt to approximate what would happen if we could get many independent samples from the population; the parametric bootstrap does so by assuming a specific model for the population distribution.
In our case, we are interested in the uncertainty in our estimate for $r(15)$. To obtain each bootstrap sample, we condition on (fix) X to the values in our original sample, and we draw new Y values from the population distribution—or, rather, what we *estimate* to be the population distribution.
Create a function in R that can simulate new Y values from the estimated population distribution. That is, the function should use the slope and intercept you obtained in part (a) to draw

$$Y = \hat{\beta}_0 + \hat{\beta}_1 X + \varepsilon,$$

where ε is drawn from a normal distribution with the variance we gave at the start of the problem. The output of the function should be a data frame of the same size as the one contained in `parametric-bootstrap.csv`, containing an X column (the original X values) and a Y column (the bootstrapped Y values).

Solution: The function can be quite simple:

```
generate_data <- function(xs) {
  ys <- predict(fit, newdata = data.frame(x = xs)) +
    rnorm(length(xs), sd = sqrt(xs^2 + 1))

  return(data.frame(x = xs, y = ys))
}
```

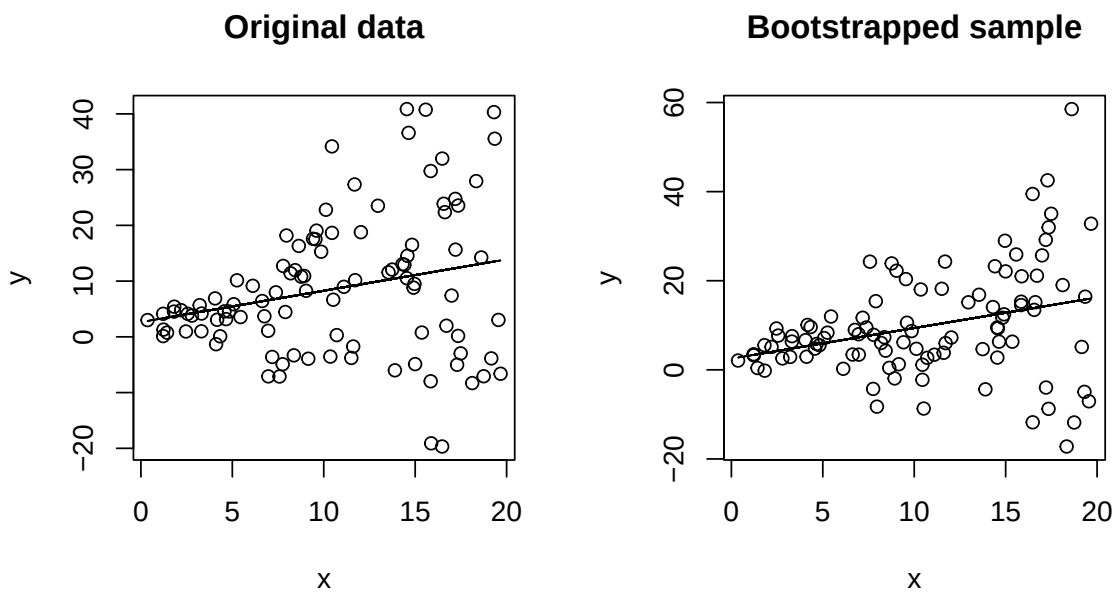
(d) Using the function you wrote in part (c), conduct a parametric bootstrap. That write a loop that runs $B = 1000$, and on each loop iteration:

- Call your function from part (c) to get a new bootstrapped dataset
- Fit a linear model to the bootstrapped data
- Use the model to estimate $r(15)$
- Store the estimate in a vector

At the end of the loop, report the standard deviation of your estimates of $r(15)$ as your new estimated standard error. How does the estimate differ from the one you got from `predict`?

Solution: Before we get into the loop, let's look at the fit to *one* bootstrap sample and see how it looks. This may give you an idea of what the bootstrap is really doing.

```
boot_data <- generate_data(orig_data$x)
boot_fit <- lm(y ~ x, data = boot_data)

par(mfrow = c(1, 2))
plot(y ~ x, data = orig_data,
     xlab = "x", ylab = "y", main = "Original data")
lines(orig_data$x, fitted(fit))

plot(y ~ x, data = boot_data,
     xlab = "x", ylab = "y", main = "Bootstrapped sample")
lines(boot_data$x, fitted(boot_fit))
```



You can see that the bootstrapped sample looks different, and the slope of the fitted line is somewhat different. That's the point of bootstrapping—it shows us what *could* have happened if we had a different sample of data.

Now let's do the bootstrapping for our estimate of $r(15)$.

```
B <- 1000
preds <- numeric(B)

for (ii in 1:B) {
  boot_data <- generate_data(orig_data$x)

  boot_fit <- lm(y ~ x, data = boot_data)

  preds[ii] <- predict(boot_fit, data.frame(x = 15))
}
sd(preds)
## [1] 2.056544
```

This standard error is much larger than the one we got from `predict`, reflecting the increased uncertainty caused by the large variance in Y when X is large.